# Automatic video segmentation employing object/camera modeling techniques 

Citation for published version (APA):

Farin, D. S. (2005). Automatic video segmentation employing object/camera modeling techniques. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Electrical Engineering]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR599186

## DOI:

10.6100/IR599186

## Document status and date:

Published: 01/01/2005

## Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

## Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.
Link to publication


## General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:
www.tue.nl/taverne

## Take down policy

If you believe that this document breaches copyright please contact us at:
openaccess@tue.nl
providing details and we will investigate your claim.

## Automatic Video Segmentation Employing Object/Camera Modeling

Dirk Farin

# Automatic Video Segmentation Employing Object/Camera Modeling 

## PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de Technische Universiteit Eindhoven, op gezag van de Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen op donderdag 15 december 2005 om 16.00 uur
door

Dirk Sven Farin
geboren te Tübingen, Duitsland

Dit proefschrift is goedgekeurd door de promotoren:
prof.dr.ir. P.H.N. de With
en
Prof. Dr.-Ing. W.W.J. Effelsberg

## CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Farin, Dirk S.
Automatic video segmentation employing object/camera modeling /
by Dirk Sven Farin. - Eindhoven : Technische Universiteit Eindhoven, 2005. 2 volumes.
Proefschrift. - ISBN 90-386-2381-X
NUR 959
Trefw.: videotechniek / digitale beeldverwerking / beeldcodering / beeldherkenning.
Subject headings: image segmentation / motion estimation / computer vision / object detection.
Schlagwörter: Videosegmentierung / Bewegungsschätzung / Objekterkennung.
© Copyright 2005 Dirk Farin
All rights are reserved. No part of this book may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission from the copyright owner.

The printing of the color pictures in this thesis was kindly sponsored by the Gauss Foundation.

## Contents

1 Introduction ..... 1
1.1 Motivation ..... 2
1.1.1 Video editing and scene composition ..... 2
1.1.2 Object-oriented video coding ..... 3
1.1.3 Automatic video analysis ..... 5
1.1.4 $3-\mathrm{D}$ analysis and reconstruction ..... 7
1.2 The video-object segmentation problem ..... 8
1.3 Object-oriented video coding in MPEG-4 ..... 11
1.4 Automatic video segmentation system (Thesis Part I) ..... 13
1.4.1 Design goals ..... 13
1.4.2 Segmentation-algorithm overview ..... 14
1.4.3 Framework of the segmentation algorithm ..... 14
1.5 Extensions to the segmentation system ..... 19
1.5.1 Segmentation using object models (Thesis Part II) ..... 20
1.5.2 From camera motion to $3-\mathrm{D}$ models (Thesis Part III) ..... 21
1.6 Contributions of the author ..... 23
I An Automatic Video Segmentation System ..... 31
2 Projective Geometry ..... 33
2.1 Introduction ..... 34
2.2 Projective spaces ..... 35
2.2.1 Homogeneous coordinates ..... 35
2.2.2 Lines in the projective plane ..... 37
2.3 Geometric transformations in 2-D ..... 39
2.3.1 Projective transformation ..... 40
2.3.2 Affine motion ..... 42
2.3.3 Projective motion ..... 44
2.4 Geometric transformations in 3-D ..... 48
2.4.1 Affine motion in 3-D ..... 48
2.4.2 Rotation in 3-D ..... 48
2.4.3 Perspective projection ..... 54
2.5 Image acquisition ..... 58
2.5.1 Intrinsic camera parameters ..... 58
2.5.2 Extrinsic camera parameters ..... 62
2.5.3 Camera motion in a static environment ..... 64
2.5.4 Inter-image transformation ..... 67
2.6 Summary and notational conventions ..... 70
3 Feature-based Motion I: Point-Correspondences ..... 73
3.1 Introduction ..... 74
3.1.1 Basics of feature-based motion estimation ..... 75
3.1.2 From feature-points to motion parameters ..... 77
3.2 Interest-point detectors ..... 78
3.2.1 Moravec interest-point detector ..... 80
3.2.2 Shi-Tomasi detector ..... 81
3.2.3 Harris corner detector ..... 83
3.2.4 SUSAN corner detector ..... 86
3.2.5 Evaluation ..... 88
3.3 Computing feature-correspondences ..... 92
3.3.1 Fast greedy algorithm ..... 93
3.3.2 Evaluation ..... 96
3.4 Summary ..... 103
4 Feature-Based Motion II: Parameter Estimation ..... 105
4.1 Introduction ..... 106
4.2 Computing motion model parameters ..... 106
4.2.1 One-dimensional affine motion ..... 106
4.2.2 Two-dimensional affine motion ..... 108
4.2.3 One-dimensional projective motion ..... 109
4.2.4 Two-dimensional projective motion ..... 110
4.2.5 Non-linear least-squares estimation ..... 113
4.3 Robust estimation algorithms ..... 115
4.3.1 Breakdown of least-squares fit on data with outliers ..... 116
4.3.2 Robust estimation using RANSAC ..... 116
4.3.3 Robustness of the RANSAC algorithm ..... 121
4.4 Summary ..... 126
5 Background Reconstruction ..... 131
5.1 Introduction ..... 132
5.2 Frame alignment ..... 135
5.2.1 Motion models for sprite generation ..... 135
5.2.2 Geometry of background image generation ..... 136
5.2.3 Long-term motion estimation ..... 138
5.3 Background estimation ..... 146
5.3.1 Introduction and previous work ..... 146
5.3.2 The SimMat background-estimation algorithm ..... 149
5.3.3 Results ..... 157
5.4 Summary of the background reconstruction module ..... 160
6 Multi-Sprite Backgrounds ..... 165
6.1 Introduction ..... 166
6.2 Limitations of the single-sprite approach ..... 167
6.3 Detecting degenerated transforms ..... 171
6.4 Examples of single-sprite inefficiencies ..... 173
6.4.1 Example case: camera zoom-out ..... 173
6.4.2 Example case: horizontal camera pan ..... 174
6.4.3 Example case: camera zoom-in ..... 177
6.5 Sprite cost definitions ..... 179
6.5.1 Bitstream length ..... 179
6.5.2 Coded sprite area ..... 179
6.5.3 Sprite buffer size ..... 181
6.5.4 Adding a resolution preservation constraint and lim- iting sprite buffer requirements ..... 182
6.6 Multi-sprite partitioning algorithm ..... 183
6.6.1 Cost matrix calculation and reference frame placement ..... 183
6.6.2 Optimal sequence partitioning ..... 184
6.7 Experiments and results ..... 186
6.8 Integration into the segmentation system ..... 189
6.9 Online calculation of constrained sprites ..... 190
6.10 Coding multi-sprites in MPEG-4 streams ..... 191
6.11 Conclusions ..... 192
7 Background Subtraction ..... 199
7.1 Introduction ..... 200
7.2 Pixel-based classification ..... 201
7.2.1 Distance metrics ..... 201
7.2.2 Influence of the color-space ..... 202
7.2.3 Classes of errors ..... 205
7.2.4 Evaluation method ..... 206
7.2.5 Results ..... 207
7.3 Multi-pixel based significance tests ..... 208
7.3.1 Classification using a $\chi^{2}$ test ..... 209
7.3.2 Extension to color images ..... 211
7.3.3 Fast implementation ..... 212
7.3.4 Evaluation ..... 212
7.4 Classification using Markov random fields ..... 216
7.4.1 MRF model for segmentation masks ..... 216
7.4.2 Obtaining a MAP estimate ..... 217
7.4.3 Extension to color images ..... 220
7.4.4 Optimization algorithm ..... 220
7.4.5 Evaluation ..... 222
7.5 Sources of errors and robustness improvements ..... 223
7.5.1 Map of misregistration risk ..... 225
7.5.2 Map of interpolation errors ..... 227
7.5.3 Integrating risk maps into the segmentation process ..... 230
7.6 Postprocessing the object mask ..... 230
7.6.1 Filling holes in the object ..... 230
7.6.2 Heuristics for removing clutter in the mask ..... 231
7.7 Overview of the segmentation process ..... 232
8 Results and Applications ..... 235
8.1 Algorithm modules ..... 236
8.2 Variants of the segmentation system ..... 237
8.2.1 Surveillance with a static camera ..... 237
8.2.2 Surveillance with a moving camera ..... 238
8.2.3 Offline video analysis ..... 239
8.2.4 Online video segmentation and transmission ..... 240
8.3 Implementation ..... 243
8.4 Segmentation results ..... 244
8.5 Applications of the segmentation system ..... 255
8.5.1 MPEG-4 video coding ..... 255
8.5.2 Video editing ..... 259
8.5.3 Pseudo 3-D video generation ..... 260
8.5.4 Video-object recognition ..... 262
8.6 Extensions ..... 262
8.6.1 MPEG-4 coding with sprite-mode detection ..... 263
8.6.2 Camera auto-calibration ..... 263
8.6.3 Absolute coordinate transfer ..... 264
8.6.4 Object models ..... 264
II Segmentation Using Object Models ..... 265
9 Object Detection based on Graph-Models I: Cartoons ..... 267
9.1 Introduction ..... 268
9.2 Principle of region-based graph matching ..... 269
9.3 Model editor ..... 272
9.4 Automatic color segmentation ..... 275
9.5 Feature extraction and matching criteria ..... 276
9.5.1 Color ..... 276
9.5.2 Size ..... 276
9.5.3 Distance ..... 277
9.5.4 Shape ..... 277
9.5.5 Orientation ..... 278
9.5.6 Node and edge costs ..... 278
9.5.7 Generalization of costs for $1: N$-matching ..... 279
9.6 Matching algorithm ..... 280
9.6.1 Candidate-region selection ..... 281
9.6.2 Matching algorithm ..... 282
9.6.3 1: $N$-matching ..... 285
9.7 Results ..... 288
9.8 Conclusions ..... 289
10 Object Detection based on Graph Models II: Natural ..... 295
10.1 Introduction ..... 296
10.2 Segmentation system architecture ..... 297
10.3 Step 1: motion detection ..... 300
10.4 Step 2: model matching ..... 300
10.4.1 Model editor ..... 301
10.4.2 Model detection ..... 304
10.5 Step 3: spatial segmentation ..... 307
10.5.1 Spatial segmentation algorithm ..... 307
10.5.2 Merging criterion ..... 308
10.6 Experiments and results ..... 308
10.7 Conclusions ..... 310
10.8 Appendix: notes on ellipse processing ..... 316
11 Manual Segmentation and Signature Tracking ..... 319
11.1 Introduction ..... 320
11.2 From Intelligent to Corridor Scissors ..... 321
11.2.1 Intelligent Scissors algorithm ..... 321
11.2.2 Problems of the Intelligent Scissors tool ..... 323
11.2.3 The Corridor Scissors tool ..... 325
11.2.4 Experiments and results with Corridor Scissors ..... 326
11.3 Shortest circular paths ..... 328
11.3.1 Definition of circular paths ..... 328
11.3.2 Computation of shortest circular paths ..... 331
11.3.3 Computational complexity ..... 338
11.4 Signature tracking ..... 342
11.4.1 A first tracking algorithm ..... 342
11.4.2 Signature tracking algorithm ..... 343
11.4.3 Circular-path search with object signatures ..... 344
11.4.4 Tracking results ..... 347
11.5 Summary and conclusions ..... 348
11.6 Discussion on the signature tracking technique ..... 349
11.7 Appendix: step-by-step examples ..... 353
III From Camera Motion to 3-D Models ..... 359
12 Estimation of Physical Camera Parameters ..... 361
12.1 Introduction ..... 362
12.1.1 Geometry of background image generation ..... 363
12.1.2 Global motion estimation ..... 365
12.2 Previous work ..... 367
12.2.1 Estimation of focal length ..... 367
12.3 Linear camera calibration ..... 371
12.3.1 Calibration using the image of the absolute conic ..... 371
12.3.2 Integration of multi-sprite motion estimation ..... 374
12.4 Non-linear camera calibration ..... 375
12.4.1 Parameterization ..... 376
12.4.2 Generalizing to multi-sprites ..... 378
12.4.3 Optimization algorithm ..... 379
12.4.4 Recovering rotation angles ..... 381
12.5 Experimental results ..... 382
12.6 Conclusions ..... 385
13 Camera Calibration for the Analysis of Sport Videos ..... 391
13.1 Introduction and previous work ..... 392
13.2 Calibration-algorithm principle ..... 394
13.3 Overview of the calibration system ..... 395
13.4 Court-line pixel detection ..... 397
13.4.1 Filter 1: luminance threshold ..... 398
13.4.2 Filter 2: non-flat regions ..... 399
13.4.3 Filter 3: linear structure ..... 399
13.5 Line-parameter estimation ..... 401
13.5.1 Line detection with the Hough transform ..... 401
13.5.2 Line detection with RANSAC ..... 402
13.5.3 Line-segment boundary detection ..... 403
13.6 Court-model fitting ..... 405
13.6.1 Fast fitting method ..... 406
13.6.2 Robust fitting method ..... 407
13.6.3 Fast calibration-parameter rejection test ..... 408
13.7 Model tracking ..... 411
13.8 Experiments ..... 413
13.9 Conclusions ..... 414
14 Panoramic Video and Floor Plan Reconstruction ..... 419
14.1 Introduction ..... 420
14.1.1 From background sprites to panoramic images ..... 420
14.1.2 Visualization of panoramic images ..... 421
14.1.3 Floor plan reconstruction ..... 422
14.1.4 Chapter outline ..... 423
14.2 Capturing panoramic images and video ..... 423
14.2.1 Panoramic image generation ..... 424
14.2.2 Cameras for recording panoramic videos ..... 425
14.3 Visualization of panoramic videos ..... 427
14.4 Reconstruction of rectangular rooms ..... 432
14.4.1 The circular arc of possible camera locations ..... 433
14.4.2 Searching for the camera position ..... 435
14.4.3 Creating a virtual room visualization ..... 435
14.5 Reconstruction of floor plans ..... 436
14.5.1 Previous work ..... 436
14.5.2 Reconstruction algorithm concept ..... 437
14.5.3 Modeling the floor plan geometry ..... 438
14.5.4 Estimating the floor plan parameters ..... 440
14.5.5 Improving the convergence behaviour ..... 441
14.5.6 Initialization of the floor plan layout ..... 446
14.5.7 Obtaining wall textures from the panoramic images ..... 447
14.6 Experimental Results ..... 448
14.7 Conclusions ..... 449
15 Conclusions ..... 453
15.1 Discussion on the individual chapters ..... 454
15.1.1 Chapter 3 and 4: camera-motion estimation ..... 454
15.1.2 Chapter 5: background estimation ..... 455
15.1.3 Chapter 6: multi-sprites ..... 455
15.1.4 Chapter 7: background subtraction ..... 457
15.1.5 Chapter 9 and 10: graph-based object models ..... 457
15.1.6 Chapter 11: Corridor Scissors and circular paths ..... 460
15.1.7 Chapter 12: physical camera-parameter extraction ..... 460
15.1.8 Chapter 13: camera calibration for sport videos ..... 461
15.1.9 Chapter 14: floor plans from panoramic images ..... 462
15.2 Explicit vs. implicit models ..... 462
15.3 Future of segmentation ..... 463
IV Appendices ..... 465
A Video-Summarization with Scene Preknowledge ..... 467
A. 1 Introduction ..... 467
A. 2 Summarization algorithm ..... 469
A.2.1 Feature extraction ..... 469
A.2.2 Determining segment boundaries ..... 470
A.2.3 Clustering ..... 472
A.2.4 Integration of domain-knowledge ..... 473
A. 3 Evaluation ..... 474
A. 4 Conclusions ..... 476
B Efficient Computation of Homographies From Four Corre- spondences ..... 479
C Robust Motion Estimation with LTS and LMedS ..... 483
D Additional Test Sequences ..... 487
E Color Segmentation Using Region Merging ..... 489
E. 1 Introduction ..... 489
E.1.1 The region-merging algorithm ..... 489
E. 2 Merging criteria ..... 491
E.2.1 Mean luminance difference ..... 491
E.2.2 Ward's criterion ..... 492
E.2.3 Mean/Ward mixture ..... 492
E.2.4 Linear-luminance model ..... 492
E.2.5 Border criterion ..... 492
E. 3 Criteria properties ..... 493
E.3.1 General behaviour ..... 493
E.3.2 Comparison ..... 494
E. 4 Multi-stage merging ..... 495
E.4.1 Applying a watershed presegmentation ..... 497
E. 5 Results and conclusions ..... 498
F Shape-Based Analysis of Object Behaviour ..... 501
F. 1 Classification of object shapes ..... 502
F. 2 Simple model for object behaviour ..... 503
F. 3 Behaviour analysis ..... 503
References ..... 507
Summary ..... 527
Samenvatting ..... 533
Zusammenfassung ..... 539
Acknowledgments ..... 545
Biography ..... 547

The White Rabbit put on his spectacles.
"Where shall I begin, please your Majesty ?" he asked.
"Begin at the beginning," the King said, very gravely, "and go on till you come to the end: then stop."
(Lewis Carroll)


## Introduction

This chapter presents an introduction to the applications and challenges of video-segmentation and it provides an outline of the thesis structure. The motivation starts with a survey of typical application areas that apply segmentation algorithms. This includes video editing, compression, content analysis, and 3-D reconstruction applications. A particular focus is on the concept of object-oriented video coding in the MPEG-4 standard. Afterwards, the requirements are defined for a segmentation system that is compliant with the MPEG-4 video-coding approach. A proposal for such a segmentation system is made and the main components are briefly introduced. The detailed description of this segmentation system establishes the first half of the thesis (Part I). Finally, various extensions of the segmentation system are proposed which are also discussed further in the second half of the thesis (Part II and III). This introduction concludes with an overview of the individual chapters, indicating the relevant publications and contributions of the author.

Chapter 1. Introduction

### 1.1 Motivation

In 1966, the Artificial Intelligence pioneer Marvin Minsky directed an undergraduate student to solve the problem of computer vision as a summer project. Now, 40 years later, the computer-vision problem is still unsolved, despite the huge amount of joint efforts that have been undertaken in the research community. While the general video-understanding problem is widely regarded as too ambitious to be solved in the near future, various successful spin-offs for practical problems in controlled environments have been developed. Although video understanding is a very difficult area for automatization, it is worthwhile to adopt it in many applications that are currently operating mainly at the signal level without content-adaptive processing. The better we understand not only the video-signal statistics, but the semantic meaning of it, the better we can adapt the associated video processing and the more possibilities for interaction with the video content are made available.

The video understanding problem can be specialized into a large variety of applications. In the following sections, four application areas for video segmentation are outlined. This overview also provides references to the relevant chapters in this thesis.

### 1.1.1 Video editing and scene composition

Until recently, video editing was equivalent to the temporal cutting of video to create movies or documentaries. However, the production of movies is currently making the step towards an object-oriented scene composition. An increasing number of scenes are not recorded directly as a whole, but recorded object-by-object and composed later.

For the composition of video objects, not only the raw texture data is required but also the object shape in form of a mask to seamlessly insert the object into a background image. In the case of computer-generated objects, this mask is easy to obtain, but for captured real-world objects, the mask must be deduced from the image itself. For difficult cases or when no compromise on quality is allowed, this segmentation task is still done manually. Automatic segmentation is usually carried out with the chroma-key technique. Here, by providing a scene background in the designated background color, the object can be distinguished easily from the background. The disadvantage of the choma-key technique is clearly that it is only applicable in a strictly controlled environment. For the extraction of objects from general video-sequences, other segmentation techniques are required. (High-quality segmentation for editing application will be addressed in Chapter 11).

### 1.1.2 Object-oriented video coding

Most current techniques for video coding like MPEG-2 are non-adaptive to the content. Choosing between different coding modes for individual macroblocks allows for some adaptation to the video content, but the decision about coding modes is usually based on minimizing the resulting bit-rate. However, people started to intentionally misuse the possibilities of the video-coding tools to achieve a higher subjective image quality. These techniques exploit properties of the human visual system at the semantic level of attention to detect those areas in the image that are most important to the viewer. These areas are then coded with a better quality at the expense of a lower quality in other regions.

Such a system has been proposed by the author in earlier work to achieve a better visual quality for an MPEG-2 video coding system [67, 66]. In this system, the image was classified into regions of texture, text/graphics, and smooth areas. Since it is important to keep text at a high quality, its quality was increased by using more bits for coding text regions instead of spending the bits on the coding of texture regions. Additionally, the encoder used a scene-change detection algorithm to decrease the coding quality in a short time interval around the scene-change, based on the observation that the human perception is not sensitive to a low image quality close to global changes in the image. Compared with a system that optimizes on PSNR [64], the content-adaptive coding can achieve a better visual quality for the same bit-rate.

With the advent of the MPEG-4 video coding standard, object-oriented video coding was for the first time integrated as a substantial part of a video coding standard. The object-oriented approach offers several advantages and new possibilities. Let us present two of them by using the example case of news programs.

- Composing the visual elements of the news program into a single picture and transmitting this picture as a whole leads to low compression efficiency, because the image is composed of objects of different nature. While the anchorman is a natural video object, there is also superimposed text, graphical images like maps, and computergenerated videos like the weather chart. A better coding efficiency could be obtained by using specialized codecs for the different types of content and composing the final scene at the decoder.
- An object-oriented video representation also offers more possibilities to interact with the content. For example, the viewer could choose his favourite design of the news studio, or he may increase the text fontsize if he is visually impaired. Another possibility of interaction is to
provide object-specific annotations or to make the objects clickable. This would make it possible to create videos with hyperlinks that can be activated by clicking on objects.

A prerequisite for the above application features is the availability of efficient object-segmentation algorithms. However, the accuracy required for the segmentation results depends on how the segmentation masks are employed in the coding process. We can identify three coding approaches that we present in the following in the order of increasing requirements on the segmentation accuracy.

- Region-of-Interest (ROI) coding. The image is coarsely separated into background and foreground. The background includes the content that is unimportant for the viewer whereas the foreground comprising the more important objects. The video coder can then be controlled to code the foreground with a higher quality at the expense of lower quality in the background. The ROI-coding approach is interesting, for example, for surveillance-video recording systems, since the amount of recorded video can be increased, while still keeping the high quality of the objects that matter in the analysis. Moreover, because of the static background in surveillance videos, it is also easy to define the important foreground objects.
- Coding improvement by object-border detection. Traditional video coding approaches mostly employ block-based transforms that do not adapt to the boundaries of objects. However, the texture usually changes suddenly at the object border. Filtering across this border consequently leads to a low decorrelation of the pixel values.
Both problems can be eliminated by coding the interior and exterior regions independently. The MPEG-4 video-coding standard applies this approach by employing a shape-adaptive DCT to include only object pixels in the transformation, and by restricting the effect of the motion vectors to the current object area.
For this coding approach, it is not required that the segmentation masks are semantically meaningful, as long as they serve to improve the coding efficiency.
- Composition of video objects. Semantically correct segmentation masks are certainly required if the purpose of the object-oriented video coding is not only to provide better compression efficiency, but also to enable the composition of new scenes from independently captured video objects. It should be noted that this area of applications
not only covers TV broadcasts, which is traditionally a more passive medium for the viewer, but it also covers especially more interactive internet applications. Numerous possible applications exist and include web-based games, product presentations (e.g., show a specific piece of furniture in various environments), virtual realities, or interactive design applications.


### 1.1.3 Automatic video analysis

Video-object segmentation is also an indispensable technique for an indepth analysis of video content. Before the objects in a scene can be identified and their behaviour is analysed, video object have to be detected and separated in the image. The segmentation accuracy that is required depends on the subsequent analysis steps. In case that the object behaviour is derived from the motion trajectory of the object, an accurate segmentation mask is not required. Contrariwise, if the object shape is used to derive its pose and ultimately its behaviour, the accuracy of the segmentation mask is of significant importance.

Automatic video analysis is relevant for a broad range of applications. In the following, we provide only a few examples:

- Surveillance. One application area that is quickly growing is the automatic analysis of surveillance videos. Currently, surveillance systems are still non-intelligent video recording systems, often comprising a larger number of cameras. The analysis is primarily still performed by humans watching the videos either in real-time or from the recording. Automatic surveillance systems can help in this situation by either doing the analysis completely automatic, or by providing a pre-alarm indicating situations that require a closer look by a human observer.

Surveillance is also attractive for automatic analysis from a technical point of view because the input video is relatively easy to analyze. Often, the cameras are statically mounted, such that the environment is well-defined. Moreover, the objects that should be observed can usually be well defined. (Related to Appendix F.)

- Sports. Another application that is, from an algorithmic point of view, similar to the surveillance application is the automatic analysis of sport events like tennis or soccer games. Well-known examples are offside analysis in soccer games or court-line checks in tennis. Automatic sports analysis can extract statistical information about the game or for individual players. This information can subsequently
be used to enrich the sports transmission with additional information about the player performance, which is for the entertainment of the viewers, but which can also provide valuable information for the coaches to analyze the strengths and weaknesses of their own athlets or the competitors. From the technical point of view, sports analysis is also interesting because the variety in the scenes is rather limited, thereby enabling a more detailed analysis. The playfield is usually well defined by the markers that are drawn on the playfield. Moreover, the behaviour of the players is well defined and can be described by the rules of the game. (Related to Chapter 13.)
- Video databases. Storage costs for video data become increasingly lower and large amounts of video data are already collected in professional archives and at home. The search and retrieval in these media databases poses the new problem of efficiently searching in video data. Manual annotation of the videos with meta-data is often not feasible, so that the search must be carried out either on the raw video data or on automatically generated meta-data. This again requires detailed video analysis, since the queries have typically a high semantic level. An optimal query system should be able to transfer a linguistic description of the scene to a suitable query into the video data. For specific applications like surveillance, this is easier to accomplish, since the nomenclature is well defined. (Related to Chapters 9 and 10.)
Another problem specific to video databases is the quick browsing in the archive. Since video is a medium that takes place also in time, it cannot be understood quickly from a static snapshot. However, the computer can help to reduce the amount of video that should be viewed by preselecting the most important scenes, or the scenes which are most characteristic to deduce an impression of the full video. Current algorithms in this area usually only consider the global appearance of the image, but the systems can be extended to more in-depth analysis by detecting specific objects of interest [104] and extracting preferably those scenes where these objects occur. (Related to Appendix A.)

Further applications of video analysis which we do not consider further are medical applications, industrial image processing for, e.g., quality control, robotics, or remote-sensing. Even though such applications may be very different, the basic video-analysis techniques are comparable and their principles can be reused. For example, semi-automatic segmentation algorithms as described in Chapter 11 for natural images are also
very popular in medical applications, e.g., for defining tumor areas on CT scans. Industrial image processing often employs of simple color segmentation (Appendix E), because the environment can be controlled easily. Mobile robots require video-object segmentation for collision prevention or for the interaction with the objects. Finally, remote-sensing applications apply the same change-detection algorithms as the ones in the foreground extraction presented in this thesis, but for remote-sensing, their usage is to identify changes in vegetation.

### 1.1.4 3-D analysis and reconstruction

Video-object segmentation as discussed up to now was related to the extraction of a two-dimensional mask of the object in the input image. However, the input image itself is only a projection of the 3-D world onto a flat image. More information about the scene can be obtained if the analysis system is successful in recovering the 3-D geometry and motion of objects. The 3-D reconstruction approaches can be coarsely classified in techniques generating volumetric models and techniques reconstructing surface models. A volumetric model is obtained with reconstruction-from-projections approaches as they are known, e.g., from computer tomography. These volumetric models are beyond the scope of this thesis. In a surface model, the objects are represented using only their textured surface. Apart from the object geometry, 3-D reconstruction also includes estimating the 3-D motion of objects as well as the motion of the camera.

Even though general 3-D reconstruction is out of the scope of this thesis, there is a gradual transition between video analysis and 3-D reconstruction. For example, to derive an appropriate model for camera motion, we have to consider the 3-D motion of the camera. It turns out that the depth of a scene is insignificant as long as the camera motion is restricted to rotational motion around a fixed optical center. This type of camera motion plays a central role in the thesis and is therefore examined in more detail. Especially, it is discussed how the physical camera-movements can be recovered from the observed camera motion (Chapter 12). This information establishes a link between the 2-D video image and the 3-D real-world geometry. Knowing this relation, techniques to augment the input video with computer-generated objects are made possible. Thereby, the virtual camera for the generation of the computer images is controled by the parameters extracted from the input video. This has the effect that the virtual camera follows the motion of the real camera, which enables a seamless integration of the virtual objects into the original scene.

The physical camera-parameters are also helpful in the video-content analysis, since the type of camera motion often provides information about
the intention of the editor. For instance, a camera zoom onto a face indicates that this person plays a major role in the scene.

Another case considered in the thesis is the calibration of the camera for sports sequences (Chapter 13). In this application, the calibration establishes the link between the 2-D image coordinate system and a real-world coordinate system. The transformation to absolute coordinates is required for in-depth analysis of the content, since in sports sequences, the position of the players on the playfield is important, but not their position in the image.

Although the thesis does not cover general 3-D reconstruction, the segmentation algorithm employs synthesized background images as a representation of the scene. These background images can also be considered as 360 -degree panoramic images, which, when unwrapped, are also rectangular flat images, but covering a full 360 -degree panoramic view. In this context, the question arises how these panoramic images can be best visualized to the user. The proposed solution from Chapter 14 is a simplified semi-automatic 3-D reconstruction which recovers the global room geometry to give coarse orientation hints to the viewer.

### 1.2 The video-object segmentation problem

The task of video-object segmentation is to identify and separate the important objects in a video scene from the scene background. Clearly, to approach this problem, it is necessary to define what is exactly meant with important objects and how the correct object masks should look like. However, in practice, it turns out that even an unambiguous definition of video objects is a fundamental problem. In the following, the involved definition problems are addressed and grouped into physical problems, being a consequence of the image formation, and semantic problems. The physical problems are as follows.

- Shadows. Objects cast a shadow onto the ground or background. Since this shadow moves with the object, it can be considered part of the object. However, this might be undesired in video analysis applications. For example, if a subsequent object recognition is based on shape information, this can lead to an erroneous object classification. For this reason, special algorithms for shadow identification and removal (Fig. 1.1(a)) have been proposed [72, 162]. In video editing applications, it may be desired instead to reproduce a similar shadow on a new background [27, 20].
- Reflections. The problem of handling reflections is actually similar to object shadows. However, reflections are more difficult, because the appearance of the reflected image depends on the physical properties of the reflecting surface (Fig. 1.1(b)) and because the reflection is not necessarily attached to the object.
- Occlusions. The object shape can also change because of occlusions. It depends on the application whether the masks of occluded objects should be extended to their original shape.
- Translucent objects. Objects can appear partially translucent since they are made of translucent materials, or because thin structures like hair or cloth appear translucent. Moreover, pixels along object boundaries are always a mixture of foreground color and background color. To model the translucency, the segmentation algorithm has to compute an alpha-channel mask which identifies the translucency factor for each pixel instead of only computing a binary object mask. Accurate alpha-channel information cannot be obtained from a single image, but algorithms using heuristic approaches have been proposed [26, 161].

Apart from the physical problems, there are semantic definition problems, like the following.

- Objects of interest (foreground objects). The first and obvious question of video segmentation is what parts of an image constitue the foreground objects. This issue is already surprisingly difficult, since the intuitive human understanding of foreground objects is strongly depending on the scene context. Mostly, human intuition expects that this should be the main acting objects. For example, in a sports broadcast, the players are usually considered foreground and the audience is considered background, even if the audience is moving (see Fig. 1.1(c) for an example). This distinction is on a very high semantic level, since it assumes knowledge about the meaning of the scene. Note that the object definition can also vary with the application. A surveillance system in a sports stadium will be interested in other objects than a system for automatic analysis of the sports game.
- Small background movements. When taking a more detailed view on the last point, it can be observed that the distinction between foreground and background is in fact gradual. The question is to what extent a background should change such that it is considered part of the foreground. For example, trees may occur in the background with

(a) Segmentation without and with shadow removal.

(c) Background objects: not only the players are detected, but also the referee on the right side.

(b) Semantically difficult segmentation with reflections and transparencies.

(d) Occlusion: the car passes behind a pole, which separates the car into two parts.

Figure 1.1: Various segmentation problems, such as shadows, reflections, and occlusion.
leaves moving slightly in the wind, or there may be a clock on a wall at the back of the room.

- Object-status change. Objects can also change their classification over time. For example, most people would consider a car that drives along a street as an important object. But how to define the object status when the car stops and parks at the side of the street? Alternatively, the opposite case may occur that a car that was parked for a long time suddenly drives away. Note that it is practically impossible to separate all objects, including the static ones, into independent
objects, since this would imply that all future actions would have to be predicted.
- Multi-body objects. Objects may be separated into several nonconnected regions in the image. One reason for this can be that an occluding object cuts the foreground object into pieces. For example, see Fig. 1.1(d). Another complex example are objects that are really composed of several parts but still belonging together like flocking birds.
- Hierarchical objects. Additional to multi-body objects, there can also exist a hierarchical relationship between objects. One example is a car object that contains a driver object.

When considering all of these problems simultaneously, it can only be concluded that a general-purpose segmentation of video objects is virtually impossible, since the definition of the expected output from the algorithm depends largely on the scene context and the application that we have in mind. However, despite all the mentioned problems, it is still possible to design algorithms that cover a multitude of specific applications and that work well in many practical cases.

### 1.3 Object-oriented video coding in MPEG-4

The MPEG-4 video-coding standard is the first and to date only videocoding algorithm that supports the coding of arbitrarily-shaped video objects. In the terms of MPEG-4, a video is composed of several independent Video Objects (VOs) that can be placed in front of a background image (sprite). This background sprite image can be larger than the display size, such that only part of the sprite is shown at a time. This concept enables an efficient transmission of video sequences with camera motion, since the current background view can be obtained from the sprite buffer. Additionally, only the foreground objects have to be transmitted to the decoder. After decompression, the decoder superimposes the foreground objects onto the background image.

This coding architecture has two pronounced advantages. First, it provides for a potentially higher compression ratio, since the background is only transmitted once and the foreground objects are considerably smaller than the complete picture. Moreover, the video quality can be regulated on a per-object basis, assigning a larger bit-rate for the important objects and a smaller bit-rate to the background $[136,188]$. Second, the separation of objects allows for new possibilities to interact with the content. Video objects can be extracted from one sequence and placed into a different scene.


Figure 1.2: Overview of an object-oriented MPEG-4 video coding system.

Note that it is also possible to remove objects from the scene. This does not result in an undefined hole in the image because the background can be obtained from the sprite image.

An overview of a typical object-oriented MPEG-4 video coding system is depicted in Figure 1.2. At the encoder side, the input video is analyzed and split into independent video objects (VOs) in the segmentation module. Instead of the indicated automatic segmentation module, the video objects can also be generated by other techniques, like synthetic content, or they may be recorded separately and segmented using a blue-screen technique. The output of the segmentation module is formed by several video objects that comprise the texture data and also the shape of the independent objects. Moreover, the segmentation module also generates a sprite image of the scene background and suitable camera-motion parameters to reconstruct the original camera motion.

Video objects and sprite data are encoded independently into separate elementary bitstreams by an MPEG-4 compliant encoder and multiplexed into a single bitstream. At the decoder side, the received bitstream is split
into the elementary bitstreams and passed to the texture and VO-shape decoders. Note that the sprite image is only transmitted at the beginning of the scene and then stored in a sprite image buffer. The scene background image is then reconstructed by displaying a geometrically transformed part of the sprite-buffer content (using a projective transformation). Finally, the video objects are superimposed onto the scene background in the scene compositor using the VO shape masks.

### 1.4 Automatic video segmentation system (Thesis Part I)

Part I of the thesis (Chapters 2-8) describes a generic, automatic segmentation system. The motivation for this segmentation system is to implement a video-object segmentation module that can be integrated into an MPEG4 encoding framework, as depicted in Figure 1.2. The requirement that the segmentation system should be compliant to MPEG-4 imposes restrictions onto the algorithm design. These design constraints are described in the subsequent section. Afterwards, an overview of the segmentation system is presented, briefly summarizing the processing carried out in each algorithm step.

### 1.4.1 Design goals

A principal design criterion for our segmentation system is to build a system that is compatible with the object-oriented video-coding tools as defined in the MPEG-4 video-compression standard [93]. It was discussed previously that it is not feasible to target an automatic segmentation system operating unambiguously for every possible input video. Consequently, we restrict ourselves to a limited, well-defined case which nevertheless enables a broad range of applications. More specifically, the proposed system is based on the following two fundamental assumptions.

- Static background. We assume that the scene background is static. Camera motion is allowed (see next point), but every object that changes its appearance relative to a static background is considered foreground. ${ }^{1}$
- Rotational camera motion. We assume that the recording camera is a pan/tilt/zoom camera. This means that the camera is allowed

[^0]to rotate around its optical center in any direction, and it may also change the focal length (zoom). However, translatorial camera motion is not allowed because in such a case, the parallax effect would make it impossible to synthesize a static background image.

These assumptions are sensible for many practical applications. For example, most surveillance cameras are pan/tilt/zoom cameras observing objects in a static environment. Moreover, in many typical television scenes like interviews or sport transmissions, several cameras are used at a fixed position and the operators switch only between the views. The restriction to rotational camera motion is also required in order to be compatible with the MPEG-4 video-compression standard, which only supports this type of camera motion for the background sprites.

### 1.4.2 Segmentation-algorithm overview

The segmentation system developed in this thesis is based on the backgroundsubtraction approach. In this technique, the segmentation algorithm compares the input images to a background image. The background image is a synthesized view of the scene background in which all foreground objects have been removed. Regions in the input image that differ from the background image are marked as foreground objects (Fig. 1.3).

To compute the background image, the camera motion is estimated and by compensating this motion, the input images are composed into a background image. If the camera is panning across the scene, this results in panoramic background images which are larger than the input image. Finally, the background image is reconstructed from the input sequence by integrating all frames of the sequence such that non-static foreground objects are removed from the image.

The synthesized background images together with the extracted foreground objects match the input that is required for the MPEG-4 objectoriented video coding tools. Hence, the output of our segmentation algorithm can be directly used as input for an object-oriented MPEG-4 video coder.

### 1.4.3 Framework of the segmentation algorithm

This section briefly describes the processing steps of the developed segmentation system (see Figure 1.4) and introduces the core algorithms that are used.

(d) Sprite image.

Figure 1.3: Principle of the segmentation algorithm. All input images (a) are combined into a static background sprite (d). The current camera view (marked quadrilateral) is extracted and dewarped (b). By comparing the input (a) and the background (b), the foreground object (c) is obtained.

## Camera-motion estimation

The largest part of the system is the camera-motion estimation. The difficulty of the camera-motion estimation is that it must be robust to foreground object motion, and that it also has to give very accurate motion parameters such that all input frames can be combined seamlessly into a background image. To achieve this, we apply a combination of a featurebased motion estimator and a direct estimation algorithm.

Chapter 3 starts the presentation of the feature-based motion estimator with the detection of feature-points that can be retrieved reliably in a subsequent frame. In the next step, correspondences between matching points are established. Each feature-correspondence can be viewed as the
motion of this point between the two images. Chapter 3 describes several feature-point detectors (SUSAN, Harris, Shi-Tomasi, and Moravec), which are evaluated for their accuracy and robustness, for a wide variety of video sequences. Moreover, the chapter describes an algorithm to compute feature-correspondences across pairs of images. The algorithm features prediction of the matching feature-points and a fast neighborhood-search to achieve a low computation time.

Chapter 4 discusses the estimation of the global-motion parameters from the previously computed feature-correspondences. First, the problem is considered on sequences with pure camera motion without foreground objects. Subsequently, the estimation problem is considered for the general case that the observed motion is a mixture of camera motion and object motion. Robust estimation algorithms are introduced to estimate the parameters of the dominant motion. Chapter 4 concentrates particularly on the RANSAC algorithm, while other estimators are considered in Appendix C. Finally, the chapter presents why the robust estimator breaks down earlier in practice than predicted from theory. Based on the discussion of this phenomenon, the RANSAC algorithm is modified to increase its robustness to about the theoretically predicted performance.

The first part of Chapter 5 presents the direct motion-estimation algorithm to refine the motion parameters obtained from the feature-based motion estimator. The obtained motion parameters are a good estimate for the inter-frame motion, but they are not accurate enough to build a global background image. The direct motion estimator is a gradient-based globalmotion estimator, which computes long-term motion parameters between each input frame and the background sprite. Compared to the short-term motion parameters obtained from the feature-based motion estimator, the long-term parameters have a higher accuracy, since there is no accumulation of errors as would happen with a concatenation of inter-frame motion parameters. However, the gradient-based estimator should be initialized with the result of the feature-based motion estimator, because the area of convergence is smaller and it thus requires a good initialization.

## Background reconstruction

The second part of Chapter 5 discusses the synthetization of static back-ground-sprite images with all foreground objects removed. Knowing the camera-motion parameters, a background image can be synthesized by stitching the input images to a common background image using the obtained parameters. However, the essential problem in this step is to remove the foreground objects from the background image. The chapter presents a new algorithm based on the observation that each region in the image
can be classified into one of three states: static background, moving foreground, non-moving foreground. The difficult case in this classification is the distinction between static background and temporally non-moving foreground. This classification problem is solved by building clusters of regions with stable content in temporal direction, and by considering that the times in which foreground appears in neighboring regions are similar. Finally, the proposed algorithm is compared to other algorithms, especially the median algorithm, which is the best previously known algorithm.

Our approach of generating background images uses the same motion models as those that have been defined in the MPEG-4 and MPEG-7 standards to describe camera motion. However, it is shown in Chapter 6 that this approach does not work for all kinds of camera motion. This problem has not been considered in previous work, even though it leads to major problems when camera rotations of large angles are present. For example, the MPEG-4 sprite-coding approach becomes inefficient for camera rotation angles larger than approximately 25 degrees. Moreover, ad-hoc implementations for sprite generation usually do not consider that camera zooming changes the image resolution and, if the higher resolution is not reflected in the sprite, the reconstructed view from the sprite misses small details. Chapter 6 first shows theoretically that all of these problems can be solved by computing a set of independent sprites (a multi-sprite) instead of trying to compute a single sprite representation. Afterwards, a novel multi-sprite partitioning algorithm is presented, which partitions the video sequence into a number of segments, for which independent sprites are synthesized. The partitioning is computed in an optimal way, such that the total area of the resulting sprites is minimized. Furthermore, the algorithm can incorporate constraints, such as a limited sprite-buffer size at the decoder, or the restriction that the image resolution in the sprite should never be lower than the input image resolution. The described multi-sprite approach is compatible to the MPEG-4 standard, and yet provides several advantages: any arbitrary rotational camera motion can be processed, the coding-cost for transmitting the sprite images is lower, and the quality of the decoded sprite images is better. In Figure 1.4, the multi-sprite algorithm is depicted as an extension to the baseline segmentation algorithm.

## Foreground object segmentation

Chapter 7 describes the actual foreground-object segmentation, which is based on a background-subtraction technique. Input images are compared with the corresponding camera-motion compensated view from the synthesized background sprite and areas that deviate are marked as foreground. At first, a classification of independent pixels is considered, where


Figure 1.4: Overview of the segmentation system.
the influence of the color-space and the difference measure is evaluated. Subsequently, multi-pixel based tests and Markov random fields are used to derive the foreground mask with improved accuracy. Moreover, the concept of risk maps is introduced to account for the problem that the background image may not be perfectly aligned to the input images, e.g., because of inaccuracies in the motion-estimation. These risk maps significantly reduce the errors caused by misregistration and by blurring of the background image that occurs in the image warping. Finally, post-processing filters are described that remove clutter regions from the segmentation mask.


Figure 1.5: Extensions to the segmentation system.

## System architecture

Chapter 8 discusses that a complete segmentation system comprises all or a selection of the processing steps outlined above. Depending on the application and type of sequences to be processed, the system can be implemented in different variations. For example, the algorithm can be simplified for a surveillance application with static cameras, or the application may require an online real-time segmentation. Furthermore, this chapter provides results of the segmentation algorithm on a wide variety of input sequences and typical effects and problems of the segmentation algorithm are discussed. Finally, example applications for the described segmentation algorithm are presented. This includes MPEG-4 video coding, for which the gain in the compression ratio is discussed, object-based video editing, pseudo 3-D video generation, or object recognition.

### 1.5 Extensions to the segmentation system

The segmentation system outlined in the previous section can be regarded as the core framework, which can be adapted to many specific applications.

Some of these possibilities are the subject of Part II and Part III of the thesis. These two parts relate to the two research directions object models and camera models that are considered as particularly interesting.

### 1.5.1 Segmentation using object models (Thesis Part II)

A first possibility is the integration of model knowledge about the objects to be segmented into the segmentation algorithm. The segmentation algorithm outlined so far has no explicit knowledge about the objects to be extracted. However, if object models should be added to the segmentation system, the central question is how the object description should be defined. It is important to balance between an object definition that is accurate enough to uniquely identify the object, and a definition that allows for enough freedom to recognize the object in different views.

## Graph-based object models

In Chapter 9, a graph-based object model is presented. In this model, the main regions of the object and the region features are summarized in the graph nodes, and the spatial relations between these regions are expressed with the graph edges. The approach is first described in Chapter 9 for the special case of cartoon sequences, since for this type of sequence, the object regions can be obtained easily with color segmentation. The graph-based object-detection system consists of two parts. First, the user defines the object model based on an example image of the object. At the detection stage, the algorithm applies an automatic color segmentation onto the input image to obtain a similar, but much larger graph of the input image. Using an efficient sub-graph matching algorithm, the object is identified in the input image.

Chapter 10 extends this concept to the detection of objects in natural video sequences. A similar graph-model is used with the only difference that the region shapes, which cannot be extracted easily using color segmentation, are approximated using ellipses. The concept of the object-detection algorithm is similar to the algorithm for the cartoon sequences, except that for each model region, a set of candidate regions is first extracted from the input image. This set comprises several possible placements of ellipses in the input image to cover areas that have similar color and size as indicated in the object model. Additionally, the algorithm can integrate changedetection masks providing a coarse hint about the location of the objects. The final object segmentation is carried out with a color segmentation algorithm that is modified such that the object regions are restricted to the areas covered by the detected object position.

The graph-based object models can be integrated into the core segmentation system as depicted in Figure 1.5. In this configuration, the segmentation result of the core segmentation system is applied only as a coarse indication of object location. Note that this indication can be incomplete if only a part of the object is detected in the segmentation system (e.g., see Fig. 8.14). The subsequent model detection step uses this first indication of the object location and the object model that has been created manually with the object-model editor to determine the image area that comprises the object. Accurate pixel-level object boundaries are computed in the final spatial segmentation step.

## Object signatures for tracking

Chapter 11 presents a different approach to describe specific objects. The segmentation problem is approached from a different perspective. For applications requiring highly accurate segmentation masks, the quality provided by an automatic segmentation may not be sufficient. For these cases, it should considered to use a semi-automatic segmentation algorithm where the user controls the segmentation, but the computer relieves him from working at the pixel level. To this end, an advanced algorithm based on the concept of the Intelligent Scissors algorithm [130] is presented. This is an edge-based segmentation algorithm, in which the user traces along the object edge to define the boundary. This approach is generalized to search the object contour in a user-drawn corridor that he draws along the object boundary. The exact object boundary is obtained using a newly developed shortest circular-path search algorithm.

Apart from the semi-automatic segmentation algorithm, the chapter deals with the novel concept of object signatures. The object signature is defined as the image texture along the object boundary. Once this object signature is known, for example from a segmentation of the first frame of a sequence, its information can be integrated into the segmentation of the successive frames. This enables an automatic tracking of the object through the sequence without manual intervention. The tracking step can also be added to the core segmentation system in order to carry out the computationally expensive segmentation only for one frame and then switch to the more efficient object tracking.

### 1.5.2 From camera motion to 3-D models (Thesis Part III)

During the work on camera-motion estimation, it was observed that there is a close connection between the camera-induced motion in the image and the scene geometry. This relation between the 2-D image and the 3-D
world becomes especially important when the segmentation result is used to analyze the video content. Whenever object motion is used for the analysis, the object motion in the image has to be translated to motion in the real-world, because the motion in the image is composed of the object motion as well as the camera motion. Different techniques are explored to derive information about the physical 3-D world for the special case of rotational camera motion.

The core segmentation system employs camera-motion estimation to compensate for any camera motion. When we derived the camera-motion model in Chapter 2, we started with a 3-D model of the environment and the image-formation process of a camera. Based on this physically motivated model, the projective motion model was derived, which was subsequently used in the camera-motion estimation. However, although the motion model was derived from a physical description including rotation angles and the focal length of the camera, it is not easily possible to recover these parameters from the parameters of the estimated projective motion model.

Chapter 12 adresses the inverse problem of factorizing the motion parameters into physically meaningful parameters using camera autocalibration techniques. Our approach uses first a linear estimation approach based on the concept of the image of the absolute conic. To refine the motion parameters, the accuracy is further increased with a non-linear optimization similar to bundle-adjustment techniques. The speciality of the new algorithm is that it can integrate camera motion that spans several sprites according to the earlier introduced multi-sprite technique. Consequently, the algorithm can be applied to arbitrary unrestricted rotational camera motion.

For the analysis of video sequences, it is often required to know and follow the position of the objects. Clearly, the object position in terms of image coordinates provides little information as long as the viewing direction of the camera is not known. In some application, like sport videos, the camera view can be determined from markings on the ground of the playing field. Chapter 13 provides a new algorithm to deduce the transformation between the image coordinates and the real-worlds coordinates, based on the lines defining the playfield. The theory behind this approach is closely related to the estimation of camera motion, since also the mapping between two image planes (the image and the flat real-world ground plane) is estimated. However, the difference is that in this case, a mapping onto absolute coordinates is obtained.

The camera calibration for sports sequences employs a special model for the arrangement of lines in the playfield, which is usually defined in the rules
of the game. After detecting lines in the input image, a combinatorial search is carried out to establish correspondences between lines in the input image and lines in the model. Comparable to the feature-based motion estimator, motion parameters are deduced from a set of corresponding lines. To reduce the overall computation time, a tracking step is additionally presented that updates the transformation parameters during camera motion with reduced computational complexity.

Chapter 14 describes a step towards the reconstruction of 3-D models from video images. During the work on background sprites and their generalization to multi-sprites, the question arose how the sprite images, which are informative pictures by themselves, can be best presented to the user. Usually, very wide-angle images are presented in the form of panoramic images, which are actually a mapping of the environment on a cylinder instead of a plane, as it is the case for MPEG-4 compliant background sprites. However, the disadvantage of panoramic images is that complete 360 -degree views are unwrapped into one rectangular image with the consequence that all straight lines in the image become bent and, more importantly, the viewer has no good orientation in the image because he looks into all directions at the same time. In order to provide a more intuitive presentation for wide-angle views, a visualization technique is developed which is specialized for the case of indoor environments. The visualization program recomputes the 3 -D shape of the room in which the image was captured and projects the panoramic image onto these virtual room walls. The advantage of this presentation is that the room shape helps the user in the orientation, making it clear which part of the image corresponds to which wall. This concept is further generalized with an algorithm to reconstruct the complete floor plan from several panoramic images. This enables to conduct virtual walk-throughs in the reconstructed rooms.

### 1.6 Contributions of the author

Most parts of the chapters in this thesis have been published in conference proceedings or scientific journals. An overview of which chapters are covered by corresponding publications is summarized in Table 1.1.

## Part I - An Automatic Video-Object Segmentation System

The concept of Part I of the thesis is to provide the reader with a complete discussion of a segmentation system in every detail. Because of this, some chapters also comprise background information, additional to the contributions of the author. In particular, Chapter 2 provides an introduction
to projective geometry, which can be skipped by the reader who is familiar with this topic.

The implementation of a complete segmentation system is a difficult task because of the complexity of the system. Various algorithm types and techniques covering many research areas have to be combined, like globalmotion estimation, feature extraction, statistics (Markov Random Fields), and various linear, non-linear, and combinatorial optimization techniques. Each of the processing steps has to be designed carefully, since a low accuracy in one step can lead to a complete failure of the system. Consequently, it was of significant importance to evaluate different alternatives for each processing step and select those algorithms that provide the most robust result when combined. For example, only the combination of the featurebased motion estimator with the direct estimation leads to high-accuracy parameters as well as robustness against fast camera motion, and the accuracy limitations of the motion estimator requires the adaptation of the change-detection algorithm with the risk-map approach.

An important algorithmic invention is the use of multi-sprites as a replacement for single static background images (Chapter 6). This technique has made it possible for the first time to process arbitrary camera motion. Although this is a crucial part to enable a practical implementation, this problem has been overlooked in the literature. The attractivity of our multi-sprite approach is that the problem is solved in an optimal way, also minimizing the MPEG-4 sprite-coding cost.

The two papers [59, 55] about multi-sprite coding both received the Best Student Paper award at the SPIE Visual Communications and Image Processing conference 2004, and at the 24th Symposium on Information Theory in the Benelux, 2003.

Another new development is the algorithm for background synthetization (Section 5.3). Compared with previous algorithms, this new algorithm also succeeds in reconstructing the background if it is visible for only short periods of time. This is important, because for a given video-sequence, the total observation time can be short (only one camera-pan). Because of this reason, existing algorithms that were primarily designed for background reconstruction in surveillance video, where the same scene is observed for a long time, cannot be applied.

The segmentation system was summarized in a book chapter [62] and it will also be presented by the author as a tutorial at the IEEE International Conference on Consumer Electronics 2006. Furthermore, the research on segmentation and object models (see below) has led to the organization of a special session about content analysis at the same conference.

## Part II - Segmentation using object models

Because we made the observation that a semantically meaningful segmentation requires pre-knowledge about the object to be extracted, we explore in Chapters 9 and 10 how this object description can be specified. We combined ideas of image databases supporting region-based queries [23, 17, 112] with articulated-object models as they are used for objecttracking applications [154, 70]. This is implemented in a new integrated framework supporting the creation of object models from sample images, as well as two algorithms for detecting the objects in real-world or cartoon images. We extended the concept of a 1:1-matching as it is used for tracking to an $1: N$-matching to enable a complete coverage of the object to be segmented. The object-detection algorithm uses a cascade of steps (candidate-region detection, skeleton-tree based graph-matching, extension of the mapping from an isomorphism to an homomorphism) to be computationally efficient. Finally, the object detection is combined with a color segmentation to obtain accurate object boundaries.

In Chapter 11 a new approach for semi-automatic segmentation is developed. Based on the Intelligent Scissors algorithm, we present an interactive segmentation tool that is easier to use and which also comprises a tracking component. Our tool replaces the shortest-path search with a shortest circular-path search. This not only provides a more intuitive user-interface, but it is also used in the tracking step. The tracking step is special because it uses a model of the object that is derived automatically from a previously segmented image. A main innovation in this chapter is the development of the circular-path search algorithm. It is the fastest algorithm currently known for planar graphs, with a typical computation complexity equal to the ordinary shortest-path search. This algorithm is generic and can be used for many applications apart from our manual segmentation tool, such as shape matching.

## Part III - From camera motion to 3-D models

The contribution of Chapter 12 is the integration of the multi-sprite concept into the autocalibration for rotational cameras. This enables the recovery of physical camera parameters from projective-motion parameters for unrestricted camera motion.

In Chapter 13 discusses a special kind of model: the model of a sports court to compute camera-calibration parameters. In order to obtain a robust court detection, the algorithm is based on line features. Similar to the graph-based object models, we use a combinatorial optimization to establish the correspondences between image features and the model. Our
results show that a specialized model can lead to highly robust object detection, invariant to observation conditions like illumination or court colors. It is also interesting to note that the robustness of the algorithm allows to adapt it to various kinds of sport by simply exchanging the court model.

Later, the work on camera calibration for sport sequences was integrated into the Philips Cassandra demonstrator [133] that was presented at the IEEE International Conference on Multimedia and Expo (ICME), 2005. The research in this area has also led to a special session on that topic at the same conference, co-organized by the author and Xinguo Yu. The calibration algorithm also builds the basis of ongoing research on sports analysis, extending it to a complete analysis system [81]. Furthermore, several international research groups employed our algorithm in their tennis or soccer analysis systems [89, 116].

Due to the work on background sprites, the author was invited in 2003 to stay at the Stanford Center for Innovations in Learning (SCIL) in the context of the Diver project [142], in which the human interaction with panoramic-video content was studied. In this time, the author developed the room reconstruction algorithm for rectangular rooms, presented in Chapter 14. Later, the algorithm was extended for general floor plans. A main contribution is the new approach of combining pre-knowledge about the room shape with measurement data from the panoramic image. Using the panoramic image for obtaining measurements is convenient since internal camera parameters like the focal-length are easily computed from the panoramic image instead of being estimated a difficult process.

## Appendices

Apart from the work on segmentation, the author contributed to the German BMBF project "L3-Lifelong Learning". In this project, the author contributed a video-database application featuring an automatic videosummary generation (Appendix A). Furthermore, the segmentation system has also been integrated into a video-abstracting system that was developed in the context of the European ECHO project (European CHronicles Online) [104]. In this project, a video-archiving system for historical films was established.

Appendix E describes early work about color-segmentation that has later been integrated into the model-based object detection. This color segmentation features a new speed-improved variant of the region-merging algorithm and a multi-stage approach, in which the merging criterion is switched during the segmentation. The paper about multi-stage segmentation [12] also received a best student paper award at the 22nd Symposium on Information Theory in the Benelux, 2001.

[^1]

| Part II - Segmentation Using Object Models |
| :---: |
| $9 \quad$ "Recognition of User-Defined Video Object Models using | Weighted Graph Homomorphisms", SPIE IVCP, 2003, [56]

Specification of objects by attributed graph models. Detection of the user-supplied object model by detection of graph homomorphisms ( $1: N$ mapping). Efficient implementation using a matching algorithm based on dynamic programming. Algorithm is specific to cartoon sequences.

| Chapter | Publication title and contribution - continued |
| :---: | :---: |
| 10,E | "A Segmentation System with Model Assisted Completion of Video Objects", SPIE VCIP, 2003, [58] <br> Graph-based object models are applied to the detection of objects in natural video sequences. Model matching yields approximate location of object. Accurate boundaries are computed by combining the positional hint with pixel-level color segmentation. |
| E | "Multi-Stage Region Merging for Image Segmentation", 22nd Symposium on Information Theory in the Benelux, 2001, [12] <br> A color-segmentation algorithm based on region-merging that features a multi-stage segmentation, where the segmentation criterion is changed during the segmentation to adapt it to the typical signal characteristics at the respective stage. (Award paper.) |
| E | "Towards Real-Time MPEG-4 Segmentation: A Fast Implementation of Region-Merging", 21st Symposium on Information Theory in the Benelux, 2000, [46] <br> Fast implementation of region-merging color segmentation. |
| 11 | "Corrisor Scissors: A Semi-Automatic Segmentation Tool Employing Minimum-Cost Circular Paths", IEEE ICIP, 2004, [68] |
| 11 | Development of the Corridor Scissors segmentation algorithm as an extension of the Intelligent Scissors algorithm. A first fast algorithm for shortest circular paths is presented. <br> (drafted as journal paper) |
|  | Enhancement of the circular-path search algorithm to remove special cases and achieve a high computation speed for any input data. |

Part III - From Camera Motion to 3-D Models
12 "Estimating Physical Camera Parameters for 3DAV video coding", 25th Symposium on Information Theory in the Benelux, 2004, [48]
and
"Estimating Physical Camera Parameters based on Multi-Sprite Motion Estimation", SPIE IVCP, 2005, [49] Auto-calibration of rotational cameras. Rotation angles and focal-length are estimated from the projective-motion parameters obtained in the segmentation system. Speciality of this algorithm is that it supports the estimation also in the case of unrestricted camera motion, since the multi-sprite technique is integrated.

| Chapter | Publication title and contribution - continued |
| :---: | :---: |
| 13 | "Robust Camera Calibration for Sport Videos using |
|  | Court Models", SPIE Storage and Retrieval Methods and Applications for Multimedia, 2004, [65] |
|  | Estimation of mapping between image coordinates and realworld coordinates for sport sequences, based on a model of the playfield. Lines are estimated and a set of four corresponding |
|  | lines between the image and the model are determined. Also supports a tracking step to reduce the computational complexity after initial calibration. |
| 13 | "Fast Camera Calibration for the Analysis of Sport Sequences", IEEE ICME, 2005, [63] |
|  | Enhancement of the camera calibration algorithm that reduces the computation time by only requiring the matching of two line-segments for the calibration instead of four correspondences. |
| 13 | "Current and Emerging Topics in Sports Video processing", IEEE ICME, 2005, [199] (co-work with Xinguo Yu) |
|  | Overview of techniques and applications for sport-video analysis. |
| 14 | "Reconstructing Virtual Rooms From Panoramic Images", 26th Symposium on Information Theory in the Benelux, 2005, [52] |
|  | Geometry reconstruction of rectangular rooms from panoramic images. Generation of 3-D room-models where the walltextures are extracted from the panoramic image. |


| Appendices |  |
| :---: | :---: |
| A | "Robust Clustering-Based Video-Summarization with In- <br> tegration of Domain-Knowledge", IEEE ICME, 2002, [61] <br> Development of a video-summarization algorithm that can in- <br> corporate pre-knowledge about scenes that should be excluded <br> from the summary. |
| A,2-8 | "Automatic generation of video summaries for histori- <br> cal films", IEEE ICME, 2004, [104], (co-authored with |
|  | Stephan Kopf) |
| and |  |

Table 1.1: Contributions of the author in the respective chapters. If not noted otherwise, the thesis author is also the first author of the paper.

## Part I

## An Automatic Video Segmentation System

Equations are just the boring part of mathematics.
I attempt to see things in terms of geometry.
(Stephen Hawking)


## Projective Geometry

The geometric relations between objects in the 3-D world and a 2-D image of them is of central importance when we want to estimate the motion of the camera from a sequence of images. In particular, we need a geometric model that describes the observed motion fields resulting from different kinds of camera motion. Object motion and image formation can be expressed mathematically with the concept of projective geometry. This chapter gives an introduction to projective geometry as far as it is required to understand the subsequent chapters. We start with defining the projective space and deriving basic operations on points and lines in it. We proceed by discussing geometric transformations in the projective plane and in the 3-D Euclidean space, including the projection of 3-D space onto a 2-D image plane. Finally, we construct a detailed model of the image formation process for a moving camera. The reader familiar with projective geometry and its application to computer vision can jump to the concluding section of this chapter where we summarize the notation that will be used for the following chapters.

Chapter 2. Projective Geometry

### 2.1 Introduction

The theory of projective geometry establishes the basis for three-dimensional computer vision and computer graphics. Compared to Euclidean geometry, it facilitates the description of rigid ${ }^{1}$ three-dimensional motion and the perspective projection onto planar images, because it enables to formulate both with linear algebra techniques. In particular, projective geometry provides a uniform description of situations that require special cases in Euclidean geometry, like the intersection of parallel lines. Consequently, projective geometry has become the standard technique to describe three-dimensional geometry.

Since we employ the mathematical tools of projective geometry in many of the succeeding chapters, this chapter gives an introduction to projective geometry. Thereby, we concentrate on the aspects that are required in our applications. First, we describe the basic notion of homogeneous coordinates and we show how points and lines are represented in two-dimensional images. Second, we consider geometric transformations in the plane. We show how elementary operations like translations or rotations can be described and we introduce the classes of affine and projective motion. Third, the planar transformations are generalized to three-dimensional rigid motion. This gives us the required tools to describe the complete geometric image formation process as a concatenation of elementary operations. Finally, we deduce motion models for important classes of camera motion like rotational camera motion.

While this chapter only gives a brief introduction, a thorough discussion of 3-D geometry, estimation of camera parameters, and multi-view geometry can be found in the books [85] and [69]. Book [85] concentrates on the description of camera geometry for 3-D reconstruction, while [69] covers a wider range and also includes computer-vision algorithms, e.g., for multi-view tracking of objects. A detailed introduction to the mathematical theory of projective geometry can be found in [166]. Many practical aspects of projective transformations for image warping are described in [90]. These include optimal filtering of the transformed image to prevent aliasing and fast algorithms to compute transformation parameters and inverse transforms. Finally, [99] and [127] provide condensed introductions to the field.

[^2]
### 2.2 Projective spaces

Elementary geometry is usually described in the Euclidean space, which is a direct description of space as we perceive it with our human intuition. In this formulation, points in $n$-dimensional Euclidean space $\mathbb{E}^{n}$ are simply represented as vectors of length $n$. However, this supposedly convenient definition of Euclidean space has several practical drawbacks. First, there is no notion about points at infinity, so that these must be considered separately as a special case. For example, this problem arises when computing the intersection of parallel lines, which is not defined in Euclidean space. A second drawback becomes apparent when we use geometric transforms to describe object motion. Even for the basic types of motion, different formalisms are required. While translation is described using a vector addition, rotation is written as a matrix multiplication, and the perspective projection of points onto a plane requires a division operation.

The concept of projective spaces, provides an alternative convenient formalism to describe all types of rigid motion and the perspective projection in a unified way. Moreover, points at infinity are an integral part of the projective space and require no special consideration.

The remainder of Section 2.2 introduces the construction of projective space, its relationship to Euclidean space, and it introduces basic operations on points and lines.

### 2.2.1 Homogeneous coordinates

In the $n$-dimensional Euclidean space $\mathbb{E}^{n}$, each point is written as a vector of length $n$, where each component is the position along one coordinate axis. In contrast, points in $n$-dimensional projective space $\mathbb{P}^{n}$ are represented by $n+1$ dimensional vectors. The construction is such that each point $\left(x_{1}, \ldots, x_{n}\right)^{\top}$ in $\mathbb{E}^{n}$ corresponds to a one-dimensional subspace $\left(w x_{1}, \ldots, w x_{n}, w\right)^{\top}$ in $\mathbb{P}^{n}$ with the free scaling parameter $w \neq 0$. Euclidean space $\mathbb{E}^{n}$ can be embedded into $\mathbb{P}^{n}$ in a simple way by using the canonical injection $\mathbb{E}^{n} \ni\left(x_{1}, \ldots, x_{n}\right)^{\top} \mapsto\left(x_{1}, \ldots, x_{n}, 1\right)^{\top} \in \mathbb{P}^{n}$. In the reverse direction, Euclidean coordinates can be recovered by the mapping

$$
\begin{equation*}
\mathbb{P}^{n} \ni\left(x_{1}, \ldots, x_{n}, w\right)^{\top} \longmapsto\left(\frac{x_{1}}{w}, \ldots, \frac{x_{n}}{w}\right)^{\top} \in \mathbb{E}^{n} \tag{2.1}
\end{equation*}
$$

This projective representation is denoted as the point's homogeneous coordinates. A direct consequence of the definition is the important property that homogeneous coordinates are scaling invariant. Thus, $\left(x_{1}, \ldots, x_{n}, w\right)^{\top}$ and $\left(\lambda x_{1}, \ldots, \lambda x_{n}, \lambda w\right)^{\top}$ represent the same point for all $\lambda \neq 0$. In other words, each Euclidean point $\mathbf{p}$ is represented by one equivalence class $E(\mathbf{p}) \subset \mathbb{P}^{n}$


Figure 2.1: The Euclidean plane $\mathbb{E}^{2}$ can actually be viewed as the plane $w=1$ in projective space $\mathbb{P}^{2}=\left\{(x, y, w)^{\top}\right\}$.
in the projective space. All vectors in the equivalence class can be obtained by a uniform scaling of the coordinates.

Projective points with $w=0$ represent ideal points at infinity with no correspondence in Euclidean space. For these points at infinity, the first $n$ vector components indicate the direction in which the point is located. The non-sense null-vector $(0, \ldots, 0)^{\top}$ is not included in $\mathbb{P}^{n}$.

## Visualization of $\mathbb{P}^{2}$

In the following discussion, the projective plane $\mathbb{P}^{2}$ is of central importance. For this case, the relation between points in Euclidean space $\mathbb{E}^{2}$ and corresponding points in projective space $\mathbb{P}^{2}$ can be visualized easily. We draw $\mathbb{P}^{2}=\left\{(x, y, w)^{\top}\right\}$ as a three-dimensional space with the dimensions $x, y, w$ (see Figure 2.1). In this space, the equivalence class induced by a point $\mathbf{p}$ in $\mathbb{P}^{2}$ can be visualized as the line through the origin and the point $\mathbf{p}$. The intersection of this line with the plane $w=1$ defines the coordinate in the Euclidean plane. Since a scaling of the homogeneous coordinates of point $\mathbf{p}$ by a non-zero constant only moves the point along the line, its position on the Euclidean plane stays the same. This explains the scaling invariance of homogeneous coordinates. By choosing the special representation with the last coordinate $w=1$, it becomes clear that the Euclidean plane itself can be embedded into the projective space as the plane at $w=1$.

Ideal points $\mathbf{p}$ with $w=0$ correspond to lines parallel to the Euclidean plane and thus intersect them ideally at infinity. Since these lines are parallel to the Euclidean plane $w=1$, the ideal points are not part of the Euclidean space. This is consistent with the definition of Euclidean space, because it has no notion about points at infinity.


Figure 2.2: Lines $\mathbf{1}=(a, b, c)^{\top}$ in the $\mathbb{P}^{2}$ can be represented as planes through the origin with normal vector $\mathbf{1}$. The intersection of this plane with the Euclidean plane forms the corresponding line in the Euclidean plane.

## Normalization of homogeneous coordinates

It is common practice to normalize homogeneous coordinates to have unit norm $\|\mathbf{p}\|=1$ during computations. Regarding the above-mentioned visualization model, this has the effect that all points in homogeneous coordinates lie on the unit sphere around the origin of $\mathbb{P}^{2}$. The normalization resolves numerical problems that can occur when working with coordinates of significantly different magnitude. We do not explicitly mention the normalization of coordinates in the following, but simply note that this normalization procedure can be applied at arbitrary times during the computation, whenever it seems appropriate.

### 2.2.2 Lines in the projective plane

In Euclidean geometry, lines are defined as $a x+b y+c=0$, where $a, b, c$ are the line parameters. Note that these line parameters are invariant to scaling with a constant $w \neq 0$, since $w(a x)+w(b y)+w c=0$ describes the same line. If we denote the line parameters by the vector $\mathbf{l}=(a, b, c)^{\top}$ and specify points using homogeneous coordinates $\mathbf{p}=(w x, w y, w)^{\top}$, we can rewrite the line equation conveniently as $\mathbf{l}^{\top} \cdot \mathbf{p}=0$.

It is possible to visualize lines in the projective plane in the same way as we did with points in the last section. The construction represents each line in $\mathbb{E}^{2}$ as a plane in $\mathbb{P}^{2}$ which goes through the origin and which has a normal vector equal to the line parameters 1 . This plane intersects the Euclidean plane $w=1$ in the desired line (see Figure 2.2). Since the norm of the plane normal is irrelevant, the line parameters are also invariant to scaling with a non-zero value.


Figure 2.3: (a) The intersection point $\mathbf{p}$ of two lines $\mathbf{l}_{\mathbf{1}}, \mathbf{l}_{\mathbf{2}}$ can be calculated by $\mathbf{p}=\mathbf{l}_{\mathbf{1}} \times \mathbf{l}_{\mathbf{2}}$, since the vector $\mathbf{p}$ must lie on both planes and hence, it must be orthogonal to $\mathbf{l}_{\mathbf{1}}$ and $\mathbf{l}_{\mathbf{2}}$. (b) For two parallel lines, the intersection is an ideal point at infinity.

Similarly to the points at infinity, a special line at infinity is defined by $\mathbf{l}_{\infty}=(0,0,1)^{\top}$. Since $\mathbf{l}_{\infty}^{\top} \cdot(x, y, w)^{\top}=0$ iff $w=0$, the line $\mathbf{l}_{\infty}$ is the set of all points at infinity. Note that just like the points at infinity, $l_{\infty}$ has no correspondence in Euclidean geometry. In our visualization, where we consider the vector of line parameters to be a plane normal, $\mathbf{l}_{\infty}$ defines a plane parallel to the Euclidean plane, intersecting it ideally at infinity (see Figure 2.4(b)).

## Intersection of lines

Let $\mathbf{l}_{\mathbf{1}}$ and $\mathbf{l}_{\mathbf{2}}$ denote two lines in $\mathbb{P}^{2}$. Visualized in the $(x, y, w)$-space, these can be viewed as two planes through the origin with normal vectors $\mathbf{l}_{1}$ and $\mathbf{l}_{\mathbf{2}}$ as shown in Figure 2.3(a). The intersection point $\mathbf{p}$ of the two lines $\mathbf{l}_{\mathbf{1}}, \mathbf{l}_{\mathbf{2}}$ obviously must lie on both planes in the $(x, y, w)$-space. This means that $\mathbf{p}^{\top} \cdot \mathbf{l}_{1}=0$ and $\mathbf{p}^{\top} \cdot \mathbf{l}_{\mathbf{2}}=0$ must hold. In other words, $\mathbf{p}$ is orthogonal to both $\mathbf{l}_{1}$ and $\mathbf{l}_{2}$. Consequently, we can compute it as $\mathbf{p}=\mathbf{l}_{\mathbf{1}} \times \mathbf{l}_{\mathbf{2}}$.

Note that when using homogeneous coordinates, the intersection of two parallel lines is well defined and results in an ideal point at infinity (Figure $2.3(\mathrm{~b})$ ). In this case, the intersection point has $w=0$, and the other two coordinates indicate the direction towards the intersection at infinity.

## Line through two points

Let $\mathbf{p}_{\mathbf{1}}$ and $\mathbf{p}_{\mathbf{2}}$ be two points in $\mathbb{P}^{2}$. Following a similar approach as above, we remember that $\mathbf{p}_{\mathbf{1}}$ and $\mathbf{p}_{\mathbf{2}}$ can be visualized by two rays, starting from


Figure 2.4: The line $\mathbf{l}$ through two points $\mathbf{p}_{\mathbf{1}}, \mathbf{p}_{\mathbf{2}}$ is defined by a plane, which must include the two vectors $\mathbf{p}_{\mathbf{1}}, \mathbf{p}_{\mathbf{2}}$. Consequently, its normal vector $\mathbf{l}$ must be orthogonal to both vectors $\mathbf{p}_{\mathbf{1}}, \mathbf{p}_{\mathbf{2}}$.
the origin of the $(x, y, w)$-space (Figure 2.4). To find the projective parameters of the line through $\mathbf{p}_{\mathbf{1}}$ and $\mathbf{p}_{\mathbf{2}}$, we have to find a plane in $(x, y, w)$-space which contains $\mathbf{p}_{\mathbf{1}}, \mathbf{p}_{\mathbf{2}}$ and the origin. If we specify this plane by its plane normal $\mathbf{l}$, this means that $\mathbf{l}^{\top} \cdot \mathbf{p}_{\mathbf{1}}=0$ as well as $\mathbf{l}^{\top} \cdot \mathbf{p}_{\mathbf{2}}=0$ must hold. Therefore, we can determine $\mathbf{l}$ by using the cross product $\mathbf{l}=\mathbf{p}_{\mathbf{1}} \times \mathbf{p}_{\mathbf{2}}$. Note that the line through two points at infinity results in the line $\mathbf{l}_{\infty}$.

The similarity of the expressions for computing intersection points between two lines and the expression for computing the line through two points is not accidental. In fact, it is one example of the duality principle between points and lines in projective space. This fundamental property implies that every theorem in projective space $\mathbb{P}^{2}$ stays true if all references to lines and points are interchanged.

### 2.3 Geometric transformations in 2-D

In this section, we introduce the projective transformation, which is used frequently to describe object motion as a geometric transformation in the image plane. We also derive some basic properties of projective transforms and introduce affine motion as an important sub-class of projective transforms. The class of affine transformations allows us to develop an intuitive understanding of the physical meaning of the transform parameters. Finally, we see how the projective motion model, which is commonly used for motion analysis in its inhomogeneous formulation, can be derived directly from the homogeneous definition.

### 2.3.1 Projective transformation

A projective transformation is defined as a linear transformation between homogeneous coordinates. We only consider non-degenerate cases where the transform is invertible. Since the transform is linear and invertible, it can be written as a multiplication with a non-singular matrix $\mathbf{H}=\left\{h_{i k}\right\}$. For a projective space $\mathbb{P}^{n}$, the matrix is of size $(n+1) \times(n+1)$. In the planar case $\mathbb{P}^{2}$, we get specifically

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.2}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{lll}
h_{00} & h_{01} & h_{02} \\
h_{10} & h_{11} & h_{12} \\
h_{20} & h_{21} & h_{22}
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
w
\end{array}\right) .
$$

The matrix entries $\left\{h_{i k}\right\}$ are the transform parameters. Note that because of the use of homogeneous coordinates, the transformation matrix is only defined up to a scaling factor. Hence, in $\mathbb{P}^{2}$, the transform has only 8 degrees of freedom even though the transformation matrix has nine elements.

Outlook: Estimation of projective transforms.
To estimate the transformation between two frames, we will investigate feature-based motion estimators in Chapters 3 and 4. The idea is that we can identify a set of points $\left(x_{i}, y_{i}, 1\right)$ in one image and a corresponding set of points $\left(x_{i}^{\prime}, y_{i}^{\prime}, 1\right)$ in the second image. When inserting pairs of points into Eq. (2.2), we obtain two constraints for each pair of points. With at least four pairs of points and one additional constraint (e.g., $h_{22}=1$ ) to remove the scaling invariance, we can determine the transformation between the two images. These transformation parameters can then be used, e.g., to stitch both images together (Fig. 2.5).

## Equivalence to collineations and the mapping of lines

A collineation in the plane is defined as a transform that maps lines onto lines. It can be shown that every collineation can be written as a projective transformation, and vice versa. In the following, we only give the proof for one direction. Given a non-singular transformation matrix $\mathbf{H}=\left\{h_{k l}\right\}$, we can easily show that lines are always mapped onto lines. To prove this, let $\left\{\mathbf{p}_{\mathbf{i}}\right\}$ be a set of points that all lie on a line $\mathbf{l}$, so $\mathbf{l}^{\top} \cdot \mathbf{p}_{\mathbf{i}}=0$. Since we assumed that $\mathbf{H}$ is invertible, this equals $\mathbf{l}^{\top}\left(\mathbf{H}^{-1} \mathbf{H}\right) \mathbf{p}_{\mathbf{i}}=0$. By changing parenthesis, we can read this equation as $\left(\mathbf{l}^{\top} \mathbf{H}^{-1}\right)\left(\mathbf{H} \mathbf{p}_{\mathbf{i}}\right)=0$. But this means that the transformed points $\mathbf{H} \mathbf{p}_{\mathbf{i}}$ all lie on the line $\mathbf{l}^{\top} \mathbf{H}^{-1}$. Thus, the projective transformation preserved lines. The reverse direction, which

(a) Two images with four point-correspondences.

(b) Both images can be aligned using the computed projective transform.

Figure 2.5: Images captured with a rotating camera can be aligned by identifying four point-correspondences and computing the projective transformation matrix from these correspondences.
states that every collineation can be written as a projective transform, is far more complicated to show, and the proof is omitted here.

The above proof also gives us a useful side-result. For a given transformation $\mathbf{p}^{\prime}=\mathbf{H p}$ between points, we can find a corresponding transformation that maps a line parameter vector $\mathbf{l}$ to the line parameters $\mathbf{l}^{\prime}$ of the transformed line. From the above proof, it follows immediately that

$$
\begin{equation*}
\mathbf{l}^{\top}=\mathbf{l}^{\top} \mathbf{H}^{-1} \quad \text { or, equivalently, } \quad \mathbf{l}^{\prime}=\mathbf{H}^{-\top} \mathbf{l} . \tag{2.3}
\end{equation*}
$$

Consequently, we can say that for a point-transform $\mathbf{H}$, the corresponding line-transform is $\mathbf{H}^{-\top}$.

Outlook: Estimation of transform parameters based on corresponding lines.

Equation (2.3) does not only provide the parameters of lines after a transformation. The equation can also be used to determine the transformation parameters if a set of corresponding lines are known in two images. The approach is similar to estimating the transformation parameters from a set of points, with the only difference that an additional matrix inversion has to be computed. In some applications, the estimation of lines is more reliable than the estimation of points, and the approach to estimate the transformation directly from the lines is more convenient (Fig. 2.6). We use this technique in Chapter 13 to obtain a camera calibration for sport sequences. In this case, we detect the lines of the sports court and match them with lines in a model of the court

## Equivalence to perspective plane-to-plane mapping

One transform of special importance for computer graphics is the perspective projection of one plane onto another. As depicted in Figure 2.7, points $\mathbf{p}_{\mathbf{i}}$ on plane $\Pi$ are projected onto points $\mathbf{p}_{\mathbf{i}}^{\prime}$ on plane $\Pi^{\prime}$ along rays emanating from the origin $\mathbf{0}$. Consider the line $\mathbf{l}$ through the points $\mathbf{p}_{\mathbf{1}}$ and $\mathbf{p}_{\mathbf{2}}$. The plane through the points $\mathbf{0}, \mathbf{p}_{\mathbf{1}}, \mathbf{p}_{\mathbf{2}}$ intersects $\Pi$ at $\mathbf{l}$. The same plane will intersect $\Pi^{\prime}$ in a line $\mathbf{l}^{\prime}$. Consequently, each perspective plane-to-plane mapping is also a collineation and therefore a projective transform.

### 2.3.2 Affine motion

Before we discuss properties of the general perspective transform, let us consider the important sub-class of affine motion. From elementary geometry, we know that planar affine motion in $\mathbb{E}^{2}$ is written as

$$
\binom{x^{\prime}}{y^{\prime}}=\left[\begin{array}{ll}
a_{00} & a_{01}  \tag{2.4}\\
a_{10} & a_{11}
\end{array}\right]\binom{x}{y}+\binom{t_{x}}{t_{y}},
$$

comprising a $2 \times 2$ transformation matrix $\mathbf{A}=\left\{a_{k l}\right\}$ and a translation vector $\mathbf{t}=\left(t_{x}, t_{y}\right)^{\top}$. The matrix $\mathbf{A}$ can always be factorized into a sequence of the elementary operations rotation, non-isotropic scaling, and skewing. The transformation matrices for each of these elementary operations is given in Table 2.1. It is easy to see that all of these elementary operations map parallel lines onto parallel lines. Consequently, this property also holds for the general affine transform, since it can be decomposed into a sequence of elementary operations.


Figure 2.6: Rectifying a tennis court image based on the court lines. It is more robust to determine the transformation from line correspondences, because the court-lines are not occluded as often as specific feature-points.

We can also write the affine transform using homogeneous coordinates, in which case we can integrate the translation vector into the matrix, leading to the unified formulation

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.5}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{ccc}
a_{00} & a_{01} & t_{x} \\
a_{10} & a_{11} & t_{y} \\
0 & 0 & 1
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
w
\end{array}\right) .
$$

The affine transformation has six degrees of freedom. However, in this general form, the affine transformation includes types of motion that are no valid motion of rigid objects. Therefore, we can further restrict the transformation by disallowing skewing (force $k=0$ ) and allowing only isotropic scaling $\left(s:=s_{1}=s_{2}\right)$. As a consequence, only four parameters


Figure 2.7: In a projective plane-to-plane mapping, points on plane $\Pi$ are projected along rays starting from the origin $\mathbf{0}$ onto the plane $\Pi^{\prime}$. This plane-to-plane transform maps lines on $\Pi$ onto lines on $\Pi^{\prime}$.
remain, namely one for rotation, one for isotropic scaling, and two for the translation vector. In this case, we call the transform a similarity transform, which can be expressed as

$$
\left[\begin{array}{ccc}
s \cos \alpha & s \sin \alpha & t_{x}  \tag{2.6}\\
-s \sin \alpha & s \cos \alpha & t_{y} \\
0 & 0 & 1
\end{array}\right]
$$

with $\alpha$ being the rotation angle and $s$ representing the isotropic scaling factor.

In Section 2.4.3, we will see that the affine motion model can describe the motion of planar objects in 3-D space if the camera is located at an infinite distance. Even though this is never true in practice, the affine camera model is still often used as an approximation if the distance between camera and scene is large and the motion is small.

### 2.3.3 Projective motion

After we have covered the case of affine motion and described it as as a sub-class of projective motion, we now make the step to the general projective transformation. If we recall Equation (2.5) for affine motion in homogeneous notation, we see that the last row of the matrix is always $(0,0,1)$. Therefore, $w^{\prime}=w$ holds for every possible transform, i.e., the transform does not change the $w$-coordinate. Considering our visualization


Table 2.1: Transformation matrices $\mathbf{A}$ for the elementary affine transforms, written in inhomogeneous form $\mathbf{p}^{\prime}=\mathbf{A p}+\mathbf{t}$. Isotropic scaling is obtained for $s_{1}=s_{2}$.
model from Figure 2.1, this means that motion of a point is only performed within a plane parallel to the Euclidean projection plane. Since no change of depth occurs, we do not observe a perspective projection effect.

If we generalize the model such that the last line can hold arbitrary values, we have the general projective transformation. As we have seen previously, this models a general plane-to-plane transform, which allows to describe the 3-D motion of a plane, viewed by a pinhole camera. The difference to the affine transform is that, in general, parallel lines do not remain parallel after the transform (Figure 2.8(a)). Instead, the projective transform has the property that parallel lines are mapped to lines that intersect in a common vanishing point. Note that this includes the special case that parallel lines remain parallel, since parallel lines are intersecting at an ideal point at infinity. Because parallel lines intersect at the same vanishing point, it is clear that the location of the vanishing point can only depend on the direction of the lines in the source image.

## The horizon and the line at infinity

The set of all vanishing points is called the vanishing line. In the real world, we know this vanishing line of the ground plane as the horizon (Figure 2.8(b)). Since the horizon line and its relationship to the line at infinity plays a special role, we explore in this section how the parameters of the horizon line can be obtained from a transformation matrix. Consider the case depicted in Figure 2.8(b), where an object plane $\Pi^{\prime}$ is projected by a general projective transformation onto an image plane $\Pi$. Let us denote the transform of image coordinates back onto the object plane by $\mathbf{H}$. In this formulation, we can define the horizon line as those points $\mathbf{p}$ that are


Figure 2.8: Planes under perspective projection.
mapped to points at infinity by the transform H. Recalling that points $(x, y, w)$ lie at infinity iff $w=0$, it must hold that

$$
\left(\begin{array}{l}
x^{\prime}  \tag{2.7}\\
y^{\prime} \\
0
\end{array}\right)=\left[\begin{array}{lll}
h_{00} & h_{01} & h_{02} \\
h_{10} & h_{11} & h_{12} \\
h_{20} & h_{21} & h_{22}
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
w
\end{array}\right)
$$

From this, we derive the constraint that for points on the horizon,

$$
\begin{equation*}
\left(h_{20}, h_{21}, h_{22}\right) \cdot \mathbf{p}=0 \tag{2.8}
\end{equation*}
$$

must hold. Since this equation has the same form as a general line equation, we can regard the last row of the matrix $\mathbf{H}$ as the line parameters for the horizon line $\mathbf{l}_{\mathbf{h}}$. Hence, we have the simple result that

$$
\begin{equation*}
\mathbf{l}_{\mathbf{h}}=\left(h_{20}, h_{21}, h_{22}\right)^{\top} \tag{2.9}
\end{equation*}
$$

Let us finally examine the special case of an affine transform. We know already that the affine transform preserves parallelism. Consequently, the vanishing points will always lie on the line at infinity. This observation is supported by the fact that the last line of the affine transformation matrix is $(0,0,1)$, which equals $\mathbf{l}_{\infty}^{\top}$. Hence, the location of the line at infinity is invariant to any affine motion.

## Inhomogeneous formulation of the projective transformation

Writing the projective transformation with the inhomogeneous formulation, we have

$$
\begin{equation*}
x^{\prime}=\frac{h_{00} x+h_{01} y+h_{02}}{h_{20} x+h_{21} y+h_{22}}, \quad y^{\prime}=\frac{h_{10} x+h_{11} y+h_{12}}{h_{20} x+h_{21} y+h_{22}} . \tag{2.10}
\end{equation*}
$$

Since the parameters $\left\{h_{k l}\right\}$ are invariant to an overall scaling, it is common practice to apply the normalization $h_{22}=1$. Especially in the literature about motion estimation for video coding, the inhomogeneous form is usually chosen. However, it should be noted that this normalization is only possible if $h_{22} \neq 0$. It is important to know in which cases $h_{22}$ equals zero to decide if the inhomogeneous form is applicable.

The examination about the line at infinity gives us more insight into the case. We know that the horizon line in the destination image is given by $\left(h_{20}, h_{21}, h_{22}\right)$. The specific case $h_{22}=0$ induces that the horizon line has the form $h_{20} x+h_{21} y=0$, which is just the pencil of lines that goes through the origin. Consequently, the normalization $h_{22}=1$ is invalid iff the horizon line includes the origin. If this situation can occur in a specific application, the inhomogeneous form should not be used.

In our application of describing object motion by a perspective transform between successive frames, motion is so small that the normalization is usually not a problem. On the other hand, if motion is estimated over a large time distance (as it is the case in the background sprite construction), this can be a problem. However, especially for background sprite generation, other factors make the long-distance motion description impractial, even before we approach the problem of the inhomogeneous formulation.

Ignoring the cases where the normalization $h_{22}=1$ is invalid, we get the most widely-used formulation of projective motion as

$$
\begin{equation*}
x^{\prime}=\frac{h_{00} x+h_{01} y+h_{02}}{h_{20} x+h_{21} y+1}, \quad y^{\prime}=\frac{h_{10} x+h_{11} y+h_{12}}{h_{20} x+h_{21} y+1}, \tag{2.11}
\end{equation*}
$$

or, when using different symbols for the transformation parameters, to better reflect better their relation to the affine transform, we write it as

$$
\begin{equation*}
x^{\prime}=\frac{a_{00} x+a_{01} y+t_{x}}{p_{x} x+p_{y} y+1}, \quad y^{\prime}=\frac{a_{10} x+a_{11} y+t_{y}}{p_{x} x+p_{y} y+1} . \tag{2.12}
\end{equation*}
$$

We see that the affine transformation results as a special case of the perspective transformation for $p_{x}=p_{y}=0$. Because the denominator only disappears in the affine case, the perspective transformation is a non-linear transform. We will see in Chapter 4 that this complicates the parameter estimation process.

Chapter 2. Projective Geometry

### 2.4 Geometric transformations in 3-D

To describe object or camera motion in 3-D, we make use of the same formulation as for 2-D. We also use homogeneous coordinates in $\mathbb{P}^{3}$ even though we only consider affine motion. Affine motion is sufficient to describe rigid object motion, but the use of homogeneous coordinates enables to use the unified formulation in which affine motion can be described as a matrix multiplication.

### 2.4.1 Affine motion in 3-D

The generalization of the affine motion in 2-D to affine motion in 3-D is straightforward. In the 2-D case, we described affine motion in inhomogeneous coordinates by $\mathbf{p}^{\prime}=\mathbf{A p}+\mathbf{t}$, where $\mathbf{p}^{\prime}, \mathbf{p}, \mathbf{t}$ are two-component vectors and $\mathbf{A}$ is a $2 \times 2$ matrix. When we go to 3 -D, we can take the same equation and simply substitute the vectors by three-component vectors and the matrix by a $3 \times 3$ matrix.

In the 2-D case, we used homogeneous coordinates to unify the formulation. Clearly, we can apply the same approach also in the 3-D generalization of affine motion by augmenting vectors with a homogenization element and matrices with an additional row and column. As a consequence, translation can now also be described as a simple matrix multiplication. In its general form, affine motion in 3-D is consequently described by a multiplication with a matrix, where the last row is $(0,0,0,1)$ :

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.13}\\
y^{\prime} \\
z^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{cccc}
a_{00} & a_{01} & a_{02} & t_{x} \\
a_{10} & a_{11} & a_{12} & t_{y} \\
a_{20} & a_{21} & a_{22} & t_{z} \\
0 & 0 & 0 & 1
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) .
$$

A collection of the most important elementary transformations in this homogeneous formulation is shown in Table 2.2. Besides the basic affine operations translation, scaling, and rotation, a special operation is included to carry out a perspective projection from 3-D space onto a 2 -D plane. This is a special operation that will be described in Section 2.4.3.

### 2.4.2 Rotation in 3-D

Describing rotation in 3-D space is a surprisingly complicated problem. Several techniques for parameterization of rotations have been proposed, such as Euler angles or Quaternions [108]. We use the Euler-angle notation here since this mimics the frequently-used practice to describe rotation by elementary rotations around the coordinate system axes.
$\left.\begin{array}{ccccc}\hline \hline \text { translation } & \text { isotropic scaling } & \text { rotation } & \text { projection } \\ \hline\left[\begin{array}{llll}1 & 0 & 0 & t_{x} \\ 0 & 1 & 0 & t_{y} \\ 0 & 0 & 1 & t_{z} \\ 0 & 0 & 0 & 1\end{array}\right] & {\left[\begin{array}{llll}s & 0 & 0 & 0 \\ 0 & s & 0 & 0 \\ 0 & 0 & s & 0 \\ 0 & 0 & 0 & 1\end{array}\right]} & {\left[\begin{array}{cc}\mathbf{R} & \mathbf{0} \\ \mathbf{0}^{\top} & 1\end{array}\right]}\end{array} \begin{array}{cccc}f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0\end{array}\right]$

Table 2.2: Most important transformation matrices $\mathbf{H}$ expressed in homogeneous coordinates. Matrix $\mathbf{R}$ is a $3 \times 3$ rotation matrix; see Section 2.4.2 for details.

In the following, we only consider rotation around the coordinate system origin, which is sufficient for the successive discussion. Furthermore, we assume that the observer (i.e., the camera) is located at the origin and that observations are made in a local camera coordinate system.

Let us begin with a simple rotation around one of the coordinate system axes, which can be described like a rotation in two dimensions where the third dimension is not affected. This can be written as an elementary rotation matrix (Jacobi rotation matrix). In 3-D space, we obtain the three elementary rotation matrices

$$
\begin{gather*}
\mathbf{R}_{\mathbf{x}}(\alpha)=\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & c_{\alpha} & -s_{\alpha} \\
0 & s_{\alpha} & c_{\alpha}
\end{array}\right], \quad \mathbf{R}_{\mathbf{y}}(\beta)=\left[\begin{array}{ccc}
c_{\beta} & 0 & s_{\beta} \\
0 & 1 & 0 \\
-s_{\beta} & 0 & c_{\beta}
\end{array}\right]  \tag{2.14}\\
\mathbf{R}_{\mathbf{z}}(\gamma)=\left[\begin{array}{ccc}
c_{\gamma} & -s_{\gamma} & 0 \\
s_{\gamma} & c_{\gamma} & 0 \\
0 & 0 & 1
\end{array}\right]
\end{gather*}
$$

where we use the abbreviations $c_{\alpha}=\cos \alpha$, and $s_{\alpha}=\sin \alpha$. The signs of the $\sin (\cdot)$-terms have been chosen such that it conforms to a right-handed coordinate system (see Figure 2.11(a)). It is clear that the inverse rotation to some angle $\alpha$ is the rotation with the same angle in the opposite direction (i.e., a rotation with the angle $-\alpha$ ). Since the rotation matrices are skew symmetric and $\sin (-\alpha)=-\sin \alpha$, the inverse of each elementary rotation matrix is simply its transpose $\left(\mathbf{R}^{-1}=\mathbf{R}^{\top}\right)$.

According to Euler's rotation theorem, an arbitrary rotation can be decomposed into three successive rotations around predefined axes. Consequently, we can choose a sequence of three elementary rotation matrices to describe every possible rotation. Obviously, two successive rotations must use different elementary rotations, since rotating two times around the same axis can be trivially combined into only one rotation. However, it is possible to use the same axis twice (e.g., a rotation sequence using a


Figure 2.9: Rotation with a fixed order of axes can result in non-intuitive behaviour. The camera is located at the origin and looks along the z-axis. In the rotation sequence $\mathbf{R}_{\mathbf{z}}(\gamma) \mathbf{R}_{\mathbf{y}}(\beta)$, we intuitively understand $\beta$ as the horizontal pan. However, for $\gamma \approx 90^{\circ}$, a change of $\beta$ results in vertical motion.

Z-Y-Z axes order is also capable to describe all possible rotations).
Rotations are not commutative and hence, the order in which we perform the rotations is important. Moreover, without further restrictions on allowed rotation angles, the same rotation can often be specified with different sets of parameters even if the order of rotation axes is fixed.

## Euler angles vs. human intuition

The human understanding of rotations is strongly related to our physical environment. If there are several ways to describe the same rotation, we have a strong tendency to describe it in a way which corresponds best to the physically most probable action. For example, when we watch our image in the mirror, we perceive it with swapped left and right orientation. However, it could also be understood just as well with swapped up and down orientation. Only the fact that we turn around our vertical axis more frequently than we stand on our head makes the distinction.

Moreover, it seems that human intuition always considers rotation as an iterative action which is relative to the last state. Changing the angle of a rotation that is not the last one in the rotation sequence does not match
our expectations. As an example, consider the set-up of Figure 2.9 with the rotation sequence $\mathbf{R}_{\mathbf{z}}(\gamma) \mathbf{R}_{\mathbf{y}}(\beta)$, which is a rotation by $\beta$ around the vertical axis (camera pan), followed by a rotation by $\gamma$ around the optical axis. Usually, we intuitively associate a camera pan (rotation around $y$ axis) with a left-right motion, which is true if we consider the rotation independently. However, if the pan is followed by another rotation, e.g., $\mathbf{R}_{\mathbf{z}}(\gamma)$, the motion direction can change completely because of the second transform. If $\gamma \approx 90^{\circ}$, a change of the camera pan angle $\beta$ does not result in horizontal motion, but it induces a vertical motion in the camera image.

This conflict with our intuition occurs, because we tend to think that changes of rotation angles occur relative to the last position. If we make a small change of camera pan $\Delta \beta$, our intuition suggests that the overall camera transform would be $\mathbf{R}_{\mathbf{y}}(\Delta \beta) \mathbf{R}_{\mathbf{z}}(\gamma) \mathbf{R}_{\mathbf{y}}(\beta)$, instead of $\mathbf{R}_{\mathbf{z}}(\gamma) \mathbf{R}_{\mathbf{y}}(\beta+\Delta \beta)$. If we stick to the Euler-angles parameterization, we have to live with this discrepancy to our intuition. Fortunately, this problem only shows at large rotation angles. Since the freedom of camera motion is usually very limited (mostly horizontal pan, sometimes vertical pan, but usually no tilting), this problem can be reduced in practice. Using the pre-knowledge of typical camera motion, we select an appropriate parameterization that mostly behaves according to our expectations. Since rotations can be influenced by subsequent transformations, the rotation that has the largest dynamic range should be the last in the sequence. Using this order, we get

$$
\begin{equation*}
\mathbf{R}(\alpha, \beta, \gamma)=\mathbf{R}_{\mathbf{y}}(\beta) \mathbf{R}_{\mathbf{x}}(\alpha) \mathbf{R}_{\mathbf{z}}(\gamma) \tag{2.15}
\end{equation*}
$$

as a good choice for the rotation sequence.

## Gimbal lock

An especially annoying effect of the Euler-angle parameterization is the gimbal lock phenomenon. Gimbal lock is the situation that the Euler angles are chosen such that two of the rotation axes coincide. In our parameterization of Eq. (2.15), this is the case when $\alpha= \pm \pi / 2$. In this situation, the $y$ and the $z$ axes coincide and the rotation angles $\beta$ and $\gamma$ both induce similar motion (Fig. 2.10). Thus, the degrees of freedom is reduced from three to only two in the gimbal lock position. To prove this, we insert $\alpha=\pi / 2$ into


Figure 2.10: The object is mounted into a gimbal that corresponds to our Euler rotation sequence. In the position (b), the inner frame is positioned with $\alpha= \pm \pi / 2$, so that the $z$ and the $y$ axes coincide.

Eq. (2.15) and get

$$
\begin{array}{r}
\mathbf{R}_{\mathbf{y}}(\beta) \mathbf{R}_{\mathbf{x}}(\pi / 2) \mathbf{R}_{\mathbf{z}}(\gamma)=\left[\begin{array}{ccc}
c_{\beta} & 0 & s_{\beta} \\
0 & 1 & 0 \\
-s_{\beta} & 0 & c_{\beta}
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & -1 \\
0 & 1 & 0
\end{array}\right]\left[\begin{array}{ccc}
c_{\gamma} & -s_{\gamma} & 0 \\
s_{\gamma} & c_{\gamma} & 0 \\
0 & 0 & 1
\end{array}\right] \\
=\left[\begin{array}{ccc}
c_{\beta} c_{\gamma}+s_{\beta} s_{\gamma} & s_{\beta} c_{\gamma}-c_{\beta} s_{\gamma} & 0 \\
0 & 0 & -1 \\
c_{\beta} s_{\gamma}-s_{\beta} c_{\gamma} & s_{\beta} s_{\gamma}+c_{\beta} c_{\gamma} & 0
\end{array}\right]=\left[\begin{array}{ccc}
c_{(\beta-\gamma)} & s_{(\beta-\gamma)} & 0 \\
0 & 0 & -1 \\
-s_{(\beta-\gamma)} & c_{(\beta-\gamma)} & 0
\end{array}\right] . \tag{2.16}
\end{array}
$$

Hence, $\beta$ and $\gamma$ induce the same rotation (in opposite directions).

## Rotation matrices

Often, we do not require the parameterization into Euler angles but we can work with only the out-multiplied rotation matrix. In these cases, we can just use general $3 \times 3$ rotation matrices

$$
\mathbf{R}=\left[\begin{array}{lll}
r_{00} & r_{01} & r_{02}  \tag{2.17}\\
r_{10} & r_{11} & r_{12} \\
r_{20} & r_{21} & r_{22}
\end{array}\right]
$$

without knowing its factorization into angle parameters. However, since we know that $\mathbf{R}$ is a rotation matrix, we can make use of some properties


Figure 2.11: Naming conventions used on the right-handed camera coordinate system.
of rotation matrices. We have seen previously that for the inverse of a elementary rotation matrix, it holds that $\mathbf{R}_{\mathbf{i}}{ }^{-1}=\mathbf{R}_{\mathbf{i}}{ }^{\top}$. Now, let $\mathbf{R}$ be composed of a sequence of rotations $\mathbf{R}=\mathbf{R}_{1} \cdots \mathbf{R}_{n}$. It follows that

$$
\begin{equation*}
\mathbf{R}^{\top}=\left(\mathbf{R}_{1} \cdots \mathbf{R}_{n}\right)^{\top}=\mathbf{R}_{n}^{\top} \cdots \mathbf{R}_{1}^{\top}=\mathbf{R}_{n}^{-1} \cdots \mathbf{R}_{1}^{-1} \tag{2.18}
\end{equation*}
$$

and

$$
\begin{equation*}
\mathbf{R} \cdot \mathbf{R}^{\top}=\mathbf{R}_{1} \cdots \mathbf{R}_{n} \cdot \mathbf{R}_{n}^{-1} \cdots \mathbf{R}_{1}^{-1}=\mathbf{I} \tag{2.19}
\end{equation*}
$$

Consequently, the transpose of the composed rotation matrix must be equal to its inverse $\mathbf{R}^{\top}=\mathbf{R}^{-1}$. In other words, $\mathbf{R R}^{\top}=\mathbf{I}$, which shows that $\mathbf{R}$ must be an orthogonal matrix.

## Obtaining Euler angles from a rotation matrix

Sometimes, we have an arbitrary rotation matrix and we want to know the Euler angles for a factorization into a given sequence of elementary rotations. Assume that we fix the rotation sequence to that of Eq. (2.15), we get the out-multiplied matrix

$$
\mathbf{R}=\left[\begin{array}{ccc}
c_{\beta} c_{\gamma}+s_{\alpha} s_{\beta} s_{\gamma} & -c_{\beta} s_{\gamma}+s_{\alpha} s_{\beta} c_{\gamma} & c_{\alpha} s_{\beta}  \tag{2.20}\\
c_{\alpha} s_{\gamma} & c_{\alpha} c_{\gamma} & -s_{\alpha} \\
s_{\alpha} c_{\beta} s_{\gamma}-s_{\beta} c_{\gamma} & s_{\alpha} c_{\beta} c_{\gamma}+s_{\beta} s_{\gamma} & c_{\alpha} c_{\beta}
\end{array}\right] .
$$

Because the trigonometric functions are cyclic, we require additional restrictions to find a unique solution. A possible restriction is to assume that

Chapter 2. Projective Geometry
$-\pi / 2<\alpha<\pi / 2$. This is a sensible assumption for a normal camera, since it limits the up-down rotation to $\pm 90^{\circ}$. Now, $\alpha$ can be obtained directly from $r_{12}=-\sin \alpha$. Further, we see from the last column of $\mathbf{R}$ that

$$
\begin{equation*}
\frac{r_{02}}{r_{22}}=\frac{c_{\alpha} \cdot s_{\beta}}{c_{\alpha} \cdot c_{\beta}}, \text { and consequently, } \quad \tan \beta=\frac{s_{\beta}}{c_{\beta}}=\frac{r_{02}}{r_{22}} . \tag{2.21}
\end{equation*}
$$

The correct quadrant of $\beta$ can be determined from the signs of nominator and denominator. Thus, it is important that we keep the signs at their respective terms. Since we assumed that $|\alpha|<\pi / 2$, we know that $c_{\alpha}>0$. This makes it easier to obtain the correct quadrant for $\beta$ in Eq. (2.21). The angle $\gamma$ can be obtained similarly from $r_{10}$ and $r_{11}$.

## Quaternion rotation

Because of the gimbal lock problems of Euler rotation sequences, a different representation of rotations has become popular in the computer graphics community. This representation uses quaternions, which are a generalization of complex numbers to four elements. A quaternion is a number $q=a+b i+c j+d k$ together with the rules $i^{2}=j^{2}=k^{2}=-1, i j=-j i=k$, $j k=-k j=i$, and $k i=-i k=j$. Furthermore, the quaternion conjugate is defined as $\bar{q}=a-b i-c j-k j$.

Alternatively to using the imaginary units, a quaternion can also be written as a vector of its four components $\mathbf{q}=(a, b, c, d)$. It can be shown that a quaternion with $\|\mathbf{q}\|=1$ represents a rotation in 3-D space using the transformation $\mathbf{p}^{\prime}=\mathbf{q} \mathbf{p} \overline{\mathbf{q}}$, where $\mathbf{p}=(0, x, y, z)$ is the point coordinate.

It is especially easy to construct a quaternion that represents a rotation of $\theta$ around the axis vector $\mathbf{n}$. This rotation can be written as the quaternion $\mathbf{q}=\left(\cos (\theta / 2), \sin (\theta / 2) \cdot \mathbf{n}^{\top}\right)$. On the other hand, a quaternion rotation can also be written as the $3 \times 3$ rotation matrix

$$
\mathbf{R}=\left[\begin{array}{ccc}
1-2 q_{y}^{2}-2 q_{z}^{2} & 2 q_{x} q_{y}-2 q_{w} q_{z} & 2 q_{x} q_{z}+2 q_{w} q_{y}  \tag{2.22}\\
2 q_{x} q_{y}+2 q_{w} q_{z} & 1-2 q_{x}^{2}-2 q_{z}^{2} & 2 q_{y} q_{z}-2 q_{w} q_{x} \\
2 q_{x} q_{z}-2 q_{w} q_{y} & 2 q_{y} q_{z}+2 q_{w} q_{x} & 1-2 q_{x}^{2}-2 q_{y}^{2}
\end{array}\right] .
$$

The advantage of the quaternion representation compared to the Euler rotation sequences is that there is no singularity as with the gimbal lock position. Furthermore, they are faster to compute since no transcendental functions are involved. We will use the quaternion representation in Chapter 12 to describe camera rotation.

### 2.4.3 Perspective projection

In this section, we introduce the projection of points from 3-D space onto a 2-D image plane. This projection operation is based on the idealized


Figure 2.12: Model of an ideal pinhole camera.
model of a pinhole camera, which is a very good approximation to most real cameras. We derive the projection equations and formulate them, using the previously explained homogeneous coordinates framework. Finally, we introduce the affine camera model, which is an often used approximation that leads to the special case of affine motion in the image plane.

## Perspective pinhole camera

The setup for the perspective camera model is illustrated in Figure 2.12. Object points are projected along the ray from the camera center to the object point. The intersection of the ray with the image plane defines the point's position in the image. In a real camera, the image plane is actually behind the camera center, and the image is projected onto it up-side down, but for simplicity, we assume that the image plane is in front of the camera. This is equivalent, but it relieves us from considering many minus signs.

Let us denote the distance between the camera center and the image plane by $f$ (focal length) and let the object points have coordinates $(x, y, z)^{\top}$. Since the camera is located at the coordinate system origin, and since it is looking along the positive $z$-axis, the projection of the point onto the image plane is given by $\left(x^{\prime}, y^{\prime}\right)^{\top}=(f x / z, f y / z)^{\top}$. This projection can also be written using a matrix multiplication in homogeneous coordinates as

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.23}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{llll}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & 1 & 0
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) .
$$

We call this matrix the matrix of intrinsic camera parameters. In our ideal pinhole camera model, the matrix only consists of the focal length, which can be considered as an internal parameter of the camera. However,


Figure 2.13: From perspective projection the orthographic projection.
in the following sections, we extend the intrinsic matrix with additional parameters.

## Affine camera

When we use the perspective camera model together with inhomogeneous coordinates, we obtain non-linear equations for the transformation. In several situations, this non-linearity makes the computation more complex. A popular approach is to use a linear estimation to the perspective camera which assumes that the camera is placed infinitely far from the object. As is shown in the following, the simplified camera model results in affine motion in the image plane.

In Figure 2.13(a), the normal perspective camera setup is depicted. Now assume that the camera is shifted away from the image plane by an additional distance $d$. This shift is compensated by increasing the focal length to $f+d$ to keep the distance between object and image plane constant. We can formally write this construction by inserting a matrix for the camera movement before the projection is made:

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.24}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{cccc}
f+d & 0 & 0 & 0 \\
0 & f+d & 0 & 0 \\
0 & 0 & 1 & 0
\end{array}\right]\left[\begin{array}{cccc}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & d \\
0 & 0 & 0 & 1
\end{array}\right]\left(\begin{array}{l}
x \\
y \\
z \\
w
\end{array}\right) .
$$

Multiplying out the two matrices and increasing the camera distance $d$ up to infinity, we get

$$
\left[\begin{array}{cccc}
f+d & 0 & 0 & 0  \tag{2.25}\\
0 & f+d & 0 & 0 \\
0 & 0 & 1 & d
\end{array}\right] \sim\left[\begin{array}{cccc}
\frac{f}{d}+1 & 0 & 0 & 0 \\
0 & \frac{f}{d}+1 & 0 & 0 \\
0 & 0 & \frac{1}{d} & 1
\end{array}\right] \xrightarrow{d \rightarrow \infty}\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{array}\right],
$$

where the first similarity holds because of the scaling invariance of homogeneous coordinates. The resulting matrix is the camera projection matrix for affine cameras. Clearly, it is independent of a focal length and the scene depth $z$ (the column that multiplies $z$ only contains zeroes). Written in inhomogeneous coordinates, this is simply

$$
\begin{equation*}
x^{\prime}=x / w \quad y^{\prime}=y / w, \tag{2.26}
\end{equation*}
$$

which is just the conversion of homogeneous coordinates into inhomogeneous coordinates (see Eq. (2.1)). Consequently, the affine camera model describes a parallel projection of the 3-D coordinates onto the image plane (Fig. 2.13(b)). This type of projection is also called an orthographic projection.

Let us now examine the complete imaging process with an affine camera. We start with an arbitrary affine object motion in 3-D, followed by an orthographic projection:

$$
\left(\begin{array}{l}
x^{\prime}  \tag{2.27}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\underbrace{\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]}_{\text {orthographic projection }} \underbrace{\left[\begin{array}{cccc}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
0 & 0 & 0 & 1
\end{array}\right]}_{\text {affine motion in 3-D }}\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) .
$$

Furthermore, we assume that the observed object is planar, which means that, e.g., its $z$-coordinate is dependent on the others, and we can write $z=p_{x} x+p_{y} y+p_{w}$. Incorporating this constraint into Eq. (2.27), we get

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.28}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{llll}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{cccc}
a_{00} & a_{01} & a_{02} & a_{03} \\
a_{10} & a_{11} & a_{12} & a_{13} \\
a_{20} & a_{21} & a_{22} & a_{23} \\
0 & 0 & 0 & 1
\end{array}\right]\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & 1 & 0 \\
p_{x} & p_{y} & p_{w} \\
0 & 0 & 1
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
w
\end{array}\right) .
$$

Multiplying the three matrices, the total mapping onto the image plane computes as

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.29}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{ccc}
a_{00}+p_{x} a_{02} & a_{01}+p_{y} a_{02} & p_{w} a_{02}+a_{03} \\
a_{10}+p_{x} a_{12} & a_{11}+p_{y} a_{12} & p_{w} a_{12}+a_{13} \\
0 & 0 & 1
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
w
\end{array}\right) .
$$

However, since the last row is $(0,0,1)$, this defines an affine transformation. Consequently, planar object motion in 3-D is observed on the image plane as affine motion if an orthographic projection is applied.


Figure 2.14: Image formation in the case of an ideal pinhole camera.

### 2.5 Image acquisition

In this section, we discuss the image formation process in more detail. We start by looking at the pinhole camera model again and extend its description to cover more general cases. Furthermore, we include a brief discussion of lens-distortion artifacts that can become relevant in a practical application. Finally, we analyse which camera setups allow to use the projective motion model to describe global motion.

### 2.5.1 Intrinsic camera parameters

We start with a more detailed discussion of the ideal pinhole camera, which we will successively extend to include more general camera configurations. When we specify the camera position by saying, that the camera should be located at the origin of the coordinate system, this means more specifically that the pinhole of the camera is at the origin. The pinhole position, which is the center of the projection rays, is denoted as the optical center of the camera.

Let us assume that a local coordinate system is attached to the camera. We define this coordinate system to be right-handed, and the camera to be viewing in the direction of the positive $z$-axis (Figure 2.14). The $x$-axis increases to the right and the $y$-axis, consequently, to the bottom. The image plane is perpendicular to the $z$-axis, at a distance of the focal length $f$. The viewing direction, which is in our case simply the $z$-axis, is called the principal axis, and the position where the principal axis intersects the image plane is denoted as the principal point.

## Projection of $3-\mathrm{D}$ points onto the image plane

Consider the point $\mathbf{p}=(x, y, z)^{\top}$ in $\mathbb{E}^{3}$. To determine its position on the image plane using the pinhole camera model, we construct a line through the optical center and the point $\mathbf{p}$. The intersection of this line with the image plane defines its position $\mathbf{p}^{\prime}=\left(x^{\prime}, y^{\prime}\right)$ on the image plane. If the origin of the local coordinate system of the image plane is also at the principal point and the coordinate axes are parallel to that of the camera, the projection can be calculated by

$$
\begin{equation*}
x^{\prime}=\frac{x \cdot f}{z} \quad ; \quad y^{\prime}=\frac{y \cdot f}{z} . \tag{2.30}
\end{equation*}
$$

Using homogeneous coordinates, the projection can also be written as the matrix multiplication

$$
\mathbf{p}^{\prime}=\left(\begin{array}{c}
x^{\prime}  \tag{2.31}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{llll}
f & 0 & 0 & 0 \\
0 & f & 0 & 0 \\
0 & 0 & 1 & 0
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) .
$$

Note that in the derivation of the equation, we have silently assumed a number of idealized properties: we assumed that pixels in the image plane are square, and we assumed that there is no skew in the image sensor-array.

## Non-ideal image sensors

Usually, we can assume square pixels in the recording camera. However, if they are not square, the intrinsic camera-parameter matrix can be extended by a parameter $\eta$, denoting a vertical scaling factor. Moreover, the sampling grid may be skewed (see Fig. 2.15(a)). This can also be represented by an additional parameter $\tau$. If we include these parameters, we get the more general camera projection matrix

$$
\left[\begin{array}{cccc}
f & \tau & 0 & 0  \tag{2.32}\\
0 & \eta f & 0 & 0 \\
0 & 0 & 1 & 0
\end{array}\right] .
$$

However, in most cases, it can simply be assumed that $\eta=1$ and $\tau=0$. In fact, in Chapter 12, we use these assumptions as constraints to recover the focal length from estimated camera motion.


Figure 2.15: (a) In a non-ideal camera, the sampling-grid may have nonsquare pixels and the sampling-grid may be skewed. (b) Usually, the image coordinate system is assumed to have its origin at the top-left corner instead of the principal point.

## Changing the image coordinate system

The usual convention of storing images in memory is to place the origin of the image coordinate system at the top left of the image, with the $x$-axis extending to the right, and the $y$-axis pointing downwards. Previously, we have assumed that the origin coincides with the principal point. Now, we relieve this constraint and assume that the principal point is located at $\left(o_{x}, o_{y}\right)$ in image coordinates. Moreover, we might want to flip some of the coordinate system axes if their definition in the image is different. For the purpose of demonstration, we assume that we also want to flip the direction of the $y$-axis. By concatenating the projection matrix with the flip of the $y$-axis and the shift of the coordinate system, we get the generalized camera projection matrix

$$
\underbrace{\left[\begin{array}{ccc}
1 & 0 & o_{x}  \tag{2.33}\\
0 & 1 & o_{y} \\
0 & 0 & 1
\end{array}\right]}_{\text {translate }} \cdot \underbrace{\left[\begin{array}{ccc}
1 & 0 & 0 \\
0 & -1 & 0 \\
0 & 0 & 1
\end{array}\right]}_{\text {flip } y \text {-axis }} \cdot\left[\begin{array}{cccc}
f & \tau & 0 & 0 \\
0 & \eta f & 0 & 0 \\
0 & 0 & 1 & 0
\end{array}\right]=\left[\begin{array}{cccc}
f & \tau & o_{x} & 0 \\
0 & -\eta f & o_{y} & 0 \\
0 & 0 & 1 & 0
\end{array}\right]
$$

The right-hand side of this equation is the most general form of the intrinsic camera-parameters matrix. The last column is often omitted so that only a $3 \times 3$ matrix is considered. Note that even the general intrinsic matrix is upper triangular, which is sometimes exploited in camera calibration techniques where we try to estimate the intrinsic matrix.


Figure 2.16: Radial lens distortian. Opposed to the ideal pinhole model, real cameras are subject to lens distortion.

## Radial lens distortion

Real cameras consist of a number of lenses instead of a simple pinhole. These lenses are usually not ideal and show some non-linear geometric distortion, which can become significant for applications requiring a high accuracy.

There is no simple general model for lens distortion, but for most practical purposes, it can be approximated by a simple radial lens distortion model. To define a distortion model, let $\left(x^{\prime}, y^{\prime}\right)$ be the coordinates of a pixel including the lens distortion (the projected position on the sensor array), and let $(\hat{x}, \hat{y})$ be the coordinates of their ideal, undistorted coordinate (Figure 2.16). A popular model for radial lens distortion can then be described by

$$
\begin{align*}
& \hat{x}=o_{x}+\left(x^{\prime}-o_{x}\right) \cdot(1+D) \\
& \hat{y}=o_{y}+\left(y^{\prime}-o_{y}\right) \cdot(1+D) \tag{2.34}
\end{align*}
$$

with $D$ being the radial correction term

$$
\begin{equation*}
D=\left(\kappa_{1} r^{2}+\kappa_{2} r^{4}+\kappa_{3} r^{6}+\cdots\right), \tag{2.35}
\end{equation*}
$$

where

$$
\begin{equation*}
r=\sqrt{\left(x^{\prime}-o_{x}\right)^{2}+\left(y^{\prime}-o_{y}\right)^{2}} \tag{2.36}
\end{equation*}
$$

is the distorted point's distance from the principal point. In most cases, it is sufficient to consider only the two low-order coefficients $\kappa_{1}, \kappa_{2}$, and assume that the higher-order coefficients are zero.


Figure 2.17: General setting of a camera located at $\mathbf{c}$, observing a point $\mathbf{p}_{\mathbf{w}}$ in 3D space.

For many cameras, the effect of radial distortion can be neglected. However, for extreme wide-angle lenses, the effect can become significant and the radial distortion should be compensated. In this thesis, we assume that radial distortion is not present, or that it has been compensated previously.

### 2.5.2 Extrinsic camera parameters

While the intrinsic camera parameters describe properties of the camera like its focal length and the image geometry, the external camera parameters describe the camera placement and orientation in the 3-D world.

Assume that the camera is located at a position $\mathbf{t}$ and rotated according to a rotation matrix $\mathbf{R}$. To determine the image position of a 3-D point $\mathbf{p}$ in the observed image, we first have to bring the camera to the origin and rotate it so that its local camera coordinate system is aligned with the world coordinate system. Written as a sequence of transformations, we obtain

$$
\begin{align*}
\left(\begin{array}{c}
x^{\prime} \\
y^{\prime} \\
w^{\prime}
\end{array}\right) & =\left[\begin{array}{cccc}
f & 0 & o_{x} & 0 \\
0 & f & o_{y} & 0 \\
0 & 0 & 1 & 0
\end{array}\right] \underbrace{\left[\begin{array}{cc}
\mathbf{R} & \mathbf{0}_{3} \\
\mathbf{0}_{3}^{\top} & 1
\end{array}\right]}_{\text {rotation }} \underbrace{\left[\begin{array}{cc}
\mathbf{1}_{3 \times 3} & -\mathbf{t} \\
\mathbf{0}_{3}^{\top} & 1
\end{array}\right]}_{\text {translation }}\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) \\
& =\underbrace{\left[\begin{array}{cccc}
f & 0 & o_{x} & 0 \\
0 & f & o_{y} & 0 \\
0 & 0 & 1 & 0
\end{array}\right]}_{\text {intrinsic parameters }} \underbrace{\left[\begin{array}{cc}
\mathbf{R} & -\mathbf{R t} \\
\mathbf{0}_{3}^{\top} & 1
\end{array}\right]}_{\text {extrinsic parameters }}\left(\begin{array}{c}
x \\
y \\
z \\
w
\end{array}\right) . \tag{2.37}
\end{align*}
$$

Since the last column of the intrinsic parameters matrix is all zero, it is common practice to remove the last column from the intrinsic parameters
matrix and as a consequence thereof, also the last row of the extrinsic parameters matrix. Note that this is a slight misuse of notation, since homogeneous coordinates are mixed with an inhomogeneous notation. However, it saves us from writing matrices with constant zero rows or columns. The reduced matrices can then be defined as

$$
\mathbf{K}=\left[\begin{array}{ccc}
f & 0 & o_{x}  \tag{2.38}\\
0 & f & o_{y} \\
0 & 0 & 1
\end{array}\right] \quad ; \quad \mathbf{E}=[\mathbf{R} \mid-\mathbf{R t}]
$$

where we call $\mathbf{K}$ the $3 \times 3$ intrinsic camera parameters matrix and $\mathbf{E}$ the $3 \times 4$ extrinsic camera parameters matrix. Note that the matrix $\mathbf{E}$ transforms from $\mathbb{P}^{3}$ to $\mathbb{E}^{3}$ and $\mathbf{K}$ further from $\mathbb{E}^{3}$ to $\mathbb{P}^{2}$. The annotated arrows in the following equation show the type of the vector after each matrix multiplication:

## Inverse transformation

When a 3-D scene is projected onto the 2-D image plane, it is obvious that information about the depth is lost. In the perspective projection, every point along the projection ray is mapped onto the same image point. Therefore, it is impossible to say which point on the ray is viewed in the image. This becomes apparent if we try to invert the above transformation pipeline. Using the intrinsic parameters matrix in the inverse direction to map 2-D points $\left(x^{\prime}, y^{\prime}, 1\right) \in \mathbb{P}^{2}$ into 3 -D space $(x, y, z) \in \mathbb{E}^{3}$ again, results in

$$
\left(\begin{array}{l}
x  \tag{2.40}\\
y \\
z
\end{array}\right)=\mathbf{K}^{-1}\left(\begin{array}{c}
\lambda \cdot x^{\prime} \\
\lambda \cdot y^{\prime} \\
\lambda \cdot 1
\end{array}\right) .
$$

The factor $\lambda$ on the right-hand side accounts for the fact that the same point in homogeneous coordinates can also be specified by an arbitrarily scaled version of that vector. After applying the inverse $\mathbf{K}^{-1}$, each point on the image plane therefore defines a one-dimensional subspace in $\mathbb{E}^{3}$. Clearly, this subspace is the projection ray with the free parameter $\lambda \neq 0$.

To invert the extrinsic parameters matrix, we have to use the full $4 \times 4$ matrix, since the abbreviated $3 \times 4$ matrix is not square. Since the result of the multiplication with $\mathbf{K}^{-1}$ is in $\mathbb{E}^{3}$, we also have to augment that vector


Figure 2.18: Steps for transforming a 3-D point position to its coordinate in a camera image, including distortions.
with the homogenizing constant 1 . When we compute the inverse of the external parameters matrix, we get

$$
\left[\begin{array}{cc}
\mathbf{R} & -\mathbf{R t}  \tag{2.41}\\
\mathbf{0}_{3}^{\top} & 1
\end{array}\right]^{-1}=\left[\begin{array}{cc}
\mathbf{R}^{-1} & \mathbf{t} \\
\mathbf{0}_{3}^{\top} & 1
\end{array}\right] .
$$

It can be seen that the last matrix row of the inverse is still $(0,0,0,1)$. This is clear since the external parameters matrix is affine, and hence, its inverse must also be affine.

## The complete imaging process

If we put all the described image transforms together, we obtain a complete formulation for the entire image formation process. This image formation is divided into several steps and can be illustrated as a pipeline process, as shown in Figure 2.18. When we follow the path of a point from the 3-D real world to the image sensor, we start by transforming the point's position in the world coordinate system into the camera coordinate system using the extrinsic parameters matrix $\mathbf{E}$. The succeeding intrinsic parameters matrix further maps the point to the position on the image sensor-array, but still neglecting lens distortion. Finally, the distortions that are introduced by non-ideal lens optics are modeled with a radial lens distortion.

### 2.5.3 Camera motion in a static environment

Suppose that two pictures of a static 3-D scene are taken from different positions or at different angles. We would like to explore in which cases it is possible to describe the transform between the two images as a projective transform $\mathbf{H}$.


Figure 2.19: Top view of a 3-D scene with three object $A, B, C$. Seen from two different angles, the objects appear on the camera images in a different order.

Since we have seen in Section 2.3.1 that a projective transformation is equivalent to a plane-to-plane mapping, it is clear that the transform can model arbitrary camera motion, as long as the observed object is planar. However, the restriction that the complete observed environment is planar is too strict in practice, and we would like to know if there are more situations which can still be described using the projective motion model.

Obviously, it is also clear that we cannot describe the general case with a simple perspective transformation. Take for example Figure 2.19 which shows a scene with four objects, viewed from two different positions. It can be observed that the objects are projected onto the camera images in a different ordering, depending on the camera position.

## Parallax effect

From everyday experience, we know the effect that if we move a camera at a constant speed, near objects seem to move faster than objects at a larger distance. This visual effect is commonly known as the Parallax effect. The Parallax effect has consequences for 3-D vision, since an arrangement of objects in space can look very dissimilar from different camera locations. In general, it is not possible to describe the transform between these camera images by a projective transform.

To illustrate this, assume that we observe two points with inhomogeneous coordinates $\mathbf{p}_{\mathbf{0}}$ and $\mathbf{p}_{\mathbf{1}}$ at different distances from the camera, but which are projected onto the same point in the image plane (Fig. 2.20(a)). For simplicity, we assume w.l.o.g. that the camera is first located at the origin of the world coordinate system and that it is looking along the positive $z$-axis. Since both points are projected onto the same image position,
it holds that $\mathbf{K} \mathbf{p}_{\mathbf{0}} \sim \mathbf{K}_{\mathbf{1}}$, and consequently $\mathbf{p}_{\mathbf{0}} \sim \mathbf{p}_{\mathbf{1}}$. This means that there exists a $\lambda$ such that $\mathbf{p}_{\mathbf{0}}=\lambda \mathbf{p}_{\mathbf{1}}$.

Now, we translate the camera by a distance $\mathbf{d}$ and rotate it according to a rotation matrix $\mathbf{R}$. The question is whether the two points $\mathbf{p}_{\mathbf{0}}, \mathbf{p}_{\mathbf{1}}$ are also mapped onto the same image point in the new camera view. In other words, we want to know, if

$$
\underbrace{\mathbf{K}\left[\begin{array}{cc}
\mathbf{R} & -\mathbf{R d}  \tag{2.42}\\
\mathbf{0} & 1
\end{array}\right]\binom{\mathbf{p}_{\mathbf{0}}}{1}}_{\text {new image position of } \mathbf{p}_{\mathbf{0}}} \sim \underbrace{\mathbf{K}\left[\begin{array}{cc}
\mathbf{R} & -\mathbf{R d} \\
\mathbf{0} & 1
\end{array}\right]\binom{\lambda \mathbf{p}_{\mathbf{0}}}{1}}_{\text {new image position of } \mathbf{p}_{\mathbf{1}}}
$$

holds. After multiplying with $\mathbf{K}^{\mathbf{1}}$ from the left, we get

$$
\begin{equation*}
\mathbf{R p}_{\mathbf{0}}-\mathbf{R d} \sim^{?} \lambda \mathbf{R}_{\mathbf{0}}-\mathbf{R d} \tag{2.43}
\end{equation*}
$$

It is easy to see that this is only true if $\mathbf{d}=\mathbf{0}$, i.e., there is no translatorial camera motion, or $\lambda=1$. The former case holds if camera motion is restricted to rotation around the optical center (Fig. 2.20(b)). Since this case is of high practical importance, it will be described in more detail in the next subsection. The second case means that the two points $\mathbf{p}_{\mathbf{0}}, \mathbf{p}_{\mathbf{1}}$ are actually the same point. However, this means that on each projection ray, there may only be one object point. Otherwise, the two points could coincide in one image, but have a different position in another image. This second case applies if the object is planar.

## Rotating camera

In the special case of a purely rotational camera, let us first assume that the optical center of the camera is located at the coordinate system origin, and the rotation is performed around the coordinate system origin. In this case, the camera translation is $\mathbf{t}=\mathbf{0}$, and the complete viewing transformation can be written as

$$
\left(\begin{array}{l}
x^{\prime}  \tag{2.44}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\mathbf{K} \mathbf{R}\left(\begin{array}{l}
x \\
y \\
z
\end{array}\right) .
$$

The rotation matrix $\mathbf{R}$ assumes rotation of the world around a fixed camera, which is equivalent to rotating the camera in a fixed world, if the matrix is inversed (negative angles in opposite order).

One property of rotational camera motion is of special importance and should therefore be emphasized here. For the transformation between images taken by a rotating camera, the object depth is of no importance, since no parallax effect occurs. This fact can be justified easily with Eq. (2.44).

(a) Observing two points from different positions.

(b) Rotation-only camera.

Figure 2.20: What kind of camera motion is allowed in arbitrary 3-D scenes? (a) Object points at different depth and moving camera. Points may fall onto the same position in one image and onto different positions in another image. Hence, there cannot be a bijective transform between both. (b) Rotational camera motion. Two points on the same projection ray stay on one ray independent of a camera rotation. Hence, a transform between images is possible even with arbitrary object depths.

Since the camera is assumed to be at the origin, all pixels on the projection ray through a point can be obtained by scaling the point coordinate $(x, y, z)^{\top}$ on the right-hand side. Moreover, the transform is linear which means that the same scaling will be effective on the left-hand side. But since homogeneous coordinates are scaling invariant, this is actually the same point in the image. This holds for any choice of $\mathbf{R}$ such that points projected onto the same image position will always coincide, independent of any camera rotation.

### 2.5.4 Inter-image transformation

Up to now, we only considered transformations that map points from the 3 -D world onto the 2-D image plane. However, in many cases, we want to describe the motion between successive images.

Chapter 2. Projective Geometry

## Inter-image transforms with a rotating camera

Let us discuss again the situation of a rotational camera. In general, the scene around the camera has varying depths, but we have seen in the last subsection that the depth of objects does not play any role in the image formation. Hence, we can assume that the environment is actually located in the camera-image plane.

Now consider that we take two images $\Pi_{0}$ and $\Pi_{1}$ at different rotation angles $\mathbf{R}_{0}$ and $\mathbf{R}_{1}$ (as depicted in Figure $2.20(\mathrm{~b})$ ). According to Equation (2.44), we can compute the $3-\mathrm{D}$ ray on which an image point from $\Pi_{0}$ lies, and we can also compute the intersection of this ray with $\Pi_{1}$ by inverting the equation. Combining these two parts gives us the transformation between image $\Pi_{0}$ and $\Pi_{1}$ as

$$
\begin{equation*}
\mathbf{p}_{\mathbf{1}}=\underbrace{\mathbf{K} \mathbf{R}_{\mathbf{1}} \mathbf{R}_{\mathbf{0}}^{-1} \mathbf{K}^{-1}}_{\mathbf{H}} \mathbf{p}_{\mathbf{0}} \tag{2.45}
\end{equation*}
$$

Clearly, the four transformation matrices can be combined into a single $3 \times 3$ inter-image transformation matrix $\mathbf{H}$.

## Motion field for panning camera motion

A common misunderstanding about panning camera motion is that it is often assumed that the motion field resulting from a rotating camera equals a pure translation. However, as can be seen in Figure 2.21, this is not true. The figure depicts the motion field that is observed when a camera performs a rotation around the vertical axis. The motion field depends on the focal length of the camera. For larger focal lengths, the motion trajectories get straighter, and for the limit case of infinite focal length (affine camera), the motion field becomes translatorial. For practical cases, $f$ is in the order of the image width and, consequently, the motion cannot be approximated with a simple translation.

## Inter-image transforms with a planar object

If the observed object is planar, it is also possible to find an inter-image transform. In this case, this is obvious, since the mapping from the object plane onto each of the image planes $\Pi_{0}, \Pi_{1}$ is an invertible plane-to-plane mapping (see Figure 2.22). Hence, we get the inter-frame transform by projecting the $\Pi_{0}$ plane onto the background plane, and then projecting from the background plane onto plane $\Pi_{1}$. Following the derivation in Section 2.5.2, let the transform from the background plane to image plane $i$ be denoted as $\mathbf{M}_{i}=\mathbf{K}_{\mathbf{i}} \mathbf{E}_{\mathbf{i}}$. Since this product is a $3 \times 4$ matrix, it cannot

(a) Short focal length $(f=400)$.

(b) Long focal length $(f=700)$.

Figure 2.21: Motion field of a pure horizontal camera pan for an image width of 800 .


Figure 2.22: A planar background is observed by a moving camera.
be inverted. However, we are free to choose the world coordinate system and can thus define that the background plane coincides with the $z=0$ plane. In this case, the projection from the background plane to image coordinates becomes

$$
\left(\begin{array}{c}
x^{\prime}  \tag{2.46}\\
y^{\prime} \\
w^{\prime}
\end{array}\right)=\left[\begin{array}{llll}
m_{00} & m_{01} & m_{02} & m_{03} \\
m_{10} & m_{11} & m_{12} & m_{13} \\
m_{20} & m_{21} & m_{22} & m_{23}
\end{array}\right]\left(\begin{array}{c}
x \\
y \\
z=0 \\
w
\end{array}\right)
$$

and the third column of the matrix $\mathbf{M}=\left\{m_{i k}\right\}$ can be omitted. Let us denote the matrix $\mathbf{M}$ without the third column as $\overline{\mathbf{M}}$. Since this matrix $\check{\mathbf{M}}$ is now a $3 \times 3$ matrix, it can be inverted and we can combine the two plane-to-plane mappings to obtain the inter-frame transform as $\mathbf{H}=\check{\mathbf{M}}_{\mathbf{1}} \overline{\mathbf{M}}_{\mathbf{0}}^{-1}$.

Outlook: Camera calibration from video sequences.
In this section, we have shown that the projective transform between two images can be modeled as a sequence of elementary physically meaningful operations. Usually, a camera-motion estimator only determines the final set of parameters that are not directly related to the elementary operations. In Chapter 12, we develop an algorithm to factorize the projective transformation parameters back into the elementary operations. This makes it possible to describe camera motion in physically meaningful terms, like rotation angles or the current focal length of the camera zoom lenses.

### 2.6 Summary and notational conventions

This chapter has introduced homogeneous coordinates to describe point positions. In two-dimensional space, a point is specified with a column vector $\mathbf{p}=(x, y, w)^{\top}$, where the Euclidean coordinates can be recovered as $(x / w, y / w)^{\top}$. To facilitate the formulation, we also use homogeneous coordinates as arguments to two-dimensional functions in the following chapters, assuming implicit conversion to Euclidean coordinates. This means that we use the simplified notation $I(\mathbf{p})$ instead of $I(x / w, y / w)$.

We have also seen that the projection of rigid 3-D motion of planar patches onto 2-D images can be described by a simple matrix multiplication. Considering the plane-to-plane mapping which transforms $\mathbf{p}$ onto $\mathbf{p}^{\prime}$, it can be formulated as multiplication with a $3 \times 3$ matrix $\mathbf{H}$ as $\mathbf{p}^{\prime}=\mathbf{H p}$.

An important case is the transformation of Euclidean 3-D coordinates onto a planar image as it is seen by a pinhole camera at the origin of the coordinate system. This transformation is usually decomposed into two parts, namely the intrinsic camera parameters matrix $\mathbf{K}$, comprising internal camera parameters like the focal length and the principal point $o_{x}, o_{y}$, and the extrinsic camera parameters matrix $\mathbf{E}$, describing camera rotation and translation (see Eq. (2.38)). For the special case of rotational camera motion, the intrinsic and extrinsic matrices can be combined to a transformation matrix $\mathbf{H}$ denoting the motion between a pair of images.

The most important equations are summarized in Table 2.3 with references to the sections in which they were introduced.

| Equation | Description | Section |
| :---: | :--- | :---: |
| $\mathbf{l}=\mathbf{p}_{\mathbf{1}} \times \mathbf{p}_{\mathbf{2}}$ | line through two points | 2.2 .2 |
| $\mathbf{p}=\mathbf{l}_{\mathbf{1}} \times \mathbf{l}_{\mathbf{2}}$ | intersection point of two lines | 2.2 .2 |
| $\mathbf{p}^{\prime}=\mathbf{H} \mathbf{p}$ | projective transformation of points | 2.3 .1 |
| $\mathbf{l}^{\prime}=\mathbf{H}^{-\top} \mathbf{l}$ | projective transformation of lines | 2.3 .1 |
| $\mathbf{K}=\left[\begin{array}{ccc}f & \tau & o_{x} \\ 0 & \eta f & o_{y} \\ 0 & 0 & 1\end{array}\right]$ | intrinsic camera parameters | 2.5 .1 |
| $\mathbf{E}=[\mathbf{R} \mid-\mathbf{R t}]$ | extrinsic camera parameters | 2.5 .2 |
| $\mathbf{H}=\mathbf{K}_{\mathbf{i}} \mathbf{R K} \mathbf{K}_{\mathbf{j}}^{-\mathbf{1}}$ | rotational camera motion | 2.5 .4 |

Table 2.3: Summary of the most important equations.

# Feature-based Motion I: Point-Correspondences 

Algorithms for motion estimation can be coarsely divided into feature-based techniques and dense estimation techniques. The feature-based techniques are known to perform better with large motions while dense estimation techniques provide a higher accuracy. To combine the advantages, we integrated both approaches into one motion-estimation system. We describe the feature-based motion estimator in this and the following chapter while dense estimation is covered in Chapter 5. A feature-based motion estimator comprises three steps: detection of feature-points, establishing featurecorrespondences, and estimation of camera motion parameters. In this chapter, we give an introduction to feature-based motion estimation and we cover the first two steps. We provide a survey of feature-point detectors and evaluate their accuracy. After that, we present a fast algorithm for determining correspondences between two sets of feature-points. The remaining parameter estimation step will be discussed in the next chapter.

### 3.1 Introduction

Video sequences generally comprise two types of motion: the motion of objects visible in the scene and the global motion that is caused by the moving camera. Object motion is usually also difficult to describe, since in general, objects can show articulated or deformable motion. In general, the motion of objects can be rather complex and very difficult to model. Examples are water waves, explosions, traffic lights, or other sudden changes. On the other hand, camera motion is restricted to only a few degrees of freedom like the camera rotation angles or its focal length. In the previous chapter, camera motion was described using a geometric model with a small set of parameters. When analyzing a video sequence, the inverse problem occurs: find the parameters which describe the apparent motion in the video sequence.

Techniques for estimating the motion parameters generally follow one of two fundamental approaches: direct estimation algorithms and featurebased algorithms.

- In the direct estimation algorithms, the motion parameters $\mathbf{H}$ are computed by minimizing the motion-compensated image difference between the two images $I_{t}$ and $I_{t+1}$, thus computing

$$
\begin{equation*}
\min _{\mathbf{H}} \sum_{\mathbf{p}}\left|I_{t}(\mathbf{p})-I_{t+1}(\mathbf{H p})\right|^{2} . \tag{3.1}
\end{equation*}
$$

If the motion model is simple, like the translatorial motion model of MPEG-2, the motion parameters can be found by an exhaustive search through the parameter space. However, for the projective motion model, there are eight free parameters, which raises the need for fast optimization algorithms. Usually, gradient-descent algorithms are applied to solve Equation (3.1). When initialized with a good set of parameters, the direct estimation methods can achieve high accuracy. On the other hand, convergence of the gradient-descent algorithms requires a good initialization, especially if the motion between images is large.

- The second approach for motion estimation are feature-based techniques. They determine a small set of feature points in each of the input frames and establish correspondences between matching points. The feature points are selected such that the motion of the point can be computed with high reliability and accuracy. After the pointcorrespondences have been established, the motion parameters are determined by fitting the motion model to the point-correspondence data.

(a) Aperture problem along a line.

(b) No aperture problem at corners.

Figure 3.1: (a) A small area with a linear structure cannot be located precisely in a larger image. (b) The aperture problem does not occur if a corner or any other texture that varies in two dimensions is visible in the window.

Since the feature-based methods primarily use the position of featurepoints instead of a direct comparison of the image data, the feature-based methods are more robust to changes of illumination or noise than the direct methods. Furthermore, they allow large motions between images and they are also faster to compute. On the other hand, their estimation accuracy is generally below that of dense estimation algorithms. To combine the respective advantages of both approaches, it is possible to use the result of the feature-based approach as an initialization of the successive dense estimation step. Since the initialization is already close to the correct motion, the gradient-descent algorithm will converge to the correct minimum. Without this close initialization, the algorithm may not find this minimum if the motion is too fast.

Our segmentation system also follows this approach of combining a feature-based initialization of the motion model with a refinement of accuracy in a succeeding direct estimation step. We discuss feature-based motion estimation in this chapter and the direct estimation algorithms in the successive chapter.

### 3.1.1 Basics of feature-based motion estimation

Estimating image motion is an ill-posed problem in many situations. For small image patches or even single pixels, there is usually not enough information available to determine the motion reliably. Consider for example a small image patch with a straight region border (Fig. 3.1(a)). If we take this small piece and try to find the matching position in a larger picture, we see that this is impossible, since the pattern can be found all along the


Figure 3.2: (a) Example of detected interest-points and their correspondence in the previous frame. (b) Example of a repetitive image texture. Correspondences cannot be established easily, since many similar interest-points exist.
region border. The problem can already be identified by only examining the search pattern. In our case, the pattern is only structured in one direction, namely perpendicular to the line direction. Along the line, the texture is uniform, making it impossible of finding a best-matching position. The problem becomes even worse if the pattern only shows uniform color, providing no directed structure in the texture at all. This undeterminism is known as the aperture problem. It is only possible to determine the position of the pattern reliably if the pattern shows variations in two directions. This is the case, for example, at corners (see Fig. 3.1(b)). Here, the position of the pattern can be determined precisely in both dimensions.

We have seen that for the purpose of motion estimation, little information can be obtained from areas of uniform or texture that is structured only in one-dimension. Hence, the idea of feature-based methods is to concentrate only on a small set of interest-points, for which the motion can be determined reliably. An example of detected interest-points and the respective motion vector to the corresponding point in the previous frame is shown in Figure 3.1.1.

The decision whether a position in the image is a good feature-point is generally made as a local decision based only on the image content in the neighborhood of the interest-point. However, on a more global view, it can still be difficult to determine the motion for an interest-point, even when it shows a clear corner. If the image content shows a repetitive structure (like seen in Figure 3.1.1), many interest-points with comparable a neighborhood exist and it is not clear which points corresponds to each other.

Resolving this problem is the task of a second algorithm, which com-
putes the actual correspondences between the two sets of interest-points. A variety of algorithms have been developed for this problem and they can be distinguished as algorithms that only consider the positions of the interest-points, and algorithms that also take the image content around each interest-point into account. The second approach has the advantage that the image content can guide the matching algorithm. On the other hand, this approach does not work well if the transformation between both images is so large that the image content cannot be easily compared. Similarity metrics that are invariant to e.g. affine transformations have been proposed in the literature, but since the motion between frames in a video sequence is usually small, we will not consider them further.

### 3.1.2 From feature-points to motion parameters

Once we obtained a set of corresponding points between two images, we can use this information to determine the motion parameters. Since each point-correspondence gives us two equations of constraints - one for the horizontal component and one for the vertical - we need four pointcorrespondences to solve for the eight parameters of the perspective motion model (see Figure 3.3). Inserting any four correspondences $\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}$ with $\mathbf{p}_{i}=\left(x_{i}, y_{i}, 1\right), \hat{\mathbf{p}}_{i}=\left(\hat{x}_{i}, \hat{y}_{i}, 1\right)$ into the inhomogeneous formulation from Eq. (2.10) and multiplying with the denominator results in the linear equation system

$$
\left[\begin{array}{cccccccc}
x_{1} & y_{1} & 1 & 0 & 0 & 0 & -x_{1} \hat{x}_{1} & -y_{1} \hat{x}_{1}  \tag{3.2}\\
0 & 0 & 0 & x_{1} & y_{1} & 1 & -x_{1} \hat{y}_{1} & -y_{1} \hat{y}_{1} \\
x_{2} & y_{2} & 1 & 0 & 0 & 0 & -x_{2} \hat{x}_{2} & -y_{2} \hat{x}_{2} \\
0 & 0 & 0 & x_{2} & y_{2} & 1 & -x_{2} \hat{y}_{2} & -y_{2} \hat{y}_{2} \\
x_{3} & y_{3} & 1 & 0 & 0 & 0 & -x_{3} \hat{x}_{3} & -y_{3} \hat{x}_{3} \\
0 & 0 & 0 & x_{3} & y_{3} & 1 & -x_{3} \hat{y}_{3} & -y_{3} \hat{y}_{3} \\
x_{4} & y_{4} & 1 & 0 & 0 & 0 & -x_{4} \hat{x}_{4} & -y_{4} \hat{x}_{4} \\
0 & 0 & 0 & x_{4} & y_{4} & 1 & -x_{4} \hat{y}_{4} & -y_{4} \hat{y}_{4}
\end{array}\right]\left(\begin{array}{l}
\hat{x}_{1} \\
h_{01} \\
h_{02} \\
\hat{y}_{1} \\
\hat{x}_{2} \\
\hat{y}_{2} \\
h_{10} \\
\hat{x}_{3} \\
\hat{y}_{31} \\
h_{12} \\
h_{20} \\
h_{21} \\
\hat{y}_{4}
\end{array}\right),
$$

which can be easily solved for the model parameters $h_{i k}{ }^{1}$. While four correspondences are enough to solve for the motion parameters, we usually have many more correspondences available. However, these are contaminated with outliers and inaccuracies in the feature positions. Apart from errors in the computation of the point-correspondences, we also consider correspondences that are part of foreground object motion as outliers. To separate this mixed data into clean sets of compatible motion is the main topic of Chapter 4.

[^3]

Figure 3.3: Four point-correspondences between two images define the projective transformation.

To summarize the typical feature-based motion estimation approach, we can conclude that a feature-based motion estimator comprises three main processing steps:

1. detection of interest-points,
2. establishing correspondences between two sets of interest-points, and
3. estimating the motion-model parameters for the dominant motion.

Algorithms for the first two steps are discussed and evaluated in the remainder of this chapter, while the third step is covered in the following chapter.

### 3.2 Interest-point detectors

In the past, a large number of interest-point detectors have been proposed and this topic is still an active area of research. One of the first interestpoint detectors was the Moravec detector [129]. It is a simple ad-hoc algorithm to detect image locations that show variations in many different directions. Shi and Tomasi [167] describe an interest-point detector that is derived directly by identifying the sort of texture for which the motion estimation process is well defined.

Many authors propose to use interest-point detectors that are in fact corner-point detectors ${ }^{2}$. However, it is interesting to note that for our application, it is not important that the interest-points are placed at some

[^4]specific position like the true corner position. As long as the detector consistently places the interest-points to the same corresponding positions in each image, its output is valuable.

A commonly used corner detector is the Harris (also known as Plessey) corner detector [84]. Interestingly, the Harris detector appears to be very similar to the Shi-Tomasi algorithm using only a different detection criterion on the same extracted data. A more recent corner detector that claims to have especially good corner localization capabilities is the SUSAN corner detector [170]. We will describe and evaluate the performance of the four mentioned corner detectors in the following sections.

Recent work concentrates on designing interest-point detectors that are invariant to some classes of image transformations. Since image transformations are usually small between successive frames of a video sequence, ordinary interest-point detectors work well for our application and we will not consider these invariant detectors here. For further information see, e.g., $[126,110]$.

## Criteria for good detectors

Interest-points should be placed at positions where the image content around the feature allows to identify the corresponding position in a second image with high reliability. Placing a feature-point onto a line is not sufficient, since a line is a one-dimensional feature that only allows to fix the position perpendicular to the line. The position of the interest-point along the line, on the other hand, remains uncertain. Consequently, good features must show a change of texture in two directions so that they can be localized reliably. In practice, corners, T-junctions, or complex textures provide a good localization (see Fig. 3.4(a)-(c)). However, interest-points are only selected locally and an interest-point that seems easy to track at first sight can actually be a bad choice if there are ambiguities because of, e.g., a repetitive pattern in the image (Fig. 3.1.1). Another problem case can occur if the video sequence shows several objects moving in different directions. If two of these objects overlap, artificial T-junctions can be visible along the object boundary. If one object moves, this junctions may seem to move in a different direction than the objects (see Fig. 3.4(d)). Both problems can only be solved globally and thus, they are not considered in the feature-point detector.

For the successive processing steps, where we want to establish correspondences between points in pairs of images, two properties of the interestpoint detector are especially important.

- First, for images which show basically the same content in a slightly


Figure 3.4: (a)-(c) Suitable structures for placing of interest-points. (d) An artificial T-junction is visible at the boundary between two objects. If one objects moves, the T-junction moves virtually in a different direction.
different view, the same features should be detected in both images. Features that are only found in one of the images cannot be part of a point-correspondence and would be useless for the succeeding steps. Even worse, these features will contaminate the data with noise and may lead to wrong correspondences which will then again make the robust estimation harder.

- A second criterion for a good interest-point detector is that the position of the interest-points should be invariant to the image transform that is present between the two images. Consider for example that the interest-point placement is biased to one direction, e.g., all points have some offset to the left. If we have a rotational transform between the two considered images, the interest-points would not lie at corresponding image positions anymore. This imposes problems in later steps where we want to estimate the image transforms based on the assumption that corresponding feature-points in a pair of images move according to the image transform. Clearly, a larger error in the localization of feature-points also increases the error in the estimation of the transform.


### 3.2.1 Moravec interest-point detector

The Moravec interest-point operator is based on the observation that the image content around a good feature-point should show variations in every possible direction. Following this principle, the Moravec operator considers a window $\mathcal{W}(x, y)$ of size $w \times w$ around a pixel $(x, y)$ and calculates the sum of absolute differences between pairs of pixels that are either horizontal,
vertical, or diagonal neighbours. More specifically, the sums for the four directions are calculated by

$$
\begin{align*}
& S_{\text {horiz }}(x, y)=\sum_{\left(x^{\prime}, y^{\prime}\right) \in \mathcal{W}(x, y)}\left|I\left(x^{\prime}, y^{\prime}\right)-I\left(x^{\prime}+1, y^{\prime}\right)\right|  \tag{3.3}\\
& S_{v e r t i}(x, y)=\sum_{\left(x^{\prime}, y^{\prime}\right) \in \mathcal{W}(x, y)}\left|I\left(x^{\prime}, y^{\prime}\right)-I\left(x^{\prime}, y^{\prime}+1\right)\right|  \tag{3.4}\\
& S_{\text {diag } 1}(x, y)=\sum_{\left(x^{\prime}, y^{\prime}\right) \in \mathcal{W}(x, y)}\left|I\left(x^{\prime}, y^{\prime}\right)-I\left(x^{\prime}+1, y^{\prime}+1\right)\right|  \tag{3.5}\\
& S_{\text {diag } 2}(x, y)=\sum_{\left(x^{\prime}, y^{\prime}\right) \in \mathcal{W}(x, y)}\left|I\left(x^{\prime}, y^{\prime}\right)-I\left(x^{\prime}-1, y^{\prime}+1\right)\right| \tag{3.6}
\end{align*}
$$

Since the differences along all directions must be large, the operator determines the minimum

$$
\begin{equation*}
S(x, y)=\min \left\{S_{h o r i z}(x, y), S_{v e r t i}(x, y), S_{\text {diag } 1}(x, y), S_{\text {diag } 2}(x, y)\right\}, \tag{3.7}
\end{equation*}
$$

it applies a local non-maximum suppression to these values, and it selects those positions as interest-points for which $S(x, y)$ exceeds a threshold $\tau_{m o}$.

A disadvantage of the Moravec operator is that it is an anisotropic operator that is sensitive to image rotation. The reason for this is that only four directions are considered and that the diagonal pixel distances are a factor of $\sqrt{2}$ larger than the horizontal and vertical distances.

### 3.2.2 Shi-Tomasi detector

Instead of using an intuitive ad-hoc definition of an interest detector, Shi and Tomasi propose in [167] to define the interest-point detector by looking at the motion-estimation problem itself and finding conditions when this problem can be solved reliably. The interest-points they search for are not defined by intuitive terms like corners, but only by their ability to track the features reliably. They assume a translatorial motion $\mathbf{d}$ between two images $I_{t}$ and $I_{t+1}$. If we consider again a small window $\mathcal{W}$ around the feature, the matching error $E$ can be written as

$$
\begin{equation*}
E=\sum_{\mathbf{p} \in \mathcal{W}}\left(I_{t}(\mathbf{p}+\mathbf{d})-I_{t+1}(\mathbf{p})\right)^{2} . \tag{3.8}
\end{equation*}
$$

After approximating $I_{t}$ by a linear Taylor expansion $I_{t}(\mathbf{p}+\mathbf{d}) \approx I_{t}(\mathbf{p})+$ $\nabla I_{t}^{\top} \mathbf{d}$, we can write the matching error as

$$
\begin{equation*}
E=\sum_{\mathbf{p} \in \mathcal{W}}\left(I_{t}(\mathbf{p})-I_{t+1}(\mathbf{p})+\nabla I_{t}^{\top} \mathbf{d}\right)^{2} \tag{3.9}
\end{equation*}
$$

To determine the motion $\mathbf{d}$, we find the minimum matching error by setting the derivatives to zero:

$$
\begin{equation*}
\frac{\partial E}{\partial \mathbf{d}}=2 \sum_{\mathbf{p} \in \mathcal{W}}\left(I_{t}(\mathbf{p})-I_{t+1}(\mathbf{p})+\nabla I_{t}^{\top} \mathbf{d}\right) \nabla I_{t}=0 . \tag{3.10}
\end{equation*}
$$

It follows that

$$
\begin{equation*}
\sum_{\mathbf{p} \in \mathcal{W}}\left(\nabla I_{t} \nabla I_{t}^{\top}\right) \cdot \mathbf{d}=\sum_{\mathbf{p} \in W} \nabla I_{t} \cdot\left(I_{t}(\mathbf{p})-I_{t+1}(\mathbf{p})\right), \tag{3.11}
\end{equation*}
$$

which is actually a linear equation system $\mathbf{G d}=\mathbf{e}$, where

$$
\begin{gather*}
\mathbf{G}=\left[\begin{array}{cc}
\sum_{i}\left(\partial I_{t}(\mathbf{p}) / \partial x\right)^{2} & \sum\left(\partial I_{t}(\mathbf{p}) / \partial x\right)\left(\partial I_{t}(\mathbf{p}) / \partial y\right) \\
\sum\left(\partial I_{t}(\mathbf{p}) / \partial x\right)\left(\partial I_{t}(\mathbf{p}) / \partial y\right) & \sum\left(\partial I_{t}(\mathbf{p}) / \partial y\right)^{2}
\end{array}\right],  \tag{3.12}\\
\mathbf{e}=\left(\sum _ { \sum } ^ { \sum ( \partial I _ { t } ( \mathbf { p } ) / \partial x ) \cdot ( I _ { t } ( \mathbf { p } ) - I _ { t + 1 } ( \mathbf { p } ) ) } \left(\sum_{t}\left(\partial I_{t}(\mathbf{p}) / \partial y\right) \cdot\left(I_{t}(\mathbf{p})-I_{t+1}(\mathbf{p})\right)\right.\right. \tag{3.13}
\end{gather*}
$$

and all the sums go over the window $\mathcal{W}$. To solve reliably for $\mathbf{d}$, the equation system should be well-conditioned. Shi and Tomasi claim that this is the case if the value of both Eigenvalues of $\mathbf{G}$ do not differ much and if the Eigenvalues exceed a minimum threshold. Speaking in terms of the image content, two small Eigenvalues correspond to an almost uniform content while two Eigenvalues with very different size indicate an edgelike content [96]. Consequently, the authors propose to use the smaller of the two eigenvalues $\lambda_{1}, \lambda_{2}$ as a detection criterion for feature-points. Since $\mathbf{G}=\left\{g_{i k}\right\}$ is a symmetric $2 \times 2$ matrix, the Eigenvalues can be computed easily as

$$
\begin{equation*}
\lambda_{1}, \lambda_{2}=\frac{1}{2}\left(g_{00}+g_{11} \mp \sqrt{\left(g_{00}+g_{11}\right)^{2}-4\left(g_{00} g_{11}-g_{01}^{2}\right)}\right) . \tag{3.14}
\end{equation*}
$$

Detection is carried out by taking $\lambda(x, y)=\min \left\{\lambda_{1}, \lambda_{2}\right\}$ for each image position and assuming a feature-point if $\lambda(x, y)>\tau_{s t}$ and $\lambda(x, y)$ is a local maximum in the image.

## Using a weighted window

In the previously described algorithm, all pixels within the window $\mathcal{W}$ are included equally in the calculation. However, this leads to the problem that the window $\mathcal{W}$ is placed such that the window maximizes the number of high-gradient pixels in the window, regardless of their position in the window. This induces that the detected interest-point position will be


Figure 3.5: Placement of the feature-point at a corner. (a) If all pixels within a window contribute equally (left window), the featurepoint will not be located exactly at the corner, since the algorithm tries to maximize the amount of edge pixels within the window. When pixels at the center are weighted more than pixels further away (right window), the feature-point moves closer to the window center. (b) A Gaussian weighting function.
biased (see Fig. 3.5(a)). To prevent this effect, a weighting function $w$ can be introduced to increase the weight of the center pixels. In our application, we use a cascade of five computationally efficient binomial filters on the gradient vector components to approximate a Gaussian windowing function (Fig. 3.5(b)).

### 3.2.3 Harris corner detector

The Harris corner detector [84] (also known as Plessey detector) is in fact very similar to the Shi-Tomasi operator, but in this section, we would like to derive the operator in a different way to give an alternative, more intuitive explanation of its function. We begin with examining the texture in a window $\mathcal{W}$ around the considered pixel position by observing the gradient vectors in this window. The idea is to classify the texture in the window based on the distribution of the gradient vectors. Figure 3.6 shows scatterplots of the gradient vectors for three example window locations. If we consider Window 1, which contains a linear edge, we obtain the gradientvector distribution as shown in Figure 3.6(c). We see that all the gradient vectors have approximately the same orientation, where the major axis of the distribution is perpendicular to the image edge. For an image corner (Fig. 3.6(e)), different gradient orientations are present, each corresponding to one of the edges. This is comparable to complex texture content (Fig. 3.6(d)) where the gradient vectors are distributed more or less uni-


Figure 3.6: (c)-(e) Scatter plots of gradient vectors for the indicated windows in (a). The detected corner positions are shown in (b).
formly. Finally, for image content with only small luminance variations, all the gradient vectors are close to zero. To classify the different situations, we model the distribution of gradient vectors with a bivariate Gaussian distribution whose principal axes can be obtained as the principal components of the correlation matrix of the gradient vectors. Note that the correlation matrix is exactly equal to $\mathbf{G}$ as we defined it for the Shi-Tomasi operator in Eq. (3.12). Moreover, the length of the principal axes are again the Eigenvalues $\lambda_{1}, \lambda_{2}$ of the correlation matrix. If we look at the length of the principal axes for each of the texture types, we observe that two small axes indicate a flat image content, one large and one small axis indicates a linear structure, while two large axes indicate a corner or other complex texture. Following this observation, Harris and Stephens proposed the classification scheme of Figure 3.7(a) to classify a specific gradient-vector distribution into one of the classes flat, edge, and corner. According to this, a flat re-

(a) Harris operator: texture classification.

(b) Determining sub-pel corner position.

Figure 3.7: (a) Classification of a pixel into one of the three classes flat,edge,corner is based on the two Eigenvalues $\lambda_{1}, \lambda_{2}$ of the correlation matrix. (b) The sub-pel accurate corner position is determined by fitting a quadratic function and taking their minimum position.
gion is detected if $\lambda_{1}+\lambda_{2}<\tau_{h a}$, where $\tau_{h a}$ is a flat-region threshold. The decision function for the corner region is defined by

$$
\begin{equation*}
r=\lambda_{1} \lambda_{2}-k\left(\lambda_{1}+\lambda_{2}\right)^{2}=\operatorname{Det}(\mathbf{G})-k \cdot \operatorname{Tr}(\mathbf{G})^{2}>0, \tag{3.15}
\end{equation*}
$$

where $k \approx 0.06$. Note that the decision function is chosen such that it is not necessary to actually compute the Eigenvalues. Instead, it is sufficient to compute the matrix determinant and the matrix trace by exploiting their equivalence to the product and sum of the Eigenvalues, respectively. To locate the corner-points, local minima of $r$ are determined and a corner is reported at this position if $r>0$ and an additional flat area test $(\operatorname{Tr}(\mathbf{G}) \geq$ $\tau_{h a}$, with $\tau_{h a}$ being a flat area threshold) is passed.

The Harris detector shows the same problem of biased corner-point placement as described earlier for the Shi-Tomasi detector. However, this problem can be easily approached in the same way by introducing pixelweighting within the window. An example result of applying the Harris detector is shown in Figure 3.6(b).

## Extension to sub-pel accuracy

The Harris corner detector described so far determines the location of corners with integer pixel accuracy. We have extended the algorithm by fitting a second-order polynomial to the horizontal and vertical neighborhood and taking the minimum of these polynomials as the sub-pixel position of the corner.

Assume that we found an (integer-accurate) local minimum of decision function $r$ at $r\left(x_{0}, y_{0}\right)$. Since a translation does not affect the shape of the polynomial function, we can assume w.l.o.g. that $x_{0}=y_{0}=0$. In the following, only the computation along the horizontal direction is described, since the computations for the vertical direction are equivalent. Inserting the values $r(-1,0), r(0,0), r(1,0)$ into the model of the polynomial as $a x^{2}+$ $b x+c=r(x, 0)$ allows to easily compute the parameters as $a=(r(-1,0)+$ $r(1,0)) / 2-r(0,0), b=(r(1,0)-r(-1,0)) / 2$, and $c=r(0,0)$. Computing the derivative and setting it to zero results in the position of the minimum

$$
\begin{equation*}
x=\frac{(r(-1,0)-r(1,0)) / 2}{r(-1,0)+r(1,0)-2 r(0,0)} \tag{3.16}
\end{equation*}
$$

which provides the sub-pel correction offset.

### 3.2.4 SUSAN corner detector

The principle of the SUSAN corner detector [170] is based on a morphology approach. While the previous detectors identify positions where the image content changes in differing directions, the SUSAN corner detector directly uses a definition of a corner in its algorithm. For this, it uses the USAN principle (Univalue Segment Assimilating Nucleus). The USAN is the area of a local neighborhood that has the same color as the center pixel of the considered neighborhood. This is illustrated in Figure 3.8 where the circles depict several examples of local neighborhoods.

In the cases where the circle center lies inside of the rectangle, the USAN consists of pixels from the rectangle. If the center is outside of the rectangle, the USAN only consists of pixels outside of the rectangle. When we calculate the size of the USAN area for every possible location, we observe that the USAN area is smallest at the corners (see Figure 3.9). This principle is now used in the SUSAN corner detector (SUSAN="Smallest Univalue Segment Assimilating Nucleus") by searching for local minima of the USAN area. Since the authors of [170] provide source code for the SUSAN corner detector, we integrated the original implementation into our experiments.


Figure 3.8: USAN principle. The USAN is defined as the part of the neighborhood window that has the same color as the center pixel. The USAN area for different window positions is drawn black.


Figure 3.9: SUSAN corner detector. The USAN area for a small sample image. Corners are detected by finding local minima of the USAN area. (Picture is taken from [170].)

### 3.2.5 Evaluation

## Reference correspondences

For the successive evaluations, we need ground-truth data that tells us which feature-point $\mathbf{p}_{i}$ in one frame corresponds to a feature-point $\hat{\mathbf{p}}_{k}$ in the second frame (written as $\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k}$ ). Unfortunately, this information is not available easily and a manual generation of the data is impractical, since there are usually several hundred feature-points in each frame. As a solution, we assume that the transform $\mathbf{H}^{\star}$ between images is known. In our experiments, we used the output of our complete motion estimation system where we checked beforehand that the estimation result was very accurate. Based on this reference transformation $\mathbf{H}^{\star}$, we define a set of reference correspondences $\mathcal{R}$ which we use instead of ground truth data.

Let us denote the set of feature-points in the first image $I_{t}$ as $\mathcal{F}_{t}=\left\{\mathbf{p}_{i}\right\}$ and as $\mathcal{F}_{t+1}=\left\{\hat{\mathbf{p}}_{k}\right\}$ for the second image $I_{t+1}$. We define the set of reference correspondences basically as $\mathcal{R}_{\epsilon}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k} \mid d\left(\hat{\mathbf{p}}_{k}, \mathbf{H}^{\star} \cdot \mathbf{p}_{i}\right)<\epsilon\right\}$, but with the additional constraint that each feature-point is only included once. This set is constructed in a greedy approach where a pair of points is iteratively added to the set if their projection distance $d\left(\hat{\mathbf{p}}_{k}, \mathbf{H}^{\star} \cdot \mathbf{p}_{i}\right)$ is smallest and both points are not used yet in $\mathcal{R}_{\epsilon}$. The parameter $\epsilon$ defines the maximum displacement error that is allowed for a correpondence. This threshold is required, since the interest-point detectors can only estimate the point positions with some amount of error. Choosing a too low value for $\epsilon$ would result in missed feature-correspondences, whereas a too high value would include wrong matches. A good choice for $\epsilon$ will become apparent after we evaluate the repeatability of the interest-point detectors.

## Evaluation criteria

We evaluate the performance of the described interest-point detectors according to two criteria:

- Repeatability: The fraction of detected interest-points in one frame that can also be found in the other frame. Features that are only found in one of the images cannot be part of a point-correspondence and consequently, they are useless for the succeeding motion analysis. Even worse, unmatched features would contaminate the data with noise and may lead to wrong correspondences, which would make the parameter estimation harder. To quantify this property, we define the repeatability of the interest-point detector as the fraction of detected
feature-points in one frame that can also be found in the other frame:

$$
\begin{equation*}
\text { repeatability }=\frac{\left|\mathcal{R}_{\epsilon}\right|}{\min \left(\left|\mathcal{F}_{t}\right|,\left|\mathcal{F}_{t+1}\right|\right)} . \tag{3.17}
\end{equation*}
$$

This is the number of available correspondences, normalized by the minimum number of features in the two input frames. Since we cannot get more correspondences than the minimum number of features, the maximum value for repeatability is 1.0 .

- Accuracy: A good interest-point detector provides unbiased positions that are invariant to the image transform that occurs between the two images. Clearly, a more consistent placement of feature-points leads to a higher accuracy of the motion estimation parameters, since the displacement errors are smaller. Note that there is no direct definition what the ideal position for an feature-point is, but the position should not jitter between frames.
Assuming that we know the ground-truth transformation $\mathbf{H}^{\star}$ between the two frames, we define the accuracy of an interest-point detector as the mean distance between the feature-point position $\hat{\mathbf{p}}_{\mathbf{k}}$ in the second frame and the position of the corresponding feature-point in the first frame, mapped onto the second frame as $\mathbf{H}^{\star} \cdot \mathbf{p}_{\mathbf{i}}$. Feature-points which have no counterpart in the other image are not considered. This gives the definition of the accuracy as

$$
\begin{equation*}
\text { accuracy }=\frac{1}{|\mathcal{R}|} \sum_{\mathbf{p}_{\mathbf{i}} \hookleftarrow \hat{\mathbf{p}}_{\mathbf{k}} \in \mathcal{R}} d\left(\hat{\mathbf{p}}_{k}, \mathbf{H}^{\star} \cdot \mathbf{p}_{i}\right) . \tag{3.18}
\end{equation*}
$$

Note that a lower accuracy value indicates a better accuracy. Optimal accuracy is reached for accuracy $=0$.

## Evaluation results

The diagrams in Figure 3.10 show the repeatability for four test sequences ${ }^{3}$ and all of the previously described interest-point detectors, using $\epsilon$ as a parameter. In each diagram, the repeatability is averaged over the complete sequence length. From these diagrams and the results of other sequences, we see that the Harris detector reaches a repeatability between $80-90 \%$, where saturation is almost reached for $\epsilon \approx 1.5$. Note that a repeatability near $100 \%$ cannot be reached since new image content appears and old

[^5]

Figure 3.10: Repeatability of the detected feature-points: fraction of corresponding feature-points. Feature-points from one image are transformed into the second image. They match a featurepoint in the second image if there is a detected feature-point within a neighborhood $\epsilon$. (Continued in Fig. 3.11.)


Figure 3.11: Continuation of Fig. 3.10.

| Accuracy (pixels) | roma | rail | opera4 | nature2 | Average |
| :--- | :---: | :---: | :---: | :---: | :---: |
| Harris (sub-pel) | 0.34 | 0.56 | 0.51 | 0.46 | 0.47 |
| Harris (integer) | 0.53 | 0.74 | 0.65 | 0.67 | 0.65 |
| Shi-Tomasi | 0.53 | 0.80 | 0.95 | 0.83 | 0.78 |
| SUSAN | 0.75 | 1.13 | 0.98 | 1.11 | 0.99 |
| Moravec | 0.74 | 1.05 | 0.76 | 0.98 | 0.88 |

Table 3.1: Accuracy results for the four test sequences roma, rail, opera4, and nature2.
content disappears during the sequences. Especially the feature-points near the image border are often impossible to match since their counterpart is outside of the visible image area.

The repeatabilities of all other detectors are clearly below the Harris detector and they also do not reach a clear saturation, which is due to a lower localization accuracy. The Shi-Tomasi detector and SUSAN reach both a repeatability rate between $40-70 \%$, while the Moravec operator only reaches about $30-40 \%$. ${ }^{4}$

The results for the accuracy of the four sequences are summarized in Table 3.1. It can be noticed that the Harris detector again shows the best performance and that its accuracy can indeed be increased by our sub-pel refinement.

## Conclusion

According to our experiments, the Harris detector showed the best performance for repeatability and also for accuracy. While all detectors yield a good accuracy, the repeatability is not satisfactory except for the Harris detector. As a consequence, we selected the Harris detector for our motionestimation system and we will omit the other detectors in the successive sections. It can also be seen that the sub-pel refinement can in fact increase the accuracy of the Harris detector by about 0.2 pixels.

### 3.3 Computing feature-correspondences

After feature-points have been extracted for each video frame, it is necessary to establish correspondences between the feature-points. Each correspondence indicates the motion of one image position with a high confidence. Computing the correspondences is subject to a number of problems. As

[^6]we have seen in the previous section, no interest-point detector is able to yield an ideal repeatability. This means that our data-sets will always be contaminated with feature-points that have no matching counterpart in the other image. However, this is not only due to imperfections of the detection algorithms, but it is also caused by the video content itself. One reason are foreground objects that occlude parts that still were visible in the last frame, or objects that uncover previously hidden areas. Comparable to the occlusion problem is the effect that a moving camera results in displacing the video content such that areas at the image border fall outside of the image area while new parts move into the image at the opposite side. Finally, it should be noted that the number of feature-points is usually very high (about 1000-2000 for CIF-resolution video), which emphasizes the need for a computationally efficient algorithm.

The algorithms for feature-correspondences can be coarsely classified into two categories: algorithms that take the texture around a feature-point into account to guide the matching process, and algorithms that only use the geometric point locations. The second type of algorithm has advantages when the transformation between successive images is large, like a rotation of $180^{\circ}$. In this case, it is difficult to define a reliable similarity measure for feature-points. Similarity measures that are invariant to some geometric transformations are a current area of research. However, in our application to video data, the transformation between images is relatively small and we can use a simple measure like the sum of absolute differences of the luminance data in the neighborhood to measure feature-point similarity.

### 3.3.1 Fast greedy algorithm

Several algorithms for computing feature-point correspondences have been proposed in the literature [121, 147]. Some of them are especially designed to allow for large motions [182]. However, for our application, the motion is relatively small and well predictable from previous frames. On the other hand, many feature-points are extracted for typical video content, so that a low computational complexity becomes important. For this reason, we have developed a fast $a d$-hoc algorithm to compute point-feature correspondences. In the following, we first describe the algorithm core, which we subsequently modify to decrease the computational complexity and also enhance the accuracy.

## Algorithm core

Let us denote the feature-points in the first image $I_{t}$ as $\mathbf{p}_{i}=\left(x_{i}, y_{i}, 1\right)$ and in the second image $I_{t+1}$ as $\hat{\mathbf{p}}_{k}=\left(\hat{x}_{k}, \hat{y}_{k}, 1\right)$. Furthermore, we define a
dissimilarity metric on the pairs of feature-points, using the sum of absolute errors distance, defined as

$$
\begin{equation*}
m_{i, k}=\sum_{r=-7}^{7} \sum_{s=-7}^{7}\left|I_{t}\left(x_{i}+r, y_{i}+s\right)-\hat{I}_{t+1}\left(\hat{x}_{k}+r, \hat{y}_{k}+s\right)\right| . \tag{3.19}
\end{equation*}
$$

Finally, we denote the Euclidean distance between the points $\mathbf{p}_{i}$ and $\hat{\mathbf{p}}_{k}$ as $d\left(\mathbf{p}_{i}, \hat{\mathbf{p}}_{k}\right)$. Under the assumption that the motion between successive frames is small, we only consider feature correspondences between points that are within a search-range $d_{\text {max }}$. The purpose of this threshold is to reduce the computational complexity and also to block correspondences between points that are obviously too far away.

The feature-matching algorithm proceeds as follows.

1. We start the algorithm by establishing the set of all admissible candidate correspondences

$$
\begin{equation*}
\mathcal{C}_{0}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k} \mid d\left(\mathbf{p}_{i}, \hat{\mathbf{p}}_{k}\right) \leq d_{\max }\right\} . \tag{3.20}
\end{equation*}
$$

2. For each of the correspondences $\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k}$ in $\mathcal{C}_{0}$, we compute and store the dissimilarity measure $m(i, k)$ in a matrix.
3. In a greedy approach, we iteratively select the candidate correspondence $c=\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k}$ with the lowest dissimilarity measure and add that correspondence to the final set of correspondences: $\mathcal{C}:=\mathcal{C} \cup\{c\}$.
4. Since each feature-point may only participate in one correspondence, we remove all candidate correspondences from $\mathcal{C}_{0}$ that have a featurepoint in common with the selected correspondence:

$$
\begin{equation*}
\mathcal{C}_{0}:=\left\{\mathbf{p}_{c} \leftrightarrow \hat{\mathbf{p}}_{d} \in \mathcal{C}_{0} \mid \mathbf{p}_{c} \neq \mathbf{p}_{i} \wedge \hat{\mathbf{p}}_{d} \neq \hat{\mathbf{p}}_{k}\right\} . \tag{3.21}
\end{equation*}
$$

The algorithm ends when $\mathcal{C}_{0}$ is empty or the dissimilarity error exceeds a threshold $m_{\max }$. This threshold $m_{\max }$ is required to prevent the algorithm from associating completely dissimilar feature-points which may remain after most correct correspondences have been found.

## Reducing the search-range with motion prediction

The computational complexity of the above algorithm is primarily influenced by the size of the initial candidate set $\mathcal{C}_{0}$, since for each candidate correspondence, the dissimilarity measure has to be calculated. One possible approach to reduce the candidate set would be to reduce the search-range


Figure 3.12: Reducing the required search-range with a motion prediction. Assuming that the past motion $\mathbf{H}_{0}$ does not change much, the feature-point $\mathbf{p}_{i}$ is projected into the future to the position $\mathbf{H}_{0} \mathbf{p}_{i}$. This allows for a smaller search-range d dax compared to the search-range $d_{\max }$ in the case without prediction.
$d_{\text {max }}$, but this would make the above algorithm unusable for sequences with fast motion. Using a similar prediction approach as has been used previously for block-based motion [35, 36], we assume that the motion between successive frames is smooth without abrupt changes. This allows us to use the motion model from the previous pair of frames as a predictor to estimate the position of the feature-points in the current frame. Hence, we can limit the search-range for matching features to only a small neighborhood around this estimated feature position. Using the motion model that we computed for the previous pair of images, we can replace Eq. (3.20) by

$$
\begin{equation*}
\mathcal{C}_{0}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{k} \mid d\left(\mathbf{H}_{0} \mathbf{p}_{i}, \hat{\mathbf{p}}_{k}\right) \leq d_{\max }^{\prime}\right\}, \tag{3.22}
\end{equation*}
$$

where $\mathbf{H}_{0}$ is the motion model prediction and $d_{\text {max }}^{\prime}$ is the radius of the search-range around the predicted position (Fig. 3.12). Because the predicted feature position will be closer to the actual position, we can choose $d_{\text {max }}^{\prime}$ smaller than $d_{\text {max }}$, which reduces the computational complexity without sacrifying the ability to handle large motions. The predicted displacement is not limited, so that the actual effective search-range is unlimited. However, we have the restriction that the motion may not change in speed too quickly.

## Fast neighborhood search with 2-D bucket sort

The evaluation of Eq. (3.20) and (3.22) requires finding the set of featurepoints $\hat{\mathbf{p}}_{k}$, that are within a maximum distance $d_{\max }$ around a position $\mathbf{H}_{0} \mathbf{p}_{i}$. The naive approach to iterate through all feature-points and com-


Figure 3.13: The image area is divided into small squares (buckets) of size $b \times b$. A list of feature-points is associated with each bucket. To get a list of features in a neighborhood of a given point, only the features in the buckets intersecting the neighborhood have to be checked (indicated with grey).
pute the distance requires substantial computation time because the number of feature-points can be larger than 1000 . Consequently, $1000 \times 1000$ comparisons would be required for each frame. We can reduce this considerably by storing the feature-points in a data structure which allows for an efficient search for close points. A simple solution is to partition the image area into a lattice of small square cells where each cell $(m, n)$ consists of a set $\mathcal{B}_{m, n}$, storing the feature-points located in the area of this cell:

$$
\begin{equation*}
\mathcal{B}_{m, n}=\left\{\hat{\mathbf{p}}_{k}=(x, y) \left\lvert\,\left\lfloor\frac{x}{b}\right\rfloor=m \wedge\left\lfloor\frac{y}{b}\right\rfloor=n\right.\right\} . \tag{3.23}
\end{equation*}
$$

The parameter $b$ is the cell width and can be chosen as, e.g., $b \approx d_{\text {max }}^{\prime}$, but the actual choice is not critical. Locating the set of features within a maximum distance can now be carried out efficiently by first determining which cells intersect the search-range (Fig. 3.13). The points within cells that are completely covered by the search-range can be taken into the solution set without further testing. Points within cells that are only partially covered by the search-range have to be tested individually.

### 3.3.2 Evaluation

A typical output of the algorithm is visualized in Figure 3.14. The correspondences are drawn with lines where the small circle indicates the feature position in the second frame. Unmatched feature-points are drawn with
larger circles (unfilled for features of the first frame and filled for features of the second frame). Most unmatched features are located along the image border, since this part of the image is visible in only one of the two frames. Other sporadic unmatched features are due to a non-perfect repeatability in the interest-point detector. It is obvious that the number of unmatched features increases quickly if the search-range is too small and motion prediction is not used (see Figure 3.15). However, even in this case, a correct estimate of the background motion would be possible, because enough correspondences have still been found.

In general, a large search-range is required if the motion is very fast. But unfortunately, a large search-range increases the probability to find non-correct matches (see Fig. 3.16(a)), and it also increases the computation time. However, if we use motion prediction to relocate the center of the search-range, we can obtain the correct correspondences with a much smaller search-range (see Fig. 3.16(b)). At the same time, the number of wrong matches is reduced, because the algorithm locks to the camera motion $\mathbf{H}_{0}$ and since it only detects correspondences that fit into the globalmotion model.

The effect of motion prediction is evaluated further in Figure 3.17 for four test sequences. Depicted is the recall rate $|\mathcal{C} \cap \mathcal{R}| /|\mathcal{R}|$ (the fraction of correctly found correspondences) depending on the search-range. Also shown is the fraction of the total correspondences found $|\mathcal{C}| /|\mathcal{R}|$ (including the wrong matches). All values are averaged over the complete sequence length. The diagrams show clearly that a much smaller search-range of only about two pixels (on the average) is required when motion prediction is enabled, whereas a much larger range is required without prediction. Obviously, both approaches converge for larger search-ranges.

In a second experiment, we explored the dependency of the recall rate on the maximum matching error that is allowed for a feature-correspondence. Since the image motion in general is non-translatorial, but the feature similarity is computed assuming only a translatorial model, the feature error increases for larger non-translatorial transforms. Hence, it is clear that a higher threshold on the matching error allows more feature-correspondences to be established. On the other hand, a higher threshold also induces more erroneous features matches. Figure 3.19 again shows the recall rate, but now for different maximum matching errors. Since we found that for higher thresholds, the number of correct correspondences as well as the number of errors increases, a good trade-off value has to be found. Based on the results using our test sequences, we chose $m_{\max }=5000$ as threshold. However, we also noticed that the choice of this threshold is non-critical, as the final result of the motion estimation was similar over a wide range of $m_{\max }$.


Figure 3.14: Typical output of our greedy point-correspondence algorithm. Spurious unmatched feature-points exist from the imperfect interest-point detector. Since these points are not passed to the next stage, they do not degrade the total algorithm output.


Figure 3.15: Computed motion between frames 236 and 238 without motion prediction. It can be seen that the search-range ( 40 pixels) is not sufficient to find correspondences at the left part of the audience and in the tennis-player object. (Motion on the left side is faster since a left pan is combined with a zoom-out.)

(a) No motion prediction. A large search-range (70 pixels) is required, which also increases the number of wrong matches.

(b) Enabled motion prediction with small searchrange ( 8 pixels). The algorithm locks to the camera motion and the number of wrong matches is reduced considerably.

Figure 3.16: Comparison of the correpondence algorithm without motion prediction and with motion prediction for a scene with fast motion.


Figure 3.17: Recall rate of correct correspondences for different searchranges $d_{\max }$ (solid line). The maximum matching error $m_{\max }$ is fixed to 5000. For comparison, the total number of returned correspondences (also normalized to $|\mathcal{R}|$ ) is shown with a dashed line. The wrong correspondences can be seen as the difference between the two values. (Continued in Fig. 3.18.)


Figure 3.18: Continuation of Fig. 3.17.


Figure 3.19: Recall rate of correct correspondences for different maximum matching errors $m_{\max }$ (solid line). The search-range is fixed to $d_{\max }^{\prime}=3$, motion prediction is turned on. For comparison, the total number of returned correspondences (also normalized to $|\mathcal{R}|)$ is shown with a dashed line. The wrong correspondences can be seen as the difference between the two values.

### 3.4 Summary

This chapter presented the feature-point detection and matching steps of the feature-based motion estimator. First, we evaluated four different interest-point detectors for their performance in our application. In particular, we defined and measured the repeatability and the accuracy of the detectors. The repeatability is the number of features that are detected in both images of a pair. A high repeatability is desired, since only features that are detected in both images can establish a correct correspondence for the successive motion estimation step. The accuracy of an interest-point detector is the variance of the detected feature position in different images. A high accuracy is desired, because inconsistent placement of the feature points would finally lead to less accurate motion parameters. Based on our experiments, we concluded that the Harris interest-point detector with our sub-pel refinement provided the best repeatability of approx. $90 \%$ and also the highest accuracy of about 0.5 pixels.

Furthermore, we presented the feature-matching algorithm. The algorithm is a greedy highest-confidence-first algorithm that first groups features for which the SAD matching error of a small window around the feature is smallest. We added a motion prediction step that predicts the position of a feature in the new frame based on the motion model between the last pair of frames. The feature matching is limited to a small neighborhood around this predicted position. This motion prediction has two advantages: first, the small neighborhood allows for a computationally fast search, compared to considering all detected features. Second, it prevents that features are matched that deviate much from the current camera motion. This effectively decreases the number of wrong matches.

## Architecture of the feature-point detection and matching

The data-flow of the feature-point detection and the feature matching step is depicted in Figure 3.20. For a pair of temporally successive pictures, feature-points are extracted with the Harris detector. Note that for each step only one input frame has to be processed, since the features of the other frame can be reused from the last step. The features $\mathcal{F}_{t}$ are projected onto the next frame, using the previously computed camera-motion model $\mathbf{H}_{t ; t-1}$. Matching the two feature sets $\mathcal{F}_{t}$ and $\mathcal{F}_{t+1}$ results in a set of correspondences $\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}\right\}$ that will be used to determine the cameramotion parameters (as described in the subsequent chapter). The computed camera-motion parameters are then used to predict the feature positions in the next pair of frames.


Figure 3.20: Feature-point detection and matching of corresponding features.

Never mistake motion for action. (Ernest Hemingway)

## Chapter

## Feature-Based Motion II: Parameter Estimation

This chapter describes the algorithm for computing the parameters of the projective motion model, based on the feature-correspondences that we obtained in the last chapter. We construct the algorithm step by step, starting with a simpler affine motion model, prior to considering the projective motion model. Whereas the parameter estimation for affine motion can be realized with linear least-squares, an equivalent problem formulation for projective motion leads to a non-linear optimization problem. Furthermore, we study the case of images with multiple independent motions in the same frame. To extract the dominant motion model in this case, we apply the RANSAC algorithm, which is a robust estimation algorithm that is not affected by outlier data. An evaluation of the robust estimation algorithm shows that the accuracy of the results in practice is worse than expected from a theoretic evaluation. However, after analyzing this discrepancy, we propose a modification to reach the theoretical performance.

### 4.1 Introduction

Chapter 2 presented how camera motion can be described with a projective motion model. In this chapter, we address algorithms to solve the inverse problem of estimating the model parameters from a set of pointcorrespondences.

We describe the algorithm in two steps. First, we assume that the video sequence shows only background motion without foreground objects that move in a different direction. This allows us to include all the correspondences in the parameter estimation. In Section 4.3, we describe an enhanced algorithm that generalizes the algorithm such that foreground motion and outlier data are excluded from to estimating the motion parameters. The applied algorithm is the RANSAC (Random Sample Consensus) [73] algorithm, which is a probabilistic algorithm only succeeding with a certain probability. Our experiments show that the practical performance does not reach the theoretically predicted probability of success. We will derive an explanation and propose a modification to increase the robustness in Section 4.3.3.

Besides the RANSAC algorithm, other robust estimation algorithms have been proposed. We conducted experiments with the LTS (Least Trimmed Squares) and LMedS (Least Median of Squares) algorithms. These algorithms are explained and compared to RANSAC in Appendix C.

### 4.2 Computing motion model parameters

In the following two sections, we will first consider the estimation problem for the affine motion model, since this can be solved with linear leastsquares. After that, we will discuss the projective motion model in Sections 4.2.3 and 4.2.4.

### 4.2.1 One-dimensional affine motion

Let us first illustrate the principles for simple one-dimensional affine motion. If we denote the positions in the first (one-dimensional) picture by $x_{i}$ and the corresponding position in the second picture by $x_{i}^{\prime}$, we can formulate the affine motion model as $x_{i}^{\prime}=a \cdot x_{i}+b$. Each selection of model parameters defines a line in an $x, x^{\prime}$ diagram which illustrates the corresponding positions between $x$ and $x^{\prime}$. The two possible types of motion which are possible with this simple transform are depicted in Figure 4.1(a). The two types of motion are translatorial motion, which is specified using the $b$ parameter and zoom, which is specified with the $a$ parameter.


Figure 4.1: One-dimensional affine motion. The horizontal axis shows the position $x$ in the first picture while the vertical axis shows the corresponding position $x^{\prime}$ in the second picture. (a) Lines in the $x, x^{\prime}$ diagram depict the motion between $x$ and $x^{\prime}$. (b) From the feature-correspondence step, we get a set of (noisy) correspondences $\left\{x \leftrightarrow x^{\prime}\right\}$ which are drawn as dots. A leastsquares fit is carried out by minimizing the sum of model errors $\left|g\left(x_{i}\right)-x_{i}{ }^{i}\right|$.

The center of the zoom is at the position where the model line crosses the identity line $x=x^{\prime}$.

The parameter-estimation problem is now to obtain an estimate for the parameters $a$ and $b$, based on a set of measured point-correspondences. We denote a set of points in the first image as $\left\{x_{i}\right\}$ and their corresponding points in the second image as $\left\{\hat{x}_{i}\right\}$. Since the point measurements are not exact, we cannot assume that they will all fit perfectly to the motion model. Hence, the best solution is to compute a least-squares fit to the data. We consequently define the model error as the sum of squared distances between the measured positions $\hat{x}_{i}$ and the positions obtained from the motion model (Figure 4.1(b)). This results in the definition of the model error as $E=\sum_{i}\left(\left(a x_{i}+b\right)-\hat{x}_{i}\right)^{2}$. To minimize the model error $E$, we take its derivatives with respect to the motion parameters

$$
\begin{equation*}
\frac{\partial E}{\partial a}=\sum_{i} 2\left(a x_{i}+b-\hat{x}_{i}\right) x_{i} \quad ; \quad \frac{\partial E}{\partial b}=\sum_{i} 2\left(a x_{i}+b-\hat{x}_{i}\right), \tag{4.1}
\end{equation*}
$$

and set them to zero. This leads to the two equations

$$
\begin{equation*}
\sum_{i}\left(a x_{i}^{2}+b x_{i}-\hat{x}_{i} x_{i}\right)=0 \quad ; \quad \sum_{i}\left(a x_{i}+b-\hat{x}_{i}\right)=0 \tag{4.2}
\end{equation*}
$$

which can be written in matrix form as

$$
\left[\begin{array}{ll}
\sum_{i} x_{i}^{2} & \sum_{i} x_{i}  \tag{4.3}\\
\sum_{i} x_{i} & \sum_{i} 1
\end{array}\right]\binom{a}{b}=\binom{\sum_{i} \hat{x}_{i} x_{i}}{\sum \hat{x}_{i}} .
$$

By solving this linear equation system, we can determine the unknown model parameters $a$ and $b$. Note that the problem solved so far is mathematically identical to the problem of simple linear regression.

### 4.2.2 Two-dimensional affine motion

In the two-dimensional case, our measurements consist of positions $\left(x_{i}, y_{i}\right)$ in the first image and corresponding position $\left(\hat{x}_{i}, \hat{y}_{i}\right)$ in the second image. The position that is obtained by transforming ( $x_{i}, y_{i}$ ) according to the motion model will be denoted as $\left(x_{i}^{\prime}, y_{i}^{\prime}\right)$. Now, it is the 2-D affine motion model

$$
\binom{x^{\prime}}{y^{\prime}}=\left[\begin{array}{ll}
a_{00} & a_{01}  \tag{4.4}\\
a_{10} & a_{11}
\end{array}\right]\binom{x}{y}+\binom{t_{x}}{t_{y}},
$$

for which we want to find a good estimate of the six parameters $\left\{a_{i k}\right\}, t_{x}, t_{y}$. As a direct generalization of the model error of the one-dimensional case, we can define the model error as: $E_{2}=\sum_{i}\left(x_{i}^{\prime}-\hat{x}_{i}\right)^{2}+\left(y_{i}^{\prime}-\hat{y}_{i}\right)^{2}$. In a geometrical sense, this is the sum of Euclidean distances between the measured positions in the second frame and the positions to which the features from the first image are transformed (Fig 4.2). Note that this definition assumes that the measurements in the first frame are exact and errors are only made in measuring the position in the second picture. Since this is not true in practice, it is proposed in [85] to use a more symmetric error definition like the symmetric transfer error or the reprojection error. However, this would lead to a more complicated solution with only very little increase of accuracy. Consequently, we will use the definition of Euclidean error $E_{2}$.

To solve for the minimum error $E_{2}$, we again take the partial derivatives with respect to the model parameters $a\{i j\}, t_{x}, t_{y}$ and set them to zero. This gives the equation system

$$
\left[\begin{array}{cccccc}
\sum_{i} x_{i}^{2} & \sum_{i} x_{i} y_{i} & \sum_{i} x_{i} & 0 & 0 & 0 \\
\sum_{i} x_{i} y_{i} & \sum_{i} y_{i}^{2} & \sum_{i} y_{i} & 0 & 0 & 0 \\
\sum_{i} x_{i} & \sum_{i} y_{i} & \sum_{i} 1 & 0 & 0 & 0 \\
0 & 0 & 0 & \sum_{i} x_{i}^{2} & \sum_{i} x_{i} y_{i} & \sum_{i} x_{i} \\
0 & 0 & 0 & \sum_{i} x_{i} y_{i} & \sum_{i} y_{i}^{2} & \sum_{i} y_{i} \\
0 & 0 & 0 & \sum_{i} x_{i} & \sum_{i} y_{i} & \sum_{i} 1
\end{array}\right]\left(\begin{array}{c}
a_{00} \\
a_{01} \\
t_{x} \\
a_{10} \\
a_{11} \\
t_{y}
\end{array}\right)=\left(\begin{array}{c}
\sum_{i} \hat{x}_{i} x_{i} \\
\sum \hat{x}_{i} y_{i} \\
\sum \hat{x}_{i} \\
\sum_{i} \hat{y}_{i} x_{i} \\
\sum \hat{y}_{i} y_{i} \\
\sum \hat{y}_{i}
\end{array}\right),
$$

which obviously can be solved more easily by splitting the equation system into two independent systems. The first one determines the parameters for


Figure 4.2: The model error is specified as the Euclidean distance between the detected feature position in the second frame and the ideal feature position according to the motion model.
the horizontal motion component $a_{00}, a_{01}, t_{x}$, while the second one determines parameters $a_{10}, a_{11}, t_{y}$ for the vertical component.

### 4.2.3 One-dimensional projective motion

Let us now switch from the affine model to projective motion. We again consider the one-dimensional case first, for which we use a one-dimensional projective motion model in the inhomogeneous representation $x_{i}^{\prime}=(a$. $x+b) /(c \cdot x+1)$. The most important difference for the estimation is the fact that this motion model is not linear anymore. Would we use the same model error definition as above and proceed with the same approach, we would get a non-linear equation system which is much more difficult to solve. However, we can apply a trick to linearize the equation system by modifying the model error definition. Instead of using the Euclidean distance

$$
\begin{equation*}
E_{2}(a, b, c)=\sum_{i}\left(x_{i}^{\prime}-\hat{x}_{i}\right)^{2}=\sum_{i}\left(\frac{a x_{i}+b}{c x_{i}+1}-\hat{x}_{i}\right)^{2} \tag{4.5}
\end{equation*}
$$

we multiply with the nominator of the motion model and obtain the algebraic error
$E_{a}(a, b, c)=\sum_{i}\left(\left(\frac{a x_{i}+b}{c x_{i}+1}-\hat{x}_{i}\right) \cdot\left(c x_{i}+1\right)\right)^{2}=\sum_{i}\left(a x_{i}+b-\hat{x}_{i}\left(c x_{i}+1\right)\right)^{2}$.
With this new error definition, we can again compute the partial derivatives and set them to zero to obtain the optimal parameter estimate. After reordering the obtained equations, we can write them as the equation sys-


Figure 4.3: Estimation of parameters for the perspective motion model using linear least-squares on a algebraic distance, and using nonlinear least-squares on the Euclidean distance.
tem

$$
\left[\begin{array}{ccc}
\sum_{i} x_{i}^{2} & \sum_{i} x_{i} & \sum_{i}-x_{i}^{2} \hat{x}_{i}  \tag{4.7}\\
\sum_{i} x_{i} & \sum_{i} 1 & \sum_{i}-x_{i} \hat{x}_{i} \\
\sum_{i} x_{i}^{2} \hat{x}_{i} & \sum_{i} x_{i} \hat{x}_{i} & \sum_{i}-x_{i}^{2} \hat{x}_{i}^{2}
\end{array}\right]\left(\begin{array}{c}
a \\
b \\
c
\end{array}\right)=\left(\begin{array}{c}
\sum_{i} x_{i} \hat{x}_{i} \\
\sum_{i} \hat{x}_{i} \\
\sum_{i} x_{i} \hat{x}_{i}^{2}
\end{array}\right) .
$$

The use of the algebraic error instead of the Euclidean error enables a more easy computation of the parameters, since only a small linear equation system has to be solved. However, the penalty for this simplification of the computation is a reduction of parameter accuracy. Because of the changed definition of our model error, we now optimize a geometrically meaningless algebraic distance. As long as the noise level in the data is low, the difference between both models is small, but it increases with a larger noise variance. This behaviour is illustrated in Figure 4.3, where random sample data was generated for an example model with the parameters $a=$ $2, b=3, c=0.5$. In Fig $4.3(\mathrm{a})$, the data was distorted by Gaussian noise with $\sigma=0.01$ and in Fig $4.3(\mathrm{~b})$, a higher noise level of $\sigma=0.03$ was chosen. It can be seen that the non-linear least-squares fit using squared Euclidean distances closely approximates the internal parameters. The fit using algebraic distances results in a reasonable solution for low noise, but it is strongly biased in the case of high noise.

### 4.2.4 Two-dimensional projective motion

Let us now extend the one-dimensional case to estimating the parameters of two-dimensional projective motion. Recall that we want to determine the
homography matrix $\mathbf{H}$, describing the motion from points $\mathbf{p}_{i}$ in one frame to points $\mathbf{p}_{i}^{\prime}=\mathbf{H} \mathbf{p}_{i}$. When estimating the parameters $\left\{h_{i k}\right\}$, we have to consider that the parameters are scaling invariant. In the previous section, we adopted the inhomogeneous representation of the motion model. In the two-dimensional case, we pursue a similar approach by assuming that $h_{22}=1$. As we have seen in Section 2.3.3, this normalization fails for the case where the horizon line includes the coordinate origin, since in that case $h_{22}=0$. An alternative is to use the overcomplete parameterization and to impose additional constraints like the unit norm $\|\mathbf{H}\|_{F}=1$ where $\|\cdot\|_{F}$ is the Frobenius norm. This second approach imposes no restrictions on the transform and thus works in any case. However, it is computationally more complex since it leads to computing a Singular Value Decomposition [85]. For inter-frame motion, usage of the inhomogeneous formulation is no problem because the motion is relatively small. The problem only becomes apparent for large rotation angles (as we will see in Chapter 6).

Recall the normalized perspective motion equations

$$
\begin{equation*}
x^{\prime}=\frac{h_{00} x+h_{01} y+h_{02}}{h_{20} x+h_{21} y+1}, \quad y^{\prime}=\frac{h_{10} x+h_{11} y+h_{12}}{h_{20} x+h_{21} y+1} . \tag{4.8}
\end{equation*}
$$

Since the definition of an Euclidean error measure $E_{2}=\sum_{i}\left(x_{i}^{\prime}-\hat{x}_{i}\right)^{2}+$ $\left(y_{i}^{\prime}-\hat{y}_{i}\right)^{2}$ would again lead to a complicated non-linear equation system, we will use an algebraic error in a similar way as in the previous section by defining

$$
\begin{align*}
E_{a}= & \sum_{i} \underbrace{\left(\left(x_{i}^{\prime}-\hat{x}_{i}\right)^{2}+\left(y_{i}^{\prime}-\hat{y}_{i}\right)^{2}\right)}_{\text {Euclidean distance }}\left(h_{20} x+h_{21} y+1\right)^{2} \\
= & \left(h_{00} x+h_{01} y+h_{02}-\hat{x}_{i}\left(h_{20} x+h_{21} y+1\right)\right)^{2}+  \tag{4.9}\\
& \left(h_{10} x+h_{11} y+h_{12}-\hat{y}_{i}\left(h_{20} x+h_{21} y+1\right)\right)^{2} .
\end{align*}
$$

Imposing the necessary condition $\partial E_{a} / \partial h_{i k}=0$ for a minimum error results in the linear equation system of the form

$$
\begin{equation*}
\left(\sum_{i} \mathbf{A}_{i}\right) \mathbf{h}=\sum_{i} \mathbf{b}_{i}, \tag{4.10}
\end{equation*}
$$

consisting of a sum of matrices $\mathbf{A}_{i}$ and a sum of vectors $\mathbf{b}_{i}$ on the righthand side. Using the abbreviation $\hat{s}_{i}=\left(\hat{x}^{2}+\hat{y}^{2}\right)$, the $\mathbf{A}_{i}$ and $\mathbf{b}_{i}$ evaluate


Figure 4.4: (a) The correspondences obtained from the previous processing steps. Most of the correspondence vectors are correct, only a few are established between unmatching feature-points. (b) The projective motion-model fitted to the correspondences using linear least-squares with the algebraic distance measure.
as

$$
\mathbf{A}_{i}=\left[\begin{array}{cccccccc}
x_{i}^{2} & x_{i} y_{i} & x_{i} & 0 & 0 & 0 & -x_{i}^{2} \hat{x}_{i} & -x_{i} y_{i} \hat{x}_{i}  \tag{4.11}\\
x_{i} y_{i} & y_{i}^{2} & y_{i} & 0 & 0 & 0 & -x_{i} y_{i} \hat{x}_{i} & -y_{i}^{2} \hat{x}_{i} \\
x_{i} & y_{i} & 1 & 0 & 0 & 0 & -x_{i} \hat{x}_{i} & -y_{i} \hat{x}_{i} \\
0 & 0 & 0 & x_{i}^{2} & x_{i} y_{i} & x_{i} & -x_{i}^{2} \hat{y}_{i} & -x_{i} y_{i} \hat{y}_{i} \\
0 & 0 & 0 & x_{i} y_{i} & y_{i}^{2} & y_{i} & -x_{i} y_{i} \hat{y}_{i} & -y_{i}^{2} \hat{y}_{i} \\
0 & 0 & 0 & x_{i} & y_{i} & 1 & -x_{i} \hat{y}_{i} & -y_{i} \hat{y}_{i} \\
x_{i}^{2} \hat{x}_{i} & x_{i} y_{i} \hat{x}_{i} & x_{i} \hat{x}_{i} & x_{i}^{2} \hat{y}_{i} & x_{i} y_{i} \hat{y}_{i} & x_{i} \hat{y}_{i} & -x_{i}^{2} \hat{s}_{i} & -x_{i} y_{i} \hat{s}_{i} \\
x_{i} y_{i} \hat{x}_{i} & y_{i}^{2} \hat{x}_{i} & y_{i} \hat{x}_{i} & x_{i} y_{i} \hat{y}_{i} & y_{i}^{2} y_{i} & y_{i} \hat{y}_{i} & -x_{i} y_{i} \hat{s}_{i} & -y_{i}^{2} \hat{s}_{i}
\end{array}\right]
$$

and

$$
\mathbf{b}_{i}=\left(\begin{array}{llllllll}
x_{i} \hat{x}_{i} & y_{i} \hat{x}_{i} & \hat{x}_{i} & x_{i} \hat{y}_{i} & y_{i} \hat{y}_{i} & \hat{y}_{i} & x_{i} \hat{s}_{i} & y_{i} \hat{s}_{i} \tag{4.12}
\end{array}\right)^{\top} .
$$

The solution is collected in the parameter vector

$$
\mathbf{h}=\left(\begin{array}{llllllll}
h_{00} & h_{01} & h_{02} & h_{10} & h_{11} & h_{12} & h_{20} & h_{21} \tag{4.13}
\end{array}\right)^{\top} .
$$

Figure 4.4 shows an example result of applying the linear least-squares fitting algorithm for the perspective motion model. The rail sequence is a pure background sequence without foreground objects, so no correspondence outliers from foreground motion are present. However, there is a
small number of outliers that are errors of the feature-correspondence algorithm. These few outliers have not much influence since the number of correct correspondences is significantly larger.

### 4.2.5 Non-linear least-squares estimation

We have seen previously that the algebraic error measure can result in an inaccurate estimate if the noise in the data is high. Consequently, we will develop an alternative least-squares estimator in this section which directly uses the Euclidean error metric.

Let $\mathcal{C}=\left\{\mathbf{x}_{i} \leftrightarrow \hat{\mathbf{x}}_{i}\right\}$ be a set of point-correspondences. We want to find parameters for $\mathbf{H}$ to minimize the error defined as

$$
\begin{equation*}
E_{2}=\sum_{i} e_{i ; x}^{2}+e_{i ; y}^{2} \tag{4.14}
\end{equation*}
$$

where $e_{i ; x}$ and $e_{i ; y}$ are the residuals of a single point-correspondence in horizontal and vertical direction:

$$
\begin{equation*}
e_{i ; x}=\frac{h_{00} x_{i}+h_{01} y_{i}+h_{02}}{h_{20} x_{i}+h_{21} y_{i}+1}-\hat{x}_{i} \quad ; \quad e_{i ; y}=\frac{h_{10} x_{i}+h_{11} y_{i}+h_{12}}{h_{20} x_{i}+h_{21} y_{i}+1}-\hat{y}_{i} . \tag{4.15}
\end{equation*}
$$

To find a solution, we use the Levenberg-Marquardt algorithm [151]. This algorithm is a combination of a gradient-descent and Newton-like algorithm. Apart from the error function, the algorithm also requires the partial derivatives with respect to the parameters (for the gradient-descent) and the Hessian matrix (for the Newton optimization-algorithm). Using the abbreviations $D=h_{20} x_{i}+h_{21} y_{i}+1, N_{x}=h_{00} x_{i}+h_{01} y_{i}+h_{02}$, and $N_{y}=h_{10} x_{i}+h_{11} y_{i}+h_{12}$, we can determine the derivatives as

$$
\begin{align*}
& \frac{\partial e_{i ; x}}{\partial h_{00}}=\frac{\partial e_{i, y}}{\partial h_{10}}=x_{i} / D \quad ; \quad \\
& \frac{\partial e_{i ; y}}{\partial h_{00}}=\frac{\partial e_{i ; x}}{\partial h_{10}}=0 \\
& \frac{\partial e_{i ; y}}{\partial h_{11}}=y_{i} / D ; \quad  \tag{4.16}\\
& \frac{\partial e_{i ; x}}{\partial h_{02}}=\frac{\partial e_{i ; y}}{\partial h_{01}}=\frac{\partial e_{i ; x}}{\partial h_{11}}=0 \\
& \frac{\partial e_{i ; x}}{\partial h_{20}}=-N_{x} x_{i} / D^{2} ; \quad
\end{align*} \quad \frac{\partial e_{i ; y}}{\partial h_{02}}=\frac{\partial e_{i ; x}}{\partial h_{12}}=0, \quad \frac{\partial e_{i ; y}}{\partial h_{20}}=-N_{y} x_{i} / D^{2} .
$$

Based on these derivatives, we obtain the gradient vector and Hessian matrix for each iteration step. The optimization can be started with $\mathbf{H}$ equal

|  | Avg. <br> algebr. | Avg. <br> nonlin. | Avg. <br> RANSAC | Max. <br> algebr. | Max. <br> nonlin. | Max. <br> RANSAC |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
| roma | 0.150 | 0.150 | 0.455 | 0.513 | 0.509 | 1.014 |
| rail | 0.112 | 0.112 | 0.569 | 0.450 | 0.449 | 1.659 |
| opera4 | 0.600 | 0.600 | 1.122 | 3.177 | 3.180 | 3.682 |
| nature2 | 0.176 | 0.176 | 0.541 | 0.795 | 0.803 | 1.710 |

Table 4.1: Motion model error $E_{v}$ for different estimation techniques. Shown are the average and maximum values, computed over the complete sequence. Note that the RANSAC column shows the model error for the drawn sample excluding the reestimation step using all inliers.
to the identity matrix as the initial starting condition. Note that the minimization of $E_{2}$ is considerably more complex than for the linear case, because it is an iterative process and in each iteration, the derivatives of Eq. (4.16) have to be computed and summed over all feature-points.

## Comparison to linear least-squares

To decide if the simplified linear estimation algorithm using algebraic distances can be used instead of the more complex non-linear algorithm, we compared the difference between the estimated motion model and the reference model $\mathbf{H}^{\star}$. Since the reference model is unknown, we instead use the result of our complete motion-estimation system including the parameter refinement from Section 5.2. We assume that these parameters are very accurate, since no alignment errors are visible in the reconstructed sprite image, which is based on these motion parameters.

We quantify the distance between two transforms by transforming a point using both transforms, computing the distance between the two resulting positions and averaging over the image area. More specifically, if $\mathcal{A}$ is the image area and $\mathbf{H}$ and $\mathbf{H}^{\star}$ are the two transforms, we define the transform distance $E_{v}$ as

$$
\begin{equation*}
E_{v}=\frac{1}{|\mathcal{A}|} \iint_{\mathcal{A}} d\left(\mathbf{H} \mathbf{p}, \mathbf{H}^{\star} \mathbf{p}\right) \mathrm{d} x \mathrm{~d} y \tag{4.17}
\end{equation*}
$$

We computed the transform distance $E_{v}$ for the four test sequences and computed the average and the maximum value over all frames. The results are shown in Table 4.1 (the RANSAC column will be discussed later).

It is clearly visible that the results obtained with the algebraic distance do not differ much from the results obtained with the Euclidean distance.


Figure 4.5: Detected feature-correspondences. While the sequence (a) has many features that are well distributed over the frame, only a few features could be found in sequence (b). Moreover, the features are not distributed uniformly over the image area. Consequently, the accuracy of the motion estimation is lower for sequence (b). See also Table 4.1.

Apparently, the noise in the feature location is so small that no difference is observable between both parameter estimation algorithms. We can conclude that the simpler algebraic distance can be used without sacrifying accuracy. ${ }^{1}$

### 4.3 Robust estimation algorithms

As long as we can assume that the only source of errors are inaccuracies in the feature-point positions, the parameters can be determined using a least-squares approximation as described above. Unfortunately, this is only the case for video sequences showing pure camera motion and no independent object motion. In most practical situations, the data is disturbed by gross outliers or it comprises multiple concurrent motions, so that robust

[^7]estimation algorithms have to be applied. The purpose of the robust estimation algorithms is to fit a given function to a set of data points, even if the data is contaminated with a considerable number of outliers.

In this section, we present a robust estimation algorithm that extracts the dominant motion model from the a mixture of different motions. The robust estimation algorithm separates the input data into inliers (part of the dominant motion), and outliers (non-dominant motion or erroneous correspondences). For the estimation of the motion parameters from the inlier data, we apply the parameter-estimation algorithm derived in Section 4.2.

### 4.3.1 Breakdown of least-squares fit on data with outliers

The direct least-squares approach for parameter estimation works well for a small number of outliers that do not deviate too much from the correct motion. However, the result is significantly distorted when the number of outliers is larger, or the motion is very different from the correct camera motion. Especially if the sequence shows independent object motions, a least-squares fit to the complete data would try to include all visible object motions into a single motion model. Obviously, this cannot give a reasonable result.

Figure 4.6 shows an example taken from a sequence with panning camera motion (background moves to the left) and object motion (human walks to the right) at the same time. The result of fitting the model to all correspondences is shown in Fig 4.6(b). This non-sense result presents a motion field which indeed moves to the left at the right part of the picture (where mostly camera motion is visible) and in the other direction at the left side (where a large object is visible). However, this motion field is neither a good representation for the camera motion nor for the object motion.

The solution to this problem is to separate feature-correspondences that originate from different motions and to compute independent motion fields for each set of correspondences. However, this is a chicken-and-egg problem. How can we classify the correspondences into different motion types if the motion fields are unknown, and on the other hand, how can we compute the motion-field parameters, if the sets of consistent feature-correspondences are unknown? This problem is addressed in the following sections.

### 4.3.2 Robust estimation using RANSAC

We consider the following inverse problem. We are given two video frames that contain several areas with different motions. Two motions are considered different if the motions cannot be explained by a single projective motion model. The apparent motion model parameters as well as the seg-

(a) Detected correspondences (outliers found by RANSAC are drawn in white color).

(b) Motion field computed from all correspondences.

(c) Motion field computed using inlier correspondences only.

Figure 4.6: Example from the human sequence. The computed correspondences are shown in (a). They are classified as either inliers (black) or outliers (white) by a RANSAC algorithm. (b) shows the result of fitting a projective motion model on the whole data-set using a least-squares estimation with algebraic distance measure. (c) shows the result of using the same estimation technique, but fitting only to the inlier correspondences.


Figure 4.7: Illustration of multiple motions. Each point represents the motion of one feature-correspondence. Correspondences for different motion models lie on different manifolds.
mentation into differently moving image areas are unknown. The only input is a sparse set of samples of the image motion. The objective is to obtain the model parameters for the dominant motion, i.e., the motion model that has the largest support of input data. In practice, this dominant motion is usually the camera motion.

The main difference to the last section is that we now have a mixture of several motions with unknown parameters. For the one-dimensional case, this is visualized in Figure 4.7. As mentioned earlier, we cannot start with estimating motion parameters for one of the models, since the partitioning into uniform motion areas is still unknown, and we also cannot start with the partitioning until the motion model parameters are known. This deadlock situation can be solved with robust estimation algorithms, of which RANSAC (RANdom SAmple Consensus) [73] is the most prominent one (other approaches [175, 183, 159] are described in Appendix C). The idea is to repeatedly guess a set of model parameters using small subsets of data that are drawn randomly from the input. The hope is to draw a subset with samples that are part of the same motion model. After each subset draw, the motion parameters for this subset are determined and the amount of input data that is consistent with these parameters is counted. The set of model parameters that has the largest support of input data is considered to be the most dominant motion model visible in the image.

## Introductory examples

Let us consider again the previous example of estimating a one-dimensional perspective motion model. Since we have three free parameters, we also need three input correspondences to determine one set of parameters. Con-


Figure 4.8: Two steps of the RANSAC algorithm. A sample set of size three is drawn to compute the parameters of a one-dimensional perspective motion model. All input data that is close to the model computed from the drawn samples is considered as inliers (black dots). Circles mark the outlier data.
sequently, every draw from the input data must contain three samples. From these samples, we can directly calculate the motion parameters. Now, basically two cases are possible. If we are unlucky, the samples will be drawn from different motions (Figure 4.8(a)) and their support of inlier input data (the data which is close to the computed motion model) is small. However, if we draw the samples from a consistent motion (Figure 4.8(b)), the obtained parameter set will have a larger support. To increase the probability of finding a consistent set of samples, we have to repeat the random drawing of subsets several times where the number depends on the fraction of inlier data. Finally, we select the largest set of inliers and assume that it mainly consists of data from only one motion model. Consequently, we can now use a standard least-squares estimation on this inlier data to obtain an accurate parameter set for the motion model.

## RANSAC algorithm

Let us now describe the RANSAC algorithm for the special case of estimating the parameters of a two-dimensional perspective motion model. We denote the set of correspondences, which we use as algorithm input, by $\mathcal{C}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}\right\}$, and we further denote the Euclidean distance between two points $\mathbf{p}_{i}$ and $\mathbf{p}_{k}$ as $d\left(\mathbf{p}_{i}, \mathbf{p}_{k}\right)$. The RANSAC algorithm can then be described with the following steps.

1. Draw a subset $\mathcal{S}$ of size $|\mathcal{S}|=4$ from $\mathcal{C}$. Four correspondences are required to solve for the eight free parameters of the motion model.
2. Compute the parameters $\left\{h_{j k}\right\}$ of the motion model $\mathbf{H}$ from the correspondences in $\mathcal{S}$ using the linear system in Eq. (3.2).
3. Determine the set of inliers $\mathcal{I}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i} \in \mathcal{C} \mid d\left(\hat{\mathbf{p}}_{i}, \mathbf{H} \mathbf{p}_{i}\right)<\epsilon\right\}$ which is the set of correspondences that comply with the motion model. In other words, this means that we use the current set of parameters to transform the features from the first image into the second and compare this with the measured positions. If the distance is low, then the pair of points is assumed to comply with the motion model, and it is selected as an inlier.
4. Repeat Steps 1-3 several times $(N)$ and choose the set of inliers for which $|\mathcal{I}|$ is largest.
5. Perform a least-squares approximation of the motion parameters with the set of inliers as described in Equation (4.10). The solution is the result of the RANSAC algorithm.

The RANSAC algorithm has two parameters that have to be chosen initially: the number of draws $N$ and the inlier threshold $\epsilon$. A good value for the inlier threshold can be obtained from the evaluation of the feature-point detector. The more accurate it can locate the features, the smaller $\epsilon$ can be chosen. Section 3.2.5 showed that the number of found correspondences by increasing $\epsilon$ saturates very quickly. Hence, we have chosen a small value around 1.5 for $\epsilon$, but the right selection of $\epsilon$ is not critical. If it is chosen too low, some correct correspondences will be sorted out as outliers, but usually the set of inliers is still large enough to estimate accurate model parameters. If it is chosen too high, some outlier data will be included, but since these outliers cannot differ much from the inliers (their error is below $\epsilon$ ), their influence in the least-squares approximation will be limited.

The required number of draws $N$ primarily depends on the percentage of outliers $p_{o}$ we expect in the input, and it also depends on the maximum probability for algorithm failure that is acceptable. This probability $P$ that the RANSAC algorithm will fail computes as

$$
\begin{equation*}
P\left(p_{o},|\mathcal{S}|, N\right)=(\underbrace{1-\underbrace{\text { percentage of inliers }}_{\text {probability to draw set with at least one outlier }} \underbrace{1-p_{o}})^{|\mathcal{S}|}}_{\text {probability to get only outlier sets after all draws }})^{N}, \tag{4.18}
\end{equation*}
$$

where $|\mathcal{S}|$ is the size of the subset to be drawn (four in our case). By fixing a probability $P(o,|\mathcal{S}|, N)$ of algorithm failure, we can compute the
required number of draws $N$. Clearly, we can always increase $N$ to be more robust, however, this will also increase the required computation time. Let us assume as an example that we have an outlier percentage of $30 \%$, then only 20 draws would be enough to reduce the probability of algorithm failure to 0.004 . Since the inner loop of the RANSAC algorithm is not very computationally expensive, we can even choose a larger number of draws, like 50. Section 3.3.1 discussed that by using motion prediction, the correspondences will lock to the camera motion and fewer correspondences will be generated for foreground objects. This favourable effect can reduce the number of outliers beforehand, so that the typical percentage of outliers is even lower than in the example.

### 4.3.3 Robustness of the RANSAC algorithm

The RANSAC algorithm is a probabilistic technique that is not always successful. However, by increasing the number of draws, the probability of failure can be reduced to arbitrarily small values. Using Eq. (4.18) implies that the number of draws $N$ depends on the fraction of outliers $p_{o}$, the sample subset size $|\mathcal{S}|$ and the maximum allowed probability of failure $P$. For simplicity, we will abbreviate the probability that a non-fitting subset is drawn by $p_{f}=1-\left(1-p_{o}\right)^{|\mathcal{S}|}$ during the following discussion. Consequently, to achieve a maximum error rate of not more than $P$, we need at least $N=\log P / \log p_{f}$ draws.

## Robustness against outliers

To validate the theoretical derivation of the probability of success, we generated synthetic input data, consisting of a fraction $p_{i}$ of inlier correspondences (motion vectors) that were consistent with a given motion model. Furthermore, this set of data was contaminated with a fraction $p_{n}$ of random motion vectors, and a fraction $p_{2}$ of object motion vectors that are consistent with a second motion model. In total, this gives an outlier fraction of $p_{o}=p_{2}+p_{n}$. We carried out a large number of random draws and compared the obtained motion model with the predefined inlier model, which gave us a measured probability $p_{f}^{\prime}$ to draw a non-fitting subset. The obtained $p_{f}^{\prime}$ was very close to the theoretical value $p_{f}$. The result did not depend on the type of outlier (noise or secondary motion model). RANSAC could also successfully find the correct motion model if $p_{m}>50 \%$. However, it should be noted that the fraction of secondary motion data must be smaller than the fraction of inliers $\left(p_{2}<p_{i}\right)$, since otherwise the second motion model is the dominant one.

## Difference between theory and practice

In a second set of experiments, we measured the robustness of the RANSAC algorithm for noisy real-world data. We selected sequences for which the correct motion model $\mathbf{H}^{\star}$ was previously computed using our complete motion-estimation system. We computed feature-correspondences and classified them into inliers $\mathcal{I}^{\star}$ and outliers based on the precomputed accurate motion model. This gave us the fraction of outliers $p_{o}$ in our input data. Afterwards, the RANSAC algorithm was executed with a large number of subset draws. For each subset (not only for the best one), the refined motion model was computed as described in Step 5 of the RANSAC algorithm and another set of inliers $\mathcal{I}_{R}$ was determined based on the refined motion model. If this set of inliers was equal for more than $90 \%$ to the set of inliers $\mathcal{I}^{\star}$ obtained with the accurate motion model, the computed motion model was considered to be correct. Note that a direct comparison between motion models is not possible because of small differences in the parameters. The fraction $p_{f}^{\prime}$ of incorrect motion models that we obtained in the simulation should approximately equal the theoretical fraction $p_{f}$. However, we noticed that the actual probability to draw a non-fitting subset is much higher (see Table 4.2; compare columns theory vs. refinement steps=1). As a consequence, an inaccurate motion model is often computed even if all of the four correspondences in our subset are inliers. This effect will now be further analyzed.

## Dependency of the failure probability on the sample distances

In order to find the reason for this degraded performance, we marked the randomly drawn subset and the obtained set of inliers in the input image (see Fig. $4.12(\mathrm{~b})$ ). It can be seen that the inliers are spatially concentrated with an almost clear border to the area with outliers. Moreover, it can also be verified that the inlier area is larger if the points from the drawn subset are spatially distant (Fig. 4.12(a)).

The reason for this behaviour are numerical instabilities that can be easily visualized in the simpler one-dimensional affine case (Fig. 4.9). In this case, a linear model is computed through two sample points. However, the position of the sample points is distorted by some noise. This uncertainty of the sample positions has a higher influence on the obtained model parameters if the samples have a smaller distance. In our one-dimensional case, this means that the slope of the model line will be inaccurate and only a few points near the two samples will be classified as inliers.

To validate this explanation, we have further analyzed the dependency between the probability of having found a successful set of parameters and


Figure 4.9: RANSAC for a linear estimation problem. Even though both selected points are inliers, the model defined by these points differs much from the optimum model. Inaccuracies in the point positions have a large influence on the model if the points are close together.
the distance between the samples from which the parameters were derived. In order to show this dependency, we computed the total sample distance

$$
\begin{equation*}
d_{s}=\frac{1}{2} \sum_{i, k \in\{1,2,3,4\}} d\left(\mathbf{p}_{s_{i}}, \mathbf{p}_{s_{k}}\right) \tag{4.19}
\end{equation*}
$$

for each selected subset $\left\{s_{1}, s_{2}, s_{3}, s_{4}\right\}$ and plotted the measured probability of failure $p_{f}^{\prime}$ depending on $d_{s}$. It can be observed (Fig. 4.10) that the probability of failure indeed decreases with larger sample distances. On the other hand, the estimation will almost certainly fail if the distances are very small.

## Improving RANSAC by equalizing the sample distribution

One possible solution (even though described for the related problem of computing a fundamental matrix) has been proposed in [201]. The idea is to disable the selection of samples which are too close by dividing the image into a grid of rectangular buckets, similar to the technique described in Section 3.3.1. Random samples are now obtained in two steps. First, a bucket is randomly selected, followed by a random selection of a featurepoint within this bucket. Since the number of points in the buckets are unequal, the selection is weighted by the number of points. To get spatially distant samples, a bucket may only be chosen once in each iteration.

We do not follow this technique, since it favours the selection of distant feature-points, but it cannot prevent that the position of the sample set is degenerated. For example, all feature-points could lie in one line.


Figure 4.10: Probability of generating an inaccurate motion model depending on the distance between the samples in the drawn subset. The opera 4 sequence is not included since the number of features is very low and unequally distributed.

## Improving RANSAC using iterative model refinement

As an alternative solution, we propose to keep the original random sample selection strategy, but to carry out the motion-parameter refinement (Step 5) of the RANSAC algorithm several times. The idea is that the initially obtained motion model is not always accurate, but it still includes a considerable number of inliers. Each time the model parameters are adapted to the newly obtained set of inliers, the number of inliers will increase.

A sample result is shown in Figure $4.12(\mathrm{~b})-(\mathrm{d})$, where the set of inliers after each of the refinement steps is marked. It is clearly visible that the area of inliers grows with each refinement step. The measured probabilities of failure $p_{f}^{\prime}$ for the improved algorithm are shown in Figure 4.11 and Table 4.2. It is interesting to note that for a larger number of refinement steps $(\geq 3)$, the measured probabilities of failure are even below the theoretical value. The reason for this is that some of the outlier correspondences are very close to being classified as inliers. Consequently, even if one of these almost-inliers is selected, the motion model still converges to the correct model.

Because each additional refinement step might improve the final motion model, the probability of failure $p_{f}^{\prime}$ decreases which also means that


Figure 4.11: Probability of generating an inaccurate motion model depending on the distance between the four samples in the drawn subset. The values are based on the rail sequence. The probability is drawn for different numbers of model refinement steps (the original RANSAC uses a single step). Also shown is the distribution of the sample distances as they were drawn randomly from the image.
the number of required subset draws $N$ can be reduced. On the other hand, each refinement step requires some additional computation time. Hence, the question arises what the optimum number of refinement steps is. Since the most computational intensive step in the RANSAC algorithm is the separation of the samples into inliers and outliers, we count the total required computation time in units of these classification steps to be performed. If we denote the number of refinement steps as $R$, we get the total computation time $C$ as

$$
\begin{equation*}
C=(R+1) \cdot\left\lceil\log P / \log p_{f}^{\prime}\right\rceil . \tag{4.20}
\end{equation*}
$$

After conducting experiments on several test-sequences, we could see that three refinement steps resulted in the lowest computation time. Since the probability of failure $p_{f}^{\prime}$ for three refinement steps is usually close to or smaller than the theoretically determined value $p_{f}$, we can use the theoretically computed number of iterations.


Figure 4.12: Examples for obtained sets of inliers (black color). (a) A good sample set provides an accurate motion model. (b)-(d) A nonfitting sample gives inaccurate motion parameters, but they can be improved by additional refinement steps.

### 4.4 Summary

This chapter described the second half of the feature-based camera-motion estimation. Whereas the previous chapter presented the computation of feature-point correspondences, the current chapter explored the estimation of motion parameters from feature-point correspondences.

First, we considered the parameter estimation for scenes in which only camera motion is present. We found that a simple linear algorithm can be used for affine motion models, but that non-linear optimization is required for the projective motion model. However, a comparison between the non-

| Failure risk |  | Refinement steps |  |  |  |  |  |
| :--- | ---: | ---: | ---: | ---: | ---: | ---: | ---: |
| $P=0.1 \%$ | Theory | 1 | 2 | 3 | 4 | 5 |  |
| rail | $p_{f}^{\prime}$ | $\mathbf{4 2 . 7 \%}$ | $\mathbf{6 6 . 9 \%}$ | $46.5 \%$ | $36.9 \%$ | $22.7 \%$ | $17.9 \%$ |
| $p_{o}=13 \%$ | $N$ | 8 | 18 | 9 | 7 | 5 | 4 |
| roma | $p_{f}^{\prime}$ | $\mathbf{1 4 . 6 \%}$ | $\mathbf{3 1 . 6 \%}$ | $13.7 \%$ | $7.9 \%$ | $6.1 \%$ | $4.7 \%$ |
| $p_{o}=4 \%$ | $N$ | 4 | 6 | 4 | 3 | 3 | 3 |
| opera4 | $p_{f}^{\prime}$ | $\mathbf{5 4 . 1 \%}$ | $\mathbf{6 2 . 3 \%}$ | $48.8 \%$ | $42.0 \%$ | $38.5 \%$ | $36.3 \%$ |
| $p_{o}=18 \%$ | $N$ | 12 | 15 | 10 | 8 | 8 | 7 |
| nature2 | $p_{f}^{\prime}$ | $\mathbf{3 1 . 7 \%}$ | $\mathbf{5 5 . 3 \%}$ | $32.8 \%$ | $21.8 \%$ | $17.0 \%$ | $14.8 \%$ |
| $p_{o}=9 \%$ | $N$ | 6 | 12 | 7 | 5 | 4 | 4 |

Table 4.2: Probability of degenerated subset draws for different number of refinement steps. The original RANSAC algorithm corresponds to refinement steps $=1$. Also shown is the number of draws $N$ that are needed to reach an algorithm failure rate below $0.1 \%$.
linear parameter estimation and a linear approximation showed that the accuracy of the linear-approximation algorithm is comparable.

Afterwards, we extended the algorithm to differentiate between foreground motion and background motion, such that the camera-motion parameters can also be estimated even when the camera motion is mixed with object motion. We applied the RANSAC algorithm ${ }^{2}$ to detect the dominant motion and to compute its model parameters. The RANSAC algorithm is a probabilistic algorithm that succeeds only with a certain probability, which can be increased arbitrarily by carrying out more program iterations. However, experiments showed that the probability of failure was larger than predicted by a theoretical analysis. It was found that the reason for the reduced performance are degenerate sets of samples, which lead to numerical instabilities in the parameter estimation. We addressed this problem by re-estimating the parameters based on the obtained inlier and then recomputing the set of inliers for a small number of iterations. This increases the set of inliers in each iteration such that the parameter estimation is based on more input data, resulting in a more accurate estimation. With this modification to the RANSAC algorithm, we could increase the probability of success to reach or even exceed the theoretically predicted performance.

[^8]

Figure 4.13: Estimation of motion parameters based on a RANSAC algorithm. Depicted is only one iteration. The algorithm is repeated several times and the solution with the largest number of inliers is selected.

## Resulting algorithm flow-graph

The data-flow of the motion-parameter estimation is depicted in Figure 4.13. It shows the RANSAC algorithm with an additional loop for the refinement steps. The algorithm input is formed by the feature-point correspondences that are extracted in the previous step (see Chapter 3). Four random samples are selected and a candidate motion model is computed from these samples. All input correspondences are compared with this motion model to separate them into an inlier set and the outliers. After this, refined motion parameters are computed with a least-squares approximation on all inliers. The last two steps, selection of inliers, and least-squares approximation, is repeated three times to converge to a maximum coverage of feature-points. This whole process is repeated several times, and only the motion model that had the largest number of inliers is returned as result.

## Experimental results

The camera-parameter estimator has been tested on many sequences that were either recorded from regular DVB broadcasts, or recorded with a camcorder. Additionally we also used some standard test sequences. The algorithm proved to be very robust on most sequences. Problems only arose if the video contained too few features in the background, or if it had a very low contrast such that the feature-point extraction could not find good features to track. In most cases, errors in the feature-based motion estimator could be corrected in the direct motion estimator that will be
explained in the next chapter.
Some example results, in which both background and foreground motion are present, are depicted in Figure 4.14. In Figures 4.14(a) to (d), surveillance-type scenes were recorded with a hand-held camera. In a real application, this could be a remotely controlled pan-tilt-zoom camera in a surveillance system. Finally, Figures $4.14(e)$ and (f) show two scenes of the stefan test sequence, one with a slow camera motion and one with a very fast camera pan.

For the experiments, we selected the Harris algorithm to detect features, the search-range of the feature-matching algorithm was set to 16 pixels around the predicted feature position, and the RANSAC algorithm used 25 iterations with 3 refinement steps. All algorithm parameters were fixed for all sequences. The pictures show the inlier (background motion) vectors in black color and the outliers (foreground motion and erroneous vectors) in white color.

The examples show that foreground motion and background motion are well separated. In addition to the foreground motion, a small number of outliers can be observed that result from bad feature-correspondences. An interesting effect is visible in Fig. 4.14(f): the foreground object contains almost no features. The reason is that the feature-correspondence algorithm only searches for matching correspondences in a small neighborhood around the predicted feature position. Since the feature positions are predicted with the camera motion parameters, the predicted position is far away from the object motion. Consequently, the algorithm does not find the correspondences for the object motion. For our application, this is an advantage because the number of outliers in the input for the RANSAC algorithm is decreased.


Figure 4.14: Inliers (black) and outliers (white) as detected by the RANSAC algorithm for different scenes with foreground objects. See Section 4.4 for more details about (f).

It's not an optical illusion, it just looks like one. (Phil White)

## Chapter 5

## Background Reconstruction

If a video sequence is recorded with a rotational camera, it is possible to reconstruct an image of the scene background. In the context of our videoobject segmentation system, this synthetic background image can be used for two purposes. First, the background images enable an easy segmentation algorithm that detects the foreground objects based on the differences to this background image. Second, the background image can be used together with the MPEG-4 sprite coding tools to transmit the background content of a video scene with a very low bit-rate. This section describes the background reconstruction process, involving the following main tasks. At the start, the accuracy of the camera-motion parameters is increased such that long-term consistency is achieved. After that, the input frames are combined into the background image such that moving foreground objects are removed. We briefly review some of the existing approaches and present a new algorithm for background estimation which is designed for difficult reconstruction cases in sequences where the background is only visible for a short time period.

Chapter 5. Background Reconstruction

### 5.1 Introduction

The segmentation algorithm that will be described in Chapter 7 requires a pure background image of the captured scene. When the scene background is available, the segmentation can be obtained by detecting changes between the background image and the input images. For some applications, the background image can be captured manually. One example are indoor surveillance systems that observe a non changing scene. However, in other cases, the background image should be synthesized automatically for various reasons. First, it can be impractical to wait for a moment when there is no foreground object visible, like when observing a busy highway. Second, the background may change slowly during the day, e.g., because of changing light conditions. Finally, automatic video segmentation for content analysis may operate on many small sequences for which it is tedious or impossible to generate background images by hand. Because of these reasons, algorithms that synthesize backgrounds automatically become important.

The basic principle of these algorithms can be explained using Figure 5.1. The figure shows a slice through the 3-D video volume, comprising the spatial $x, y$ axes and the time $t$. Since the sequence was recorded with a static camera, background pixels do not change over time. However, during some time periods, the background image content can be occluded by foreground objects. Under the assumption that the background is visible for a longer time than the foreground, we can apply some averaging method to filter out the time periods showing foreground content and identify the background content.

In practice, it is not always true that the background content is visible during most of the time like in Figure 5.1(b). A much more difficult scene is shown in Figure 5.2, in which some parts of the background are only visible during a very short time, or they are not visible at all. The second part of this chapter will present a new algorithm that is more robust than previous algorithms in such difficult cases.

In the first part of this chapter, we study camera-motion compensation. Whereas background estimation for static cameras can simply observe pixels at constant position during time, this is not possible for moving cameras. As is visualized in Figure 5.3, a moving camera leads to trajectories through $x-y-t$ space that should be followed for each background pixel. This is achieved by aligning all input frames to a common reference frame, which is at the same time the reference for the background image to be generated.

In the previous chapters, we described the estimation of camera motion, but unfornately, the obtained motion parameters cannot be used directly for the camera compensation because of two reasons. First, the motion


Figure 5.1: For a static camera, the background does not change along the $t$-axis. However, foreground objects appear for some time and occlude the background.


Figure 5.2: Two $x$-t slices for an especially difficult sequence with many foreground objects. See Figure 5.21(a) for an example input frame.

(a) Cut through a 3-D video volume for a sequence captured with a moving camera.

(b) An $x-t$ slice of the 3-D video volume.

Figure 5.3: Same visualization as in Fig. 5.1, but now for a moving camera. With a moving camera, the background pixels are not aligned in temporal direction.


Figure 5.4: The feature-based motion estimator described in the previous chapters gives us short-term motion parameters between successive frames. To combine all the images into a single background frame, long-term motion parameters are required.
parameters describe the motion between of successive frames (short-term motion estimation) instead of being relative to a common background reference frame (long-term motion estimation). The difference is shown in Figure 5.4. Second, a simple chaining of short-term transforms to obtain long-term motion parameters would lead to an accumulation of estimation errors. Hence, we carry out a parameter refinement step that computes accurate long-term motion parameters.

### 5.2 Frame alignment

The first step in synthesizing a background image is to determine the accurate placement of the input frames in the background image. With the feature-based motion estimator described in the last chapter, we computed the transforms between successive input frames. These inter-image transforms, we have to derive the transformation from each of the input images to a common, virtual background image. Even though we can obtain the motion parameters with a very high accuracy of about 0.15 pixels (Table 4.1), small errors in these parameters will accumulate when we combine these transforms. These errors will show as blurring when several images are averaged. In the worst case, the alignment errors can lead to discontinuities along the boundaries of images that are combined together. Consequently, it is required to further refine the motion parameters by aligning each input image to the constructed background image with high accuracy. We commence with applying a direct estimation method [92] to obtain the motion parameters. This technique does not rely on detected feature-points and hence, in combination with the feature-based estimator, it can increase the robustness in those cases where the feature-based method yields only low accuracy because the number of features is low. The direct method requires a good initialization of the parameters, which can be obtained from the previous feature-based estimator. This combination of both estimation algorithms brings together the advantages of both. While the feature-based estimator supports fast camera motion, the direct estimation method preserves long-term consistency due to its higher accuracy.

### 5.2.1 Motion models for sprite generation

The MPEG-4 sprite coding tools allow to choose between four different motion models of which the projective model is the most general. In MPEG4, a slightly different (but equivalent) parameterization of the projective model with four motion-vectors at the image corners is used. The other models are affine (where only three motion-vectors are transmitted), a rota-
tion/scale/translation model, which only requires two motion-vectors, and a translatorial model using only one vector. Each of these models is actually a superset of those with less motion-vectors.

As discussed in Section 2.5.4, all images recorded with a rotating camera can be aligned into a common reference image by applying the projective motion model with eight parameters. The parameter estimation of the projective motion model is often considered difficult since it is non-linear. For this reason, various authors, e.g., $[97,24]$ propose to approximate it with an affine model comprising six parameters. However, the affine model is only valid for an orthographic projection, i.e., for the case of very large focal length (see Section 2.4.3). In [172, 122], the bilinear model (eight parameters) or the biquadratic model (twelve parameters) have been proposed. But neither of these models is capable to describe a rotational camera motion, so that they are only applicable as an approximation to camera motion when the rotation angle is very small.

### 5.2.2 Geometry of background image generation

Let us recall the physical image-formation process for a rotation-only camera. The restriction to a rotating and zooming camera is necessary because with translatorial camera motion, the parallax effect leads to different speeds for objects at different depths. This would make it impossible to align the background images into a seamless mosaic ${ }^{1}$.

Let us define the 3-D world coordinate system such that the camera is located at its origin (Figure 5.5). The camera captures a number of images $I_{i}$ with different rotations $\mathbf{R}_{i}$ and focal lengths $f_{i}$. In a rotated local image coordinate-system, where the frontal viewing direction is along the positive $z$-axis, the corresponding 3 -D position of each image pixel $(\hat{x}, \hat{y})$ in the focal plane is $\left(\hat{x}, \hat{y}, f_{i}\right)^{\top}$. For simplicity of notation, we assume that the origin of image coordinates is at the principal point, which can usually be assumed to be at the center of the image. Now let the virtual sprite-plane be placed orthogonal to the $z$-axis of the world coordinate system at a distance $f_{s}$.

[^9]The projection of the image point $(\hat{x}, \hat{y})$ can then be determined by

$$
\begin{align*}
\left(\begin{array}{c}
x^{\prime} \\
y^{\prime} \\
w^{\prime}
\end{array}\right) & =\underbrace{\left(\begin{array}{ccc}
f_{s} & 0 & o_{x} \\
0 & f_{s} & o_{y} \\
0 & 0 & 1
\end{array}\right)}_{\begin{array}{c}
\text { intrinsic cam- } \\
\text { era parameters: }
\end{array}} \underbrace{\substack{r_{00} \\
r_{22} \\
r_{01}}}_{\begin{array}{c}
\text { exterinsic } \\
\text { etetation): } \mathbf{R}_{i} \\
\text { param- } \\
\mathbf{K}_{20}
\end{array} r_{21}} \begin{array}{rl}
r_{02} \\
r_{21} & r_{12}
\end{array})
\end{align*}\left(\begin{array}{c}
\hat{x}  \tag{5.1}\\
\hat{y} \\
f_{i}
\end{array}\right)
$$

where ( $o_{x}, o_{y}$ ) is the position of the principal point on the sprite plane, and the resulting sprite coordinates $\left(x^{\prime}, y^{\prime}, w^{\prime}\right)$ are given in homogeneous coordinates. Multiplying the intrinsic and extrinsic transformation matrices together, we obtain the combined matrix $\mathbf{H}_{i}$, describing the projection of the image coordinates from frame $i$ onto the background plane. Taking the image-to-background transformations $\mathbf{H}_{i}, \mathbf{H}_{k}$ for two images $i, k$, we can obtain the transformation from image $k$ to $i$ by first mapping the point of image $k$ onto the background and then mapping it back onto image $i$. We denote this inter-image transform as $\mathbf{H}_{i ; k}=\mathbf{H}_{i}^{-1} \mathbf{H}_{k}$. The motion $\mathbf{H}_{i+1, i}$ between successive frames $i$ and $i+1$ is known, since this is the output of our feature-based motion estimator. However, it is not yet possible to obtain the image-to-background transform from the inter-image transforms, because the location of the background plane has not been fixed. Basically, the background plane can be placed at an arbitrary position (see Fig. 5.6), but we will see in Chapter 6 that the placement of the background plane still has some important practical consequences. However, in this section, we will simply choose one arbitrary input frame as the reference frame $r$, and use it as the background image reference by setting the transformation $\mathbf{H}_{r}$ to the identity matrix: $\mathbf{H}_{r}=\mathbf{I}$. Since this fixes the relationship between the input frames and the background frame, we can obtain the remaining background transforms $\mathbf{H}_{i}$ by a suitable concatenation of known transforms. Let, for example, frame 4 be the reference frame, then we can set $\mathbf{H}_{4}=\mathbf{I}$ and thus get $\mathbf{H}_{1}$ as

$$
\begin{equation*}
\mathbf{H}_{4} \mathbf{H}_{4,3} \mathbf{H}_{3,2} \mathbf{H}_{2,1}=\mathbf{H}_{4,3} \mathbf{H}_{3,2} \mathbf{H}_{2,1}=\mathbf{H}_{1}, \tag{5.2}
\end{equation*}
$$

while we have to take inverse transforms for frames $i>r$. For, e.g., frame 7, we get

$$
\begin{equation*}
\mathbf{H}_{7}=\mathbf{H}_{4} \mathbf{H}_{5,4}^{-1} \mathbf{H}_{6,5}^{-1} \mathbf{H}_{7,6}^{-1} . \tag{5.3}
\end{equation*}
$$



Figure 5.5: The rotating camera is located at the origin of the world coordinate system. The sprite plane is assumed to be orthogonal to the z-axis. Input images are at a distance to the origin that is equal to the focal length when the image was taken. A point $(\hat{x}, \hat{y})$ on the image is projected onto the sprite position $\left(x^{\prime} / w^{\prime}, y^{\prime} / w^{\prime}\right)$.

One example result of aligning several frames from the stefan sequence into a reference frame is presented in Figure 5.7.

### 5.2.3 Long-term motion estimation

The computation of the image-to-mosaic transforms by concatenating interimage transforms is subject to a practical problem. The parameters that we obtain from the feature-based motion estimator have small inaccuracies that are negligible when only pairs of images are considered, but which can accumulate when many transforms are concatenated. As a consequence, images that are temporally far apart will not fit together in a composed background image. For a straight camera pan, this effect is almost invisible, since only successive images overlap. However, if the camera motion returns to a previous position after some time, the accumulated errors might be well visible (see Figure 5.8(a)).

To prevent this accumulation of parameter errors, we refine the image-to-mosaic transforms with a high-accuracy motion estimation algorithm, where the concatenation of inter-image transforms is used to initialize the


Figure 5.6: The placement of the background plane can be chosen arbitrarily, because an image on each of these planes will look identical, seen from the optical center.
optimization.

## Accurate alignment with direct estimation methods

In contrast to the feature-based motion estimation algorithms that estimate the parameters from a set of corresponding point features, direct methods estimate these parameters directly by minimizing the motion-compensated residual image. This makes use of the brightness-constancy constraint, which states that the brightness of pixels does not change during motion. In other words, this means that if $\overline{\mathbf{H}}_{t+1, t}$ is the true image motion, then $\left|I_{t+1}\left(\overline{\mathbf{H}}_{i} \mathbf{p}\right)-I_{t}(\mathbf{p})\right|=0$.

Let us denote the background image as $I_{B}(x, y)$ and the current frame $t$ as $I_{t}(x, y)$. In the direct motion estimation algorithm, the refined motion parameters $\breve{\mathbf{H}}_{t}$ are computed by minimizing the motion-compensated image difference

$$
\begin{equation*}
\breve{\mathbf{H}}_{t}=\arg \min _{\mathbf{H}_{t}} \sum_{\mathbf{p} \in \mathcal{A}}\left|I_{B}\left(\mathbf{H}_{t} \mathbf{p}\right)-I_{t}(\mathbf{p})\right|^{2}, \tag{5.4}
\end{equation*}
$$

where $\mathcal{A}$ denotes the complete image area of $I_{t}$. Since the number of parameters is large and the function to be minimized is non-linear, an iterative gradient-descent algorithm has to be applied to find a solution. A gradientdescent algorithm does not guarantee to find the global optimum, since it usually converges to a local optimum near the initialization. Hence, it is important to start with a good initialization, which we can in our case get from the motion parameters of the feature-based motion estimator.

Practically, we cannot use Eq. (5.4) in the presented form, because of three reasons. First, during the construction process, the complete background image is not available yet, which means that some of the trans-


Figure 5.7: Alignment of input frames into a common reference frame. The inter-image transforms $\mathbf{H}_{i+1, i}$ are obtained from the feature-based motion estimator. The input-to-mosaic transforms $\mathbf{H}_{i}$ can be obtained by a concatenation of inter-image transforms as soon as one reference transform $\mathbf{H}_{r}$ has been defined.
formed pixels $\mathbf{H}_{t} \mathbf{p}$ will fall onto pixels which are still undefined. It is not sufficient to set all undefined pixels in $I_{B}$ to an arbitrary color value, since this can cause large matching errors if the input image $I_{t}$ has a different color. As a consequence, the optimization would try to squeeze $I_{t}$ onto the already defined area of $I_{B}$ as much as possible. A solution is to introduce a special background pixel value that denotes transparent pixels and set the matching error between any value and transparent pixels to zero. Of course, this also means that we get the minimum matching error if both images do not overlap at all, but since the optimization algorithm locks to the nearest local minimum, this unfavourable solution is not reached. ${ }^{2}$

A second difficulty is that our input sequences usually do not show pure

[^10]
(b) Alignment using long-term motion parameters. The long-term parameters were determined as a refinement of the transform between the image and the mosaic.

Figure 5.8: Alignment quality using short-term or long-term motion parameters. Only every $12^{\text {th }}$ input image has been included in the mosaic to make the misalignments more visible.
background images, but they also contain foreground objects. These can create large matching errors which would dominate the optimization and therefore bias the estimation towards an inaccurate solution. The solution for this is to use an M-estimator instead of the squared error to limit the effect of non-matching areas. We use a saturated squared-error function which does not increase if the pixel difference exceeds a threshold $\tau$. Other

(a) Ignore outliers.

(b) Limit error to maximum.

Figure 5.9: Robust M-estimators.
authors [171, 39] propose to use a clipped squared-error function, which drops to zero if the difference exceeds the threshold (Fig.5.9(a)). However, we think that this can have the unfavourable effect that a bad match can result in a low matching error, especially with high-contrast backgrounds.

The third problem of Eq. (5.4) is that the transformed pixel positions $\mathbf{H}_{t} \mathbf{p}$ will usually not be integer positions, whereas the background image is discretized. Here, it is important that the transformed positions are not just rounded to the nearest-neighbor pixel, since this would degrade the sub-pixel accuracy of the solution. We applied a bi-linear interpolation ${ }^{3}$ on $I_{B}$ to obtain values also at sub-pixel locations.

Also note that the total matching error is not normalized to the mapped image size. This means that a smaller image would lead to less error. However, adding the normalization would make the equations for the optimization process far more complex. Fortunately, the behaviour of gradientdescent algorithms to lock to a local minimum is again to our benefit, since it prevents that the projected area collapses to a small size (Fig. 5.10).

Introducing the above-mentioned considerations into a modified cost function, we obtain the modified optimization equation

$$
\begin{equation*}
\breve{\mathbf{H}}_{t}=\arg \min _{\mathbf{H}_{t}} \sum_{\mathbf{p} \in \mathcal{A}} \rho\left(I_{B}\left(\mathbf{H}_{t} \mathbf{p}\right), I_{t}(\mathbf{p})\right), \tag{5.5}
\end{equation*}
$$

where $I_{B}\left(\mathbf{H}_{i} \mathbf{p}\right)$ is evaluated using bilinear interpolation and the M-estimator

[^11]

Figure 5.10: Cost function of an optimization problem. Often, cost functions are defined such that the global minimum corresponds to a degenerate solution. For example, in our motion-estimation problem, we minimize the luminance difference in the overlapping image region. Consequently, the global solution would be to place the two images beneath each other, resulting in zero cost since there is no overlapping area. In these cases, it is an advantage to reach the nearest local optimum instead of the global solution.
is defined as the clipped squared error

$$
\rho\left(y_{B}, y_{I}\right)= \begin{cases}0 & \text { if } y_{B} \text { is transparent }  \tag{5.6}\\ \left(y_{B}-y_{I}\right)^{2} & \text { if }\left|y_{B}-y_{I}\right|<\tau \\ \tau^{2} & \text { else }\end{cases}
$$

## Parameter optimization

Gradient descent algorithms optimize the parameters with a number of iterations. Note that to evaluate the cost function, we have to transform the input image using a projective transform with bilinear interpolation and compare the result with the background image. Since this computationintensive calculation has to be carried out in each iteration, it is important to keep the number of iterations low. For this reason, we apply the Levenberg-Marquardt minimization algorithm which is more complex than a steepest descent, but which is known to converge in a small number of iterations. We do not describe the Levenberg-Marquardt algorithm here (refer to [151, 179] for an in-depth description), but we show how to apply it to our motion estimation problem.

Let us write the transform parameters in vector form as $\boldsymbol{\theta}=\left(h_{00}, \ldots, h_{21}\right)$. For the optimization process, the algorithm requires the gradient vector of
the cost function

$$
\begin{equation*}
E_{\boldsymbol{\theta}}=\sum_{\mathbf{p} \in \mathcal{A}} \rho\left(I_{B}\left(\mathbf{H}_{t} \mathbf{p}\right), I_{t}(\mathbf{p})\right) \tag{5.7}
\end{equation*}
$$

with respect to the optimization parameters, i.e., $\nabla E=\partial E_{\boldsymbol{\theta}} / \partial \boldsymbol{\theta}$ and the Hessian matrix $\nabla^{2} E$. To get the expressions for the gradient vector, we substitute the motion model Eq. (2.10) into Eq. (5.5). Let us for the ease of notation abbreviate the residual for a single pixel $k$ as

$$
\begin{equation*}
e_{k}=I_{B}\left(\mathbf{H}_{t} \mathbf{p}_{k}\right)-I_{t}\left(\mathbf{p}_{k}\right)=I_{B}\left(x^{\prime}, y^{\prime}\right)-I_{t}(x, y) \tag{5.8}
\end{equation*}
$$

Then we can obtain the derivatives for the case where $\rho\left(y_{I}, y_{B}\right)=\left|y_{I}-y_{B}\right|^{2}$ as

$$
\begin{equation*}
\nabla E=2 \sum_{k} e_{k} \cdot\left(\frac{\partial e_{k}}{\partial \theta_{1}}, \cdots, \frac{\partial e_{k}}{\partial \theta_{8}}\right) \tag{5.9}
\end{equation*}
$$

To compute the derivatives of $e_{k}$, we apply the chain rule to get, for example,

$$
\begin{equation*}
\frac{\partial e_{k}}{\partial \theta_{1}}=\frac{\partial e_{k}}{\partial h_{00}}=\frac{\partial I_{B}}{\partial x^{\prime}} \frac{\partial x^{\prime}}{\partial h_{00}}=\frac{\partial I_{B}}{\partial x^{\prime}} \frac{x}{D} \tag{5.10}
\end{equation*}
$$

or

$$
\begin{equation*}
\frac{\partial e_{k}}{\partial \theta_{7}}=\frac{\partial e_{k}}{\partial h_{20}}=\frac{\partial I_{B}}{\partial x^{\prime}} \frac{\partial x^{\prime}}{\partial h_{20}}+\frac{\partial I_{B}}{\partial y^{\prime}} \frac{\partial y^{\prime}}{\partial h_{20}}=-\frac{y}{D} \cdot\left(x^{\prime} \frac{\partial I_{B}}{\partial x^{\prime}}+y^{\prime} \frac{\partial I_{B}}{\partial y^{\prime}}\right) \tag{5.11}
\end{equation*}
$$

with the abbreviation $D=h_{20} x+h_{21} y+1$.
For the computation of the Hessian matrix, we start with Eq. (5.9), which we derive a second time. To simplify the result, we follow the suggestion in [151] to ignore the second-order derivatives of $e_{k}^{2}$

$$
\begin{equation*}
\frac{\partial^{2} e_{k}^{2}}{\partial \theta_{m} \cdot \partial \theta_{n}}=2(\frac{\partial e_{k}}{\partial \theta_{m}} \cdot \frac{\partial e_{k}}{\partial \theta_{n}}+\underbrace{e_{k} \frac{\partial^{2} e_{k}}{\partial \theta_{m} \cdot \partial \theta_{n}}}_{\text {ignore }}) \approx 2 \frac{\partial e_{k}}{\partial \theta_{m}} \cdot \frac{\partial e_{k}}{\partial \theta_{n}} \tag{5.12}
\end{equation*}
$$

because near the optimum, the errors $e_{k}$ can be assumed to be distributed around zero. Hence, these terms will cancel out to a large extent when summing over all pixels. Moreover, since the Hessian matrix is only used to compute the direction of search, the optimization process is very tolerant to inaccuracies in the Hessian matrix (this is similar to Quasi-Newton approaches). Consequently, we compute the Hessian matrix $\nabla^{2} E$ from the entries of the gradient vector as

$$
\begin{equation*}
\left(\nabla^{2} E\right)_{m n}=2 \cdot \sum_{k} \frac{\partial e_{k}}{\partial \theta_{m}} \cdot \frac{\partial e_{k}}{\partial \theta_{n}} \tag{5.13}
\end{equation*}
$$

## Image alignment process

The complete background image is constructed by computing refined motion parameters $\breve{\mathbf{H}}_{t}$ between each input image and the background image, while the background image is updated with the new image after each step. In the update of the background image, it is important that only the background pixels that have previously been transparent are replaced in the background to prevent a slow drift over time.

The construction process is as follows:

1. Copy the reference image $I_{r}$ into the initially transparent background image $I_{B}$ using the identity transform $\breve{\mathbf{H}}_{\mathbf{r}}=\mathbf{1}$. Set the next image $t$ to be processed to $t=r+1$.
2. Calculate optimized motion parameters $\breve{\mathbf{H}}_{t}$ for the current frame using the prediction $\breve{\mathbf{H}}_{t-1} \mathbf{H}_{t, t-1}^{-1}$.
3. Add $I_{t}$ to the background $I_{B}$ where only previously transparent pixels are modified.
4. Proceed to the next image $t:=t+1$ and continue at Step 2 until all images are processed.

Images $t$ that are before the reference $r$, i.e., with $i<r$ are added similarly in a second pass.

An example result of the long-term parameter estimation is shown in Figure $5.8(\mathrm{~b})$. It is visible that the accumulated errors from the short-term esimations have been corrected to a globally consistent background image.

Chapter 5. Background Reconstruction

### 5.3 Background estimation

The background image that was synthesized during the alignment process was obtained by simply copying the input frames into the background image. Consequently, the background image will still contain foreground objects. In this section, we will recompute the background image with the intention to remove the foreground objects and to get a pure background image. An example is shown in Fig. 5.8(b), which is actually the output of this background estimation step. Note that the foreground object has been removed, unlike Fig. 5.8(a).

### 5.3.1 Introduction and previous work

Background estimation is the problem of obtaining an image of the scene background from sequences where the background might be occluded by foreground objects during most of the time. Since there is no a-priori knowledge how the background looks like, it can only be estimated by observing the scene for a longer time and by considering anything as foreground that is not static. Note that this definition of background may differ from the usual intuitive meaning, which is very dependent on the context. For example, in a tele-conferencing scene with several participants in a meeting-room, we would consider the meeting-room to be the background, while the humans are foreground. Now consider that at the wall there is a clock, which apparently changes its appearance over time. Based on our intuition, we would still classify the clock as background, but in the sense of image background estimation, the clock would be foreground.

In many cases, the decision is even more difficult to make, or the decision can depend on the time-interval during which we observe the scene. For example, assume that we can see some book shelves at the back of our meeting-room. Normally, we would consider this shelf to be background. However, if one of our persons is removing a book from the shelf, this book is suddenly becoming a foreground object. Further difficulties can arise from gradual environmental changes like the direction of the sunlight or sudden changes when someone switches on a light. A good survey of the problems of background estimation can be found in [186].

Basically, we can identify two classes of background-estimation problems which arise from different applications. The difference between both is the duration for which we observe the scene.

- Long-term. In surveillance applications, we have long-term observations, where a scene is continuously recorded. Background-estimation algorithms for this application must use an update strategy, since the


Figure 5.11: Iterative background estimation for hall-and-monitor sequence up to image 120.
number of available frames is too large to consider them all at once. A typical requirement for these long-term observations is that the background image in fact is not static, but also reflects a gradual change of the scene. Typical problems for this class of algorithm are sudden changes of the background or changes of the semantic meaning as described above.

- Short-term. In a video-analysis application, where the video content includes movies or home-videos, we usually have only short scenes from which we want to create a background image. The main problem in this application is that the video sequence may be very short and yet the algorithm has to find the best possible solution for this limited input. Fortunately, the scenes are usually so short that we do not have to take gradual changes into account.


## Iterative update algorithm

Since most algorithms proposed in the literature have surveillance applications in mind, they use a frame-based update strategy that maintains a current background estimate $I_{B}$ which is updated iteratively with each new input frame $I_{t}$. In the simplest case, the update can be made with a constant update factor $\alpha$ as

$$
\begin{equation*}
I_{B}:=\alpha I_{t}+(1-\alpha) I_{B} \tag{5.14}
\end{equation*}
$$

The disadvantage is that slowly moving objects will appear in the background image as shadows (Fig. 5.11(a)). It has also been proposed [18,


Figure 5.12: A foreground object adds a bias to the median filter, such that it will not output the mean background color.
$148,156]$ to adapt the aging factor to the amount of motion in the input image to reduce the influence of moving objects. More specifically, the update factor is reduced if the frame-to-frame difference or the frame-tobackground difference is high. However, the problem is that because there is no distinction between foreground and background, the update factor is not only reduced when a foreground object covers the background, but also when the background is uncovered again. Consequently, it also takes a longer time to remove erroneous objects from the background (Fig. 5.11(b)). This class of algorithms can only work reliably when foreground objects do not move too slowly and background is visible during most of the time. Otherwise, the reconstruction stays unstable or converges to an average of foreground and background color.

## Temporal median algorithm

A different approach that works better for short image sequences is to use a temporal median-filter over all input frames [123]. This guarantees that the background color is correctly found if the background was visible for more than half of the time. The disadvantage of the algorithm is that a large number of input frames have to be stored. But note that the median value can be computed efficiently by computing the pixel histogram over time and deriving the median value from that histogram.

An advantage of the median-filter algorithm is that the effect of blurring is not as severe as with the weighted update algorithms. However, even with the median-filter algorithm, the background image content can deviate from the correct background color. To understand this, assume that a bright
background is occluded for some period by a dark object. Since the median is computed including the dark foreground object pixels, as well as the bright background pixels, the median will not be at the mean background luminance, but it will be shifted to slightly darker values, caused by shadows or image noise (see Fig. 5.12). The effect can be observed in the results shown in Fig. 5.21(d). Even though all foreground objects are dark and the background is bright, objects appear half-transparent in the reconstruction. At first glance, this should not happen since the median filter should either select the darker foreground or the bright background. However, because of image noise or because there has been a shadow on the background, darker pixels occur and the foreground objects give a bias to the estimation such that darker pixel values are selected. This bias from the correct background color is not always perceivable, but it can cause difficulties in automatic segmentation algorithms, because this bias might already be detected as a significant change.

## Pixel mode algorithm

Another algorithm that follows the same approach as the median algorithm is the mode algorithm. Instead of computing the median of the pixel brightness over time, we select for each pixel the most frequently occurring color. At first view, this seems to be a good solution, but the problem lies in the definition of pixel similarity. If we count the number of occurrences of a specific luminance, we have to allow for some tolerance of the luminance. Otherwise, effects like quantization from an optimally preceding compression step might influence the result.

## Manual synthesis

For the purposes of comparison and ground-truth generation, we wrote a tool that simplifies the manual extraction of the background from a video sequence. The program presents the input sequence to the user and lets him navigate through the sequence. When the user marks an area in a specific input frame, this area is copied into the background image. To simplify this process, the image is divided into small blocks, such that the complete block is copied to the background image when the user selects it.

### 5.3.2 The SimMat background-estimation algorithm

In this section, we discuss a new background-estimation algorithm that does not show the luminance bias as the median algorithm does, and that


Figure 5.13: Sample result of automatic block classification into the three classes (a) static background, bright areas; (b) moving foreground, dark areas; (c) static foreground, black squares. It is visible that the legs of the left human are static, while his upper body is moving.
provides robust results even for difficult scenes. This new algorithm (SimMat) also succeeds in reconstructing the background image for areas that are visible in less than half of the frames. It achieves this by integrating contextual information from neighboring areas, for which the decision is easier, and by also considering motion information.

When we examine typical video sequences, we can see that areas in each of the images can usually be assigned to one of three classes: static background, moving foreground, and static foreground. In this context, the terms static and moving are related to only a single frame. For a foreground object, the classification can change between moving foreground and static foreground during the sequence, since the object can move in some images, but it can also stay at the same position during some other time. It can even happen that parts of the object are in different classes in the same image, like a human that is standing still, but waving with his hands. However, foreground objects never belong to the static background class. An illustration of the three classes for an example picture can be found in Fig. 5.13.

To reconstruct the background image, it is required that we can detect these three classes. While it is not difficult to detect moving objects, the differentiation between static background and static foreground is the main problem. To help in this classification, we can further use the following two assumptions about backgrounds:

- A background never changes its appearance, even if it was occluded by
a foreground object for some time. This means that the background pattern will be visible repeatedly without change, while a foreground object might appear static for a limited time, but not on a global scale.
- If an image area is occluded by a foreground object for some periods, it is probable that a neighboring area is occluded for comparable periods (see Fig 5.17).


## Algorithm overview

The principal idea of our algorithm is as follows. First, we apply a rough classification of the data in the input images into the two foreground classes and the background classes. Afterwards, the background image is synthesized from typical representatives of the background class data. Since foreground data is not considered in the background reconstruction, any bias towards the foreground color is prevented. Moreover, small classification errors do not lead to errors, because only one typical representative of the class is selected, so that outliers do not have much influence.

The classification is carried out on units of small blocks of about $(8 \times 8$ pixels) to reduce the computational cost and to make the classification between background and foreground more robust. Periods in which a block shows background content are identified by searching for the subset of frames in which the block shows a stable content. The similarity of the contents of a single block over time is collected into a similarity matrix $\mathbf{M}$, which contains the difference $\mathbf{M}_{a, b}$ between the image content in this block for each pair of frames $(a, b)$. High values correspond to similar content, whereas low values are found for each pair of frames that contains differing content. Every subset of frames $\mathcal{T}$ induces a decomposition of the matrix into elements that correspond to pairs of frames, where both frames are within $\mathcal{T}$, and entries where at least one corresponding frame is not in $\mathcal{T}$ (see Fig. 5.14). Our goal is to find a subset $\mathcal{T}$ that includes all the framenumbers for which background content is visible in a specific block. For a good solution, the sum of the matrix elements that are covered by $\mathcal{T}$ should be large, since these entries correspond to static background. On the other hand, entries that are not covered by $\mathcal{T}$ should show considerably smaller similarity. This criterion is used in an optimization to find for each block the subset of background frames that has the most static content.

## Block similarity matrices

We begin with presenting the definition of a block similarity matrix. Let the block size be $N \times N$ pixels and let the input sequence be of length $L$.


Figure 5.14: $A$ set of frames $\mathcal{T}$ induces a decomposition of the matrix into entries $M_{a ; b}$, where $a, b \in \mathcal{T}$ (drawn in white), and entries $M_{a, b}$, where either $a$ or $b$ is not in $\mathcal{T}$.

Furthermore, let $I_{t}(x, y)$ be the luminance of pixel $(x, y)$ in input frame $t$. In this section, we assume that $I_{t}(x, y) \in[0 ; 1]$. To simplify the notation, we assume in the following that the camera motion has been compensated as described previously. For each block $(u, v)$ with top left pixel at position $(u N, v N)$, we calculate a symmetric similarity matrix $\mathbf{M}^{(u, v)}$ of size $L \times L$ with

$$
\begin{equation*}
\mathbf{M}_{a, b}^{(u, v)}=1-\frac{1}{N^{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1}\left|I_{a}(u N+i, v N+j)-I_{b}(u N+i, v N+j)\right| . \tag{5.15}
\end{equation*}
$$

This equation states that each matrix element $\mathbf{M}_{a, b}$ is set according to the sum of absolute differences (SAD), measured between the blocks in frame $a$ and frame $b$ at the same block position in the image.

For time periods in which the content in the block does not change, a square block along the matrix diagonal will contain high values (Fig. 5.15). If a specific block content disappears for some time and reemerges later, a corresponding rectangle of matrix elements beneath the matrix diagonal will show low values. Periods with moving content show as low-valued matrix elements. If the content is only visible for a short time, the corresponding matrix rows and columns will contain mostly low values.


Figure 5.15: Structure of a block similarity matrix $\mathbf{M}=\left\{\mathbf{M}_{a ; b}\right\}_{a ; b}$. White matrix elements indicate a high block similarity, while dark elements show low similarity pairs.


Figure 5.16: A sample block similarity matrix taken from the sequence shown in Fig. 5.21.

## Matrix decomposition

To identify the periods in which only background is visible in a block, we decompose the matrix into two parts: the stationary elements (high similarity values), and the non-stationary elements (low similarity values). Let $\mathcal{T}^{(u, v)} \subseteq\{1, \ldots, L\}$ be the set of frames for which we assume that block ( $u, v$ ) only contains background content ${ }^{4}$. Because the background is static, we consider a matrix element $\mathbf{M}_{a, b}$ stationary iff $a$ and $b \in \mathcal{T}$, i.e.,

[^12]background is seen in frame $a$ and frame $b$.
Since stationary matrix elements should have large values and nonstationary elements should have small values, we can separate them by choosing $\mathcal{T}$ such that the stationary elements are as large as possible and the non-stationary elements are as small as possible. More specifically, we optimize the objective function $C$
\[

$$
\begin{equation*}
\max _{\mathcal{T}} C=\max _{\mathcal{T}} \underbrace{\sum_{a, b \in \mathcal{T}} \mathbf{M}_{a, b}}_{\text {stationary elements }}+\underbrace{\sum_{a \notin \mathcal{T} \vee \notin T}\left(1-\mathbf{M}_{a, b}\right)}_{\text {non-stationary elements }} . \tag{5.16}
\end{equation*}
$$

\]

Optimization is carried out using an iterative process. Starting with a good estimate of $\mathcal{T}$ (obtaining a good initialization is described later), we calculate the difference that results from adding or removing each of the input frames from $\mathcal{T}$. If the objective function can be increased by adding or removing a frame, $\mathcal{T}$ is modified accordingly. Optimization is stopped when $C$ cannot be increased further. We have found that this process converges in only about two or three passes over the input frames. Note that instead of the naive way of computing the objective function by summing over the complete matrix, it is sufficient to compute the difference, which can be obtained by summing only over a single matrix row. In the case of adding a frame $k$ to $\mathcal{T}$, the difference is

$$
\begin{align*}
\Delta C_{+k} & =2(\underbrace{\sum_{a \in \mathcal{T}} \mathbf{M}_{a, k}+\sum_{a \notin \mathcal{T}}\left(1-\mathbf{M}_{a, k}\right)}_{\text {new costs }}-\underbrace{\sum_{a \in\{1, \ldots, L\}}\left(1-\mathbf{M}_{a, k}\right)}_{\text {old costs }})  \tag{5.17}\\
& =2\left(\sum_{a \in \mathcal{T}}\left(2 \mathbf{M}_{a, k}-1\right)\right) .
\end{align*}
$$

Clearly, for the case of removing a frame $k$ from $\mathcal{T}$, the difference is just the negative value $\Delta C_{-k}=-\Delta C_{+k}$.

Since the above matrix decomposition process converges to a local minimum close to the initialization, an initialization near the correct minimum must be chosen. Note that the global minimum need not necessarily correspond to the correct background periods. If the sequence contains many foreground objects of the same color, and if the objects are visible during most of the time, the global optimum can correspond to those periods in which foreground objects are visible. To reduce this problem, we apply two additional steps, preceding the optimization step. First, we exclude periods from $\mathcal{T}$ for which we observe motion in the block. Since we assume that camera motion has been compensated beforehand, moving content cannot


Figure 5.17: A foreground object moves across three blocks in the image. The times during which the object is visible in the blocks are correlated.
occur in background regions. Second, we exploit the correlation of background periods between neighboring blocks. Both steps are described in the next two sections.

## Integration of motion information into the similarity matrix

Motion estimation is carried out for each block using a block-matching algorithm in a small neighborhood. If the minimum block matching error is lower than $90 \%$ of the null-vector matching error, the block is considered as moving and the matrix row and column corresponding to the current input frame are artificially set to 0 . This prevents the optimization algorithm from selecting the block in this frame as a background block. Figure 5.16 shows an example how this exclusion disambiguates an otherwise unclear situation.

## Initializing the optimization by background periods prediction

If there is object motion visible in a block, it will most probably also be present in a neighboring block during a comparable time period (see Fig. 5.17). Hence, when calculating $\mathcal{T}^{(u, v)}$, we use the previously calculated $\mathcal{T}^{(u-1, v)}$ and $\mathcal{T}^{(u, v-1)}$ to initialize the optimization process. If an input frame $a$ is contained in $\mathcal{T}^{(u-1, v)}$ and $\mathcal{T}^{(u, v-1)}$, it is also included in $\mathcal{T}^{(u, v)}$. If it is only contained in one of both, it is decided randomly whether to include it. At the left and top border, predictions are formed directly from the solution of the block above or to the left, respectively. The very first block (top-left) is initialized with all input frames active in $\mathcal{T}^{(0,0)}$. This is a sensible assumption, since image activity is usually centered in the image such that the border contains mainly background content.

The spatial prediction scheme has two advantageous properties. First, it provides an accurate initialization of the optimization, leading to fast


Figure 5.18: Spatial background-period prediction (first column of blocks in the input images). The block at the top left $T^{(0,0)}$ contains background content throughout the sequence (background marked in a dark shade). The background periods of $T^{(0, i)}$ form the initialization for background periods of $T^{(0, i+1)}$ (prediction is drawn in a light shade). The matrix decomposition step then refines this prediction to get the final result for this block. Since optimization is started with the last block's result, the optimization will converge to the correct minimum even for blocks that are clearly dominated by foreground objects (e.g., $\left.T^{(0,4)}\right)$.
convergence. Second, the prediction helps to select the correct local minimum, even when the object is visible for a longer time than the background. Since the prediction provides the initialization, even a strong minimum has not enough support in the beginning that the optimization could be attracted to it. This is illustrated in Figure 5.18 and a real-world example is shown in Fig. 5.19.

## Using SimMat for background updating

The algorithm described so far was presented as an offline algorithm that is started when all input frames are available. In fact, this is the way the algorithm is used in our segmentation system. However, as noted previously, surveillance applications usually update the background during the observation to adapt to changing illumination and other changes in the background. If the SimMat algorithm should be used for this application, then it can be done with a simple modification.

Instead of using similarity matrices of size $L \times L$, where $L$ is the sequence length, we set $L$ to a history size. This is the number of past frames that are considered in the background reconstruction. Note that this need not be a continuous stream of frames, but it can also consist of only, e.g., every $10^{\text {th }}$ frame to same memory and computation time.

Whenever a new input frame $t$ is added to the background estimation,


Figure 5.19: Prediction of foreground time periods between adjacent blocks.
we modify the row and column $(t \bmod L)$ in the matrix. This means that the matrix is filled frame by frame and it cyclically starts over without deleting the previous content when the matrix is full. After the matrices are modified, one pass of the optimization algorithm is carried out to adapt the background classification $\mathcal{T}$.

An example result of background updating with SimMat is shown in Fig. 5.24 and it will be described in the following section.

### 5.3.3 Results

We have applied our algorithm to a variety of popular test sequences like the hall-and-monitor, road1, road2, or urbicande sequences. For these sequences, the background could be reconstructed without any visible errors. Even the background from seq_17 of the Video Quality Expert Group (VQEG) test set was recovered without error (see Fig. 5.23). To discuss the properties of the algorithm in comparison to other algorithms, we will evaluate some scenes in more detail.

Chapter 5. Background Reconstruction

## The queue sequence

To see the limits of the reconstruction algorithms, we applied our algorithm to a very difficult sequence (see Fig. 5.21) containing many people, where some persons are walking around and some are standing still for a long time. We carried out background reconstruction with all described algorithms. Because parts of the background are never visible during the whole sequence, it is impossible to get a complete background reconstruction even with a manual synthesis (Fig. 5.21(b)). Some background areas in the center of the image are visible only for a very short time, such that the iterative update, median, and mode algorithms all fail to obtain a good reconstruction. Compared to these algorithms, the SimMat algorithm is able to recover larger areas of the background. Furthermore, it is visible that for the median algorithm (and even worse for the iterative update algorithm), there is a strong bias towards the dark foreground object color. This bias is not present with the SimMat algorithm, which reconstructs the background without any blurring.

## The road1 sequence

The road1 sequence shows a traffic scene, where a car on the right lane slows down and finally stops (Fig. 5.22). We synthesized two background images for the median and the SimMat algorithm, respectively. One background image is computed for frames $1-150$, while the other one is computed for the whole sequence (frames 1-300). Since the car slows down and finally stops in about frame 200, the median algorithm cannot decide clearly if the car should belong to the background or not. The result is a blurred region. The SimMat algorithm does not have this problem and it adapts the background image almost instantaneously. Consequently, we have a background image without the car if we consider the sequence up to frame 150 , and a background image with the car for the whole sequence. Notice that the median algorithm also failed to remove the cars at the end of the street completely. However, applied to the whole 300 frames, the median algorithm provided a similar result as the SimMat algorithm.

## The hall-and-monitor sequence

To further evaluate the behaviour for changing backgrounds, we applied the median and the SimMat algorithms also to the hall-and-monitor sequence. Figure 5.24 depicts the background images that were obtained after every 50 frames. For better visibility in the figure, the images have been cropped to the most important part. To compute the background images, a history size of 150 frames was used, but only every $5^{t h}$ frame was used for the

|  | PSNR |
| :--- | :---: |
| iterative update | 29.86 dB |
| median (cropped) | 30.89 dB |
| median (1.02 dB |  |
| our algorithm | 35.15 dB |
| camera noise | 38.74 dB |

Table 5.1: PSNR between reconstructed background and real background (hall-and-monitor sequence).
computations. The interesting point in this sequence is the bag that the left man is placing on the pedestral, and the monitor that the right man is carrying away. These objects change their status during the sequence to background or foreground, respectively. The left column in the figure shows the input frame, the middle column the output of the median algorithm, and the right column the SimMat algorithm. It is well visible that the median algorithm has difficulties to keep a clean background without the two men. Since the men are walking in the direction of the optical axis, they stay at the same position for a long time. Consequently, they appear partly in the background reconstruction. The same holds for the bag, which appears gradually on the pedestral. The SimMat algorithm does not show this problem. The background stays clean, and it adapts the background after some time to accomodate for the bag on the pedestral.

## Quality of the reconstructed background

To evaluate the quality of the background image with respect to applicability for automatic segmentation algorithms, we measured the difference between the reconstructed background image and the ground-truth background image (Table 5.1). Since the real background image is only available for the hall-and-monitor sequence (in the first frames of the sequence), we used this sequence to obtain the results. We measured the PSNR of several reconstruction algorithms and estimated the camera noise by calculating the PSNR between the first two frames of the sequence. Because the median algorithm cannot remove the foreground objects completely, we calculated the PSNR a second time, now with the erroneous regions excluded. Even with these regions excluded, our algorithm achieves considerably higher PSNR than the median algorithm, which makes it a better choice for segmentation applications.

### 5.4 Summary of the background reconstruction module

This chapter presented the processing steps that are required to synthesize a background sprite image from a video sequence, when approximate models for the camera parameters are already available (see Chapters 3 and 4). These two main processing steps are

- obtaining the accurate motion models to align all input images into the common background image, and
- fusing the input images to remove moving foreground objects from the background image.

The framework of these processing steps is illustrated in Figure 5.20. As input, we apply the approximate parameters $\mathbf{H}_{t, t-1}$ for describing camera motion between two frames $t-1$ and $t$. These parameters are refined and converted to frame-to-mosaic motion parameters with a long-term motion estimation algorithm that also provides a high accuracy. After each step, the accurate parameters $\check{\mathbf{H}}_{t-1}$ are used together with the frame-to-frame motion parameters $\mathbf{H}_{t, t-1}$ to initialize the parameters for the next frame $t$.

The accurate motion parameters are subsequently used in the background reconstruction algorithm to obtain camera-motion compensated input frames. The background-estimation algorithm generates one background image for the whole sequence, which is subsequently saved as background sprite and which will also be used for the subsequent segmentation step (see Chapter 7).

Clearly, the system can be simplified significantly if we know beforehand that the camera is static (as it is often the case for surveillance sequences). In this case, we can omit the camera-motion estimation steps and only keep the SimMat background synthesis algorithm.


Figure 5.20: Data-flow for motion parameter refinement and background estimation.

(a) Typical input frame.

(c) Iterative update.

(e) Mode.

(b) Manual synthesiss

(d) Median.

(f) SimMat.

Figure 5.21: Results for a very complex scene with many walking and standing people. Note that the background cannot be reconstructed completely, since some background regions are never visible in this sequence.

(a) Input frame 150.

(c) SimMat, frame 1-150.

(b) Median, frame 1-150.

(d) SimMat, frame 1-300.

Figure 5.22: Results for the well-known road1 test sequence. The Median algorithm over all 300 frames (not shown) has comparable quality as the SimMat algorithm in (d).


Figure 5.23: Results for $V Q E G$ test sequence 17. Note that we have increased the background image brightness for clarity.

(a) input, 50

(d) input, 100

(g) input, 150

(j) input, 200

(m) input, 250

(p) input, 300

(b) medi., 50

(e) medi., 100

(h) medi., 150

(k) medi., 200

(n) medi., 250

(q) medi., 300

(c) SimM., 50

(f) SimM., 100

(i) SimM., 150

(l) SimM., 200

(o) SimM., 250

(r) SimM., 300

Figure 5.24: Results of online background generation for the hall-andmonitor sequence. History size is 150 frames.

The most exciting phrase to hear in science, the one that heralds new discoveries, is not 'Eureka!' but 'That's funny ...' (Isaac Asimov)


## Multi-Sprite Backgrounds

The previous three chapters presented an algorithm to reconstruct a scene background image from a video sequence. In the MPEG-4 standard, this background image (sprite) can be coded and transmitted independently from the foreground objects. This separation saves bandwidth when the background sprite is sent less often or only once, and the image area that has to be coded comprises only the foreground objects. Whereas it seems optimal to combine as many images into one background sprite as possible, we have found that the counter-intuitive approach of splitting the background into several independent parts can reduce the overall amount of data needed to transmit the background sprite. Furthermore, we show that in the general case, the synthesis of a single background sprite is even impossible and that the scene background should be sent as multiple sprites instead. For this reason, we propose an algorithm that provides an optimal partitioning of a video sequence into independent background sprites (a multi-sprite), resulting in a significant reduction of the involved coding cost. The generated multi-sprite backgrounds are a generalization of the previously discussed background reconstruction algorithm, with the main difference that several background images are used throughout the sequence.

### 6.1 Introduction

The background image that we reconstructed with the algorithms of the previous chapters serves two purposes. First, we need the pure background images for the foreground-object segmentation algorithm that is presented in the next chapter. But we can also use the obtained background images and motion parameters directly as input for an MPEG-4 encoder that supports sprite coding. The concept of sprite coding is that the static sprite image is reused for the decoding of many frames, where each output image shows a partial view of the sprite image. One main advantage of using sprite coding is to reduce the required bandwidth, since the background areas are only sent once with a common sprite image [194].

In this chapter, we take a closer look at the coding efficiency of sprites and relate this to the image formation geometry and the camera motion. Interestingly enough, it will be shown that transmitting the background in a single sprite is generally not the most efficient approach and that the amount of data can be reduced by splitting the background into several separate sprites. Moreover, we clarify that when applying the projective motion model, which is used in MPEG-4, it is only possible to cover at most $180^{\circ}$ field of view in a single sprite, which makes the use of independent sprites a necessity. We also address the optimal placement of the reference frame for defining the sprite coordinate system and we consider a possible loss of resolution for camera zoom-in sequences.

In [25], coding with multiple sprites was proposed to reduce the distortions in a sprite. However, observed distortions are mainly due to using the affine motion-model for sprite generation, which is an inappropriate model for rotational camera motion. Hence, the multiple sprites can only reduce the perceived distortion, but cannot achieve a geometrically correct sprite construction.

If we examine the derivation of the projective motion model from the image formation equations of a rotating camera, we observe that the motion model cannot be applied for large camera rotation angles. Even for small rotation angles, sprite-coding can be inefficient, because the perspective deformation increases rapidly with the rotation angle. We propose to solve this inherent problem of the projective motion model by distributing the background image data over a set of independent sprites instead of trying to code the entire sequence with a single sprite. Despite the increased overhead of using multiple sprites, the total amount of data can be considerably smaller than in the single-sprite case.

Another open point, which we ignored in the last chapter, is the selection of a reference coordinate system. The usual approach to date is to use the first frame of the input sequence as the reference frame. Alternatively,

Massey and Bender [123] propose to use the middle frame of a sequence, which results in a more symmetric sprite shape if the camera performs a continuous panning motion. Instead of using a heuristic reference frame placement, our algorithm also computes the optimal reference frame to minimize the synthesized sprite size.

A further problem that has not yet been treated in the literature is the problem of camera zoom-in operations. If the camera performs a zoom-in, the visible part of the scene becomes smaller, but the relative resolution increases. When the zoomed image is aligned to the sprite background, it means that the sprite area that is covered by the image is smaller. If we do not want to lose the increased resolution of the input, it means that we also have to increase the resolution of the sprite. Otherwise, the input image would be scaled down to the coarser sprite resolution and fine detail would be lost. To prevent this unfavourable loss of resolution, our sprite generation algorithm can incorporate a constraint that ensures that the resolution of no input frame is decreased during the warping process. As a result, sprite coding will not cause any loss of resolution and, consequently, the quality of the decoder output will increase.

The remainder of the chapter is structured as follows. Section 6.2 reveals limitations of the MPEG-4 sprite model and introduces the concept of multi-sprites. Section 6.3 derives a classification method to detect camera configurations for which no appropriate projective transform onto a sprite plane exists. In Section 6.4, the three idealized examples of pure camera zoom-out, zoom-in, and camera rotation are analyzed. It will be shown theoretically that using multi-sprites can in fact reduce the total sprite size. Furthermore, the resolution-preservation constraint is derived from the zoom-in example. Section 6.5 presents several definitions of sprite coding cost, differing in accuracy and computation speed. Moreover, it is shown how to incorporate practical constraints like a limited sprite buffer size. The multi-sprite partitioning algorithm is described in Section 6.6, while experimental results are presented in Section 6.7. An overview how the algorithm can be integrated into a video-object segmentation framework is given in Section 6.8, and we discuss briefly how multi-sprites can be transmitted in standard MPEG-4 sprite VOPs in Section 6.10.

### 6.2 Limitations of the single-sprite approach

MPEG-4 sprite coding is based on the previously described projective motion model, which is defined as Eq. (2.12). In geometric terms, this transformation is a plane-to-plane mapping. Thus, the sprite image can be envisioned as the projection of the 3-D world onto a plane. This is illustrated

(a) The use of ordinary sprites leads to large geometric deformations.

(b) Multi-sprites reduce the deformations.

Figure 6.1: (a) Top-view of projecting the input frames onto the sprite plane. The more the camera rotates away from the frontal view $(|\theta|$ increases), the larger the projection area on the sprite plane. For $|\theta| \geq 90^{\circ}$, the projection ray does not intersect the sprite plane. Hence, only $180^{\circ}$ field of view can be covered with one sprite. (b) Using several sprites reduces the geometric deformation and allows coverage of larger viewing angles.


Figure 6.2: $M P E G-4$ sprite coding is inefficient for large camera rotation angles. Frame 255 covers a much larger area in the sprite than in the original sequence. Hence, a magnified view is transmitted, but only the low resolution is displayed. Note that the reference frame 1 has the same size in the sprite as in the original sequence.
in Fig. 6.1(a), which shows a top-view of a camera that rotates around its vertical axis. If the camera rotates away from the frontal view position, the area on the sprite that is covered by each image projection becomes larger. For projection angles that exceed $90^{\circ}$, input image pixels cannot be projected onto the planar sprite anymore (see, e.g., the pixel position $\mathbf{p}$ ).

As a consequence, ordinary MPEG-4 sprites have the direct limitation that only $180^{\circ}$ field of view can be represented in a single sprite image. If this $180^{\circ}$ limitation is neglected and a wide camera pan is still forced into a single background sprite [118, 119], very strong image distortions are inevitable. In practice, the usable viewing angle is even smaller, since the perspective deformation increases rapidly when the camera rotates away from its frontal view position. Consequently, the required sprite size also increases quickly during a camera pan with short focal length. Unfortunately, even though some input images are projected onto a larger area in the sprite than their original size, this does not result in an increased resolution at the decoder output, since the image will be scaled down to its original resolution again at the decoder. In this sense, the sprite-coding is rather inefficient, since it uses a high resolution for transmitting the sprite although this extra resolution is never displayed (Fig. 6.2).

An alternative representation for background images would be to use spherical or cylindrical image mosaics [180] for the background image, instead of a planar mapping (Fig. 6.3). However, this approach has several disadvantages compared to using the projective motion model. First, generation of spherical/cylindrical mosaics requires that the internal camera pa-


Figure 6.3: Using a cylindrical background model.
rameters like focal length and the principal point of the camera are known. Even though these values can be estimated from the parameters of the projective motion model, the estimation is difficult, since the calculation is numerically sensitive (see Chapter 12). Furthermore, the estimation of the transformation parameters for cylindrical and spherical mosaics requires complicated non-linear optimization techniques, and the reconstruction at the decoder is computationally expensive, because transcendental functions are required. Finally, and above all, the obtained cylindrical/spherical background is not compliant with the MPEG-4 video coding standard, since MPEG-4 only supports the projective transformation model.

In the remainder of this chapter, we propose a more efficient coding technique, based on partitioning the video sequence into several intervals and calculating a separate background sprite for each of them. Although some parts of the background may be transmitted twice, the overall sprite area to be coded is reduced. This counter-intuitive property results from the fact that the perspective deformations do not accumulate much in the multi-sprite case, so that larger parts of the sprite can be transmitted in a lower resolution. Figure 6.1(b) depicts the same scene as in Fig. 6.1(a), but using a two-part multi-sprite instead of only one single sprite. Two advantages of the multi-sprite approach can be observed.

- First, the complete scene can be represented in the multi-sprite because additional sprite planes can be placed as required to cover an arbitrarily large field of view.
- Second, the total projected area becomes smaller, since the sprite plane onto which the input is projected can be switched to a different sprite plane, if this results in a smaller projected area.

Our algorithm for multi-sprite generation finds the optimal partitioning of a video sequence into multi-sprites and also determines for each sprite the optimal placement in 3-D space. Different sprite cost definitions can be selected to adapt the optimization criteria to different application requirements. Finally, the proposed algorithm also allows to integrate additional constraints into its optimization process. These include the specification of a maximum sprite-buffer size at the decoder or a resolution-preservation constraint, which prevents loss of detail during camera zoom-in operations.

### 6.3 Detecting degenerated transforms

As we have seen in Figure 5.5, the projective transform maps image positions using a central projection through the origin onto the flat sprite plane. As a consequence, only points in the half-space in front of the camera should be projected onto the sprite plane, since points on the back-side would be mapped ambiguously onto the same points. However, when applying the projective transform without special treatment for objects behind the camera, these objects are also projected through the optical center onto the sprite plane, where they will appear up-side down. As an example, the point $\mathbf{p}$ in Figure 6.1(a) lies on the right side behind the camera and would be mapped onto the left side of the sprite. In the following, we will call a transformation which maps some image points from behind the camera onto the sprite-plane as degenerated (see Figure 6.4). These transforms must be avoided in the sprite-generation process.

Usually, the camera motion $\mathbf{H}_{i ; i+1}$ between successive frames is small and the problem of degenerated transforms will not appear. However, the concatenation of the frame-to-frame motions to determine the frame-to-sprite transform can lead to this degenerated case which maps points from behind the camera to the other side. Since the sprite construction process only knows the camera motion in the formulation of the eightparameter motion-model from Eq. (2.12), no direct knowledge about the three-dimensional layout is available and an appropriate detection of a degenerated transform has to be performed using only the parameters of the eight-parameter motion-model.

To derive an appropriate detection rule, let us consider again the imageformation Equation (5.1). According to our assumption that the viewing direction is along the positive $z$-axis, the degenerated case occurs if the $z$-coordinate of a pixel after multiplication with the rotation matrix $\mathbf{R}=$ $\left\{r_{i k}\right\}$ becomes zero or negative. If $z=0$, the point would be projected to infinity, which we also subsume into the degenerated case. Since the intrinsic camera-parameters matrix $\mathbf{K}_{s}$ and the shift of the image onto


Figure 6.4: Top view of a horizontal pan set-up. Images a and b include pixels that are at the back-side of the camera; their projection onto the sprite-plane must be avoided. The matrix columns $\left(h_{0 i}, h_{1 i}, h_{2 i}\right)^{\top}$ correspond to the basis vectors of the rotated and scaled coordinate system. Since the basis vector $\left(h_{02}, h_{12}, h_{22}\right)$ corresponds to the rotated viewing direction, a negative $h_{22}$ indicates a rotation of more than $90^{\circ}$ degrees away from the frontal view onto the sprite. A scaled version of the basis vectors can also be found in the inhomogeneous formulation as $\left(a_{00}, a_{10}, p_{x}\right)^{\top},\left(a_{01}, a_{11}, p_{y}\right)^{\top},\left(t_{x}, t_{y}, 1\right)^{\top}$. However, because of the normalization process which sets $h_{22}=1$, these basis vectors may swap their orientation. This is depicted for the input image $I_{a}$. For image $I_{b}$, the matrix entry $h_{22}$ is $>0$ and no swapping occurs.
its focal plane by the matrix $\mathbf{K}_{i}^{-1}$ does not modify the sign of this value, the degenerated case for a specific point $(\hat{x}, \hat{y})$ can be detected with the condition

$$
\begin{equation*}
w^{\prime}=h_{20} \hat{x}+h_{21} \hat{y}+h_{22} \leq 0 . \tag{6.1}
\end{equation*}
$$

However, since $\mathbf{H}_{i}$ is scaling invariant and the motion parameters are normalized to $h_{22}=1$ in the formulation of the eight-parameter model, the sign of all matrix entries $h_{i k}$ can change because of the normalization, and the above test condition would be reversed to its opposite. Therefore, the condition must be modified to be invariant to the normalization.

To derive a suitable condition for the normalized parameters, we have to identify when the normalization altered the signs. Consider the case where $h_{22}<0$ prior to the normalization process. In this case, the normalization process changes the sign of all matrix entries. Since each column of the
rotation matrix represents the direction of a rotated basis vector, changing the signs of all matrix entries $h_{i k}$ will swap the directions of all of those basis vectors. Because we assumed that the coordinate system is originally right-handed, each swap of a basis vector will change the orientation of the coordinate system, so that after the three basis vector swaps, the basis becomes now left-handed. To detect this, we can observe the sign of the determinant $D$ of the matrix of normalized parameters

$$
D=\left|\begin{array}{ccc}
a_{00} & a_{01} & t_{x}  \tag{6.2}\\
a_{10} & a_{11} & t_{y} \\
p_{x} & p_{y} & 1
\end{array}\right|
$$

If the determinant $D>0$, the coordinate system is right-handed (it is not necessarily equal to unity, since the length of the basis vectors is not unity), otherwise, it is left-handed. Note that the matrix entry $h_{22}$ corresponds to the $z$-coordinate of the basis-vector in $z$ direction. Since the camera looks along the $z$-axis, a negative $h_{22}$, or equivalently, $D<0$, corresponds to a rotation of more than $90^{\circ}$ away from the frontal viewing position, so that the camera is looking into the opposite direction.

Finally, this lets us derive the condition to decide whether a point $\mathbf{p}=$ $(\hat{x}, \hat{y})$ is projected onto the sprite in a non-degenerated way. For this, we start with Eq. (6.1) using normalized parameters, obtaining the condition $p_{x} \hat{x}+p_{y} \hat{y}+1 \leq 0$. Combining this with the sign of $D$ leads to the final condition

$$
D \cdot\left(p_{x} \hat{x}+p_{y} \hat{y}+1\right) \quad \begin{cases}>0 & \text { non-degenerated case }  \tag{6.3}\\ \leq 0 & \text { degenerated case }\end{cases}
$$

To decide if an image as a whole would be mapped non-degenerated onto the sprite plane, we examine the four corner points of the image, which all must be transformed in a non-degenerated way.

### 6.4 Examples of single-sprite inefficiencies

Let us first describe some idealized examples to clarify why the MPEG-4 sprites are inefficient in the general case, and how this problem can be alleviated using multi-sprites. However, note that the algorithm described in Section 6.6 is not limited to these special cases, but finds the optimum solution for any real-world sequence.

### 6.4.1 Example case: camera zoom-out

As a first example, we consider the case that the camera is performing a continuous zoom-out operation. Since each image covers a larger view
than the previous one, the projection area on the sprite plane is constantly increasing. At first, using a single sprite is advantageous, because most of the image was already visible in the previous image. However, when the zoom continues, the situation will eventually change, so that the increase of the total sprite size outweighs the reuse of the already existing background content and it would be better to start with a new sprite (also see the real-world example in Figure 6.17).

If we denote the zoom factor between two successive frames as $s$ and the image size as $W \times H$, the sprite size after $n$ frames will be $W H s^{2 n}$. Considering the alternative, in which a two-part multi-sprite is constructed with each sprite comprising only half of the frames, the total size of the multi-sprite is $2 \cdot W H s^{2 n / 2}$. Consequently, coding the scene as a two-part multi-sprite results in a lower total sprite area iff

$$
\begin{equation*}
W H s^{2 n}>2 \cdot W H s^{n} \quad \leftrightarrow \quad n>\log _{s} 2 \tag{6.4}
\end{equation*}
$$

Generalizing this result, it is easy to derive that a $p$-part multi-sprite gives a smaller sprite area than a $(p-1)$-part multi-sprite, provided that the sequence length $n$ satisfies

$$
\begin{align*}
p W H s^{2 n / p} & <(p-1) W H s^{2 n /(p-1)} \\
\log _{s} p+\frac{2 n}{p} & <\log _{s}(p-1)+\frac{2 n}{p-1}  \tag{6.5}\\
n & >\frac{p(p-1)}{2}\left(\log _{s} p-\log _{s}(p-1)\right)
\end{align*}
$$

### 6.4.2 Example case: horizontal camera pan

Alternatively, let us now assume a camera set-up where the camera only performs rotation around the vertical axis (camera pan, Fig. 6.5). Input images are assumed to have normalized size $W \cdot H=1$ and the aspect ratio $W: H=4: 3$. Furthermore, we assume that the sprite plane is placed at a distance from the camera which is equal to the focal length $f$. Hence, if the camera is in the frontal view position, input images projected onto the sprite plane remain at the same size. If the camera leaves this frontal view position, the projection area on the sprite increases. In the following, we observe the sprite size resulting from a camera pan with angle $\alpha$. Obviously, the sprite size will be minimal if the pan is performed symmetrically. This means that when starting from the frontal view position, we rotate the camera $\alpha / 2$ to the left and an equal amount $\alpha / 2$ to the right. Since we assume that the origin of the input image coordinate system is positioned


Figure 6.5: Set-up for the example case of horizontal camera pan.


Figure 6.6: The sprite area that is covered by projecting images from a rotating camera onto the sprite plane in the set-up of Figure 6.5. Regular intervals of camera rotation are depicted with small vertical marks along the area contour. Note that the projection area increases much faster at larger rotation angles.
at the image center, it is sufficient to consider only one corner of the image, because the other corners can be obtained by mirroring the $x$ and $y$ coordinates.

Using the abbreviations $c_{\theta}=\cos \theta$ and $s_{\theta}=\sin \theta$, our camera model in this example is

$$
\begin{align*}
\left(\begin{array}{c}
x^{\prime} \\
y^{\prime} \\
w^{\prime}
\end{array}\right) & =\left(\begin{array}{lll}
f & 0 & 0 \\
0 & f & 0 \\
0 & 0 & 1
\end{array}\right)\left(\begin{array}{ccc}
c_{\theta} & 0 & s_{\theta} \\
0 & 1 & 0 \\
-s_{\theta} & 0 & c_{\theta}
\end{array}\right)\left(\begin{array}{c}
\hat{x}=W / 2 \\
\hat{y}=H / 2 \\
\hat{z}=f
\end{array}\right) \\
& =\left(\begin{array}{c}
f c_{\theta} W / 2+f^{2} s_{\theta} \\
f \cdot H / 2 \\
-s_{\theta} W / 2+f c_{\theta}
\end{array}\right) . \tag{6.6}
\end{align*}
$$

Hence, if the camera is rotated by an angle $\theta$, the top right image corner ( $W / 2, H / 2$ ) projects to (see also Fig. 6.6)

$$
\begin{equation*}
x(\theta)=\frac{f c_{\theta} \frac{W}{2}+f^{2} s_{\theta}}{-s_{\theta} \frac{W}{2}+f c_{\theta}} \quad ; \quad y(\theta)=\frac{f \frac{H}{2}}{-s_{\theta} \frac{W}{2}+f c_{\theta}} . \tag{6.7}
\end{equation*}
$$

Consequently, the area $A$ that is covered by image content can be calculated by

$$
\begin{equation*}
A(\alpha)=W \cdot H+4 \int_{\theta=0}^{\theta=\alpha / 2} y(\theta) \frac{\mathrm{d} x}{\mathrm{~d} \theta} \mathrm{~d} \theta \tag{6.8}
\end{equation*}
$$

where the integral covers one of the four symmetric "wings" of the sprite. Figure 6.7 depicts the total covered sprite area $A$ for two different camera set-ups, one using $f=1$ (wide-angle) and the other for $f=10$ (tele). For both set-ups, three alternatives were examined using Eq. (6.8). The first one is the coding with an ordinary sprite ${ }^{1}$, whereas the other two alternatives are using multi-sprites with two or three parts. In the multisprite cases, the total pan angle was divided into equal parts and a separate sprite was generated for each part. Hence, the total sprite area that is required for a $n$-part sprite is $n \cdot A(\alpha / n)$. Figure 6.7 depicts the sprite area $A$, depending on the total pan angle $\alpha$ for the different set-ups and number of sprites used. For very low pan angles, it is clear that the ordinary sprite construction is more efficient, since the multi-sprite coding has the overhead of multiple transmission of mainly the same content. However, because of the fast increasing geometric distortion in the single-sprite case, the two-part multi-sprite becomes more efficient for pan angles over about $25^{\circ}(f=10)$. Finally, for angles exceeding approximately $45^{\circ}$, using a three-part sprite becomes the most efficient partitioning.

[^13]

Figure 6.7: The covered sprite area for horizontal camera pan at two different focal lengths. The reference image size is 1 . If the camera rotation exceeds a specific angle, the area to be coded in the multi-sprite case is lower than for a single sprite.

### 6.4.3 Example case: camera zoom-in

Another sprite-generation problem, which is different from the above two cases, occurs if the camera performs a zoom-in after the reference frame. Since the resolution of the input frame is reduced when the image is mapped into the sprite, the resulting output quality at the decoder degrades because fine details of the input frames are lost. To prevent this undesirable property, we introduce a constraint to ensure that the sprite resolution is never lower than the corresponding input resolution.

Let us first define a magnification factor $m_{l}\left(x^{\prime}, y^{\prime}\right)$ that indicates for each pixel in the sprite, by which factor its size has been magnified with respect to the input image $l$. To prevent quality loss, $m_{l}\left(x^{\prime}, y^{\prime}\right)$ should always be $\geq 1$ (project to the same size or larger). Obviously, this will not be the case during zoom-in sequences, but it can also be violated for rotational motion. Hence, from now on, we will not only concentrate on the zoom-in case, but indicate the solution for the general case.

Because we want to ensure that $m_{l}\left(x^{\prime}, y^{\prime}\right) \geq 1$ for all pixels in the whole video sequence, we have to determine the minimum $m_{l}\left(x^{\prime}, y^{\prime}\right)$ for the whole sequence and increase the sprite resolution by the reciprocal value. Since the motion model includes perspective deformation, the scaling factor is


Figure 6.8: Change of local resolution. The input image (left) is warped to the sprite coordinate system (right). In general, this transformation will change the size of a pixel.
not constant over a single input frame (see Fig. 6.8). The local scaling factor can be computed using the Jacobian determinant of the geometric transformation Eq. (2.10), which maps the input-image coordinate system to the sprite coordinate system. Consequently,

$$
\begin{align*}
& m_{l}\left(x^{\prime}, y^{\prime}\right)=\left|\begin{array}{ll}
\frac{\partial x^{\prime}}{\partial x} & \frac{\partial x^{\prime}}{\partial y} \\
\frac{\partial y^{\prime}}{\partial x} & \frac{\partial y^{\prime}}{\partial y}
\end{array}\right|=\frac{1}{D^{2}}\left[\left|\begin{array}{cc}
a_{00} & a_{01} \\
a_{10} & a_{11}
\end{array}\right|\right. \\
&\left.-\left|\begin{array}{cc}
a_{00} & a_{01} \\
p_{x} & p_{y}
\end{array}\right| y^{\prime}+\left|\begin{array}{cc}
a_{10} & a_{11} \\
p_{x} & p_{y}
\end{array}\right| x^{\prime}\right], \tag{6.9}
\end{align*}
$$

where $D=p_{x} x+p_{y} y+1$ is the denominator of the motion model equations ${ }^{2}$. For non-degenerated image projections, $m_{l}\left(x^{\prime}, y^{\prime}\right)$ is monotonic in $x^{\prime}$ and $y^{\prime}$, and its minimum value over the image area can be found in one of the image corners. Hence, to determine the minimum value $\bar{m}_{l}=\min _{x^{\prime}, y^{\prime}}\left\{m_{l}\left(x^{\prime}, y^{\prime}\right)\right\}$ over a complete input image $l$, we only have to compute $m_{l}\left(x^{\prime}, y^{\prime}\right)$ for the four image corners and select the minimum value.

We will now consider a sprite which is built from input frame $i$ to $k$. Let $\bar{m}_{i ; k}=\min _{l ; i \leq l \leq k} \bar{m}_{l}$ be the minimum scaling factor of all frames between $i$ and $k$. To preserve the full input resolution for all frames that were merged into the sprite, the sprite resolution has to be scaled up by a factor of $1 / \bar{m}_{i ; k}$. The increase of coding cost induced by the enlarged sprite area can be integrated into the definition of coding cost as will be shown in Section 6.5.4.

[^14]
### 6.5 Sprite cost definitions

Optimization towards minimum sprite coding cost requires a formal definition of coding cost. Thus, let $S_{i ; k}^{r}$ be the sprite which is constructed using input frames $i$ to $k$ and which uses frame $r$ as its reference coordinate system. The following sections propose several definitions of costs $\left\|S_{i ; k}^{r}\right\|$ which differ in accuracy and computational complexity. Finally, we show how constraints can be introduced into the optimization process by combining several cost definitions.

### 6.5.1 Bitstream length

The obvious choice for defining the sprite coding-cost is the bitstream length itself. However, this definition is not practical, because of the high computational complexity required. The optimization algorithm for determining the optimal sprite arrangement (see Section 6.6) requires the cost for coding sprites of all possible frame ranges and reference frames. Calculating these costs is the most computation intensive part of the algorithm. For this reason, estimates which are more easy to compute will be pursued.

### 6.5.2 Coded sprite area

As an approximation to the actual bitstream length, we can use the sprite area that is covered with image content. In a real implementation, Eq. (6.8) cannot be used, since the covered area is composed of discrete projections. Instead, we describe the coded sprite area using a polygon $x_{i}, y_{i}$ along the sprite border (see Figure 6.9). Whenever an image is added to the sprite, the quadrilateral of the image border is combined with the boundary polygon around the sprite to represent the new contour. The polygon area can be calculated rapidly using Green's theorem by

$$
\begin{equation*}
\|\cdot\|_{A}=\frac{1}{2} \sum_{i \in\{0 ; \ldots ; l-1\}}\left(x_{i} y_{i+1}-x_{i+1} y_{i}\right) . \tag{6.10}
\end{equation*}
$$

Computing the sprite area for sprites over the same frame range, but with a different reference frame, can be simplified. Obviously, the relative placements of the input frame projections stay the same, regardless of the reference coordinate system. Hence, the contour polygon only has to be computed once, say, for frame $k$. In order to compute the contour polygon for another frame $i$, we only have to apply $\mathbf{H}_{\mathbf{i} ; \mathbf{k}}$ to every point of the contour polygon and recompute the polygon area.

The sprite-area criterion assumes that the bitstream length is proportional to the number of coded macroblocks. This is only the case if on


Figure 6.9: The boundary polygon around the sprite area is computed as the combined outlines of all transformed quadrilaterals. For simplicity of notation, we double the last point $\left(x_{l}, y_{l}\right)=$ $\left(x_{0}, y_{0}\right)$.
the average, the block content does not depend on the sprite-construction process. However, as we have seen previously, different areas of the sprite are synthesized with differing local resolution. Since the amount of image detail per block decreases when the image is magnified in the projection, the relative coding-cost per block also decreases. This is not reflected with the cost definition of $\|\cdot\|_{A}$, which only considers the sprite area, regardless of the detail that is left. Hence, when using an area-based cost definition, there will be a small bias towards making magnified areas more costly than when using the theoretically optimal bitstream length cost definition of the previous section.

To determine the relationship between the resolution scaling factor and the bitstream length, we scaled images from several sequences to different sizes and compared the bitstream length after coding the scaled images as MPEG-4 sprite images. All the coding parameters were held constant during the experiment. The results are depicted in Figure 6.10. Even though the input images have very different content, the relationships between scaling-factor and increase of bitstream length seem to be comparable. Small peaks in bitstream size can be observed at $100 \%, 150 \%$ and $50 \%$. Since bi-linear interpolation was used for the scaling, which smoothes the image a little bit, the bit-rate decreases. At integer and other regular scaling factors, the pixels are sampled without effective interpolation, which explains the slightly higher bit-rate. It can be seen that, as assumed, the bitstream length does not increase linearly with the image size, but only with an exponent of $1.6 / 2$. Up to now, we assume a simple linear relationship, but future work might try to compensate for this effect by integrating


Figure 6.10: Increase of bitstream length for rescaled image resolutions. Resizing the image dimensions each by a factor $\sqrt{m(x, y)}$ increases the bitstream size by about $m(x, y)^{1.6 / 2}$, which is a bit less than a linear increase relative to the image area.
a detail-loss factor. However, we do not expect a significant difference since for the optimal sprite-partitionings, we observed that $m(x, y)$ is close to 1 over large parts of the sprite.

### 6.5.3 Sprite buffer size

A further approximation to the real bitstream length, providing a quick computation, is to take the area of the bounding box (which we will denote by $\|\cdot\|_{B}$ ) around the sprite. Also note that the bounding box size is equal to the required sprite buffer size at the decoder. Hence, optimizing for the bounding box size is equivalent to minimizing memory requirements for sprite storage at the decoder. Except for rare extreme cases, the result when using the bounding box as an optimization criterion $\left(\|\cdot\|_{B}\right)$ differs not much from using the really covered sprite area $\left(\|\cdot\|_{A}\right)$. The explanation is that an optimal multi-sprite arrangement will have as little perspective deformation as possible. Hence, the covered sprite area will be almost rectangular and obviously, the bounding box is a good approximation for almost rectangular shapes.

### 6.5.4 Adding a resolution preservation constraint and limiting sprite buffer requirements

A cost definition based only on the sprite area gives inappropriate results if the camera zooms into the scene. Since the algorithm tries to minimize the total sprite area, it will select the frame at the beginning of the zoomin as reference. As we have described in Section 6.4.3, this would lead to a poor quality for the decoded images at the end of the zoom sequence. Hence, we have to constrain the solution such that the local scale $m\left(x^{\prime}, y^{\prime}\right)$ in the sprite $S_{i ; k}^{r}$ never falls below unity. This is achieved by calculating the magnification factor $\bar{m}_{i ; k}$ and multiplying the area size with $\bar{m}_{i ; k}^{-1}$. This correction factor reflects the potential resolution increase which is carried out in the final sprite synthesis. Note that increasing the sprite resolution by the factor $\bar{m}_{i ; k}$ corresponds to shifting the sprite plane in 3-D closer to the origin $\left(f_{s}^{\prime}=\bar{m}_{i ; k}^{-0.5} f_{s}\right)$.

A further constraint may be a limited sprite buffer size at the decoder. For example, the MPEG-4 profile Main@L3 (CCIR-601 resolution) defines a maximum sprite buffer size of 6480 macroblocks. Consequently, the encoder has to consider this maximum size in its sprite construction process. We can include this constraint into the cost function by setting the cost to infinity when the sprite size exceeds the buffer size limitation. Finally, we also set the cost to infinity if the input image cannot be projected onto the sprite plane because the transform would be degenerated. This case is detected using the test condition derived in Section 6.3.

Adding the described constraints to the area cost definition results in the following combined cost definition

$$
\left\|S_{i ; k}^{r}\right\|_{C}= \begin{cases} & \begin{array}{l}
\text { if frame range } i ; k \text { cannot } \\
\text { be projected onto a single } \\
\\
\text { sprite }, \\
\text { if }\left\|S_{i ; k}^{r}\right\|_{B} \text { exceeds the max- } \\
\text { imum sprite buffer size } \\
\frac{\left\|S_{i ; k}^{r}\right\|_{A}}{m_{i ; k}} \tag{6.11}
\end{array} \text { else. }\end{cases}
$$

It is easy to see that for any sensible definition of sprite coding-cost, the cost is monotone for the beginning and the end of the frame range. More specifically, for a frame range $a ; b$ with $a \leq i$ and $b \geq k$, it holds that $\left\|S_{a ; b}\right\| \geq\left\|S_{i ; k}\right\|$, since a sprite over a range $a ; b$ must also contain at least the same information as the sprite constructed from every sub-range $i ; k$.

We use the combined cost definition $\|\cdot\|_{C}$ in the optimization, since it is fast to compute and it also ensures that the obtained sprite fits into the decoder sprite buffer.


Figure 6.11: Sprite for frames 1 to 7 with frame 4 as the reference frame. In this case, the bounding-box for $S_{1 ; 7}^{4}$ has been computed by combining the bounding boxes of $S_{1 ; 4}^{4}$ and $S_{4 ; 7}^{4}$.

### 6.6 Multi-sprite partitioning algorithm

To find the best multi-sprite configuration, the algorithm has to determine the optimal range of input frames for each multi-sprite part, and additionally, for each sprite the optimal reference frame.

The multi-sprite partitioning algorithm comprises two main steps. In the first step, it computes the cost for coding a sprite $S_{i ; k}$ for all possible input frame ranges $i ; k$. Moreover, it determines the best reference-coordinate system for each of these frame ranges by selecting that input frame as a reference, for which the sprite area for this frame range would be smallest. The second step partitions the complete input sequence into frame ranges, such that the total sprite coding cost is minimized.

### 6.6.1 Cost matrix calculation and reference frame placement

In this preprocessing step, we prepare all the sprite costs required for the main optimization step. For each pair of frames $i, k$ with $(i \leq k)$, we consider the cost $\left\|S_{i ; k}^{r}\right\|$ for all reference frame placements $r$ with $i \leq r \leq k$. Since we can choose the optimal reference frame for each of the sprite ranges independently, we select the placement for which the sprite cost is lowest. The sprite cost for optimal placement of the reference is denoted with

$$
\begin{equation*}
\left\|S_{i ; k}^{*}\right\|=\min _{r}\left\|S_{i ; k}^{r}\right\| \tag{6.12}
\end{equation*}
$$

The enumeration of all possible configurations of $i, k$, and $r$ may seem computationally complex, but can be calculated efficiently for most cost definitions (including $\|\cdot\|_{A},\|\cdot\|_{B}$, and $\|\cdot\|_{C}$ ) using a two-step approach.

In the following, it is assumed for simplicity that the cost definition is based on the sprite bounding box, but the same principle can also be applied to the area computation.

We begin with computing all bounding boxes for the case that the first frame in a range is selected as reference frame $\left(S_{r ; k}^{r}\right)$. These costs can be computed efficiently for all $k$ by starting with the bounding box of $S_{r ; r}^{r}$, which has simply the input image size. Each $S_{r: k}^{r}$ can now be computed iteratively from its predecessor $S_{r ; k-1}^{r}$ by enlarging the predecessor's bounding box to include frame $k$. The same process is repeated in the backward direction to compute all $S_{i ; r}^{r}$. When both directions are processed, is it possible to quickly determine the bounding box for $S_{i ; k}^{r}$ by computing the enclosing bounding box of $S_{i ; r}^{r}$ and $S_{r ; k}^{r}$ (see Fig. 6.11). Let us denote the computation of the enclosing bounding box of two sprites $S_{i ; r}^{r}$ and $S_{r ; k}^{r}$ as $\left\|S_{i, r}^{r} ; S_{r ; k}^{r}\right\|=\left\|S_{i ; k}^{r}\right\|$. The difference to using the sprite size $\left\|S_{i ; k}^{r}\right\|$ directly is that this would require a look-up in a three-dimensional array of precomputed cost-values. By splitting the sprite-range into two parts, namely, the range preceding the reference frame and the remaining range after the reference frame, precomputed sprite costs can be determined with lower memory requirements, since only two triangular matrices are stored.

Consequently, when we determine $\left\|S_{i ; k}^{*}\right\|$ by searching for the $r$ that results in the minimum area bounding box, we do not apply Eq. (6.12) directly, but combine the cost using the two sprite halves as

$$
\begin{equation*}
\left\|S_{i ; k}^{*}\right\|=\min _{r}\left\|S_{i ; r}^{r} ; S_{r ; k}^{r}\right\| . \tag{6.13}
\end{equation*}
$$

The results are stored in an upper triangular data matrix consisting of the values $\left\|S_{i: k}^{*}\right\|$. These values serve as the input data for the subsequent optimization algorithm. Additionally, we store the reference frame $r$ for each $S_{i ; k}^{*}$ as it was found in the minimization Eq. (6.13). This value is not needed for the optimization, but the final sprite-image generation uses the information for selecting the reference coordinate system.

### 6.6.2 Optimal sequence partitioning

In the sequence-partitioning step, the input frames are divided into separate ranges, so that the total cost to code the sprites for all frame ranges is minimal. More formally, let $P=\left(\left(1, p_{1}-1\right),\left(p_{1}, p_{2}-1\right), \ldots,\left(p_{n-1}, N\right)\right)$ be a partitioning of the video sequence of length $N$ into $n$ sub-sequences. The optimization problem can then be formulated as determining the partitioning $P^{*}$ for which the sum of all sprite costs is minimal:

$$
\begin{equation*}
P^{*}=\arg \min _{P} \sum_{(i, k) \in P}\left\|S_{i ; k}^{*}\right\| . \tag{6.14}
\end{equation*}
$$



Figure 6.12: Determination of the optimal sequence partitioning. Each state $c_{k}$ is assigned the minimum coding cost for a partitioning ending in frame $k$. Each arrow represents the cost for the sprite built from the covered frames. For each $c_{k}$ with $k \geq 1$, the sprite that results in the minimum cost in node $c_{k}$, is marked with a bold arrow. Tracing back the bold arrows from the last node ( $c_{5}$ ) provides the optimal partitioning with minimum cost.

This minimization problem can actually be viewed as a minimum-cost path search in a graph, where the graph nodes correspond to the input frames plus an additional dummy start node, $V=\{0, \ldots, N\}$. The graph is fully connected with directed edges $E=\{(i ; k) \mid i, k \in V ; i<k\}$. Each edge $(i ; k)$ is attributed with edge costs $\left\|S_{i+1 ; k}^{*}\right\|$. Every path from the start node 0 to node $N$ defines a possible partitioning, where each edge on the path corresponds to one frame range for which a sprite is generated. Consequently, the minimum cost path gives the minimum-cost partitioning $P^{*}$. The shortest-path search can be carried out using a standard Dijkstra algorithm or $A^{*}$ search.

However, because of the regular graph structure, the minimization problem can also be computed with simple iterative algorithm (Figure 6.12). For each image $i$, we compute the minimum $\operatorname{cost} c_{k}\left(c_{0}=0\right)$ of a partitioning ending in image $k$ as

$$
\begin{equation*}
c_{k}=\min _{i \in[1, k]}\left\{c_{i-1}+\left\|S_{i ; k}^{*}\right\|\right\} . \tag{6.15}
\end{equation*}
$$

The index $i$ denotes the beginning of the last sub-sequence in the partitioning up to frame $k$. For each image, we store the $i$ for which the minimum was obtained. Tracing back these stored $i$-values, starting at frame $N$, results in the optimal partitioning with respect to total sprite size.

When searching through the possible values of $i$ in Eq. (6.15), a common
case is that the sprite cost $\left\|S_{i ; k}^{*}\right\|$ will reach $\infty$ when a cost definition according to Eq. (6.11) is used. As the cost cannot decrease if the frame range is extended (see Section 6.5.4), an efficient way is to carry out the search for $i$ backwards, starting with $k$ and stopping the search as soon as $\infty$ is obtained for the sprite cost $\left\|S_{i ; k}^{*}\right\|$.

### 6.7 Experiments and results

We have implemented the algorithm with the sprite cost definition of Section 6.5.4. This section describes the algorithm results for the three sequences table-tennis, rail, and stefan. The sequences table-tennis and stefan are well-known test-sequences, whereas the rail sequence was recorded from a public DVB broadcast. In the Figures 6.17-6.22, we indicate the frame range which was used to generate the sprite, the bounding-box size, and the covered sprite area in 1000-pixel units. The obtained sprite sizes are also summarized in Table 6.1.

From the table-tennis sequence, the first camera-shot consisting of 132 frames has been selected. This camera-shot shows a long zoom-out, starting from a close-up of the player's hand to a wide-angle view of the complete player. Our algorithm prevents the sprite from growing too large by splitting the sequence into a three-part multi-sprite (Fig. 6.17). Compared with the size of an ordinary single-part sprite, the area of the multi-sprite is a factor of 2.9 smaller. The resolution-preservation constraint enforced that the first frame of each part was selected as the reference frame. Since the first frames appear with the highest resolution in the sprites, optimal reconstruction quality is assured.

The rail sequence (Fig. 6.18) contains a complicated camera rotation. It starts with the camera looking downwards and continues with the camera rotating to the left and around its optical axis at the same time. At the end, the camera is looking to the left side. Integration of the complete sequence into a single sprite leads to a very strong deformation of the input frames which makes the conventional approach rather impractical (Fig. 6.19). Applying the multi-sprite algorithm to the sequence results in a three-part multi-sprite, where each of the sprites shows only little perspective deformations.

Since the rail sequence does not contain foreground objects, it was possible to measure the quality of the sprite reconstruction compared to the input sequence. The reconstruction quality of uncompressed background sprites is measured by synthesizing the sprites from the input sequence and then applying the global-motion compensation on the sprites to reconstruct the input sequence again. The measurements were carried out using three


Figure 6.13: Comparison of sprite reconstruction quality for the rail sequence from Fig. 6.18.
different types of sprite-construction: multi-sprite coding with integration of the scale-factor $\bar{m}_{i ; k}$, without the scale-factor, and a heuristic sprite partitioning. In the heuristic sprite-partitioning, the sprite was built iteratively until the sprite width exceeded a threshold. The threshold was chosen such that the first sprite covers frame 1-82 (see Fig 6.19), which corresponds to the frame range of the first two sprites obtained from the multi-sprite partitioning. Figure 6.13 depicts the reconstruction quality of the different approaches. Apart from the fact that the multi-sprite reconstruction clearly outperforms the single-sprite reconstruction by about 1 dB , it can also be seen that the integration of the scaling-factor in fact increases the reconstruction quality in the last part of the sequence.

Since the reconstruction from the sprite is always based on a static sprite, although the input is a moving image sequence, variations in the image apart from camera motion cannot be reconstructed from the sprite. Even if there are no perceivably moving objects in the sequence, the input images can still vary, e.g. because of motion-blur during a fast camera pan. Moreover, the camera optic can also deform the image by radial lens distortion, which cannot be represented in the sprite. Hence, it is clear that the sprite reconstruction cannot be perfect. On the other hand, since many input frames are combined when synthesizing the background sprite, a super-resolution effect occurs, so that the amount of detail in the sprite is even higher than in the original video. This can be observed in Fig. 6.20, which shows a magnification of part of the Fig. 6.18(a). Since the input was

Chapter 6. Multi-Sprite Backgrounds

|  | Single sprite |  | Multi sprite |  |
| :--- | ---: | ---: | ---: | ---: |
| Sequence | Bounding box | Covered area | B. box | Area |
| table-t. (1-132) | $2557 \mathrm{k}(292 \%)$ | $2540 \mathrm{k}(295 \%)$ | 875 k | 860 k |
| rail | $\mathrm{N} / \mathrm{A}$ | $\mathrm{N} / \mathrm{A}$ | 630 k | 492 k |
| rail (1-82) | $748 \mathrm{k}(152 \%)$ | $427 \mathrm{k}(125 \%)$ | 493 k | 340 k |
| stefan | $\mathrm{N} / \mathrm{A}$ | $\mathrm{N} / \mathrm{A}$ | 936 k | 841 k |
| stefan $(1-255)$ | $2509 \mathrm{k}(481 \%)$ | $1208 \mathrm{k}(264 \%)$ | 521 k | 457 k |

Table 6.1: Comparison of sprite sizes using single sprites and the multisprite approach. The area of the bounding-box and the covered sprite area are expressed in units of 1000 pixels.
originally MPEG-2 compressed, it shows some noise, which is not present in the sprite reconstruction. Furthermore, clearly more detail is visible in the sprite reconstruction. Consequently, a decrease in PSNR compared to the input does not necessarily correspond to a reduction of perceived quality.

For the MPEG-4 sequence stefan, we first attempted to generated an ordinary sprite-image for the complete 300 frames. However, because the total viewing angle during the sequence is too large, it is not possible to synthesize a single background sprite. When adding images after frame 255 (which is approximately in the middle of the final fast camera pan), the geometric distortion increases very quickly. Hence, we used only the first 255 frames for building the sprite. The resulting sprite is shown in Figure 6.21. Applying the multi-sprite algorithm on the complete sequence resulted in a four-part multi-sprite, which is shown in Figure 6.22. We have measured that the total required sprite size for the multiple-sprite approach is a factor of 2.6 smaller than for the single-sprite case. However, note that the multi-sprite covers the complete 300 frames of the sequence, while the ordinary sprite covers only the first 255 frames. The effect of the resolution-preservation constraint can be observed in the fourth sprite (Fig. 6.22(d)). Here, the algorithm decided to use the last frame of the camera zoom-in as a reference to preserve the full resolution. This also explains why the algorithm separated the last 45 frames (256-300) into two separate sprites. If all frames would have been combined into a single sprite, all frames would be scaled up to preserve the resolution of the last frame. However, by splitting the sequence into two sprites, frames 256-292 could be coded with a lower resolution, which outweighs the overhead of an additional sprite.


Figure 6.14: The integration of the multi-sprite partitioning into the framework for background reconstruction.

### 6.8 Integration into the segmentation system

This section discusses how the multi-sprite partitioning algorithm can be integrated into the video-object segmentation system. The segmentation system that we started to construct in Chapters 3 to 5 already included all the steps required to synthesize background sprites from the input sequence. However, it comprised all the limitations that we described in the beginning of this chapter.

These limitations can be removed by integrating the multi-sprite partitioning algorithm into our framework. The goal is to take the motion parameters from the feature-based motion estimator and determine the multi-sprite partitioning. This information is then used in the background reconstruction stage to actually synthesize the sprite images from the specified frame ranges.

An overview of our multi-sprite enabled segmentation system is depicted in Figure 6.14. Processing starts with a feature-based global-motion estimator as described in Chapters 3 and 4 . This results in a set of approximate motion parameters which are used to compute the multi-sprite partitioning from this chapter. This partitioning defines from which input frames the respective background sprites are synthesized. The multi-sprite algorithm also computes the optimal reference frame, which is used in the refinement step for the long-term motion estimation. This motion parameters refine-


Figure 6.15: Online calculation of multi-sprite partitionings. After $s_{\max }$ frames (here $s_{\max }=7$ ) have been collected, the optimal partitioning is computed. The first frame-range is returned and the computation proceeds when the buffer fills again to $s_{\max }$ frames. Note also that graph-edges spanning less than $s_{\text {min }}$ frames have been disabled (dotted lines). In our example, $s_{\min }=3$.
ment step and the background synthetization is carried out as described in Chapter 5.

### 6.9 Online calculation of constrained sprites

The presented multi-sprite algorithm computes the optimal partitioning for the complete video sequence. Therefore, it has to consider the motion parameters for all frames in its computation. This has the disadvantage that the sequence has to be processed in at least two passes. The first pass computes the motion parameters and the multi-sprite partitioning, based on which the second pass can synthesize the sprite images. However, for many applications, online processing of the sequence is desired such that virtually infinite input sequences can be partitioned with only a small delay.

In these cases, it is convenient to limit the number of frames per sprite to a maximum of $s_{\text {max }}$. Furthermore, for reasons that will be described in Section 8.2.4, it can also be required to set a minimum number of frames $s_{\text {min }}$ that must be included in a sprite. The multi-sprite partitioning can be modified to an online algorithm as follows.

Instead of constructing the complete computation graph for the complete sequence, a graph covering only the first $s_{\max }$ frames is generated. All graph edges that span less than $s_{\text {min }}$ frames are omitted (or their cost is set to $\infty$ ). Note that this graph can be built online while new input frames are received. When $s_{\max }$ frames are available, the minimum-cost path is computed as before. This path again defines a partitioning, but now we only output the first frame-range. The subsequent processing stages can then begin to work on the sprite defined by this frame-range in parallel. As new input images arrive at the multi-sprite partitioning algorithm, it again constructs the graph for the next $s_{\max }$ frames, as shown in Figure 6.15.

This algorithm does generally not result in a globally optimal solution, but it limits the maximum memory requirement and processing delay that is introduced by the sprite generation to $s_{\max }$ frames.

### 6.10 Coding multi-sprites in MPEG-4 streams

The obtained background sprites can be coded as a sprite VOP with a standard MPEG-4 encoder. To transmit the multi-sprite, we have to consider that we will switch between several sprites. This switching is not addressed in the MPEG-4 standard, but two approaches are possible to simulate this without modifications in the MPEG-4 encoder and decoder. In the first approach, the sprites are transmitted sequentially, where a new sprite is sent just in time to show a continuous video at the decoder. However, this requires a high peak data-rate to send the new sprite. The second approach is to assemble the individual sprites into a single sprite image, where the sprites are placed independently beneath each other. To select the correct sprite at the decoder, the top-left corner position of the sprite is added to the camera-parameters in order to decode the correct sprite. In particular, if the sprite is stored at a displaced position $\left(t_{x}^{0}, t_{y}^{0}\right)$ in the sprite buffer, we send the motion parameters $\mathbf{H}^{\prime}$, which are computed as

$$
\mathbf{H}^{\prime}=\left(\begin{array}{ccc}
1 & 0 & t_{x}^{0}  \tag{6.16}\\
0 & 1 & t_{y}^{0} \\
0 & 0 & 1
\end{array}\right) \cdot \mathbf{H}
$$

where $\mathbf{H}$ are the original camera-motion parameters. This second approach does not require a high peak data-rate, but needs more decoder memory for

(a) Sequence of motion-compensated images.

(b) Frames that are synthesized into the background sprites.

Figure 6.16: Determining the frame ranges for background synthesis. (a) The video is depicted as a stack of motion-compensated frames. Along the borders of the sprite, foreground objects cannot be removed reliably, since too little temporal information is available. (b) To obtain an optimal suppression of foreground objects, the frame range of each sprite is extended to include more frames that overlap with the sprite area.
the sprite buffer. Future work might combine the multi-sprite partitioning with optimized decoder-buffer management, such that the decoder buffer for example always contains two sprites, where one is displayed while the other is updated.

### 6.11 Conclusions

This chapter has shown that partitioning a background sprite into several independent parts results in a clearly reduced coding cost and better resolution at the same time. Our algorithm computes the optimal partitioning of a sequence, the reference frame for each partition, and associated scaling factors. As a consequence, the proposed algorithm solves the subsequent problems.

- It removes the limitation of camera motion and enables to use sprite coding for arbitrary rotational camera motion.
- It selects optimal reference frames and defines multi-sprite partitions to considerably reduce the required amount of data for coding the background sprite, and
- it increases the reconstruction quality of the sprite.

The above features are achieved while remaining fully compatible to the MPEG-4 standard.

Clearly, the reduction of sprite area depends on the type of camera motion in the sequence. For e.g. the stefan sequence, a reduction by a factor of at least 2.6 has been achieved. Moreover, note that the proposed algorithm can synthesize sprites for all kinds of camera motion, which cannot be guaranteed with previous approaches. The ability to handle arbitrary rotational camera motion is an important generalization to previous spriteconstruction algorithms which in these cases would simply fail to create the sprite. Moreover, the generalization is not only important for the coding of the background sprite, but also for other image analysis algorithms like our video-object segmentation, which is based on background subtraction. This algorithm also requires a complete coverage of the background environment as obtained from the computed set of background images.

(c) Frames $78-132,585 \times 478$.

Figure 6.17: Multi-sprite synthesized from a long zoom-out operation. The sequence is partitioned into three separate sprites of almost the same size. The center image has been selected as the reference coordinate system (shown in a darker shade). For comparison, a single-part sprite generated from the same sequence would result in a size of $1687 \times 1516$.

(a) Frames 1-45, $541 \times 445$, area: 170k.

(b) Frames 46-82, $479 \times 446$, area: 171 k .

(c) Frames 83-140, $455 \times 387$, area: 152 k .

Figure 6.18: Multi-sprite for the rail sequence. The sequence shows a camera rotation around two axes at the same time. At the beginning of the sequence, the camera is looking down. It turns left and around its optical axis until it looks left in the last frame. For each sprite, the covered sprite area is indicated in 1000-pixel units and the size of the bounding-box. Reference frames are depicted in a darker shade.


Figure 6.19: Frames 1-82 integrated into a single background sprite (1264×592, area: 427k). The attempt to integrate the entire rail sequence into a single background sprite fails because of the complicated camera motion. The camera performs an approximate $90^{\circ}$ rotation around two axes.


Figure 6.20: Super-resolution effect. Since many input frames are integrated in the background synthesis step based on an accurate motion-model, a high-resolution image can be derived from a sequence of low-resolution images.


Figure 6.21: Sprite synthesized from stefan sequence. Only the first 255 frames can be used, since it is impossible to create the sprite for the complete sequence if the first frame is selected as reference. Sprite resolution is $2445 \times 1026$ pixels, area: 1208 k .


In theory, there is no difference between theory and practice. But in practice, there is.
(Manfred Eigen)

## Chapter <br> 7

## Background Subtraction

When the background view excluding the foreground objects is available, it becomes obvious that the foreground objects can be obtained by comparing the background image with the current video frame. In practice, camera noise and regions in which the object has the same color as the background make the separation of foreground objects and background more difficult. The previous chapters showed how this background image is reconstructed from a video sequence, and how the camera motion can be compensated. This chapter discusses the background-subtraction module, which determines the segmentation masks that are the final output of the segmentation system. The chapter commences with simple independent pixel classification and then proceeds to more complex tests that include contextual information to decrease the number classification errors. Furthermore, typical problems that lead to segmentation errors are identified and a modification to the segmentation algorithm is provided to reduce these effects. Finally, a few postprocessing filters are presented that can remove obvious errors like small clutter regions.

Chapter 7. Background Subtraction

### 7.1 Introduction

Chapter 5 described how a pure background image of a video scene can be synthesized from the input video itself. By combining many video frames into this background image, we are able to reconstruct the background image excluding the foreground objects. Such a background image can be elegantly used to determine the foreground objects by comparing the input frame with the background image and mark the differences as foreground objects. This technique is commonly known as background subtraction or change detection. It is the most popular approach in video surveillance applications, because it is a computationally efficient technique and it is relatively easy to obtain background images for static surveillance cameras.

## Previous work

The change-detection problem has been studied for a variety of applications, where surveillance, medical diagnosis, and remote sensing are currently the most important. In all of these applications, the objective is to compare an image to a background image and identify the regions that have changed. Unfortunately, the exact definition of what is actually meant with changed depends on the application and cannot be generalized.

A large number of change-detection operators have been proposed and good surveys about available techniques can be found in [41, 152, 157, 19]. Apart from independent-pixel classification schemes, algorithms have been developed that integrate contextual information to improve the robustness. Prominent techniques are classifiers using statistical models of the pixel neighborhood [1, 202], Markov Random Field (MRF) based models [101, $15,14,3,98]$, or algorithms combining intensity and texture differences [113].

Since surveillance is a major application, many algorithms are designed to be invariant to changes in global illumination. A standard technique to approach the problem of varying illumination is to model the luminance distribution of each background pixel with one or several Gaussian distributions [174]. However, since we consider only short video sequences in our application, the problem of global illumination changes is usually neglegible.

Another common difficulty is the problem of image misregistration. It has been shown in [33] that even small registration errors of less than 0.2 pixels can lead to about $10 \%$ of additional false detections. As a solution, it has been proposed in [13] to estimate the expected distribution of misregistration noise and adapt the distance measure accordingly.

## Chapter outline

This chapter gradually develops a robust background subtraction algorithm in a number of steps. We start with independent classification of the input pixels, for which we compare different metrics to detect changed image content. It will be shown that classifying the pixels independently does not provide robust results, since objects often have colors similar to the background and there is also a considerable amount of noise in the input video. Both effects cause misclassification errors, namely foreground pixels that are not detected, and background pixels that are detected as changed. To enhance the robustness, we implement a classification scheme [1] that considers a neighborhood of pixels at once, using a $\chi^{2}$ significance test. This approach is further refined to using a Markov Random Field model, allowing to integrate pre-knowledge about the object shape. Finally, we propose a few modifications to reduce problems caused by image misregistration and we add morphological postprocessing operations to optimize the object masks.

### 7.2 Pixel-based classification

Change detection can be carried out on either greyscale information only, or using the full color information. Many algorithms in the literature are described for the greyscale case, but it is obvious that the robustness can be increased by examining all three color channels (e.g., consider two differently colored objects with equal luminance).

### 7.2.1 Distance metrics

We denote the pixel values in the RGB color-space as triples $\mathbf{I}^{R G B}=$ $\left(I^{R}, I^{G}, I^{B}\right)$ and in the YUV color-space as $\mathbf{I}^{Y U V}=\left(I^{Y}, I^{U}, I^{V}\right)$. The background and input image are indicated with a subscript, such that $I_{B}$ denotes the background image and $I_{t}$ denotes the current input image.

The following distance metrics are implemented:

- the pixel difference of only the luminance channel

$$
\begin{equation*}
d_{y}=\left|I_{t}^{Y}-I_{B}^{Y}\right|, \tag{7.1}
\end{equation*}
$$

- the sum of pixel differences of the three color channels

$$
\begin{equation*}
d_{1}^{R G B}=\left|I_{t}^{R}-I_{B}^{R}\right|+\left|I_{t}^{G}-I_{B}^{G}\right|+\left|I_{t}^{B}-I_{B}^{B}\right|, \tag{7.2}
\end{equation*}
$$

- the sum of squared pixel differences of the three color channels

$$
\begin{equation*}
\left(d_{2}^{R G B}\right)^{2}=\left|I_{t}^{R}-I_{B}^{R}\right|^{2}+\left|I_{t}^{G}-I_{B}^{G}\right|^{2}+\left|I_{t}^{B}-I_{B}^{B}\right|^{2} \tag{7.3}
\end{equation*}
$$

- the maximum pixel difference in the three color channels

$$
\begin{equation*}
d_{\infty}^{R G B}=\max \left\{\left|I_{t}^{R}-I_{B}^{R}\right|,\left|I_{t}^{G}-I_{B}^{G}\right|,\left|I_{t}^{B}-I_{B}^{B}\right|\right\}, \tag{7.4}
\end{equation*}
$$

- and the Mahalanobis distance in YUV space

$$
\begin{equation*}
\left(d_{M}^{Y U V}\right)^{2}=\left(\mathbf{I}_{t}^{Y U V}-\mathbf{I}_{B}^{Y U V}\right) \mathbf{S}^{-1}\left(\mathbf{I}_{t}^{Y U V}-\mathbf{I}_{B}^{Y U V}\right)^{\top} \tag{7.5}
\end{equation*}
$$

where $\mathbf{S}$ is the $3 \times 3$ covariance matrix of the color differences. It is discussed later how the entries of this matrix are chosen. Note that the three metrics $d_{1}, d_{2}, d_{\infty}$ correspond to the $L_{1}, L_{2}$, and $L_{\infty}$ norms. Additionally to the RGB space, these three metrics are also implemented in the YUV space, where they are denoted as $d_{1}^{Y U V}, d_{2}^{Y U V}, d_{\infty}^{Y U V}$. All proposed metrics have in common that their values increase when the color $\mathbf{I}_{t}$ in the current image deviates more from the reference color $\mathbf{I}_{B}$. This enables a classification in which a pixel is noted as changed (foreground) if the difference to the background exceeds a threshold $\tau$. For example, using the greyscale difference, we obtain the decision function $d_{y}>\tau$.

### 7.2.2 Influence of the color-space

The previous section has defined a number of metrics for measuring the distance between two colors. It is not obvious, which of these metrics will yield the most accurate result in combination with a certain colorspace. Theoretically, the choice of the color-space has no influence on the classification accuracy. To understand this, assume that we have a best reference metric in some color-space. This metric together with a threshold defines a decision boundary in this color-space. Now we take an arbitrary different color-space. If there is a transform from the first color-space to the second, we can also transform the decision boundary into the second colorspace, so that we realize a classification function of similar performance in every color-space.

In practice, the question is for which color-space a computationally simple metric can be defined that leads to a good classification. Primarily, this depends on the distribution of noise in the color-space. Let us assume that the noise is mainly a variation of luminance, while the hue of the color remains stable. Consequently, the noise distribution in the YUV space will have its largest variance along the Y -axis and it will have considerably

(b) YUV color-space.

Figure 7.1: Distribution of color-differences for the stefan sequence, separately for changed and unchanged pixels. The images show projections of the 3-D distribution.
smaller variance in the U and V dimension (see Figure 7.1). In the RGB color-space, the noise distribution is oriented along the diagonal vector $(1,1,1)$. This is of course a simplification, because the orientation of the distribution will vary for different colors. However, we can still make use of it, since the majority of the colors in natural images are usually not very saturated.

Let us now discuss how the isosurfaces of constant cost are defined by the different metrics. Since the $L_{2}$ metric equals Euclidean distance, the isosurfaces of constant cost are spheres. For $L_{\infty}$, they are axis-parallel cubes and for $L_{1}$, they are cubes with corners on the axes. Selecting a


Figure 7.2: Isolines of constant cost for $L_{\infty}$ in the $R G B$ space (a), and for $L_{1}$ in the YUV space (b). Displayed is a cut through the three-dimensional color-space such that the reference background color is at the origin. The optimal metric to separate a foreground color from the background distribution depends on the orientation of the background distribution.
specific threshold corresponds to selecting one of the isosurfaces to separate the class of background colors (inside of the closed isosurface) from the foreground-colors (outside space).

Figure 7.2 depicts typical noise distributions in the RGB and the YUV color-space. It can be concluded that the distribution along the diagonal (RGB space) can be enclosed most tightly with the $L_{\infty}$ norm. It is easy to note that using the $L_{1}$ or $L_{2}$ metrics would enclose more foreground colors when the complete background distribution should still be contained. The same argument is also valid in the YUV-space, in which the $L_{1}$ norm provides a good separation.

In both color-spaces, neither of the $L_{1}, L_{2}$, or $L_{\infty}$ metrics enclose the background distribution optimally. However, assuming that the noise distribution is a multi-variate gaussian and provided that its covariance matrix is known, the results can be improved by using the Mahalanobis distance metric. In the case of the YUV space, this is especially easy, since it can be assumed that the luminance is independent from the two chrominance dimensions. Consequently, the distribution is axis-parallel and the covariance matrix is diagonal $\mathbf{S}=\sigma^{2} \operatorname{diag}\left(1, \beta^{2}, \beta^{2}\right)$ with the standard deviation $\sigma$ in the luminance channel and $\beta \cdot \sigma$ in the color channels. This gives the

(a) Thresholded greyscale difference.

(b) Thresholded RGB error maximum.

Figure 7.3: Background differencing using independent pixels (stefan sequence, frame 6).
particularly simple formulation of $d_{M}^{Y U V}$ as

$$
\begin{equation*}
\left(d_{M}^{Y U V}\right)^{2}=\frac{1}{\sigma^{2}}\left(I_{t}^{Y}-I_{B}^{Y}\right)^{2}+\frac{1}{\sigma^{2} \beta^{2}}\left(\left(I_{t}^{U}-I_{B}^{U}\right)^{2}+\left(I_{t}^{V}-I_{B}^{V}\right)^{2}\right) \tag{7.6}
\end{equation*}
$$

Because a change of $\sigma^{2}$ can be compensated by adapting the threshold $\tau$, the standard deviation can be arbitrarily set $\sigma=1$. Note that this gives basically the equation for Euclidean distance with an additional scaling factor for the chrominance components. According to our experiments, good results are obtained for $\beta^{2} \approx 0.1$.

### 7.2.3 Classes of errors

Example results of independent pixel classification are depicted in Figure 7.3. Two kinds of errors can be identified in the results.

1. Pixels that actually belong to the background, but which are classified as foreground, because of camera noise or misregistration errors.
2. Pixels that belong to the foreground object, but which are classified as background, because their color is very similar to the background at the same position.

It is not possible to minimize both errors at the same time. Depending on the decision threshold, it is only possible to reduce one type of error while simultaneously increasing the other type of error.


Figure 7.4: Example of a manually generated reference mask. Grey areas indicate foreground regions, black areas represent don't care regions that are excluded from the evaluation of segmentation results.

### 7.2.4 Evaluation method

To be able to quantify the quality of the obtained segmentation masks, we manually created reference masks for several sequences. Because an object often has a blurred boundary or a soft shadow, it is difficult to define the correct border of an object. We therefore separated the pixels in our reference masks into three different classes: Foreground pixels, background pixels, and don't care pixels that are excluded from the evaluation (see Fig. 7.4(b)).

To evaluate the quality of the segmentation algorithms, we compare the obtained segmentation masks with the reference masks by counting the percentage $e_{b}$ of incorrectly classified pixels in the background region and the percentage of incorrectly classified pixels in the foreground region $e_{f}$. Note that both error percentages are depending on the chosen threshold. Lowering the threshold will lead to less pixels that are classified as foreground and hence, it will decrease $e_{b}$, but at the same time, it will increase $e_{f}$. Increasing the threshold has the opposite effect.

The dependence of the two error classes for different thresholds are collected in an ROC (Receiver Operating Characteristic) curve [150]. This curve depicts the relation between the two errors that the algorithm yields for different thresholds (see Fig. 7.5). An ideal segmentation algorithm would succeed to reach $100 \%$ correct foreground pixels and $100 \%$ correct background pixels at the same time. Practically, this ideal case cannot be
obtained.
In general, two ROC curves for different algorithms are not necessarily better or worse along the whole curve. It is well possible that two curves intersect. To still quantify the accuracy of a segmentation algorithm, we measure its performance using the area under the ROC curve.


Figure 7.5: ROC curves for single pixel classification. Because the stefan sequence comprises more saturated colors, the difference between the greyscale metric and the color metrics are larger than for the surveillance sequence.

### 7.2.5 Results

The performance of all pixel-based classification metrics on the three sequences stefan, surveillance, and hall-and-monitor was measured. A description about the content of these sequences can be found in Appendix D.

## Greyscale vs. color metrics

The first observation is that color-based metrics provide higher accuracy than the greyscale metric $d_{y}$ (see the ROC curves in Fig. 7.5). This is not surprising, since each colored pixel comprises three channels which should all be non-changing when background is shown, whereas a change in one of the color channels is sufficient to detect a foreground object. However, the advantage of a color-based metric over a greyscale metric diminishes if the sequence shows mostly low-saturated colors. This can be observed when


Figure 7.6: Area under ROC curve for greyscale and color-based distance metrics.
comparing the colorful stefan with the low-saturated surveillance sequence. While the difference between the metrics is well visible for the colorful stefan sequence, it is smaller for the surveillance scene.

## Influence of the color-space

We claimed in Section 7.2.2 that from the implemented metrics, the $L_{\infty}$ metric should give the most accurate results in the RGB-space. In the YUVspace, the best results should be obtained using the $L_{1}$ or, even better, the Mahalanobis distance.

This predicted behaviour can indeed be seen in Fig. 7.6. For RGB, $L_{\infty}$ gives the best results, followed by $L_{2}$ and $L_{1}$ at the end. For YUV, it is just the opposite, with the Mahalanobis distance giving the best results, followed by $L_{1}, L_{2}$, and $L_{\infty}$.

### 7.3 Multi-pixel based significance tests

Up to this point, the pixels in the image were considered to be independent. However, since objects are compact regions of foreground pixels, the probability that a pixel belongs to a specific class increases with the number of pixels of the same class in its neighborhood. This regularity can be exploited to increase the robustness of the segmentation algorithm. Two
main approaches are possible. The first assumes that all pixels in a small neighborhood belong to the same class (either foreground or background). With this assumption, the measurements in the pixel neighborhood can be combined to deduce a more robust decision function. The second approach uses a Markov Random Field model to define $a$-priori probabilities for the pixel classification based on the class of its neighborhood pixels. Both approaches will be discussed in the current and the subsequent section, respectively, as they are both used in our final segmentation system.

### 7.3.1 Classification using a $\chi^{2}$ test

If the change detection is based only on observations of individual pixels, it is often not possible to detect whether small color differences are the consequence of a foreground object or if they are only camera noise. To make change detection more robust, the neighborhood of a pixel should be included into the decision. Let us introduce this approach by reviewing the technique proposed in [1] for greyscale images and Gaussian noise. Subsequently, we will enhance this algorithm to color images.

Let us denote the difference between the current input greyscale image and the background as $D^{Y}=I_{t}^{Y}-I_{B}^{Y}$. For simplicity, we omit the superscript for $D$ in the following. In the case that the pixel did not change (null-hypothesis $H_{0}$ ), we assume that it is subject to Gaussian noise with variance $\sigma^{2}$, so that the pixel difference satisfies the distribution

$$
\begin{equation*}
p\left(D \mid H_{0}\right)=\frac{1}{\sqrt{2 \pi \sigma^{2}}} \exp \left\{-\frac{D^{2}}{2 \sigma^{2}}\right\} . \tag{7.7}
\end{equation*}
$$

It is a valid assumption to use Gaussian noise, since the pixel differences are only a consequence of the camera noise. Another distribution that is frequently used in this context is the Laplace distribution. However, the Laplace distribution is usually considered as the distribution of prediction errors. In fact, we conducted experiments which showed that the real distribution is in-between Gaussian and Laplace. Since the variance of the noise is very small, both distributions can be used with comparable results. ${ }^{1}$

Let us now observe a window $w$ centered at the considered pixel. We construct a vector $\mathbf{D}$ of all pixel differences within the window as $\mathbf{D}=$ $(D(x, y))_{(x, y) \in w}$. Since the pixel labels have a high spatial correlation, we employ the simple assumption that the center pixel of the window is classified as unchanged only if every pixel in the window has a low difference

[^15]

Figure 7.7: In the multi-pixel classification test, we label a pixel as unchanged only if all pixels in the neighborhood window have small differences. This leads to the correct decision for pixels $A$ and $B$ (dark pixels denote object pixels with large difference values). However, pixel $C$ will be erroneously classified as object pixel. Consequently, the detected object will be slightly larger than the true object.
value. This assumption is valid for most areas of the image, but obviously, it is not true along the object boundaries (Fig. 7.7). The defects that are introduced along the object boundary are eliminated in the subsequent processing stage that is described in Section 7.4.

Under the assumption that all pixels in the window are unchanged and their noise is independent, the probability distribution becomes

$$
\begin{equation*}
p\left(\mathbf{D} \mid H_{0}\right)=\left(\frac{1}{\sqrt{2 \pi \sigma^{2}}}\right)^{N_{w}} \exp \left\{-\sum_{(x, y) \in w} \frac{D(x, y)^{2}}{2 \sigma^{2}}\right\} \tag{7.8}
\end{equation*}
$$

where $N_{w}$ denotes the number of pixels within the window $w$. Because it is very difficult to deduce any property of the probability distribution for the case of the counter-hypothesis $H_{1}$ that a pixel has changed, we use a significance test based on $p\left(\mathbf{D} \mid H_{0}\right)$. Notice that this distribution can also be written as a function of the sum of squared pixel differences

$$
\begin{equation*}
\Delta=\sum_{(x, y) \in w}(D(x, y) / \sigma)^{2} . \tag{7.9}
\end{equation*}
$$

Assuming that changed foreground pixels will show larger pixel differences, we will put a threshold $t_{\alpha}$ on $\Delta$ such that a pixel is detected as changed, if $\Delta>t_{\alpha}$. Hence, the probability of obtaining a false positive equals $P(\Delta>$ $\left.t_{\alpha} \mid H_{0}\right)$. Now, we can choose a significance level $\alpha$ defined as

$$
\begin{equation*}
P\left(\Delta>t_{\alpha} \mid H_{0}\right)=\alpha, \tag{7.10}
\end{equation*}
$$

from which we can deduce $t_{\alpha}$. Since $\Delta$ is a sum of Gaussian-distributed random variables, $\Delta$ itself is distributed according to a $\chi^{2}$ distribution with
$N_{w}$ degrees of freedom. The complete classification process based on the $\chi^{2}$ test can be summarized as follows.

- Choose an observation window $w$, a significance level $\alpha$, e.g. $10^{-5}$, and set an appropriate standard deviation of the image noise $\sigma \approx 10$.
- Use the inverse cumulative function of the $\chi^{2}$ distribution to obtain a threshold $t_{\alpha}$ according to Eq. (7.10).
- Iterate through the picture and compute the sum of squared differences $\Delta$ within each neighborhood window. Set the center pixel to changed if $\Delta>t_{\alpha}$.


### 7.3.2 Extension to color images

The above significance test was designed for greyscale images, but Section 7.2 revealed that our results can be improved if the color information is included into the change detection algorithm. This can be achieved in a straight-forward way by adding the color channel values as extra pixels to the difference vector $\mathbf{D}$, so that it is extended to three times its original length:

$$
\begin{equation*}
\mathbf{D}=(\underbrace{D^{Y}(x, y), \ldots}_{(x, y) \in w}, \underbrace{D^{U}(x, y), \ldots}_{(x, y) \in w}, \underbrace{D^{V}(x, y), \ldots}_{(x, y) \in w}) \tag{7.11}
\end{equation*}
$$

Clearly, this means that the threshold has to be determined using the $\chi^{2}$ distribution with $3 \cdot N_{w}$ degrees of freedom. Additionally, two important aspects are considered. First, we have to ensure that the differences in the two color channels are statistically independent to the luminance channel values and also between the color channels themselves. This can be considered true if we operate in the YUV color-space, but it is not valid for the RGB color-space, as we saw earlier. Therefore, we only consider the YUV color-space in the following. Second, the variances for difference signals in the luminance channels and the chrominance channels are different. This means that we should replace the constant $\sigma$ in the monochrome case with $\sigma$ for the luminance values and $\beta \cdot \sigma$ for the chrominance values. This leads to a modified definition of $\Delta$, so that

$$
\begin{align*}
\Delta & =\sum_{(x, y) \in w} \frac{1}{\sigma^{2}} D^{Y}(x, y)^{2}+\frac{1}{\sigma^{2} \beta^{2}}\left(D^{U}(x, y)^{2}+D^{V}(x, y)^{2}\right)  \tag{7.12}\\
& =\sum_{(x, y) \in w}\left(d_{M}^{Y U V}(x, y)\right)^{2} \tag{7.13}
\end{align*}
$$

which is based on the Mahalanobis metric for the YUV color-space.


Figure 7.8: The sum over the dark area can be computed from the cumulative sums $C\left(x_{1}, y_{1}\right)-C\left(x_{0}-1, y_{1}\right)-C\left(x_{1}, y_{0}-1\right)+C\left(x_{0}-\right.$ $\left.1, y_{0}-1\right)$.

### 7.3.3 Fast implementation

A direct implementation of the above algorithm becomes inefficient for larger window sizes. To reduce the complexity, we propose to use the technique of cumulative sums [190]. Instead of computing the sum over a rectangular window $w$ directly from the pixel costs $d_{M}^{Y U V}(x, y)^{2}$, we first compute the cumulative costs

$$
\begin{equation*}
C(x, y)=\sum_{i=0}^{x} \sum_{k=0}^{y}\left(d_{M}^{Y U V}(i, k)\right)^{2} . \tag{7.14}
\end{equation*}
$$

Each $C(x, y)$ equals the sum of all pixel costs in the rectangle starting at the top-left corner and the bottom-right corner at $(x, y)$. Note that $C(x, y)$ can be computed iteratively with two passes over the cost image. In the first pass, costs are cumulated in the horizontal direction, followed by a second pass in the vertical direction. When the cumulative costs $C(x, y)$ are available, the sum over an arbitrary rectangular area with top-left position ( $x_{0}, y_{0}$ ) and bottom-right position ( $x_{1}, y_{1}$ ) can be obtained in constant time with (see Fig. 7.8)
$C\left(x_{0}, y_{0}, x_{1}, y_{1}\right)=C\left(x_{1}, y_{1}\right)-C\left(x_{0}-1, y_{1}\right)-C\left(x_{1}, y_{0}-1\right)+C\left(x_{0}-1, y_{0}-1\right)$.

### 7.3.4 Evaluation

Two questions arise for the classification algorithms using the significance test. First, how does the quality of the result depend on the size of the neighborhood window, and second, does the inclusion of color information improve the segmentation result? To illustrate the typical segmentation


Figure 7.9: Background differencing using significance tests on YUV color (stefan sequence, frame 80). Compare the reduced noise in the mask compared to an independent pixel classification with the Mahalanobis metric. But also note the increasing aura around the object larger window sizes.
masks that result from different window sizes, Figure 7.9 portrays the outcome of the color-based significance test.

## Influence of window size

The immediate observation is that the amount of small pixel-noise is significantly reduced when using even a small $3 \times 3$ window, instead of the independent pixel classification. The amount of background errors is further reduced by applying larger window sizes, but simultaneously, a disadvan-


Figure 7.10: Area under ROC curve for the color-based significance test with different window sizes. Each bar is divided into two parts. The lower (brighter) part shows the result using the greyscale distance measure, while the upper (darker) part shows the result for using the Mahalanobis color distance.
tageous effect becomes more apparent. Foreground objects are surrounded by a growing aura of pixels that are falsely attributed to the foreground object.

The reason for this effect is the initial assumption that a pixel is only classified as unchanged if every pixel in the window is unchanged. On the other hand, this means that the center pixel is classified as changed even if some pixels at arbitrary position in the window are changed, but not the center pixel itself. Consequently, foreground pixels at the object boundary have influence even on pixels that are further away from the object.

The same effect also leads to closing small holes in the segmentation mask. Even though closing these holes seems to be a good effect, this is not so important, since this can also be achieved with a simple post-processing step (will be described in Section 7.6.1).

We evaluated the quality of the obtained segmentation masks by comparing them to our reference masks in the same way as for the single-pixel classification case. Finally, we computed the ROC curves (see Fig. 7.11 for an example) and measured the area under the ROC curves (Fig. 7.10).

It can be noticed that the accuracy of the segmentation in fact increases with larger window sizes, where the maximum is at about $9 \times 9$ pixels. However, in this case, this evaluation with the ROC-area as a quality criterion is


Figure 7.11: $R O C$ curves for $\chi^{2}$ significance test with varying window size (hall-and-monitor).
partly misleading because of two reasons. First, remember that the objects in the reference masks have an aura of don't-care pixels surrounding them, meaning that segmentation errors in these regions are not counted. When using a large neighborhood window, it appears that the masks grow larger than the objects. However, this error is not detected during the comparison with the reference masks, so that the quality measure still reports good results. For larger windows, the results start to deteriorate (Fig. 7.11). This phenomenon only occurs after the object aura extends beyond the don't-care areas of the reference masks.

Second, the background is typically larger than the objects, which means that false background pixels as in the object aura count much less than missing pixels in the object mask. This does not match the subjective quality, where errors of both classes count as equally imporant.

## Greyscale vs. color

As expected, Fig. 7.11 shows that color information gives an additional gain to the segmentation accuracy. However, it can also be deduced that the differences are not as large as before, since good results can also be obtained
using only greyscale information. Especially for the hall-and-monitor sequence, the color information seems to have almost no influence on the result. It is surprising that adding the color information does not make a big difference for the stefan sequence, because we obtained big improvements for this sequence with single-pixel decision functions. However, it should be noted that the results for stefan are already very close to the optimum.

### 7.4 Classification using Markov random fields

The results of the statistical-significance algorithm showed that the segmentation can in fact be improved by incorporating contextual information into the decision process. While reducing the noise in the segmentation mask, the previously discussed algorithm has the main disadvantage that the positions of the object boundaries are not preserved. The obtained segmentation masks are extended beyond the true object boundary, resulting in the aura effect.

### 7.4.1 MRF model for segmentation masks

A change detection algorithm that is based on a Markov Random Field (MRF) model for the segmentation mask has been proposed in [2]. The idea is to determine the segmentation mask as a Maximum A-Posteriori (MAP) approximation, in which the $a$-priori probability of the segmentation masks are modeled as Gibbs/Markov Random Fields. We will review the model in this section, extend it to color images, and discuss its implementation.

Let $Q=\{q(x, y)\}$ be the segmentation mask, where $q(x, y) \in\{u, c\}$ is the label of one pixel. We label an unchanged (background) pixel as $q(x, y)=u$ and a changed pixel (foreground) as $q(x, y)=c$. We model the $a$-priori probability of a segmentation mask as a Gibbs field [114] with second-order cliques. To define the clique potentials, we further divide the second-order cliques into the set of horizontal/vertical cliques

$$
\begin{equation*}
\mathcal{C}_{h v}=\{\{(i, k),(x, y)\}| | i-x|+|k-y|=1\} \tag{7.16}
\end{equation*}
$$

and the set of diagonal cliques

$$
\begin{equation*}
\mathcal{C}_{\text {diag }}=\{\{(i, k),(x, y)\}| | i-x|=|k-y|=1\} \tag{7.17}
\end{equation*}
$$

These two sets of cliques are illustrated in Fig. 7.12. Let us define the probability of a given segmentation mask $Q$ by

$$
\begin{equation*}
p(Q)=\frac{1}{Z} \exp \{-U\} \tag{7.18}
\end{equation*}
$$



Figure 7.12: The cliques set $\mathcal{C}_{h v}$ comprises the all direct horizontal/vertial neighbors, while $\mathcal{C}_{\text {diag }}$ comprises diagonal neighbors.
where the constant $Z$ normalizes the sum of probabilities to unity. However, it will be shown in the sequel that the value of $Z$ is not required for the resulting computations. The properties of the segmentation masks are modeled with the energy function $U$ as

$$
\begin{equation*}
U=\sum_{\left\{(x, y),\left(x^{\prime}, y^{\prime}\right)\right\} \in \mathcal{C}_{h v}} \gamma_{h v} \cdot V\left(q(x, y), q\left(x^{\prime}, y^{\prime}\right)\right)+ \tag{7.19}
\end{equation*}
$$

where the function $V\left(q, q^{\prime}\right)$ is defined as

$$
V\left(q, q^{\prime}\right)=1-\delta\left(q, q^{\prime}\right)= \begin{cases}1 & \text { if } q \neq q^{\prime}  \tag{7.21}\\ 0 & \text { if } q=q^{\prime}\end{cases}
$$

Consequently, the parameters $\gamma_{h v}$ and $\gamma_{\text {diag }}$ control the regularity of neighbored labels. Setting both values to zero results in equally probably segmentation masks ( $p(Q)=$ const), which leads to an independent pixel classification. Higher values for $\gamma_{h v}, \gamma_{\text {diag }}$ increase the probability of masks in which neighboring pixels have the same label. Consequently, the resulting segmentation masks will have smoother boundaries, but small details can be lost. See Table 7.1 for the values that we used during our experiments.

### 7.4.2 Obtaining a MAP estimate

Let us now consider a single pixel $(x, y)$. We want to know which state (changed or unchanged) is more probable, given the difference image $d(x, y)$
and keeping the remaining segmentation mask fixed. In other words, we want to know if

$$
\begin{equation*}
p(q=u \mid d) \gtrless_{c}^{u} p(q=c \mid d) \tag{7.22}
\end{equation*}
$$

where the pixel coordinates are omitted for brevity. Using the Bayes rule, we can rewrite this as

$$
\begin{equation*}
\frac{p(d \mid q=u) \cdot p(q=u)}{p(d)} \gtrless_{c}^{u} \frac{p(d \mid q=c) \cdot p(q=c)}{p(d)} \tag{7.23}
\end{equation*}
$$

or simply

$$
\begin{equation*}
p(d \mid q=u) \cdot p(q=u) \gtrless_{c}^{u} p(d \mid q=c) \cdot p(q=c) \tag{7.24}
\end{equation*}
$$

To solve this, an assumption about $p(d \mid q)$ is needed. In the case of unchanged pixels $(q=u)$, we model this as Gaussian noise with variance $\sigma^{2}$ according to Eq. (7.7). We cannot derive much about the case of changed pixels so that we model this also with a Gaussian distribution, but with a much larger $\sigma_{c}$. Finally, we determine the probabilities $p(q)$, where $Q$ is fixed with the exception of this one pixel at $(x, y)$. Since we modeled the segmentation mask with a Gibbs random field, the influence of one pixel is limited. The total field probability is computed from all cliques, but the number of cliques that are influenced by the choice of one pixel is small (see Fig. 7.12). This allows us to reorder the two sums in Eq. (7.20) into two new sums, namely one sum over all cliques $\mathcal{C}^{0}$ that are not affected by the choice of the pixel (the majority), and the sum over cliques $\mathcal{C}^{1}$ that are affected by the pixel. This results in the desired probability

$$
\begin{equation*}
p(q)=\underbrace{\frac{1}{Z} \cdot \exp \left\{-\sum_{\mathcal{C}^{0}} \cdots\right\}}_{\text {unaffected by label of } q} \cdot \underbrace{\exp \left\{-\sum_{\mathcal{C}^{1}} \cdots\right\}}_{\text {depending on label of } q} \tag{7.25}
\end{equation*}
$$

The second sum can be written as

$$
\begin{equation*}
\sum_{\mathcal{C}^{1}} \cdots=n_{h v} \gamma_{h v}+n_{d i a g} \gamma_{d i a g} \tag{7.26}
\end{equation*}
$$

where $n_{h v}$ is the number of horizontally or vertically neighboring pixels with a different label, and $n_{\text {diag }}$ is the number of diagonally neighboring pixels with different labels (see Fig. 7.13(b)). Since we consider both cases of an unchanged and a changed center pixel, we denote the number of inhomogeneous cliques as $n_{h v}^{u}, n_{d i a g}^{u}$ and $n_{h v}^{c}, n_{d i a g}^{c}$, respectively. Inserting (7.25) and (7.26) into (7.24) results in

$$
\begin{equation*}
\frac{p(d \mid q=u) \cdot \frac{1}{Z} \exp \left\{-\sum_{\mathcal{C}^{0}} \cdots\right\} \exp \left\{-n_{h v}^{u} \gamma_{h v}-n_{\text {diag }}^{u} \gamma_{d i a g}\right\}}{p(d \mid q=c) \cdot \frac{1}{Z} \exp \left\{-\sum_{\mathcal{C}^{0}} \cdots\right\} \exp \left\{-n_{h v}^{c} \gamma_{h v}-n_{\text {diag }}^{c} \gamma_{\text {diag }}\right\}} \gtrless_{c}^{u} 1 \tag{7.27}
\end{equation*}
$$


(a) Local influence of a pixel.

(b) Homogeneous and inhomogeneous cliques.

Figure 7.13: (a) The state of one pixel has only influence on the eight indicated clique potentials. (b) In this example configuration, there is one inhomogeneous vertical clique $\left(n_{h v}=1\right)$ and three inhomogeneous diagonal cliques $\left(n_{\text {diag }}=3\right)$.
which can be simplified to

$$
\begin{equation*}
\frac{p(d \mid q=u) \cdot \exp \left\{-n_{h v}^{u} \gamma_{h v}-n_{d i a g}^{u} \gamma_{d i a g}\right\}}{p(d \mid q=c) \cdot \exp \left\{-n_{h v}^{c} \gamma_{h v}-n_{d i a g}^{c} \gamma_{d i a g}\right\}} \gtrless_{c}^{u} 1 \tag{7.28}
\end{equation*}
$$

Inserting Gaussian distributions for $p(d \mid q)$ leads to

$$
\begin{equation*}
\frac{1 / \sqrt{2 \pi \sigma^{2}} \exp \left\{-d^{2} /\left(2 \sigma^{2}\right)-n_{h v}^{u} \gamma_{h v}-n_{\text {diag }}^{u} \gamma_{\text {diag }}\right\}}{1 / \sqrt{2 \pi \sigma_{c}^{2}} \exp \left\{-d^{2} /\left(2 \sigma_{c}^{2}\right)-n_{h v}^{c} \gamma_{h v}-n_{\text {diag }}^{c} \gamma_{d i a g}\right\}} \gtrless_{c}^{u} 1 \tag{7.29}
\end{equation*}
$$

and after taking the logarithms, we obtain the final decision function

$$
\begin{equation*}
d^{2} \gtrless_{u}^{c} 2 \frac{\sigma_{c}^{2} \sigma^{2}}{\sigma_{c}^{2}-\sigma^{2}}\left(\ln \frac{\sigma_{c}}{\sigma}+\left(n_{h v}^{c}-n_{h v}^{u}\right) \gamma_{h v}+\left(n_{\text {diag }}^{c}-n_{\text {diag }}^{u}\right) \gamma_{d i a g}\right) \tag{7.30}
\end{equation*}
$$

The right-hand side of this equation is effectively an adaptive threshold that depends on the local neighborhood. If the number of inhomogeneous cliques is the same, independent of the state of the center pixel ( $n_{h v}^{c}=n_{h v}^{u}$ and $\left.n_{\text {diag }}^{c}=n_{\text {diag }}^{u}\right)$, the neighboring pixels have no influence on the decision and we obtain the special case

$$
\begin{equation*}
d^{2} \gtrless{ }_{u}^{c} 2 \frac{\sigma_{c}^{2} \sigma^{2}}{\sigma_{c}^{2}-\sigma^{2}} \ln \frac{\sigma_{c}}{\sigma} \quad\left(\text { if } n_{h v}^{c}=n_{h v}^{u} \text { and } n_{d i a g}^{c}=n_{d i a g}^{u}\right) \tag{7.31}
\end{equation*}
$$

Chapter 7. Background Subtraction

Otherwise, the threshold is shifted in either direction to bias the decision level. If, for example, the neighborhood has many unchanged pixels, $n_{h v}^{u}$ and $n_{\text {diag }}^{u}$ will be low, leading to a higher threshold. This again will bias the decision for the center pixel towards unchanged.

### 7.4.3 Extension to color images

Up to this point, the MRF-based approach was described for luminance images only. However, it is not difficult to extend this approach to color images, assuming that the color distribution is a multi-variate Gaussian distribution (as done previously when introducing the Mahalanobis distance metric in Section 7.2.2). When considering again images in the YUV colorspace, we can assume that the noise variance in the luminance channel is $\sigma^{2}$, while it is $\beta^{2} \sigma^{2}$ in the color channels. Consequently, the covariance matrix of color pixel-differences $\mathbf{d}=\left|\mathbf{I}_{t}^{Y U V}-\mathbf{I}_{B}^{Y U V}\right|$ is a diagonal matrix given as $\mathbf{S}=\sigma^{2} \operatorname{diag}\left(1, \beta^{2}, \beta^{2}\right)$. This enables to write the probability density as a multi-variate Gaussian

$$
\begin{equation*}
p(\mathbf{d} \mid q=u)=\frac{1}{\sqrt{(2 \pi)^{3} \sigma^{3} \beta^{2}}} \exp \left\{-\frac{1}{2} \mathbf{d S}^{-1} \mathbf{d}^{\top}\right\} \tag{7.32}
\end{equation*}
$$

Inserting this into Eq. (7.28) results in

$$
\begin{equation*}
\frac{\sqrt{\left(2 \pi \sigma_{c}\right)^{3} \beta^{2}} \exp \left\{-\frac{1}{2} \mathbf{d} \cdot \mathbf{S}^{-1} \cdot \mathbf{d}^{\top}-n_{h v}^{u} \gamma_{h v}-n_{\text {diag }}^{u} \gamma_{d i a g}\right\}}{\sqrt{(2 \pi \sigma)^{3} \beta^{2}} \exp \left\{-\frac{1}{2} \mathbf{d} \cdot \mathbf{S}^{-1} \cdot \mathbf{d}^{\top}-n_{h v}^{c} \gamma_{h v}-n_{d i a g}^{c} \gamma_{d i a g}\right\}} \gtrless_{c}^{u} 1 \tag{7.33}
\end{equation*}
$$

which finally leads to the decision function

$$
\begin{equation*}
\left(d_{M}^{Y U V}\right)^{2} \gtrless{ }_{u}^{c} 6 \frac{\sigma_{c}^{2} \sigma^{2}}{\sigma_{c}^{2}-\sigma^{2}}\left(\ln \frac{\sigma_{c}}{\sigma}+\left(n_{h v}^{c}-n_{h v}^{u}\right) \gamma_{h v}+\left(n_{d i a g}^{c}-n_{d i a g}^{u}\right) \gamma_{d i a g}\right) \tag{7.34}
\end{equation*}
$$

where $d_{M}^{Y U V}$ is the Mahalanobis distance. Note that this decision function differs from Eq. (7.34) only in the changed constant at the right-hand side.

### 7.4.4 Optimization algorithm

The optimization problem to find the segmentation mask with the maximum a-posteriori probability (MAP) for the described Markov field model is difficult, since the number of variables equals the number of pixels in the image. A number of algorithms has been proposed to find an approximate solution to maximize the joint probability (see [114] for a comparison). We have chosen to use the iterated conditional modes (ICM) algorithm, which was initially described in [9]. This algorithms runs several passes over the
image, where each pixel is assigned the most probable label, considering its current local neighborhood. Specifically, this means that we iterate through all pixels in the segmentation mask image and set the label of a pixel according to Eq. (7.30). When all the labels have converged, the algorithm is stopped. It is assured that the algorithm converges, because each modification of a pixel increases the joint probability of the Markov field.

## Efficient implementation with a queue of boundary pixels

Fortunately, the algorithm complexity can be reduced, since most of the pixels will not change their label from one iteration to the next. As a consequence, we maintain a queue of pixels that should be checked for modification. In each iteration, we take a new pixel out of the queue until the queue is empty. If we check a pixel and it does not change its label, the neighboring pixels will not be affected. However, if the pixel label is changed, this may also influence the label of the neighboring pixels. Consequently, when the label of a pixel is changed, we put the coordinates of the eight neighboring pixels into the queue to check them for modification in a later iteration. This algorithm will converge, because we know that the number of label changes are limited and the queue does only grow in size as long as labels are modified.

## Initialization

Up to now, the inner loop of the segmentation has been described. In each step, this iterative process improves the segmentation mask from the previous step. However, we still need a segmentation mask to start with. Taking a completely transparent or completely opaque segmentation mask does not converge to a good solution, because the shape prior effectively inhibits any change.

One solution is to start with a very weak shape prior with parameters $\gamma_{h v}, \gamma_{\text {diag }} \approx 0$, and to increase these parameters gradually. Note that if both are set to zero, we obtain the pixel-based decision function Eq. (7.31).

An easier approach is to start with the result of one of the previous segmentation algorithms. In our system, we use the $\chi^{2}$ significance test on $5 \times 5$ windows with the Mahalanobis distance. After the initialization of the segmentation mask, the queue of pixels is initialized with pixels to be checked. To limit the number of pixels in the queue, we only fill in pixels along the object boundary.


Figure 7.14: Results of the MRF-based segmentation. (Compare to Figs. 7.9 and 7.3.)

### 7.4.5 Evaluation

The result of the Markov field based segmentation for two example frames of the stefan sequence is depicted in Fig. 7.14. When comparing these results to the results in Figs. 7.9 and 7.3 that were obtained with the previous segmentation algorithms, it can be concluded MRF-based segmentation yields more accurate results. The quality of the interior of the object is comparable to the $\chi^{2}$ significance test, but the algorithm does not yield the disadvantageous aura effect along the object boundary.

## Parameter selection

The MRF-based segmentation algorithm includes two parameters $\gamma_{h v}, \gamma_{\text {diag }}$ that control the influence of the shape prior. In our final system, we used $\gamma_{h v}=2.5$ and $\gamma_{\text {diag }}=1.25$, which are the values proposed in [2]. However, we noticed that an increase of these parameters does not have much effect on the obtained segmentation mask.

Surprisingly, we could even obtain comparable results with setting $\gamma_{h v}=$ $\gamma_{\text {diag }} \approx \infty$. If we examine Eq. (7.34), it can be derived that setting these two parameters to very high values can force a pixel to foreground or background just because of the labels of its neighbors. Only for the case that $n_{h v}^{c}=n_{h v}^{u}=2$ and $n_{\text {diag }}^{c}=n_{\text {diag }}^{u}=2$, the shape prior is zero and the decision function reduces again to Eq. (7.31). The cases for which this happens are depicted in Fig. 7.15. In all other cases, the label for the center pixel can be simply derived from the pixel neighborhood. This simplified


Figure 7.15: Patterns or neighborhood pixels, for which $n_{h v}^{c}=n_{h v}^{u}=$ $n_{\text {diag }}^{c}=n_{\text {diag }}^{u}=2$. In these cases, the label for the central pixel is chosen only based on the difference value at the central pixel.
algorithm is not used in our implementation, but it can be advantageous for a hardware implementation, for which it is easier to implement.

### 7.5 Sources of errors and robustness improvements

In the previous section, it was assumed that scenes were recorded with a static camera, or that the camera motion was compensated by taking background pictures from the synthesized background mosaic. At first view, this should assure that corresponding pixels are co-located in the foreground and the background image. However, in practicer, three problems occur:

- Registration errors. When the background view is reconstructed from a synthesized panorama, small inaccuracies in the motion-model can occur. Even though this is usually much less than a pixel distance, it can cause difficulties along strong edges. If there is a large difference in brightness across the edge, a tiny inaccuracy in the motion model or aliasing in the input video can cause a large value in the difference image.
- Interpolation errors. In the background reconstruction process, the input images are resampled when they are warped onto the background image. This resampling involves bi-linear interpolation that results in some image blurring. To obtain the camera-motion compensated image, a second resampling is carried out to obtain the current background view from the synthesized background overview image. The consequence of these two resampling steps is that a blurred background reconstruction is compared with the sharp input image. Especially within fine texture, this can lead to false positives.
- Motion blur. For the case that the input sequence comprises fast camera motion, the image is not only transformed geometrically, but


Figure 7.16: A background image that shows different strengths of motion blur, because it combines slow and fast camera motion in a single image. At the right side, camera motion was slow, so that the image is sharp. At the left side, the camera motion is much faster, resulting in significant image blurring. Especially note the difference between the two stair rails in the center area.
it can also show motion blur. The amount of blur can differ throughout the sequence, but this cannot be reflected in the synthesized background. Consequently, errors occur, e.g., when comparing a motionblurred input image with a sharp background image (see Fig. 7.16). However, note that motion blur itself does not cause segmentation errors provided that the same motion blur occurs both in the input image and the background image.

Our general concept to approach these problems is to identify those parts of the image, that have a high error risk. We then modify the previously discussed segmentation algorithms to exclude the identified pixels of high risk from the difference image computation. We will present solutions to improve the robustness against the first two problems in the remainder of this section. First, we describe the concept of using maps of risky pixels and how to obtain these maps. Afterwards, the modifications to the segmentation algorithms to consider these maps in the object segmentation are presented.


Figure 7.17: Two edge-profiles, with (a) low-contrast, (b) high-contrast edge. In the high-contrast case, a misregistration of the edges induces a larger luminance error than in low-contrast cases.

### 7.5.1 Map of misregistration risk

The previous segmentation algorithms assumed that the difference image comprises only camera noise and differences to foreground objects. However, the camera-motion compensation itself can lead to misregistration and resampling errors that also appear in the difference image. This kind of error depends on the local contrast in the pixel neighborhood and has the largest value at strong edges (see Fig. 7.17). This is not in accordance with the assumption that the variance of noise is independent of the position in the image.

As a first solution [47], we investigated to change the image difference measurement. Instead of using the direct luminance difference $\mid I_{B}(x, y)$ $I_{t}(x, y) \mid$, we compensated for the expected misregistration along edges by dividing by the luminance gradient in the background image, leading to

$$
\begin{equation*}
d(x, y)=\frac{\left|I_{B}(x, y)-I_{t}(x, y)\right|}{\left\|\nabla I_{B}(x, y)\right\|} . \tag{7.35}
\end{equation*}
$$

Even though this approach helps in reducing the misregistration errors, we observed problems in cases where the background is textured and the object has uniform color. In these cases, the gradient in the background is high, which consequently leads to a lower total error. This is not optimal, because we know that the observed region cannot be background as the object shows no texture. This observation leads to a different approach, which explicitly identifies a pixel to be probably affected by misregistration errors. More specifically, we identify a pixel as risky if there is a high-contrast edge in the background image, and there is also a high-contrast edge in the foreground image at the same position. To detect steep edges, we simply apply a threshold onto the gradient magnitudes. Since a misregistration only results in a segmentation error when there are edges is both images, we combine the detected edges into the map $R_{M}(x, y)$ of misregistration


Figure 7.18: Example for misregistration, following from a lightly moving camera caused by wind.
risk with the definition that

$$
\begin{equation*}
R_{M}(x, y)=\left(\left\|\nabla I_{B}(x, y)\right\|^{2}>\tau_{m}\right) \wedge\left(\left\|\nabla I_{t}(x, y)\right\|^{2}>\tau_{m}\right) \tag{7.36}
\end{equation*}
$$

where $\tau_{m}$ is a fixed threshold for edge detection. An example scene with an particularly strong misregistration effect is shown in Fig. 7.18. Note that in this example, the camera was assumed static, so that a static background image was used for the segmentation. However, the camera seems to have moved a little bit, probably because of wind. This tiny motion results in the observed misregistration.

### 7.5.2 Map of interpolation errors

The second problem is introduced by the image warping in the camera motion compensation. Remember that during the background reconstruction, the input frames are warped into the reference coordinate system. In this process, the target pixel values are interpolated from the input pixels using bi-linear interpolation. A second interpolation step is carried out to reconstruct the current camera view from the background reconstruction. In total, this means that this interpolation is applied twice, such that a reconstructed background view looks blurred, compared to the original input frame. Suppose that we now take the difference to the original input, large differences can occur just because of this blurring effect (see Fig.7.19). We approach this problem in a comparable way as with the misregistration errors.

To simulate the blurring of the two interpolation steps, we apply a simple low-pass filter to the current input image. This blurred image is compared with the input image and a risk of interpolation error is detected when the difference exceeds a threshold $\tau_{i}$. This defines the map of interpolation risk as

$$
R_{I}(x, y)=\text { true } \quad \text { iff }\left|I_{t}(x, y)-\frac{1}{16}\left(\begin{array}{ccc}
1 & 2 & 1  \tag{7.37}\\
2 & 4 & 2 \\
1 & 2 & 1
\end{array}\right) \otimes I_{t}(x, y)\right|>\tau_{i}
$$

A different approach to eliminate the interpolation problem that could be further studied in future research is to avoid the double interpolation by reorganizing the segmentation process. The current background subtraction data-flow is shown in Fig. 7.20(a). Note that the actual background subtraction is performed in the input coordinate space. However, it is also possible to carry out the background subtraction in the coordinate system of the background image as shown in Fig. 7.20(b). The advantage of this would be that the input image is transformed exactly once. Moreover, in both cases, for reconstructing the background and also for carrying out the background subtraction, exactly the same transformation is used. Consequently, both images will show the same interpolation artifacts. When comparing the two images, this means that both interpolation artifacts cancel out. However, the disadvantage of this approach is that we obtain the segmentation mask in the background coordinate system and we may require an additional step to transform the obtained mask to the image coordinate system again.


(e) Segmentation without misregistration reduction.

(f) Segmentation with misregistration and interpolation reduction.

Figure 7.19: Example for interpolation error reduction. When comparing the input image (a) and the reconstructed background view (b), it is visible that the background view is slightly blurred. Since this blur is only visible at the pixel level, detail views are provided in (c) and (d). Segmentation without misregistration/interpolation reduction erroneously includes regions like the score display or the logo. These regions are removed by enhancing the segmentation algorithm with misregistration and interpolation risk maps.


Figure 7.20: Two possible data-flows for carrying out the background subtraction. (a) The background is transformed to the current camera view, background subtraction is performed in the input coordinate space. (b) The input is transformed to the background coordinate space, background subtraction is carried out in the background coordinate space.

### 7.5.3 Integrating risk maps into the segmentation process

Once the pixels are obtained for which a possible unreliability is identified in the difference image, the previously discussed algorithms can be modified such that these unreliable pixels are excluded from the computation.

## $\chi^{2}$ significance test

The central operation of the $\chi^{2}$ significance test is to sum the squared difference values in a small neighborhood window. The change when integrating the risk maps is that pixels that we classified as risky are not included in the sum. Note that the degrees of freedom in the $\chi^{2}$ test are not reduced by the number of removed pixels.

## MRF-based segmentation

The MRF-based segmentation uses Eq. (7.30) as the decision function. If the center pixel is classified as risky, we base the segmentation only on the neighborhood information, i.e., the shape prior. This leads to the decision function

$$
\begin{equation*}
0 \gtrless_{u}^{c}\left(n_{h v}^{c}-n_{h v}^{u}\right) \gamma_{h v}+\left(n_{\text {diag }}^{c}-n_{\text {diag }}^{u}\right) \gamma_{\text {diag }} \tag{7.38}
\end{equation*}
$$

if the considered pixel is risky, otherwise Eq. (7.30) is used without modification.

### 7.6 Postprocessing the object mask

When performing a subjective evaluation of the segmentation masks, we can discover several obvious errors, which on a closer look are only considered obvious because we recognize the object and can immediately judge that a specific mask cannot be correct. When the type of the observed objects are known, various heuristic rules can be derived for postprocessing the segmentation mask to remove these errors. A selection of these rules seem to be true in many practical situations, so that they can be provided as general postprocessing filters in a multi-purpose segmentation system.

### 7.6.1 Filling holes in the object

Most natural or man-made objects have no holes (in the sense of a torus, not in the sense of a coffee cup). The majority of the objects are either compact (like cars, balls), or they are star- or tree-shaped (like humans, animals, flowers). Consequently, small holes in objects are most likely segmentation errors and we can usually improve the segmentation by closing the holes


Figure 7.21: Postprocessing of segmentation masks. (a) Output from MRFbased segmentation. (b) Output of mask post-processing. Small clutter regions are removed and the disconnected object region (the leg) is added to the main object mask.
in objects. The most frequent case in which this heuristic fails is when articulated objects like humans build loops with their limbs (see Fig. 7.21). Hence, to limit the impact of the modification, we put a small threshold $\tau_{h}$ on the size of the hole and close it only if the size if below this threshold.

### 7.6.2 Heuristics for removing clutter in the mask

Another frequent case is that we know in advance to expect only relatively large objects in the image. Small regions in the mask are likely either just errors or small changes in which we are not interested. One example of the latter is a recording of sports, where the actors are visible in the front and the audience is located at the background. In this case, not only the actor will move, but also the audience has little motion. Usually, this motion irrelevant for most applications. Hence, we also provide a filter to remove regions if their size is below a small threshold $s_{o}$. If it is known that the scene contains only one object, we can even exclude all regions except the largest one.

However, in the case that there is only one object of interest, excluding all regions but the largest one can result in incomplete segmentation masks. This is the case if parts of the object are disconnected from the main region. One example case is shown in Figure 7.21(a), where part of one leg has no connection to the body. If we would only keep the largest region, this part of the leg would be lost. To prevent this behaviour, we modify the single-
object heuristic to include also regions which are close to the largest region and which are not too small. The algorithm becomes as follows.

1. Find the largest region in the segmentation mask.
2. Grow this largest region by $d_{o}$ pixels. This is done with $d_{o}$ dilate operations.
3. If another region is touched during the dilate operations, check if the size of this region exceed the minimum size $s_{d}$.
4. If the region size is larger than $s_{d}$, add this region to the final segmentation mask.

The result of this postprocessing heuristic is shown in Fig. 7.21(b).

### 7.7 Overview of the segmentation process

This section summarizes which of the previously described algorithms are used in our segmentation system and how they are connected. The dataflow of the complete background-subtraction module is shown in Figure 7.22. The input of the module is the original input video stream and a synchronous stream of reconstructed background images. These background images are obtained from the synthesized multi-sprite background model by extracting the current camera view from the multi-sprite. This means that the background sequence shows the same content as the input sequence, but without the foreground objects.

The background-subtraction module first computes the risk map, which identifies for which pixels the segmentation is possibly unreliable. Taking this risk map into account, a $\chi^{2}$ significance test is carried out to obtain an initial segmentation mask. The Mahalanobis distance in the YUV colorspace is employed as distance measure. This algorithm already yields a good segmentation mask, but it still shows the aura effect along the object boundary and it usually also contains small clutter. Both problems are reduced in the successive refinement based on the Markov random field model. This algorithm also takes the risk map into account.

The final two postprocessing steps comprise filling of object holes and removing small clutter regions from the segmentation mask. We consider these two steps as optional, since their integration may depend on the application. Small regions that remain after the MRF optimization step usually correspond to actual changes in the image. However, many of these small changes are often unimportant actions.


Figure 7.22: Architecture of the background-subtraction module.

The background-subtraction module is influenced by many adjustable parameters. The parameter values selected in our system are summarized in Table 7.1. Most of these parameters are not critical and can be changed without strong effects on the output. The most sensitive parameter is $\sigma$, for which we obtained the best results for the range $\sigma=6, \ldots, 16$, depending on the quality of the input sequence. For compressed input sequences, a higher $\sigma$ of around 12 proved to be most suitable.

| Processing step | Parameters |
| :--- | :--- |
| $\chi^{2}$ significance test | $w=5 \times 5, \sigma=8, \alpha=10^{-6}, \beta^{w}=0.2$ |
| MRF optimization | $\sigma=8, \sigma_{c}=40, \gamma_{h v}=2.5, \gamma_{\text {diag }}=1.25$ |
| Fill object holes | $s_{h}=20$ |
| Remove clutter | $s_{o}=500$ or largest only, $d_{o}=15, s_{d}=150$ |
| Misregistration risk map | $\tau_{m}=200$ |
| Interpolation risk map | $\tau_{i}=12$ |

Table 7.1: Parameters used in our background-subtraction module.

An expert is someone who knows some of the worst mistakes that can be made in his subject, and how to avoid them. (Werner Heisenberg)

## Chapter <br> 

## Results and Applications

This chapter describs the combination of the algorithms that were presented in Chapters 3 to 7 into a complete segmentation system. Several variants of the segmentation system are discussed, each optimized for a different type of input sequence, like scenes captured with a static camera or scenes with a known background. Furthermore, it is addressed, which of these systems are suitable for real-time processing, and how the more complex offline systems can be implemented as an online system. For evaluation purposes, results are presented for a large variety of sequences to show the quality of th segmentation masks, but also to show the limitations of the proposed approach. Algorithm enhancements to overcome these limitations are proposed for future work. Finally, examples are provided for application of the segmentation system in MPEG-4 video coding, 3-D video generation, video editing, and video analysis.

### 8.1 Algorithm modules

The proposed segmentation system is composed of several algorithms that were described in the preceding chapters. The following list provides an overview of these algorithms and summarizes their execution behaviour.

- Feature-based motion estimation (Chapter 3 and 4). This module computes camera-motion parameters for pairs of successive frames. The computation can be carried out online, since the computation for each frame only requires the feature-points from the current and preceding frame, as well as (optionally) the previous motion model for computing the motion prediction.
- Multi-sprite partitioning (Chapter 6). Given the set of interframe motion parameters from the feature-based motion estimator, the multi-sprite partitioning separates the input sequence in ranges for which independent background images are synthesized (in the sequel called segments). The multi-sprite partitioning requires the camera-motion parameters of the complete sequence to make an optimal decision. Therefore, it is an offline process that provides the result at the end of the sequence. However, the algorithm can be modified to output a new segment online as soon as it is known. Generally, there is no maximum delay, after which these ranges are known, but if strict optimality is not required, it is possible to set a maximum number of frames $s_{\max }$ per sprite. This allows to implement the multi-sprite partitioning as an online process with a maximum delay and buffer requirement of $s_{\max }$ frames.
- Direct motion estimation (Section 5.2). Once the frames that should be incorporated into one background image are known, they have to be aligned with a high precision. This step uses the parameters of the feature-based motion estimator as initialization. Furthermore, it also requires all input frames of the processed frame range. The output of this step is a refined set of global-motion parameters that are used to synthesize a background image.
- Background image reconstruction (Section 5.3). With the accurate motion parameters from the direct motion estimation, this step combines the input frames into a global background image and eliminates the foreground objects from this background image. Here, we can distinguish two algorithm variants. If a single background image is desired as output, the background reconstruction algorithm will first collect all available images and, afterwards, synthesize the background image (offline mode). This first algorithm is used, e.g., for


Figure 8.1: Online segmentation system for sequences captured with a static camera. An optional background-update module adapts the background image to changes in the scene.

MPEG-4 video coding to generate the sprite image. A second variant occurs in a video analysis application that does not output the background image itself. In this case, it is possible to keep a single background image that is continuously updated (online mode). This is a common approach in surveillance systems that have to cope with a varying illumination.

- Background subtraction (Chapter 7). This step computes the video-object masks based on a pure background image. For non-static cameras, it also incorporates global-motion parameters to compensate the camera motion. Since every frame is processed independently, the algorithm can run online.


### 8.2 Variants of the segmentation system

The algorithm modules can be combined in different ways to adapt to different types of input that are typically processed in specific applications. Consequently, the system can be tuned to process easy sequences with reduced computational complexity, or to use more complex algorithms for more difficult scenes. In the following, we will discuss several architectures for example applications.

### 8.2.1 Surveillance with a static camera

In the simplest case, it is preferred to analyse or store the video data captured with a static camera that observes a scene with known background. For this case, no camera-motion estimation is required and the segmentation system comprises only the background-subtraction module (Fig. 8.1). If the scene is static, we can even assume that the background image is manually captured once in the initialization. However, usually, it is desired that the background adapts automatically to changes in the scene. These


Figure 8.2: Online segmentation system for sequences captured with a rotational camera in a known environment. Camera-motion is computed directly relative to a known background image. The optional background-update module adapts the background image to changes in the scene.
can be gradual changes in the illumination or changes in the environment like moving of furniture. To allow for an automatic update of the background image, an optional background-reconstruction module can be added to the system. This background reconstruction can be implemented either with an explicit iterative background update algorithm (see Section 5.3.1 on page 147), or background images can be generated at regular intervals (e.g., once every minute) using an algorithm as described in Section 5.3.2.

This set-up can be implemented as a real-time online application, and it is the currently most frequently-used architecture for surveillance systems.

### 8.2.2 Surveillance with a moving camera

Let us now relax the restriction to a static camera, while still assuming that the scene background does not change. This is for example the case for user-controlled pan/tilt/zoom cameras or automatic scanning cameras, mounted at a fixed position. For this application, we can also initialize the system manually with a known background image, which covers the complete field-of-view of the camera.

In this set-up, the system has to align the input image with the corresponding view of the background image prior to carrying out background subtraction. Even if information about the camera position is available from the system (e.g., because the camera motors are computer controlled), the accuracy is usually not sufficient for image alignment ${ }^{1}$. Consequently, an accurate alignment has to be computed from the image data. Because the

[^16]background image is already available, the motion parameters can be computed directly relative to this background image, instead of first computing interframe motion. These motion parameters are subsequently used in the background-subtraction algorithm to obtain camera-motion compensated views from the background image (Figure 8.2).

Similarly to the static camera set-up, we can optionally update the background image iteratively to adapt to varying illumination or changes in the scene. The update algorithm also requires the computed motion parameters to update the correct area of the background image.

### 8.2.3 Offline video analysis

Automatic video analysis is expected to become increasingly important in future video processing systems. Examples are intelligent searching in video databases, analysis of sports recordings, or medical applications. In these applications, the analysis can usually be carried out offline, which also allows for more in-depth analysis using computationally intensive algorithms. If the environment is known, it is possible to often use a computationallyefficient approach as described in the previous two sections. However, for many analysis applications the environment is not known. For instance, in a video database system, we desire to place queries such as "find all scenes with cars", independent of the video source. Especially for the analysis of television broadcasts in, e.g., a personal video recorder, no pre-knowledge about the video content is available. Consequently, a general video analysis system has to synthesize suitable background images automatically.

We implemented an offline video analysis system to compute the segmentation in four passes over the input data (see Fig. 1.2 for the architecture). These passes comprise the following processing steps.

- Pass 1: Feature-based motion estimation, combined with the multisprite partitioning. The outputs of this step are approximate motion parameters and the sequence partitioning.
- Pass 2: Calculation of accurate motion parameters with the direct motion-estimation algorithm.
- Pass 3: Synthetization of a background image without foreground objects.
- Pass 4: Object segmentation with the background-subtraction algorithm.

Each of these four passes requires the original input video sequence and parameters from the preceding computation steps. This offline segmentation
system is easy to implement, but it requires access to the complete video data at once. While this is usually not a problem for implementations on personal computers, it can impose difficulties for resource-constrained platforms. A modified architecture for resource-constrained systems is proposed in the successive section.

### 8.2.4 Online video segmentation and transmission

The previous offline video segmentation system is not applicable if the segmentation is required in real-time. An important application where realtime performance is required, is the transmission of live video with MPEG-4 video coding. However, a general low-delay video transmission system using MPEG-4 sprite coding is not possible, because the sprite image first has to be built in the encoder. Since the sprite image combines video content from possibly long video scenes, and because the sprite image should not contain foreground objects, the encoder has to collect input frames until the background sprite can be synthesized and the objects have moved sufficiently so that they can be removed from the background. These processing steps require buffers that store the frames of at least one segment, i.e., the frames that will be composed into a single sprite image.

To enable online processing, it is required to restrict the maximum segment length. This allows to limit the number of image buffers and herewith also the maximum delay that results from this buffering. A complete segmentation system using this approach is depicted in Figure 8.3. The processing in this framework is organized in a pipeline structure, where each stage operates concurrently on one segment of the input sequence.

The first stage in the pipeline computes the interframe-motion parameters for each pair of input frames. Additionally, a multi-sprite algorithm is interleaved with the computation of the motion parameter. The multisprite algorithm is modified to limit the maximum length of a segment to a maximum of $s_{\max }$ frames. Consequently, whenever the input buffer contains $s_{\max }$ frames, the multi-sprite algorithm computes the number of frames for the next segment. Once the segment length is known, the second stage can begin to compute accurate motion parameters for each input frame of the segment (Fig. 8.4). The input images for the second stage are taken from the image buffers that delayed the input by $s_{\max }$ frames. The first stage can continue its computations for the next sprite, while the second stage works on the current segment. Note that both stages can run synchronously, such that Stage 1 processes one frame $t$ at the same time as Stage 2 processes frame $t-s_{\max }$.

Stage 3 uses the accurate motion parameters to synthesize a background image. Again, this stage runs in parallel to the previous stages. However,


Figure 8.3: Online segmentation system for general video sequences. The maximum number of frames that are covered by a single sprite is limited to $s_{\text {max }}$ frames. Hence, the processing can be pipelined with image buffers (each with space for $s_{\max }$ images) at each processing stage. Although not shown here, the motion parameters should also be delayed between successive processing stages.


Figure 8.4: Scheduling diagram of the online segmentation system. The processing pipeline consists of four consecutive stages. The $3 \times s_{\max }$ image buffers (shown along the vertical axis) are organized as a ring-buffer. In the beginning, new images are fed into the buffer until it contains $s_{\max }$ images. At this time, the first stage can decide on the size of the first segment. Subsequently, Stage 2 begins to work on this segment, and so on. Note that a frame is removed from the queue, when Stage 4 finishes the segmentation of that frame. Hence, the total processing delay is $3 \cdot s_{\text {max }}$ frames.
because most background-synthetization algorithms do not operate frame-by-frame but, e.g., scanline after scanline, it requires access up to $s_{\max }$ frames in parallel. When Stage 3 finishes the synthetization of the background image, the image is moved to a queue of sprite-image buffers such that Stage 3 can reconstruct the next sprite image, while Stage 4 works on the segmentation of the previously computed segments.

If the system generates several short (compared to $s_{\max }$ ) segments after each other, several sprite-image buffers are required to store these images (Fig. 8.5). In the worst case, if every sprite segment only includes one image, we would require $s_{\max }$ sprite image buffers. To limit the number of buffers, we can limit the minimum length of a segment to include at least $s_{\text {min }}$ frames. With this limitation, we only require $s_{\max } / s_{\text {min }}$ sprite image buffers in the worst case. Note that cases where very short segments are proposed by the multi-sprite partitioning, the encoder can also switch to a different coding mode and decide not to use sprite coding at all. This is a sensible approach anyway, since a short sprite segment will not be efficient to code and the segmentation will probably be of low quality (not enough information to build correct background image).


Figure 8.5: Scheduling diagram for the case of many small segments. At the end of Stage 3, a new sprite image is generated. This sprite image must be available until the segmentation in Stage 4 has completed. If the sprite segments are short, several sprite images have to be buffered.

### 8.3 Implementation

We implemented the offline version of the proposed segmentation system (Section 8.2.3) and a simplified version for cases in which we know in advance that the camera is static (Section 8.2.1). In the latter case, we synthesized the background image with the same automatic algorithm, but we disabled camera-motion compensation. The simplified version carries out two passes over the input sequence (1. synthesize background, 2 . object segmentation). The full version runs in five passes (1. feature-motion, 2. multi-sprite partitioning, 3 . direct motion, 4. synthesize sprites, 5 . object segmentation). Since we did not combine the first two passes into one, this is one pass more than required. However, for a full software implementation, this is not a disadvantage because the second pass is computationally inexpensive as it only works on the global motion parameters.

All programs were implemented in system-independent C++. Some computationally-expensive functions like feature-point detection, featurematching, low-pass/high-pass filters, color-space conversions, and parts of the direct motion estimation were additionally implemented using the SIMD extension of the x86 processor architecture (MMX, MMX-2, SSE). The software was developed under Linux, but has also been tested with other UNIX variants (HP-UX, Solaris), and MS-Windows (using the CygWin environment).

### 8.4 Segmentation results

We tested our segmentation system with a wide variety of sequences. A selection of representative results is depicted in Figures 8.6 to 8.12. All of these results were obtained by running the segmentation system with the same parameter settings. For the results in Fig. 8.6, and 8.12(g)-8.12(l), we used the simplified algorithm for static cameras. In the sequel, we comment on special observations that can be made on these sequences.

- Fig. 8.6, badminton: In frame 0, we can see a "segmentation error" in the top-left corner. This is due to the superimposed score display, which was not visible in the first frames. Apart from the players, also the two referees on both sides of the court move a little bit and hence appear sometimes in the segmentation. The ball and the rackets are not visible in the segmentation, since they are too small and partly transparent (ball is transparent because of motion blur).
- Fig. 8.7, trampoline: Additionally to the athlete, also some people move and hence appear in the segmentation. Also the trampoline appears when the athlete jumps on it. Errors along the high-contrast edges in the balustrade are eliminated by the misregistration and blur reduction algorithms.
- Fig. 8.8, gymnastics: A sequence with camera motion. The bottom part of the image only contains lines, along which no feature-points can be identified. Hence, the feature-based camera-motion estimation is based only on features in the top image part. However, the successive direct motion estimator considers the whole image and refines the alignment of the lines. The segmentation algorithm provides an accurate segmentation mask, but it also includes many people that walk in the background of the scene.
- Fig. 8.9, surveillance: The sequence was recorded with a handheld camera and the motion is not perfectly rotational around the optical center. The wheels of the bicycle are partly missing, since they are mostly transparent. Also, part of the bottom car in frame 400 is missing, because its color is very close to the street color. The small regions are no segmentation errors, but people walking through the scene. Since the legs of these people are thin, they are missing (compare input and segmentation of the top left human in frame 250).
- Fig. 8.10, stefan: Segmentation of the MPEG-4 test-sequence stefan. Using the multi-sprite approach (see Fig. 6.22), all 300 frames can be processed.
- Fig. 8.11, walking/sitting: Note the reflections on the ground that also appear in the segmentation. This sequence is further analyzed in Appendix F to recognize the actions (walking, sitting).
- Fig. 8.12, miscellaneous sequences: In the tennis sequence, note that the player shadow is included in the segmentation. Also note that the logo of the TV-station is in the segmentation, since the logo is fixed even when the camera image moves (logo moves relative to background).

While the results are good for the previous examples, we observed problems with some other sequences. We discuss the main difficulties based on the following examples.

- Fig. 8.13, claire: This well known test-sequence shows a newsspeaker in front of a uniform blue background. We observe two problems. First, since the background has uniform color, we cannot identify any feature-points on it. Consequently, the camera-motion estimation cannot lock to the (static) background. Instead, the motion estimation calculates parameters that fit to the foreground object motion, since this is the dominant motion (in terms of number of feature-points). The algorithm cannot detect that this is the wrong motion, since the motion-compensated difference in the background area is low for any arbitrary motion field. If it is known beforehand that the camera is static (like in this example), we can eliminate this problem by simply disabling the camera-motion estimation.
The second problem is that the object does not move enough to enable the construction of a clean background image. In this example, it would be possible to create a synthetic background image with pure blue color, but in general, this is not always possible. Consequently, the background image includes large parts of the foreground object and the segmentation result only includes those areas that move.
- Fig. 8.14, news: This is a real-world example showing the same problems as with the claire sequence. In this case, the background includes enough information to obtain the correct (static) camera motion. However, the problems of the non-moving foreground object remains. Depending on the current pose of the foreground object, this can lead to an acceptable, but also to insufficient segmentations.
- Fig. 8.15, hurdles: This sequence shows a hurdles race, captured by a panning camera. Since it is a wide camera pan, the algorithm splits the background sprite into several parts. For the first sprite


Figure 8.6: Results for a badminton scene, recorded from a DVB source. The segmentation error at the top-left corner of (c) is due to the score display that was superimposed. At the beginning and end of the sequence, also the two referees appear in the segmentation because they move their heads.


(d) Frame 25.

(g) Frame 100

(j) Frame 175.

(e) Frame 50 .

(h) Frame 125.

(k) Frame 200.

(f) Frame 75 .

(i) Frame 150 .

(l) Frame 225.

Figure 8.7: Results for a trampoline sports scene, recorded from a DVB source. Note the segmentation of frame 150, where the trampoline is included in the segmentation mask, since its cloth is deformed.


(c) Input, frame 250.
(f) Frame 150 .
(i) Frame 300 .


(d) Frame 50 .

(e) Frame 100 .

(g) Frame 200.

(j) Frame 350 .

(h) Frame 250.

(k) Frame 400.

Figure 8.8: Results for a gymnastics scene, recorded from a DVB source. Note that the segmentation errors are due to people that walk around in the background.


Figure 8.9: Results for a street surveillance scene, recorded with a handheld DV camera.


Figure 8.10: Results for the MPEG-4 test-sequence stefan. The four sprites that were used for segmentation are depicted in Figure 6.22. Some people in the audience moved slightly and are therefore included in the segmentation.


Figure 8.11: Results for a scene that was recorded with a hand-held DV camera. Because the camera was held in the hand, small camera motion is present.

(a) Tennis, frame 140, moving camera.

(d) Aquarium, input
(d) Aquarium, input
frame 1, static camera.

(g) Surveillance, scene 1, static camera.

(j) Surveillance, scene 3 , static camera.

(b) Frame 60 .

(e) Frame 0 .

(h) Scene 1.

(k) Scene 3.

(c) Frame 140 .

(l) Scene 4.

Figure 8.12: Results for various sequences. The two surveillance sequences are courtesy of Bosch Security Systems B. V..


Figure 8.13: Results for the claire sequence. This is a difficult sequence for the proposed segmentation system, since the motio estimation cannot place any feature-points in the background area. Moreover, the body of the foreground object does not move, so that it cannot be detected.


Figure 8.14: Results for a news sequence recorded from a DVB source. Comparable result to Fig. 8.13.

(a) Hurdles, sprite 1-93.

(c) Hurdles, input frame 26.

(e) Hurdles, segmentation frame 26 .

(b) Hurdles, sprite 164-200.

(d) Hurdles, input frame 172.

(f) Hurdles, segmentation frame 172.

Figure 8.15: Results for a hurdles race sequence (see Appendix D). Construction of the first sprite (a) is successful, since enough feature-points are available for motion estimation. A later sprite (b) cannot be constructed well, since large parts of the background are uniform or only show lines. Consequently, the segmentation results for frames within the first sprite are better (e) than for frames from later sprites (f).
(Fig. 8.15(a)), the background has enough texture to enable an accurate motion estimation and, consequently, a good segmentation. Later in the sequence, the background is almost uniform or contains only lines along which no feature-points can be identified. As a result, the estimated camera-motion parameters have less accuracy and the synthesized background image has a low quality (Fig. 8.15(b)). In the end, this leads to an inaccurate segmentation mask.

A framework to solve the problem of static foreground objects is described in Section 8.6.4 and particularly in Chapters 9 and 10. A proposal to solve the camera-motion estimation problem for scenes with an insufficient number of features is outlined in Section 15.1.1 on page 454.

### 8.5 Applications of the segmentation system

The segmentation masks that are generated by a segmentation algorithm have no direct application themselves. However, to determine the segmentation masks is an important step of higher level, object-oriented video processing. In this section, we discuss some example applications.

### 8.5.1 MPEG-4 video coding

The proposed segmentation system has been designed to enable an easy integration into an object-oriented MPEG-4 video codec. All necessary input data for the MPEG-4 encoder (background sprite images, camera-motion parameters, and binary object masks) is generated by the segmentation system. The central question is, whether it is possible to obtain a better image quality, if shaped object coding with sprite coding (shape + sprite coding) is used instead of standard rectangular frame coding. The advantage of the sprite coding mode is that the background image only has to be sent one. On the other hand, the background can also be coded efficiently with standard motion-compensated coding. Furthermore, object-oriented coding requires the transmission of additional shape information for the foreground objects.

To compare the coding efficiency for both coding modes, we carried out the experiment to encode the stefan sequence using both approaches at various bit-rates. For these experiments, we used a beta-version (MoMuSys-FDIS-V1.0-990812) of the MoMuSys reference encoder software [94] to generate MPEG-4 compliant bitstreams. The obtained rate-distortion curve is depicted in Figure 8.16, where the PSNRs for luminance and chrominance are shown separately. According to this result, the sprite-coding mode can in fact achieve a much lower bit-rate for a similar PSNR. However, even for


Figure 8.16: Rate-distortion curve for rectangular MPEG-4 coding vs. shape+sprite coding of the stefan sequence. PSNR for luminance and chrominance is shown separately. The ratedistortion curve includes the bits for sprite coding (approx. 1800 bits/frame averaged). Two decoded pictures at the indicated PSNR of 27.9 dB are depicted in Fig. 8.18.


Figure 8.17: Difference image between an input image and a decoded image of the shape+sprite coding mode, that was encoded with a very high bit-rate. It is visible that the complete sprite is displaced by a small geometric misalignment. This reduces the PSNR even though it is hardly visible by a human observer. Note that the foreground object is not affected by the shift.
 (sprite+object).

(b) Rectangular frame coding, PSNR-Y $=26.84 \mathrm{~dB}$, bits/frame $=12635$.

Figure 8.18: Comparison of the coding quality for rectangular frame coding and sprite + shaped object coding. Even though the PSNR of both sequences is almost similar, the visual quality of the shape + sprite coding is significantly better.

(g) Panoramic view.

Figure 8.19: Foreground object from the stefan sequence placed onto a new background.
the same PSNR values, we observed that the sprite-mode provides better subjective image quality than the rectangular coding mode. As an example, Figure 8.18 shows decoded pictures of both coding modes at 26.9 dB .

There are several reasons, why the PSNR does not match the subjective quality. These reasons are addressed below.

- In the sprite-warping process, the background image can be reconstructed slightly shifted relative to the original image. This is due to inaccuracies in the motion computations, or in camera-lens distortions that cannot be reconstructed from the sprite image. These small geometric distortions are hardly visible to a human observer, but the objective quality measurement is affected because the PSNR is computed over non-corresponding pixel positions. To illustrate this effect, Fig 8.17 shows the difference image between the input image and an image encoded with the shape+sprite coding mode at a very high bit-rate. It is visible that the background sprite has a small spatial offset. Clearly, the foreground object region is not affected by this effect.
- During the construction of the sprite-image, the images are geometrically transformed and filtered several times. This leads to a difficult implementation of the processing of pixels near image borders. In the end, this can lead to missing pixels in the sprite image, such that undefined pixels are copied from the sprite into the output image (see the bottom-left corner of Fig. 8.18).
- Segmentation errors along the object boundary lead to pixels that should be part of the foreground object, but which are simply replaced by background content. Most frequently, these are soft shadows or thin objects.

In our example, the bit-rate for the shape+sprite coding mode includes the bits for sprite and shape. Averaged over the complete sequence, the number of bits/frame for the background sprite is about 1800 bits/frame, and approximately 530 bits/frame for the binary-shape information.

### 8.5.2 Video editing

An alternative application area for object segmentation can surely be found in video editing systems. Current video editing systems operate usually at the frame-level. They enable the cutting of sequences and they can generate transitions between scenes, but they provide few tools to manipulate the video content itself. However, for still images, it is a common editing
operation to cut out objects in one image and paste them into a new image. To extract an object from an image, its boundary is usually marked manually. Special tools for manual segmentation can simplify this operation (see Chapter 11). However, for video sequences, manual segmentation is impractical because of the amount of work involved. Automatic segmentation algorithms can assist to define the object masks and to cut-and-paste the video object into a new sequence.

As an example, we have used automatic segmentation to extract the foreground object and the camera-motion parameters from the stefan sequence. Moreover, we composed a new background sprite image from a set of still images with a panoramic image stitching program (see Chapter 14). By simulating the camera motion on the new background image and superimposing the foreground objects, we can generate the illusion that the object is acting in the new environment (see Fig. 8.19).

Using the same input data, it is also possible to keep the background image static, but project the foreground objects at the current camera view, which is defined by the global-motion parameters. With this approach, we obtain a static visualization of the object motion (Fig. 8.19(g)).

### 8.5.3 Pseudo 3-D video generation

Object segmentation also plays an important role in the generation of pseudo 3-D video content from $2-\mathrm{D}$ sequences. The need for $3-\mathrm{D}$ video sequences is quickly gaining interest because new developments of display technology enable the presentation of 3-D images without special glasses (e.g., with red/blue or polarization filters). However, the introduction of 3-D television into the market suffers from the chicken-and-egg problem of suitable video content. Consequently, it is important at least for the transition phase to generate pseudo 3-D video from existing standard 2-D recordings.

One approach for realizing this could be to use automatic foreground object segmentation and to arrange the foreground objects and the background into different depth-layers, such that foreground objects appear closer to the viewer. As an experiment, we applied this approach to the stefan sequence to generate virtual 3-D views, which are depicted in Figure 8.20 as anaglyph images. In these images, we have defined the background layer as reference plane. Objects that are closer to the viewer are displayed with a horizontal offset between the left and right view. The value of this offset determines the depth of the object (Fig. 8.21).

(a) Frame 15 .

(b) Frame 100 .

Figure 8.20: Pseudo-3D generation from the segmented stefan sequence (view this with red/blue-glasses). The foreground objects are placed closer to the viewer.


Figure 8.21: Geometry of stereoscopic imaging. Relative to the background image, the foreground object appears shifted between the left and right images. The closer the object is located towards the viewer, the larger the is the offset between the two views.

### 8.5.4 Video-object recognition

The segmentation of video objects is also the first step for higher-level video analysis like the recognition of objects. Once the exact object mask is known from the segmentation, we can use for example the object shape to obtain more semantic information about the object itself. This includes the classification of the object into pre-defined categories, the recognition of specific objects, or the analysis of the object behaviour.

We have implemented a prototype application that compares the object outline with a database of objects to determine the object class. In our example, we used different shapes of humans that were labeled with their current action (walking, standing, sitting, standing-up). Each frame of the input object was classified into one action category. To increase the robustness, we also included a transition model between these states (the sitting-state cannot directly change to walking, but has to transit the standing-up state first). The approach is described in more detail in Appendix F. An example result is visualized in Figure 8.22.

### 8.6 Extensions

The proposed segmentation system can be extended in several ways to adapt it to special applications, while still keeping the general core. In the following, we briefly mention some possibilities of extensions. Some of these extensions are discussed in detail in the second part of this thesis.


Figure 8.22: Automatic segmentation was applied to a sequence that shows an actor (see Fig 8.11). Based on the object shape, the action in each frame was classified into the classes walking, standing, sitting, standing-up.

### 8.6.1 MPEG-4 coding with sprite-mode detection

When general video sequences are compressed with MPEG-4, it is not efficient to use the sprite coding-mode for the complete sequence. Many parts of typical video sequences comprise camera motion along 3-D paths or complex motion that cannot be reconstructed from a static background sprite. A practical encoder should automatically detect those scenes in which sprite-coding can be used efficiently.

One approach to implement this detection is to observe internal parameters of the feature-based global-motion estimator (Chapter 4). Remember that this estimator uses the RANSAC algorithm to identify the dominant global-motion model. The number of inlier vectors in this process indicates how well the estimated model fits to the observed motion. If the fraction of inliers is small, it means that the scene motion is not described very well by the estimated parameters, and a standard non-sprite coding mode should be used.

### 8.6.2 Camera auto-calibration

The global-motion parameters that we estimated represent the motion of a rotational camera. However, the parameters do not correspond directly to physically meaningful parameters like camera rotation angles or the focal length. Knowing the physical parameters enables several new applications. For example, the computer could superimpose artificial 3-D objects into
the scene that move consistently with the camera motion. Moreover, the physical parameters can be used to control a motorized pan/tilt/zoom camera to simulate the camera motion of the input video. The latter can be applied in the video editing process to capture new background scenes for an existing video sequence. The foreground objects can be extracted from the old video sequence and placed in the new sequence. The advantage of this approach compared to simply replacing the background sprite image (Fig. 8.19) is that the new scene "background" can contain arbitrary object motion. An algorithm to extract physical camera parameters from the global-motion parameters is described in Chapter 12.

### 8.6.3 Absolute coordinate transfer

For the analysis of video sequences, it is often required to describe the object positions not only in terms of image coordinates, but in real-world coordinates. One example is the analysis of sport videos like soccer or tennis. In this case, not the position of the players on the display is relevant, but the position on the playing field or on the tennis court. Another example are surveillance applications, where certain areas should be monitored for intrusion.

If the camera is static, specific points with known coordinates can be identified to manually calibrate the system. However, a static mapping between image coordinates and real-world coordinates is impossible if camera motion is visible in the input. In Chapter 13, we propose an algorithm for automatic camera calibration in sports-sequences.

### 8.6.4 Object models

The proposed segmentation system is based on the definition that everything that differs from a static background image is classified as foreground. As can be observed in the segmentation results, this definition is not always sufficient for all applications. In many situations, there are several moving objects in an image, but we are only interested in one of them (Fig. 8.8). On the contrary, sometimes only parts of the interesting object shows in the segmentation, since parts of the object do not move (Fig. 8.14). To separate the object of interest from the remaining objects, we have to define more specifically what objects should be extracted.

The difficulty of this is to find a model that is accurate enough to detect the correct object, but on the other hand, it has to be robust against object deformations, occlusions and similar problems. We propose an algorithm for general object detection in Chapter 9 and 10.

## Part II

## Segmentation Using Object Models

It is a mistake to think you can solve any major problems just with potatoes. (Douglas Adams)

# Object Detection based on Graph-Models I: Cartoons 

The first part of this thesis has presented an automatic segmentation system that does not use any pre-knowledge about the foreground objects to be extracted. However, sometimes it is desired to specify more precisely which objects should be extracted. This allows to disambiguate between the important acting objects and unimportant objects that should not be extracted (like the audience in sport videos). Moreover, having a model about the objects can provide a better segmentation in cases where the scene background is not known or where it is very similar to the foreground. This and the following chapter present a graph-based model to describe video objects. This model is applied to extract a user-defined object from video sequences or still images. The model-detection algorithm is based on an inexact graph-matching between the user-defined object model and an automatically extracted graph describing the input image. This chapter discusses the model-generation process, and we present a matching algorithm tailored to the object detection in cartoon sequences. Cartoon sequences are especially difficult to process with ordinary segmentation algorithms, but they are fitting well to the graph-model based approach. The successive chapter further extends the object-detection algorithm to natural video sequences.

### 9.1 Introduction

In the first part of the thesis, an automatic segmentation system was described that is based on a background-subtraction technique. Background subtraction employs knowledge about the scene background, but it uses no pre-knowledge about the foreground objects. This has the advantage that no information about the foreground objects is required. On the other hand, a simple background subtraction cannot decide if the detected objects are important for the successive processing steps. In many cases, we are only interested in some of the visible objects, even though more objects are extracted. One example are sport videos, where the athletes are extracted, but also moving people in the audience. In contrast with this, we also experience problems when only part of the important object is moving, like in news broadcasts in which the anchorman usually only moves his head. In this case, the previously proposed segmentation system cannot know that the body of the anchorman is not simply part of the background. These problems can only be solved by providing the segmentation algorithm with pre-knowledge about the objects to be extracted.

A video object can generally appear in many different deformed or articulated appearances in an image (see Fig. 9.1). Moreover, it can be occluded or viewed from different sides. The crucial problem in defining an object model is to find a representation that is flexible enough to fit to all the different appearances of the object. On the other hand, the model should not be too general, since it will fit to incorrect places otherwise.

In this chapter, we introduce a system for video-object detection and extraction based on user-defined models. Our object models are described by "model graphs", in which nodes represent image regions and edges denote spatial proximity. Each node is attributed with color and shape information about the corresponding image region. Model graphs are specified manually based on a sample image of the object. Object recognition starts with automatic color segmentation of the input image. For each region, the same features are extracted as specified in the model graph. Recognition is based on finding a subgraph in the input graph that matches the model graph. Evidently, it is not sufficient to search for an isomorph subgraph, since node and edge attributes will not match exactly. Furthermore, the automatic segmentation step leads to an oversegmented image. For this reason, we employ inexact graph matching, where several nodes of the input graph may be mapped onto a single node in the model graph.

Graph matching is a well-known technique in computer vision and several efficient heuristics have been developed for the graph isomorphism problem. These include algorithms based on nonlinear optimization [76], quadratic programming [163, 145], relaxation labeling [184], or algorithms


Figure 9.1: The same object can appear in many different shapes.
that are specialized for a specific class of graphs [43]. A completely different approach to region correspondence uses the Earth Movers Distance (EMD), which is a popular distance measure in the field of image retrieval [79]. Recently, region-based algorithms have also become popular in the context of searching in video databases [17, 112, 23]. In this case, characteristic regions are first extracted from a query image and subsequently, these regions are used to form a database query for images that contain similar regions. However, since the relevant regions are extracted automatically, no pre-knowledge about the spatial object structure is available. Consequently, the object structure is often neglected.

In this chapter, we concentrate on the object recognition in cartoon sequences. This class of sequences is difficult to handle with current automatic segmentation algorithms, because the motion estimation has difficulties arising from large homogeneous regions and because the object appearance is typically highly variable. In the next chapter, we extend the graph-model based object detection approach to a similar detection system for natural images.

### 9.2 Principle of region-based graph matching

Our approach for object detection is based on the assumption that objects can be described reliably by a set of attributed regions and their spatial relationship. The model structure and features are expressed by an object model graph $G_{M}=\left(V_{M}, E_{M}\right)$, where each node in $V_{M}$ represents an image region with uniform color. Nodes have attributes describing region color, shape, and size. Edges in the model graph define spatial proximity. This means that if $\left(v_{1}, v_{2}\right) \in E_{M}$, region $v_{1}$ must be near region $v_{2}$. The model graph representation allows to recognize objects independent of their exact
spatial layout as long as the characteristic spatial structure of the objects remains. In particular, articulated object motion can be modeled in a straightforward way.

Model graphs are defined manually by the user in a graphical editor (top part of Figure 9.2). To ease the definition, a sample image of the object can be segmented semi-automatically. Subsequently, the region features are extracted automatically from the sample image. Finally, the spatial structure of the object is defined by connecting neighboring regions.

Object recognition starts with an automatic color segmentation of the input image. For each of the regions obtained from this segmentation, the same features are extracted as for the model graph (bottom part of Figure 9.2). A fully connected input graph is defined, generating nodes from the regions and attributing the edges with the distance between pairs of regions. Since the regions are generated by an automatic segmentation process covering the whole image, this input graph will be much larger than the model graph. Furthermore, due to oversegmentation, regions that belong together semantically may be split into separate regions.

The object recognition is based on the idea to find a subgraph in the input graph that matches our model graph. Obviously, it is not possible to find an isomorph subgraph, because node and edge attributes will not match exactly and model-graph regions are possibly split into oversegmented regions. Hence, we apply an inexact graph matching where several nodes of the input graph can be mapped onto a single node in the model graph (1 : $N$-matching). The quality of a match is described by judging the compatibility of the node and edge attributes.

In order to reduce the high computational complexity of graph matching, we employ a fast three-step matching algorithm.

- The first step reduces the search space by eliminating nodes in the input graph that are very unlikely to occur in the match.
- The second step performs a 1:1-matching of the skeleton tree of the model graph. The skeleton tree is a sub-graph $T=\left(V_{M}, E_{S}\right)$ (with $E_{S} \subseteq E_{M}$ ) of the model graph that only contains a subset of the edges such that it forms a tree. This 1: 1-matching of the skeleton tree can be carried out very efficiently using a dynamic-programming approach.
- The third matching step considers the whole model graph and extends the matching to a $1: N$-matching.

Figure 9.2: Framework of the object-detection system. First, an object model is defined by the user in a graphical object-model editor (top part of drawing). The object detection starts with an automatic color segmentation of the input image (bottom part of drawing), which results in a large input graph. A matching algorithm localizes the position of the model graph in the input graph to identify the regions that belong to the object.


Figure 9.3: Marker-driven manual watershed algorithm. Whereas in the standard algorithm each local minimum creates a new segment (a), the marker-driven watershed algorithm only builds watersheds between markers (b). Note that the markers $M_{2 a}$ and $M_{2 b}$ are assigned to the same region, so that no watershed is built between them.

### 9.3 Model editor

This section describes the editor for generating the model graphs. The object models which are used during the object-detection process, are defined manually by the user in a graphical editor. Manual user interaction is required, because only the user knows exactly the semantic meaning of the object model and thus only he can specify the characteristic attributes of a particular object. Since the model specification is an easy task and the models can be saved into a database of frequently-used models, the required time for user interaction is low.

Segmentation of the object regions is based on a marker-driven watershed algorithm, which is applied on the gradients of a sample image. The difference to the standard watershed algorithm is that the water does not start flooding from the various local minima. Instead, the water commences to flow from the markers. Several markers can be grouped together such that water basins of these markers are attributed to the same region (i.e., no watershed is built between markers for the same region, see Figure 9.3(b)).

Relevant object regions are defined manually by placing markers in a sample image. The exact region boundaries are subsequently located by the watershed algorithm. Errors in the segmentation can be corrected by joining regions that have been separated by the watershed algorithm. Internally, this is realized by considering the markers of both regions equivalent (compare the markers $M_{2 a}, M_{2 b}$ in Figure 9.3(b)). The region attributes are extracted from the sample image (see Figure 9.4(d)), but they can be modified by the user in case the sample image does not contain a typical view of the object.

Finally, graph edges are added to define regions that should be close


Figure 9.4: The creation process of a model graph. Based on the sample input image (a), the user places markers into the image to separate the regions (b). Edges are introduced (c), where edges of the model skeleton tree are depicted with strong red lines, and the fine green lines denote the refinement edges used in the 1:N matching step. The region features can be visualized in an abstract presentation (d).


Figure 9.5: Result of the input graph obtained from the automatic color segmentation. Each color region established a node in the input graph, enriched with features like node color and size.
to each other, independent of a specific object view. Note that the model should only contain those edges that are semantically necessary for the specific object. For a model of a human, for example, the head is directly connected to the body, but not to one of the arms even though this may be the case in a specific sample image.

As we will see later when considering the matching algorithm, matching becomes particularly efficient when the model graph has a tree topology. Therefore, we classify the model graph edges into two classes: skeleton tree edges, and refinement edges. The principal 1:1-matching step only uses the skeleton tree edges. This forms no severe restriction, since most natural objects can be described sufficiently using trees. The refinement edges are used in the $1: N$-matching step when oversegmented regions are combined to cover the whole object.

### 9.4 Automatic color segmentation

Automatic color segmentation is carried out using a combination of watershed segmentation and region merging. The watershed algorithm provides a very fast pre-segmentation, but this is usually strongly oversegmented and thus not sufficient for our purpose. Hence, an additional region-merging algorithm is applied on the pre-segmentation result to further combine neighboring regions obtained from the watershed algorithm. Although the watershed pre-segmentation is not required, it considerably speeds up the segmentation process, because the region-merging algorithm can start with larger initial regions (more detail about the color segmentation can be found in Appendix E).

Region merging has proven to be a powerful segmentation algorithm, enabling the use of various merging criteria to control the merging process. We have chosen the Ward criterion, which results in a segmentation in which the region variance is minimized. The fundamental idea is to consider every neighboring pair of regions and calculate the increase of variance that a possible merge of the two regions would impose. Let $\sigma_{i}^{2}$ denote the variance of region $r_{i}, \mu_{i}$ the region mean brightness, and $\left|r_{i}\right|$ the region size. Then, the increase of variance when regions $r_{i}$ and $r_{j}$ are merged, can be calculated by the new variance $\sigma_{i j}^{2}$ minus the original individual variances, which gives

$$
\begin{equation*}
\Delta_{i j}=\sigma_{i j}^{2}-\sigma_{i}^{2}-\sigma_{j}^{2}=\frac{\left|r_{i}\right| \cdot\left|r_{j}\right|}{\left|r_{i}\right|+\left|r_{j}\right|}\left(\mu_{i}-\mu_{j}\right)^{2} \tag{9.1}
\end{equation*}
$$

The region-merging algorithm now successively combines the two regions $r_{a}, r_{b}$ for which $\Delta_{a b}$ is minimal until the minimum $\Delta_{a b}$ exceeds a threshold.

We denote the final set of regions as $R=\left\{r_{i}\right\}$. A more in-depth treatment of the color segmentation is provided in Chapter 10.

### 9.5 Feature extraction and matching criteria

To evaluate the similarity of image regions, a set of features is extracted for each region during model creation, as well as for each region generated by the automatic segmentation process. Based on these features, node and edge cost-functions are defined which serve as matching criteria in the graph matching step. The calculation of features that are not required for candidate selection (see below) can be postponed after the candidate selection step. Since only a smaller subset of the regions is actually used in the matching process, the computation time is reduced.

### 9.5.1 Color

The color of each region is described by its coefficients in the Hue, Value, Saturation (HVS) color space. This color space allows an easy definition of a distance metric having a close relationship with the human perception. HVS space can be visualized as a cone with black at the tip and the rainbow colors around the base. After transforming the HVS color ( $h, v, s$ ) with $h \in[0 ; 2 \pi], v, s \in[0 ; 1]$ into cartesian coordinates using $x=v \cdot s \cdot \cos h$, $y=v \cdot s \cdot \sin h, z=v$, we use the Euclidean distance between the two colors as color matching cost. We denote the matching cost for assigning region $r_{i} \in R$ to model node $m_{i} \in V_{M}$ as $C_{m_{i}}^{C}\left(r_{i}\right)$.

### 9.5.2 Size

The shape feature is simply the size of a region in pixels. During the matching process, two cost measures are used for region sizes: one based on the absolute region size and one based on relative region sizes. The absolute region size measure is applied during the candidate selection step to sort out regions that are much larger than the object model. The absolute size feature is computed as the ratio of the input region size $\left|r_{i}\right|$ with respect to the model region size $\left|m_{i}\right|: f_{m_{i}}^{S}\left(r_{i}\right)=\left|r_{i}\right| /\left|m_{i}\right|$.

For the computation of the relative size measure, let $\left|r_{i}\right|,\left|r_{j}\right|$ be the sizes of two connected regions and $\left|m_{i}\right|,\left|m_{j}\right|$ the sizes of the corresponding model regions. Since the size of the object in the image may vary, we do not compare the absolute region sizes to the model in the actual matching step. In fact, only the relative sizes of connected regions are compared to the model. Following this approach, we define the matching cost $C_{m_{i}, m_{j}}^{R e l S}\left(r_{i}, r_{j}\right)$ by the piecewise linear function depicted in Figure 9.6. This measure does


Figure 9.6: Relative size cost $C_{m_{i}, m_{j}}^{R e l S}\left(r_{i}, r_{j}\right)$ for matching a pair of connected model nodes to a pair of input regions.
not penalize variations of region sizes up to a factor of two. This is to be robust for varying region sizes because of many factors like occlusions, differing viewing position, deformable objects, or inaccurate segmentation.

### 9.5.3 Distance

Connected model regions are assumed to have zero distance. However, the distance between a pair of input regions $r_{i}, r_{j}$ is measured as the minimum pixel distance $d\left(r_{i}, r_{j}\right)$ between both region borders (see Fig. 9.7(a)). The region distance-cost is defined as (see Fig. 9.7(b))

$$
C^{D}\left(r_{i}, r_{j}\right)= \begin{cases}0 & \text { for } d\left(r_{i}, r_{j}\right)<d_{\min }  \tag{9.2}\\ \frac{d\left(r_{i}, r_{j}\right)-d_{\min }}{d_{n o r m}-d_{\min }} & \text { else }\end{cases}
$$

Truncating the error for small distances has been introduced to tolerate small region distances between input regions. These small distances can be caused by an inaccurate segmentation. We have chosen $d_{\min }=5$ and $d_{\text {norm }}=30$ pixels in our experiments, in which the image size was $720 \times 576$ pixels.

### 9.5.4 Shape

Automatic segmentation usually generates some regions having a "fuzzy" shape, being thin and having many concavities. These regions almost never belong to any object, but rather appear in background regions between objects. Since the regions are often located near object boundaries, they are close to all regions in the object and thus, for the matching algorithm, they seem to be part of the object. Hence, it is preferable to early identify these regions and exclude them from the matching.

To find such "misleading" regions, we make use of a shape feature that describes the shape complexity of a region. It is computed as

$$
\begin{equation*}
f^{S h}\left(r_{i}\right)=4 \pi \frac{\left|r_{i}\right|}{\operatorname{border}\left(r_{i}\right)^{2}}, \tag{9.3}
\end{equation*}
$$



Figure 9.7: (a) Region distances are calculated as the minimum Euclidean distance between the region borders. (b) The cost function increases linearly with the distance, but tolerates small distances between regions due to inaccurate segmentation.
where $\operatorname{border}\left(r_{i}\right)$ is the length of the region border and $\left|r_{i}\right|$ is the region area. Clearly, the shape feature is maximal $\left(f^{S h}=1\right)$ when the region boundary is a circle and approaches zero when the region is long and thin. Note that $f^{S h}$ is invariant to scaling.

### 9.5.5 Orientation

Edge orientation is an optional matching criterion and can be activated manually for each individual edge. When matching symmetric objects, the orientation of the matched graph is ambiguous. To break this symmetry, edges can be declared as oriented edges. These edges remember which of the two regions is left of the other (or above the other). The relative orientation of two regions is determined by comparing the coordinates of their centers of gravity. If the model edge $e=\left(m_{i}, m_{j}\right)$ is an oriented edge and the orientation of the assigned input regions $r_{i}, r_{j}$ differs, the costs is set to $C_{m_{i}, m_{j}}^{O}\left(r_{i}, r_{j}\right)=1$; otherwise we set $C^{O}=0$.

### 9.5.6 Node and edge costs

The above-mentioned costs are combined into node-cost and edge-cost functions, which are computed by

$$
\begin{equation*}
C_{m_{i}}^{N}\left(r_{i}\right)=\alpha C_{m_{i}}^{C}\left(r_{i}\right), \tag{9.4}
\end{equation*}
$$

for the node cost and

$$
\begin{equation*}
C_{m_{i}, m_{j}}^{E}\left(r_{i}, r_{j}\right)=\beta C^{D}\left(r_{i}, r_{j}\right)+\gamma C_{m_{i}, m_{j}}^{R e l S}\left(r_{i}, r_{j}\right)+\theta C_{m_{i}, m_{j}}^{O}\left(r_{i}, r_{j}\right) \tag{9.5}
\end{equation*}
$$

for the edge cost, respectively, where the parameters $\alpha, \beta, \gamma, \theta$ are weighting factors which we have set to 1 . They can be increased or decreased


Figure 9.8: Visualization of computing the distance cost between two sets of regions. The distance is defined as the minimum distance cost of all regions pairs $\left(r_{a} \in q_{i}, r_{b} \in q_{j}\right)$ plus the minimum distance costs between the regions in each set.
depending on the application. For example, when it is $a$-priori known that the color may vary because of differing lighting conditions, the weight of the color cost $\alpha$ should be decreased.

### 9.5.7 Generalization of costs for $1: N$-matching

For the $1: N$-matching, we generalize the cost measures to handle mappings from several input regions to a single model region. The measures are defined such that the cost measures for 1:1 matching result as a special case. We denote the generalized cost definitions with hats on the variable names. We define the 1:N-matching cost for model nodes with $q_{i} \subset R$ as

$$
\begin{equation*}
\hat{C}_{m_{i}}^{N}\left(q_{i}\right)=\frac{1}{\sum_{r \in q_{i}}|r|} \sum_{r \in q_{i}}|r| \cdot C_{m_{i}}^{C}(r), \tag{9.6}
\end{equation*}
$$

which is the sum of all node costs for the region in $R$, weighted with the region size.

The generalized distance measure has to capture the distance between two sets of regions. At the same time, it should also prevent that the regions within one set are itself distributed over the image with large distance. Hence, we introduce the coherency of a set of regions as the pairwise spatial proximity of the regions within both sets. This leads to our definition of
the generalized distance measure, which we compute as (see Fig. 9.8)

$$
\begin{align*}
\hat{C}^{D}\left(q_{i}, q_{j}\right)= & \underbrace{\operatorname{mistance}^{r_{a} q_{i}, r_{b} \in q_{j}} C_{\text {distateen }} C^{D}\left(r_{a}, r_{b}\right)}_{\text {minimum }}+ \\
& \text { both sets of regions }
\end{align*} \quad \underbrace{\frac{1}{2} \sum_{r_{a} \in q_{i}} \min _{r_{b} \in q_{i}, r_{b} \neq r_{a}} C^{D}\left(r_{a}, r_{b}\right)}_{\text {coherence of region } q_{i}}+\underbrace{\frac{1}{2} \sum_{r_{b} \in q_{j}} \min _{r_{a} \in q_{j}, r_{a} \neq r_{b}} C^{D}\left(r_{a}, r_{b}\right)}_{\text {coherence of region } q_{j}} .
$$

Note that this definition of coherency would assign two distant clusters of regions a high coherency, which is not desired. However, in the $1: N$ matching algorithm, regions are added one by one. Consequently, the set of regions cannot grow easily into two clusters of regions, because the cost for the first distant regions would be high.

The generalized cost functions for relative region size and orientation $\hat{C}^{\text {RelS }}$ and $\hat{C}^{O}$ are computed by determining the sum of region sizes and the center of gravity for the set of regions. The generalized total edge cost $\hat{C}^{E}$ is defined as the weighted sum of the individual costs similar to the definition of $C^{E}$ in Eq. (9.5), replacing all cost components $C$ with the generalized counterpart $\hat{C}$.

### 9.6 Matching algorithm

Graph matching is carried out in a three-step process.

- Step 1 Since the color or size of many of the regions generated by the automatic segmentation will strongly deviate from the model regions, they can be excluded from the matching process to decrease the computational time. The first matching step determines for each model region a subset of input regions and performs the previously mentioned exclusion of unsuitable regions.
- Step 2 The second matching step involves only the regions of the selected subset as candidates for a model node. The second matching step computes a 1:1 matching of the model graph skeleton tree using a dynamic programming approach.
- Step 3 This 1: 1 matching acts also as the initialization for the third matching step, where the $1: 1$ matching is enriched to form a 1: $N$ matching. Enriching means that additional input regions can be assigned to a single model region to decrease overall cost.

The three matching steps can also be viewed as incrementally imposing additional structural information. While the first step (candidate selection) is completely free from any structural constraints, the 1:1-matching obeys the structure of the model skeleton tree, and the final $1: N$-matching step considers the full model-graph structure.

### 9.6.1 Candidate-region selection

The candidate input regions for a model region are selected based on the color, the region size and the shape feature. The idea of the candidateselection step is to sort out regions that have the wrong color, a clearly wrong size, or a non-compact shape. Two selection strategies are possible: we can fix the number of candidates $N_{C}$ for each model region and select the $N_{C}$ best input regions as candidates, or we can set a threshold on the region similarity and consider all input regions with higher similarity as candidates. The choice of selection strategy is not critical when the number of candidates is sufficiently high and the thresholds are set high enough to ensure that the correct matches are not sorted out. We adopted a strategy with fixed number of candidates for each model region and observed that about $10-20$ candidates for each model region are sufficient.

At first, we filter out regions that are a factor $\theta$ larger than the corresponding model region or that have a significantly different shape. We apply two thresholds

$$
\begin{equation*}
C_{v}^{S}(c(v, i))>\theta \quad \text { and } \quad f^{S h}(c(v, i))>\nu \tag{9.8}
\end{equation*}
$$

which were chosen as $\theta=3$ and $\nu=0.15$ in our experiments. These values can be adjusted or even selected individually for each model region, depending on the amount this region can change its size in different input images and on the application or model. Since $\theta$ and $\nu$ are only used to sort out clearly non-matching regions, they can be set arbitrary large or even can be omitted at all (but more candidates would have to be considered in this case). Since the automatically segmented regions are possibly only part of a single model-graph region and the total size of the object to be found in the input image is not known yet, regions that are too small should not be excluded. The remaining regions are sorted with a mapping

$$
\begin{equation*}
c: V_{M} \times\{1,2, \ldots,|R|\} \rightarrow R, \tag{9.9}
\end{equation*}
$$

such that

$$
\begin{equation*}
C_{v}^{C}(c(v, i))<C_{v}^{C}(c(v, j)) \rightarrow i<j . \tag{9.10}
\end{equation*}
$$

Hence, $c$ sorts the input regions according to increasing matching costs, with the best matching region $c(v, 1)$ for model node $v$, and the worst matching region that is still considered $c(v,|R|)$.

Note that the same input region can appear as candidate region for several model regions. The constraint that the same input region must not be assigned twice to different model nodes must be satisfied by the following algorithmic step.

### 9.6.2 Matching algorithm

The 1 : 1-matching step is the most important step, since it does the primary localization of the model regions. The found matches are the seed for the 1: $N$-matching step, where they are further extended with additional input regions.

Weighted graph-matching can be described as finding the maximum weight clique in the corresponding association graph [144], which is known to be $N P$-hard. However, for special classes of graphs, such as e.g. trees, efficient algorithms exist. Since almost all real-world objects can be accurately described by trees and because efficient algorithms for trees exist, we restrict our 1: 1-matching step to finding the best matching tree for the skeleton tree of a model graph.

Our algorithm is based on a dynamic-programming approach. The objective is to find the mapping $\mathcal{M}_{1: 1}: V_{M} \rightarrow\left\{1,2, \ldots,\left|N_{C}\right|\right\}$ that minimizes the sum of node costs and edge costs in the tree:

$$
\begin{align*}
\min _{\mathcal{M}_{1: 1}}\{ & \sum_{v \in V_{M}} C_{v}^{N}\left(c\left(v, \mathcal{M}_{1: 1}(v)\right)\right)+ \\
& \left.\sum_{\left(v_{1}, v_{2}\right) \in E_{S}} C_{v_{1}, v_{2}}^{E}\left(c\left(v_{1}, \mathcal{M}_{1: 1}\left(v_{1}\right)\right), c\left(v_{2}, \mathcal{M}_{1: 1}\left(v_{2}\right)\right)\right)\right\} . \tag{9.11}
\end{align*}
$$

Let us introduce the concept of computing the minimum cost mapping with a simple example. Assume that the model tree is e.g. a simple linear chain (Fig. 9.9(a)). We construct a computation graph by duplicating each model node to $N_{C}$ nodes, each representing the decision that the model region is mapped to a specific candidate node. The node costs $C^{N}$ are assigned to the nodes, i.e., the first column of nodes get costs $C_{a}^{N}(c(a, 1)), C_{a}^{N}(c(a, 2)), \ldots, C_{a}^{N}\left(c\left(a, N_{C}\right)\right)$. Similarly, the edge costs $C^{E}$ are assigned to the edges. Now minimizing the sum (9.11) is equivalent to computing the minimum cost path through the resulting computation graph. To compute the minimum cost path, we proceed column by column from left to right and determine for each node the predecessor node that gives the minimum total cost so far. More specifically, we assign attributes mincost and last to each node in the computation graph. The nodes in the left column are initialized with mincost equal to their respective node


Figure 9.9: Object-model skeleton trees with their respective computation graphs.
costs $C^{N}$ and last $=$ nil. Continuing with the next column to the right, we calculate for each node the total cost that results from each choice for the predecessor candidate. This cost consists of the mincost of the predecessor node, the edge cost $C^{E}$ linking the predecessor node to the current node and the current node cost $C^{N}$. The predecessor node that gave the least cost is stored into last and the corresponding minimum cost in mincost. When we arrive at the rightmost column, the candidate with the minimum cost is selected and the minimum cost path is traced back using the last attributes.

If the model tree contains junctions like shown in Figure 9.9(b), the algorithm above has to be extended. Since model node $c$ has multiple incoming edges, the best predecessor candidate has to be selected from both, model node $b$ and model node $f$. Consequently, mincost is now obtained by minimizing over the sum of all previous nodes and incoming edge costs. The computation time required therefore increases from $N_{C}^{2}$ steps for a column to indegree $\cdot N_{C}^{2}$ computations (indegree $=2$ in our example). However, the total computation time does not increase, because the total number of edges in the computation tree remains constant. Hence, the complexity is $O\left(N_{C}^{2} \cdot\left|V_{M}\right|\right)$. The complete matching algorithm is described in Algorithm 1 and 2. Algorithm 1 initializes the pred attributes that define the order in which the model nodes have to be considered in the calculation. The set $\operatorname{pred}(v)$ of a node $v$ is the set of adjacent nodes that have to be

```
\(\overline{\text { Algorithm } 1 \text { Initialization of the computation graph for the subsequent }}\)
dynamic-programming algorithm.
Require: the model tree \(V_{M}=\left\{v_{1}, \ldots, v_{N}\right\}, E_{S} \subset V_{M} \times V_{M}\)
    \(\operatorname{pred}\left(v_{1} \in V_{M}\right) \leftarrow \emptyset\)
    \(l \leftarrow\left\{v_{1}\right\}\)
    \(r \leftarrow V_{M} \backslash\left\{v_{1}\right\}\)
    while \(l \neq \emptyset\) do
        select an arbitrary \(v \in l\) and set \(l \leftarrow l \backslash\{v\}\)
        for all \((v, w) \in E_{S} \wedge w \in r\) do
            \(\operatorname{pred}(v) \leftarrow \operatorname{pred}(v) \cup\{w\}\)
            \(l \leftarrow l \cup\{w\}\)
        end for
    end while
```

processed prior to node $v$. This attribute defines the depth-first recursion order in which the costs for the nodes are calculated. Algorithm 2 performs the actual 1:1 matching. It processes all nodes in the depth-first order defined by $\operatorname{pred}()$. The algorithm is initialized to start at the tree root: calccolumn $\left(v_{1}\right)$.

In each junction node, instead of only storing a single predecessor, we haev to store the best candidate node for each incoming model-tree edge. When tracing back the minimum cost path, we obtain a minimum-cost tree instead of a linear chain.

The algorithm described thus far has still one drawback. When the same input region occurs as candidate for different model nodes, the algorithm may use the same input region more than once. This is not desirable. For example, consider searching for a human with equal left and right arm (see Figure 9.10). The model nodes for both arms are the same and both subtrees are connected to the same body node. Since either the left or right arm in the input graph will match better to the model, the algorithm will assign the best one to both arms of the model.

This problem can be alleviated using two techniques. First, it is possible to make the edges connecting the arm and the body oriented edges (see Section 9.5.5) inducing extra cost when the left arm in the input graph is mapped to the right arm in the model graph and vice versa. However, this does not work in all situations and we have to extend the algorithm described previously to prevent double assignments. This can be done by introducing a blocked attribute to each computation graph node. This attribute stores the set of input regions that are used so far. In each junction node $v$ of the computation graph $(|\operatorname{pred}(v)|>1)$, combinations of previous node candidates $k_{1}, k_{2}$ that collide $\left(\operatorname{blocked}\left(k_{1}\right) \cap \operatorname{blocked}\left(k_{2}\right) \neq \emptyset\right)$

```
Algorithm 2 Compute the minimum cost assignment.
    procedure calccolumn \(\left(v \in V_{M}\right)\)
    for all \(w \in \operatorname{pred}(v)\) do
        call calccolumn \((w)\)
    end for
    for \(n=1\) to \(N_{C}\) do
        if \(\operatorname{pred}(v)=\emptyset\) then
            \(\operatorname{mincost}(v, n) \leftarrow C_{v}^{N}(v, c(v, n))\)
        else
            \(\operatorname{cost} \leftarrow C_{v}^{N}(v, c(v, n))\)
            for \(w \in \operatorname{pred}(v)\) do
            \(\operatorname{cost} \leftarrow \operatorname{cost}+\min _{i}\left(\operatorname{mincost}(w, i)+C_{w, v}^{E}(c(w, i), c(v, n))\right)\)
            last \((v, n, w)\) is set to the \(i\) that minimized the above sum
            end for
            \(\operatorname{mincost}(v, n) \leftarrow\) cost
        end if
    end for
```

cannot be selected. Note that since this is a combinatorial problem, the best candidate node for all preceding nodes cannot be determined independently. In fact, all combinations are enumerated and checked for validity. The valid combination with the minimum cost defines the best candidates for the preceding model nodes.

As an example, consider Figure 9.10. Note that Node 3 selects input regions $a$ and $b$ as its left arm. Under the assumption that the right arm looks identical to the left arm in the model, the dynamic-programming algorithm without blocking attribute would select the same input regions for the right arm. However, since $a$ and $b$ are contained in the blocked set of Node $3 b$ and Node $5 b$, Node 2 has to choose Nodes $d$ and $e$ for the right arm. Unfortunately, it cannot ensure that both arms are assigned to the correct side, because the orientation is lost in the graph description. This orientation ambiguity can be resolved by defining the edges connecting the arms with the body as oriented edges.

### 9.6.3 1: $N$-matching

Starting with the 1 : 1 -matching result, the $1: N$-matching algorithm assigns additional input regions to model nodes if this decreases the total cost. We define the $1: N$ matching through a mapping $\mathcal{M}_{1: N}: V_{M} \rightarrow 2^{R}$. It is initialized with the result of the preceding $1: 1$ matching by

$$
\begin{equation*}
\mathcal{M}_{1: N}\left(m_{i}\right):=\left\{c\left(m_{i}, \mathcal{M}_{1: 1}\left(m_{i}\right)\right)\right\} . \tag{9.12}
\end{equation*}
$$



Figure 9.10: Example calculation for a model graph (left) describing a human. The arrows denote the order of calculation as induced by the pred attributes. For simplicity, the computation graph on the right has been constructed with only two candidate nodes for each model node. The model nodes are denoted by numbers and the input nodes by letters. Selected edges are drawn with thick arrows and the corresponding blocked attribute is shown at each node. Calculation proceeds from left to right.

More input regions are added with a greedy algorithm. In each iteration, a cost difference $\delta_{m_{i}}\left(q_{i}, r_{k}\right)$ is computed that equals the cost difference induced from adding input region $r_{k}$ to region set $q_{i}$. As long as the cost difference is below zero, the input region with the largest decrease of cost is added. Otherwise, the algorithm ends. We define the cost difference as

$$
\begin{align*}
\delta_{m_{i}}\left(q_{i}, r_{k}\right)= & \underbrace{\hat{C}_{m_{i}}^{N}\left(q_{i}\right)+\sum_{\left(m_{i}, m_{j}\right) \in E_{M}} \hat{C}_{m_{i}, m_{j}}^{E}\left(q_{i}, \mathcal{M}_{1: N}\left(m_{j}\right)\right)}_{\text {old node and edge costs for region set } q_{i}} \\
& \underbrace{-\hat{C}_{m_{i}}^{N}\left(q_{i} \cup\left\{r_{k}\right\}\right)-\sum_{\left(m_{i}, m_{j}\right) \in E_{M}} \hat{C}_{m_{i}, m_{j}}^{E}\left(q_{i} \cup\left\{r_{k}\right\}, \mathcal{M}_{1: N}\left(m_{j}\right)\right)}_{\text {new node and edges costs for regions set } q_{i} \text { plus region } r_{k}}
\end{align*}
$$

where the first terms are the generalized node and edge costs as defined in Section 9.5.6. The last term $\hat{C}^{B}$ decreases the cost for region $r_{k}$ if it shares a common boundary with the regions in $q_{i}$ (see Fig. 9.11). The weighting factor $\epsilon$ was set to 0.5 in our experiments.

To calculate $\hat{C}^{B}\left(q_{i}, r_{k}\right)$, we consider each pixel on the boundary of $r_{k}$ and search for the region $r^{\prime}$ that is nearest to the pixel among all candidate regions for all model nodes. $\hat{C}^{B}\left(q_{i}, r_{k}\right)$ is set to the fraction of pixels for which the nearest region $r^{\prime} \in q_{i}$. Clearly, if $r_{k}$ is completely surrounded by regions in $q_{i}$, then $\hat{C}^{B}\left(q_{i}, r_{k}\right)=1$.

Finally, we can summarize the effects of the $1: N$ matching step with a simple rule: a region will be added to the set of assigned regions of a model node if

- the region is mostly surrounded by other regions mapped to the same model node,
- the region bridges the space between two regions that should be neighboring, or
- the combined region size (or color) matches better to the model node size.

Figure 9.12 portrays an example showing that $1: N$ matching improves the accuracy of the object-model detection. Since model regions have been split


Figure 9.11: (a) In 1: $N$ matching, the hypothetical cost after adding input region $r_{k}$ is computed. If the cost difference is lower, $r_{k}$ is attributed to the model node. (b) Definition of the common boundary of a region, i.e., the part of a boundary that is inside the region set.
into several parts by the occlusion of a foreground object, $1: N$ matching is required to cover the whole model region.

### 9.7 Results

An example matching result for a scene with several objects having similar characteristics is shown in Figure 9.13. The object defined by the model is detected correctly. Small errors occur at the left hand of the object since the algorithm cannot decide whether the fingers are part of the hand or not. As the size of the hand without fingers matches better to the size of the jacket, the fingers are discarded. Figure 9.14 shows more detection results for the same object model.

Another example result is depicted in Figures 9.15 and 9.16. For these examples, a model of the Goofy cartoon character was generated. Note that the 1: 1 matching result that is presented in Figure 9.16(a) does not cover the complete object, since some regions have remained as several independent regions in the color segmentation. In the $1: N$ matching step, most of these regions are added to the object.

### 9.8 Conclusions

This chapter has presented a new algorithm for the detection of video objects that are described by manually-defined object models. A graph-based modeling is used to describe region features and spatial relationships between regions. Thereby, the model allows for enough flexibility to find objects, even when they are deformed by articulated motion. The central matching step is carried out using a fast dynamic-programming algorithm, which is enabled by restricting the object-graphs to be trees. The computation time is currently about 1 second for a $720 \times 576$ video frame on a 550 MHz Pentium-III processor. Most of the time is used for the color segmentation step. Hence, it is possible to test the same input image for several object models even faster, because the initial color segmentation only has to be computed once.

As an alternative to the dynamic-programming based object-detection algorithm, we have also implemented a direct $1: N$-matching using a genetic programming approach [10]. However, the description of this approach was omitted here, since the results were inferior compared with the presented algorithm. Even though the cost function was defined in the same way, the genetic algorithm did not show robust convergence.

Our experiments for various sequences revealed that the described matching algorithm is robust if the visible object does not deviate too much from the model and if there is no significant occlusion. Possible errors are mostly introduced due to an erroneous color segmentation. Sometimes, regions having the same color are combined into the same region when they are close to another. Since the matching algorithm can map several input regions to a single model region, but not several model regions to a single input region, the matching algorithm searches for another region instead of assigning the undersegmented input region twice. Another major source of errors are pictures in which parts of the object are occluded, since the algorithm cannot find matching regions for the occluded parts. The consequence is that either wrong regions from the background are added to the object, or that the object is not detected at all, since the missing regions increase the total matching error too much.

Our overall impression is that the proposed approach is applicable for objects that only move within a plane. When the 3-D view onto the object is changed, many parts of the object become occluded or change their size. This leads to high matching errors and the object regions are not assigned correctly. To resolve this problem, it may be required to use a 3-D model of the object, such that the computer can reflect the changes from the differing viewing direction with the model. However, this would also require that the object pose is estimated, which complicates the matching process.

(a) 1:1-matching.

(b) 1:N-matching.

Figure 9.12: Matching the object model from Fig. 9.4. The image shown is part of a larger input image with several other objects. (a) Result after matching the object skeleton tree. Matched regions are marked. Note that the jacket is not covered completely since it has been oversegmented into several regions. (b) Matching results after the 1:N-matching step. After the 1:N matching, the jacket is completely covered.


Figure 9.13: The object model shown in Figure 9.4 is searched for in an input image with several similar objects.

(a)

(b)

Figure 9.14: Detection results for two images with the object model from Figure 9.4.


Figure 9.15: (a) Model of the Goofy cartoon character. (b) Detection result of $a 1: N$ matching with several similar objects.

(a) 1:1-matching.

(b) 1:N-matching.

Figure 9.16: Detection result of Goofy for 1 : 1 matching (a) and $1: N$ matching (b). It can be observed that the 1:1 does not include all regions. The $1: N$ matching is almost complete.

The function of the expert is not to be more right than other people, but to be wrong for more sophisticated reasons.
(David Butler)

## Object Detection based on Graph Models II: Natural

This chapter presents an algorithm for video-object segmentation that combines motion information, a high-level object-model detection, and spatial segmentation into a single framework. This joint approach overcomes the disadvantages of these algorithms when they are applied independently. These disadvantages include the low semantic accuracy of spatial color segmentation, the inexact object boundaries obtained from object-model matching and the often incomplete motion information. The described algorithm alleviates three of the problems that we encountered in the segmentation system that was described in the first part of the thesis. First, it completes object areas that cannot be clearly distinguished from the background because their color is similar to the background color. Second, parts of the object that were not extracted because they are not moving, are now added to the object mask. Finally, when several objects are moving, of which only one is of interest, it is detected that the remaining regions do not fit to any object model and these regions are removed from the foreground. This suppresses regions that are considered erroneously as moving, or objects that are moving but irrelevant to the user.

### 10.1 Introduction

Segmentation-techniques can be coarsely classified into spatial and temporal segmentation. Spatial segmentation includes segmentation based on texture or simply color. Color segmentation provides accurate region boundaries, but since it is working at the signal level, a semantically meaningful object separation cannot be found without further information. On the other hand, semantic objects can be identified rather accurately by observing areas that are moving consistently. Hence, temporal segmentation techniques can give superior results for separating different objects. The background-subtraction technique that we used as the core of the segmentation system described in the first part of the thesis, is also a temporal segmentation approach, since it detects only the moving areas. Unfortunately, temporal segmentation does not always include the complete objects. We have observed for example, that head-and-shoulder sequences like they are common in video conferencing or news report scenes, do not yield good results because large parts of the objects are not moving.

One approach to solve this problem is to combine spatial segmentation and motion information into a joint segmentation framework. Several algorithms following this approach have been proposed in the literature. Fablet et al. [45] propose to apply a neighbourhood graph over spatially segmented image-regions. A Markovian framework is used to label the region into foreground and background according to their motion. However, this approach only differentiates between regions moving compatible with dominant motion, and regions that do not. Hence, when only part of the object is moving, the segmentation algorithm will not cover the complete object. A comparable algorithm that labels the regions into several consistent motion classes has been proposed by Patras et al. [141]. The approach presented by Alatan et al. in [6] obtains motion information in the form of change-detection masks [124]. Fusion of motion cues and a spatial colorsegmentation is carried out using a system of hard-coded rules.

A weak point of the previous algorithms is the fact that motion and spatial information are still fused in a heuristic way, without integrating real semantic knowledge about the object to be extracted. However, difficult segmentation problems exist which cannot possibly be solved without this high-level knowledge. Consider for example a head-and-shoulder sequence, where only the head is moving and the body is static. Even though every human would probably consider head and body to be a single object, the computer has no indication why the body should be assigned to the same object as the head. Consequently, the body will be treated as background.

To solve this problem, we propose a new algorithm for the fusion of temporal and spatial segmentation that applies an additional model-based

| Technique | Advantage | Disadvantage | Example |
| :--- | :--- | :--- | :--- |
| Spatial <br> segmentation | Accurate <br> region bound- <br> aries | Weak semantic <br> meaning | Does not work <br> with static <br> objects |
| Temporal <br> segmentation | Good object <br> detection | ner |  |

Table 10.1: Advantages and disadvantages of various segmentation techniques.
approach to combine all three segmentation approaches (Table 10.1). The object model enables the algorithm to find the complete object even when parts of it do not move, and using the model, it can lock to a specific object even when several objects are moving at the same time. We use an object model based on the graph representation that was introduced in the previous chapter. In this chapter, we adapt this model and the matching algorithm to obtain a segmentation framework for natural images.

### 10.2 Segmentation system architecture

Our segmentation system comprises three main steps, corresponding to the three approaches spatial, temporal, and model-based segmentation. An overview flow-graph of the algorithm is depicted in Figure 10.1.

In the first step, a change-detection algorithm is employed to find the moving regions in the image, giving an approximate location of the object. The change detection can be computed as long-term change detection, relative to a globally reconstructed background image (see Chapter 7). In this case, Step 1 is basically the complete segmentation system as described in Part I. Alternatively, the change detection can also be computed as short-
term change detection between pairs of successive frames. The two most common errors observed in the change detection are incomplete objects, because of insufficient motion, and detection of uninteresting objects (like moving trees in the background). Both classes of errors, missing object parts and uninteresting objects, cannot be removed without further highlevel information about the object's appearance.

In the second step, we use an object model to search for the location of the object in the input image. The object model is represented by a graph similar to that described in the previous chapter. The nodes in the graph represent homogeneously colored regions and graph-edges indicate spatial proximity. To generate the object model, we reuse an extended version of the graphical editor described in the previous chapter. The edited object model is a general description that can be used throughout the sequence or even for multiple sequences when the same object appears again, thereby keeping the need for user intervention low. A matching algorithm searches for the most likely position of the model in an input frame. The matching uses information about the region color and shape from the model, to search for a compatible location in the current input image. Differing from the matching process described in the previous chapter, we now also integrate additional information from the change-detection mask to help the matching lock to the moving object. Since the motion information provides us with a good first guess about the object position and object shape, the model can be fitted even when the scene content is difficult. The output of this step is an indicator of object location which does not provide exact object boundaries, but that covers the whole object including parts that are non-moving.

In a third step, spatial color-segmentation is employed to obtain exact object boundaries. Here, a region-merging algorithm is used (see Appendix E), starting with the regions obtained from a watershed [189] presegmentation. Usually, region-merging is based solely on color information. The difficulty with this is that real objects are mostly not uniformly colored. Color variations inside the object can even be larger than color differences between the object and its background. This makes it impossible to find a single threshold at which color segmentation should stop (Fig. 10.8(a)). Hence, it is important to indicate the object location to the spatial segmentation. In our algorithm, this object-location indicator is taken from the object-model matching step and it is integrated into the region-merging criterion. The criterion favors segmentation of regions inside of the object and prohibits merging of regions crossing the object/background boundary. Finally, regions which are covered by the object model are combined to give the final object segmentation-mask.


Figure 10.1: Overview of the segmentation system incorporating object models. First an abstract object model is built manually in a graphical editor (right side). This model is used to help the automatic segmentation to identify the correct object location. The automatic segmentation starts with a change detection to obtain some indication of the object position. Subsequently, the object location is found by fitting the constructed model onto the image using color information and the changedetection mask. Finally, a color segmentation is applied to compute the accurate object boundary.

### 10.3 Step 1: motion detection

The purpose of the motion detection is to distinguish static regions in the image from moving areas. This can be done either in a short-term perspective by considering two successive frames, or in a long-term perspective by reconstructing a background which only contains the static parts of the sequence and computing differences to this background.

Short-term change-detection masks (CDMs) are obtained easily by computing difference-frames between successive frames. If applicable, cameramotion compensation can be applied prior to computing the difference image. This simple approach shows the problem that occluded areas and uncovered areas cannot be distinguished in the mask. Moreover, changedetection masks of neighbouring frames only include the borders around moving object regions.

The alternative approach is to detect long-term changes relative to a static background image. This approach has been extensively studied in the first part of the thesis. The advantage of this approach is that backgroundsubtraction masks are usually more accurate than the masks obtained from short-term change detection. However, for some classes of scenes, the sequences are too short or do not provide sufficient information for the reconstruction of the complete background image. Our algorithm can operate with both kinds of motion-area detection. However, we prefer to use longterm motion detection, since it provides better estimates of moving regions.

### 10.4 Step 2: model matching

Our object model describes the appearance of articulated objects independent from a special realization in an image. The model defines the geometric structure of the object, specifying the colors of the main regions and the neighbourhood relationships. In our approach, we represent object models with graphs $G_{M}=\left(V_{M}, E_{M}\right)$. Each graph node $v \in V_{M}$ represents one object region with uniform color, while neighbouring regions are connected with a graph edge $e \in E_{M}$. In this respect, the object models are similar to the models used in the previous chapter. However, we use a slightly modified set of features for the graph nodes. Additionally to an attribute specifying the region's mean color, we approximate the shape of a region with an ellipse ${ }^{1}$. Figure 10.2(c) shows an example object model. The use of ellipses was motivated by tracking algorithms that use a mixture of multi-variate Gaussians as object observation model [196]. Ellipses allow

[^17]

Figure 10.2: Manual object model creation in the model editor.
to represent compactly the key feature of a compact region such as size, shape aspect-ratio, and orientation.

### 10.4.1 Model editor

The graphical editor for defining object models is similar to the editor described in the previous chapter. The user places markers in the essential object regions (Fig. 10.2(a)). These markers are used in a watershed algorithm to extract the region boundaries. After each user modification, the segmentation boundaries are recomputed, which makes it very easy for the user to control the segmentation process and correct errors. Additionally to the object region, the user also defines the graph edges between connected regions.

Differing from the editor described in the previous chapter, a different set of features is extracted. First, an ellipse is fitted to the shape of each object region. This abstraction is a good approximation to most region shapes and it still allows easy processing. Each ellipse is further attributed with its mean color of the corresponding region. We denote the color assigned to ellipse $e^{(i)}$ as $e_{r}^{(i)}, e_{g}^{(i)}, e_{b}^{(i)}$ when referring to its color components in RGB space.

## Ellipse fitting to region-shape

To obtain the model parameters, the region border resulting from the manual segmentation process, has to be approximated by an ellipse. Our implementation uses two different representations for ellipses in different parts of the algorithm, since each may be more suitable in a specific context. One
representation is the explicit form, where an ellipse is specified by its center $\vec{c}$ and its two principal axes $\vec{a}_{1}, \vec{a}_{2}$. However, ellipse fitting starts with the implicit form

$$
\left(\begin{array}{lll}
x & y & 1
\end{array}\right)\left(\begin{array}{lll}
A & B & D  \tag{10.1}\\
B & C & E \\
D & E & F
\end{array}\right)\left(\begin{array}{l}
x \\
y \\
1
\end{array}\right)=0
$$

Formulas for converting a representation to the alternative form can be found in an appendix section to this chapter (Section 10.8).

The parameter estimation to determine the ellipse parameters $A, \ldots, F$ of Eq. (10.1) proceeds in two steps. The first step uses algebraic minimization to obtain a first estimate. This is a very fast approach, but it does not always yield the expected solution, because a semantically meaningless, algebraic residual is minimized. Hence, in a second step, we further refine the solution using a gradient-descent approach to minimize Euclidean distances.

## Linear estimation

The first step applies an algebraic fitting using the implicit ellipse representation, where we apply the normalization $A+C=1$ to avoid the trivial solution (see [200]). To solve for the parameters, an equation system is constructed by enumerating all pixels ( $x_{i}, y_{i}$ ) on the region boundary and appending one equation for each pixel. This results in the overdetermined system

$$
\left(\begin{array}{ccccc}
x_{0}^{2}-y_{0}^{2} & 2 x_{0} y_{0} & 2 x_{0} & 2 y_{0} & 1  \tag{10.2}\\
x_{1}^{2}-y_{1}^{2} & 2 x_{1} y_{1} & 2 x_{1} & 2 y_{1} & 1 \\
x_{2}^{2}-y_{2}^{2} & 2 x_{2} y_{2} & 2 x_{2} & 2 y_{2} & 1 \\
\vdots & \vdots & \vdots & \vdots & \vdots
\end{array}\right)\left(\begin{array}{c}
A \\
B \\
D \\
E \\
F
\end{array}\right)=\left(\begin{array}{c}
-y_{0}^{2} \\
-y_{1}^{2} \\
-y_{2}^{2} \\
\vdots
\end{array}\right)
$$

which is solved in the least-squares sense, using a Singular Value Decomposition.

## Nonlinear estimation

Since this algebraic optimization may produce inexact results, the parameters are refined with a subsequent gradient-descent process, minimizing the geometric distance between the region boundary and the ellipse. We use a symmetric distance measure that consists of the sum of the region-to-ellipse distances $d_{r \rightarrow e}$, and the ellipse-to-region distances $d_{e \rightarrow r}$ :

$$
\begin{equation*}
d_{\text {geom }}=d_{r \rightarrow e}+d_{e \rightarrow r} . \tag{10.3}
\end{equation*}
$$


(a) Distance $d_{r \rightarrow e}$, measured from region to ellipse.

(c) Distance $d_{e \rightarrow r}$, measured from ellipse to region.

(b) Example of degenerated fit for $d_{r \rightarrow e}$.

(d) Example of degenerated fit for $d_{e \rightarrow r}$.

Figure 10.3: If only one of the measures $d_{r \rightarrow e}$ or $d_{e \rightarrow r}$ would be used, the result would include undesired fitting results (b), (d). This can be prevented by combining both distance measures.

The difference between both distances is that in the first case, the region border is sampled and the corresponding minimum distances to the ellipse are calculated. In the second case, the ellipse is sampled and nearest regionborder pixel is being searched for. The symmetric distance $d_{\text {geom }}$ results in better region-shape approximations than an approximation with only one asymmetric distance (see Figure 10.3).

For the region-to-ellipse distance, we iterate through all point on the region border and compute the distance of that point to the ellipse. Since the computation of the distance of a point to an ellipse is computationally complex, we use an approximation to the point-to-ellipse distance. We define the approximate point-to-ellipse distance as the distance measured along the ray from the ellipse center $\vec{c}$ to the point $\vec{p}_{i}$ (Fig. 10.4). To determine this distance, we first compute the vector $\vec{s}-\vec{c}$. Let $\mathbf{M}=\left(\vec{a}_{1} \vec{a}_{2}\right)$ be a matrix consisting of the axes of the ellipse and let $\vec{c}$ be the ellipse center. By transforming the ellipse back to a unit circle using $\mathbf{M}^{-1}$, we easily see that in the back-transformed coordinate system, $\vec{s}-\vec{c}$ must have unit length. Hence, by scaling $\vec{p}_{i}-\vec{c}$ with the inverse of the $\left\|\vec{p}_{i}-\vec{c}\right\|$ in the back-transformed coordinate system, the vector $\vec{s}-\vec{c}$ is obtained. Consequently, the approximate distance of a point $\vec{p}_{i}$ to the ellipse computes


Figure 10.4: The distance of a point $p_{i}$ to an ellipse is approximated by the distance along the ray connecting the point with the ellipse center c.
as

$$
\begin{equation*}
d\left(\vec{p}_{i}\right)=\left\|\left(\vec{p}_{i}-\vec{c}\right)-(\vec{s}-\vec{c})\right\|=\left\|\vec{p}_{i}-\vec{c}-\frac{\left(\vec{p}_{i}-\vec{c}\right)}{\left\|\mathbf{M}^{-1}\left(\vec{p}_{i}-\vec{c}\right)\right\|}\right\| \tag{10.4}
\end{equation*}
$$

This enables the calculation of the region-to-ellipse distance in closed form as

$$
\begin{equation*}
d_{r \rightarrow e}=\frac{1}{\mid \text { border } \mid} \sum_{\vec{p}_{i} \in \text { border }} d\left(\vec{p}_{i}\right) . \tag{10.5}
\end{equation*}
$$

The point $\vec{p}$ iterates through all pixels on the region border.
The distance $d_{e \rightarrow r}$ is computed by sampling the ellipse with a sufficient number of points and each time searching for the nearest region-boundary pixel. Hence, the ellipse-to-region distance computes as

$$
\begin{equation*}
d_{e \rightarrow r}=\frac{1}{N} \sum_{n=0}^{N-1} \min _{\vec{p} \in \text { border }}\left\|\vec{p}-\vec{a}_{1} \cos \frac{2 \pi n}{N}-\vec{a}_{2} \sin \frac{2 \pi n}{N}\right\| . \tag{10.6}
\end{equation*}
$$

An example result of this ellipse-fitting process is shown in Fig. 10.2(b).

### 10.4.2 Model detection

The purpose of model detection is to find the position of the object model in an input frame. Remember that in the previous chapter, we applied an automatic segmentation to the input frame to obtain a graph representation similar to the object model. Model detection could then be carried out as a graph-matching problem. This was well possible, since we were
concentrating on cartoon sequences for which an automatic color segmentation is easy. However, for natural sequences, we cannot rely on a good color segmentation. Hence, we search for the object model directly in the input image.

The fitting value of a specific configuration of ellipses is again defined through a combination of node and edge costs.

- Node costs evaluate the difference between the color of the model node and the mean color of an area in the image. Additionally, the node cost is reduced if the considered image area contains moving image content. This reduction on moving areas gives the algorithm a preference for locking on moving objects, which will most probably include the actual object we are searching for.
- Edge costs represent the geometric distance between areas that are connected in the object model. The larger the distance between these areas, the higher the edge cost.


## Node costs

To define the node cost, let $e^{(i)}(x, y, \alpha)$ denote the set of pixels contained in the ellipse $i$, where the ellipse is shifted so that its center is at $(x, y)$ and where the ellipse is additionally tilted by an angle $\alpha$. Consequently, the ellipse area is $\left|e^{(i)}\right|$. We define the node matching-cost for ellipse $i$ at position $(x, y)$ and angle $\alpha$ as

$$
\begin{align*}
s^{(i)}(x, y, \alpha)= & \underbrace{\left\|\frac{1}{\left|e^{(i)}\right|}\left[\sum_{\left(x^{\prime}, y^{\prime}\right) \in e^{(i)}(x, y, \alpha)}\left(\begin{array}{l}
f_{r}\left(x^{\prime}, y^{\prime}\right) \\
f_{g}\left(x^{\prime}, y^{\prime}\right) \\
f_{b}\left(x^{\prime}, y^{\prime}\right)
\end{array}\right)\right]-\left(\begin{array}{c}
e_{r}^{(i)} \\
e_{g}^{(i)} \\
e_{b}^{(i)}
\end{array}\right)\right\|}_{\text {color matching-cost }}  \tag{10.7}\\
& -\underbrace{\frac{\gamma}{\left|e^{(i)}\right|} \sum_{\left(x^{\prime} ; y^{\prime}\right) \in e^{(i)}(x, y, \alpha)} m\left(x^{\prime}, y^{\prime}\right)}_{\text {motion-area bonus }},
\end{align*}
$$

where the first term measures the color difference and the second term reduces the cost for moving areas. The parameter $\gamma$ is a fixed weighting factor. Note that the sum over all pixels in the ellipse area can be computed very efficiently using the scheme presented in the Section 10.8. An example error-map $s^{(i)}(x, y, \alpha)$ is shown in Figure 10.8(c).


Figure 10.5: Approximate computation of distance between two ellipses. See Eq. (10.8).

## Edge costs

Edge costs are defined as the distance between two ellipses $e^{(i)}, e^{(j)}$. This distance is approximated by the distance along the line connecting the two ellipse centers (Figure 10.5). The approximation can be computed in closed form by

$$
\begin{align*}
& d^{(i, j)}\left(x_{i}, y_{i}, \alpha_{i}, x_{j}, y_{j}, \alpha_{j}\right)=\operatorname{abs}\{\underbrace{\left\|\vec{c}^{(i)}-\bar{c}^{(j)}\right\|}_{\text {distance between centers }} \\
& \quad-\underbrace{\left\|\frac{\left(\vec{c}^{(i)}-\vec{c}^{(j)}\right)}{\left\|\mathbf{M}^{(\mathbf{i})^{-1}}\left(\vec{c}^{(i)}-\vec{c}^{(j)}\right)\right\|}\right\|}_{\text {center-to-border distance }}-\underbrace{\left\|\frac{\left(\vec{c}^{(i)}-\vec{c}^{(j)}\right)}{\left\|\mathbf{M}^{(\mathbf{j})^{-1}}\left(\vec{c}^{(i)}-\vec{c}^{(j)}\right)\right\|}\right\|}_{\text {center-to-border distance }}\} . \tag{10.8}
\end{align*}
$$

## Fitting process

The best matching location of the object model in an input image is determined as the set of ellipse locations $\left\{\left(x_{i}, y_{i}, \alpha_{i}\right)\right\}$ that minimizes the following expression:
$\min _{\left\{\left(x_{i}, y_{i}, \alpha_{i}\right)\right\}} \sum_{k \in V_{M}} \underbrace{s^{(k)}\left(x_{k}, y_{k}, \alpha_{k}\right)}_{\text {node cost (ellipse position) }}+\sum_{(m, l) \in E_{M}} \underbrace{d^{(m, l)}\left(x_{m}, y_{m}, \alpha_{m}, x_{l}, y_{l}, \alpha_{l}\right)}_{\text {edge cost (ellipse distances) }}$.
If $E_{M}$ has a tree structure, a dynamic-programming approach similar to the algorithm described in the previous chapter can be used for an efficient computation of the optimum. This is a generalization of the one-dimensional shortest-path problem to a minimum-cost tree problem (Figure 10.6). In


Figure 10.6: Dynamic-programming computation-graph example for a simple model graph (Fig. 9.9(b)).
order to avoid high computational complexity, not all possible positions are included in the computation graph for dynamic-programming. Instead, $n \approx 30$ candidate positions are selected according to the following process. First, the costs $s^{(k)}\left(x_{k}, y_{k}, \alpha_{k}\right)$ for placing an ellipse $k$ at position $\left(x_{k}, y_{k}\right)$ with angle $\alpha_{k}$ are calculated (Fig. 10.8(c)). All local minimum positions could be potential candidate positions for the ellipse, but to decrease the complexity of the matching process, we only select the $n$ best positions. Positions with a cost exceeding a threshold are not included, which may decrease the number of candidates further to less than $n$ positions. Figure $10.8(\mathrm{~d})$ shows all candidate positions for the ellipse corresponding to the tie in the model of the man. Using the same process, candidate locations for each model region are extracted. Finally, using the dynamic-programming algorithm, the best combination of ellipse locations is computed, resulting in the detection of the object in the input image (Figure 10.8(e)).

### 10.5 Step 3: spatial segmentation

The automatic spatial segmentation stage uses two sources of information to compute more accurate object boundaries. The first source is color information from the current input frame while the second information comes from the fitted model-location in form of the ellipse parameters. This high-level knowledge about the approximate object location helps to control the spatial segmentation, such that areas attributed as foreground will not be merged with background areas even if color differences are small.

### 10.5.1 Spatial segmentation algorithm

We apply a two-step approach for spatial segmentation. First, a watershed pre-segmentation is applied. This is a fast algorithm which usually results in heavy oversegmentation. The reason for applying this step is that it
speeds up the subsequent region-merging algorithm, which can now start with the regions obtained from the watershed pre-segmentation instead of starting at the pixel level. The region-merging algorithm gradually merges the two most similar, neighbouring regions together (see Appendix E for a description of the region-merging algorithm). We use the fast implementation of region-merging that is described in [46].

### 10.5.2 Merging criterion

The merging criterion for evaluating region dissimilarity is composed of two terms. The first is the Ward-criterion, which minimizes the variance of the luminance component in a region. More clearly, the first term in Equation (10.10) equals the increase of variance if two regions $r_{i}$ and $r_{j}$ would be merged. This can be computed efficiently by keeping track of the mean region-luminances $\mu_{i}$, and $\mu_{j}$ and the region sizes.

$$
\begin{equation*}
S\left(r_{i}, r_{j}\right)=\underbrace{\frac{\left|r_{i}\right| \cdot\left|r_{j}\right|}{\left|r_{i}\right|+\left|r_{j}\right|}\left(\mu_{i}-\mu_{j}\right)^{2}}_{\text {Ward }} \cdot \underbrace{\left(\max _{k} \frac{\left|r_{i} \cap e^{(k)}\right|}{\left|r_{i}\right|} \cdot \frac{\left|r_{j} \cap e^{(k)}\right|}{\left|r_{j}\right|}+\beta\right)^{-1}}_{\text {penalty for crossing object/background border }} \tag{10.10}
\end{equation*}
$$

The second term increases merging cost if one or both regions are not covered by one of the object-model ellipses. This inhibits merging of regions inside the object with regions outside of the object. The parameter $\beta>0$ controls the influence of the second term. Since $r_{i} \cap r_{j}=\emptyset$ for $i \neq j$, it holds that

$$
\begin{equation*}
\frac{\left|\left(r_{i} \cup r_{j}\right) \cap e^{(k)}\right|}{\left|r_{i} \cup r_{j}\right|}=\frac{\left|r_{i} \cap e^{(k)}\right|+\left|r_{j} \cap e^{(k)}\right|}{\left|r_{i}\right|+\left|r_{j}\right|} \tag{10.11}
\end{equation*}
$$

and an updated $S\left(r_{i}, r_{j}\right)$ can be computed efficiently after each merging step by keeping track of attributes $\left|r_{i} \cap e^{(k)}\right|$ for each region $r_{i}$. Region merging stops as soon as the smallest $S\left(r_{i}, r_{j}\right)$ exceeds a threshold.

The final object mask is obtained by joining all regions for which at least $50 \%$ of their area is covered by an object-model ellipse. Small holes in the masks that can result from non-overlapping model ellipses are filled up in a final post-processing step.

### 10.6 Experiments and results

This section presents the performance of the proposed object-detection system on various input sequences. The first example is taken from the paris sequence. This is a head-and-shoulder sequence without camera motion.


Figure 10.7: The merging criterion also evaluates the position of the regions relative to the covered area of the model ellipse. Merging of two regions within the covered area is favoured. The numbers beneath the regions denote the value of $\left|r_{i} \cap e^{(k)}\right| /\left|r_{i}\right|$.

The two main difficulties with this example are that the body of the man at the left does not move much, which makes it undetectable for the motionsegmentation, and that the color difference between the hair of the man and the background is small. Because of this small difference, a colorbased segmentation without any semantic knowledge usually results in a wrong segmentation (Fig. 10.8(a)), merging hair and background into the same region. There are more errors which cannot be seen so clearly: part of the arm at the left was merged with part of the chair, dark parts in his hand were merged with shadow, and so on.

We have edited a model of the man (see Fig. 10.2) and applied the algorithm to frame 20. The detected object is superimposed onto the input frame in Figure 10.8(e). The model is placed at a sensible configuration, but the object is still not completely covered by the model, since the use of ellipses does not fit exactly to the region shape. Figure 10.8(f) shows the result after a color segmentation that integrated the detected object location. Almost the complete object is covered in the segmentation mask. Small parts of the head are missing as these areas are not covered by the model. On the other hand, a small part of the background was added to the mask, because the ellipse covering the jacket region is slightly too large.

The second example is taken from the stefan sequence, which has strong camera-motion. Since the object is small compared to the background and the object is also moving, the background image can be reconstructed without error (Fig. 6.22). Nevertheless, the long-term change-detection mask (Fig. 10.9(a)) does not give a clear result. Three problems can be identified:

- not all parts of the background vanish in the difference frame, because of small movements within the audience,
- parts of the tennis player are lost, since they have coincidentally the
same color as the background, and
- the mask contains both the tennis player and the tennis ball, while sometimes only one of them may be desired in the output.

We constructed a model of the tennis player and applied the segmentation algorithm again on frame 177 of the sequence. The result shows that the tennis player is found, but it also still has inaccuracies, especially at parts where the object has some fine texture. We also supplied the algorithm with a model of the tennis ball, which is simply a single graph node. The result is shown in Fig. 10.9(d).

Another example showing a news report scene is depicted Figure 10.10. The body of the foreground object is mostly static, but there is strong motion in the scene background. Still a good segmentation result is obtained.

Figure 10.11 shows example frames from the carphone sequence. It is visible that the detected object follows the input sequence without major problems as long as only articulated motion is present. However, in frame 180, the size of the object in the input frame is increased because the man comes closer to the camera. This change of object size is not included in our model and consequently, the object cannot be covered completely.

Finally, Figure 10.12 presents results for the foreman sequence. At the end of the sequence, the camera turns and the object leaves the visible area. During this time, the algorithm has difficulties because not all parts of the object are visible. Since the algorithm has to find all parts in the input image, it places some parts at areas beneath the object. In Figure 10.12(d), it is visible that the complete model is stretched to keep the body part inside the image as much as possible, while the head moves left.

### 10.7 Conclusions

This chapter has described a segmentation system that combines motion detection, spatial segmentation, and model-based object detection into a single framework. Motion detection is used for an approximate localization of the object position. The object detection fits a manually defined object model to the current input frame in order to cover the complete area of the modeled object. Spatial segmentation is used to refine the object boundary and to generate accurate segmentation masks.

The object model uses attributed graphs to describe the main regions of the object and their spatial relationship. Because the spatial relationships are restricted to a tree structure, the matching algorithm can apply a dynamic programming approach for the efficient detection of the model. The restriction to tree-shaped graphs is no serious limitation for practice,

(a) Color only segmentation. Note the undersegmentation at the head.

(c) Node matching cost for the region corresponding to the tie. Brighter means lower cost. Candidate positions are marked.

(e) Fitted object model.

(b) Short-term change-detection mask.

(d) Candidate configurations for the tie region.

(f) Final segmentation mask.

Figure 10.8: Frame 20 of the paris sequence.


Figure 10.9: Frame 177 of the stefan sequence.


Figure 10.10: News report. Stong motion also occurs in the background.
because most natural objects have articulated limbs, but no cycles. The shape of each object region is described compactly with an ellipse, because it provides a good compromise between computational simplicity and accuracy of describing the object shape. Note that a too detailed description of object shape is not desired, since the object shape usually varies considerably when viewing the object from different directions.

The algorithm has the clear advantage over previous techniques in the sense that it does not solely rely on motion or spatial information to decide what the object should be. Instead, the user can specify the exact object that he wants to extract, without doing the segmentation manually. A further advantage is that in cases in which only part of the object is moving, or in cases in which the object is not clearly distinguishable from the background, the object segmentation can be solved through the combination of several features.

Problems with the current approach mostly become visible at areas close to the region boundary, where the region-shape cannot be approximated well using ellipses. Therefore, future research should consider other representations for region shapes. We have conducted first experiments that use deformable templates for the object regions. This approach seems promising, since it appears to provide more accurate object boundaries. However, it is more sensitive to 3 -D motion and deformations of the object. Hence, finding a good model for the objects will be an interesting topic for further research.

Another possible improvement may be the inclusion of textured regions in the object model, since our current approach is still limited to uniformlycolored object regions. Finally, in some cases it would be advantageous to include ordering constraints into the matching process to disambiguate situations where the spatial ordering is known. For example, in a human model, we know that the head will be above the body and we may want to add this knowledge to help the matching process. However, in most cases this would introduce additional graph edges which will violate the tree-structure limitation of our graph-matching algorithm. Thus, future research should also consider object models for which fitting algorithms exist that provide a good compromise between accuracy and computational complexity.


Figure 10.11: Carphone sequence. In (c), the head is not completely covered, since the size in the image is larger than the size in the model.


Figure 10.12: Foreman sequence. Around frame 280, the object leaves the visible area and the algorithm cannot localize the complete model anymore.

### 10.8 Appendix: notes on ellipse processing

## Converting from implicit to explicit form

The conversion is carried out in three steps. First, we determine the conic center position. In a second step, the implicit parameters are modified such that the ellipse is shifted to the origin. Finally, the axes are obtained using Eigenvector analysis. A conic at the origin is defined as

$$
\begin{equation*}
A^{\prime} x^{2}+B^{\prime} x y+C^{\prime} y^{2}=F^{\prime} \tag{10.12}
\end{equation*}
$$

Shifting the conic to the center $\left(c_{x}, c_{y}\right)$ results in

$$
\begin{equation*}
A^{\prime}\left(x-c_{x}\right)^{2}+B^{\prime}\left(x-c_{x}\right)\left(y-c_{y}\right)+C^{\prime}\left(y-c_{y}\right)^{2}=F^{\prime} \tag{10.13}
\end{equation*}
$$

Comparing this with the general equation for conics (10.1), we obtain the two equations

$$
\begin{align*}
D & =-\left(A c_{x}+B c_{y}\right)  \tag{10.14}\\
E & =-\left(C c_{y}+B c_{x}\right) \tag{10.15}
\end{align*}
$$

From this, we can compute the center of the conic by

$$
\begin{equation*}
c_{y}=\frac{A \cdot E-D \cdot B}{B^{2}-A \cdot C}, \quad c_{x}=-\frac{D+B \cdot c_{y}}{A} \tag{10.16}
\end{equation*}
$$

Now, the parameters $\left(A^{\prime}, B^{\prime}, C^{\prime}, D^{\prime}, E^{\prime}, F^{\prime}\right)$ for a conic shifted to the origin can be obtained from the original parameters using

$$
\begin{align*}
& A^{\prime}=A, \quad B^{\prime}=B, \quad C^{\prime}=C, \quad D^{\prime}=E^{\prime}=0, \quad \text { and } \\
& F^{\prime}=F-\left(A c_{x}^{2}+2 B c_{x} c_{y}+C c_{y}^{2}\right) \tag{10.17}
\end{align*}
$$

This gives us the zero-centered conic equation $(x y) \mathbf{Q}(x y)^{T}=1$ with

$$
\mathbf{Q}=-\frac{1}{F^{\prime}}\left(\begin{array}{ll}
A & B  \tag{10.18}\\
B & C
\end{array}\right)
$$

By determining the Eigenvectors of $\mathbf{Q}$ and scaling according to their Eigenvalues, we obtain the principal axes $\vec{a}_{1}, \vec{a}_{2}$. Hence, we have the explicit ellipse parameters $\vec{c}=\left(c_{x} c_{y}\right)^{T}, \vec{a}_{1}$, and $\vec{a}_{2}$.

## Converting from explicit to implicit form

Assume that the ellipse is given in explicit form with center $\left(c_{x}, c_{y}\right)$, tilt angle $\alpha$ and lengths of principal axes $a_{1}, a_{2}$. To get the implicit form, we start with

$$
\begin{equation*}
\left(\frac{x^{\prime}}{a_{1}}\right)^{2}+\left(\frac{y^{\prime}}{a_{2}}\right)^{2}=1 \tag{10.19}
\end{equation*}
$$

for an ellipse whose axes are aligned to the coordinate axes. After translating and rotating the coordinate system by

$$
\binom{x^{\prime}}{y^{\prime}}=\left(\begin{array}{cc}
\cos \alpha & -\sin \alpha  \tag{10.20}\\
\sin \alpha & \cos \alpha
\end{array}\right)\binom{x-c_{x}}{y-c_{y}},
$$

we finally obtain

$$
\begin{align*}
& A=a_{2}^{2} \cos ^{2} \alpha+a_{1}^{2} \sin ^{2} \alpha \\
& C=a_{2}^{2} \sin ^{2} \alpha+a_{1}^{2} \cos ^{2} \alpha \\
& B=\left(a_{1}^{2}-a_{2}^{2}\right) \cos \alpha \sin \alpha \\
& D=a_{2}^{2} \cos \alpha\left(-c_{x} \cos \alpha+c_{y} \sin \alpha\right)-a_{1}^{2} \sin \alpha\left(c_{x} \sin \alpha+c_{y} \cos \alpha\right)  \tag{10.21}\\
& E=a_{2}^{2} \sin \alpha\left(c_{x} \cos \alpha-c_{y} \sin \alpha\right)-a_{1}^{2} \cos \alpha\left(c_{x} \sin \alpha+c_{y} \cos \alpha\right) \\
& F=a_{2}^{2}\left(c_{x} \cos \alpha-c_{y} \sin \alpha\right)^{2}+a_{1}^{2}\left(c_{x} \sin \alpha+c_{y} \cos \alpha\right)^{2}-a_{1}^{2} a_{2}^{2} .
\end{align*}
$$

## Efficient computation of the sum over an elliptical area

Assume that we want to calculate the sum of $f(x, y)$ over the area inside an ellipse, given in implicit form. In a pre-computation step that has only to be done once for every $f(x, y)$ and which is independent of the ellipse, we compute

$$
\begin{equation*}
F(x, y)=\sum_{i=0}^{x} f(i, y) . \tag{10.22}
\end{equation*}
$$

Now, the sum over part of a single line can be computed in constant time as $\sum_{x=a}^{b} f(x, y)=F(b, y)-F(a-1, y)$. To sum over the ellipse area, we proceed line by line, computing the horizontal range $\left[x_{\min } ; x_{\max }\right]$ on each scanline. Using Equation (10.1), we obtain for a fixed $y$

$$
\begin{equation*}
x_{\min }, x_{\max }=-\frac{B+D}{A} \mp \sqrt{\left(\frac{B+D}{A}\right)^{2}-\frac{F+C y^{2}+2 E y}{A}} . \tag{10.23}
\end{equation*}
$$

An adequate notation should be understood by at least two people, one of whom may be the author. (Abdus Salam)

## Chapter $\rfloor$

## Manual Segmentation and Signature Tracking

Numerous applications requiring a high segmentation accuracy exist in various domains. If these applications permit to compute the segmentation masks offline, it can be advantageous to use semi-automatic segmentation techniques. In semi-automatic segmentation, the user controls the segmentation manually, but he is supported by the computer to relieve him from working at the pixel level. A popular approach to semi-automatic segmentation is the Intelligent Scissors tool, which uses shortest-path algorithms to locate the object contour between two user-supplied control points. This chapter proposes a new interactive segmentation algorithm, which is based on the same idea as Intelligent Scissors, albeit providing a user-interface without the need for special control points. Instead of placing control points, the user specifies a rough corridor along the object boundary, in which the computer searches for a shortest circular path. This provides an intuitive user interface, which also supports a natural way to iteratively improve the segmentation by changing the corridor shape. Furthermore, the algorithm is extended with a tracking component, such that an object that has been defined in one image can be automatically segmented in the successive frames. However, the algorithm always allows to interactively intervene in the tracking and provide corrections when the automatic tracking shows errors.

### 11.1 Introduction

In various applications, the accuracy and reliability of automatic segmentation algorithms is not sufficient. One of these applications is image editing, where objects from one image should be copied into another image. Usually, high-quality results are of primary concern and therefore, it is required to carry out the segmentation manually. Clearly, a manual segmentation is possible even for the most difficult input sequences, but the work is tedious. To relieve the user from working at the pixel-detail level, semi-automatic segmentation algorithms can be used, since they provide a compromise between work-flow efficiency and accuracy of the results. With these algorithms, it is sufficient for the user to coarsely mark the object, while the algorithm extracts the detailed object boundaries at the pixel level.

Various approaches for semi-automatic segmentation have been proposed. They can be coarsely separated in region-oriented algorithms and edge-based algorithms. We have already presented a simple example of a region-oriented algorithm in the graph-editor that we described in Chapters 9 and 10. A more advanced region-based algorithm that also supports textured regions, is GrabCut [158]. The principle of all region-based algorithms is that the user places some markers inside the object and in the background. Afterwards, the algorithm examines the color and texture around these markers to separate these regions.

On the other side, there are edge-based algorithms, of which the Intelligent Scissors algorithm [130] is the most prominent one. In this algorithm, the user traces along the object as if he would cut it out with a pair of scissors. However, the cutting path is automatically locked to the nearest strong edge in the image, which is most probably the object contour. From time to time, the user places control points to fix the contour found so far. The most frequent problem with this algorithm is that the contour snaps to high-contrast clutter in the background, instead of a lower-contrast object edge. If this is discovered too late, it is difficult to make corrections, since control points have to be moved or inserted.

In this chapter, we propose a new edge-based algorithm, which uses the same concept as Intelligent Scissors, but which does not require the user to place control points. Instead, he draws a coarse corridor along the object boundary, and the computer locates the pixel-accurate path around the object within this corridor. The central part of this algorithm is a newly developed shortest circular-path algorithm. This new segmentation algorithm has two advantages: no control points have to be set and the segmentation result can be easily improved if the result is not satisfactory as the corridor can be modified at any time.

Furthermore, we propose an extension to the Corridor Scissors tool for a
more efficient segmentation of video sequences. The algorithms mentioned so far are operating on independent single images, which still requires a large amount of work if an object should be extracted from a video sequence, since every frame has to be processed independently. This manual work can be reduced by tracking the user-defined object through the video sequence to relieve the user from redefining the object in every frame. Should there be a tracking error, it can still be corrected by user intervention before the tracking continues with the succeeding frames.

What makes our tracking algorithm special is that it also integrates the texture information of the object that was found in the previous segmentation. Our algorithm extracts the texture information along the border of the object and stores it as the object signature. The successive frame is then searched for a deformable contour that shows a similar signature. To find the optimum contour location, we again apply a shortest circular-path algorithm, which now incorporates texture information from the object signature to detect the same object.

In the successive section, we briefly introduce the Intelligent Scissors algorithm and describe typical problems of that approach. This leads us to our proposal of the Corridor Scissors tool. The central algorithm of the Corridor Scissors is a shortest circular-path search, which is described in Section 11.3. Finally, in Section 11.4, we extend the tool with tracking capabilities.

### 11.2 From Intelligent to Corridor Scissors

### 11.2.1 Intelligent Scissors algorithm

The Intelligent Scissors tool is an edge-based segmentation algorithm. The user first selects a start point on the object contour, after which the computer continuously computes the minimum-cost path between the start point and the current position. A cost function assigns lower cost to stronger edges in the image, such that the minimum-cost path follows strong contours in the image (Fig. 11.1). If the user is satisfied with the current segment, he places a new control point, which ends the current contour segment and at the same time serves as the new start point. By repeating this process, the user can define the complete object contour step by step.

To compute the minimum-cost path, the Intelligent Scissors algorithm considers the input image as a graph, where each image pixel corresponds to one node in the graph. Graph edges connect nodes that correspond to neighboring pixels. Weights are assigned to these edges according to the inverse gradient strength between the two corresponding pixels. Consequently, strong gradients induce small edge weights. The total path cost


Figure 11.1: In the Intelligent Scissors tool (a), the user first selectes a start point. When he moves the pointer across the image, the computer draws the minimum-cost path between the start point and the current position. The cost function (b) assigns lower costs to stronger edges in the image.
is defined as the sum of the costs of all graph-edges on the path. After the user has placed the start point for the graph search, the computer can already begin to compute a full tree of shortest paths to all pixels with the Dijkstra algorithm [31]. A single run of the Dijkstra algorithm is sufficient, since it computes inherently all shortest paths from the start position to all other nodes. The attractive aspect is that the path between the current pointer position and the start point can be determined instantaneously by just looking up the minimum-cost path in the full shortest-path tree.

## Edge costs

Every edge $e_{i}$ in the graph representation is attributed with a cost $c_{i}$. In the original work [130], this cost was composed of a weighted sum of six different cost components. These include gradient strength, continuity of gradient direction, and object/background color. However, four of them have only a small weight ( $10 \%$ each) and according to our experiments, they have no significant influence on the result. Hence, for simplicity, we only use the two most significant components of the cost function. These two cost components are the gradient strength $c_{G}$ and a Laplacian zerocrossing detector $c_{Z}$. They are defined by

$$
c_{G}=1-\frac{\|\nabla I\|}{\max \|\nabla I\|} ; \quad c_{Z}=\left\{\begin{array}{l}
0 \text { at Laplacian zero crossings },  \tag{11.1}\\
1 \text { otherwise },
\end{array}\right.
$$

where $I$ denotes the greyscale input image. The combined edge costs are defined as $c=(1-\alpha) c_{G}+\alpha c_{Z}$, where $\alpha$ is a user-defined weighting factor.

### 11.2.2 Problems of the Intelligent Scissors tool

Although the user interface to the Intelligent Scissors algorithm can be understood quickly by most users, there is the inconvenience that the user has to place control-points regularly, because with increasing distance between start and destination point, the extracted path often leaves the object boundary. This effect is especially strong if there are high-contrast edges in the background near the foreground object (Fig. 11.2). In this case, the stronger gradients in the background area lead to a low-cost path. Even though the cost to reach this background clutter may be high, the lower cost in the high-contrast background outweighs a slightly higher cost along the desired object boundary when the path length increases. In the Intelligent Scissors user-interface, this effect shows as a sudden change of the complete path or as an unstable toggling between alternative paths, and the user has to add a control point to stabilize the path again.

As a solution for the problem of background-clutter attraction, it is proposed in [120] to limit the search area to a rectangular area between the seed and destination position. The width of the rectangle is controlled by the user (Fig. 11.2(c)). While this approach may ease the segmentation in some difficult situations, it complicates the interaction process, since another degree of freedom (width of rectangle) has to be controlled by the user.

An alternative solution is to use path cooling. In this approach, the cost of the edges on the current path are gradually decreased. This means that the parts of the path that remain stable for some time obtain a lower cost, which again decreases the probability that this part of the path is modified later. If a part of the path is stable for a longer time, that part is fixed completely and the start point of the search is moved to the end of this part. This approach eliminates the need to place special control points. However, the speed of user interaction is dictated by the path cooling and the user has to conform to this. Moreover, once the user makes a segmentation error, it is difficult to undo this error, especially if the error is in the already frozen part of the path. Finally, this algorithm is also computationally expensive, since a new shortest-path tree has to be computed after each of the frequent cooling steps.

(a) The shortest path is distracted from the true object boundary because of near high-contrast background clutter.


Figure 11.2: A high-contrast edge in the background that is much stronger than the true object edge may lead to a wrong object contour. Even though the cost to reach this high-contrast edge may be high, this is outweighed by the decreased cost along the contour. The Rubberband algorithm (c) proposes to reduce this effect by limiting the area of the graph search to a rectangle between the last control point and the current position.


Figure 11.3: Segmentation with low object/background contrast.

### 11.2.3 The Corridor Scissors tool

We propose a new segmentation tool, called Corridor Scissors, that uses a minimal, yet flexible user interface to define the desired segmentation. The Corridor Scissors algorithm does not require the explicit setting of control points, and it provides a simple and intuitive approach to modify the segmentation result until it is satisfactory. Instead of placing control points on the object contour, the user coarsely traces along the object contour with a thick brush. This defines a corridor around the object, in which the true object boundary can be found (Fig. 11.3). After the circular corridor has been defined, the computer searches for a shortest circular path inside of the corridor. The corridor not only prevents that the path is attracted by distant background clutter, but it also reduces computation time since the search space is reduced to the corridor area. If the user wants to improve the segmentation, he can do so by simply changing the shape or width of the corridor. Whenever the corridor shape is modified, a new shortest circular-path search is applied to the corridor area. An example for the problem of snapping to a near high-contrast edge is shown in Figure 11.4. First, a very wide corridor also covers a high-contrast edge that is not the object edge. Interrupting that path by narrowing the corridor forces that a different path (the correct object contour) is taken.

The underlying algorithm of Corridor Scissors is based on the same graph-search approach as the Intelligent Scissors algorithm. However, instead of searching for ordinary shortest paths, an algorithm for computing the shortest circular paths has to be applied. A new algorithm for the shortest circular-path problem, which has a comparable computation time to an ordinary shortest-path search, is presented in Section 11.3.


Figure 11.4: Improving a segmentation with Corridor Scissors. If the path is distracted to a higher contrast edge (a), the corridor can be modified to make this path impossible (b).

### 11.2.4 Experiments and results with Corridor Scissors

Some segmentation results that were obtained with the Corridor Scissors algorithm are depicted in Figure 11.5. The left column shows the input images and the right column the extracted objects, respectively. Superimposed onto the input images are the corridors that were used to obtain the results on the right-hand side. Note that in the image of the squirrel (a), there is fine texture at the ground and the contour along the hairy tail is difficult to define. In the image of the rabbit (c), the color of the foreground object is close to the background color.

It can be seen that the segmentation results were obtained with almost no manual correction of the corridor. Only at the ears of the squirrel and the rabbit, as well as the tail of the rabbit, the corridor was made a bit smaller to get the correct contour.

Our experience with a large variety of input images is that the objects are generally easy to define. The only difficult case are objects with long thin structures like antennas, since these thin objects are usually covered completely by the corridor. In this case, the algorithm prefers to simply let the path cross the thin object instead of following the long contour on both sides of the object. These errors are difficult to correct because the only possibility is to trace both sides of the object contour with the corridor, without touching itself (no bridge between both sides should be built).


Figure 11.5: Segmentation results for the Corridor Scissors algorithm. The marked corridor is depicted on the left, the segmentation result is on the right. Note the changes of the original corridor to improve the segmentation result (e.g., at the ears of the squirrel).

### 11.3 Shortest circular paths

The Corridor Scissors algorithm and also the tracking algorithm which is described in Section 11.4 are built upon an algorithm to compute shortest circular paths in graphs. This section proposes a new algorithm with low computational complexity.

### 11.3.1 Definition of circular paths

In the Corridor Scissors application, it seems intuitive to understand what we mean by a circular path. However, in the following, we develop an algorithm that works on general planar graphs. Hence, a clear definition of circular paths is required. For this reason, we define circular paths formally using the following two definitions.

Definition (cut path): Let $G=(V, E)$ be a planar graph with nodes $V$ and edges $E$. Furthermore, the graph is assumed to be embedded in the plane with two of its faces denoted as $F_{I}, F_{E}$, corresponding to the inner hole of the graph, and the exterior area (see Fig. 11.6). We call any path connecting $F_{I}$ and $F_{E}$ a cut path $\bar{p}$.

Definition (circular path): Let $G$ again be a planar graph with two faces labeled $F_{I}, F_{E}$. We define a circular path on the graph $G$ as a cyclic path $\dot{p}=v_{i} \rightsquigarrow v_{i}$, such that $\dot{p}$ and any cut path $\bar{p}$ have at least one node in common.

Two examples of graphs are depicted in Fig. 11.6. The first is a regular grid graph similar to those types of graphs that occur in the Corridor Scissors algorithm. These graphs have two large faces which are the inside area (the object to be segmented) and the exterior area (the background). The second example is a general planar graph, in which the inside and exterior faces have been specified in order to be able to define circular paths. These general graphs usually do not occur in our application, but we provide the example because our algorithm also works on these general planar graphs.

Instead of computing a shortest circular path on the original graph, it is convenient to transform it first into a corridor graph. The construction of the corridor graph can be imagined as cutting the original ring-shaped graph apart along a cut path apart to get a lane-shaped graph. Formally, this construction can be described as follows.

Construction (corridor graph): Let $G=(V, E)$ be a planar graph and $\bar{p}=v_{1} v_{2} \cdots v_{N}$ a cut path between $F_{I}$ and $F_{E}$, consisting of the nodes

(a) Circular-grid graph.

(b) General planar graph.

Figure 11.6: Two graphs with examples of valid circular paths (dark nodes). In each, an example cut path (path between inside and outside face) is marked. A circular path is defined as a cyclic path that crosses any arbitrary cut path.

(a) The circular input graph is cut into a lane-shaped corridor graph.

(b) The nodes on the cut and the left and right neighbors.

(c) The nodes on the cut are duplicated and the graph is cut apart.

Figure 11.7: The input graph is cut apart along a cut path to get a corridor graph.
$\bar{V}=\left\{v_{1}, \ldots, v_{N}\right\}$. Furthermore, we assume the following property of $G$ : if $V_{n}$ is the set of nodes adjacent to nodes in $\bar{V}$, then $V_{n}$ comprises exactly two connected components $V_{l}, V_{r}$. If $G$ has this property, we can transform $G$ into a corridor graph $G^{\prime}=\left(V^{\prime}, E^{\prime}\right)$ as follows. We set $V^{\prime}=V \cup \bar{V}^{\prime}$, where $\bar{V}^{\prime}=\left\{v_{1}^{\prime}, \ldots, v_{N}^{\prime}\right\}$ is a copy of $\bar{V}$. The edges are defined as

$$
\left.E^{\prime}=(E \underbrace{\backslash\left(V_{r} \times \bar{V}\right)}_{\begin{array}{c}
\text { cut graph be- }  \tag{11.2}\\
\text { tween } V_{r} \text { and } \bar{V}
\end{array}}) \cup \underbrace{\left\{\left\{v_{i} \in V_{r}, v_{k}^{\prime} \in \bar{V}^{\prime}\right\}\right.}_{\text {reconnect } V_{r} \text { with } \bar{V}^{\prime}} \right\rvert\,\left\{v_{i}, v_{k}\right\} \in E\} .
$$

This copies the nodes $\bar{V}$ on the cut path to $\bar{V}^{\prime}$ and reconnects the adjacent nodes $V_{l}$ and $V_{r}$ such that $V_{l}$ connects to only $\bar{V}$ and $V_{r}$ only to $\bar{V}^{\prime}$. The construction is visualized in Fig. 11.7.

Most currently known algorithms for shortest circular paths first transform the input graph to a corridor graph. On the corridor graph, searching for a circular path can be described easily as searching for a path from the "left side" $\bar{V}$ to the right side $\bar{V}^{\prime}$ with the additional constraint that a path starting at $v_{i}$ must end in $v_{i}^{\prime}$. Even though a general planar graph can be transformed into a corridor graph by cutting along any arbitrary cut path, care must be taken because shortest circular-path algorithms that operate on the corridor graph can only find paths that do not cross the cut path more than once. One way to minimize this risk is to choose a cut path which is as short as possible. In Section 11.3.2, we will present a safe technique to find a cut path that is guaranteed to be crossed only once.

### 11.3.2 Computation of shortest circular paths

## Previous work

In [178], several algorithms for the circular-path search are proposed. The Multiple Search Algorithm (MSA) uses $|\bar{V}|$ independent runs of the Dijkstra algorithm, where $|\bar{V}|$ is the length of the cut path. Each run uses fixed, opposing seed and destination nodes at both ends of the graph. After these runs, the result that gives the minimum cost is selected. This algorithm always finds the optimal solution, but it is computationally intensive because the Dijkstra algorithm has to be executed $|\bar{V}|$ times independently over the full graph.

The Image Patching Algorithm [178] only gives an approximate solution, but requires less computation time. In this case, the corridor graph is enlarged by appending part of the graph from each end to the opposite side. An ordinary shortest path is then computed through the complete, patched graph. The part of the shortest path that lies inside of the original graph is extracted and assumed to be the optimal circular path. However, even though this heuristic works in many cases, it is not assured that the optimal circular path is found. Moreover, the algorithm can even lead to non-cyclic paths. The quality of the result can be increased by enlarging the patched areas, but the required patch size is not known and the computation time increases. It is also easily possible to construct examples in which the found path remains non-circular for arbitrarily long patches.

Finally, a branch-and-bound algorithm has been proposed in [7] that, on the average, requires only $\log _{2}|\bar{V}|$ runs of the Dijkstra algorithm on the input graph. But the worst-case still requires $|\bar{V}|$ runs of the Dijkstra algorithm over the full graph.

For planar graphs, a different approach is to view the shortest circularpath problem as a maximum-flow problem. By adding two dummy nodes


Figure 11.8: Example tree of shortest paths as obtained with the Dijkstra algorithm. Consequently, there is a node (marked in the image) at which all paths to the nodes on the right side are rooted.


Figure 11.9: Two minimum-cost paths with at least two common nodes share the whole subpath between the common nodes.
inside each of the two faces $F_{I}$ and $F_{E}$, searching for the shortest circular path is equivalent to a maximum-flow problem between the two dummy nodes. However, maximum-flow algorithms have a high computational complexity of $O\left(|V|^{3}\right)$ for standard preflow-push algorithms, or $O\left(|V| \cdot|E| \log \left(|V|^{2} /|E|\right)\right)=O\left(|V|^{2} \log |V|\right)$ for the fastest algorithm currently known [77]. It should also be considered that this fast algorithm requires a complex implementation.

## Preliminary considerations

Our algorithm for computing the shortest circular paths is built upon the Dijkstra algorithm for ordinary shortest paths. In order to show the correctness of our algorithm, we will subsequently exploit the following special property of the Dijkstra algorithm.
Property: The Dijkstra algorithm always builds a complete shortest path tree, rooted at the seed node (Fig. 11.8). Hence, a single run of the Dijkstra algorithm does not give only the shortest path to the specified destination, but also the shortest path to every other node.
Additionally, we need the following theorem about shortest paths that is easy to derive.
Theorem: Let $G=(V, E)$ with $V=\left\{v_{i}\right\}_{i}$ be a (not necessarily planar) graph and let $u=v_{i_{1}} v_{i_{2}} \ldots v_{i_{n}}$ and $w=v_{k_{1}} v_{k_{2}} \ldots v_{k_{m}}$ be two minimum-cost paths (see Fig. 11.9). Then we can state that if $u$ and $w$ have two nodes $v_{p}$
and $v_{q}$ in common, there is a path $w^{\prime}=v_{k_{1}} \ldots v_{k_{m}}$ with the same cost as $w$ and which has a common subpath $s_{u}=v_{p} \rightsquigarrow v_{q}$ with path $u$.
Proof: If $s_{u}$ and $s_{w}$ have equal cost, there is nothing to show. Hence, assume that the two subpaths $s_{u}, s_{w}$ between $v_{p}$ and $v_{q}$ have different cost. Then, the cost of either $s_{u}$ or $s_{w}$ must be lower than the other. Let us assume that the cost of $s_{u}$ is lower than the cost of $s_{w}$. Consequently, $w$ cannot have minimum cost, because the cost can be lowered by replacing the subpath $s_{w}$ with $s_{u}$.

As a direct consequence of this theorem, we can state the following two corollaries, which will be used in the development of the shortest circularpath algorithm.
Corollary 1: Minimum-cost paths may cross at most once. ${ }^{1}$
Corollary 2: If two paths share a common seed, then both paths will share a common subpath to their destination until both paths split. After the split, the paths will not cross.

In other words, paths with disjoint endpoints may cross once, while paths with one endpoint in common will never cross.

## New shortest circular-path algorithm

We now define a fast algorithm for computing shortest circular paths by exploiting the properties of the Dijkstra algorithm and the above-mentioned theorem. The complete circular-path algorithm is divided into two parts. In almost all practical cases, the first part of the algorithm can already compute the optimal circular path and the second part of the algorithm can be omitted. In the practically very rare case that the first part cannot compute the optimum path, this is detected by the algorithm and the second part of the algorithm is used instead. This second, alternative algorithm is more computationally complex, but provides an optimal solution in all cases.

The algorithm is based on the observation that the corridor graphs in our application are usually much longer than they are wide. Furthermore, the data within the graph usually shows relatively clear low costs on a path along the corridor, since the corridor is placed on the object border. Consequently, the shortest circular-path problem is not difficult in the sense that the approximate path is clear. Usually, at some distance from the cut, there is only one "main route" to follow, only close to the cut itself, it is difficult to predict the path position (see Fig. 11.26). Hence, our algorithm

[^18]

Figure 11.10: Corridor cuts should be placed at narrow positions, perpendicular to the corridor. Note that the shortest path cannot cross the cut more than once. We determine a cut path by computing a shortest path between the two faces $F_{I}, F_{E}$. This ensures that it is crossed only once.
tries to identify a common subpath along the corridor that all circular shortest-paths share. Once this subpath is known, only the ends of this subpath have to be connected to form a circular path.

## Algorithm part I

The first part of the algorithm (in the following called AP1) assumes that a common subpath can be found in the corridor. If this is not the case, this will be detected and part 2 of the algorithm (AP2) will be used. AP1 comprises the following four computation steps.

AP1-Step 1: Transform the input graph into a corridor graph. In determining the cut path, it should be noted that the cut path can only be crossed once by the shortest circular path. Hence, a cut in an acute angle along the corridor should be prevented (see Fig. 11.10). One simple heuristic solution to this problem is to place the cut such that the cut path is as short as possible (minimum number of nodes in $|\bar{V}|$ ).
However, it is possible to determine a cut path that will not be crossed by the shortest circular path more than once as follows. Compute a minimumcost path $\bar{p}$ from the nodes of the face $F_{I}$ to the nodes of the face $F_{E}$. By definition, this is a cut path. If the shortest circular path crosses the cut path $\bar{p}$ more than once, then this is a contradiction to the above theorem, since both paths have minimum cost. Consequently, the computed cut path is only crossed once. Note that it is sufficient to start the computation of the minimum-cost path at only one arbitrary node of $F_{I}$ instead of considering all nodes of $F_{I}$ as start node. This significantly reduces the computation time required to find a cut path.


Figure 11.11: Illustration of Steps 2-4 of AP1. (a) A shortest path tree is computed from position $v_{1}$. This gives all shortest paths to all nodes in $\bar{V}^{\prime}$. All these paths share a common subpath up to a node $v_{r}$. (b) A shortest-path tree is computed from position $v_{N}$. Similarly as in (a), all shortest paths to $\bar{V}^{\prime}$ share a common subpath. (c) Both trees computed in Step 2 and 3 join at a common node $v_{l}$. Because the shortest circular path cannot pass the area above $v_{1} \rightsquigarrow v_{1}^{\prime}$ and below $v_{N} \rightsquigarrow v_{N}^{\prime}$, it must contain the subpath $v_{l} \rightsquigarrow v_{r}$. (d) Since a subpath of the shortest circular path is known, only the connection from $v_{r}$ to $v_{l}$ through the grey area has to be computed.

AP1-Step 2: Perform a Dijkstra-search beginning at node $v_{1} \in \bar{V}$, which is the top-left node of the corridor. This pass will compute at the same time shortest paths to nodes $v_{1}^{\prime}$ and $v_{N}^{\prime}$ (see Fig. 11.11(a)). Since the starting point is shared, both paths will share a subpath up to a node $v_{r}$. Shortest paths from the left side of the corridor to the right side cannot traverse the area above $v_{1} \rightsquigarrow v_{1}^{\prime}$, since this would mean that they have to cross $v_{1} \rightsquigarrow v_{1}^{\prime}$ twice, which is not allowed because of the above theorem. Hence, all nodes above the path $v_{1} \rightsquigarrow v_{1}^{\prime}$ can be ignored in the following steps.

AP1-Step 3: Perform a second Dijkstra-search from node $v_{N}$ to node $v_{N}^{\prime}$. In almost all practical cases of our application, especially if the corridor is long compared to the corridor width, this path will join the shortest paths $v_{1} \rightsquigarrow v_{1}^{\prime}$ from the last step at some node $v_{l}$, where $v_{l}$ is closer to the start as $v_{r}$ (see Fig. 11.11(b)). If this is the case, then we are sure that all shortest paths between the left side and the right side share at least the subpath $v_{l} \rightsquigarrow v_{r}$. In the other case that both paths do not join, we cannot use the


Figure 11.12: Situation at beginning of AP2. The disjoint shortest paths (a) and (b) are known.
simple AP1 and have to switch to the more general AP2 that is described below.

AP1-Step 4: Since it is already known that the subpath $v_{l} \rightsquigarrow v_{r}$ is part of the shortest circular path, we only have to search for a connection from $v_{r}$ back to $v_{l}$ to close the cycle. We also know that this connection must lie between the area of the previously computed shortest paths. Hence, we perform a third Dijkstra-search from $v_{r}$ to $v_{l}$ over the nodes in the shaded area depicted in Fig. 11.11(d). Appending this path to the path $v_{l} \rightsquigarrow v_{r}$ gives the shortest circular path $v_{l} \rightsquigarrow v_{r} \rightsquigarrow v_{l}$.
Note that the search for $v_{r} \rightsquigarrow v_{l}$ can be implemented more efficiently by searching backwards from $v_{l} \rightsquigarrow v_{r}$. In this case, the search can be restricted to the left area, since the shortest-path tree for the right area, rooted at $v_{r}$, is already known from the previous search of Step 3.

## Algorithm part II

The second part of the algorithm (AP2) is a more complex, generalized version of AP1. It is used if AP1 detects that there is no common subpath for all paths along the corridor. In this case, we have the situation of Fig. 11.12. The shortest path $v_{1} \rightsquigarrow v_{1}^{\prime}$ (a) and the shortest path $v_{N} \rightsquigarrow v_{N}^{\prime}$ (b) is known from the computations of AP1. Both paths are disjoint, because otherwise AP1 would have found the solution.

AP2 uses a recursive approach to continuously split the graph into smaller graphs until the shortest circular-path problem can be solved similarly as in AP1.

The input of each recursion is an input graph which is bounded to the top and bottom by shortest paths (a) and (b). Let us denote the top bounding path (a) as $v_{a} \rightsquigarrow v_{a}^{\prime}$ and the bottom bounding path (b) as $v_{b} \rightsquigarrow v_{b}^{\prime}$. The algorithm is initialized with $v_{a}=v_{1}$ and $v_{b}=v_{N}$, as shown in the figures. Note that both paths are circular paths. We now compute


Figure 11.13: $A$ shortest-path tree is computed, starting at $v_{c}$, which is the node in the middle of the left side. The bottom-most node $v_{i}^{\prime}$ on the right side, for which $v_{c} \rightsquigarrow v_{i}^{\prime}$ joins the path (a) is denoted as $v_{A}^{\prime}$ and the corresponding path as (d). Similarly, we obtain $v_{B}^{\prime}$ as the top-most node on the right side, for which $v_{c} \rightsquigarrow v_{B}^{\prime}$ (e) still joins the path (b). Furthermore, we have computed the circular path $v_{c} \rightsquigarrow v_{c}^{\prime}$ for the center node $v_{c}$.


Figure 11.14: Given (a) and (d), we know that all shortest circular paths in range $\mathbf{A}$ include the subpath $v_{l} \rightsquigarrow v_{r}$. Hence, we search for a shortest path starting from $v_{l}$ in the indicated direction to $v_{r}$. This is the shortest circular path within range A. Note that this search can be restricted to the area between (a) and (d). The path (f) is the shortest circular path $v_{A} \rightsquigarrow v_{A}^{\prime}$, which will be used as upper-bound path to further reduce the graph size in the recursion step.
a shortest-path tree, starting from the middle node $v_{c}=v_{(a+b) / 2}$ at the left side of the graph (Fig. 11.13). In one run of the Dijkstra algorithm, we obtain all shortest paths to the nodes between $v_{a}^{\prime}$ and $v_{b}^{\prime}$ at the right side. Note that we can limit the computation to the area between the two bounding paths (a),(b).

If we consider the shortest paths between $v_{c}$ and the right side of the graph, we see that the path $v_{c} \rightsquigarrow v_{a}^{\prime}$ obviously joins with (a). As we consider destination nodes $v_{k}^{\prime}$ further down $(k>a)$, there is generally a point from


Figure 11.15: Regions A and $\mathbf{B}$ are already processed, only shortest circular paths in the range in between are unknown. This range is processed recursively, first processing the graph between (f) and (c), and then the graph between (c) and (g).
which onwards the shortest path does not join (a) anymore. In fact, there is a node $v_{A}^{\prime}, A<c$ which is the last node (from top to bottom) for which the shortest path $v_{c} \rightsquigarrow v_{A}^{\prime}$ touches (a). Let us denote this path as (d), like it is depicted in Fig. 11.13. Note that This is a similar situation as in AP1. Any circular path $v_{i} \rightsquigarrow v_{i}^{\prime}$ with $a \leq i \leq A$ is known to include the common subpath of (a) and (d). This range is identified as $\mathbf{A}=[a ; A]$. We can now compute the shortest circular path that starts and ends within range $\mathbf{A}$ in a single step. To close the shortest circular path within the range $\mathbf{A}$, we compute the connection between $v_{l}$ and $v_{r}$ similarly to AP1-Step 4 (see Fig. 11.14).

A symmetrical process can be carried out to find the shortest circular path for the area $\mathbf{B}$ at the bottom of the considered graph. When the shortest circular path in each of the ranges $\mathbf{A}$ and $\mathbf{B}$ are known, we have to further consider only the remaining range in between (Fig. 11.15). Since we also know the path $v_{c} \rightsquigarrow v_{c}^{\prime}$, denoted as (c), we can split the problem for the remaining range recursively, by first considering the graph between (a) and (f) with start nodes $v_{A+1}, \ldots, v_{c-1}$, and similarly the graph between (c) and (g) with start nodes $v_{c+1}, \ldots, v_{B-1}$. Once all shortest circular paths for all ranges of nodes along the cut are known, we simply select the shortest of them as the global solution.

Step-by-step examples of the algorithm are provided in the appendix at the end of this chapter.

### 11.3.3 Computational complexity

Let us now examine the computational complexity of our shortest circularpath algorithm. In the case that AP1 succeeds, three ordinary shortest-path

(a) The worst-case.

(b) A nearly worst-case input.

Figure 11.16: In the worst-case, all $|\bar{V}|$ shortest circular paths starting in the cut are disjoint (a). For testing, we used also nearly worstcase inputs in which some noise is added to the worst-case pattern (b).
searches have to be conducted. Using the Dijkstra algorithm with a heap implementation, each search takes $O(|V| \log |V|)$ time, since the graph is planar. Hence, the total running time for AP1 is also just $O(|V| \log |V|)$. Note that in almost all practical cases in our application, AP1 is already sufficient.

The computation time of AP2 is more complicated to determine, because it is directly depending on the input data. In the worst-case (see Fig. 11.16(a)), every shortest circular path crossing the cut is independent and must be computed. However, because the size of the graph in the recursion is only about half the size of the input graph, computation of shortest paths becomes faster in later stages of the recursion. This gives a total number of nodes to be processed of approximately

$$
\begin{equation*}
\sum_{i=0}^{\log |\bar{V}|} 2^{i} \frac{|V| \log |V|}{2^{i}}=|V| \cdot \log |V| \cdot(\log |\bar{V}|+1) \tag{11.3}
\end{equation*}
$$

Furthermore, nodes from the ranges $\mathbf{A}$ and $\mathbf{B}$ can be excluded, so that the actual computation time is usually lower. Note that the cut is determined such that it is short, which makes $|\bar{V}|$ small. For increasing image resolution, the corridor area $|V|$ increases quadratically compared to the length $|\bar{V}|$, which gives us a worst-case computation time of $O\left(|V| \log ^{2}|V|\right)$.

In practice, the number of recursion steps that have to be carried out depends on the complexity of the data in the corridor. If there are clearly


Figure 11.17: Computation times for input graphs on different types of complicated data. For these complex examples, the computation time of MSA is $O\left(|V|^{3 / 2}\right)$, whereas it is only $O(|V|)$ for the proposed algorithm.
superiour paths within the corridor instead of equally good disjoint paths, a large number of possible paths can be excluded in each step. Since the global complexity of the content in the corridor does not increase with increasing image resolution, we can assume the number of recursions constant. Furthermore, the longer the corridor is compared to its width, the higher the probability that paths join. This leads to a total time for practical data of only $O(|V| \log |V|)$.

## Grid graphs

One special, but important case are regular graphs with directed edges, as shown in Figure 11.18. For this type of graph, the shortest-path search using the Dijkstra algorithm can be replaced by a more simple dynamic programming approach, which processes the nodes column by column. This reduces the computation time for an ordinary shortest-path search to $O(|V|)$ instead of $O(|V| \log |V|)$ for the Dijkstra algorithm on general planar graphs. In total, we obtain a practical computation time of also only $O(|V|)$ for the shortest circular-path search and a worst case of $O(|V| \log |V|)$.


Figure 11.18: A planar graph on a regular grid structure. With directed edges, shortest paths can be found in time $O(|V|)$ using a dynamic programming approach, compared to $O(|V| \log |V|)$ when applying the Dijkstra algorithm.

## Experimental verification

To justify the estimation of computation time, we conducted experiments with different types of inputs. In the first experiment, we took the picture shown in Fig. 11.27 as input graph with costs taken from the pixel luminance. The shortest circular path horizontally crossing the image was computed for different sizes of the input image. Note that this input is more complex than the usual case in the Corridor Scissors algorithm, since the input image is square and thus, the probability that paths across the image share a common subpath is lower than for the long and narrow corridors. Moreover, the image content is more complex than in the Corridor Scissors application, because in the latter case, the input covers mainly a single object contour.

In the second experiment, we used rectangular grid graphs that are 10 times longer than wide. The edge weights were set to random values. This is also more complex than the case for Corridor Scissors, since the input data shows no structure that increases the probability of common subpaths.

In a final experiment, we synthesized an input that is close to the worstcase. We generated input images with alternating high-cost and low-cost rows with an average cost difference of $\sigma$ and we added uniform random noise with an amplitude of $\sigma / 2$. Note that the actual choice of $\sigma>0$ has no influence on the shortest paths found.

In all experiments, we used edges arranged as in Fig. 11.18, such that a fast shortest-path search with dynamic programming can be applied. We measured the number of operations, which we defined as the number of nodes processed, for the proposed algorithm and the MSA algorithm.

The measured computation times for both algorithms are depicted in Figure 11.17. It is clearly visible that the MSA algorithm has a complexity of $O\left(|V|^{3 / 2}\right)$, while the proposed algorithm runs in only $O(|V|)$ time on normal inputs. This is remarkable, since the considered input is more complex than the usual input in practice. A computation time that comes close to the worst case could only be reached with the synthetically constructed input. Hence, we can conclude that even though the worst-case performance of the proposed algorithm is $O(|V| \log |V|)$, the computation time is linear in all practical cases.

### 11.4 Signature tracking

The Corridor Scissors algorithm described in Section 11.2 is a still-image segmentation algorithm. To process a video sequence, every frame has to be segmented separately. In this section, we propose an extension to our Corridor Scissors algorithm that provides object-tracking capabilities. With object tracking, it is no longer required to edit each frame independently. Instead, a contour that has been defined once can be tracked through the sequence automatically. However, in case of tracking errors, our algorithm supports the manual intervention to correct the segmentation result.

### 11.4.1 A first tracking algorithm

A popular approach to implement contour tracking is to search the local neighborhood of the previous contour for the new contour. Depending on the internal representation of the contour, different tracking algorithms have been proposed. See, for example, [11] for active contours, [138] for tracking with a level-set representation, or [197] for graph-cut based active contours. The algorithm proposed in [197] can essentially also be applied to our Corridor Scissors framework. By replacing the proposed graph-cut contour detection with our shortest circular paths, we obtain the following simple tracking algorithm.

Let us assume that the motion of the object is so small that the largest distance between contour points in both images is less than $w$ pixels. If we copy the contour found in the previous frame $t-1$ to the current frame $t$, and if we augment this contour to a corridor to a width of $2 w$ pixels, then this corridor contains the object contour in the new frame. To find the new object contour, we only have to apply the previously described Corridor Scissors algorithm in this new corridor. The resulting contour can again be used to generate a new corridor for the successive frame $t+1$, and so on. Note that the user can always intervene this process whenever a tracking


Figure 11.19: The texture around the object is stored in an object-signature vector. In the successive frame, we search for a closed contour showing a similar signature along the object border.
error occurs, by simply adapting the shape of the corridor.

### 11.4.2 Signature tracking algorithm

The above tracking algorithm is very simple to implement and easy to use, but its tracking robustness is low, since the contour in the new image is often distracted by background clutter. In this case, the contour quickly drifts away from the correct object contour and it cannot be found back, as no information about the object itself is used. However, note that the tracking algorithm does not use all the information available from the previous frame. In fact, we can also use the colors along the object contour as an additional source of information about the object's appearance. By searching for exactly the same colors along the object contour as in the original object, the object-contour detection is more robust, since it can distinguish background texture from foreground. We denote this texture information along the object contour as the object signature.

Since pixels exactly at the border between foreground and background do not provide stable information, we extract for each pixel along the contour a small block of the texture surrounding this pixel. This results in a signature vector with the length of the object contour, where each entry is a texture block from that contour position. The concept is now to use this signature for replacing the matching cost in the shortest circularpath search and to find a new object contour which has a similar signature (Fig. 11.19). The additional information from the signature helps to prevent that the circular path snaps to high-contrast background clutter, since the
image content along the contour should fit to the recorded signature. One problem in the matching is that the object contour in the new image may be larger or smaller, because of object deformations. Hence, we have to allow for some stretching or shrinking of the signature vector. This change of length is generally non-uniform, i.e., part of the object may become larger, while another part may get smaller at the same time.

The shortest circular-path search now operates on a graph that we can visualize best in three dimensions. Two dimensions $(x, y)$ correspond to the spatial pixel position, just as in the Corridor Scissors above. However, we introduce a third dimension $(z)$, which corresponds to the pixel position on the signature. In this sense, we denote the graph nodes as $V=\left\{v_{x, y, z}\right\}$. Each graph node $v_{\{x, y, z\}}$ is attributed with the matching cost between the texture block at $(x, y)$ in the current frame, and the block that was saved at signature position $z$. If the size of the object contour would not be allowed to stretch or shrink, every step in the $x / y$ plane should be reflected by a step in $z$ direction. But since the object contour in the new image may be larger, we also allow to make a step in the $x / y$ plane without advancing in the $z$ direction. This corresponds to staying at the same contour position in the original image, while advancing in the current image. On the other hand, to be able to shrink the object contour, it is allowed to advance two steps in the $z$ direction in only one step. For all of these possibilities, edges are introduced in the three-dimensional graph (see Fig. 11.20). Because the signature is taken from a closed contour, edges are also added from the last layer of $z$ to the first.

### 11.4.3 Circular-path search with object signatures

Searching for the new object contour can now be carried out again with a shortest circular-path search but now within the presented 3-D graph. Note that in this graph, the edges in $z$ direction are directed, since it is not allowed to move backwards on the signature. The search in this 3 D graph poses two main difficulties. First, the number of nodes in the full three-dimensional graph is so high that the computational complexity becomes impractical. Second, the graph is non-planar, and the shortest circular-path algorithm above cannot be applied. An MSA algorithm would still be possible, but it would increase the computation time even further. We approach these two problems by first significantly reducing the size of the graph, and subsequently using a very fast approximation of the above shortest circular-path algorithm.


Figure 11.20: A small part of the signature search-graph (Fig. 11.21). Directed edges from one layer to the next (a) are the standard links. Taking this edge corresponds to going to a neighboring pixel in the image and making a step to the next position on the signature. Additionally, edges within one layer represent steps in the image, without proceeding on the signature (the contour is stretched). Finally, edges skipping one layer (b) make a step in the image and skip one position in the signature (the contour is shrunk).


Figure 11.21: Instead of considering the complete 3-D graph (2-D image position + signature position), only the nodes near to the original contour are used in the computation. The center of the spiral corresponds to the previous contour.

## Graph pruning

Carrying out a shortest circular-path search on the three-dimensional graph as constructed above is intractable, since this graph would consist of $W \cdot H$. $C$ nodes, where $W \times H$ is the image size and $C$ is the object contour length. However, if we assume that the object motion is limited, we can use the approach of Section 11.4.1 to restrict the search to a small corridor around the previous contour. More clearly, this means that all nodes $(x, y, z)$ that are more than $d$ pixels away from the previous object contour are removed from the graph. The resulting graph looks like the two-dimensional graph of the Corridor Scissors algorithm, but every node is duplicated $C$ times in the $z$-direction. This can be visualized as a hollow tube, extruded from the object contour in the $z$-direction.

Even though this graph is already significantly smaller, we can reduce the size further by limiting the maximum amount of shrinking and stretching of the contour. Let $C_{i}=\left(x_{i}, y_{i}\right)$ be the sequence of pixels in the original contour. Assuming that the new contour has to follow the old contour with not more stretching or shrinking than an offset of $s$ pixels along the contour, we can also exclude nodes $v_{x, y, z}$ from the graph, if the $z$-coordinate of the node deviates more than $s$ from the previous object contour. When combined, we obtain the following set of remaining nodes.

$$
\begin{equation*}
V=\{v_{x, y, z} \mid \exists\left(x_{i}, y_{i}\right): \underbrace{\left|x_{i}-x\right|<d \wedge\left|y_{i}-y\right|<d}_{\text {limited motion }} \wedge \underbrace{|i-z|<s}_{\text {limited deformation }}\} . \tag{11.4}
\end{equation*}
$$

The resulting graph has the shape of a spiral with one turn around the object (Fig. 11.21). The number of nodes in this graph is approximately $C d^{2}$ only.

## Fast approximate calculation of shortest circular paths

The graph that we obtain for the object-signature fitting is non-planar, and we cannot use the algorithm described previously for computing the shortest circular path. However, for this graph it also holds that the length of the circular path is large compared to the corridor width. Previously, we have observed that for these long corridor graphs, there usually exists a subpath that is common to all shortest paths along the corridor. If we simply assume that such a common subpath exists, we can define a fast algorithm for shortest circular paths.

The algorithm is motivated by the AP1 algorithm for planar graphs. We first conduct an ordinary shortest-path search from the "left" side of the corridor graph to the "right" side, similar to Step 2 of AP1. Again, we mark the last possible node at which the shortest-path tree to the destination
nodes is rooted as $v_{r}$. In a second step, we conduct a similar shortest-path search, but now in the opposite direction, starting from the just found node $v_{r}$ to the "left" side. The node at which the paths to the destination nodes split is marked with $v_{l}$. It only remains to connect $v_{r}$ with $v_{l}$ in a similar way as described in Step 4 of the algorithm for planar graphs.

Assuming that a common subpath exists, this algorithm computes the correct shortest circular path. Otherwise, the correct solution might not be found, but it is still nearly optimal, as it is composed of two shortest-path segments.

### 11.4.4 Tracking results

This section, presents tracking results for three sequences. The first sequence shows a car that is seen in different views throughout the sequence, while in later parts of the sequence, the car passes behind some trees. The object contour was initialized at the first frame with the normal Corridor Scissors algorithm. This contour was used to derive the object signature, and this signature was kept fixed through the complete tracking (Fig. 11.22). We see that the tracked contour follows the object. It is interesting to notice that the tracked contour keeps locked at the parts of the car that were defined in the first frame. In later frames, when the view of the car changes, new parts become visible, but the tracked contour still follows the originally defined content. This can be observed clearly at the rear of the car, but also at the back window, which was not visible in the first frame. The front window, which was present in the first frame, is occluded in later frames. To compensate for this, the algorithm pushes the contour away from the car to follow an image texture that is equally dark as the front window in the first frame. Around frame 40, the tracking result becomes worse and should be reinitialized or corrected.

In Figure 11.23, a new contour was initialized at frame 50 of the same sequence as above. During this part of the sequence, the car passes behind some trees. Since the signature holds information about the correct contour, the algorithm is not distracted when the car crosses the first tree (frame 70). However, in later frames (75-90), also the lighting conditions change and the algorithm loses the object.

Figure 11.24 shows tracking results for the foreman and the paris sequence. In the foreman sequence, it is interesting to see that the tracked contour at the helmet sometimes deviates from the true boundary. The reason is that the signature not only includes texture from the foreground, but also from the background. Hence, because a dark part of the background was visible along the top of the helmet in the first frame, the tracking algorithm searches for similar backgrounds in later frames. This leads to the

Chapter 11. Manual Segmentation and Signature Tracking
tracking error that is visible in frame 100. On the other hand, at the left side of the helmet in frame 100, the contour is inside of the helmet, because the algorithm tries to prevent to pass by the dark area in the background.

In the paris sequence, the woman is first tracked successfully, but then, the algorithm locks to a local optimum around frame 100, because of fast object motion. However, in a later frame, the algorithm recovers again and yields the correct contour.

### 11.5 Summary and conclusions

This chapter has introduced Corridor Scissors as a new technique for semiautomatic image segmentation. This tool provides an elegant user interface, in which the object is marked coarsely with a broad corridor around the object border. Within this corridor, the object border is detected with pixel accuracy. Since the path within the corridor is adapted each time the corridor shape is changed, the interface supports incremental modifications to the segmentation to improve its quality.

The Corridor Scissors tool is based on a newly developed algorithm for computing shortest circular paths, which was also described in this chapter. In practical cases, the described algorithm has the same computational complexity as an ordinary shortest-path computation using the Dijkstra algorithm. The described algorithm is applicable to general planar graphs and in an approximation also for some types of non-planar graphs. Consequently, our algorithm is also useful for other applications like shapematching [177] or crack detection in borehole core images [7], in which comparable shortest circular-path problems occur.

Because of the low computation time of the shortest circular-path algorithm and the fact that the algorithm only has to consider the pixels within the corridor, the user-interface of the Corridor Scissors tool operates in realtime. Compared to the Intelligent Scissors algorithm, which computes the shortest paths to all pixels, this provides a significant $a$-priori reduction of the data to be processed.

Finally, the Corridor Scissors tool was extended with a tracking algorithm. This tracking algorithm saves the texture around the object outline in a signature vector, which it is using during the tracking process to increase the robustness of finding the correct object border and decrease the probability of being irritated by background clutter. The tracking algorithm is also based on the shortest circular-path algorithm, but operating on a different cost function, which incorporates the signature information.

### 11.6 Discussion on the signature tracking technique

The concept of integrating knowledge about the object texture in the tracking process appears promising, since it improved the tracking results. Note that the object signature used in the tracking can also be considered as some kind of object model. However, we have observed two problems with this model that should be considered in future work.

First, the texture blocks in the signature vector include both foreground content and background content. In many cases (see foreman sequence), this leads to wrong contours because the algorithm finds a mismatch in the background texture. We conducted experiments to exclude the background content in the comparison. This is easily possible, as the foreground object mask is available from the previous segmentation. However, this resulted often in the situation that the detected contour moved inside of the object if it has a uniform color.

The second observation is that the tracking results can be improved by including also shape information. It is our opinion that this will improve the robustness against being trapped in a locally optimum contour. Furthermore, this may also help to solve the first problem. To see this, assume again that we would exclude background texture from the computation of matching cost. To prevent the contour to move inside the object, we have to apply some force to push the contour outwards. This force could be realized by the $a$-priori shape information. Unfortunately, object shape is a global feature that is difficult to integrate in the proposed framework. One promising direction may be to replace the object contour model with a mesh-based object surface model [21, 44], because this includes more texture information from the object and a shape-deformation cost can be defined based on the mesh geometry.


Figure 11.22: Tracking in the vectra sequence. The signature is initialized in frame 1 and kept fixed throughout the tracking. Because the 3-D view of the car changes but the signature still contains the object outline from the original view, the computed outline differs from the true object boundary. Note especially the contour changes at the rear of the car and the front window.


Figure 11.23: Tracking the car while it passes behind some trees. The tracking is reinitialized at frame 50 of the sequence (a). When the car passes the first tree trunk (b), the texture information in the signature prevents the tracking from locking to the trunk as the new boundary. Later, in (c) and (d), large occlusions and changes in the lighting let the tracking deteriorate.

(a) Frame 1.

(c) Frame 100 .

(e) Frame 225.

(b) Frame 1.

(d) Frame 100.

(f) Frame 300 .

Figure 11.24: Tracking results for the foreman sequence (left column) and the paris sequence (right column).

(a) Input image that defines edge costs.

(b) After Step 3 of AP1, a long common subpath is found.

(c) The shortest circular path.

Figure 11.25: An example for which the shortest circular path can be computed with only AP1. Darker colors indicate lower cost.

### 11.7 Appendix: step-by-step examples

This section illustrates the execution of the proposed shortest circular-path algorithm (Section 11.3.2) by some example runs. All examples use rectangular grid graphs with edges places as shown in Fig. 11.18. The costs are taken from an input image such that lower luminances correspond to lower costs. Circular paths are searched for along the horizontal direction through the rectangle (right column connects to left column). The selection of the examples does not reflect the typical input in the Corridor Scissors algorithm, but the examples are selected to clearly illustrate the execution of the algorithm. From the computational point of view, all examples are more complex than the typical input in the Corridor Scissors case.

## Example 1 (Fig. 11.25)

The first example (Fig. 11.25) uses a low-frequency cost image. In Step 2 and 3 , the algorithm computes the minimum-cost paths from the top-left corner to the top-right corner and, similarly, from the bottom-left corner to the bottom-right corner. These two paths share a long subpath along most of the corridor graph. Hence, AP1 can already determine the shortest circular path by connecting the nodes $v_{r}$ and $v_{l}$ in Step 4. Note that this step only computes a shortest path in the area composed of the two small triangles at both sides, since the large black areas can be excluded from the


Figure 11.26: The shortest-paths trees on both sides of a cut (compare to Fig. 11.11(c)) for a typical real-world example. It is clearly visible that all paths join quickly to a common path along most of the corridor.
computation. This is the typical case as it usually occurs in the Corridor Scissors algorithm (see also Fig. 11.26).

## Example 2 (Fig. 11.27)

In a second experiment, we used a natural image as cost data (but note that the shortest circular path does not have any semantic meaning here). The algorithm starts with computing the two shortest paths along the top and the bottom side of the rectangle (Fig. 11.27(a)). Unlike in the last example, these two paths do not join, so that we cannot use AP1 compute the shortest circular path. Hence, we have to use AP2 for this example.

This part of the algorithm starts with the initial recursion step, in which a shortest-path search is initiated from the middle position on the left side. Indicated in Fig. 11.27(b) are the two last paths, starting from the middle position, that still touch the top and bottom paths from the last step, respectively. For the top path, the last position for which we still get a touching path, is the middle path itself. Consequently, we have the situation of two touching paths in the top half of the image and the shortest circular path in this area can be computed in one step (again by joining $v_{r}$ and $v_{l}$ ). The range of $\bar{V}$ that is completed is indicated by the bars at the sides of the image. Unfortunately, the range that could be processed in the bottom half is very small and further recursions are required.

Since the top part of the image could be completed in one step, we only have to consider the bottom part in the recursion (Fig. 11.27(c)). In the remaining area, a shortest-path search is initiated again from the middle position at the left side. Note that the graph on which the search is now carried out is much smaller, since the top half can be excluded (black


Figure 11.27: Step-by-step example of the execution of the shortest circularpath algorithm. The luminance of the input image defines the cost. Black areas are parts of the graph that could are excluded from the computation.
area). In this step, only a small range at the top and a larger region at the bottom can be completed. Because none of the two halves are finished completely in this step, both halves are further processed recursively (top half: Fig. 11.27(d), bottom half: Fig. 11.27(e)). After these steps, all nodes in $\bar{V}$ are covered. The globally shortest circular path is selected as the shortest path of the solution from each of the processed ranges. This solution is presented in Fig. 11.27(f).

## Example 3 (Fig. 11.28)

The input data that was synthesized for this example (Fig. 11.28) was motivated by the worst-case data (Fig. 11.16(a)) for the algorithm. The input shows many thin low-cost paths, so that there is no obvious common shortest subpath. Nevertheless, the two bounding paths (Fig. 11.28(b)) almost touch, and the shortest circular-path problem can be solved in only four steps of recursion. We made similar observations with input data that is just random noise. This suggests that the computation time is usually low, even with complicated input data. An important factor is the ratio of corridor length to its width, since the probability of joining paths also increases when this ratio increases.

(a) Input data.

(b) AP1, since the two paths do not touch.

(c) First recursion step.

(d) Second recursion step.

(e) Third recursion step.

(f) Result after fourth recursion step.

Figure 11.28: An example case that is close to the worst-case pattern shown in Fig. 11.16. Here, AP2 has to be applied, but the computation can be completed with only four recursion steps.

## Part III

## From Camera Motion to 3-D Models

## Chapter <br> 12

## Estimation of Physical Camera Parameters

The earlier presented camera-motion estimator is employing projective transforms to describe the geometric mapping between input frames. It was shown that these transforms are general enough to include all kinds of motion that occur for a rotational camera with varying focal length. In fact, when taking a sequence of elementary camera operations, it is easy to get the corresponding transform between two input images. However, the inverse problem of calculating the physical transform parameters from the frame-to-frame homographies is far more complicated. Despite the problems involved, it is valuable to carry out the estimation of the physical camera parameters, since it enables a whole new area of applications. Examples are video content analysis, where camera motion represents a valuable feature for sequence classification, or augmented-reality applications that insert computer-generated 3-D objects into a natural scene. This chapter presents two algorithms for factorizing the projective transforms of the camera motion into the physically meaningful camera rotation angles and the camera focal length. The first algorithm applies a linear optimization approach that is fast, but which has only limited accuracy. The second algorithm uses a non-linear bundle-adjustment algorithm that provides a high accuracy. Both algorithms have been combined with our multi-sprite technique of Chapter 6 to support unrestricted camera motion.

### 12.1 Introduction

In current video coding and video-object segmentation systems, cameramotion is usually described with the projective motion model with eight parameters. A prominent example is the sprite-coding tool of the MPEG-4 standard, which uses the projective motion model to align background frames into a common background sprite image. In our video-object segmentation algorithm, we applied the same motion model, because of its ease of use and to enable the easy integration of the segmentation algorithm into MPEG-4 encoders.

In Chapter 2, it was derived that this model can describe any image motion that results from a rotating camera with varying focal length. Furthermore, the projective motion model also allows the alignment of the input images to a larger background-sprite image that can be used to reconstruct any arbitrary camera view from this background image. Later, Chapter 6 clarified that in practice, the principle of background sprites should be generalized to prevent degenerated transforms. However, with this multi-sprite generalization, the projective motion model can be applied to describe any motion of rotatorial cameras.

The parameters of the projective motion model describe the camera motion in an abstract way, without a direct correspondence to physically meaningful operations. However, when we derived the projective motion model for rotational cameras, it was indicated that the model can be considered as a concatenation of elementary physical operations, like rotating the cameras, and perspectively projecting the 3-D object onto the image plane. Unfortunately, it is not obvious how the parameters of these elementary operations can be recovered from the combined transformation matrix. In fact, it is not even possible to factorize every projective transformation into camera rotations and changes of camera zoom, since the general projective transformation also includes physically impossible transformations like anisotropic scaling or image skewing.

Nevertheless, it is advantageous or even required for some applications to describe the camera motion in terms of rotation angles or zoom (change of focal length). One of these applications is content analysis, where the camera motion can be used as a feature to help in analysing the video content. For example, a zoom-in operation should direct the attention of the viewer to a specific detail in the scene, where the important object is usually located at the center of the zoom.

Whereas a qualitative analysis of camera motion can be sufficient for content analysis, there are other applications that require parameters with a high accuracy. An example is an augmented-reality application, where virtual 3-D objects are added to a recorded video such that they are seam-
lessly integrated. This can only be achieved if the view onto the synthetic $3-\mathrm{D}$ objects is synchronized with the current camera view. For the camera control, the current camera rotation-angles and the focal lengths have to be known. Another application is the creation of cylindrical or spherical panoramic images [30] from video sequences, where especially the focal length of the camera is required. These parameters are usually not known and have to be estimated from the video sequence itself.

In this chapter, we propose two algorithms that factorize projective motion parameters into a sequence of elementary, physically meaningful transformations. More clearly, it takes a sequence of projective motion parameters as input and generates a corresponding sequence of camera rotation angles and focal lengths for each of the input frames. The parameter estimation is carried out in two steps, where the first is a fast linear calibration algorithm based on the image of the absolute conic. While the accuracy of this first step can be sufficient for content-analysis applications, we also describe an optional refinement step. This refinement step uses a non-linear optimization algorithm to yield an increased accuracy of the obtained parameters. Both algorithms incorporate the multi-sprite partitioning of a sequence and consequently, they have no limitations about the range of parameters.

The chapter is organized as follows. We begin with a brief repetition of the global-motion estimation to clarify the notation used in this chapter. Furthermore, we give a short survey of previous approaches and discuss their performance. Section 12.3 describes the linear camera-calibration algorithm that was proposed by Hartley et al. in [85] and we present our extension of this algorithm to support the generalized multi-sprite approach. A non-linear calibration is described in Section 12.4, where we again consider in particular the integration of multi-sprites. Finally, we present visualizations of results for several sequences in Section 12.5.

### 12.1.1 Geometry of background image generation

This section gives a brief review of the geometry of rotational cameras to introduce the notation that we are going to use in this chapter.

Let us define the $3-\mathrm{D}$ world coordinate system such that the camera is located at its origin. The camera captures a number of images $i$ with different rotations $\mathbf{R}_{i}$ and focal lengths $f_{i}$. The optical axis of the camera intersects the image plane at the principal point $o_{x}, o_{y}$. The focal length $f_{i}$ and principal point are collected in an intrinsic camera parameters matrix

$$
\mathbf{K}_{i}=\left[\begin{array}{ccc}
f_{i} & \tau & o_{x}  \tag{12.1}\\
0 & \eta f_{i} & o_{y} \\
0 & 0 & 1
\end{array}\right]
$$



Figure 12.1: The camera is located at the origin of the world coordinate system. The sprite plane is assumed to be orthogonal to the $z$-axis. Input images are at a distance to the origin that is equal to the focal length when the image was captured. A point $(x / w, y / w)$ on the image is projected onto the sprite position $(\hat{x} / \hat{w}, \hat{y} / \hat{w})$.

Additionally, the parameter $\eta$ denotes the pixel aspect ratio and $\tau$ is the image skew. However, for typical CCD cameras, we can assume zero skew $\tau=0$ and square pixels $\eta=1$, so that we obtain the simpler intrinsic parameters matrix

$$
\mathbf{K}_{i}=\left[\begin{array}{ccc}
f_{i} & 0 & o_{x}  \tag{12.2}\\
0 & f_{i} & o_{y} \\
0 & 0 & 1
\end{array}\right] .
$$

Using homogeneous coordinates $\mathbf{p}=(x, y, w)^{\top}$ for 2-D image positions, we can obtain the projection of a 3-D point onto the image as $\mathbf{p}=\mathbf{K}_{i} \cdot(x, y, z)^{\top}$. Conversely, we can use the inverse $\mathbf{K}_{i}^{-1}$ to map 2-D points back to 3-D direction vectors. Concatenating these transforms with an intermediate 3-D rotation matrix $\mathbf{R}_{i}$ gives the transformation between two images or, similarly, between a background sprite image and an input frame, as

$$
\left(\begin{array}{l}
x  \tag{12.3}\\
y \\
w
\end{array}\right)=\underbrace{\left(\begin{array}{ccc}
f_{i} & 0 & o_{x} \\
0 & f_{i} & o_{y} \\
0 & 0 & 1
\end{array}\right)}_{\mathbf{K}_{i}} \mathbf{R}_{i} \underbrace{\left(\begin{array}{ccc}
1 / \hat{f}_{s} & 0 & -\left(\hat{o}_{x}\right)_{s} / \hat{f}_{s} \\
0 & 1 / \hat{f}_{s} & -\left(\hat{o}_{y}\right)_{s} / \hat{f}_{s} \\
0 & 0 & 1
\end{array}\right)}_{\hat{\mathbf{K}}_{s}^{-1}}\left(\begin{array}{l}
\hat{x} \\
\hat{y} \\
\hat{w}
\end{array}\right)=\mathbf{H}_{i}\left(\begin{array}{l}
\hat{x} \\
\hat{y} \\
\hat{w}
\end{array}\right) .
$$

This set-up is also visualized in Figure 12.1. Note that we denote all coordinates and parameters related to sprite coordinates with a hat on the variable name. Moreover, we use indices $i$ for variables relating to input images and $s$ for variables relating to sprites. Consequently, the transformation from a sprite $s$ to an image $i$ is denoted as $\mathbf{H}_{i}^{(s)}$. If the superscript is omitted, we simply mean the sprite to which the image $i$ was assigned by the multi-sprite partitioning.

Multiplying the intrinsic and extrinsic transformation matrices together, we obtain the combined matrix $\mathbf{H}_{i}$, describing the projection of coordinates on the background plane onto image $i$. Writing with inhomogeneous coordinates and normalizing $\mathbf{H}_{i}$ to $h_{22}=1$ gives the inhomogeneous formulation of the projective motion model

$$
\begin{equation*}
x=\frac{h_{00} \hat{x}+h_{01} \hat{y}+h_{02}}{h_{20} \hat{x}+h_{21} \hat{y}+1}, \quad y=\frac{h_{10} \hat{x}+h_{11} \hat{y}+h_{12}}{h_{20} \hat{x}+h_{21} \hat{y}+1} . \tag{12.4}
\end{equation*}
$$

Taking the sprite-to-image transformations $\mathbf{H}_{i}, \mathbf{H}_{k}$ for two images $i, k$, we can obtain the transformation from image $k$ to $i$ by first mapping the point of image $k$ onto the background and then mapping it back onto image $i$. We denote this inter-image transform as $\mathbf{H}_{i ; k}=\mathbf{H}_{i} \mathbf{H}_{k}^{-1}$.

### 12.1.2 Global motion estimation

The input for the camera-calibration algorithm is obtained from our globalmotion estimation, where we use the more accurate parameters after the direct estimation step (Chapter 5). From this motion estimator, we obtain the parameters $h_{00}, \ldots, h_{21}$, corresponding to the multiplied matrices of Eq. (12.3).

For the case that the rotation between two frames is large, Chapter 6 has shown that they cannot be projected onto the same sprite. Following our multi-sprite technique, those frames are assigned to separate sprites. In the multi-sprite algorithm, the intention was to minimize the size of the generated sprite images for improved coding efficiency, but we can also use the same algorithm without modification to prevent the limitation on the allowed camera rotations. The result of this multi-sprite partitioning is a partitioning of the input sequence of length $N$ into $M$ ranges

$$
\begin{equation*}
P=\left\{\left(1, p_{2}-1\right),\left(p_{2}, p_{3}-1\right),\left(p_{3}, p_{4}-1\right), \ldots,\left(p_{M}, N\right)\right\} \tag{12.5}
\end{equation*}
$$

where $p_{s}$ denotes the first frame used in sprite $s(s \in[1 ; M])$. Now, instead of computing motion parameters to a global reference sprite (which is not possible), we compute the motion parameters for each image relative to the sprite that it has been assigned to.


Figure 12.2: Transformations between sprites and input frames. Every image $i$ is connected to its assigned sprite $k$ by the transform $\mathbf{H}_{i}^{(k)}$. The first and last images from each partition are also connected to the previous or next sprite, respectively.

The multi-sprite partitioning assigned the images to separate sprites, but since we also need to know the geometric relation between images that are far apart, we also have to connect all the sprites. To achieve this, we compute for the first frame $p_{s}$ that has been assigned to a sprite $s$ not only the transform $\mathbf{H}_{p_{s}}^{(s)}$, but also the transform to the previous sprite $s-1$, which we denote by $\mathbf{H}_{p_{s}-1}^{(s-1)}$. Similarly, we also compute the transform between the last frame $p_{s+1}-1$ of the sprite to the next sprite $s+1$ (see Fig. 12.2). With these connecting transforms, we can compute the transformation between any two frames (e.g., frame $i$, assigned to sprite $s$, and frame $k$, assigned to sprite $s-1$ ) by

$$
\begin{equation*}
\mathbf{H}_{i ; k}=\left(\mathbf{H}_{i}^{(s-1)}\right) \cdot\left(\mathbf{H}_{p_{s}}^{(s-1)}\right)^{-1} \cdot\left(\mathbf{H}_{p_{s}}^{(s)}\right) \cdot\left(\mathbf{H}_{k}^{(s)}\right)^{-1} \tag{12.6}
\end{equation*}
$$

For images that are several sprites apart from each other, more transforms have to be concatenated.

When chaining the inter-sprite transformations, we should to be careful about the parameterization of the transformations. It is common practice to normalize the projective transformations $\mathbf{H}=\left\{h_{i k}\right\}$ to $h_{22}=1$, and we also apply this normalization in our motion-estimation algorithms. While this normalization is valid for small camera rotation-angles, it introduces problems if the rotation angle becomes larger. The problem is that at 90 degrees, we obtain $h_{22}=0$ after the concatenation of the transforms, which cannot be normalized, and for larger angles, $h_{22}$ becomes negative. This will result in a change of orientation that makes it impossible to recover the correct rotation angles. Consequently, we cannot normalize the transforms to $h_{22}=1$ when they are chained together. Note that it is still possible to carry out the inter-image motion estimation using normalized parameters, since the multi-sprite partitioning ensures that the transforms do not degenerate.

### 12.2 Previous work

Camera calibration is an active topic of research and several algorithms have been proposed to solve the problem. The approaches can be classified according to the applied camera models. Their main differences are in the parameters that are assumed to stay fixed, such as, e.g., the focal length or the principal point.

### 12.2.1 Estimation of focal length

Once the principal point and the focal lengths are known, it is straightforward to compute the camera rotation $\mathbf{R}$ by pre- and post-multiplying the intrinsic parameter matrices to the transform $\mathbf{H}_{k ; i}$, this leading to $\mathbf{R}=$ $\mathbf{K}_{k}^{-1} \mathbf{H}_{k ; i} \mathbf{K}_{i}$. Since the principal point is usually near the image center, the problem reduces to find an estimate for the focal lengths.

In [180], Szeliski et al. described a simple approach for estimating the focal length from two images that were captured with a rotating camera. As shown previously, the transformation from frame $i$ to $k$ is then composed of the elementary transformations $\mathbf{H}_{k ; i}=\mathbf{K}_{k} \mathbf{R} \mathbf{K}_{i}^{-1}$. Szeliski's algorithm assumes that the origin of the image coordinate system is at the principal point ( $o_{x}=o_{y}=0$ ). If this is not the case, the coordinate system can be shifted easily by factorizing $\mathbf{K}$ into $\mathbf{K}=\mathbf{T K} \mathbf{K}^{\prime}$, with

$$
\mathbf{T}=\left(\begin{array}{ccc}
1 & 0 & o_{x}  \tag{12.7}\\
0 & 1 & o_{y} \\
0 & 0 & 1
\end{array}\right) \quad \text { and } \quad \mathbf{K}^{\prime}=\left(\begin{array}{ccc}
f & 0 & 0 \\
0 & f & 0 \\
0 & 0 & 1
\end{array}\right)
$$

so that we obtain

$$
\begin{equation*}
\mathbf{H}_{k ; i}=\mathbf{T K}_{k}^{\prime} \mathbf{R K}_{i}^{\prime-1} \mathbf{T}^{-1} \tag{12.8}
\end{equation*}
$$

or alternatively,

$$
\begin{equation*}
\mathbf{H}_{k ; i}^{\prime}=\mathbf{T}^{-1} \mathbf{H}_{k ; i} \mathbf{T}=\mathbf{K}_{k}^{\prime}{ }_{k} \mathbf{R K}_{i}^{\prime-1} . \tag{12.9}
\end{equation*}
$$

Multiplying the rotation matrix $\mathbf{R}=\left\{r_{m n}\right\}$ with the intrinsic parameter matrices gives

$$
\mathbf{H}_{k ; i}^{\prime}=\left(\begin{array}{ccc}
h_{00} & h_{01} & h_{02}  \tag{12.10}\\
h_{10} & h_{11} & h_{12} \\
h_{20} & h_{21} & h_{22}
\end{array}\right) \sim\left(\begin{array}{ccc}
r_{00} & r_{01} & r_{02} f_{i} \\
r_{10} & r_{11} & r_{12} f_{i} \\
r_{20} / f_{k} & r_{21} / f_{k} & r_{22} f_{i} / f_{k}
\end{array}\right) .
$$

Since we know that the rows and columns of the rotation matrix $\left\{r_{i}\right\}$ are orthogonal, we can derive

$$
\begin{equation*}
h_{00} h_{10}+h_{01} h_{11}+h_{02} h_{12} / f_{i}^{2}=0 \tag{12.11}
\end{equation*}
$$

and

$$
\begin{equation*}
h_{00} h_{01}+h_{10} h_{11}+h_{20} h_{21} \cdot f_{k}^{2}=0 \tag{12.12}
\end{equation*}
$$

From this, we can determine the focal lengths as

$$
\begin{equation*}
f_{i}^{2}=\frac{-h_{02} h_{12}}{h_{00} h_{10}+h_{01} h_{11}} \quad \text { and } \quad f_{k}^{2}=\frac{h_{00} h_{01}+h_{10} h_{11}}{-h_{20} h_{21}} \tag{12.13}
\end{equation*}
$$

Alternatively, we also can exploit the fact that the rows and columns of rotation matrices have unit norm. Since the transformation matrix can be scaled, we cannot test directly for unit norm, but we can still test for equal norm by

$$
\begin{equation*}
h_{00}^{2}+h_{01}^{2}+h_{02}^{2} / f_{i}^{2}=h_{10}^{2}+h_{11}^{2}+h_{12}^{2} / f_{i}^{2} \tag{12.14}
\end{equation*}
$$

or

$$
\begin{equation*}
h_{00}^{2}+h_{10}^{2}+h_{20}^{2} \cdot f_{k}^{2}=h_{01}^{2}+h_{11}^{2}+h_{21}^{2} \cdot f_{k}^{2} . \tag{12.15}
\end{equation*}
$$

Again, we can derive equations ${ }^{1}$ for the focal length as

$$
\begin{equation*}
f_{i}^{2}=\frac{h_{02}^{2}-h_{12}^{2}}{h_{11}^{2}-h_{00}^{2}+h_{10}^{2}-h_{01}^{2}} \quad \text { and } \quad f_{k}^{2}=\frac{h_{11}^{2}-h_{00}^{2}+h_{01}^{2}-h_{10}^{2}}{h_{20}^{2}-h_{21}^{2}} . \tag{12.16}
\end{equation*}
$$

In total, we obtained four different equations to compute the focal lengths. Each of the focal lengths $f_{i}, f_{k}$ can be computed with two approaches, which we will further denote as the orthogonal approach and the equal-norm approach. If it is known that the focal length is fixed, the authors propose to take the geometric mean of the estimates for $f_{i}, f_{k}$.

## Degenerated cases

The described algorithm seems attractive for computing the focal length since it is easy to implement. However, it turns out to be very numerically unstable in practice. To get more insight into the behaviour, let us consider some special (but common) cases.

If the rotation angle between two frames is small, we have $h_{j j} \approx 1$, while for $i \neq j, h_{i j} \approx 0$. Consequently, all the denominators in Equations (12.13) and (12.16) approach zero, which makes the estimation unstable, especially at the presence of noise.

Let us observe the behaviour for rotations around the three coordinate axes. For the $x$-axis, it holds that

$$
\mathbf{H}^{\prime} \sim\left(\begin{array}{ccc}
1 & 0 & 0  \tag{12.17}\\
0 & \cos \alpha & f_{i} \sin \alpha \\
0 & -\left(1 / f_{k}\right) \sin \alpha & \left(f_{i} / f_{k}\right) \cos \alpha
\end{array}\right)
$$

[^19]

Figure 12.3: Estimation of focal length as proposed by Szeliski et al. in [180]. Even though the considered images were 25 frames apart, the stability of the obtained estimate is very low.
from which we can derive the constraint of orthogonal columns: $1 \cdot 0+$ $0 \cdot \cos \alpha+0 \cdot\left(-1 / f_{k}\right) \sin \alpha=0$. Obviously, this is not sufficient to solve for $f_{k}$. Similarly, the constraint for orthogonal rows does not provide $f_{i}$. However, using the approach of equal column norms works, since we obtain $f_{i}^{2}=\left((\cos \alpha)^{2}-1\right) /\left(-(\sin \alpha)^{2} / f_{k}^{2}\right)=f_{k}^{2}$. For rotation around the $y$-axis, similar results are obtained. When there is a pure rotation around the $z$-axis, all approaches fail to determine the focal lengths. There are also rotations around two axes that cause problems for the approach. Consider a rotation around the $z$-axis, followed by a rotation around the $y$-axis. In this case, the orthogonality approach for determining $f_{i}$ fails, because the transformation matrix is

$$
\mathbf{H}^{\prime} \sim\left(\begin{array}{ccc}
\cos \beta \cos \gamma & \cos \beta \sin \gamma & f_{i} \sin \beta  \tag{12.18}\\
-\sin \gamma & \cos \gamma & 0 \\
-1 / f_{k} \sin \beta \cos \gamma & -1 / f_{k} \sin \beta \sin \gamma & f_{i} / f_{k} \cos \beta
\end{array}\right)
$$

and from the orthogonality of rows, it follows that

$$
\begin{equation*}
-\sin \gamma \cos \beta \cos \gamma+\sin \gamma \cos \beta \cos \gamma+0=0 . \tag{12.19}
\end{equation*}
$$

Again, this provides no information about $f_{i}$.

## Evaluation

We used two real-world sequences to evaluate the estimation of focal length based on Eq. (12.13). The roma sequence shows a pure horizontal pan with


Figure 12.4: Estimation of focal length for different rotation angles between the frames for the roma sequence. Larger rotation angles give a better accuracy. The geometric mean of the two approaches of row and column norm gives good results, but only works if the focal length is fixed.
fixed focal length, while stefan has a more complicated camera motion with varying focal length. The transformations $\mathbf{H}$ were estimated with our global-motion estimator, and the principal point ( $o_{x}, o_{y}$ ) was assumed to be in the image center.

To estimate the focal length, we fixed the distance between pairs of pictures to 25 frames, so that the rotation between them is large. Figure 12.3 depicts results for two example input sequences. It is well visible that the accuracy of the estimation is low and often it degenerates into cases where the right-hand-side of the equations is negative, which leads to no solution, since the square-root of negative values is undefined. We only depicted the results for the column-norm approach. Results for equal row norms are not very different, and results for the orthogonality approaches are much worse.

In a second experiment, we computed the estimated focal length depending on the rotation angle between the two frames. This experiment was carried out only with the roma sequence, since this sequence shows a smooth horizontal pan. The estimation was carried out between the first frame of the sequence and a later frame. The results (Fig. 12.4) show that the estimation accuracy increases with a larger rotation angle. Interestingly, the estimation error for the row-norm approach seems to be just the opposite of the estimation error for the column-norm approach. Hence, we can get a good estimate even for small rotation angles, when we take the


Figure 12.5: The absolute conic $\Omega_{\infty}$ is transformed to the input frames and shows there as the IAC $\boldsymbol{\omega}^{(i)}$.
geometric mean between the two estimates. Obviously, this is only possible if the focal length is fixed.

To summarize, we can conclude that Eq. (12.13) can be applied to cases where the focal length is fixed and the rotation angle between the images is large. One such application can be image mosaicing from images. However, for video applications, the algorithm has a number of disadvantages: the principal point of the image should be known and the estimation accuracy is low or it even degenerates when the rotation angles are small.

### 12.3 Linear camera calibration

In the remainder of this chapter, we describe two algorithms for the estimation of physical camera parameters. The first algorithm is described in this section. It is a linear calibration algorithm based on proposal by Hartley et al. in [85], and which is extended to support multi-sprite motion estimation.

The next section will present a non-linear calibration algorithm that yields a higher accuracy. The second algorithm is an iterative optimization algorithm that can be implemented independently, or it can be initialized with the result of the first algorithm. The latter approach will provide a faster convergence of the solution.

### 12.3.1 Calibration using the image of the absolute conic

The linear calibration algorithm uses the method based on the transformation of the absolute conic $[87,85]$. The absolute conic $\Omega_{\infty}$ is defined as the

$$
\begin{align*}
& \text { points }(x, y, z, w)^{\top} \text { satisfying } \\
& \qquad(x, y, z) \mathbf{I}(x, y, z)^{\top}=0 \quad \text { and } \quad w=0, \tag{12.20}
\end{align*}
$$

with I being the $3 \times 3$ identity matrix. Hence, because of $w=0$, the absolute conic lies in the plane at infinity $\boldsymbol{\pi}_{\infty}$ (see Fig. 12.5). Since the trivial solution $x=y=z=w=0$ is not a valid point in projective space, the conic only consists of imaginary points. Transformed with the camera transformation $\mathbf{H}_{i}=\mathbf{K}_{i} \mathbf{R}_{i}$ for view $i$, the Image of the Absolute Conic (IAC) $\boldsymbol{\omega}^{(i)}$ is

$$
\begin{equation*}
\boldsymbol{\omega}^{(i)}=\mathbf{H}_{\mathbf{i}}^{-\top} \mathbf{I} \mathbf{H}_{\mathbf{i}}^{-1}=\mathbf{K}_{\mathbf{i}}^{-\top} \mathbf{R}_{\mathbf{i}}^{-\top} \mathbf{R}_{\mathbf{i}}^{-1} \mathbf{K}_{\mathbf{i}}^{-1}=\mathbf{K}_{\mathbf{i}}^{-\top} \mathbf{K}_{\mathbf{i}}^{-1}, \tag{12.21}
\end{equation*}
$$

where the last equality holds since $\mathbf{R}_{\mathbf{i}}{ }^{\top}=\mathbf{R}_{\mathbf{i}}{ }^{-1}$. Assuming zero skew and square pixels, $\boldsymbol{\omega}^{(i)}$ has the form

$$
\boldsymbol{\omega}^{(i)}=\left[\begin{array}{ccc}
1 / f_{i}^{2} & 0 & -o_{x} / f_{i}^{2}  \tag{12.22}\\
0 & 1 / f_{i}^{2} & -o_{y} / f_{i}^{2} \\
-o_{x} / f_{i}^{2} & -o_{y} / f_{i}^{2} & o_{x}^{2} / f_{i}^{2}+o_{y}^{2} / f_{i}^{2}+1
\end{array}\right] .
$$

Consequently, zero skew leads to the constraint $\omega_{01}^{(i)}=0$ and the square pixel assumption gives $\omega_{00}^{(i)}=\omega_{11}^{(i)}$.

Now, let us consider several views. The constraints that we derived previously are true for each of the views (each with its own $f_{i}$ ). Let us select one of the views as the reference view $r$. With the transformation $\mathbf{H}_{i ; r}$, we can convert the coordinates of the reference view $r$ to coordinates of view $i$. Moreover, we can transform the conic $\boldsymbol{\omega}^{(r)}$ from the reference view to other views according to

$$
\begin{equation*}
\boldsymbol{\omega}^{(i)}=\mathbf{H}_{i ; r}^{-\top} \boldsymbol{\omega}^{(r)} \mathbf{H}_{i ; r}^{-1} . \tag{12.23}
\end{equation*}
$$

The idea is now to use this transformation of conics to express the conics in all views $i$ using the conic parameters of the reference view $r$. Since the constraints for zero skew and square pixels must be fulfilled for all IACs $\boldsymbol{\omega}^{(i)}$, we can formulate constraints for all views and express these constraints in the parameters of the reference view. These equations can then be stacked into an equation system, from which we subsequently estimate the parameters of $\boldsymbol{\omega}^{(r)}$. Because the constraints of all views are expressed in the coordinate system of view $r$, we have enough equations to solve for the parameters of $\boldsymbol{\omega}^{(r)}$. When $\boldsymbol{\omega}^{(r)}$ is known, it is easy to obtain the intrinsic parameters using a Cholesky decomposition.

To exploit the constraints from all the views, the parameters of $\boldsymbol{\omega}^{(r)}$ are collected in a vector

$$
\begin{equation*}
\mathbf{c}=\left(c_{1}, \ldots, c_{6}\right)^{\top}=\left(\omega_{00}, \omega_{01}, \omega_{02}, \omega_{11}, \omega_{12}, \omega_{22}\right)^{\top} \tag{12.24}
\end{equation*}
$$

Note that $\boldsymbol{\omega}^{(r)}$ is symmetric so that $\mathbf{c}$ only holds six parameters. Based on Equation (12.23), we derive a linear equation, which expresses every component of $\boldsymbol{\omega}^{(i)}$ as a linear combination of the parameters of $\mathbf{c}$ :

$$
\begin{equation*}
\omega_{j k}^{(i)}=\sum_{n} \phi_{j k}^{n} \cdot c_{n}, \tag{12.25}
\end{equation*}
$$

where $\phi_{j k}^{n}$ depend on the transformation matrices $\mathbf{H}_{i ; r}^{-1}$.
The zero-skew assumption $\omega_{01}^{(i)}=0$ of view $i$ can now be written in parameters of the reference view as

$$
\begin{equation*}
\left(\phi_{01}^{1}, \ldots, \phi_{01}^{6}\right) \cdot \mathbf{c}=0 . \tag{12.26}
\end{equation*}
$$

Similarly, the square-pixel assumption $\omega_{00}^{(i)}-\omega_{11}^{(i)}=0$ can be expressed as

$$
\begin{equation*}
\left(\phi_{00}^{1}-\phi_{11}^{1}, \ldots, \phi_{00}^{6}-\phi_{11}^{6}\right) \cdot \mathbf{c}=0 . \tag{12.27}
\end{equation*}
$$

This process can be carried out for all views, generating two equations over $\boldsymbol{\omega}^{(r)}$ for each view. Stacking all the equations into a matrix $\mathbf{A}$, we obtain the overdetermined, homogeneous equation system $\mathbf{A c}=\mathbf{0}$, which we can use to calculate $\mathbf{c}$. A least-squares solution for this equation system can be obtained by carrying out a Singular Value Decomposition and taking the singular vector that corresponds to the smallest singular value. The parameter vector $\mathbf{c}$ directly gives the IAC $\boldsymbol{\omega}^{(i)}$. Remember that $\boldsymbol{\omega}^{(i)}=$ $\mathbf{K}^{-\top} \mathbf{K}^{-1}$, where $\mathbf{K}$ is an upper triangular matrix. To derive $\mathbf{K}_{i}$ from $\boldsymbol{\omega}^{(i)}$, we carry out a Cholesky decomposition. ${ }^{2}$

Repeating the above algorithm to iterate the reference frame through each of the views gives us the intrinsic parameters $\mathbf{K}_{i}$ for all views. Once all the intrinsic parameters are known, we can obtain the camera rotation. We start with the transformation between two views $i$ and $k$, given by

$$
\begin{equation*}
\mathbf{H}_{k ; i}=\mathbf{H}_{k} \mathbf{H}_{i}^{-1}=\mathbf{K}_{k} \mathbf{R} K_{i}^{-1} \tag{12.28}
\end{equation*}
$$

where $\mathbf{R}$ is the rotation between the two views. Since we know the homographies $\mathbf{H}_{k}, \mathbf{H}_{i}$ and the intrinsic parameter matrices, we recover the rotation matrix as

$$
\begin{equation*}
\mathbf{R}=\mathbf{K}_{k}^{-1} \mathbf{H}_{k} \mathbf{H}_{i}^{-1} \mathbf{K}_{i} . \tag{12.29}
\end{equation*}
$$

[^20]
### 12.3.2 Integration of multi-sprite motion estimation

The algorithm presented so far assumed that accurate inter-image transforms between all pairs of views are available. However, the chaining of inter-image transforms to obtain transformations between frames that are further apart leads to an accumulation of errors, which subsequently also results in an inaccurate parameter estimation. This motivates why we added an additional motion-estimation step to our feature-based motion estimator. The solution is that instead of chaining inter-image transforms, we determine long-term motion parameters relative to a global backgroundsprite image. This provides high-accuracy parameters between this background image and a current view. Inter-image transforms between a pair of views $i, k$ can be obtained from these by just concatenating the two sprite transforms $\mathbf{H}_{i} \mathbf{H}_{k}^{-1}$. Consequently, there is less accumulation of errors even for transforms between distant views.

A consequence of using a global background sprite is that the supported camera motion is limited. Since the background sprite is a planar manifold, only a (theoretical) maximum of 180 degrees field-of-view can be covered. Practically, the usable camera rotation-angles are much lower. As a solution to this problem, we used the multi-sprite approach to partition the video sequence into sub-sequences and to synthesize a separate background sprite for each of these sub-sequences. This means that we cannot apply the above calibration algorithm directly, because the transforms of two views can be relative to different background sprites.

To obtain inter-image transformations for views that were assigned to different sprites, we use the approach described in Section 12.1.2. While the original multi-sprite algorithm computed motion parameters only between the current view and the sprite to which they were assigned to, we extend this concept by computing also the transforms to the previous or successive sprites for the first and last frame of a sub-sequence, respectively. These transforms allow to chain the transformations between sprites to allow for large rotation angles. However, note that the number of chained transforms is significantly less than when chaining transforms between successive images. Typically, we have only about five sprites, which usually cover up to several thousands of frames. Hence, the error accumulation is neglegible. Moreover, notice that the motion parameters are obtained with the long-term motion estimator, such that the parameters have a-priori a higher accuracy.

To adapt our calibration algorithm to the more general multi-sprite approach, we explore how Eq. (12.23) is implemented in practice. Because our long-term motion estimator gives sprite-to-image transforms $\mathbf{H}_{i}$, we
write Eq. (12.23) as

$$
\begin{equation*}
\boldsymbol{\omega}^{(i)}=\left(\mathbf{H}_{i} \mathbf{H}_{r}^{-1}\right)^{-\top} \boldsymbol{\omega}^{(r)}\left(\mathbf{H}_{i} \mathbf{H}_{r}^{-1}\right)^{-1}=\mathbf{H}_{i}^{-\top} \mathbf{H}_{r}^{\top} \boldsymbol{\omega}^{(r)} \mathbf{H}_{r} \mathbf{H}_{i}^{-1} \tag{12.30}
\end{equation*}
$$

provided that both images $i$ and $r$ are assigned to the same sprite. If they are not on the same sprite, the inter-sprite transforms have to be used as demonstrated in Eq. (12.6) to span several sprites.

### 12.4 Non-linear camera calibration

In this section, we apply a bundle-adjustment algorithm [187] to increase the accuracy of the camera parameters that we obtained with the linear calibration algorithm. In the last section, we obtained the calibration in the linear case with an overdetermined equation system comprising constraints on the intrinsic parameters matrix. This equation system was solved in the least-squares sense. Hence, the quantity that is minimized is a physically meaningless algebraic error. This algebraic error is view-dependent and biased geometrically. The accuracy of the camera parameters can be increased by replacing this semantical weak error definition with a physically meaningful measure like Euclidean distance. The non-linear estimation algorithm resulting from it has the advantage that it allows to integrate additional constraints, which are difficult to express in the linear parameter estimation. For example, if the principal point is known, it is easy to integrate this constraint in the optimization.

The non-linear calibration uses also the high-accuracy motion parameters that we computed in the long-term motion estimation. Let us first consider the case in which we have only one sprite. For each view $i$, there is one set of motion parameters $\mathbf{H}_{i}$, mapping coordinates on the sprite to coordinates in image $i$. This transformation can be decomposed in a sequence of physically motivated transformations as

$$
\begin{equation*}
\mathbf{H}_{i}=\mathbf{K}_{i} \cdot \mathbf{R}_{i} \cdot \hat{\mathbf{K}}^{-1} \tag{12.31}
\end{equation*}
$$

If the sprite is constructed based on an input view $r$ as reference coordinate system, then $\hat{\mathbf{K}}$ will equal the intrinsic parameters $\mathbf{K}_{r}$ of view $r$. However, since our sprite construction algorithm can change the focal length for the sprite plane (see Section 6.5.4), we use an independent set of parameters for the sprite projection.

The purpose of the camera-calibration algorithm is to factorize the motion parameters $\mathbf{H}_{i}$ to the intrinsic parameters $\mathbf{K}_{i}$ and the rotations $\mathbf{R}_{i}$. Usually, it will not be possible to find an exact solution, so that a solution minimizing some error function is desired. In our algorithm, we use
the backprojection error of the four image corners on the sprite plane. To compute this, we project the image corners onto the sprite plane using the estimated transform parameters of the global motion estimation, and also using the current estimate of the physically camera parameters. The Euclidean distance between both defines the backprojection error.

More precisely, a pixel $\mathbf{p}$ on view $i$ is projected to the sprite position with the measured motion parameters $\mathbf{H}_{i}$ according to

$$
\begin{equation*}
\breve{\mathbf{p}}=\mathbf{H}_{i}^{-1} \mathbf{p} . \tag{12.32}
\end{equation*}
$$

Similarly, we project the same pixel onto the sprite position using the current estimate of the physical camera parameters,

$$
\begin{equation*}
\mathbf{p}^{\prime}=\hat{\mathbf{K}} \mathbf{R}_{i} \mathbf{K}_{i}^{-1} \mathbf{p} \tag{12.33}
\end{equation*}
$$

With $d(\cdot, \cdot)$ denoting Euclidean distance, we define the error of position $\mathbf{p}$ in view $i$ as the squared distance between both parameterizations (Fig. 12.6),

$$
\begin{equation*}
e_{i}(\mathbf{p})=d\left(\breve{\mathbf{p}}, \mathbf{p}^{\prime}\right)^{2} . \tag{12.34}
\end{equation*}
$$

The total error $E$ for all parameters is then simply the sum over all views $i$ and corner positions $n$, thus

$$
\begin{equation*}
E=\sum_{i} \sum_{n=1}^{4} d\left(\mathbf{H}_{i}^{-1} \mathbf{p}^{(n)}, \hat{\mathbf{K}} \mathbf{R}_{i} \mathbf{K}_{i}^{-1} \mathbf{p}^{(n)}\right)=d\left(\breve{\mathbf{p}}_{i}^{(n)}, \mathbf{p}_{i}^{\prime(n)}\right), \tag{12.35}
\end{equation*}
$$

where $\mathbf{p}^{(n)}$ denotes the $n$-th corner position of an input image.

### 12.4.1 Parameterization

The parameters to be estimated are the intrinsic camera parameters and the camera rotation. The intrinsic parameters for a camera comprise the variable focal length $f_{i}$, and the principal point $o_{x}, o_{y}$ which is assumed unknown but constant during the observed sequence ${ }^{3}$. Additionally, we include parameters for the projection into the sprite coordinate system, comprising the focal length $\hat{f}$ and the principal point $\hat{o}_{x}, \hat{o}_{y}$ in the sprite.

For the subsequent optimization, we need a suitable parameterization of the camera rotation. A camera rotation has three free parameters, but a rotation matrix as used previously has nine parameters. Compared to the three free parameters, this is an unnecessary over-parameterization. An

[^21]

Figure 12.6: The corners of an input image are projected into the sprite coordinate system, once with the global motion parameters and once with the physical camera parameters. The Euclidean distance between the respective corner positions define the error function for the physical camera parameters.
alternative parameterization with Euler angles has only three parameters, but it shows singularities (see Section 2.4.2) that lead to problems in the optimization [91]. It has been shown [164] that better numerical stability (see Section 2.4.2) can be obtained with a parameterization using quaternions. A quaternion $\mathbf{q}$ can be associated with a rotation, when $\|\mathbf{q}\|=1$ (see Section 2.4.2). We parameterize rotations using the four components of the quaternion, without imposing the unit norm constraint. For all transform calculations, the quaternion parameters are converted to a rotation matrix. In this conversion, the quaternion parameters are normalized to unit norm. Effectively, this means that we have one additional degree of freedom in the optimization that has no effect on the error.

All camera parameters are stacked into a parameter vector $\mathbf{x}$. In particular, each image has a quaternion for the rotation and the focal length parameter. Since the sprite plane serves as the reference, no rotation parameters are estimated for it. Hence, we only estimate the focal length and the principal point of the sprite plane. The parameter vector $\mathbf{x}$ becomes

$$
\begin{equation*}
\mathbf{x}=(\underbrace{\mathbf{q}_{1}, f_{1}, \mathbf{q}_{2}, f_{2}, \ldots, \mathbf{q}_{N}, f_{N}, o_{x}, o_{y}}_{\text {image view parameters }}, \underbrace{\hat{f}, \hat{o}_{x}, \hat{o}_{y}}_{\text {sprite param }})^{\top} . \tag{12.36}
\end{equation*}
$$

With the current vector of physical camera parameters $\mathbf{x}$, we can compute the projection of the four corners of each input image onto the sprite. Let us collect the coordinates of these projections in a data vector

$$
\begin{equation*}
\mathbf{y}=(\underbrace{\left(\mathbf{p}_{1}^{\prime(1)}, \underline{\mathbf{p}}_{1}^{\prime(2)}, \underline{\mathbf{p}}_{1}^{\prime(3)}, \underline{\underline{\prime}}_{1}^{\prime(4)}\right.}_{\text {first view }}, \ldots, \underbrace{\mathbf{p}_{N}^{\prime(1)}, \underline{\mathbf{p}}_{N}^{\prime(2)}, \underline{p}_{N}^{\prime(3)}, \underline{\mathbf{p}}_{N}^{(4)}}_{\text {view } N})^{\top}, \tag{12.37}
\end{equation*}
$$

where $\underline{\mathbf{p}}_{i}^{(n)}$ denotes the inhomogeneous coordinates of the projection of the $n^{\text {th }}$ image corner in view $i$. We can also generate a measurement data vector $\breve{\mathbf{y}}$ of similar layout, for which we obtain the projections of the image corners from the motion parameters $\mathbf{H}_{i}$ :

$$
\begin{equation*}
\breve{\mathbf{y}}=(\underbrace{\breve{\mathbf{p}}_{1}^{(1)}, \breve{\mathbf{p}}_{1}^{(2)}, \breve{\mathbf{p}}_{1}^{(3)}, \breve{\mathbf{p}}_{1}^{(4)}}_{\text {first view }}, \cdots, \underbrace{\breve{\mathbf{p}}_{N}^{(1)}, \breve{\mathbf{p}}_{N}^{(2)}, \breve{\mathbf{p}}_{N}^{(3)}, \breve{\mathbf{p}}_{N}^{(4)}}_{\text {view } N})^{\top}, \tag{12.38}
\end{equation*}
$$

These three vectors (parameters $\mathbf{x}$, measured point positions $\mathbf{y}$, and point positions $\breve{\mathbf{y}}$ for the parameter estimate) will be crucial in the optimization process. In that process, we will optimize the parameter vector $\mathbf{x}$ such that the sum of squared Euclidean distances between the projected corner coordinates is minimized. Hence, computing

$$
\begin{equation*}
\min _{\mathbf{x}} \sum_{i=1}^{N} \sum_{n=1}^{4} d\left(\mathbf{p}_{i}^{\prime(n)}, \breve{\mathbf{p}}_{i}^{(n)}\right)^{2}=\min _{\mathbf{x}}\|\breve{\mathbf{y}}-\mathbf{y}\|^{2} \tag{12.39}
\end{equation*}
$$

provides the solution vector $\mathbf{x}$ with the camera-calibration parameters.

### 12.4.2 Generalizing to multi-sprites

For a possible extension to arbitrary camera rotations, like in the linear calibration algorithm, we have to generalize the algorithm to multiple sprites. At first glance, it seems that a solution could be to process each sprite individually and then concatenate the estimated transforms. However, this approach cannot work in cases where some sprites are generated only from a camera-zoom operation (for example, see Figure 6.17 or $6.22(\mathrm{~d})$ ). As we have already seen in the beginning of this chapter, the focal length can only be estimated if there is some camera rotation present in the sequence. Although it is not possible to estimate the focal lengths for the images of a sprite that only shows camera zoom, we can solve it when we use the information about the focal lengths from a neighboring sprite. Since the last and first images of every sprite are also projected to the neighboring sprite, and because the focal length of this image is obviously the same in both projections, we can use those images as the connection that helps in the parameter estimation for the sprite showing only the zoom operation. Similar cases also occur if the estimation in some sprites are difficult to obtain because the rotation angle is small, or when there are only a small number of frames in the sprite.

Hence, to provide an approach that works in the general case, we have to jointly estimate the parameters of all sprites simultaneously. To achieve this, we build a parameter vector with the following conventions.

- The image rotations are defined relative to the sprite to which they are assigned to. This makes the estimation for these images independent from the location of the other sprite planes.
- Rotations between sprites are parameterized as the rotation between two consecutive sprites and not to a global reference. This reduces the interdependency between parameters and leads to a faster convergence in the optimization.
- Backprojection costs are calculated between images and the sprite to which they are assigned, but additionally, the first and last frame of each frame is also projected onto the previous and successive sprites, respectively.

The parameters and costs that are involved in the computations are illustrated in Figures 12.7 and 12.8. All the image and sprite-calibration parameters are again stacked into an extended parameter vector $\mathbf{x}$ such that we construct a vector of the following layout:

$$
\begin{align*}
& \mathbf{x}=(\underbrace{\mathbf{q}_{1}, f_{1}, \mathbf{q}_{2}, f_{2}, \ldots, \mathbf{q}_{N}, f_{N}, o_{x}, o_{y}}_{\text {image-view parameters }} \\
& \underbrace{\hat{f}_{1}, \hat{o}_{x ; 1}, \hat{o}_{y ; 1}, \hat{\mathbf{q}}_{2}, \hat{f}_{2}, \hat{o}_{x ; 2}, \hat{o}_{y ; 2}, \ldots, \hat{\mathbf{q}}_{M}, \hat{f}_{M}, \hat{o}_{x ; M}, \hat{o}_{y ; M}}_{\text {sprite-view parameters }})^{\top} \tag{12.40}
\end{align*}
$$

Note that every sprite has its own estimate of the principal point, but that all input images are assumed to have to same principal point.

The measurement vectors $\mathbf{y}$ and $\breve{\mathbf{y}}$ are constructed similarly as before, except that the corners of the first and last image of every sprite appear twice in these vectors, since they are projected onto two different sprites (Fig. 12.8). Furthermore, because the points are projected onto different sprites, the coordinates in these vectors are not expressed in the same coordinate system, but each position in the vector is relative to the corresponding sprite coordinate system. This does not change the error definition, which remains $E=\|\mathbf{y}-\breve{\mathbf{y}}\|^{2}$.

### 12.4.3 Optimization algorithm

For the optimization, we apply the Levenberg-Marquardt (LM) algorithm [151]. This algorithm is a combination of a steepest gradient-descent algorithm and a Gauss-Newton method. It attempts to find the parameter vector $\mathbf{x}$, for which the data vector $\mathbf{y}$ is closest to the observed measurement vector $\breve{\mathbf{y}}$ in terms of minimizing $(\breve{\mathbf{y}}-\mathbf{y})^{\top}(\breve{\mathbf{y}}-\mathbf{y})$. Starting with a


Figure 12.7: The transformations involved in the non-linear camera calibration algorithm for multi-sprite sequences. In this example, both images are assigned to sprite 1.


Figure 12.8: The transformations and backprojection costs for a multisprite case. The thin arrows in the lower half indicate the transformations that are used in the optimization. Note that most image transforms are independent from each other. The sprites are connected with transforms between consecutive sprites. The upper half of the figure indicates the backprojections that are included in the error calculation. Note that the first and last image of each sprite is projected onto two sprites to provide a connection between these sprites.


Figure 12.9: Non-zero entries of the Jacobian matrix $\mathbf{J}$.
first estimate of the parameter vector $\mathbf{x}$, the LM algorithm iteratively updates the parameter vector, where a steepest gradient-descent is carried out while the current estimate is far from the optimum. When the optimum is approached, the algorithm operates more like a Gauss-Newton algorithm. In each step, the LM algorithm updates the parameter vector $\mathbf{x}$ with an update $\boldsymbol{\delta}_{\mathbf{x}}$ such that

$$
\begin{equation*}
\left(\mathbf{J}^{\top} \mathbf{J}+\lambda \mathbf{I}\right) \boldsymbol{\delta}_{\mathbf{x}}=-\mathbf{J}^{\top}(\breve{\mathbf{y}}-\mathbf{y}) \tag{12.41}
\end{equation*}
$$

where $\mathbf{J}$ is the Jacobian matrix $\partial \mathbf{y} / \partial \mathbf{x}$. The parameter $\lambda$ is controlled by the LM algorithm to switch between steepest-descent (large $\lambda$ ) and GaussNewton $(\lambda=0)$ behaviour.

Since the camera parameters for a view $i$ only have influence on the position of the four corner points of that view, the Jacobian matrix $\mathbf{J}$ is very sparse (see Fig. 12.9). Therefore, an optimized implementation [187, 117] exploiting this spare matrix structure can be used to solve Eq. (12.41).

The non-linear optimization algorithm is initialized with the result of the linear estimation. This leads to a fast convergence in only a few iterations. We have observed that the non-linear optimization also converges reliably when it is initialized with zero rotations and a common value for the focal lengths and principal points. However, this requires more iterations and leads to a higher total computational complexity.

### 12.4.4 Recovering rotation angles

For the final algorithm output, we desire to express the output in angles of an Euler rotation sequence. This requires that we first express all rotations
relative to a common reference view. Second, we have to convert the internal quaternion representaton of rotations to the Euler-angle representation.

After selecting a reference view, the rotations to every other view can be determined easily by a concatenation of rotations. This can be done directly with the quaternion representation since the multiplicaton of two quaternions $\mathbf{q}_{b} \mathbf{q}_{a}$ gives the joint rotation, composed of first $\mathbf{q}_{a}$ and then $\mathbf{q}_{b}$.

Using the technique described in Section 2.4.2, it is possible to obtain Euler rotation angles from a rotation matrix. Instead of first converting the representation to a rotation matrix using Eq. (2.22) and then extracting the angles, the steps can be combined and we compute the angles directly from the quaternion representation as

$$
\begin{gather*}
\sin \alpha=2 q_{w} q_{x}-2 q_{y} q_{z},  \tag{12.42}\\
\tan \beta=\frac{-2 q_{x} q_{z}-2 q_{w} q_{y}}{1-2 q_{x}^{2}-2 q_{y}^{2}}, \quad \tan \gamma=\frac{2 q_{x} q_{y}+2 q_{w} q_{z}}{1-2 q_{x}^{2}-2 q_{y}^{2}} .
\end{gather*}
$$

Note that these angles are for the rotation sequence $\mathbf{R}_{\mathbf{y}}(\beta) \mathbf{R}_{\mathbf{x}}(\alpha) \mathbf{R}_{\mathbf{z}}(\gamma)$. The angles are different for other rotation sequences.

### 12.5 Experimental results

We have applied our algorithm to a number of sequences with varying complexity of camera motion. For each of the sequences, the camera rotations and the focal lengths have been extracted. The rotations are further factorized to the Euler angles.

Let us first consider the roma sequence, which consists of a pure horizontal camera pan. Figure 12.10 shows the extracted rotation angles. Since the rotation angle is limited, no multi-sprite partitioning was required. The figure depicts the results of both the linear calibration step and the nonlinear refinement. The estimated focal length was almost constant throughout the sequence, with a measured focal length of 593 pixels for the linear algorithm and 617 pixels for the non-linear optimization. The rotation angles are very close for both estimation algorithms. The difference is mainly a scaling factor that is due to the different focal length estimate.

For visualization of the results, we have designed an OpenGL-based program that places the input frames of the sequence at their virtual 3-D position, using the estimated camera calibration parameters. Figure 12.11 depicts two views onto the 3-D positioned images of the roma sequence. It is visible that the input frames align nicely into a panoramic view, which shows that the estimation accuracy is good.

In Figures 12.12 and 12.13, we have repeated the same experiment for the well-known stefan test-sequence. Since the stefan sequence comprises


Figure 12.10: Calibration result for the roma sequence, which is a pure horizontal pan.
a wide field of view, a multi-sprite partitioning is required. If we compare the linear estimation with the non-linear estimation, we can observe the same behaviour as for the roma sequence. The estimates of the rotation angles are similar except for a scaling factor, which again is a result of the differing focal length estimates.

Finally, we have applied the calibration algorithm to sequences with very complex camera motion. We show here the results for the rail and the nature-2 sequence, comprising three and seven sprites, respectively. The corresponding estimated camera-rotation parameters are depicted in Figure 12.16 and Figure 12.14. The provided results do correspond smoothly with the camera motion in the scene.

Experiments with several other sequences reveal that the accuracy of the focal length estimate is numerically more sensitive than the rotation angles. The linear algorithm alone already succeeds in computing a good estimate of the angles. However, since the amplitude of the angles is connected to the focal length, the amplitude of the angles can slightly differ in the result of the linear algorithm.

To evaluate the robustness of convergence for the non-linear optimization algorithm, we have made the experiment to omit the initialization of the parameters based on the result of the linear estimation algorithm. Instead, we started with zero rotation angles and the principal points at the


Figure 12.11: Images from the roma sequence, placed at their virtual image planes. The cube indicates the camera position. It is aligned to the camera viewing direction of the first frame, which is at the left side of the pan.
image centers. The focal length has been set to the length of the image diagonal. Even with this simple initialization, the non-linear optimization converged reliably to the same solution for all our test sequences.

So far, the accuracy of the obtained parameters could only be evaluated by visual inspection. We have used our 3-D visualization to display a virtual view from the camera position. At this position, all input images should fit together without seams between the images. Since this is achieved, we conjecture that the estimated parameters have a high accuracy. Evaluation of the absolute estimation accuracy requires the ground-truth value for the camera pose, which we do not have available yet.

### 12.6 Conclusions

This chapter has described a new algorithm to estimate camera calibration parameters from a video sequence. The output parameters are expressed in the physically meaningful units of camera rotation-angles and the varying focal length.

The difference to the previous algorithm is that it is not required to have the input sequence itself as input, since our algorithm operats directly on the projective motion parameters as used in the MPEG-4 sprite coding or in the MPEG-7 parametric-motion descriptor. Hence, the algorithm can be used e.g. to transform the parametric-motion descriptors of MPEG-7 to the physical camera-motion descriptors. Moreover, as a potential application, the algorithm can be used in an MPEG-4 decoder to derive the camera motion from the camera-motion parameters and use these e.g. to add virtual 3-D objects into the scene.

Our algorithm consists of two steps, where the first part determines a first estimate of the camera parameters. The accuracy of this estimate is improved in an optional second step. Because the calibration algorithm is combined with the multi-sprite motion-estimation approach, it is possible to carry out the estimation for general camera motion without limitation on the observable field-of-view.

We have observed that especially the camera rotation-angles obtained in the first step are already closely following the camera motion. The relative change of focal length is also reproduced accurately, but the absolute value can differ from the results obtained with the refinement step.

Hence, for applications that do not require absolute values for the camera parameters (like video-content analysis), applying only the first step is already sufficient. The second step is only required if the application depends on an accurate estimate of the focal length, such as, e.g., augmentedreality applications.


Figure 12.12: Camera-calibration results for the stefan sequence. The dotted vertical lines denote the multi-sprite boundaries. Note that the last sprite contains only a few frames and shows no camera rotation. Even in this case, the focal length of all frames could be estimated successfully.


Figure 12.13: Illustration of the virtual image planes for the stefan sequence (every $2^{\text {nd }}$ frame is shown). Different colors of the frames borders indicate the sprites to which the frames were assigned.


Figure 12.14: 3-D position of image planes for the nature-2 sequence (every $10^{\text {th }}$ frame is shown).

(a) Sprite 3.

(b) Sprite 2.

(c) Sprite 4 .

(d) Sprite 7 (last).

(e) Sprite 1 (first).

Figure 12.15: Background sprites for nature-2 sequence. Sprites 5 and 6 are not shown


Figure 12.16: Camera calibration for two sequences with complicated camera motion.

There are no such things as applied sciences, only applications of science.
(Louis Pasteur)

## Chapter <br> 13

## Camera Calibration for the Analysis of Sport Videos

In the previous chapters, the projective motion model was used to describe rotational camera motion in video sequences. This chapter discusses the application of the same motion model to establish a connection between motion in the video sequence and a fixed real-world coordinate system. Specifically, the discussion concentrates on camera calibration for the analysis of sport videos. For the in-depth analysis of sports like tennis or soccer, it is required to know the positions of the players on the tennis court or soccer field. To obtain these positions, it is necessary to establish the transformation between image coordinates and real-world coordinates. Since the ground of the sport field is flat, the geometric mapping can be described again with a projective transform. However, in the special case of sport analysis, a model of the playing field is employed to define the real-world reference coordinate system. This chapter describes an algorithm that localize a user-defined geometric model of a sport field in an input image in order to find the geometric transformation between both. The court model used in the algorithm can be switched to adapt the algorithm to a variety of different sports like e.g. tennis, soccer, badminton, or volleyball.

### 13.1 Introduction and previous work

Automatic analysis of sport videos is an interesting application of content analysis, since it enables new applications like automatic summarization of the highlight scenes of a long sport event, or virtual-view generation from arbitrary view-points. Moreover, it also enables to analyze a game with automatically deduced game statistics or to compute statistics about the performance or strategy of the players. This may help team coaches to determine strengths and weaknesses of players, or it can be used to entertain the viewer with additional information. For sports played on a court ${ }^{1}$, player positions are semantically important because the player movement or the static player configuration provides information about the current action [155, 16, 42, 198, 137]. Clearly, for the analysis of player positions, it is required to compute the positions of the players on the court in a realworld coordinate system, rather than their positions in the image. Hence, the real-world positions should be recovered from the image coordinates. For this purpose, the camera calibration parameters have to be estimated from the video input to determine the coordinate-system mapping.

Previous work on camera calibration for sport analysis is based on adhoc algorithms that were tailored to a specific kind of sport. Sudhir et al. [176] describe a calibration algorithm for tennis courts, using a simplified camera model that only considers the camera tilt angle, the camera distance from the court, and the focal length. Moreover, the algorithm requires that the lower part of the court is non-occluded and a starting position for the search has to be provided. A different approach for tennis-court calibration has been proposed by Calvo et al. [16]. They apply a Hough transform on the Sobel filter output to find the court lines. Assigning the lines to the court model is implemented with a set of heuristics. These impose tight restrictions on the sequences that can be processed. For example, it is assumed that the two lines at the net are the two lines with the most votes in the Hough transform. But in most tennis videos, the net line is not marked on the court at all. In $[102,195]$ a more robust detection of the court (for soccer videos) is described, but it requires a computationally complex initialization using an exhaustive search through the parameter space. Ekin and Tekalp [42] propose to use a Hough transform to indicate shots showing the goal area in soccer videos. However, no camera-calibration parameters are obtained. A camera-calibration algorithm for soccer is described by Yamada et al. [198]. The camera model includes two rotation axes, focal length, and the camera position. However, the camera position must be

[^22]

Figure 13.1: The geometric specification of a tennis court. Coordinates are specified in units of feet.
known as it is not estimated by the algorithm. This requires tedious manual measurements to be carried out prior to applying the algorithm. The search for the remaining three parameters is carried out using a search over the full parameter space. Especially for the first frame after a cut, when the playing-field location is unknown, this results in a high computational cost. Kim and Hong [102] also propose a calibration algorithm for soccer games based on a pan-tilt camera model. The interframe transformation is estimated by identifying corresponding lines between frames and using a nonlinear approach to determine the homography matrix that minimizes the Euclidean distance between line pairs. Tracking of the camera parameters is used to obtain a good initialization for the parameter optimization in subsequent frames.

In this chapter, we describe a more generic camera calibration algorithm that can be applied to every sport where the court contains a sufficient number of straight lines (tennis, football, volleyball, etc.). The configuration of court lines can be specified by the user and integrated into the algorithm as a court model (see Fig. 13.1). Based on this model, our algorithm computes the camera parameters for an eight-parameter perspective model. To obtain the transformation parameters, a set of features at well-known positions in the model have to be identified in the image. By establishing correspondences between the detected features and their position in the model, the transform parameters can be obtained.

Our algorithm was designed to be robust against cases where large parts of the court are occluded or out of view. Moreover, the algorithm is optimized for computational efficiency, such that calibration parameters can be determined in real-time. The next two sections give an overview of the algorithm and its processing steps. The successive sections then explain every processing step in more detail. Finally, example results are provided.


Figure 13.2: The transformation between the court ground-plane and the image plane is a homography.

### 13.2 Calibration-algorithm principle

The task of a camera-calibration system is to provide the geometric transformation that maps points in the image to real-world coordinates on the sport court. Since both the court and the displayed image are planar, this is a plane-to-plane transformation. Without loss of generality, we can place the court ground plane at $z=0$ and obtain the by geometric transformation

$$
\mathbf{p}^{\prime}=\mathbf{H} \mathbf{p}=\underbrace{\text { parameters }}_{\text {internal camera }}<\left(\begin{array}{ccc}
f & 0 & o_{x}  \tag{13.1}\\
0 & f & o_{y} \\
0 & 0 & 1
\end{array}\right) \quad \underbrace{\left(\begin{array}{llll}
r_{00} & r_{01} & r_{02} & t_{x} \\
r_{10} & r_{11} & r_{12} & t_{y} \\
r_{20} & r_{21} & r_{22} & t_{z}
\end{array}\right)}_{\substack{\text { camera rotation, } \\
\text { translation }}}\left(\begin{array}{c}
x \\
y \\
z=0 \\
1
\end{array}\right),
$$

which is a homography, represented by the $3 \times 3$ transformation matrix $\mathbf{H}$ (Fig. 13.2). This matrix transforms a point $\mathbf{p}=(x, y, w)^{\top}$ in real-world coordinates to image coordinates $\mathbf{p}^{\prime}=\left(x^{\prime}, y^{\prime}, w^{\prime}\right)^{\top}$.

Since $\mathbf{H}$ is scaling invariant, eight free parameters have to be determined. They can be calculated from four points whose positions are both known in the court model and in the image. Note that these four points need not be fixed, but should rather be selected on a case-by-case basis, as some points may be occluded in some views. Instead of using point features directly, we base our calibration algorithm on lines, because detecting the accurate position of a specific point on a court is more difficult than estimating the position of line segments. Moreover, the detection of lines is more robust, since they are hardly occluded completely.

The basic approach of the algorithm is to extract a number of straight lines from the input image, providing a set of court-line candidates. Using
a combinatorial search, line candidates are assigned to lines in the court model. For each assignment, the corresponding geometric transformation can be determined. This transformation is used to project the complete court model back to image coordinates. Each transformation is rated by measuring the match between the back-projected model lines and the court lines in the input image. The transformation with the best match is selected as the final solution.

### 13.3 Overview of the calibration system

The complete camera calibration system is depicted in Figure 13.3. It comprises the following four main algorithm steps.

1. Court-line pixel detection. This step identifies the pixels that belong to court lines. Since court lines are usually white or brightcolored, this step is essentially a white-pixel detector. However, white pixels can also appear on other objects like the players clothes. For this reason, additional constraints are imposed on the court-pixel candidates. White non-court pixels are sorted out with a cascade of filters with increasing complexity.
2. Line-parameter estimation. Starting with the detected white pixels, line parameters are extracted. For doing so, we apply a RANSACbased line detector, motivated by [28], which hypothesizes a line using two randomly selected points. If the hypothesis is verified, all points along the line are removed and the algorithm is repeated to extract the remaining dominant lines in the image. We also determine the extent of the line, to obtain line segments instead of infinite lines. Knowing the end points of the lines enables a faster model fitting as only two lines are required for the calibration instead of four.
3. Model fitting. After a set of lines has been extracted from the image, we need to know which line in the image corresponds to which line in the court model. It may also be the case that lines are detected other than those present in the model or that some of the lines were not detected. This assignment is obtained with a combinatorial optimization, in which different configurations are evaluated. We provide two combinatorial searches, using either a pair of line segments, or four infinite lines. The first algorithm using line segments is faster, but not applicable to all situations. The second search using infinite lines is more robust to occlusions, but it is slower. Consequently, it is only applied when the first search failed.


Figure 13.3: Flowchart of the court-tracking algorithm. At the first frame, the court location has to be initialized. For subsequent frames, we can use the previous approximate court location and adapt it to the new frame using a fast local search.
4. Tracking. When the initial position of the court is known, the computation in successive frames can be carried out more efficiently with a local search. We use a gradient-descent search to minimize the distance of the court lines to the white pixels in the image.

At the algorithm start and after shot boundaries, Steps 1-3 are carried out to find the initial location of the court in the first image. For the subsequent frames, only Steps 1 and 4 are applied, since the court position will be close to the old position. Because Steps 1 and 4 are computationally cheap, we achieve a high tracking speed.

An example intermediate step of the model fitting is depicted in Figure 13.4. In this example, five lines $\left(\mathbf{l}^{\prime}{ }_{1}, \ldots, \mathbf{l}^{\prime}{ }_{5}\right)$ have been detected in the input image, and for four of these lines, their corresponding lines in the court model have been identified ( $\mathbf{l}_{1}^{\prime} \leftrightarrow \mathbf{l}_{1}, \mathbf{l}_{2}^{\prime} \leftrightarrow \mathbf{l}_{2}, \mathbf{l}_{3}^{\prime} \leftrightarrow \mathbf{l}_{3}, \mathbf{l}_{4}^{\prime} \leftrightarrow \mathbf{l}_{4}$ ). Computing pairwise intersection points of the lines results in four independent feature-points in the image as well as the model. From these four point pairs, the homography $\mathbf{H}$ is computed with Eq. (3.2). Note that the intersection points themselves do not have to lie inside the image area, since their position can be computed from the line parameters. Moreover, if we represent all visible line segments in the image as infinite lines, we typi-


Figure 13.4: The pairwise line intersections define four points that are used for calibration. The intersection points of the lines $\mathbf{1}_{1}, \mathbf{l}_{3}$ and $\mathbf{l}_{2}, \mathbf{l}_{4}$ provide virtual points that are not directly observable. However, they can still be used for calibration.
cally obtain additional virtual line intersections that can also be used as calibration points.

The most computation time is spent in the combinatorial search through all possible geometric configurations in the model-fitting step. In order to reduce this computation time, we employ two fitting algorithms. The first, fast method searches for the correspondences of two line segments. With $N$ detected lines in the input and $M$ lines in the model, $O\left(N^{2} M^{2}\right)$ configurations have to be searched. The second, robust method searches for the correspondences of four lines. This algorithm is more robust to occlusions and partial court views, but requires $O\left(N^{4} M^{4}\right)$ configurations to be searched. The more complex robust fitting method is only applied if the fast fitting was not successful.

### 13.4 Court-line pixel detection

The processing of each frame starts with detecting the pixels of the court lines. In all cases that we observed, court lines have a white color. Unfortunately, court lines are usually not the only white objects in the images. Advertisment logos, parts of the stadium, the audience, or even the players themselves can have white-colored parts. Especially in tennis, white is the most common clothing color. If all white pixels would be classified as


Figure 13.5: (a) Schematic, magnified view of part of an input image containing a court line. Each square represents one pixel. The central pixel is only classified as court-line pixel if both pixels marked ' $H$ ' or both pixels marked ' $V$ ' are darker than the central pixel. In the shown case, only the 'V' pixels will be darker. (b) Corresponding white-pixel classification result. Note that most of the player pixels are not classified as court-line candidates even though the player is dressed in white.
court-line pixels, the subsequent line-detection algorithm would create too many line candidates, thereby making the fitting of the court model time consuming and unreliable. Therefore, additional criteria are needed to further constrain the set of court-line pixels. Simultaneously, a fast court-pixel detection algorithm is desired. In order to achieve accurate court-pixel detection with high computational speed, we apply several filters of increasing complexity, such that the more complex filters are only applied to the remaining court-pixel candidates. We apply filters that check the following assumptions about court lines:

- court lines are bright (white or yellow),
- lines have a limited width, and
- court-lines are in non-textured areas.


### 13.4.1 Filter 1: luminance threshold

The first filter is a simple luminance threshold that classifies each pixel $(x, y)$ as court-line pixel candidate $(l(x, y)=1)$ or not $(l(x, y)=0)$ based on its luminance value $I(x, y)$. We set $l(x, y)=1$ iff $I(x, y) \geq \sigma_{l}$. In the actual implementation, we store the pixels for which $l(x, y)=1$ in a set,
such that the successive filters have direct access to the white pixels and the image needs only be scanned once. The luminance threshold was set to $\sigma_{l}=128$ and experiments showed that this setting applies well to a large variety of sequences with different illumination conditions and court colors.

### 13.4.2 Filter 2: non-flat regions

Large regions of uniform, bright color passing the first filter impose difficulties, since the successive line estimator will also detect lines in these large, uniform regions. However, we desire that the court-line pixel detector only selects those pixels that are part of the court lines. Based on the assumption that court lines are typically not wider than $\tau$ pixels ( $\tau=8$ in our set-up), our second filter checks if the brightness levels at a distance of $\tau$ pixels from the four sides of the candidate pixel are considerably darker than the candidate pixel. Only if they are, the candidate pixel is classified as a white pixel (Figure 13.5(a)). This filter reduces the set of candidates $l(x, y)$ to a new set $l^{\prime}(x, y)$ using

$$
l^{\prime}(x, y)= \begin{cases}1 & l(x, y)=1 \wedge  \tag{13.2}\\ & I(x, y)-I(x-\tau, y)>\sigma_{d} \wedge \\ & I(x, y)-I(x+\tau, y)>\sigma_{d} \\ 1 & l(x, y)=1 \wedge \\ & I(x, y)-I(x, y-\tau)>\sigma_{d} \wedge \\ & I(x, y)-I(x, y+\tau)>\sigma_{d} \\ 0 & \text { else. }\end{cases}
$$

In this equation, the first line corresponds to the test if darker pixels can be found at some horizontal distance, assuming that the court line is mostly vertical. The second line performs the analogous test in the vertical direction, assuming that the court line is almost horizontal. The minimum brightness difference was set to $\sigma_{d}=20$. Figure $13.5(\mathrm{~b})$ shows a sample result of the court-pixel detector after the second filter. It is clearly visible that the additional constraint prevents that the player pixels are also marked as court-line candidates despite the player's white clothes.

### 13.4.3 Filter 3: linear structure

A further problem can occur with small white letters in logos, white areas in the stadium, or spectators dressed with white clothes, since these pixels in fine textured areas may still pass the previous filters. To prevent that these areas cause too many false detections in the line-extraction step, we exclude


Figure 13.6: Detected white pixels with and without applying the linestructure constraint. (a) Many false positives are found in the textured areas. (b) The number of false detections is reduced by adding the constraint that court-line pixel candidates are only allowed if the pixel neighborhood shows a linear structure.
white pixels that are in textured regions. Textured regions are recognized by observing the two eigenvalues of the structure tensor $\mathbf{J}$, computed over a small window of size $2 b+1$ around each candidate pixel $\left(p_{x}, p_{y}\right)$. The structure tensor is defined as (see [96])

$$
\begin{equation*}
\mathbf{J}=\sum_{x=p_{x}-b}^{p_{x}+b} \sum_{y=p_{y}-b}^{p_{y}+b} \nabla I(x, y) \cdot(\nabla I(x, y))^{T} . \tag{13.3}
\end{equation*}
$$

If both eigenvalues of the matrix $\mathbf{J}$, called $\lambda_{1}, \lambda_{2}\left(\lambda_{1} \geq \lambda_{2}\right)$ are large, it indicates a two-dimensional texture area. If one eigenvalue is large and the other is small, image gradients are oriented along a common axis. On the straight court lines, the latter case will apply, which can be exploited to define an additional rule that removes white pixels if $\lambda_{1}>4 \lambda_{2}$. Results of the proposed structure constraint can be seen in Fig. 13.6.

This filter is significantly more computationally expensive than the previous filters. For this reason, we only apply this filter at the initialization stage in the first frame. In the subsequent tracking steps, the white-texture pixels do not affect the solution, since they are usually too far away from the court area to distract the solution from the (correct) local optimum. In the flow-graph of Figure 13.3, the court-line pixel detector without Filter 3 is indicated as abbreviated line-pixel detector.


Figure 13.7: Visualization of Hough-transform based court-line candidate extraction. (a) Thick or slightly bent lines in the input lead to a bundle of possible lines (e.g., observe that the bent net produces many candidates). (b) Improved result after leastsquares fitting and duplicate removal.

### 13.5 Line-parameter estimation

### 13.5.1 Line detection with the Hough transform

Once the set of court-line pixels is obtained, we extract parametric equations for the lines. In our first implementation [65], we have used a Hough transformation to detect lines and determine the line parameters. However, we observed that the Hough transform has the disadvantage that thick lines in the input image usually result in a bundle of detected lines, which all lie closely together (Fig. 13.7(a)). Another disadvantage of the Hough transform is that the accuracy of the determined line parameters is depending on the quantization accuracy of the accumulator matrix. This problem cannot be easily solved by decreasing the accumulator matrix quantization step-size, since this would spread the inexact parameter samples for an input line over a larger area in the accumulator matrix, thereby making the detection unreliable.

We have reduced the above-mentioned problems by computing a leastsquares fit to the pixels close to the detected line. Furthermore, lines whose parameters are almost equal are considered duplicates of each other and only one of them is kept. With these modifications, lines could be detected robustly (see Fig. 13.7(b)).


Figure 13.8: Extraction of lines with RANSAC. Two court-line pixels are randomly selected and a line through the two points is hypothesized. The support (number of white pixels along the line) is measured and the hypothesis with largest support is selected.

### 13.5.2 Line detection with RANSAC

Even though the line detection based on the Hough transform with the postprocessing steps could achieve robust detection results, the computation speed of the Hough transform was not sufficient for a real-time implementation. For this reason, we replaced the line detection with a RANSAC-like approach.

RANSAC is a randomized algorithm that hypothesizes a set of model parameters and evaluates the quality of the parameters. After several hypotheses are evaluated, the best one is chosen (see Fig. 13.8). More specifically, we hypothesize a line by randomly selecting two court-line pixels $\mathbf{p}=\left(p_{x}, p_{y}\right), \mathbf{q}=\left(q_{x}, q_{y}\right)$. From these two points, we determine the parameters $a, b$ for the line model

$$
\begin{cases}y=a \cdot x+b & \text { if }\left|p_{x}-q_{x}\right| \geq\left|p_{y}-q_{y}\right|,  \tag{13.4}\\ x=a \cdot y+b & \text { if }\left|p_{x}-q_{x}\right|<\left|p_{y}-q_{y}\right| .\end{cases}
$$

The advantage of this line model is that it does not degenerate for vertical lines (infinite slope) and that it enables a fast approximation to calculate the distance of a point to the line. We define the approximate distance $\tilde{d}\left(\mathbf{g}, x^{\prime}, y^{\prime}\right)$ between a point $\left(x^{\prime}, y^{\prime}\right)$ and the line $\mathbf{g}$ as

$$
\tilde{d}\left(\mathbf{g}, x^{\prime}, y^{\prime}\right)= \begin{cases}\left|a \cdot x^{\prime}+b-y^{\prime}\right| & \text { if }\left|p_{x}-q_{x}\right| \geq\left|p_{y}-q_{y}\right|,  \tag{13.5}\\ \left|a \cdot y^{\prime}+b-x^{\prime}\right| & \text { if }\left|p_{x}-q_{x}\right|<\left|p_{y}-q_{y}\right| .\end{cases}
$$

For each line hypothesis, we compute a score $s(\mathbf{g})$ by

$$
\begin{equation*}
s(\mathbf{g})=\sum_{\left(x^{\prime}, y^{\prime}\right) \in \mathcal{P}} \max \left(\tau-\tilde{d}\left(\mathbf{g}, x^{\prime}, y^{\prime}\right), 0\right) \tag{13.6}
\end{equation*}
$$

where $\mathcal{P}$ is the set of court-line pixels and $\tau$ is the line width from Section 13.4.2. This score effectively computes the support of a line hypothesis as the number of white pixels close to the line, weighted with their distance to the line. The score and the line parameters are stored and the process is repeated until about 25 hypotheses are generated randomly. At the end, the hypothesis with the highest score is selected.

The described process detects the most dominant line in the data-set. Subsequently, the start and end position of the line segment are determined as described in the next section and the line parameters are further refined with a least-squares approximation to the court-line pixels in the vicinity. Finally, the court-line pixels along the line segment are removed from the data-set. The complete line-detection process is repeated several times until no more relevant lines can be found.

This line-detection algorithm operates at about 5 ms per frame on CIF resolution, while the original Hough-transform based algorithm required about 180 ms (both on a 2.8 GHz Pentium-IV). Even though this comparison may not be completely fair as the implementation of the Houghtransformation was less optimized for speed, we do not expect that this fast execution time can be obtained with the Hough-transform algorithm.

### 13.5.3 Line-segment boundary detection

Up to now, we have obtained the line parameters, but it is not yet known where the line segment starts and ends. Since this is valuable information for the subsequent model-fitting process, we also compute the line segment boundaries.

Scanning along the detected line, we obtain a sequence of $L$ pixels $p_{i}$ which are either white court-line pixels ( $p_{i}=1$ ) or black non-court-line pixels $\left(p_{i}=0\right)$. Because of classification errors and occlusions, the data is contaminated with noisy data. Assuming that the line segment starts at position start and ends at position end, we define the number of errors as the number of black pixels in the range start-end plus the number of white pixels outside of the range (Fig. 13.9). Using this error definition, we place the line-segment boundaries such that the error

$$
\begin{equation*}
E=\sum_{i<\text { start }} p_{i}+\sum_{i>\text { end }} p_{i}+\sum_{s t a r t \leq i \leq \text { end }}\left(1-p_{i}\right) \tag{13.7}
\end{equation*}
$$

is minimized. This optimization has a linear time complexity and can be carried out with the following algorithm.


Figure 13.9: Detection of line-segment boundaries. At the marked positions, classification errors occur. The boundaries start and end are placed to minimize the errors.

Let us first assume that the interval [start; end] is given, and that the end position is increased by one. If the pixel at the new position is white, the total error $E$ is decreased by one, since this pixel is now part of the white line segment. On the other hand, if the pixel is black, the total error $E$ is increased, since a black pixel is added to the white line segment. We can cumulate this change of error by setting

$$
c_{0}=0 \quad \text { and } \quad c_{i}=c_{i-1}+ \begin{cases}-1 & \text { if } p_{i}=1  \tag{13.8}\\ +1 & \text { if } p_{i}=0\end{cases}
$$

For a fixed start position, the optimal placement of the end position is where $c_{i}$ is the minimum with $i \geq$ start. This minimum position can be obtained in constant time by using an array $m_{i}$ with indices to the minimum $c_{k}$ for which $k \geq i$. This array can be filled in a single pass over $c_{i}$.

Since we can now compute the optimal end position for a given start position, we only have to search for the best start position. Clearly, the best start position must be at a first white pixel after a black pixel. Consequently, we scan $p_{i}$ for black-to-white transitions and for each of them, we lookup the end position in $m_{\text {start }}$. For this range, we compute $E$. Note that $E$ can also be computed in constant time by using an array storing the cumulative number of white pixels $w_{i}=\left|\left\{k \leq i \mid p_{k}=1\right\}\right|$ by

$$
\begin{align*}
E= & \underbrace{w_{\text {start }-1}}_{\text {white pixels before segment }}+\underbrace{w_{L}-w_{\text {end }}}_{\text {white pixels after segment }}+  \tag{13.9}\\
& \underbrace{\text { end }+1-\text { start }-\left(w_{\text {end }}-w_{\text {start }-1}\right)}_{\text {black pixels in segment }}
\end{align*}
$$



Figure 13.10: Example detection of lines and computation of line segment boundaries.

The positions for which we obtain the smallest $E$ are the desired linesegment boundaries. We denote the set of all detected line segments as $\mathcal{S}=\left\{\left(\mathbf{p}_{i}^{\prime}, \mathbf{q}_{i}^{\prime}\right)\right\}$. An example result for a tennis input picture is shown in Figure 13.10.

### 13.6 Court-model fitting

A court model consists of the lines that are drawn onto the ground to define the playfield geometry. The lines are defined in the model coordinate system, which can be arbitrarily defined. Remember that the result of our calibration algorithm will define a mapping between the image coordinate system and the model coordinate system, allowing to express player positions in model (which are usually real-world) coordinates. When using the mapping in the opposite direction, we can also mark interesting positions in the image by specifying their real-world coordinates.

The model-fitting step determines correspondences between the detected lines and the lines in the court model. Once these correspondences are known, the homography between real-world coordinates and the image coordinates can be computed. Searching for the best model requires a combinatorial search that can be computationally complex. Hence, we first try a fast fitting approach that works in most cases, but that is not robust for cases with large occlusions. If the fast algorithm fails, we determine the model location with a more robust, but also more complex approach. The following discusses these two model-fitting algorithms and a set of tests


Figure 13.11: Fast model fitting. Two corresponding line segments are identified. The transformation parameters are calculated from the four end points.
that are used to early reject court-model positions that cannot be correct.

### 13.6.1 Fast fitting method

In the fast fitting method, we find the transformation parameters by identifying two pairs of corresponding line segments between the image and the model (Fig. 13.11). To find the best transform, we iterate through all pairs of line segments in the image and in the model. Configurations with three collinear points are not considered, since transformation parameters cannot be determined from these configurations. For each configuration of lines, we have two end points for each of the two line segments in both the image and the model. Using these four pairs of points $\left(\mathbf{p}_{i} \leftrightarrow \mathbf{p}_{i}^{\prime}\right)$, we can determine the homography $\mathbf{H}$ with Eq. (3.2).

For each parameter matrix $\mathbf{H}$, we first apply some quick tests to reject impossible configurations with little computational effort (see Section 13.6.3). If the homography passes these tests, we compute the complete model matching error $E_{f}$ as

$$
E_{f}=\sum_{(\mathbf{p}, \mathbf{q}) \in \mathcal{M}} \min (\underbrace{\left.d\left(\hat{\mathbf{p}}^{\prime}, \mathbf{H p}\right)+d\left(\hat{\mathbf{q}}^{\prime}, \mathbf{H} \mathbf{q}\right), e_{m}\right)}_{\text {fitting error for one segment }})
$$

where $\mathcal{M}$ is the collection of line-segments (defined by their two end points $\mathbf{p}, \mathbf{q}$ ) in the court model and ( $\left.\hat{\mathbf{p}}^{\prime}, \hat{\mathbf{q}}^{\prime}\right) \in \mathcal{S}$ is the closest line segment in the image (Fig. 13.12). The metric $d(\cdot, \cdot)$ denotes the Euclidean distance between the two points, and the error for a line segment is bounded by a maximum value $e_{m}$.

The transformation $\mathbf{H}$ that gives the minimum error $E_{f}$ is selected as the best transformation. This model-fitting algorithm works well if most of the court is visible in the image, as it is mostly the case for tennis broadcasts. For other sports like soccer or volleyball, only small parts


Figure 13.12: Fitting cost for the fast method. The court model is backprojected into the image coordinate system. For each model line segment, the nearest detected line-segment is determined (dashed) and the distance between the end points is added to the cost. The cost is limited to $e_{m}$, which effectively denotes the cost for a non-detected line.
of the court are visible at a time and the court lines are clipped at the image boundaries. Obviously, these clipped line segments cannot be used successfully to obtain transformation parameters. However, if parts of the line segment are occluded by a player, our algorithm for detecting the segment boundaries will often close these occluded ranges if they are short compared to the total segment length.

### 13.6.2 Robust fitting method

If the fast model-fitting method does not yield a good solution, we start a robust fitting algorithm that also works with large occlusions and in cases where only a small part of the court is visible. Instead of iterating through all configurations of two line segments, we iterate through configurations of four lines in the image as well as in the model (Fig. 13.13). Intersecting the lines gives four intersection points, and we can again compute the transformation from these four points. Note that this algorithm also works if the intersection point itself is outside the image or if it is occluded by a player. Instead of computing the intersection points, we can also compute $\mathbf{H}$ directly from the line parameters by using the technique of Appendix B simply with line parameters instead of point coordinates. According to Eq. (2.3), this results in a matrix $\mathbf{H}^{-\top}$. By simply swapping image and model-line parameters, we can furthermore remove the necessity to invert this matrix, since we obtain directly $\mathbf{H}^{\top}$.


Figure 13.13: Robust model fitting. Four lines are used to calculate the transformation parameters.

## Evaluation of the model support

Each set of camera parameters is rated by projecting the court model onto the source image and verifying that the model accurately covers the white pixels. The evaluation process transforms all line segments of the model to image coordinates according to the estimated homography matrix $\mathbf{H}$. With $\mathbf{p}_{\mathbf{i}}^{\prime}=\mathbf{H} \mathbf{p}_{\mathbf{i}}$, each parameter set is rated by computing the matching score
$\sum_{\text {all model line all }} \sum_{\text {pixels }} \begin{cases}1 & \text { if } l(x, y)=1, \\ -\frac{1}{2} & \text { if } l(x, y)=0, \\ 0 & \text { if }(x, y) \text { is outside of image. }\end{cases}$
segments $\mathbf{p}_{\mathbf{i}}, \mathbf{p}_{\mathbf{j}} \frac{(x, y)}{\mathbf{p}_{\mathbf{i}}^{\prime} \mathbf{p}_{\mathbf{j}}^{\prime}} \quad$ on
Each model line $\overline{\mathbf{p}_{\mathbf{i}} \mathbf{p}_{\mathbf{j}}}$ is transformed into the image coordinates $\mathbf{p}_{\mathbf{i}}^{\prime}, \mathbf{p}_{\mathbf{j}}^{\prime}$. This line segment is sampled at discrete positions along the line and the evaluation value is increased by one if the pixel is a white court-line candidate pixel, or decreased by 0.5 if it is not. Parts of the line segment that are not visible are not taken into account (Fig. 13.14). Non-matching pixels are only penalized with half weight since the detection of court-line candidate pixels is often disturbed by shadows or occlusions. After all calibration matrices have been evaluated, the matrix with the largest matching score is selected as the best calibration-parameter setting.

### 13.6.3 Fast calibration-parameter rejection test

The model-fitting step has to consider a large number of possible line configurations. Whereas the computation of the transformation matrix is fast, especially the evaluation of the fitting quality is computationally complex. For this reason, we use several tests that allow to early reject physically


Figure 13.14: Evaluation of model match. Each pixel that is covered by the proposed model location contributes +1 if the pixel is a courtline candidate (marked with bright color), -0.5 if it is not (dark color), and 0 if the pixel lies outside the visible image area.
impossible calibration parameters (i.e., wrong configurations) without carrying out the complex fitting evaluation.

## Court area

The first test checks the area of the court in the image. Based on the four corner points of the court, we determine the area of the court outline and reject the transformation if the court area is below one eighth of the image size.

## Bounding-box aspect ratio

Since the court is always on a horizontal ground plane and it is viewed with a non-tilted camera, the image of the court will always be perspectively squeezed in the vertical direction. To check this, the bounding-box around the court is computed in both the image and the model (for the model, this is a constant). If the bounding-box in the image is taller than in the model, the calibration parameters are rejected.

## Isotropic scaling

Before we begin to describe the isotropic scaling test, let us first make the observation that our homography matrix has eight degrees of freedom, but the real-world image formation process has only seven. These comprise
three for camera position, three for camera rotation, and one for focal length. The remaining degree can be attributed to non-isotropic scaling, which refers to unequal scaling in horizontal and vertical directions. If we consider the individual steps of the image formation process, we get
where $f$ denotes focal length. Non-isotropic scaling is impossible in the real world. Hence, $\beta$ should be 1 , and we can use this condition as a rejection rule. To determine $\beta$ from $\mathbf{H}$, we first compensate the camera principal point $\left(o_{x} o_{y}\right)$, which we assume to be at the image center by multiplying an appropriate matrix to the left side. Furthermore, we simplify the equations by exploiting the fact that we construct the court model on the $z^{\prime}=0$ plane. Consequently, we obtain

$$
\begin{align*}
\left(\begin{array}{ccc}
1 & 0 & -o_{x} \\
0 & 1 & -o_{y} \\
0 & 0 & 1
\end{array}\right) \mathbf{H}=\mathbf{H}^{\prime} & =\left(\begin{array}{lll}
f & 0 & 0 \\
0 & f & 0 \\
0 & 0 & 1
\end{array}\right)\left(\begin{array}{lll}
r_{00} & r_{01} & t_{x} \\
r_{10} & r_{11} & t_{y} \\
r_{20} & r_{21} & t_{z}
\end{array}\right)\left(\begin{array}{ccc}
1 & 0 & 0 \\
0 & \beta & 0 \\
0 & 0 & 1
\end{array}\right) \\
& =\left(\begin{array}{ccc}
f r_{00} & \beta f r_{01} & f t_{x} \\
f r_{10} & \beta f r_{11} & f t_{y} \\
r_{20} & \beta r_{21} & t_{z}
\end{array}\right) \tag{13.12}
\end{align*}
$$

Since the rotation matrix $\left\{r_{i j}\right\}$ is known to be orthonormal, we can deduce from the unit norm constraint (see Section 12.2.1) that

$$
\begin{align*}
r_{00}^{2}+r_{10}^{2}+r_{20}^{2} & =r_{01}^{2}+r_{11}^{2}+r_{21}^{2} \\
\frac{h_{00}^{\prime 2}}{f^{2}}+\frac{h_{10}^{\prime 2}}{f^{2}}+h_{20}^{\prime 2} & =\frac{h_{01}^{\prime 2}}{\beta^{2} f^{2}}+\frac{h_{11}^{\prime 2}}{\beta^{2} f^{2}}+\frac{h_{21}^{\prime 2}}{\beta^{2}} \tag{13.13}
\end{align*}
$$

and from the orthogonality constraint that

$$
\begin{align*}
r_{00} r_{01}+r_{10} r_{11}+r_{20} r 21 & =0 \\
\frac{h_{00}^{\prime} h_{01}^{\prime}}{\beta f^{2}}+\frac{h_{10}^{\prime} h_{11}^{\prime}}{\beta f^{2}}+\frac{h_{20}^{\prime} h_{21}^{\prime}}{\beta} & =0 \tag{13.14}
\end{align*}
$$



Figure 13.15: Predicting the camera parameters for frame $t+1$ based on the previously computed parameters for frames $t-1$ and $t$. The dotted lines indicate predicted values, whereas the solid lines are computed from actual input data.

Finally, we get

$$
\begin{equation*}
f^{2}=-\frac{h_{00}^{\prime} h_{01}^{\prime}+h_{10}^{\prime} h_{11}^{\prime}}{h_{20}^{\prime} h_{21}^{\prime}} ; \quad \beta^{2}=\frac{h_{01}^{\prime 2}+h_{11}^{\prime 2}+f^{2} h_{21}^{\prime 2}}{h_{00}^{\prime 2}+h_{10}^{2}+f^{2} h_{20}^{\prime 2}} . \tag{13.15}
\end{equation*}
$$

Because the estimated camera parameters are not perfectly accurate and since the calculation is numerically sensitive, the restriction on $\beta$ should not be set too tight. We only accept solutions that have $0.5<\beta<2$.

### 13.7 Model tracking

The previous calibration algorithm only has to be applied in the bootstrapping process when the first frame of a new shot is processed. For subsequent frames, we can assume that the change in camera motion is small. This enables the prediction of the camera parameters for the next frame. Since the prediction provides a good first estimate of the camera parameters, a local search can be applied to refine the camera parameters to match the current view.

Let $\mathbf{H}_{\mathbf{t}}$ be the camera parameters for frame $t$. If we know the camera parameters for frames $t$ and $t-1$, we can predict the camera parameters $\hat{\mathbf{H}}_{\mathbf{t}+\mathbf{1}}$ for $t+1$ by (see Fig. 13.15)

$$
\begin{equation*}
\hat{\mathbf{H}}_{\mathbf{t}+\mathbf{1}}=\mathbf{H}_{\mathbf{t}} \mathbf{H}_{\mathbf{t}-\mathbf{1}}^{-1} \mathbf{H}_{\mathbf{t}} . \tag{13.16}
\end{equation*}
$$

The principle of the parameter refinement is to minimize the distance of the back-projected court model to the court-line pixels in the input image. To this end, white court-line pixels are extracted just as described in


Figure 13.16: Pixel costs used during the tracking step. White pixels have zero cost and the cost increases with increasing distance from the white pixels. To obtain the current total cost, all pixels along the back-projected court line-segments are added.

Section 13.4. Since we start tracking with a good first estimate of the court location and only a narrow neighborhood is considered, the accuracy of the white-pixel detector can be decreased in favour of faster execution. We therefore disable the final texture filter (Section 13.4.2) during the tracking phase. Starting with the detected white court-line pixels, we generate a distance map $D(x, y)$, where each element stores the distance to the nearest white pixel. When using the Manhattan distance, this map can be computed efficiently. To further decrease computation cost, we only consider pixels with a distance of not more than $d_{w}$ pixels to the nearest white pixel. An example distance map is depicted in Figure 13.16.

The concept of the tracking algorithm is to place the court such that the distance between its line segments and the white court-line pixels is minimized. Similarly, we compute a cost by projecting the court model back into the image using the predicted motion parameters. Subsequently, the elements of $D(x, y)$ that are covered by the line segments of the court model are summed up. To refine the transformation parameters, we use a Quasi-Newton algorithm to minimize this cost function. This process converges reliably if the prediction error of the court position is less than $d_{w}$, i.e., if the predicted lines are within the valleys of the distance map $D(x, y)$. Also note that a simplified version of the white-pixel detector can be used in the tracking step, since erroneously detected court-line pixels that would have been removed by the texture filter (Section 13.4.3) have generally no influence on the optimization.

### 13.8 Experiments

We have tested the algorithm on 21 sequences that were recorded from regular DVB television broadcasts or that were recorded on VHS tape and digitized later. The video resolution was either CIF or PAL/SDTV, the average sequence length was about 30 minutes. Five of the sequences were soccer games, four were volleyball games, and the remaining twelve were tennis games on different court classes (grass, clay, carpet). All algorithm parameters were kept constant during all experiments, only the correct court model was preselected.

Figure 13.17 shows a tennis scene onto which the court model has been superimposed with the estimated calibration parameters. With these parameters, the input image can also be rectified to a real-world ground-plane view. Clearly, the rectification is only valid for positions on the ground plane. The position and height of objects above the ground (e.g., the flying ball) cannot be extracted from this view. However, the player positions are available, if their positions are measured at their feet, where they touch the ground.

Figures 13.18 shows two examples, where some particularly difficult scenes have been selected. Example 13.18(a) contains a very large shadow on the court that darkens the image so much that most court-line pixels in the shadow area cannot not be detected. Nevertheless, the court is detected successfully. In picture 13.18(b), a superimposed text occludes most of the court, such that calibration can only be carried out using the two leftmost lines. Thus, the calibration accuracy is not sufficient for the whole court, as can be seen in the lower right part. Because of the large occlusion and many white pixels in the text area, the camera-parameter refinement step cannot correct this small error.

A collection of various difficult scenes for calibration of tennis scenes are depicted in Figure 13.19. Note that there is a variation of different court colors and perspectives. The presented scenes show large occlusions to examine the limits for which the algorithm still provides correct results. For usual tennis broadcasts, the occlusions are smaller and the algorithm shows a very high robustness.

The presented algorithm not only works for tennis videos, but it can be adapted easily to other sports by simply exchanging the court model. In Figure 13.20, we have applied the same algorithm to soccer and volleyball sequences. All algorithm thresholds were kept constant and only the court model was exchanged. For the soccer video, only scenes showing the goalarea could be processed, since the middle part of the soccer field does not contain enough information to carry out a full camera calibration. However, in this case, camera calibration is generally only possible if further
constraints are imposed on the camera parameters (e.g., no change of focal length, or known camera position).

For our test set, the algorithm can find and track the courts very reliably if the minimum required amount of court lines (two horizontal and two vertical lines) are clearly visible in the image. In the most common camera angle that shows an overview of most of the court area, no false calibrations occurred with our test set. Even in difficult scenes with strong shadows or large occlusions, the calibration is correct for $>95 \%$ of the sequences. The most common mis-calibration is caused in tennis shots like Fig. 13.19(b), where the white line at the top of the net was mistakenly assigned to a court line. On a 2.8 GHz Pentium- 4 computer using CIF-resolution input videos, the computation time for the initialization step (first frame) was between 20 ms and 35 ms , depending on the complexity of the frame. Tracking the detected court through the sequence required $4-10 \mathrm{~ms}$ per frame.

### 13.9 Conclusions

In this chapter, a new, generic algorithm for camera calibration in sport videos has been presented. The algorithm can obtain all eight parameters of a perspective motion model without any user assistance. The geometric model of the court can be adapted to virtually any kind of sport. The adaptability to different kinds of sport using definable court models is a notable improvement of flexibility compared to previously proposed algorithms. Furthermore, it is the first algorithm that applies a combinatorial search to initialize the camera parameters for the first frame. Previous algorithms concentrated mainly on the tracking step and required a manual initialization, a computationally expensive exhaustive search through the parameter space [198], or they applied heuristics that cannot be applied in the general case $[16,176]$. In this context, it is also advantageous that the proposed algorithm works reliably without the need to tune any further algorithm parameters like the playfield color. Possible applications for the algorithm exist in systems for the automatic extraction of game statistics, detection of interesting scenes, or automatic game summarization.

All steps of the algorithm have been designed for computation efficiency, such that the algorithm requires only about 30 ms for the first frame and 6 ms during the court tracking. The algorithm is robust even in difficult scenes with large occlusions or poor lighting conditions, since it adaptively chooses the lines used for the calibration process. Interestingly, the algorithm also works with dashed lines instead of continuous lines. It should also be noted that although we apply the same motion model as in earlier chapters for rotational camera motion, this algorithm (including the
tracking step) also works for arbitrarily moving cameras. This is the case because we observe a planar scene, for which the motion can be described with homographies for any arbitrary camera motion (see Section 2.5.4).

Possible enhancements of the algorithm would be the inclusion of curved line segments into the court model, which would allow calibration in cases where not sufficient straight lines are visible (e.g., in the center of soccer fields).

An interesting extension of our algorithm has been proposed by Hayet et al. in [89]. Instead of using four line-correspondences, they propose to first estimate the vanishing points for the horizontal and the vertical lines. With the vanishing points known, only two corresponding points between the model and the image are required. The points in the image are obtained by intersecting the detected lines. Each intersection point is further classified into one of 17 different intersection classes, which enables a fast detection of corresponding intersections. The authors claim that with this modification, they could reduce the computation time for the initialization to 10 ms in typical cases and 50 ms in the most complex situations, computed on a 1.6 MHz Centrino processor.


Figure 13.17: A tennis scene with detected court (a) and the rectified groundplane view (b).

(a) Scene with a strong shadow.

(b) Large occlusion.

Figure 13.18: Two difficult tennis scenes due to shadows and graphical overlay.

(a)

(c)

(e)

(b)

(d)

(f)

Figure 13.19: Collection of various tennis scenes with difficult calibration.


Figure 13.20: Soccer and volleyball scenes at different views.

The important thing in science is not so much to obtain new facts as to discover new ways of thinking about them. (Sir William Bragg)


## Panoramic Video and Floor Plan Reconstruction

Previous chapters often used background-sprite images to create a panoramic overview of the scene environment. The generated background image was modeled as a single, large image on a plane. However, other representations of panoramic images are possible that have certain advantages for other applications. The most frequently-used model is the cylindrical panorama, which allows to capture a 360-degree horizontal view in a single image. This chapter describes the geometry of cylindrical panoramic images and presents various techniques for capturing panoramic images and videos. Since cylindrical images are a special kind of image with geometric distortions, their contents are not always easy to interpret. Therefore, different visualization techniques are explored providing images that are easier to understand. In particular, a new visualization technique is proposed that reconstructs the geometry of the room in which the panoramic image was recorded, and which uses this room reconstruction to show the panoramic image as texture maps on this virtual room. Finally, this concept is generalized to reconstruct a complete floor plan based on multiple panoramic images. Additionally to the aforementioned improved visualization, this enables new applications like the presentation of real estate with virtual tours through the appartment.


Figure 14.1: Background image models.

### 14.1 Introduction

### 14.1.1 From background sprites to panoramic images

Chapter 5 described that global-motion parameters can be used to combine several images captured from the same scene into a large scene overview image. A transformation model was used that is compatible with the MPEG-4 sprite coding tools to use these synthesized background images in an object-oriented encoding system. Chapter 6 showed that the projective motion model used in MPEG-4 does not support the synthetization of images covering a large viewing angle. Since the chosen motion model was limited by the possibilities of the MPEG-4 standard, we splitted the background image into several independent images, each covering part of the scene (Fig. 14.1(a)). However, by choosing a different transformation, it is indeed possible to combine images captured during a complete $360-$ degree pan into a larger panoramic image. One of the most frequently-used models for this is the cylindrical panorama. The principle is to project the surrounding scene onto a virtual cylinder surface around the camera (Fig. 14.1(b)). Unrolling this cylindrical surface gives a rectangular image that comprises the full 360-degree view.

Cylindrical panoramic images are used in a wide variety of applications. One of their advantages is that the extended field of view, compared to the limited view of normal images, fits better to the field of view of the human eye. Hence, panoramic images provide an ideal visualization of landscape-type images. Moreover, 360-degree images show the whole surrounding in only one picture. This is ideal for presentations of hotel rooms or real-estate, because it gives a complete impression of the environment.


Figure 14.2: The Diver video annotation software. Magnified views can be extracted from the panoramic video and these clips can be described with textual annotations.

Finally, since the panoramic image covers the complete environment, it is not necessary to select a suitable view while the image is captured. Instead, events can be recorded in a panoramic video and selections of the most interesting parts of the scene can be made later. This last aspect has been extensively studied in the Diver project [142] at the Stanford Center for Innovations in Learning (to which the author had the possibility to contribute). In this project, panoramic videos of classroom education scenes were recorded from the full event. Later, psychologists could analyze the teaching methods and reaction of the students. For this purpose, video clips can be annotated with comments, where a clip is not only a temporal selection out of the video material, but also a spatially restricted view into the full panoramic overview (Fig. 14.2).

### 14.1.2 Visualization of panoramic images

One important aspect of panoramic imaging is the presentation of the images to the viewer. Directly showing the texture of the unrolled cylinder surface results in a rectangular image, but includes geometric distortions. Moreover, since this image shows all directions around the camera at the same time, the panoramic image itself can be confusing to the viewer.

For this reason, panoramic images are often presented with an interactive panoramic image browser (PIB) application which shows a geometrically rectified sub-view of the scene, as if it were captured with a user-controlled virtual camera. The disadvantage of this representation is that it is not possible to offer a fast overview of the scene, and it is not possible to see the complete environment on a static medium like a paper copy. In this chapter, we propose a new visualization technique for panoramic images that is specialized for images captured inside rectangular rooms, which is an important special case that covers many application areas like hotel room advertising or recording of group meetings.

Our visualization is based on an algorithm to reconstruct the 3-D layout of the rectangular rooms from the panoramic image. Once the geometry of the room is known, a $3-\mathrm{D}$ model of the room walls can be synthesized and the wall textures can be added, using the image data from the panoramic image (Fig. 14.8). The proposed representation provides a flexible way to visualize the scene. On one hand, the virtual camera can be placed outside of the room, such that the viewer gets an overview of the full scene appearance and room layout. On the other hand, the virtual camera can also be placed at the position of the original camera. Interactively rotating this virtual camera provides views that equal the output of the PIB technique.

The room reconstruction requires a minimum of user assistance: the user only has to indicate the position of the four room corners in the panoramic image. First, the reconstruction algorithm converts the positions of the corners into the angle between these corners as observed from the camera position. Subsequently, the room shape and the camera position are determined from these angles, and the textured 3-D model is constructed automatically.

### 14.1.3 Floor plan reconstruction

Following the same reconstruction principle as for rectangular rooms, we can extend the reconstruction algorithm to support arbitrary room shapes or even complete floor plans, comprising several rooms. This floor plan reconstruction enables new applications additional to the aforementioned visualization, like the presentation of real estate, providing virtual tours through an appartment. Generally, for more complicated room shapes or a reconstruction comprising several rooms, a single panoramic image does not provide enough information for a reconstruction. For this reason, the extended algorithm allows the usage of multiple panoramic images for the reconstruction if required.

### 14.1.4 Chapter outline

This chapter first briefly introduces the geometry of cylindrical panoramic images and describes techniques to compose panoramic images from a collection of small images or to capture panoramic video with specialized cameras. Section 14.3 discusses different visualization techniques and Section 14.4 proposes a new algorithm for the estimation of the wall sizes of rectangular rooms from captured panoramic images. This algorithm is generalized in Section 14.5 to support the reconstruction of a collection of rooms with arbitrary room shapes from multiple panoramic images.

### 14.2 Capturing panoramic images and video

The most commonly-used model for panoramic images is the projection onto a cylindrical surface. The cylinder is centered at the camera location and aligned vertically, such that a horizontal camera pan corresponds to a rotation of the cylinder axis. To transform the planar image coordinates $(x, y)$ into cylinder coordinates $(\theta, h)$, we use the transformation

$$
\begin{equation*}
\tan \theta=x / f \quad \text { and } \quad h=\frac{y \cdot r}{\sqrt{f^{2}+x^{2}}} \tag{14.1}
\end{equation*}
$$

where $f$ is the focal length (the distance of the image plane to the optical center), and $r$ is the cylinder radius (Fig. 14.3). From these equations, it can be noticed that the cylindrical transformation depends on the focal length $f$ that was used to capture the image. Some digital cameras store the focal length with which an image was recorded in the EXIF metadata. If this is not available, the focal length should be estimated from the image data (see Section 12.2.1). The radius of the cylinder surface can be chosen arbitrarily, since its only effect is a scaling factor for the vertical axis $h$ of the panoramic image. For a practical implementation, we have to consider that images are usually stored with integer pixel positions. Hence, in practice, we set $r=f$ in order to obtain a vertical image resolution in the panoramic image that is approximately the same as in the input image. Geometrically, this means that the input image plane is a tangent plane to the cylinder. It touches the cylinder along $x=0$, and it follows from Eq. (14.1) that $h=y$ along this line. For the other values of $x$, it holds that $|h|<|y|$, meaning that there is some loss of resolution in the cylinder projection.

The horizontal axis in the panoramic image represents the rotation angle $\theta$, and we have to define a discretization step-size $\Delta \theta$. Since it is desired to preserve the aspect ratio of the input image pixels for the pixels in the panoramic image, we define the discretization step-size as $\tan \Delta \theta=1 / f$, based on the assumption that the pixel width in the input image is 1 .


Figure 14.3: Projection of image coordinates onto cylindrical coordinates.

### 14.2.1 Panoramic image generation

A technique to generate panoramic images is to take a sequence of images while rotating the camera around its vertical axis. It is important to note that the rotation has to be carried out around the optical center, since otherwise the images would not fit together (see Section 2.5.3). Each of these images is first converted to cylindrical coordinates $\theta, h$ independently. Because the images were recorded with different camera rotation angles, their position on the cylindrical surface is shifted by some amount $\theta_{i}$. This shift can be determined easily with a one-dimensional search over $\theta_{i}$ to minimize the image difference

$$
\begin{equation*}
E_{i j}=\frac{1}{\left|\mathcal{A}_{i j}\right|} \sum_{(\theta, h) \in \mathcal{A}_{i j}}\left|I_{i}\left(\theta-\theta_{i}, h\right)-I_{j}\left(\theta-\theta_{j}, h\right)\right| \tag{14.2}
\end{equation*}
$$

in the overlapping image area $\mathcal{A}_{i j}$ of images $i$ and $j$.
When stitching the individual images together into a single panoramic image, the seams between the images are often visible because of small alignment errors, or because of changes in the illumination conditions between the images. We apply a cross-blending between the two images to obtain a smooth transition. More complex techniques have been proposed for this problem. For example, [34] proposes to determine a path in the overlapping area $\mathcal{A}_{i j}$ from the top to the bottom border that minimizes the


Figure 14.4: A 360-degree image recorded using a camera with a parabolic mirror.
sum of luminance differences along this path. The advantage of this approach is that it also provides a sharp transition if there are moving objects in the scene.

### 14.2.2 Cameras for recording panoramic videos

For panoramic still images from static environments, we can capture several images sequentially and compose them into one panoramic image. For the recording of panoramic video sequences, the full 360 -degree view has to be captured at the same time. This poses the problem of mechanically mounting the cameras such that they cover the complete $360^{\circ}$, but also have an identical optical center. A solution is to place a hyperbolic mirror in front of a camera. A hyperboloid has two focal points with the property that a camera at focal point $F^{\prime}$ observes a 360-degree image with virtual optical center at the other focal point $F$ (Fig. 14.4). The main disadvantage of this technique is that the image resolution is generally low and unequally distributed in the image. Moreover, the image resolution is generally highest at the floor or ceiling, which are areas that are usually not very important.

In the Diver project, we used a second solution that is based on a setup with several cameras, oriented into different viewing directions to cover the complete 360 degrees. To enable a collision-free mounting of the cameras, mirrors are placed in front of the cameras to redirect the incoming light. With this approach, the cameras can be mounted with sufficient space


Figure 14.5: Images that will be combined into a panoramic image must be recorded with a unique optical center. For a static setup with multiple cameras, this is not possible because the cameras would be located at the same place. One solution is to use mirrors to redirect the light direction such that the cameras can be mounted without mechanical problems.
while the virtual optical center of all cameras is still at a joint position (Fig. 14.5). The advantage of this camera system is a high and uniformly distributed resolution in the panoramic image. For our experiments, we used a panoramic camera composed of five independent cameras.

Since the single cameras show significant lens distortions and are not mounted in perfect geometric alignment, the cylindrical transform cannot be applied directly. Instead, the transformation between input image and cylindrical coordinates was provided by the manufacturer as a regular mesh of calibrated feature-points. Each point in the mesh hereby defines a corresponding position between image coordinates and cylinder coordinates. The transformation for the pixel positions that do not fall exactly on mesh vertices was obtained by bilinear interpolation. Figure 14.6 depicts the mesh for one of the cameras. This example shows that this transformation not only includes the cylinder projection, but also fish-eye lens distortion and a twisted camera mount.

Another issue of the camera setup was that the single cameras only provided interlaced video. Because this would introduce severe distortions at moving objects during the irregular resampling in the dewarping process, it is important to deinterlace the input image prior to synthesizing the panoramic image. We implemented a fast ad-hoc deinterlacing algorithm which carries out deinterlacing selectively for the motion areas only. ${ }^{1}$

[^23]

Figure 14.6: Calibration data for one of the five cameras as it was provided by the manufacturer. This calibration information is the direct transformation from the camera image to the cylindrical panoramic image. Hence, it includes the correction of lens distortions, tilted camera mounting, and the transform to cylinder coordinates.

Because of the high data rate generated by the five cameras at full NTSC resolution ( $720 \times 486$ ), we first recorded the video stream of each camera independently. Afterwards, each video stream was deinterlaced, and all five streams were combined into a panoramic image sequence. The resulting panoramic video has a resolution of $3552 \times 480$ pixels. Figure 14.7(a) shows an example picture.

### 14.3 Visualization of panoramic videos

A panoramic image or video is a complete 360 -degree view of the environment around the camera. Hence, it is not an ordinary flat image and a variety of visualizations for this special images have been proposed. We briefly introduce the most important in the following, ending with our new
time-difference of the two fields. Hence, if this difference is significantly larger than the difference between lines of the same parity, the block is deinterlaced by duplicating the content of on field.
proposal of a visualization employing a 3-D room reconstruction.

- Unwrapped cylinder. The most common display technique for cylindrical panoramas is to unwrap the cylindrical surface to a flat image (Fig. 14.7(a)). At first glance, this looks like an image with very wide field of view. However, there are two properties that distinguish this image in cylindrical coordinates from a normal, planar image. First, the image shows a complete 360-degree surrounding, such that the viewer looks in all directions around him at the same time. This is an unusual experience, since the normal human view is limited to about 180-200 degrees (160 degrees with one eye) [100]. Second, straight lines are not preserved by the cylindrical projection. Hence, geometrical concepts like parallel lines and vanishing points, which are important for an intuitive understanding of the scene, cannot be applied easily. As a consequence, this mapping is difficult to understand and interpret for humans.
- Panoramic image browser (PIB). A visualization technique that preserves straight lines is the generation of virtual views from the position of the capturing camera. Based on the cylindrical panoramic image, the viewer application uses the inverse of Eq. (14.1) to generate rectified, flat views with a limited field of view from the camera position. Since these views cannot cover the complete 360 degrees, the user can interactively turn the displayed view in the left and the right direction ${ }^{2}$. The advantage of this technique is that the generated views look identical to real-world views. Especially, the synthesized images preserve the straight lines from the real world. However, the disadvantage of this technique is that a static visualization (e.g., a printout on paper), of the complete environment is impossible.
- 3-D cylinder projection. A visualization that combines the interactivity of the previous method with the possibility to have a quick scene overview is to display a $3-\mathrm{D}$ view of the cylinder surface with the panoramic image as texture (Fig. 14.7(b)). This approach can be used in the following two ways. First, the virtual camera can be placed at the center of the cylinder, such that the generated views look similar to the previous PIB approach. The main user interaction at this position is to turn the camera to look into different directions. Second, moving the virtual camera outside of the cylinder gives a static scene-overview by showing the complete cylinder at

[^24]
(a) Unwrapped cylinder.

(b) 3-D cylinder model.

Figure 14.7: The unrolled image from the virtual cylinder (a), and a 3-D view onto the virtual cylinder (b).
once. The combination of these two possibilities makes the visualization very flexible. It is important to note that the global view onto the cylinder gives some indication of the spatial arrangements in the scene, but the intuitive perception of this overview is often misleading. For example, consider that the panoramic image is recorded in a square room. The intuitive assumption is that every wall should cover 90 degrees in the panoramic image. However, this is not true, since the actual angle depends on the camera position. To see this, assume that the camera is placed close to a wall. Clearly, this wall will cover almost 180 degrees in the panoramic image. In fact, the symmetric uniformity of the cylindrical visualization and the absence of distinguished geometric features is often misleading and actually complicating an intuitive scene understanding.

- 3-D room projection. Although the previously discussed 3-D cylinder projection already combines the advantages of interactive rectified
views and a scene overview, the overview image is often misleading. The reason is that the cylinder surface is a virtual object that is not related to the original scene objects. In man-made environments, especially indoor locations, the space is usually defined by flat walls, which are perpendicular to each other. These walls are important for our orientation, but during the projection onto the $3-\mathrm{D}$ cylindrical surface, these hints for the human perception are lost; the cylinder looks the same from every direction.

To provide hints about the scene geometry to the viewer, it is important to present the scene overview in a way that mimics the original geometry. In particular, we propose a new visualization technique for the special case of indoor scenes where the room walls provide the main hints for orientation. The difference to the previous approach is that instead of projecting the surrounding onto a cylinder surface, we reconstruct the real room shape and use the panoramic image as wall textures (Fig. 14.8).
From the original camera position, the visualization appears equal to the PIB technique. However, for a distant camera, the scene geometry indicates the layout of the walls and the camera position during recording. Note that the projection onto flat walls also preserves straight lines, which makes the wall textures look realistic. However, the visualization should not be misunderstood as a complete $3-\mathrm{D}$ reconstruction. Changes of depth that are not modeled in the reconstruction can lead to perspective distortions if the camera was recording the area in an acute angle. On the other hand, our visualization technique is much easier to implement and use than a full 3 -D reconstruction.

In the following, we describe the $3-\mathrm{D}$ room reconstruction technique for rectangular rooms in more detail. The generalization to arbitrary room shapes or floor plans follows in Section 14.1.3.


Figure 14.8: A sample reconstruction of a rectangular room.


Figure 14.9: The room geometry should be reconstructed from the measured angles $\alpha_{0}, \ldots, \alpha_{3}$.

### 14.4 Reconstruction of rectangular rooms

In this section, we consider the problem of determining the wall sizes of a rectangular room from a cylindrical panoramic image captured in this room. Once we know the sizes of the walls and the position of the camera, we can project the panoramic image content onto these virtual walls and create the geometrical model for our visualization. While the wall sizes and camera position could be measured by hand, it is more convenient to obtain these values directly from the recorded image. The idea of our approach is to derive this information from the angles between the room corners, which the user has to mark in the image (Fig. 14.8(a)).

Since the panoramic image is given in cylindrical coordinates, the horizontal distance between two corners in the panoramic image corresponds to the angle between these corners, measured from the camera position (Fig. 14.9). Knowing these four angles (of which only three are independent, since they sum up to $2 \pi$ ), we can determine the ratio of the room dimensions and the camera position. It is not possible to recover the absolute room size, but this is also not required for the visualization, and we can simply set the size of one wall to unity.

The reconstruction is carried out in two steps. First the algorithm makes a preselection of positions that could potentially be the true camera positions. We derive that the valid camera position must be located on a circular connecting two room corners. With this information, we restrict the possible camera positions to a one-dimensional search space. Second, the algorithm carries out a binary search to determine the specific camera position on the circular arc. The search in the second step is guided by the pre-knowledge that the reconstructed room should be rectangular.


Figure 14.10: Determining the circular arc of valid camera positions.

### 14.4.1 The circular arc of possible camera locations

Prior to developing the actual algorithm, let us first examine possible positions of the camera in the reconstructed room. For this, we make use of the following theorem, which we briefly prove here for the convenience of the reader.

Theorem: (Euclid, Elements, Book III, Proposition 20.) Given three points $A, B, C$ on a circular arc $A C B$ with center $O$. It holds that $\angle A O B=$ $2 \angle A C B$.
Proof: Consider Figure 14.10(a). Since the triangle AOC is an isosceles triangle, $\angle A C O=\angle O A C=\phi$. But then, $\angle A O D=2 \phi$ (consider the reappearing angles at $A$ ). The same construction holds for the triangle $B O C$. Hence, the total angle $\angle A O B=2 \angle A C B$.

Note that in the preceding lemma, the location of $C$ on the circular arc $A C B$ does not influence $\angle A O B$. Hence, it also holds that the angle at $C$ is independent of its position on the arc. This lets us conclude that for fixed points $A, B$, all positions of $C$ that have a fixed angle $\angle A C B=\alpha$ lie on a circular arc. To find the center position and radius of this circular arc, let us consider the special case where $C$ is on the perpendicular bisection of $A B$. Assume that the points $A$ and $B$ have unit distance (Fig. 14.10(b)). Then, we get the radius from $\sin \alpha=1 /(2 r)$ and the distance of the center from $\tan \alpha=1 /(2 s)$.

Now, we are going to apply this theorem to our estimation problem. Let us normalize the room size such that the left (and right) wall has unit length


Figure 14.11: The camera position is located on the circular arc, but its position is unknown. (a) and (b): A binary search is applied to find the position for which the error $w_{t}-w_{b}$ is zero. (c) and (d): the positions for which the rays $q_{1}$ or $q_{2}$ are horizontal define the initial interval for the binary search.
and the top (and bottom) wall has length $w$ (Fig 14.9). We denote the four angles under which the four room walls are observed with $\alpha_{0}, \ldots, \alpha_{3}$. First, we concentrate on the left wall $A B$ of unit length, which is observed with an angle $\alpha_{0}$. Because of the previously derived theorem, we know that the camera position $C$ must lie on the circular arc $A C B$, and we can compute the position and size of this arc from the angle $\alpha_{0}$.

### 14.4.2 Searching for the camera position

Once we know the circular arc on which the camera is located, the remaining step is to find its position on the arc. To verify a potential camera position, we compute the wall sizes that would result for this position and accept the camera position if the resulting room is rectangular.

We begin the construction with the left wall, which has unit length. Since the assumption is that the room is rectangular, we know that the top and bottom wall must be perpendicular to this left wall. The width of the top and bottom walls are unknown, but their widths should be equal because the wall on the right side is parallel to the left wall.

Let us choose an arbitrary camera position on the arc and consider this position. Then, the corner-to-corner angles $\alpha_{1}$ and $\alpha_{2}$ define the direction of two rays $q_{1}, q_{2}$ emanating from the camera position in the direction of the room corners (Fig. 14.11). These rays intersect the top and the bottom walls in a distance $w_{t}$ and $w_{b}$ from the left wall, respectively. Because we know that the top and bottom wall should have equal length, $w_{t}$ should equal $w_{b}$. However, if we have chosen the wrong camera position on the circular arc, this will not be true.

Notice that if we move the camera upwards along the arc, the top intersection point moves to the left ( $w_{t}$ decreases), while the bottom intersection point moves to the right ( $w_{b}$ increases). To find the camera position for which $w_{t}=w_{b}$, we can exploit this behaviour by applying a binary search for the correct camera position. If $w_{t}>w_{b}$, the camera position is further to the top, while for $w_{t}<w_{b}$, the camera position is lower.

For some camera position, the ray direction of $q_{1}$ or $q_{2}$ becomes horizontal. For these positions (and the more extreme positions), there is no intersection of the rays with the top or bottom wall. These critical camera positions can be used to determine an initial interval of camera positions for the binary search. Starting the search with this interval not only reduces the number of iterations needed for the binary search, but it also removes the requirement to handle the special case in which the rays $q_{1}, q_{2}$ do not intersect the top or bottom walls.

### 14.4.3 Creating a virtual room visualization

When the sizes of the room walls and the camera position are known, we can create a virtual 3-D model of the room and generate textures for the room walls. To create the texture maps, we scan the $3-\mathrm{D}$ wall plane with the desired resolution of the texture maps and we compute the respective pixel position in the panoramic image by using the inverse of Eq. (14.1).

This obtained 3-D room model is rendered using an OpenGL-based
viewer application. The scene is built with the estimated camera position as the origin of the 3-D coordinate system. The user can control the rotation of the scene around the $x$ and $y$ axes, as well as the distance $d$ of the camera to the origin. The viewing transform is set up as

$$
\mathbf{p}^{\prime}=\mathbf{K}\left[\mathbf{R}_{\mathbf{x}} \mathbf{R}_{\mathbf{y}} \mathbf{p}+\left(\begin{array}{l}
0  \tag{14.3}\\
0 \\
d
\end{array}\right)\right]
$$

where $\mathbf{p}$ denotes the 3 -D point position, $\mathbf{R}_{\mathbf{x}}, \mathbf{R}_{\mathbf{y}}$ are the rotation matrices and $\mathbf{K}$ is the perspective projection matrix. This particular sequence of transformations allows for a very intuitive navigation. When the distance of the camera to the origin is decreased, the program avoids $d$ to become negative. This makes it very easy to place the camera at the position of the real camera (move forward until the virtual camera reaches the original camera position). From that position, the user views the scene just as if he would be at the camera position in the real world. Panning with the virtual camera at this special position gives exactly the output as displayed by popular viewers for panoramic images. The second useful viewing position is looking down on the complete room from above the scene, since this gives a quick overview of the general scene layout. An example visualization created with the described algorithm is depicted in Figure 14.8.

### 14.5 Reconstruction of floor plans

The reconstruction algorithm described in the previous section was limited to rectangular rooms. In this section, we extend this algorithm to enable it to reconstruct the geometry of arbitrary rooms. We keep the principle that corners are manually marked in panoramic images, and that the algorithm derives the camera position and wall sizes from the angles between room corners. The layout of the room walls is also specified by the user. For more complex rooms, it is often impossible to see all walls in only one image because of occlusions. In these cases, the algorithm uses several panoramic images captured from different positions.

### 14.5.1 Previous work

Several algorithms for 3-D reconstruction have been proposed. They can be coarsely divided into algorithms without pre-knowledge about the scene and algorithms making use of a scene model. Algorithms of the first class are usually very complex to implement [149] and they are probably not robust enough in cases of low-textured surfaces. Algorithms of the second class
employ a complete geometric model of the object or scene and they only adapt the sizes based on the observed images. An algorithm of this second class is described in [37]. Another algorithm [168] considers specifically the reconstruction of room shapes from panoramic images. Compared to our algorithm, it supports more general geometries than a collection of walls, but compared to our proposal, it is more complex to implement and to use.

### 14.5.2 Reconstruction algorithm concept

Our floor plan reconstruction is based on the same user interaction as in the simpler rectangular room case. The user also marks the position of the room corners in the input image. However, while we previously only considered reconstruction from a single panoramic image, we now allow for an arbitrary number of panoramic images. This is necessary since many room corners can be occluded from some camera positions. The more panoramic images are used, the more information we have available for the reconstruction. The less pre-knowledge about the room geometries is available (non-perpendicular walls, unconnected free-standing walls), the more images are required.

The algorithm starts with an initial room configuration that defines the room layout (position of walls and constraints about perpendicular walls), but that does not yet include the correct wall sizes. For an example, see Fig. 14.13. This figure shows a user-supplied geometric room model, where the outline of the room is specified, but the correct wall sizes are still unknown. The basic principle of the algorithm is to compute the corner-tocorner angles from the current model and compare them with the measured angles. A gradient descent search is used to adapt the wall sizes such that the differences between angles in the model and the measured angles are as small as possible.

In the following, we describe the algorithm in four steps.

- Section 14.5.3. First, the parameterization of the model is constructed. Parameters are chosen such that hard constraints like perpendicular or parallel walls are enforced by the parameterization itself.
- Section 14.5.4. Second, we present the parameter-estimation algorithm. This step adapts the parameters such that the corner-to-corner angles in the model fit to the angles measured in the input images.
- Section 14.5.5. The convergence robustness depends on the definition how measured angles and angles in the model are compared. We compare an inner-angle definition with an outer-angle definition
by examining the error function for local minima or plateaus, which decrease the robustness of the optimization.
- Section 14.5.6. Because the optimization is based on a gradient descent approach, a good initialization is required. The evaluation of the error function will show that local minima and plateau region can be avoided if the initialization satisfies some ordering conditions. In this last step, we explain how the initialization is obtained.


### 14.5.3 Modeling the floor plan geometry

The floor plan reconstruction algorithm uses two types of information for the estimation:

- the angles between room corners measured from their position in the panoramic images, and
- the predefined geometrical layout of the room. This geometrical model includes the relative position of the walls, but not their size. The model also considers pre-knowledge about right angles between walls.

Let us first consider the number of degrees of freedom when estimating the floor plan geometry. A floor plan is parameterized by the 2-D positions of the room corners and the camera positions. The camera positions are required to carry out the texture mapping.

We start with a simple example of a rectangular room and one camera, which is similar to the special case that we considered in the previous section. This configuration gives $4 \times 2$ parameters for the room corners plus two parameters for the camera position (Fig. 14.12). However, the absolute placement of the room in our coordinate system is arbitrary and we can fix one corner to a predefined position, like ( 0,0 ). Moreover, we can fix the overall rotation angle of the floor plan, and as absolute size cannot be determined, we can also fix the length of one wall to, e.g., unity. The easiest way to do this is to fix the position of a second corner to, e.g., $(0,1)$. In total, this reduces the number of degrees of freedoms by four.

The reduction from ten parameters to only six was obtained by eliminating superfluous degrees of freedom in the parameterization. On the other hand, we can add more pre-knowledge about the room geometry. For example, we can assume that the room shape is rectangular. This preknowledge can be expressed with three constraints, each forcing one wall to be perpendicular to another wall. These three constraints further reduce the number of free parameters from six to three, thereby making a reconstruction possible.


Figure 14.12: By adding geometry constraints to remove unnecessary overparameterization, we can reduce the number of parameters from 10 in the general case (a) to only 3 for a rectangular room (d). The free parameters are indicated with double arrows.

For our general floor plan reconstruction, we enforce the constraints for perpendicular walls implicitly through the parameterization. We normalize the rotation of the complete floor plan such that (most) walls will be aligned along the horizontal and vertical coordinate axes. Each wall that is aligned to the coordinate axes can be parameterized with only three parameters. For example, a vertical wall is parameterized by the two corner positions, but both positions share the same $x$ coordinate. For the right wall in Fig. 14.12(d), we would get the corner positions ( $x_{1}, y_{0}$ ) and ( $x_{1}, y_{1}$ ).

Furthermore, we also add the normalization of the floor plan position and size as hard constraints in the parameterization. For this, we select one vertical wall and define one corner postition to $\left(x_{0}, y_{0}\right)=(0,0)$ and the other corner position to $\left(x_{0}, y_{1}\right)=(0,1)$. Note that using this parameterization for the rectangular-room case, only $x_{1}$ for the right wall position and $x_{2}, y_{2}$ for the camera location remain, so that we can compute these three free parameters from the three angle measurements.

A more complex example is depicted in Fig. 14.13. The room shape has eleven walls, but it is parameterized with only six free parameters $x_{1}, \ldots, x_{3}, y_{2}, \ldots, y_{4}$. Additionally, the two camera positions add four parameters $x_{4}, y_{5}, x_{5}, y_{6}$. From the image of the left camera, we can obtain nine angle measurements, since two of the walls are at least partly occluded. The right camera can contribute seven angle measurements. In total, we have 16 measurements for 10 parameters and the reconstruction is possible. Note that a reconstruction would also be possible with only the left camera. In this case, we would only have nine measurements, but also only eight parameters, since the position of the right camera is not included. On the


Figure 14.13: Room corners are specified by coordinates $x_{i}, y_{i}$. Horizontal and vertical walls will reuse the same $x_{i}$ or $y_{i}$ coordinate for both corners. This implicitly encodes the pre-knowledge that these walls have to be horizontally or vertically aligned. Camera positions are assigned their own pair of $x_{i}, y_{i}$ coordinates.
other hand, a reconstruction from only the right camera is impossible, since we would have eight parameters to estimate from only seven measurements. Note that a sufficient number of measurements does not generally assure that the reconstruction is possible. This is the case when there are more measurements available than required for some walls and at the same time, too few measurements for other parts. However, in practice, this is rarely the case.

### 14.5.4 Estimating the floor plan parameters

The central task in the floor plan reconstruction is to estimate the model parameters based on the angle measurements that were taken from the panoramic images. The model parameters consist of the coordinates $x_{i}, y_{i}$ of the wall corners and the camera positions. According to the geometric constraints, some of these coordinates can appear in the specification of several positions. All coordinate values that appear in the model are collected in a large parameter vector

$$
\begin{equation*}
\mathbf{v}=\left(x_{0}=0, x_{1}, x_{2}, x_{3}, \ldots, y_{0}=0, y_{1}=1, y_{2}, y_{3}, \ldots\right) \tag{14.4}
\end{equation*}
$$

in which three entries are fixed (namely $x_{0}=y_{0}=0$, and $y_{1}=1$ ) to remove the superfluous degrees of freedom. To find the corresponding coordinates
for a position $\mathbf{p}_{i}$, we use two index sets $m_{i}$ and $n_{i}$ into the parameter vector $\mathbf{v}$ to define $\mathbf{p}_{i}=\left(x_{m_{i}}, y_{n_{i}}\right)^{\top}$,

From the captured panoramic images, we obtain a set of angle measurements. Each measurement gives an angle $\alpha_{i, j, k}$ between corners $\mathbf{p}_{i}$ and $\mathbf{p}_{j}$, seen from camera position $\mathbf{p}_{k}$. We denote the set of available measurements as $\mathcal{M}=\{(i, j, k)\}$. Furthermore, we can compute angles $\beta_{i, j, k}$ that correspond to the measured angles from the geometric model as

$$
\begin{equation*}
\beta_{i, j, k}=\arccos \frac{\mathbf{d}_{i k}^{\top} \mathbf{d}_{j k}}{\left\|\mathbf{d}_{i k}\right\| \cdot\left\|\mathbf{d}_{j k}\right\|}, \tag{14.5}
\end{equation*}
$$

where $\mathbf{d}_{i}=\mathbf{p}_{i}-\mathbf{p}_{k}$ and $\mathbf{d}_{j}=\mathbf{p}_{j}-\mathbf{p}_{k}$ are the vectors from the camera position $k$ to the corners $i$ and $j$. This equation defines $\beta_{i, j, k}$ as the inner angle between these vectors. Actually, we will shortly replace this definition with a slightly modified one that gives a better convergence behaviour. For an error-free ideal case, all measured angles $\alpha_{i, j, k}$ should equal the angles $\beta_{i, j, k}$, computed from the adapted model. Because of noisy measurements, these angles will not be exactly equal, and we define the total error of the floor plan model as

$$
\begin{equation*}
E=\sum_{(i, j, k) \in \mathcal{M}}\left|\beta_{i, j, k}-\alpha_{i, j, k}\right|, \tag{14.6}
\end{equation*}
$$

which we minimize with a Quasi-Newton optimization. The convergence of this optimization depends on two factors: the smoothness of the cost function $E$, and the initialization. We discuss these two topics in the successive two sections.

### 14.5.5 Improving the convergence behaviour

When considering again the definition of $\beta_{i, j, k}$ from Eq. (14.5), we notice that the definition gives the non-oriented inner angle $\beta_{i, j, k} \in[0 ; \pi]$ between two vectors. During experimenting with this measure in the optimization process, we occasionally observed that the optimization did not converge. To see why the above angle definition can cause problems in the optimization process, consider the very simple case that there is only one camera which observes a single wall. Imagine that the camera is moved on a line perpendicular to the wall from one side of the wall through the wall to the other side. While approaching the wall, the angle $\beta_{i, j, k}$ increases to $\pi$ and after crossing the wall, it decreases again. For this case, the cost function $E$ is symmetric to the wall. In the optimization process, this has the disadvantage that there is a minimum of $E$ on each side of the wall (Fig. 14.14).


Figure 14.14: With the non-oriented angle $\beta_{i, j, k}$, it cannot be distinguished on which side of a wall the camera is located. The oriented angle $\beta_{i, j, k}^{\prime}$ has a single minimum at the correct side.

To prevent this effect, we changed the definition of the angle $\beta_{i, j, k}$ to an oriented angle. We define the oriented angle $\beta_{i, j, k}^{\prime}$ as the angle from corner $\mathbf{p}_{i}$ to corner $\mathbf{p}_{j}$, measured in counter-clock-wise direction (see Fig. 14.15). The orientation of the two corners is detected by computing the signed area spanned by the two vectors $\mathbf{d}_{i k}$ and $\mathbf{d}_{j k}$ from the camera to the wall corners. The signed area is obtained easily from the determinant of the matrix composed of these two vectors. Thus, we can compute the oriented angles as

$$
\beta_{i, j, k}^{\prime}= \begin{cases}\beta_{i, j, k} & \text { if } \operatorname{det}\left[\mathbf{d}_{i k} \mid \mathbf{d}_{j k}\right] \leq 0  \tag{14.7}\\ 2 \pi-\beta_{i, j, k} & \text { if } \operatorname{det}\left[\mathbf{d}_{i k} \mid \mathbf{d}_{j k}\right]>0\end{cases}
$$

In comparison with the previous inner angle definition, the new oriented angle can distinguish between the camera being on the correct side of a wall and being on the backside of the wall. Using this angle definition, we obtain a clear minimum at the correct side of the wall, and the error $E$ increases monotonically with increasing distance from the optimal position (Fig. 14.14).

## Dependency of the model error on the camera position

Let us now examine a more complex case with a variable camera position in a rectangular room having fixed walls at $x= \pm 1$ and $y= \pm 1$. The error surface of $E$ for the two angle definitions is depicted in Fig. 14.17. In the

(a) $\beta_{i, j, k}$

(b) $\beta_{i, j, k}^{\prime}$

Figure 14.15: Definition of angle differences. While $\beta_{i, j, k}$ is the inner angle between the two vectors (a), $\beta_{i, j, k}^{\prime}$ is defined as the angle from $\mathbf{p}_{i}$ to $\mathbf{p}_{j}$ in counterclockwise direction (b).


Figure 14.16: Illustration of angles in a plateau area. (a) Moving the camera within the grey plateau area does not change the total cost $E$. (b) In each of the grey areas, $\alpha_{i}<\beta_{i}$. The symmetric areas, mirrored at the walls are not shown.
case of the inner angle definition, we notice that there are plateau regions around the room walls with constant $E$. These areas impose difficulties, since the gradient based optimization can get stuck in this area.

To understand why these plateau regions exist, let us concentrate on one of these regions as depicted in Figure 14.16(a). The depicted area corresponds to the area outside of the room, for which $\alpha_{l}<\beta_{l}$, which means that the camera in the model is closer to the wall than in reality. For simplicity of notation, we use the notation $\alpha_{l}=\alpha_{1,4,5}, \beta_{l}=\beta_{1,4,5}$ as abbreviation for the angles corresponding to the left wall. We use similar abbreviations for the top $(t)$, bottom (b), and right $(r)$ walls. For the considered plateau area at the left wall, $\beta_{r}<\alpha_{r}, \beta_{t}<\alpha_{t}, \beta_{b}<\alpha_{b}$ as
illustrated in Figure 14.16(b). If we compute the total angle error for all four walls as

$$
\begin{align*}
E & =\left|\beta_{l}-\alpha_{l}\right|+\left|\beta_{t}-\alpha_{t}\right|+\left|\beta_{r}-\alpha_{r}\right|+\left|\beta_{b}-\alpha_{b}\right| \\
& =\left|\beta_{l}-\alpha_{l}\right|+\left|\beta_{t}-\alpha_{t}\right|+\left|\beta_{r}-\alpha_{r}\right|+\left|\beta_{l}-\beta_{t}-\beta_{r}-\alpha_{b}\right| \tag{14.8}
\end{align*}
$$

we can resolve the absolute-value operators to derive

$$
\begin{align*}
E & =\left(\beta_{l}-\alpha_{l}\right)-\left(\beta_{t}-\alpha_{t}\right)-\left(\beta_{r}-\alpha_{r}\right)-\left(\beta_{l}-\beta_{t}-\beta_{r}-\alpha_{b}\right) \\
& =-\alpha_{l}+\alpha_{t}+\alpha_{r}+\alpha_{b} \tag{14.9}
\end{align*}
$$

which is a constant. This explains the plateaus in the error function at each wall.

If we consider the same room geometry, but with the oriented angle, we obtain the error surface as depicted in Figure 14.17(b). This error function shows neither plateau areas nor local minima for varying camera positions. Instead, the error surface shows discontinuities whenever the camera crosses a wall plane, because in this moment, the oriented angle jumps between 0 and $2 \pi$. However, these steps in the error function impose no problem for the gradient descent search, since the step is always downwards in the direction to the minimum. Consequently, while the optimization can get stuck on the plateau areas using the inner angle definition, convergence is ensured with the oriented angle.

## Dependency of the model error on the wall positions

A similar behaviour of the model error can be observed when keeping the camera position constant and varying the wall positions. We examine again the case of a rectangular room, with walls at $x, y= \pm 1$ and the real camera at $(0,0)$. However, now we consider the position of the right wall as unknown and variable. Figure 14.18 depicts the resulting model error $E$ that is obtained for each angle definition. In Figure 14.18(a), the camera is set to the correct position at $(0,0)$, while it is set to $(-0.5,0)$ in Figure $14.18(\mathrm{~b})$. We can observe that the error function for the non-directed angles $\beta$ show larger plateau regions and even local minima. Similar to the previous example, the directed angle definition $\beta^{\prime}$ leads to steps in the error function, but no local minima. Note that the left plateau area for oriented angles starts when the right wall position is moved so far to the left, that it crosses the left wall such that it is actually left of the left wall. For non-oriented angles, the plateau region already starts when the wall crosses the camera position.

We can conclude that the oriented angle $\beta^{\prime}$ shows clear advantages over the inner angle $\beta$. The oriented angle does not result in plateau regions

(a) Model error $E$ computed with $\beta_{i, j, k}$.

(b) Model error $E$ computed with $\beta_{i, j, k}^{\prime}$.

Figure 14.17: Model error $E$ when moving the camera position while keeping the wall coordinates constant.


Figure 14.18: (a) Total model error for a rectangular room, depending on the position of the right wall. (b) If the order of the left and right wall is interchanged, it can lead to a constant error $E$.
for varying camera positions and it provides a clear error minimum. Both angle definitions lead to plateaus when the order of room walls is swapped, but for the oriented angles, these regions are smaller.

We conducted the same experiments with defining the error as the sum of squared angle differences. However, this definition leads to an error surface with many local minima, so that we did not pursue this further.

### 14.5.6 Initialization of the floor plan layout

In the last section, we observed that the error surface of $E$ is smooth and has a unique minimum as long as the cameras are placed within the rooms, and as long as the order of the room walls are not interchanged. Hence, the optimization should be started with a configuration in which these conditions are satisfied to ensure convergence.

We obtain the initial placement of the walls by examining the userspecified floor plan model. Assuming that the walls are oriented along north-south or west-east direction, we can determine the direction going from one wall corner $\mathbf{p}_{i}$ to the other corner $\mathbf{p}_{j}$. Diagonal walls are not considered here. Based on this information, we build a west-east ordering of the points such that point $\mathbf{p}_{i}<_{x} \mathbf{p}_{j}$ if corner $i$ is to the west of $j$. A similar ordering $<_{y}$ can be defined for the north-south direction. Subsequently, these orderings can be used to assign increasing coordinates. Note that the orderings do not necessarily impose a unique valid ascending enumeration of the coordinates. For example, in Fig. 14.19, the coordinates $x_{2}$ and $x_{3}$


Figure 14.19: An example initialization of a floor plan, based on the predefined ordering. Note that swapping $x_{2}$ and $x_{3}$ also gives a valid ordering.
could also be swapped and still fulfill $<_{x}$. Any of these admissible orderings provides a good initialization of the wall positions. Finally, we initialize the position of the cameras at the center of all wall corners that are seen in each camera's image.

### 14.5.7 Obtaining wall textures from the panoramic images

After the optimization process has converged, the position of the walls and the cameras are known, and we can generate texture maps for the walls by projecting the panoramic image content onto the wall planes. Since we may have several views of the same walls, recorded by different cameras, we have to decide from which camera image we extract the image data. The following points have to be considered.

- The wall should be visible. Cameras that are located on the back side of the wall, or which are occluded by other walls cannot be used.
- The larger the distance of the camera to the wall, the lower the texture resolution that is obtained.
- If the camera is too close to the wall, parts of it are viewed in an acute angle. As a result, changes of depth that are not reflected in the floor plan model can lead to perspective distortion artifacts. This applies, e.g., to furniture that is not projected orthogonally onto the wall.
- A wall may not be visible completely in any single camera view. In this case, the texture information has to be collected from several camera images.

The walls are processed independently, where we first determine which cameras are located at the front side of a wall. This information is obtained easily using the oriented angle from Eq. (14.7). If $\beta_{i, j, k}^{\prime}>\pi$, then the camera $\mathbf{p}_{k}$ is located at the backside of wall $\mathbf{p}_{i}, \mathbf{p}_{j}$. Cameras that are at the backside are excluded from the further processing.

To decide from which camera the wall texture should be taken, we evaluate the expected image quality by determining the deviation of the camera position from an ideal camera position. For room corners $\mathbf{p}_{1}, \mathbf{p}_{2}$, we define the ideal camera position as $\mathbf{p}_{c}=\frac{1}{2}\left(\mathbf{p}_{1}+\mathbf{p}_{2}\right)+\frac{1}{2} \mathbf{R}_{\perp}\left(\mathbf{p}_{2}-\mathbf{p}_{1}\right)$, where $\mathbf{R}_{\perp}$ is a rotation by $\pi / 2$. This places the ideal camera position on the perpendicular bisection of the wall at a distance that is half of the wall width. All cameras that are not at the backside are ordered according to the distance of this ideal position. In this ordering, the camera which is closest to the ideal position, comes first. For every column of texture pixels, the ray between the first camera and a pixel in the column is checked for intersection with other walls. If there is an intersection, the second camera is checked for free sight to the pixel, and so on. Note that only one pixel in the column has to be checked since all walls are upright planes.

### 14.6 Experimental Results

Experiments have been carried out for both reconstruction algorithms presented in this chapter. For the rectangular-room reconstruction, the input images were captured with the panoramic video camera described in Section 14.2.2. These are well calibrated and generate undistorted panoramic images. An example reconstruction result is shown in Fig. 14.8.

Example results for the floor plan reconstruction are shown in Figure 14.21 and Figure 14.22. The input images for the floor plan reconstruction were captured with a digital still camera and combined into a panoramic image later. The focal length of the camera had to be estimated, since the EXIF data did not contain this information. During the stitching process, small inaccuracies in the image alignment were observed, which lead to inaccurate angle measurements. The computation time for the reconstruction was clearly below one second in all of our examples. The time for generating the texture maps depends on the required resolution and the number of walls, and was about one second for our most complex model. We evaluated the accuracy of the reconstruction result by comparing the normalized size of the walls in the reconstruction with their real sizes. The average deviation was about $4 \%$, which is probably mainly due to the inaccurate alignment of the input images. Moreover, for simplicity, we assumed that the walls itself have zero depth, which is obviously wrong
in reality and which also leads to small deviations in room size. Note that these inaccuracies are not obviously visible in the reconstruction, because the wall textures are stretched by this factor. Corners in the texture image always map exactly to corners in the geometric model.

### 14.7 Conclusions

In this chapter, we have described techniques to capture panoramic images and videos and we have discussed ways for optimal presentation of these panoramic images to the user. We have proposed a visualization specialized for panoramic images recorded in a rectangular room, which reconstructs the room geometry from the panoramic image and presents the panoramic image as the projection onto the room walls. The reconstruction algorithm requires only minor user support and is guaranteed to find the optimum solution. Furthermore, we generalized the concept to the reconstruction of floor plans, comprising an arbitrary number of arbitrarily shaped rooms (preferably but not necessarily with perpendicular walls).

Our conclusion is that the proposed visualization can provide a better understanding of the scene to the user than a flattened panoramic image or a projection onto a cylinder, where the information about the room geometry is lost. Applications of our proposal, especially for the floor plan reconstruction, are also the advertisement of appartments or hotel rooms, for which virtual tours could be made available online. Another application could be the reconstruction of scenes in surveillance systems, in which the objects are extracted from the video and inserted into the 3-D model at their corresponding real-world position. It should be noted that both reconstruction algorithms can be used directly with panoramic video instead of single images, providing video textures on the walls of the 3-D model. Therefore, the geometry model only has to be computed once if the camera positions are kept fixed.

## Future research

In future research, the reconstruction could be extended to a completely automatic process. Note that in a cylindrical panoramic image of a room, the vertical lines of the room corners remain straight, while the horizontal lines at ceiling and the ground become bent (see Fig. 14.8(a)). Tracing along the bent horizontal lines, it is easy to find the room corners, because these corners are always located at sudden changes of the line direction. Depending on the angle in which these lines meet in the corner, it is even possible to distinguish between concave and convex corners, corresponding


Figure 14.20: The example room (a) is recorded with a camera located at the black spot. This results in the panoramic image (b). The convex corner $C$ is indicated with a dashed line.

(a)

Figure 14.21: Example reconstruction of a single, non-rectangular room from only one panoramic image.
to an inwards ( 90 degrees) or outwards (-90 degrees) corner (Fig. 14.20). Furthermore, corners at occluding walls show as discontinuities between the bent horizontal lines. If several panoramic images from the same room are available, corresponding corners could be identified by comparing the wall texture.


Figure 14.22: Example reconstruction for a complete appartment.

Plaudite, amici, comedia finita est.

- Applaud, my friends, the comedy is over. (Ludwig van Beethoven, on his deathbed)



## Conclusions

This thesis has presented various techniques for video-object segmentation. An automatic segmentation system for rotating cameras was presented and extended with object-model controlled segmentation and camera autocalibration. In this chapter, the achievements are summarized and it is discussed how these techniques could be enhanced in future research. Finally, interesting research directions are highlighted that may be promising approaches for future video-segmentation systems.

### 15.1 Discussion on the individual chapters

In the sequel, the achievements of this thesis and ideas for future research are discussed separately for each of the chapters of this thesis.

### 15.1.1 Chapter 3 and 4: camera-motion estimation

The first step in the proposed segmentation system is the alignment of all input frames into a background image. To compensate for camera motion, a combination of a feature-based motion estimator (Chapter 3 and 4) and a direct motion estimator (Section 5.2 ) is proposed. This combination couples the high accuracy of the direct motion estimation with the robustness to fast motion because of the feature-based estimator. The separation of object motion from the camera motion is carried out with a robust-estimation algorithm (RANSAC). This algorithm has been modified since the original algorithm does not reach the theoretically predicted performance.

## Feature-based vs. direct estimation

In our experiments, the computation speed of the feature-based estimator reaches real-time execution, while the direct motion estimation is approximately a factor of ten slower. Hence, it would be interesting to explore if a comparable estimation accuracy can be obtained without the direct motion estimator. The reader should recall that while the frame-to-frame parameters of the feature-based estimator can be computed with good accuracy, the error accumulation, when concatenating the motion parameters to long-term frame-to-sprite parameters, reduces the final accuracy. A possible approach to prevent this drift could be to compute the featurebased motion parameters directly between the input frames and the sprite image. Our conjecture is that this approach would reach comparable accurate motion parameters at a computation speed comparable to the current feature-based estimator, i.e., real-time.

## Robustness in difficult scenes

In our experiments, we have occasionally observed sequences, for which the feature-points are concentrated in one corner of the image. In this case, the estimation of motion parameters is of low numerical stability and usually results in skewing motion $(k \neq 0$ in Table 2.1).

Although this kind of motion is included in the projective motion model, it represents a physically impossible motion. In fact, compared to the eight free parameters of the projective motion model, there are only four varying physical parameters (camera rotation and focal length). Another
two parameters (principal point) are constant throughout the sequence. One approach to increase the robustness of the motion estimation may be to impose constraints or a parameterization that only permits physically possible motions.

### 15.1.2 Chapter 5: background estimation

In Chapter 5, a new algorithm for synthesizing a pure background image without foreground objects has been proposed. In contrast to previous algorithms, which usually estimate a statistical background model, we classify the content of each frame explicitly into foreground and background classes. This approach yields better results particularly in short sequences. For video segmentation, this is particularly important because the input is often a short scene from a longer movie for which the amount of input frames is fixed.

The proposed background-estimation algorithm divides the image into small blocks and determines for each block the periods of time, in which the image content is stable. Additionally, the pre-knowledge that background content is visible in comparable periods as in neighboring blocks is exploited to select those periods during which the background is visible. The central data-structure in this algorithm is the similarity matrix which collects information if the block content at two time instants is similar.

## Alternative implementation with block-diagonal matrices

While our algorithm uses a combinatorial hill-climbing optimization to find the periods with high similarity of content, this problem can also be understood as resorting the matrix into block-diagonal form. An algorithm to bring a matrix to block-diagonal form has been described in [32] for the application of multi-object 3-D reconstruction. It will be interesting to compare both optimization approaches with respect to efficiency.

When the matrix is resorted to block-diagonal form, each block can be subsumed into one state. Exploiting again the spatial coherency between blocks, a Markov random field can be applied to determine the background label for each block.

### 15.1.3 Chapter 6: multi-sprites

The projective motion model, which is commonly used to describe rotational camera motion, assumes that the scene can be projected on a flat plane. As shown in Chapter 6, this does not work for large rotation angles. The proposed multi-sprite technique makes it possible for the first time to


Figure 15.1: When the 3-D position of the sprite planes is known, the texture can be projected onto a cube-map.
cover scene backgrounds recorded by arbitrary rotational camera motion. Rather than of projecting the background scene onto a single plane, the scene is now covered by a collection of planes. Note that although we know the homography between these planes, their 3-D positions in space are unknown. However, these 3-D positions can be computed when the multisprite technique is combined with the camera autocalibration described in Chapter 12.

## Cube maps

A further possible model for scene backgrounds, which is popular in computer graphics, is the cube map model. The cube map is a virtual cube that encloses the camera. The background scene is projected onto the faces of the cube, resulting in six square texture images. Compared to a spherical model, the cube map has the advantage that the homography transform can be reused and no transcendental functions are required. Compared to the multi-sprite representation, it has the advantage that it is easier to work with, since the number of planes and their position is a-priori known, and because simple homographies can be used as the transforms. On the other hand, unlike the multi-sprite model, the cube map cannot adapt to different resolutions that may be required to prevent a loss of detail (e.g., the resolution varies during a camera zoom). Moreover, a cube map has always a constant size, covering every viewing direction, independent of the actual camera motion, which can also lead to inefficient storage.

Note that it is not possible to estimate the textures of a cube map directly from the input sequence, because it requires that the camera pa-
rameters are available in absolute physical units. However, we can start with a multi-sprite model onto which the camera autocalibration is carried out to determine the position of the multi-sprite planes in 3-D space. After this, the multi-sprite representation can be converted to cube map textures (see Fig. 15.1).

### 15.1.4 Chapter 7: background subtraction

The proposed video-object segmentation is based on an extended back-ground-subtraction algorithm. This algorithm compares the input image with the background image and identifies the changed areas as foreground. Compared to previously proposed background-subtraction algorithms, we have added the concept of risk maps to prevent segmentation errors that are due to misalignment of the input image to the background.

## Null-hypothesis vs. object models

In future research, the foreground-object segmentation can be enhanced in several simple or more fundamental ways. Currently, the segmentation is carried out independently for each input frame, but improvements might be obtained by considering several frames at once. This can be modeled, e.g., with a three-dimensional Markov random field that enforces temporal stability. This concept can even be extended with a local object-motion compensation to connect corresponding pixels along the time axis.

A more fundamental change is to redefine the definition of changed pixels. In the current system, pixels are compared only to a null-hypothesis for an unchanged pixel. An alternative formulation would be to also have a model for the foreground, so that the choice for the best-fitting hypothesis could be made. However, the main difficulty of the two-hypotheses approach is to obtain an object model, especially for non-rigid objects.

### 15.1.5 Chapter 9 and 10: graph-based object models

An essential problem in segmentation is to define the object that is intended to be extracted. We have approached this problem by employing user-defined object models to identify specific objects in the input. A major design decision is the choice of representation that should be used for the object models. We decided to use a graph-based model in which nodes represent image regions and edges represent spatial proximity. The advantage of this model is that it allows articulated object motion. Moreover, since we restricted the graph to tree structures, an efficient implementation of the object detection is possible with a dynamic-programming approach.

## Object-model matching

A principal difficulty in applying object models is the definition of a robust cost function for object localization. Our cost function is a combination of region-color differences, region shapes, object deformation, and other terms (see Eqs. (9.4), (9.5) or (10.9)). Note that an optimal weighting of these cost contributions cannot easily be derived. Even the sensible definition of each single cost contribution is difficult. Hence, the definition of a suitable model is always a compromise between a robust detection and accurate placement. More insight into this problem can probably be obtained with a further exploration how the human visual system perceives its environment.

## Object models in the segmentation process

Because the adopted object model allows for some variability, the detected object position does not cover the real object area accurately. In order to extract detailed object boundaries, the object detection should be combined with other techniques that operate at the pixel level. In our two proposals in Chapter 9 and 10, we combined the object detection with a color segmentation. The primary difference between our two proposals is that for the object detection in cartoon sequences, color segmentation is carried out prior to object detection, whereas for natural video sequences, color segmentation follows the object detection.

In the object-detection system for natural video, we have observed in our experiments that for textured objects, the object boundaries cannot be accurately found. For example, in the results shown in Figures 10.11 and 10.12 , it is visible that segmentation borders along uniformly-colored regions are acceptable, whereas the borders at textured regions are poorly segmented (they appear fringy). The reason is that the color segmentation splits textured areas into a multitude of small regions. Many of them are not covered by the object model and consequently, they are excluded from the object mask. More accurate object boundaries have been obtained with change-detection masks (CDMs), which work particularly well if the object is textured. The disadvantage of CDMs is that they can only detect objects if there is a significant difference to the background or to a preceding frame (see Fig. 8.14).

Considering these properties of the algorithms, we can propose a promising object-detection system for further research. The framework of this system is depicted in Figure 15.2. This segmentation system combines the following two principal techniques.

- Depicted at the left side is a change-detection algorithm, similar to the


Figure 15.2: Proposal for a low-latency real-time segmentation system.
system described in Part I of the thesis. However, a major difference is that the change detection is not based on the comparison to a scene-background image, but to the previous frame. Note that this means that all steps of the background synthetization can be omitted, but the global-motion estimation is still required to compensate the camera motion of successive frames.

- The remaining part of the system comprises an object-detection algorithm as described in Chapter 9 and 10. The results of both techniques should subsequently be combined such that the object boundary is derived from the CDM and the interior of the object is filled from the detected object position.

This new alternative segmentation system would have the advantage that it enables a real-time segmentation with very low delay. Because no background-sprite is generated, the long latency introduced by this step vanishes. On the other hand, the segmentation relies to a large extent on the object-model detection, since the CDM is only computed between successive frames, leading to incomplete segmentation masks. To achieve a real-time execution speed and a high temporal consistency, we suppose that the object model should be further extended with a dynamic model, describing its possible motions.

### 15.1.6 Chapter 11: Corridor Scissors and circular paths

Chapter 11 has approached the segmentation problem with a semi-automatic segmentation technique. The Intelligent Scissors algorithm has been extended to a Corridor Scissors tool, in which the user coarsely marks the outline of the object with a broad corridor. Afterwards, the algorithm extracts the accurate object contour within the corridor and tracks this contour through the successive images. The elegancy of the approach is that the manual segmentation and the tracking step are both based on the same core algorithm, computing shortest circular paths in graphs. Both computation processes differ only in the cost definition for the path computation. In the manual segmentation, a cost based on the image gradient is used. For the tracking, the texture along a previously-segmented contour is compared with the current image.

The core of the Corrisor Scissors tool is a new algorithm for computing shortest circular paths in planar graphs. Even though our shortest circularpath algorithm only guarantees to find the optimum for planar graphs, it yields a good approximation even for almost planar graphs like grid graphs with nodes connected to their eight-neighborhood. The approximation is better when the width of the corridor is small compared to the cycle length. In fact, we use the approximation for non-planar graphs in the tracking step. Since the typical cycle length is about 100 times longer than the corridor width, the optimal solution is usually obtained for the non-planar graph in the tracking step.

### 15.1.7 Chapter 12: physical camera-parameter extraction

Part III of the thesis commenced with Chapter 12 by describing an algorithm to convert the projective motion parameters into the physically meaningful absolute camera rotation-angles and the focal length. The difference to previous camera-autocalibration algorithms is that the input data comprises the projective motion parameters as they are stored for example in the MPEG-4 sprite-motion parameters or the MPEG-7 camera-motion descriptors. This enables to obtain the physical parameters of coded motion data without access to the original image data. Note that this is not possible with previous autocalibration algorithms since they are usually based on a bundle-adjustment technique, applied to the detected feature-points. Another speciality of our algorithm is that it also applies the multi-sprite technique to enable the processing of unrestricted rotational camera motion.

The current algorithm applies a global optimization over all available frames. This leads to a high-accuracy calibration, but it introduces long
processing delays. An interesting future research topic is to find an online algorithm that does not introduce large delays. The main problem is that a significant change of rotation angle is required to obtain the parameters with sufficient accuracy. Especially the estimation of the focal length is numerically unstable for small rotation angles.

### 15.1.8 Chapter 13: camera calibration for sport videos

A different kind of physical-parameter estimation is described in Chapter 13. For applications like sport-video analysis, the position of the players on the screen has to be translated into coordinates in the real-world in order to derive a semantic meaning. For this special application, we have proposed a new algorithm that employs a model of the arrangement of lines on the playing field to establish a correspondence between playing field lines in the image and the position of lines in the real-world. The advantage of our algorithm is that it is not affected by secondary features, such as the court color, varying illumination, or even significant occlusions.

## Calibration with insufficient visual markers

A remaining limitation of the calibration is that, similar to the globalmotion estimation, at least four line-correspondences have to be established. In situations where only a small part of the playing field is visible (e.g., only a T-junction of two lines on a soccer field), insufficient information is available for a calibration. This is a fundamental problem of any calibration algorithm and it can only be solved by further restricting the camera model. A full set of camera parameters involves seven parameters: three rotation angles, three for the camera position, and the focal length. However, the cameras are usually mounted at a fixed position, such that only four parameters remain per frame. Additionally, three parameters for the position are unknown but constant during the sequence. It is an interesting topic to investigate if a camera calibration can be developed that uses the restricted set of parameters when insufficient are available. In fact, Hayet et al. recently proposed an extension to our calibration algorithm that further restricts the number of camera parameters [88]. Additionally, a dynamic model for camera motion could be employed to extrapolate the camera motion, in case that insufficient calibration information can be obtained from the image.

Chapter 15. Conclusions

### 15.1.9 Chapter 14: floor plans from panoramic images

Panoramic background images from rotating cameras are usually visualized in cylindrical panoramic images. This is inconvenient for the viewer since the image shows the scene background from any direction around the camera at once. To provide a better orientation for the viewer, a new algorithm was described in Chapter 14. This algorithm reconstructs the shape of rooms or even complete floor plans from the panoramic images, in which the user has previously marked the room corners.

Note that the floor-plan reconstruction is conceptually comparable to the camera calibration for sport sequences. In both approaches, a model of the observed object is used to connect the real-world geometry with the observed images. The difference is that for the analysis of sport videos, the model was fixed, whereas for the floor-plan reconstruction, the camera positions and the shape of the model are both estimated simultaneously.

### 15.2 Explicit vs. implicit models

In this thesis, several approaches for video-object segmentation have been presented. Common to most of these techniques is that they apply explicit models for describing objects or the camera. Even in the generic segmentation system described in the first part of the thesis, the objects are effectively defined with a negative model: everything that differs from the background should be foreground. Hence, the background synthetization defines the foreground objects. Since the background is derived from the video itself, the approach works only if there is enough video data available to construct a correct background model. Switching from a background model that is derived from the sequence itself to an explicitly-defined model, like the graph-based object models, enables a successful segmentation on much shorter videos or even single frames, since no model information has to be derived from the input video.

The choice for a particular approach depends on the typical video content. For a surveillance-type input, the segmentation system proposed in Part I proves already very robust. In more general video sequences like movies, this approach is not satisfactory since single scenes are too limited in length to derive suitable background models. Moreover, movie sequences often comprise non-rotational camera motion, such that a background sprite cannot be created.

To enable video-object segmentation for general video sequences, we consider two techniques to be of principal importance. First, we require a generalized background model that also supports translatorial camera motion. Probably, this means that a true 3-D model of the background is
required. Second, we need a good workable definition of an object model in order to extract objects in scenes with incomplete background information. The problem of defining a suitable object model is probably the most ambitious in segmentation.

### 15.3 Future of segmentation

As elaborated in the introduction of the thesis, there are numerous applications that can be enhanced with segmentation. On the other hand, segmentation will remain a challenging problem for the foreseeable future, and it is not clear yet whether it can ever be considered solved. Currently, it appears more probable that instead of a general solution to the segmentation problem, a variety of specialized algorithms will be developed for specific applications. It should also be considered that many applications do not require exact segmentation masks, which might be too difficult to compute. Instead, the detection of video-objects and extraction of some of their features can often be sufficient (e.g., the motion path of players and the ball in a sport game is usually sufficient for the analysis). The most important application requiring accurate segmentation masks are videoediting applications, but for these applications, the requirements are so high that segmentation will probably stay a semi-automatic tool to ease the editing (Intelligent Scissors tools, alpha-channel estimation [26], or reconstruction of 3-D models from still pictures, like for the preservation of cultural heritage).

Many applications, like video-archive databases and compression systems, have to deal with general video sequences, for which an automatic segmentation does not seem tractable. However, for these systems, an accurate segmentation is also less important, as it is only a tool to support high-level database queries or enable higher compression factors. In this sense, segmentation-like techniques can be applied for the analysis, but to yield a high robustness, the system should not depend on an error-free segmentation. For example, in 3-D compression systems, the separation into independently-coded objects can help in interpolating new views, but there are too many special cases (e.g., reflecting surfaces) that should be processed with content non-adaptive techniques, as one can never rely on a successful segmentation. In video databases, special object-detection algorithms for specific frequently-occuring queries (e.g., face-recognition) might replace a generic segmentation-based analysis.

For applications concentrating on sequences of a specific domain, like specialized video databases, content analysis, or medical image-processing, more detailed object models will become increasingly important. In the
near future, there will also be an increasing merge between image-analysis techniques and computer-generated graphics or visualizations. Early applications are appearing in the movie industry (e.g., motion capture) and sports analysis for augmented event-visualization.

## Part IV

## Appendices

## Appendix <br> A

# Video-Summarization with Scene Preknowledge 

## A. 1 Introduction

For long video sequences, it is not easy to quickly get an overview of the content. Usually, this means browsing through the video in a fast forward mode to save time. To relieve the user from this searching task, algorithms $[115,104,78]$ have been developed to automatically generate video abstracts, which are short videos extracted from the original to give a good overview about its content. However, often we desire to have a static overview of the video either to print it or to present a list of video sequences that are available in a data-base. In this case, the video content can be summarized with a collection of representative key-frames.

One approach to extract key-frames is to apply a cut detection to separate the video into a large number of shots. Key-frames are obtained by choosing a representative frame of each shot (see [22]). The disadvantage of this approach is that errors in the cut detection are propagated into the key-frame selection process. Furthermore, the number of key-frames is directly coupled to the number of shots and cannot be adjusted by the user. Even if the content changes much within a single shot (consider a camera pan), only a single key-frame is extracted for the shot.

A second technique to find a good set of key-frames is to extract a feature-vector that describes the image content in a low dimensional space.

The features should be chosen such that images from the same scene are mapped to similar feature-vectors. Usually, some kind of color histogram is used to describe the rough image content. Feature-vectors belonging to the same scene can be grouped with clustering algorithms. Finally, some of the feature-vectors are selected as key-frames.

Various work has been carried out to find key-frames which show especially important scenes, or which are very charateristic for the input video. However, the evaluation, which scenes are representative is very subjective and on a high semantic level, that cannot easily be automatized. In fact, our experiments have shown that it is difficult to distinguish summaries that were generated with sophisticated algorithms from summaries that simply take pictures randomly or at regular intervals.

However, there are four kinds of key-frames that should be avoided, since they show an obviously bad selection of images:

- Repetitions. Several images that show almost similar input images. These do not provide any new information. Usually, repetitions can be avoided by a well designed clustering process.
- Cross-fades, wipes. The transition between scenes is usually editied as a short cross-fade between the two scenes. Pictures from these transitions show a mixture of two scenes and are not meaningful when viewed statically. Unfortunately, these frames are often selected in a clustering process, because their feature-vector is clearly different from both scenes and as such, the algorithm treats the transition like a scene of its own.
- Black or white frames. Sometimes there are ranges where the video is simply black or white to have a pause between two scenes. These kind of pictures with no information should be excluded from the summary. This class of undesired images also includes pictures that are too bright because of some flashlights.
- Uninteresting scenes. For some applications, it may be known beforehand that some of the scenes are not essential and they should be excluded from the summary. One example are scenes showing the news-speaker or the weather chart in news broadcasts, while we are mainly interested in the reports in between.

This appendix presents a new clustering-based algorithm, providing a solution to these problems. The problem of frames selected out of transitions is solved by a two-stage clustering process. The first stage provides a soft form of shot separation, determining periods of stable image content
with good key-frame candidates. These candidates are then used in the second clustering stage to select the final set of key-frames. Image content which is not desired in the summary can be explicitly excluded by providing domain-knowledge in form of sample images of shots to exclude. This aspect is solved by modifying the clustering step to circumvent the building of clusters for these shots.

## A. 2 Summarization algorithm

Our algorithm is composed of three steps which operate from low-level features to semantically more meaningful data structures. As a first step, a small set of features is extracted from each input frame. These feature vectors are subsequently used to determine similarity between frames. The second step then groups time consecutive feature vectors to small segments. We define a segment as a short period in the video sequence (usually even smaller than a shot) such that the content in the segment is as static as possible. More specifically, no cut should be present in a segment. The third step combines the segments to clusters such that as much as possible of the input video content is covered by the clusters. Domain-knowledge is integrated by inhibiting the building of clusters in areas of the feature-space which are known to be irrelevant.

## A.2.1 Feature extraction

For each input frame $n$, a feature vector $f_{n}$ is extracted. The subsequent steps of our algorithm work on arbitrary feature vectors. Thus, a variety of features can be used, provided that an appropriate distance measure $\left\|f_{a} ; f_{b}\right\|$ can be defined which corresponds to visual similarity.

For our implementation, we have chosen to use quantized luminance histograms as feature vectors $f_{n}=\left(h_{1}, \ldots, h_{m}\right)$. We are using two different distance measures for the segment positioning and clustering steps. In segment positioning, the sum of absolute difference measure (SAD) is used, which is defined as

$$
\begin{equation*}
\left|\left|f_{a} ; f_{b} \|_{S A D}=\sum_{i=1}^{m}\right| f_{a}(i)-f_{b}(i)\right| . \tag{A.1}
\end{equation*}
$$

For the clustering step, the Earth-Mover's Distance (EMD) is used. This measure has been used by several authors in the context of image retrieval from large databases; see [160] for an in-depth description. In the one-
dimensional case, the EMD can be determined efficiently as

$$
\begin{equation*}
\left\|f_{a} ; f_{b}\right\|_{E M D}=\sum_{x=1}^{m}\left|\sum_{i=1}^{x} f_{a}(i)-f_{b}(i)\right| . \tag{A.2}
\end{equation*}
$$

The reason for using two different distance measures is that the two steps operate on different time-scales. In short periods of time, the image contents varies less. Hence, the more sensitive SAD measure is used to accurately segment slow transitions. From a global perspective, shots with large distances in time can show large differences even when the semantic content is comparable. Therefore, the more liberal EMD is more appropriate for the high-level clustering step.

## A.2.2 Determining segment boundaries

Video sequences may contain gradual transition effects like fades and wipes between shots. It is desired that video frames from these transitions are not present in the final video summary. However, when two subsequent shots are grouped into the same cluster, clustering algorithms usually tend to select the transition frames as cluster centers because these features are mixtures of the features from both shots. To avoid the selection of those frames, we split the input video into short segments of about 4 seconds length. The objective of the first clustering step is to position the segment boundaries such that they are favourably positioned at cuts and within transitions. The intention is to obtain small segments of video with almost homogeneous content (see Fig. A.1). Consequently, the frame in the middle of each segment will be a good key-frame candidate.

Our algorithm for positioning the segment boundaries is motivated by the time-constrained clustering technique described in [153]. However, that paper used the clustering technique to generate hierarchical summaries of existing key-frames, whereas we are using it for the low-level placement of segment boundaries.

Let $p_{i}$ be the first frame of segment $i$. To determine the positions $p_{i}$, the total sum of inhomogeneity over all $N$ segments is minimized. The


Figure A.1: The input video is divided into a large number of segments ( $N$ ). Segment boundaries are positioned such that they coincide with cuts or that they are placed in the middle of transitions between shots (shown as shaded areas).
inhomogeneity of a segment $i$ is determined by summing up the distances between all frames in the segment and the mean feature vector $s_{i}$ of the segment with

$$
\begin{equation*}
s_{i}=\frac{1}{p_{i+1}-p_{i}} \sum_{n \in\left[p_{i} ; p_{i+1}\right)} f_{n} . \tag{A.3}
\end{equation*}
$$

The positions of the segment boundaries $p_{i}$ are chosen to minimize the total segment inhomogeneities:

$$
\begin{equation*}
\min _{p_{1}, \ldots, p_{N-1}} \underbrace{\sum_{i \in[0 ; N)} \underbrace{}_{\text {inhomogeneity of segment } i} \sum_{n \in\left[p_{i} ; p_{i+1}\right]}\left\|s_{i} ; f_{n}\right\|_{S A D} .}_{\text {minimized over all segments }} \tag{A.4}
\end{equation*}
$$

If there are enough segments available, the above optimization will place segment boundaries into transitions between shots. In the usual case in which many more segments are available than shots, long shots will be split into several segments. According to the optimization criterion, the segment length will depend on the amount of change in the video. Static parts will be assigned longer segments, while fast changing parts will be split into shorter segments.

Since the computational complexity of an exact optimization would be too high for practical implementations, we are using a time-continuous variant of the $k$-means algorithm for optimization. The algorithm approaches the global optimum by performing many local optimizations as follows (see Fig. A.2):

1. Distribute $p_{i}$ equally spaced over the full length of the video;
2. for all pairs of adjacent segments $\left[p_{i-1} ; p_{i}\right)$ and $\left[p_{i} ; p_{i+1}\right)$ set $p_{i}$ to

3. repeat step 2 until convergence is reached.


Figure A.2: Local optimization step. The boundary between two adjacent segments is moved to a position such that the homogeneity of the segments on both sides is maximized.

Usually, the solution found does not exactly correspond to the global optimum, however at cuts, the segment boundaries are positioned reliably between shots. This property makes the solution sufficient for later steps of the algorithm. From each segment $i$, the frame at the middle of the segment (at position $\left.m_{i}=\left(p_{i}+p_{i+1}\right) / 2\right)$ is taken as the representative frame of that segment and as a later key-frame candidate. Further processing steps operate only on the feature vectors $K=\left\{k_{i}=f_{m_{i}}\right\}$, leading to a significant reduction of computation time compared to clustering algorithms using all input frames.

## A.2.3 Clustering

Our clustering algorithm is based on the approach described in [134, 38], which is outlined in the following. Differing from the algorithm described in the literature, we are using the EMD distance as clustering criterion which results in perceptually more reasonable clusters. Moreover, we observed that an arbitrary initialization of cluster centers sometimes results in bad convergence. Hence, we perform a gradual increase of the number of cluster centers and initiate new centers into areas where new clusters are most likely to be found.

The basic principle is to find a predetermined number of clusters (corresponding to the number of key-frames) such that the dissimilarity of the frames in each cluster is minimized. To define this more formally, let $c_{i}$ be the set of $M$ cluster centers and let $K=\left\{k_{i}\right\}$ be the set of key-frame candidates extracted in the last step of the algorithm. For each cluster center, we define its neighbourhood $N_{c_{i}}$ as:

$$
\begin{equation*}
N_{c_{i}}=\left\{k \in K \mid \forall j:\left\|k ; c_{i}\right\|_{E M D} \leq\left\|k ; c_{j}\right\|_{E M D}\right\}, \tag{A.5}
\end{equation*}
$$

meaning that each feature vector is assigned to the neighbourhood of the nearest cluster-center (according to the EMD-distance).

We say that a set of cluster centers is optimal iff they fulfill

$$
\begin{equation*}
\min _{c_{0}, \ldots, c_{M-1}} \underbrace{\sum_{i \in[0 ; M)} \underbrace{\sum_{s \in N_{c_{i}}}\left\|s ; c_{i}\right\|_{E M D}}_{\text {dissimilarity of cluster } i} .}_{\text {summed over all clusters }} \tag{A.6}
\end{equation*}
$$

The clustering is carried out by a $k$-means algorithm without the timeconsecutiveness constraint used in the last step. Our experiments have shown that the $k$-means algorithm works best when the initial cluster centers are not chosen randomly, but rather added one at a time. Each new cluster center is initialized to the $k_{i}$ with the largest distance to any existing cluster center. The overall clustering algorithm can be summarized as:

1. Set $c_{1}$ to a random $k_{i}$ (e.g. $k_{1}$ ), set $n=1$;
2. determine the neighbourhood $N_{c_{i}}$ for all clusters;
3. reassign $c_{i}$ to $c_{i}:=\frac{1}{\mid N_{c_{i}}} \sum_{k \in N_{c_{i}}} k$;
4. continue at step 2 until convergence is reached;
5. if $n=M$ stop the algorithm, else set $n:=n+1$, set $c_{n}=\operatorname{argmax}_{k_{i}} \min _{c_{j}}\left\|k_{i} ; c_{j}\right\|_{E M D}$ and continue at step 2.

For each cluster obtained from the last step, the $k_{i}$ which is nearest to the cluster center is selected as a key-frame. The selected $k_{i}$ are sorted to the correct temporal order, and the input frames corresponding to the $k_{i}$ are composed to the final summary.

## A.2.4 Integration of domain-knowledge

In this section, we modify the clustering step from the last section to insert domain-knowledge about irrelevant video scenes. To prevent these scenes from occuring in the summary, we compute the feature vectors for all scenes to be excluded. These feature vectors can also be provided by the user if he detects an image in the summary that he wants to exclude. After feeding this information back into the algorithm, a new summary can be computed without the undesired scenes. After a period of interactivity with the user, the classes of scenes to be excluded are known to the system, and summaries will only contain the desired scenes.

Exclusion of the scenes is accomplished by introducing the feature vectors $u_{i}$ of uninteresting scenes as additional cluster centers (see Figure A.3).


Figure A.3: Schematic example of the clustering process with integrated domain-knowledge. The lengthy report scene is divided into two separate clusters while the repeated anchorman scenes are combined into a single cluster. Black and white frames are removed by the predefined clustering centers. User defined domain-knowledge has been applied by adding a cluster-center to remove the weather chart scene from the summary.

They are treated the same as the $c_{i}$ with the exception that the position of $u_{i}$ is fixed and that no key-frames will be generated for their clusters. The consequence in the clustering process is that the $u_{i}$ centers grab the feature vectors of scenes near the $u_{i}$ vectors. These vectors will have no influence on the clustering because they are contained in the neighbourhood of a vector $u_{i}$. The total number of generated key-frames will remain the same.

## A. 3 Evaluation

We demonstrate the behaviour of our algorithm with two test sequences. The first is the well-known Foreman sequence. This sequence is 400 frames long and contains no cuts. Its plot is depicted in Figure A.4(a). After showing the speaking man for over half of the sequence, the camera pans to the right and shows a building. Note that even though there are no cuts, our algorithm finds the three most important parts in the video. Algorithms that are based on cut detection fail on this sequence.

(a) Test sequence plot.

(b) Summary (three key-frames).

Figure A.4: Summary of the Foreman sequence.

(a) Test sequence plot.

(b) Without domain-knowledge: $1 \times$ anchorman, $1 \times$ report, $3 \times$ weather chart, $1 \times$ logo.

(c) With knowledge to ignore weather chart: $1 \times$ anchorman, $4 \times$ report, $1 \times$ logo.

Figure A.5: Summary of the last two minutes of a news broadcast.

The second video contains the last two minutes of a news broadcast. Again, the video plot is shown in Figure A.5(a). First, we started our algorithm without any domain-knowledge (Figure A.5(b)). Note that the news anchorman only appears once in the summary even though he appears three times in the input sequence. Since the weather chart contains very different image content, the summary contains three pictures of it but only one picture of the preceding report. Let us now suppose that we are not interested in the weather chart. So we provided some pictures of the chart as domain-knowledge in a succeeding experiment and restarted the algorithm (Fig. A.5(c)). All key-frames of the weather-chart were removed, and additionally, more meaningful key-frames of the news report were generated.

## A. 4 Conclusions

We have described a new algorithm for automatic generation of video summaries. User domain-knowledge about the video-content can be provided to improve the quality of the generated summary. We consider the approach of a modified clustering step superior to specialized filters for excluding undesired frames because our generic approach can be adapted to new application areas by simple user interaction. A topic of further research may be to integrate an algorithm for automatic detection of irrelevant featurevectors.

Finally, the fact that our algorithm does not depend on an accurate cut detection algorithm (known to have difficulties with soft cuts) increases robustness and enables the summarization of video material without scene changes.

Our algorithm has been integrated into the video-database of the $\mathrm{L}^{3}$ project (LifeLong Learning) for learning-videos (see Figure A.6) and the ECHO project (European CHronicles On-line) to generate abstracts for four major national audio-visual archives (Italy, France, the Netherlands, Switzerland). The $L^{3}$ video-database application is based on the opensource Scientific Image Data-Base (SIDB) project, which we extended for video-sequences.

(a) Summary display of a video sequence.

(b) List of available sequences in the data-base.

Figure A.6: Integration of the video abstracting algorithm in a web-based video database application.

Appendix A. Video-Summarization with Scene Preknowledge

# Efficient Computation of Homographies From Four Correspondences 

The usual way to compute the parameters of a projective transform from four point coordinates is to use the inhomogeneous formulation of the projective transform (Eq. (2.11)). Using four point-correspondences $\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}$, we can set up an equation system (Eq. (3.2)) to solve for the homography matrix $\mathbf{H}$. However, this requires the solution of an $8 \times 8$ equation system.

## Efficient algorithm

An algorithm to obtain these parameters requiring only the inversion of a $3 \times 3$ equation system is as follows. From the four point-correspondences $\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}$ with $(i \in\{1,2,3,4\})$, compute $\mathbf{h}_{1}=\left(\mathbf{p}_{1} \times \mathbf{p}_{2}\right) \times\left(\mathbf{p}_{3} \times \mathbf{p}_{4}\right), \mathbf{h}_{2}=$ $\left(\mathbf{p}_{1} \times \mathbf{p}_{3}\right) \times\left(\mathbf{p}_{2} \times \mathbf{p}_{4}\right), \mathbf{h}_{3}=\left(\mathbf{p}_{1} \times \mathbf{p}_{4}\right) \times\left(\mathbf{p}_{2} \times \mathbf{p}_{3}\right)$. Also compute $\hat{\mathbf{h}}_{1}, \hat{\mathbf{h}}_{2}, \hat{\mathbf{h}}_{3}$ using the same principle from the points $\hat{\mathbf{p}}_{i}$. Now, the homography matrix H can be obtained easily from

$$
\mathbf{H} \cdot\left[\begin{array}{lll}
\mathbf{h}_{1} & \mathbf{h}_{2} & \mathbf{h}_{3}
\end{array}\right]=\left[\begin{array}{lll}
\hat{\mathbf{h}}_{1} & \hat{\mathbf{h}}_{2} & \hat{\mathbf{h}}_{3} \tag{B.1}
\end{array}\right],
$$

which only requires the inversion of the matrix $\left[\begin{array}{lll}\mathbf{h}_{1} & \mathbf{h}_{2} & \mathbf{h}_{3}\end{array}\right]$.

Correspondences

## Proof

The validity of the efficient algorithm can be proven as follows.
First, it can be shown easily that

$$
\begin{equation*}
\mathbf{u} \times(\mathbf{v} \times \mathbf{w})=\left(\mathbf{w}^{\top} \mathbf{u}\right) \mathbf{v}-\left(\mathbf{v}^{\top} \mathbf{u}\right) \mathbf{w} \tag{B.2}
\end{equation*}
$$

Furthermore, it is known that for the triple scalar product $[\mathbf{u}, \mathbf{v}, \mathbf{w}]=$ $\mathbf{u}^{\top}(\mathbf{v} \times \mathbf{w})$ it holds that $[\mathbf{u}, \mathbf{v}, \mathbf{w}]=\operatorname{det}(\mathbf{u}, \mathbf{v}, \mathbf{w})$. Because $\operatorname{det}(\mathbf{A B})=$ $\operatorname{det}(\mathbf{A}) \cdot \operatorname{det}(\mathbf{B})$, it also holds that

$$
\begin{equation*}
[\mathbf{H u}, \mathbf{H v}, \mathbf{H w}]=\operatorname{det}(\mathbf{H}) \cdot[\mathbf{u}, \mathbf{v}, \mathbf{w}] . \tag{B.3}
\end{equation*}
$$

If we now compare

$$
\begin{align*}
\mathbf{H}((\mathbf{a} \times \mathbf{b}) \times(\mathbf{c} \times \mathbf{d})) & =\mathbf{H}\left(\mathbf{d}^{\top}(\mathbf{a} \times \mathbf{b}) \mathbf{c}-\mathbf{c}^{\top}(\mathbf{a} \times \mathbf{b}) \mathbf{d}\right) \\
& =\mathbf{H}([\mathbf{d}, \mathbf{a}, \mathbf{b}] \mathbf{c}-[\mathbf{c}, \mathbf{a}, \mathbf{b}] \mathbf{d})  \tag{B.4}\\
& =\operatorname{det}(\mathbf{d}, \mathbf{a}, \mathbf{b}) \mathbf{H} \mathbf{c}-\operatorname{det}(\mathbf{c}, \mathbf{a}, \mathbf{b}) \mathbf{H d}
\end{align*}
$$

with

$$
\begin{align*}
(\mathbf{H a} \times \mathbf{H b}) \times(\mathbf{H c} \times \mathbf{H d}) & =\mathbf{H d}^{\top}(\mathbf{H a} \times \mathbf{H b}) \mathbf{H} \mathbf{c}-\mathbf{H c} \mathbf{c}^{\top}(\mathbf{H a} \times \mathbf{H b}) \mathbf{H d} \\
& =[\mathbf{H d}, \mathbf{H a}, \mathbf{H} b] \mathbf{H c}-[\mathbf{H c}, \mathbf{H a}, \mathbf{H b}] \mathbf{H d} \\
& =\operatorname{det}(\mathbf{H})(\operatorname{det}(\mathbf{d}, \mathbf{a}, \mathbf{b}) \mathbf{H} \mathbf{c}-\operatorname{det}(\mathbf{c}, \mathbf{a}, \mathbf{b}) \mathbf{H d}), \tag{B.5}
\end{align*}
$$

we see that

$$
\begin{equation*}
\mathbf{H}((\mathbf{a} \times \mathbf{b}) \times(\mathbf{c} \times \mathbf{d}))=\frac{1}{\operatorname{det}(\mathbf{H})}((\mathbf{H a} \times \mathbf{H} \mathbf{b}) \times(\mathbf{H} \mathbf{c} \times \mathbf{H d})) . \tag{B.6}
\end{equation*}
$$

Considering again the problem to compute the parameters of the homography transform, we can state the problem as finding the matrix $\mathbf{H}$ such that

$$
\begin{equation*}
\mathbf{H} \mathbf{p}_{1}=c_{1} \hat{\mathbf{p}}_{1}, \quad \mathbf{H} \mathbf{p}_{2}=c_{2} \hat{\mathbf{p}}_{2}, \quad \mathbf{H} \mathbf{p}_{3}=c_{3} \hat{\mathbf{p}}_{3}, \quad \mathbf{H} \mathbf{p}_{4}=c_{4} \hat{\mathbf{p}}_{4}, \tag{B.7}
\end{equation*}
$$

where $c_{i}$ are unknown constants that provide suitable scaling for the homogeneous coordinates. These four matrix equations have $9+4=13$ unknowns, but since they give only $3 \times 4=12$ constraints, any scaled version of $\mathbf{H}$ is a solution.

Since the actual value of the constants $c_{i}$ do not matter, the trick is to reduce the equations such that the $c_{i}$ are removed to only a single scaling


Figure B.1: Location of the three points $\mathbf{h}_{1}, \mathbf{h}_{2}, \mathbf{h}_{3}$.
constant $\lambda$ that we can choose arbitrarily. This can be achieved by considering the product $\left(\mathbf{p}_{1} \times \mathbf{p}_{2}\right) \times\left(\mathbf{p}_{3} \times \mathbf{p}_{4}\right)$ and all permutations up to a sign change, which gives us the vectors $\mathbf{h}_{1}, \mathbf{h}_{2}, \mathbf{h}_{3}$ which were defined previously. But because of Eq. (B.6),

$$
\begin{align*}
\mathbf{H h}_{1} & =\mathbf{H}\left(\left(\mathbf{p}_{1} \times \mathbf{p}_{2}\right) \times\left(\mathbf{p}_{3} \times \mathbf{p}_{4}\right)\right) \\
& =\frac{1}{\operatorname{det}(\mathbf{H})}(\mathbf{H a} \times \mathbf{H} \mathbf{b}) \times(\mathbf{H c} \times \mathbf{H d}) \\
& =\frac{1}{\operatorname{det}(\mathbf{H})}\left(\left(c_{1} \hat{\mathbf{p}}_{1} \times c_{2} \hat{\mathbf{p}}_{2}\right) \times\left(c_{3} \hat{\mathbf{p}}_{3} \times c_{4} \hat{\mathbf{p}}_{4}\right)\right)  \tag{B.8}\\
& =\underbrace{\frac{c_{1} c_{2} c_{3} c_{4}}{\operatorname{det}(\mathbf{H})}}_{\lambda} \cdot\left(\left(\hat{\mathbf{p}}_{1} \times \hat{\mathbf{p}}_{2}\right) \times\left(\hat{\mathbf{p}}_{1} \times \hat{\mathbf{p}}_{2}\right)\right) \\
& =\lambda \cdot \hat{\mathbf{h}}_{1},
\end{align*}
$$

where we can set $\lambda=1$, since any scaled version of $\mathbf{H}$ is a valid solution. Similar, we get $\mathbf{H h}=\hat{\mathbf{h}}_{2}, \mathbf{H h}_{3}=\hat{\mathbf{h}}_{3}$. These three equations can be combined into one equation system, giving Eq. (B.1).

It is interesting to note that the three points correspond to the intersection points of the opposite sides and the diagonals of a quadrilateral, made from the four points $\mathbf{p}_{i}$ (remember that the cross product of two points defines a line, and the cross product of two lines defines a point). This is visualized in Figure B.1. Clearly, the inverse is not possible, i.e., the homography cannot be determined unambiguously from just these three points.

## Acknowledgement

The author wants to thank Denis Zorin for providing the algebraic proof of this technique.

Appendix B. Efficient Computation of Homographies From Four

It would be possible to describe everything scientifically, but it would make no sense; it would be without meaning, as if you described a Beethoven symphony as a variation of wave pressure. (Albert Einstein)


## Robust Motion Estimation with LTS and LMedS

In Chapter 4, we employed the RANSAC algorithm to calculate globalmotion parameters for a set of point-correspondences between two images. A robust estimation algorithm is required for this problem, since the input is a mixture of foreground object motions and the camera motion. Additionally, the input data is contamined with erroneous correspondences.

One disadvantage of the RANSAC algorithm is its dependence on the threshold $\epsilon$, which decides on how close a point-correspondence must be to the computed model to be considered an inlier. To relieve from the dependency on this threshold, modifications of the RANSAC algorithm have been proposed that do not require an explicit inlier threshold. We will discuss two algorithms, which are based on a common principle: Least-Median-of-Squares (LMedS), and Least-Trimmed-Squares (LTS). Both algorithms are similar to RANSAC with the only exception that Steps 3 and 4 are modified. Since LMedS and LTS are very similar, we will describe them together in the following.

We start again by drawing a random subset of input data to compute a motion model candidate $\mathbf{H}$. After this, we also compute the residuals between all input correspondences and the model $\mathbf{H}$. However, the difference is that we do not mark the data with high residuals as outliers, but that we sort the input data according to increasing residual errors. More
specifically, let again $\mathcal{C}=\left\{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i}\right\}$ be the input correspondences, then we impose an ordering $r(i)$ on the input data such that

$$
\begin{equation*}
d\left(\hat{\mathbf{p}}_{i}, \mathbf{H p}_{i}\right) \leq d\left(\hat{\mathbf{p}}_{k}, \mathbf{H} \mathbf{p}_{k}\right) \leftrightarrow r(i) \leq r(k) . \tag{C.1}
\end{equation*}
$$

The LMedS algorithm repeats the random sampling process and selects the motion model, for which the median of the residuals

$$
\begin{equation*}
d\left(\hat{\mathbf{p}}_{m}, \mathbf{H} \mathbf{p}_{m}\right) \quad \text { with the median position } \quad m=o(|\mathcal{C}|) / 2 \tag{C.2}
\end{equation*}
$$

is lowest. Under the assumption that at least half of the feature-points will be moving according to the global motion, LMedS will select the correct global motion model.

The LTS algorithm [159] computes the same ordering $r(i)$ over the input data. However, not only the median of the residual errors is considered to select the best transform, but a fixed percentage of the best ranked input data. This subset of the input data is considered the inliers, even though they might only be part of the real inlier data. On this inlier data, an additional compaction step (CSTEP) is carried out. This is a refinement of the estimated motion model on the selected inlier set, comparable to Step 5 of the RANSAC algorithm. The CSTEP can even be applied several times where the set of inliers is adapted to the refined motion model in each iteration. The complete LTS algorithm can then be summarized as follows.

1. Draw a random subset $\mathcal{S}$ from the input data as in the RANSAC algorithm.
2. Compute the motion model $\mathbf{H}_{0}$ based on the drawn subset $\mathcal{S}$.
3. Rank the input data to increasing residual errors and determine the inlier set $\mathcal{I}_{1}=\left\{c_{r(k)}|k \leq p \cdot| \mathcal{C} \mid\right\}$ so that it includes a fixed fraction $p$ of the correspondences $c_{i} \in \mathcal{C}$ with lowest error. Compute a refined motion model $\mathbf{H}_{1}$ using a least-squares approximation to $\mathcal{I}_{1}$. Let the total residual error be $Q_{1}=\sum_{\mathbf{p}_{i} \leftrightarrow \hat{\mathbf{p}}_{i} \in \mathcal{I}_{1}} d\left(\hat{\mathbf{p}}_{i}, \mathbf{H p}_{i}\right)$. Based on the new motion model, compute a new set of inliers $\mathcal{I}_{2}$ and a new motion model $\mathbf{H}_{2}$ and residual $Q_{2}$. Iterate this process until $Q_{k-1}=Q_{k}$.
4. Repeat Steps 1-3 $N$ times and choose the motion model, for which $Q_{k}$ was lowest.

Comparing the LTS algorithm with RANSAC, we see that no threshold $\epsilon$ is required any more, but on the other hand, we have to specify a percentage $p$ which is equal to the minimum fraction of inlier data. The LMedS does not have a comparable parameter, but since the median of the residuals is taken, it implicitly assumes that the inlier fraction is at least $50 \%$.


Figure C.1: Examples of inlier classification for (a) RANSAC, and (b) LTS. Inliers are marked in black, outliers are drawn in white.

However, it is obvious that the LMedS algorithm can be generalized for other inlier fractions by not considering the median residual but a different position in the ranked list of residuals.

## Evaluation

Compared to the RANSAC algorithm, LTS and LMedS did not show clear advantages. In fact, the accuracy of the obtained motion parameters were even slightly worse than with RANSAC. For LTS, this can be explained as follows. We have seen in the evaluation of the RANSAC algorithm that the accuracy improves with an increasing number of refinement iterations, where the image area which is supported by inliers gets larger in each step. Note that this iterative refinements are comparable to the compaction steps of LTS. However, LTS only includes a fixed percentage of best fitting data into the refinement. The effect is that the covered set of the inlier data does not grow larger than this fraction and the final model estimate is based on a smaller set of input samples. It is a good model for this subset, but not for all the inliers. A similar reasoning also holds for LMedS.

An example comparison between RANSAC and LTS is depicted in Figure C.1. It is clearly visible that RANSAC achieves an accurate separation of background motion and foreground motion. The LTS algorithm does not give this clear separation, but all selected inlier vectors are part of the background motion. Consequently, LTS also successfully locked to the correct background motion.

Appendix C. Robust Motion Estimation with LTS and LMedS

All science is either physics or stamp collecting. (Ernest Rutherford)

## Appendix <br> 

## Additional Test Sequences

These test-sequences were recorded from DVB broadcasts and downsampled to CIF resolution (one field of the interlaced sequence was extracted and horizontally reduced by a factor of two).

## Roma - 210 frames


frame 1

frame 100

frame 200

Slow horizontal camera pan, no foreground objects. See also Figure 12.11 and Figure 4.12.

Opera4-90 frames


Vertical camera pan. No foreground objects and very low texture. See also Figure 4.5.

## Rail - 140 frames


frame 1

frame 70

frame 140

Complex camera rotation at medium speed. No foreground objects. See also Figures 4.4 and 6.18 to 6.20 .


Zoom in at right side, rotate to the left, and zoom out again. No foreground objects. See also Figures 12.14 and 12.15 .

## Hurdles - 200 frames


frame 1

frame 100

frame 200

Camera is tracking the athlets. Completely textured image at the beginning, but large green areas at the end of the sequence. See also Figure 8.15.

## Appendix <br> 

# Color Segmentation Using Region Merging 

## E. 1 Introduction

Color segmentation is the process of segmenting an image into homogeneouslycolored regions. In most cases, a color segmentation alone cannot yield a semantically good segmentation, but it can be useful as a preprocessing step to transform an image into a set of regions that are processed further in successive stages. We apply region merging in the automatic segmentation step of our object-model detection algorithm (Chapter 9). Furthermore, in Chapter 10, we integrate object-model knowledge and motion information in the region-merging algorithm to obtain a semantically meaningful result.

In this appendix, we briefly introduce the region-merging algorithm and present several merging criteria and evaluate their performance in terms of noise robustness and subjective segmentation quality. Furthermore, we introduce a new merging criterion yielding a better subjective segmentation quality, and propose to change to merging criterion during processing to further increase the overall robustness and segmentation quality.

## E.1.1 The region-merging algorithm

The objective of region merging is to group image pixels to regions which are similar with respect to a predetermined criterion. The algorithm proceeds by sequentially merging the two most similar neighbouring regions.


Figure E.1: Two steps of a region-merging process. Thicker edges represent more similar regions.

The merging process stops when no more regions are found with sufficient similarity, or the minimum number of regions is reached.

Region merging can be viewed as an algorithm working on a graph, where the nodes represent regions of pixels and the edges indicate a neighbouring relationship. We assign edge weights to the edges to represent the dissimilarity between adjoining regions. Let $P=\left\{p_{i}\right\}$ be the set of pixels in the input image with corresponding luminance $I\left(p_{i}\right)$. Furthermore, let a neighbouring-relation $n\left(p_{i}, p_{j}\right)$ be true iff $p_{i}$ and $p_{j}$ are neighbours; however note that $n\left(p_{i}, p_{i}\right)=$ false.

The input of the region-merging algorithm is a set of regions $R=\left\{r_{i}\right\}$ with $r_{i} \subset P, \bigcup r_{i}=P$ and $r_{i} \cap r_{j}=\emptyset$ for $i \neq j$. The initial set of regions can be obtained in a several different ways. The regions can be the result of a preceding segmentation step such as watershed segmentation, they can be chosen arbitrarily (e.g. blocks of fixed size), or in the extreme case, each input pixel can be considered as a separate region.

The algorithm first builds a neighbourhood graph $G=(R, E)$ with edges $E=\left\{\left(r_{i}, r_{j}\right) \mid \exists p_{k} \in r_{i}, p_{l} \in r_{j}: n\left(p_{k}, p_{l}\right)=\right.$ true $\}$. Additionally, we define an edge weight $w$ on the edges $w: E \rightarrow \mathbb{R}$ which describes a measure of dissimilarity of the regions connected by the edge. The definition of the edge weights (i.e. the merging criterion) is the crucial part of the algorithm, directly affecting the quality of the segmentation result.

Region merging is a greedy algorithm following the intuitive process to continuously merge the two most similar regions into a single region. Merging stops when the lower bound of regions $\# r_{\text {min }}$ is reached or the minimum edge weight exceeds a threshold $w_{\max }$. The algorithm is outlined in Algorithm 3 and illustrated in Figure E.1.

```
Algorithm 3 Basic merging algorithm
    while \(|R|>\# r_{\text {min }}\) do
        \(e_{\text {min }}=\left(r_{j}, r_{k}\right) \leftarrow \underset{e \in E}{\operatorname{argmin}} w(e)\)
        if \(w\left(e_{\min }\right)>w_{\max }\) then
            STOP
        else
            Join regions \(r_{n} \leftarrow r_{j} \cup r_{k}\)
            Update edges \(E \leftarrow E \cup\left(E_{\text {new }}=\left\{\left(r_{n}, r_{i}\right) \mid\left(r_{j}, r_{i}\right) \in E \vee\left(r_{k}, r_{i}\right) \in E\right\}\right)\)
            Remove old edges \(E \leftarrow E \cap\left\{R \backslash\left\{r_{j} ; r_{k}\right\}\right\} \times\left\{R \backslash\left\{r_{j} ; r_{k}\right\}\right\}\)
            Remove regions \(r_{j}\) and \(r_{k} \quad R \leftarrow R \backslash\left\{r_{j} ; r_{k}\right\}\)
            for all \(e \in E_{\text {new }}\) do
                Update edge weight \(w(e)\)
            end for
        end if
    end while
```


## E. 2 Merging criteria

A merging criterion consists of two parts: a region model, describing each image region with a set of features, and a dissimilarity measure, defining a metric on the features of the region model. The range of possible region models reaches from simple models like uniform luminance up to texture, shape or motion parameters. In the following, we will concentrate on low-level features which are applied at early stages of the algorithm. Furthermore, we only consider greyscale images. However, all presented critera can be readily generalized to work on color images.

The better a region model matches the real image-data, the longer the minimum edge-weights remain small and the steeper is the relative increase in region dissimilarity as soon as the segmentation has reached its final state. This makes the segmentation process more robust to the selection of the fixed threshold for the stopping condition.

## E.2.1 Mean luminance difference

The simplest region model is to describe each region $r_{i}$ by its mean luminance $\mu_{i}$. A straightforward possibility to define a dissimilarity measure for this model is to use the squared difference, from now on referred to as the Mean-criterion

$$
\begin{equation*}
w_{i j}^{M}=\left(\mu_{i}-\mu_{j}\right)^{2} \tag{E.1}
\end{equation*}
$$

## E.2.2 Ward's criterion

Another measure which operates on the mean-luminance model is the Wardcriterion [193]. The idea is to consider the model error for a region $r_{i}$, defined as $\mathcal{E}_{i}=\sum_{p \in r_{i}}\left(I(p)-\mu_{i}\right)^{2}$. The dissimilarity associated with a pair of regions is defined as the additional total error that is introduced by merging the two regions: $w_{i j}^{W}=\mathcal{E}_{i j}-\mathcal{E}_{i}-\mathcal{E}_{j}$ (with $\mathcal{E}_{i j}$ being the error after a hypothetical merge of $r_{i}$ and $r_{j}$ ). After elementary simplifications, this can be expressed as

$$
\begin{equation*}
w_{i j}^{W}=\frac{\left|r_{i}\right| \cdot\left|r_{j}\right|}{\left|r_{i}\right|+\left|r_{j}\right|}\left(\mu_{i}-\mu_{j}\right)^{2} . \tag{E.2}
\end{equation*}
$$

## E.2.3 Mean/Ward mixture

As will become clear in the following section, neither the Mean-criterion nor the Ward-criterion produce a subjectively approporiate segmentation. A better criterion may be a compromise between the characteristics of Mean and Ward. For this reason, we introduce the geometrical mean of both criteria $\left(w_{i j}^{G}=\left(w_{i j}^{M} \cdot w_{i j}^{W}\right)^{1 / 2}\right)$ as a new Mean-Ward criterion. Since the absolute value of the criterion is not important, the square-root can be ignored, resulting in

$$
\begin{equation*}
w_{i j}^{G}=\frac{\left|r_{i}\right| \cdot\left|r_{j}\right|}{\left|r_{i}\right|+\left|r_{j}\right|}\left(\mu_{i}-\mu_{j}\right)^{4} . \tag{E.3}
\end{equation*}
$$

## E.2.4 Linear-luminance model

Because of illumination effects, natural images seldomly consist of completely homogeneous regions. Almost all regions that we perceive as homogeneous, contain a small luminance gradient. Therefore, it is sensible to use a region model that is capable of describing slowly varying luminance gradients. A possible region model defines the luminance distribution as $I^{\prime}(x, y)=\alpha+\beta x+\gamma y$ with the three parameters $\alpha, \beta, \gamma$. For each individual region, these parameters are estimated from the image data, using a least-squares approach. Comparable to the Ward-criterion, we define the model error as $\mathcal{E}_{i}^{L}=\sum_{p \in r_{i}}\left(I(x, y)-I^{\prime}(x, y)\right)^{2}$ and the region dissimilarity as

$$
\begin{equation*}
w_{i j}^{L}=\mathcal{E}_{i j}^{L}-\mathcal{E}_{i}^{L}-\mathcal{E}_{j}^{L} . \tag{E.4}
\end{equation*}
$$

## E.2.5 Border criterion

Although the linear-luminance model handles well most regions occurring in natural images, the model has two main drawbacks: it is rather compu-


Figure E.2: Performance of several criteria for an unsharp edge (50 regions remaining).
tationally intensive, and it still cannot handle all cases of small luminance variations. Especially curved surfaces have complicated luminance distributions. Both problems can be circumvented by using the following Border criterion.

Let $B_{i j}=\left\{\left(p_{k}, p_{l}\right)\right\}$ be the set of pairs of pixels along the common boundary between region $r_{i}$ and $r_{j}$ (with $p_{k} \in r_{i}$ and $p_{l} \in r_{j}$ ). We define the Border-criterion as the sum of squared differences along the boundary

$$
\begin{equation*}
w_{i j}^{B}=\frac{1}{\left|B_{i j}\right|} \sum_{\left(p_{k}, p_{l}\right) \in B_{i j}}\left(I\left(p_{k}\right)-I\left(p_{l}\right)\right)^{2} . \tag{E.5}
\end{equation*}
$$

Note that this criterion only considers how the regions fit together along the border, not considering the interior of the region area.

## E. 3 Criteria properties

## E.3.1 General behaviour

Figure E. 2 depicts a detail view of an image containing an unsharp edge. As the image is part of a real-world image, it contains a hardly visible luminance gradient in the "flat" image regions and some camera noise. The image has been segmented independently with the Mean, Ward, and Linear-Luminance criterion until only 50 regions were remaining.

It is easily visible that the Ward criterion favours the removal of small noisy areas instead of combining large, but only slightly different regions. This occurs because the Ward criterion considers the total error and small

Appendix E. Color Segmentation Using Region Merging

| criterion <br> problem class | Mean | Ward | Mean- <br> Ward | Linear- <br> Lum. | Border | Water- <br> shed |
| :--- | :---: | :---: | :---: | :---: | :---: | :---: |
| noise | - | ++ | + | - | - | - |
| blurred edges | - | + | + | + | -- | + |
| double edges | - | + | + | - | - | + |
| illumination | - | -- | + | ++ | ++ | + |
| subjective eval. | + | -- | ++ | + | + | - |
| stopping criterion | -- | + | ++ | + | + | N/A |
| comp. complexity | + | + | + | -- | + | ++ |

Table E.1: Comparison of the performance of different criteria on a number of typical problem classes.
differences in very large regions outweigh larger differences in very small regions. The Mean criterion does not show this effect, because it does not take region size into account. Similarly, the Linear-Luminance criterion can adapt its model to approximate the gradient with sufficient accuracy. Furthermore, it is also capable to model the unsharp edge itself and does not lead to the oversegmentation with many narrow regions, as is the case with the other two criteria.

## E.3.2 Comparison

In natural image segmentation, several classes of commonly occuring difficulties can be identified. The robustness of each criterion ${ }^{1}$ on the problem classes was evaluated and is depicted in Table E.1. In the following, some problem classes are described in more detail.

- Noise. Camera noise has a well visible effect at the beginning of the segmentation process. Dissimilarity measures which are normalized to their region sizes, like Ward's criterion, give superior results, because single noisy pixels introduce no large overall error.
- Blurred edges and double edges. Objects which are out of camera focus appear with blurred edges in the image. This can lead to an oversegmentation into many thin rings around the object boundary. The Linear-Luminance criterion can approximate the blurred edge with a single region if the object boundaries are straight lines. Curved boundaries can be handled by the border criterion.
However, the more general model of the Border criterion has the disadvantage to ignore the pixels inside a region. Thus, it is possible

[^25]for an object to grow along its unsharp border until it is completely merged with the background (see Figure E.4(c)).
Almost all sharp edges in real-world images consist of pixels midway between the colors of the two regions. After a segmentation with a too low threshold, objects seem to have double edges. Because of its tendency to merge small regions, the Ward-criterion can remove these double edges well.

- Illumination effects. Large regions in natural images usually have differing brightness because the lighting is not strictly uniform. If the region model assumes a uniform region color, these areas are split into several pieces (see Fig. E.5). This is not the case if the region model allows brightness gradients, or if only the border between regions is considered.
- Subjective evaluation. As can be seen in Figure E.3, the Ward criterion has a tendency to split large regions into several segments, whereas the Mean criterion removes large regions equally likely as small regions. Figure E.3(d) shows the segmentation using the MeanWard criterion. The result is much more subjectively pleasing as the large regions are preserved, and much of the text is kept. The same effect is shown in the natural image in Figure E.6.


## E. 4 Multi-stage merging

As discussed in the last sections, each criterion shows both advantages and disadvantages. Choosing a single criterion for the complete segmentation process results in a dissatisfactory segmentation. This motivates a multistage approach. A criterion is used as long as it can well handle the current configuration. Afterwards, the criterion is exchanged by another one. By using several stages, the selection of an appropriate threshold in the stopping criterion is not critical. The threshold should be chosen sufficiently low to ensure that control is passed on to the next criterion, before the situation exceeds the capabilities of the criterion's region model. A sequence of criteria that produced good segmentation results was:

1. Ward, removing much of the image noise and eliminating double edges,
2. Mean-Ward, which does the main work, before finally
3. Border merges regions in which illumination effects play a central role.


Figure E.3: Segmentation results for three criteria, 50 regions remaining.


Figure E.4: Effect of applying a watershed presegmentation.

## E.4.1 Applying a watershed presegmentation

Instead of starting the algorithm with single-pixel regions, it is possible to perform a presegmentation with the watershed algorithm on a gradient map of the input image.

This presegmentation has the advantage to considerably reduce the computational complexity, as the watershed transform is a fast algorithm and reduces the amount of input regions. Furthermore, it alleviates the problem of the Border criterion to destroy complete objects having unsharp boundaries (see Figure E.4). The watershed transform splits these blurred areas at the object boundaries along the position of the maximum gradient into only two regions. For this reason, the threshold in the stopping condition for the Border criterion can be set to a lower value.

The disadvantage of applying this presegmentation is that small image structures may be deteriorated or even vanish. Additionally, in the presence of camera noise, smooth edges in the image can become "fuzzy" in the segmentation.


Figure E.5: Even though the sky looks like a homogeneous color, it is actually a gradient that is split into several regions by the Ward criterion.

## E. 5 Results and conclusions

We have described region-merging as an image-segmention algorithm where merging criteria play a key role for improving the segmentation result. Several low-level merging criteria have been evaluated for application in natural image segmentation. Based on the properties of the criteria, a multi-stage approach has been presented. The Ward criterion is used in the first stage to reduce the influence of image noise. The subsequent Mean-Ward stage performs the actual color segmentation, and finally, the Border criterion reduces oversegmentation due to illumination effects.

Figure E. 7 shows a sample image with the results of the multi-stage segmentation algorithm. Neither the Mean criterion, nor the Ward criterion alone achieves acceptable segmentation results. Only by using a multi-stage approach of Ward, Mean-Ward, and Border (Figure E.7(d)), a subjectively superior object separation is obtained.

(a) Input image.

(c) Ward criterion.

(b) Mean criterion.

(d) Mean-Ward criterion.

Figure E.6: Example segmentation result with various merging criteria.


Figure E.7: Segmentation results for three different criteria using an image of several objects on a table.


## Shape-Based Analysis of Object Behaviour

Once the shapes of the video objects have been determined by an automatic segmentation algorithm, it is interesting to apply further processing to extract semantically high information. For example, the obtained object masks can be used to identify the object and assign it to classes like human, car, bird, and so on. Furthermore, objects usually do not appear static, but they perform some action in the video sequence, which can also be analysed and assigned to sub-classes like "walking human", "standing human", or "sitting human". The analysis of the sequence of object sub-classes over time can be considered as extraction of object behaviour.

In this appendix, experiments are described that we conducted to extract a description of the object behaviour based on the object shape. For the analysis, we combined a classification of the object shape into several pre-defined classes with a model of the transition probability between these classes over time. Having a model that describes the transitions between classes makes the classification more robust than an independent classification for each input frame, because occasional false classifications are avoided by small transition probabilities.


Figure F.1: Best database match for some automatically segmented object masks. The query masks as shown in the top row and the best matching shape is depicted in the bottom row, respectively.

## F. 1 Classification of object shapes

A popular technique to classify objects based on their shape is the Curvature Scale Space (CSS) technique [128, 4]. Essentially, this technique represents the shape of an object with a low-dimensional feature-vector. The representation makes it easy to obtain rotation and scaling-invariant feature vectors. To classify a specific object, we use a database of manuallyclassified objects, in which the CSS feature-vector and the class identifier is saved for each object. Using a specifically designed distance function for CSS feature-vectors [62, 105], we compare the shape of the segmented object with all objects in the database. For an independent classification, we would select the object class with the smallest CSS distance value. Unfortunately, segmentation errors can distort the shape of the object, and also the CSS technique itself has a certain error rate. Both can lead to false classifications. An example of independent classification is depicted in Figure F.1.

To increase the robustness, we do not perform an independent classification for the shape extracted from each frame, but we use contextual information from other frames to make the classification more robust. In order to do this, we compute for each input frame $f$ the CSS distances of the query shape to each object class $c$ and store it as $d_{c}(f)$.


Figure F.2: Transitions between various object classes. Bolder arrows indicate more probable transitions.

## F. 2 Simple model for object behaviour

Real-world objects cannot suddenly change their class in an arbitrary way. For example, a human can never become a car for just a couple of frames, even if its shape suggests this. But, if we further subdivide each class into behaviour sub-classes, transitions may occur. Usually, even though an object can appear in all sub-classes, there are restrictions for state changes. In Figure F.2, transitions between some sub-classes for human motion and an independent car class are shown. According to this model, a human can walk, stand, and sit, but prior to sitting, he has to pass the sitting-down state first. The possible transitions can also be weighted, such that sittingdown has a higher cost (because it is not so probable) as just continuing to walk. More formally, the weights for all transitions from a general state $i$ to state $k$ can be collected in a state transition matrix $w_{i, k}$. For our experiments, we have manually edited this transition matrix based on typical error values as observed in the CSS shape matching step.

## F. 3 Behaviour analysis

Having the independent shape-matching costs and the state-transition costs available, we can compute the most probable class labels $c_{f}$ for each frame $f$ by minimizing the total cost

$$
\begin{equation*}
\min _{\left(c_{f}\right)_{f}}\left\{d_{c_{1}}(1)+\sum_{k=2}^{N}\left(d_{c_{k}}(k)+w_{c_{k}-1, c_{k}}\right)\right\}, \tag{F.1}
\end{equation*}
$$

where $N$ is the total number of frames. This minimization problem can be solved by considering it a minimum-cost path problem in a graph with nodes $V=\bigcup_{f} V_{f}$ composed of columns $V_{f}=\{(c, f)\}_{c}$ and edges $E=\bigcup_{f} V_{f} \times V_{f+1}$


Figure F.3: Computation graph for the classification of object shapes. The minimum-cost path in the graph defines the class for each frame.
between successive columns. Nodes $(c, f)$ are attributed with costs $d_{c}(f)$ and edges $\left(\left(c_{i}, f\right),\left(c_{k}, f+1\right)\right)$ with $w_{i, k}$. The resulting graph is depicted in Figure F. 3 for the example outlined above.

Results for the described example model are depicted in Figure F.4. The results for both humans were obtained with the same model without any parameter adaptation. We consider the presented algorithm and example result as a proof-of-concept implementation. Future work should replace the heuristic cost functions with experimentally-determined probabilities $p_{f}(c)$ instead of $d_{c}(f)$, and transition probabilities $p\left(c_{f} \mid c_{f-1}\right)$. Similarly to Eq. (F.1), we obtain the total probability

$$
\begin{equation*}
\min _{\left(c_{f}\right)_{f}}\left\{p_{1}\left(c_{1}\right) \cdot \prod_{k=2}^{N}\left(p\left(c_{k} \mid c_{k-1}\right) p_{k}\left(c_{k}\right)\right)\right\} \tag{F.2}
\end{equation*}
$$

The products in this equation can be transformed to sums by considering the log-likelihoods. This results in an optimization problem similar to the above shortest-path problem.


Figure F.4: Example results for automatic classification of object behaviour for the human-motion model of Fig. F.2.

## References

[1] T. Aach and A. Kaup. Statistical model-based change detection in moving video. Signal Processing, 31:165-180, 1993.
[2] T. Aach and A. Kaup. Bayesian algorithms for adaptive change detection in image sequences using markov random fields. Signal Processing: Image Communication, 7:147-160, 1995.
[3] T. Aach, A. Kaup, and R. Mester. Change detection in image sequences using gibbs random fields: A bayesian approach. In International Workshop on Intelligent Signal Processing and Communication Systems, ISPACS, pages 56-61, Oct. 1993.
[4] S. Abbasi and F. Mokhtarian. Shape similarity retrieval under affine transform: application to multi-view object representation and recognition. In Proc. Seventh IEEE International Conference on Computer Vision (ICCV), volume 1, pages 450-455, Sept. 1999.
[5] L. Agapito, E. Hayman, and I. Reid. Self-calibration of rotating and zooming cameras. International Journal of Computer Vision, 45(2):107-127, 2001.
[6] A. Alatan, L. Onural, M. Wollborn, R. Mech, E. Tuncel, and T. Sikora. Image sequence analysis for emerging interactive multimedia services - the european cost 211 framework. IEEE Transactions on Circuits and Systems for Video Technology, 8(7):802-813, Nov. 1998.
[7] B. Appleton and C. Sun. Circular shortest paths by branch and bound. Pattern Recognition, 36(11):2513-2520, Nov. 2003.
[8] A. Bartoli, N. Dalal, B. Bose, and R. Horaud. From video sequences to motion panoramas. In Proc. Workshop on Motion and Video Computing, pages 201-207, Dec. 2002.
[9] J. Besag. On the statistical analysis of dirty pictures. Journal of the Royal Statistical Society, Series $B$ 48, pages 259-302, 1986.
[10] S. Birringer. Inexaktes Teil-Graph-Matching für die Suche von Videoobjekten mit Hilfe evolutionärer Algorithmen. Studienarbeit, Universität Mannheim, Mar. 2003.
[11] A. Blake and M. Isard. Active Contours. Springer Verlag, 1998.
[12] T. Brox, D. Farin, and P. H. N. de With. Multi-stage region merging for image segmentation. In $22^{\text {nd }}$ Symposium on Information Theory in the Benelux, pages 189-196, May 2001.
[13] L. Bruzzone and R. Cossu. An adaptive approach to reducing registration noise effects in unsupervised change detection. IEEE Transactions on Geoscience and Remote Sensing, 41:2455-2465, 2003.
[14] L. Bruzzone and D. Fernandez Prieto. An mrf approach to unsupervised change detection. In Proc. IEEE International Conference on Image Processing (ICIP), volume 1, pages 143-147, 1999.
[15] L. Bruzzone and D. Prieto. Automatic analysis of the difference image for unsupervised change detection. IEEE Transactions on Geoscience and Remote Sensing, 38:1171-1182, 2000.
[16] C. Calvo, A. Micarelli, and E. Sangineto. Automatic annotation of tennis video sequences. In DAGM-Symposium, pages 540-547. Springer, 2002.
[17] C. Carson, S. Belongie, H. Greenspan, and J. Malik. Region-based image querying. In CVPR'97 Workshop on Content-Based Access of Image and Video Libraries, pages 42-49, June 1997.
[18] A. Cavallaro and T. Ebrahimi. Video object extraction based on adaptive background and statistical change detection. In Proc. of SPIE VCIP, pages 465-475, Jan. 2000.
[19] A. Cavallaro and T. Ebrahimi. Classification of change detection algorithms for object-based applications. In Proc. of Workshop on Image Analysis For Multimedia Interactive Services (WIAMIS-2003), Apr. 2003.
[20] A. Cavallaro, E. Salvador, and T. Ebrahimi. Shadow-aware objectbased video processing. IEE Vision, Image and Signal Processing, to appear.
[21] I. Celasun and A. M. Tekalp. Optimal 2-d hierarchical content-based mesh design and update for object-based video. IEEE Trans. on Circuits and Systems for Video Technology, 10(7):1135-1153, Oct. 2000.
[22] A. Chakraborty. Video structuring for multimedia applications. In SPIE Proc. of Visual Communication and Image Processing, pages 496-507, 2000.
[23] Y. Chen and J. Z. Wang. A region-based fuzzy feature matching approach to content-based image retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence, pages 1252-1267, 2002.
[24] S.-Y. Chien, C.-Y. Chen, W.-M. Chao, C.-W. Hsu, Y.-W. Huang, and L.-G. Chen. A fast and high subjective quality sprite generation algorithm with frame skipping and multiple sprites techniques. In Proc. IEEE International Conference on Image Processing (ICIP), volume 1, pages 193-196, Sept. 2002.
[25] S.-Y. Chien, C.-Y. Chen, Y.-W. Huang, and L.-G. Chen. Multiple sprites and frame skipping techniques for sprite generation with high subjective quality and fast speed. In Proc. IEEE International Conference Multimedia and Expo (ICME), pages 785-788, 2002.
[26] Y.-Y. Chuang, B. Curless, D. H. Salesin, and R. Szeliski. A bayesian approach to digital matting. In Proc. IEEE Computer Vision and Pattern Recognition (CVPR), volume 2, pages 264-271. IEEE Computer Society, December 2001.
[27] Y.-Y. Chuang, D. B. Goldman, B. Curless, D. H. Salesin, and R. Szeliski. Shadow matting and compositing. ACM Trans. Graph., 22(3):494-500, 2003.
[28] J. Clarke, S. Carlsson, and A. Zisserman. Detecting and tracking linear features efficiently. In R. B. Fisher and E. Trucco, editors, Proc. 7th British Machine Vision Conf. (BMVA), Edinburgh, pages 415-424, 1996.
[29] D. Connor and J. Limb. Properties of frame-difference signals generated by moving images. IEEE Transactions on Communications, 22:1564-1575, 1974.
[30] S. Coorg and S. Teller. Spherical mosaics with quaternions and dense correlation. International Journal on Computer Vision, 37(3):259273, 2000.
[31] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms. The MIT Press, 1990.
[32] J. Costeira and T. Kanade. A multibody factorization method for independent moving objects. International Journal on Computer Vision, 29(3), September 1998.
[33] X. Dai and S. Khorram. The effects of image misregistration on the accuracy of remotely sensed change detection. IEEE Transactions on Geoscience and Remote Sensing, 36:1566-1577, 1998.
[34] J. Davis. Mosaics of scenes with moving objects. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR'98), pages 354360, June 1998.
[35] G. de Haan and P. W. A. C. Biezen. An efficient true-motion estimator using candidate vectors from a parametric motion model. IEEE Transactions on Circuits and Systems for Video Technology, 8(1):8591, Feb. 1998.
[36] P. H. N. de With. A simple recursive motion estimation technique for compression of hdtv signals. In Proc. International Conference on Image Processing and its Applications, pages 417-420, Apr. 1992.
[37] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach. In SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, pages 11-20, New York, NY, USA, 1996. ACM Press.
[38] N. D. Doulamis, A. D. Doulamis, Y. S. Avrithis, and S. D. Kollias. Video content representation using optimal extraction of frames and scenes. In Proc. IEEE International Conference on Image Processing (ICIP), pages 875-879, 1998.
[39] F. Dufaux and J. Konrad. Efficient, robust, and fast global motion estimation for video coding. IEEE Transactions on Image Processing, 9(3), Mar. 2000.
[40] F. Dufaux and F. Moscheni. Background mosaicking for low bit rate video coding. In Proc. IEEE International Conference on Image Processing (ICIP), volume 1, pages 673-676, 1996.
[41] E. Durucan and T. Ebrahimi. Change detection and background extraction by linear algebra. Proceedings of the IEEE, 89:1368-1381, 2001.
[42] A. Ekin and A. M. Tekalp. Automatic soccer video analysis and summarization. In SPIE Storage and Retrieval for Media Databases $I V$, pages 339-350, Jan. 2003.
[43] D. Eppstein. Subgraph isomorphism in planar graphs and related problems. Technical report, Dept. of Information and Computer Science, University of California, May 1994.
[44] P. E. Eren and A. M. Tekalp. Bi-directional 2-D mesh representation for video object rendering, editing and superresolution in the presence of occlusion. Signal Processing: Image Communication, 18(5):321336, May 2003.
[45] R. Fablet, P. Bouthemy, and M. Gelgon. Moving object detection in color image sequences using region-level graph labeling. In $6^{\text {th }}$ IEEE International Conference on Image Processing (ICIP), volume 2, pages 939-943, Oct. 1999.
[46] D. Farin and P. H. N. de With. Towards real-time MPEG-4 segmentation: A fast implementation of region-merging. In 21st Symposium on Information Theory in the Benelux, pages 173-180, May 2000.
[47] D. Farin and P. H. N. de With. A new similarity measure for subpixel accurate motion analysis in object-based coding. In $5^{\text {th }}$ World Multi-Conference on Systemics, Cybernetics and Informatics (SCI), pages 244-249, July 2001.
[48] D. Farin and P. H. N. de With. Estimating physical camera parameters for 3DAV video coding. In $25^{\text {th }}$ Symposium on Information Theory in the Benelux, pages 201-208, June 2004.
[49] D. Farin and P. H. N. de With. Estimating physical camera parameters based on multi-sprite motion estimation. In SPIE Image and Video Communications and Processing, Vol. 5685, volume 5685, pages 489-500, Jan. 2005.
[50] D. Farin and P. H. N. de With. Evaluation of a feature-based globalmotion estimation system. In SPIE Visual Communications and Image Processing, pages 1331-1342, July 2005.
[51] D. Farin and P. H. N. de With. Misregistration errors in change detection algorithms and how to avoid them. In Proc. IEEE International Conference on Image Processing ICIP, volume 2, pages 438-441, Sept. 2005.
[52] D. Farin and P. H. N. de With. Reconstructing virtual rooms from panoramic images. In $26^{\text {th }}$ Symposium on Information Theory in the Benelux, pages 301-308, May 2005.
[53] D. Farin and P. H. N. de With. Automatic video-object segmentation employing multi-sprites with constrained delay. In IEEE International Conference on Consumer Electronics (ICCE), Jan. 2006.
[54] D. Farin and P. H. N. de With. Enabling arbitrary rotational cameramotion using multi-sprites with minimum coding-cost. IEEE Transactions on Circuits and Systems for Video Technology, accepted for publication.
[55] D. Farin, P. H. N. de With, and W. Effelsberg. Optimal partitioning of video sequences for MPEG-4 sprite encoding. In $24^{\text {th }}$ Symposium on Information Theory in the Benelux, pages 79-86, May 2003.
[56] D. Farin, P. H. N. de With, and W. Effelsberg. Recognition of userdefined video object models using weighted graph homomorphisms. In SPIE Image and Video Communications and Processing (IVCP), volume 5022, pages 542-553, Jan. 2003.
[57] D. Farin, P. H. N. de With, and W. Effelsberg. Robust background estimation for complex video sequences. In International Conference on Image Processing (ICIP), volume 1, pages 145-148, Sept. 2003.
[58] D. Farin, P. H. N. de With, and W. Effelsberg. A segmentation system with model assisted completion of video objects. In SPIE Visual Communications and Image Processing (VCIP), volume 5150, pages 366-377, July 2003.
[59] D. Farin, P. H. N. de With, and W. Effelsberg. Minimizing MPEG-4 sprite coding-cost using multi-sprites. In SPIE Visual Communications and Image Processing (VCIP), volume 5308, pages 234-245, Jan. 2004.
[60] D. Farin, P. H. N. de With, and W. Effelsberg. Video-object segmentation using multi-sprite background subtraction. In IEEE International Conference on Multimedia and Expo (ICME), volume 1, pages 343-346, June 2004.
[61] D. Farin, W. Effelsberg, and P. H. N. de With. Robust clusteringbased video-summarization with integration of domain-knowledge. In International Conference on Multimedia and Expo (ICME), volume 1, pages 89-92, Aug. 2002.
[62] D. Farin, T. Haenselmann, S. Richter, G. Kühne, and W. Effelsberg. Segmentation and classification of moving video objects. In B. Furht and O. Marques, editors, Handbook of Video Databases, pages 561591. CRC Press, Sept. 2003.
[63] D. Farin, J. Han, and P. H. N. de With. Fast camera calibration for the analysis of sport sequences. In Proc. IEEE International Conference on Multimedia and Expo (ICME), July 2005.
[64] D. Farin, M. Käsemann, P. H. N. de With, and W. Effelsberg. Ratedistortion optimal adaptive quantization and coefficient thresholding for MPEG coding. In 23 ${ }^{\text {rd }}$ Symposium on Information Theory in the Benelux, pages 131-138, May 2002.
[65] D. Farin, S. Krabbe, W. Effelsberg, and P. H. N. de With. Robust camera calibration for sport videos using court models. In SPIE Storage and Retrieval Methods and Applications for Multimedia, volume 5307, pages 80-91, Jan. 2004.
[66] D. Farin, N. Mache, and P. H. N. de With. SAMPEG, a scene adaptive parallel MPEG-2 software encoder. In SPIE Visual Communications and Image Processing (VCIP), volume 4310, pages 272-283, Jan. 2001.
[67] D. Farin, N. Mache, and P. H. N. de With. A software-based high-quality MPEG-2 encoder employing scene change detection and adaptive quantization. IEEE Transactions on Consumer Electronics, 48:887-897, Nov. 2002.
[68] D. Farin, M. Pfeffer, P. H. N. de With, and W. Effelsberg. Corridor scissors: A semi-automatic segmentation tool employing minimumcost circular paths. In International Conference on Image Processing (ICIP), pages 1177-1180, Oct. 2004.
[69] O. Faugeras. Three-Dimensional Computer Vision. MIT Press, 1993.
[70] P. F. Felzenszwalb and D. P. Huttenlocher. Efficient matching of pictorial structures. In Proc. IEEE International Conference on Computer Vision (ICCV), volume 2, pages 66-73, June 2000.
[71] S. Finke. Robustes Tracking von kleinen Objekten unter Berücksichtigung von Überdeckungen. Diplomarbeit, Universität Mannheim, Jan. 2004.
[72] G. D. Finlayson, S. D. Hordley, and M. S. Drew. Removing shadows from images. In ECCV '02: Proceedings of the 7th European Conference on Computer Vision-Part IV, pages 823-836, London, UK, 2002. Springer-Verlag.
[73] M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381$395,1981$.
[74] D. Geiger, A. Gupta, L. A. Costa, and J. Vlontzos. Dynamic programming for detecting, tracking, and matching deformable contours. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), 17(3):294-302, 1995.
[75] T. Gleixner. Alpha-Kanal-Schätzung aus Einzelbildern. Studienarbeit, Universität Mannheim, Sept. 2003.
[76] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 18:377-388, Apr. 1996.
[77] A. V. Goldberg and R. E. Tarjan. A new approach to the maximumflow problem. Journal of the ACM, 35(4):921-940, 1988.
[78] Y. Gong and X. Liu. Video summarization using singular value decomposition. In Proceedings of Computer Vision and Pattern Recognition ( $C V P R$ ), volume 2, pages 174-180, June 2000.
[79] H. Greenspan, G. Dvir, and Y. Rubner. Region correspondence for image matching via EMD flow. In IEEE Workshop on Content-based Access of Image and Video Libraries, pages 27-31, June 2000.
[80] T. Haenselmann and W. Effelsberg. Wavelet based semi-automatic live-wire segmentation. In SPIE Human Vision and Electronic Imaging VIII, pages 260-269, January 2003.
[81] J. Han, D. Farin, and P. H. N. de With. Multi-level analysis of sports video sequences. In Visual Communications and Image Processing (VCIP), Jan. 2006.
[82] J. Han, D. Farin, P. H. N. de With, and W. Lao. Automatic tracking method for sports video analysis. In $26^{\text {th }}$ Symposium on Information Theory in the Benelux, pages 309-316, May 2005.
[83] K. Haris, S. N. Efstratiadis, and N. Maglaveras. Watershed-based image segmentation with fast region merging. In IEEE International Conference on Image Processing (ICIP), volume 3, pages 338-342, 1998.
[84] C. Harris and M. Stephens. A combined corner and edge detector. In Proceedings of The Fourth Alvey Vision Conference, Manchester, pages 147-151, 1988.
[85] R. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, 2000.
[86] R. I. Hartley. Self-calibration from multiple views with a rotating camera. In ECCV '94: Proceedings of the third European conference on Computer vision (vol. 1), pages 471-478. Springer-Verlag New York, Inc., 1994.
[87] R. I. Hartley. Self-calibration of stationary cameras. International Journal of Computer Vision, 22(1):5-23, 1997.
[88] J. Hayet, J. Piater, and J. Verly. Robust incremental rectification of sports video sequences. In Proc. British Machine Vision Conference (BMVC), Kingston (UK), 2004.
[89] J.-B. Hayet, J. H. Piater, and J. G. Verly. Fast 2D model-to-image registration using vanishing points for sports video analysis. In Proc. IEEE International Conference on Image Processing ICIP, volume 3, pages 417-420, Sept. 2005.
[90] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, Dept. of Electrical Engineering and Computer Science, University of California, Berkeley, CA 94720, June 1989.
[91] J. Hornegger and C. Tomasi. Representation issues in the ml estimation of camera motion. In IEEE International Conference on Computer Vision (ICCV), pages 640-647, 1999.
[92] M. Irani and P. Anandan. All about direct methods. In W. Triggs, A. Zisserman, and R. Szeliski, editors, Vision Algorithms: Theory and practice. Springer-Verlag, 1999.
[93] ISO/IEC 14496-2, International Standard: Information technology coding of audio-visual objects - part 2: visual.
[94] ISO/IEC 14496-5, International Standard: Information technology coding of audio-visual objects - part 5: reference software.
[95] S. Iwase and H. Saito. Tracking soccer players based on homography among multiple views. In Visual Communications and Image Processing (VCIP) 2003, volume 5150, pages 283-292, July 2003.
[96] B. Jähne. Digital Image Processing. Springer Verlag, 2005.
[97] K. Jinzenji, H. Watanabe, S. Okada, and N. Kobayashi. MPEG-4 very low bit-rate video compression using sprite coding. In Proc. IEEE International Conference on Multimedia and Expo (ICME), page 2, Aug. 2001.
[98] S. Kamijo, K. Ikeuchi, and M. Sakauchi. Segmentations of spatiotemporal images by spatio-temporal markov random field model. In Proc. of Energy Minimization Methods in Computer Vision and Pattern Recognition, pages 298-313, 2001.
[99] K. Kanatani. Computational projective geometry. CVGIP: Image Understanding, 54(3):333-348, 1991.
[100] J. J. Kanski. Klinische Ophthalmologie. Urban and Fischer at Elsevier, 2005.
[101] T. Kasetkasem and P. Varshney. An image change detection algorithm based on markov random field models. IEEE Transactions on Geoscience and Remote Sensing, 40:1815-1823, 2002.
[102] H. Kim and K. Hong. Robust image mosaicing of soccer videos using self-calibration and line tracking. Pattern Analysis $\mathcal{E}^{\prime}$ Applications, 4(1):9-19, 2001.
[103] S. Kopf, T. Haenselmann, D. Farin, and W. Effelsberg. Automatic generation of summaries for the web. In Proceedings of SPIE, Storage and Retrieval for Media Databases, Vol. 5307, volume 5307, pages 417-428, Jan. 2004.
[104] S. Kopf, T. Haenselmann, D. Farin, and W. Effelsberg. Automatic generation of video summaries for historical films. In IEEE International Conference on Multimedia and Expo (ICME), volume 3, pages 2067-2070, June 2004.
[105] S. Kopf, G. Kühne, and O. Schuster. Contour-based classification of video objects. In Proceedings of SPIE, Storage and Retrieval for Media Databases, pages 608-618, Jan. 2001.
[106] M. Kourogi, T. Kurata, J. Hoshino, and Y. Muraoka. Real-time image mosaicing from a video sequence. In Proc. IEEE International Conference on Image Processing (ICIP), volume 4, pages 133-137, Oct. 1999.
[107] S. Krabbe. Metadatenextraktion aus Videosequenzen innerhalb eines bekannten Weltmodells am Beispiel von Sportübertragungen. Diplomarbeit, Universität Mannheim, Dec. 2002.
[108] J. B. Kuipers. Quaternions and Rotation Sequences. Princeton University Press, 1998.
[109] T. Kurita. An efficient clustering algorithm for region merging. In IEICE Trans. of Information and Systems, volume E78-D, No. 12, 1995.
[110] R. Laganière and É. Vincent. Wedge-based corner model for widely separated views matching. In IEEE International Conference on Pattern Recognition, volume 3, pages 672-675, 2002.
[111] M. C. Lee, W. Chen, C. B. Lin, C. Gu, T. Markoc, S. I. Zabinsky, and R. Szeliski. A layered video object coding system using sprite and affine motion model. IEEE Trans. on Circuits and Systems for Video Technology, 7(1):130-145, Feb. 1997.
[112] J. Li, J. Z. Wang, and G. Wiederhold. IRM: Integrated region matching for image retrieval. In ACM Multimedia, pages 147-156, 2000.
[113] L. Li and M. Leung. Integrating intensity and texture differences for robust change detection. IEEE Transactions on Image Processing, 11:105-112, 2002.
[114] S. Z. Li. Markov Random Field Modeling in Computer Vision. Springer, 1995.
[115] R. Lienhart, S. Pfeiffer, and W. Effelsberg. Video abstracting. In Communications of the ACM, volume 40, pages 55-62, 1997.
[116] Y. Liu, Q. Huang, Q. Ye, and W. Gao. A new method to calculate the camera focusing area and player position on playfield in soccer video. In SPIE Visual Communications and Image Processing, 2005, pages 1524-1533, July 2005.
[117] M. I. A. Lourakis and A. A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based
on the levenberg-marquardt algorithm. Technical report, Institute of Computer Science of the Foundation for Research and Technology Hellas FORTH, Aug. 2004.
[118] Y. Lu, W. Gao, and F. Wu. Sprite generation for frame-based video coding. In Proc. IEEE International Conference on Image Processing (ICIP), volume 1, pages 473-476, 2001.
[119] Y. Lu, W. Gao, and F. Wu. Efficient background video coding with static sprite generation and arbitrary-shape spatial prediction techniques. IEEE Trans. on Circuits and Systems for Video Technology, 13(5):394-405, 2003.
[120] H. Luo and A. Eleftheriadis. Rubberband: An improved graph search algorithm for interactive object segmentation. In Proc. IEEE International Conference on Image Processing (ICIP), volume 1, pages 101-104, 2002.
[121] J. Maciel and J. P. Costeira. A global solution to sparse correspondence problems. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 25(2):187-199, Feb. 2003.
[122] S. Mann and R. W. Picard. Video orbits of the projective group: A simple approach to featureless estimation of parameters. IEEE Transactions on Image Processing, 6(9), Sept. 1999.
[123] M. Massey and W. Bender. Salient stills: Process and practice. IBM Systems Journal, 35(3\&4):557-573, 1996.
[124] R. Mech and M. Wollborn. A noise robust method for segmentation of moving objects in video sequences. In IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP), pages 2657-2660, Apr. 1997.
[125] B. T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI), 20(5):493-504, May 1998.
[126] K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In European Conference on Computer Vision (ECCV), pages 128-142. Springer, 2002. Copenhagen.
[127] R. Mohr and B. Triggs. Projective geometry for image analysis; a tutorial given at ISPRS, Vienna, Sept. 1996.
[128] F. Mokhtarian and A. K. Mackworth. A theory of multiscale, curvature-based shape representation for planar curves. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 14:789-805, Aug. 1992.
[129] H. Moravec. Visual mapping by a robot rover. In Proceedings of the 6th International Joint Conference on Artificial Intelligence, pages 599-601, August 1979.
[130] E. N. Mortensen and W. A. Barrett. Interactive segmentation with intelligent scissors. Graphical Models and Image Processing, 60:349384, 1998.
[131] Y. Morvan, D. Farin, and P. H. N. de With. Matching-pursuit dictionary pruning for MPEG-4 video object coding. In Internet and multimedia systems and applications, volume 1, pages 476-481, Feb. 2005.
[132] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. Communications in Pure and Applied Mathematics, 42(5):577-685, 1989.
[133] J. Nesvadba, P. Fonseca, A. Sinitsyn, F. de Lange, M. Thijssen, P. van Kaam, H. Liu, R. van Leeuwen, J. Lukkien, A. Korostelev, J. Ypma, B. Kroon, H. Celik, A. Hanjalic, U. Naci, J. Benois-Pineau, P. de With, and J. Han. Real-time and distributed AV content analysis system for consumer electronics networks. In IEEE International Conference on Multimedia and Expo (ICME), July 2005.
[134] C.-W. Ngo, T.-C. Pong, and H.-J. Zhang. On clustering and retrieval of video shots. In ACM Multimedia, pages 51-60, 2001.
[135] H. Nicolas. Optimal criterion for dynamic mosaicking. In Proc. IEEE International Conference on Image Processing (ICIP), volume 4, pages 133-137, Oct. 1999.
[136] P. Nunes and F. M. Pereira. Scene level rate control algorithm for MPEG-4 video coding. In SPIE Visual Communications and Image Processing (VCIP), pages 194-205, 2001.
[137] Y. Ohno, J. Miura, and Y. Shirai. Tracking players and estimation of the 3D position of a ball in soccer games. In "Proc. International Conference on Pattern Recognition (ICPR), volume 1, pages 145-148, Sept. 2000.
[138] N. Paragios and R. Deriche. Geodesic active contours and level sets for the detection and tracking of moving objects. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 22:266-280, Mar. 2000.
[139] M. Pastrnak, D. Farin, and P. H. N. de With. Adaptive decoding of MPEG-4 sprites for memory-constrained embedded systems. In $26^{\text {th }}$ Symposium on Information Theory in the Benelux, pages 137-144, May 2005.
[140] M. Pastrnak, P. Poplavko, P. H. N. de With, and D. Farin. Data-flow timing models of dynamic multimedia applications for multiprocessor systems. In 4 th IEEE International Workshop on System-on-Chip for Real-Time Applications (SoCRT), pages 206-209, July 2004.
[141] I. Patras, E. A. Hendriks, and R. L. Lagendijk. Video segmentation by MAP labeling of watershed segments. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 23(3):326-332, 2001.
[142] R. Pea, M. Mills, J. Rosen, K. Dauber, W. Effelsberg, and E. Hoffert. The diver project: Interactive digital video repurposing. IEEE Multimedia, 11(11):54-61, 2004.
[143] H. Peinsipp. Implementation of a Java applet for demonstration of block-matching motion-estimation algorithms. Studienarbeit, Universität Mannheim, Oct. 2003.
[144] M. Pelillo, K. Siddiqi, and S. W. Zucker. Matching hierarchical structures using association graphs. Technical report, Yale University, Center for Computational Vision \& Control, Nov. 1997.
[145] M. Pelillo, K. Siddiqi, and S. W. Zucker. Continuous-based heuristics for graph and tree isomorphisms, with application to computer vision. In NIPS 99 Workshop on Complexity and Neural Computation, Dec. 1999.
[146] M. Pfeffer. Entwicklung eines Algorithmus zur benutzerunterstützten Segmentierung mehrerer unabhängiger Videoobjekte. Diplomarbeit, Universität Kaiserslautern and Universität Mannheim, Aug. 2003.
[147] M. Pilu. A direct method for stereo correspondence based on singular value decomposition. In Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), pages 261-266, June 1997.
[148] P. Piscaglia, A. Cavallaro, M. Bonnet, and D. Douxchamps. High level description of video surveillance sequences. In Proc. of ECMAST, pages 316-331, 1999.
[149] M. Pollefeys, R. Koch, M. Vergauwen, B. Deknuydt, and L. V. Gool. Three-dimensional scene reconstruction from images. In SPIE Electronic Imaging, Three-Dimensional Image Capture and Applications III, volume 3958, pages 215-226, 2000.
[150] H. V. Poor. An Introduction to Signal Detection and Estimation, 2nd ed. Springer-Verlag, 1994.
[151] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. Numerical recipes in C. Cambridge Univ. Press, 1988.
[152] R. J. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: A systematic survey. IEEE Transactions on Image Processing, to appear.
[153] K. Ratakonda, M. I. Sezan, and R. Crinon. Hierarchical video summarization. In SPIE Proc. Visual Communications and Image Processing (VCIP), pages 1531-1541, 1999.
[154] J. M. Rehg and T. Kanade. Visual tracking of high DOF articulated structures: an application to human hand tracking. In European Conference on Computer Vision (2), pages 35-46, 1994.
[155] I. D. Reid and A. Zisserman. Goal-directed video metrology. In Proc. European Conference on Computer Vision (ECCV), pages 647-658, 1996.
[156] C. Ridder, O. Munkelt, and H. Kirchner. Adaptive background estimation and foreground detection using Kalman-filtering. In Proc. of ICRAM, pages 193-199, 1995.
[157] P. Rosin. Thresholding for change detection. In Computer Vision, 1998. Sixth International Conference on, pages 274-279, 1998.
[158] C. Rother, V. Kolmogorov, and A. Blake. "grabcut": interactive foreground extraction using iterated graph cuts. ACM Trans. Graphics (special issue, Proc. of SIGGRAPH 2004), 23(3):309-314, 2004.
[159] P. J. Rousseeuw and K. Van Driessen. Computing LTS regression for large data sets. Institute of Mathematical Statistics Bulletin, 27(6), 1998.
[160] Y. Rubner. Perceptual Metrics for Image Database Navigation. PhD thesis, Stanford University, 1999.
[161] M. Ruzon and C. Tomasi. Alpha estimation in natural images. In Proc. IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 1, pages 18-25, June 2000.
[162] E. Salvador, A. Cavallaro, and T. Ebrahimi. Shadow identification and classification using invariant color models. In IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP), volume 3, pages 1545-1548, May 2001.
[163] C. Schellewald and C. Schnörr. Subgraph matching with semidefinite programming. In V. D. G. Alberto Del Lungo and A. Kuba, editors, Electronic Notes in Discrete Mathematics, volume 12. Elsevier Science Publishers, 2003.
[164] J. Schmidt and H. Niemann. Using quaternions for parametrizing 3-d rotations in unconstrained nonlinear optimization. In Vision, Modeling, and Visualization, pages 399-406, Nov. 2001.
[165] S. Seedorf. Implementierung eines Java-Applets zur Visualisierung von Geometrietransformationen für Image-Mosaicing. Studienarbeit, Universität Mannheim, July 2003.
[166] J. G. Semple and G. T. Kneebone. Algebraic Projective Geometry. Oxford University Press, 1952.
[167] J. Shi and C. Tomasi. Good features to track. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 593-600, 1994.
[168] H.-Y. Shum, M. Han, and R. Szeliski. Interactive construction of 3D models from panoramic mosaics. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 427-433, June 1998.
[169] M. A. Smith and T. Kanade. Video skimming and characterization through the combination of image and language understanding techniques. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 775-781, 1997.
[170] S. M. Smith and J. M. Brady. SUSAN - a new approach to low level image processing. International Journal of Computer Vision (IJCV), 23(1):45-78, May 1997.
[171] A. Smolic and J. Ohm. Robust global motion estimation using a simplified M-estimator approach. In IEEE International Conference on Image Processing (ICIP), volume 1, pages 868-871, Sept. 2000.
[172] A. Smolic, T. Sikora, and J.-R. Ohm. Direct estimation of long-term global motion parameters using affine and higher order polynomial models. In Proc. Picture Coding Symposium (PCS), pages 239-242, Apr. 1999.
[173] A. Smolic, T. Sikora, and J.-R. Ohm. Direct estimation of long-term global motion parameters using affine and higher order polynomial models. In Proc. Picture Coding Symposium (PCS), Apr. 1999.
[174] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), volume 2, pages 246-252, 1999.
[175] C. V. Stewart. MINPRAN: a new robust estimator for computer vision. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 17(10):925-938, Oct. 1995.
[176] G. Sudhir, J. C. M. Lee, and A. K. Jain. Automatic classification of tennis video for high-level content-based retrieval. In IEEE International Workshop on Content Based Access of Image and Video Databases, pages 81-90, 1998.
[177] H. Suesse and W. Ortmann. Robust matching of affinely transformed objects. In IEEE International Conference on Image Processing (ICIP), volume 2, pages 375-378, Sept. 2003.
[178] C. Sun and S. Pallottino. Circular shortest path in images. Pattern Recognition, 36(3):711-721, Mar. 2003.
[179] R. Szeliski. Image mosaicing for tele-reality applications. In IEEE Workshop on Applications of Computer Vision (WACV), pages 4453, Dec. 1994.
[180] R. Szeliski and H.-Y. Shum. Creating full view panoramic image mosaics and environment maps. In SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques, pages 251-258. ACM Press/Addison-Wesley Publishing Сo., 1997.
[181] C. Thiel. Entwicklung einer skriptgesteuerten Videoanalyse basierend auf MPEG-7 Deskriptoren. Diplomarbeit, Universität Mannheim, Dec. 2003.
[182] P. H. S. Torr and C. Davidson. IMPSAC: synthesis of importance sampling and random sample consensus. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 25(3):354-364, Mar. 2003.
[183] P. H. S. Torr and A. Zisserman. MLESAC: a new robust estimator with application to estimating image geometry. Compututer Vision and Image Understanding, 78(1):138-156, 2000.
[184] A. Torsello and E. R. Hancock. Efficiently computing weighted tree edit distance using relaxation labeling. In Energy Minimization Methods in Computer Vision and Pattern Recognition, Third International Workshop, EMMCVPR 2001, France, volume 2134 of Lecture Notes in Computer Science, pages 438-453. Springer, Sept. 2001.
[185] J. Townshend, C. Justice, C. Gurney, and J. McManus. The impact of misregistration on change detection. IEEE Transactions on Geoscience and Remote Sensing, 30:1054-1060, 1992.
[186] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: Principles and practice of background maintenance. In International Conference on Computer Vision, page 255, 1999.
[187] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment - A modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, Vision Algorithms: Theory and Practice, LNCS, pages 298-375. Springer Verlag, 2000.
[188] A. Vetro, H. Sun, and Y. Wang. MPEG-4 rate control for multiple video objects. IEEE Transactions on Circuits and Systems for Video Technology (CSVT), 9(1):186-199, Feb. 1999.
[189] L. Vincent and P. Soile. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 13(6):583-597, 1991.
[190] P. Viola and M. Jones. Robust real-time object detection. International Journal of Computer Vision - to appear, 2002.
[191] J. Vogel, J. Widmer, D. Farin, M. Mauve, and W. Effelsberg. Prioritybased distribution trees for application-level multicast. In Proceedings of the 2nd Workshop on Network and System Support for Games (ACM NETGAMES 2003), pages 148-157, May 2003.
[192] J. Y. A. Wang and E. H. Adelson. Representing moving images with layers. The IEEE Transactions on Image Processing Special Issue: Image Sequence Compression, 3(5):625-638, September 1994.
[193] J. H. Ward. Hierarchical grouping to optimize an objective function. J. American Stat. Assoc., 58:236-245, 1963.
[194] H. Watanabe and K. Jinzenji. Sprite coding in object-based video coding standard: MPEG-4. In Proc. World Multiconf. on SCI 2001, volume XIII, pages 420-425, 2001.
[195] T. Watanabe, M. Haseyama, and H. Kitajima. A soccer field tracking method with wire frame model from TV images. In Proc. IEEE International Conference on Image Processing (ICIP), pages 16331636, Oct. 2004.
[196] C. R. Wren, A. Azarbayejani, T. Darrell, and A. P. Pentland. Pfinder: real-time tracking of the human body. IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI), 19(7):780-785, July 1997.
[197] N. Xu and N. Ahuja. Object contour tracking using graph cuts based active contours. In Proc. of IEEE International Conference on Image Processing (ICIP), volume 3, pages 277-280, Sept. 2002.
[198] A. Yamada, Y. Shirai, and J. Miura. Tracking players and a ball in video image sequence and estimating camera parameters for 3d interpretation of soccer games. In Proc. 16th Int. Conf. on Pattern Recognition, pages 303-306, Aug. 2002.
[199] X. Yu and D. Farin. Current and emerging topics in sports video processing. In IEEE International Conference on Multimedia and Expo (ICME), July 2005.
[200] Z. Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. Technical Report RR-2676, INRIA, Oct. 1995.
[201] Z. Zhang. Determining the epipolar geometry and its uncertainty a review. International Journal of Computer Vision, 27(2):161-195, Mar. 1998.
[202] F. Ziliani. An image segmentation procedure based on statistical change detection. Technical Report LTS 98.02, Ecole Polytechnique Federale de Lausanne (EPFL), May 1998.

## Summary

Practically established video compression and storage techniques still process video sequences as rectangular images without further semantic structure. However, humans watching a video sequence immediately recognize acting objects as semantic units. This semantic object separation is currently not reflected in the technical system, making it difficult to manipulate the video at the object level. The realization of object-based manipulation will introduce many new possibilities for working with videos like composing new scenes from pre-existing video objects or enabling user-interaction with the scene.

Moreover, object-based video compression, as defined in the MPEG-4 standard, can provide high compression ratios because the foreground objects can be sent independently from the background. In the case that the scene background is static, the background views can even be combined into a large panoramic sprite image, from which the current camera view is extracted. This results in a higher compression ratio since the sprite image for each scene only has to be sent once.

A prerequisite for employing object-based video processing is automatic (or at least user-assisted semi-automatic) segmentation of the input video into semantic units, the video objects. This segmentation is a difficult problem because the computer does not have the vast amount of pre-knowledge that humans subconsciously use for object detection. Thus, even the simple definition of the desired output of a segmentation system is difficult. The subject of this thesis is to provide algorithms for segmentation that are applicable to common video material and that are computationally efficient.

The thesis is conceptually separated into three parts. In Part I, an automatic segmentation system for general video content is described in detail. Part II introduces object models as a tool to incorporate userdefined knowledge about the objects to be extracted into the segmentation process. Part III concentrates on the modeling of camera motion in order to relate the observed camera motion to real-world camera parameters.

The segmentation system that is described in Part I is based on a background-subtraction technique. The pure background image that is required for this technique is synthesized from the input video itself. Sequences that contain rotational camera motion can also be processed since the camera motion is estimated and the input images are aligned into a panoramic scene-background. This approach is fully compatible to the MPEG-4 video-encoding framework, such that the segmentation system can be easily combined with an object-based MPEG-4 video codec.

After an introduction to the theory of projective geometry in Chapter 2 , which is required for the derivation of camera-motion models, the estimation of camera motion is discussed in Chapters 3 and 4. It is important that the camera-motion estimation is not influenced by foreground object motion. At the same time, the estimation should provide accurate motion parameters such that all input frames can be combined seamlessly into a background image. The core motion estimation is based on a feature-based approach where the motion parameters are determined with a robust-estimation algorithm (RANSAC) in order to distinguish the camera motion from simultaneously visible object motion. Our experiments showed that the robustness of the original RANSAC algorithm in practice does not reach the theoretically predicted performance. An analysis of the problem has revealed that this is caused by numerical instabilities that can be significantly reduced by a modification that we describe in Chapter 4.

The synthetization of static-background images is discussed in Chapter 5. In particular, we present a new algorithm for the removal of the foreground objects from the background image such that a pure scene background remains. The proposed algorithm is optimized to synthesize the background even for difficult scenes in which the background is only visible for short periods of time. The problem is solved by clustering the image content for each region over time, such that each cluster comprises static content. Furthermore, it is exploited that the times, in which foreground objects appear in an image region, are similar to the corresponding times of neighboring image areas.

The reconstructed background could be used directly as the sprite image in an MPEG-4 video coder. However, we have discovered that the counterintuitive approach of splitting the background into several independent parts can reduce the overall amount of data. In the case of general camera motion, the construction of a single sprite image is even impossible. In Chapter 6, a multi-sprite partitioning algorithm is presented, which separates the video sequence into a number of segments, for which independent sprites are synthesized. The partitioning is computed in such a way that the total area of the resulting sprites is minimized, while simultaneously
satisfying additional constraints. These include a limited sprite-buffer size at the decoder, and the restriction that the image resolution in the sprite should never fall below the input-image resolution. The described multisprite approach is fully compatible to the MPEG-4 standard, but provides three advantages. First, any arbitrary rotational camera motion can be processed. Second, the coding-cost for transmitting the sprite images is lower, and finally, the quality of the decoded sprite images is better than in previously proposed sprite-generation algorithms.

Segmentation masks for the foreground objects are computed with a change-detection algorithm that compares the pure background image with the input images. A special effect that occurs in the change detection is the problem of image misregistration. Since the change detection compares co-located image pixels in the camera-motion compensated images, a small error in the motion estimation can introduce segmentation errors because non-corresponding pixels are compared. We approach this problem in Chapter 7 by integrating risk-maps into the segmentation algorithm that identify pixels for which misregistration would probably result in errors. For these image areas, the change-detection algorithm is modified to disregard the difference values for the pixels marked in the risk-map. This modification significantly reduces the number of false object detections in fine-textured image areas.

The algorithmic building-blocks described above can be combined into a segmentation system in various ways, depending on whether camera motion has to be considered or whether real-time execution is required. These different systems and example applications are discussed in Chapter 8.

Part II of the thesis extends the described segmentation system to consider object models in the analysis. Object models allow the user to specify which objects should be extracted from the video. In Chapters 9 and 10, a graph-based object model is presented in which the features of the main object regions are summarized in the graph nodes, and the spatial relations between these regions are expressed with the graph edges. The segmentation algorithm is extended by an object-detection algorithm that searches the input image for the user-defined object model. We provide two objectdetection algorithms. The first one is specific for cartoon sequences and uses an efficient sub-graph matching algorithm, whereas the second processes natural video sequences. With the object-model extension, the segmentation system can be controlled to extract individual objects, even if the input sequence comprises many objects.

Chapter 11 proposes an alternative approach to incorporate object models into a segmentation algorithm. The chapter describes a semi-automatic segmentation algorithm, in which the user coarsely marks the object and
the computer refines this to the exact object boundary. Afterwards, the object is tracked automatically through the sequence. In this algorithm, the object model is defined as the texture along the object contour. This texture is extracted in the first frame and then used during the object tracking to localize the original object. The core of the algorithm uses a graph representation of the image and a newly developed algorithm for computing shortest circular-paths in planar graphs. The proposed algorithm is faster than the currently known algorithms for this problem, and it can also be applied to many alternative problems like shape matching.

Part III of the thesis elaborates on different techniques to derive information about the physical 3-D world from the camera motion. In the segmentation system, we employ camera-motion estimation, but the obtained parameters have no direct physical meaning. Chapter 12 discusses an extension to the camera-motion estimation to factorize the motion parameters into physically meaningful parameters (rotation angles, focal-length) using camera autocalibration techniques. The speciality of the algorithm is that it can process camera motion that spans several sprites by employing the above multi-sprite technique. Consequently, the algorithm can be applied to arbitrary rotational camera motion.

For the analysis of video sequences, it is often required to determine and follow the position of the objects. Clearly, the object position in image coordinates provides little information if the viewing direction of the camera is not known. Chapter 13 provides a new algorithm to deduce the transformation between the image coordinates and the real-world coordinates for the special application of sport-video analysis. In sport videos, the camera view can be derived from markings on the playing field. For this reason, we employ a model of the playing field that describes the arrangement of lines. After detecting significant lines in the input image, a combinatorial search is carried out to establish correspondences between lines in the input image and lines in the model. The algorithm requires no information about the specific color of the playing field and it is very robust to occlusions or poor lighting conditions. Moreover, the algorithm is generic in the sense that it can be applied to any type of sport by simply exchanging the model of the playing field.

In Chapter 14, we again consider panoramic background images and particularly focus ib their visualization. Apart from the planar backgroundsprites discussed previously, a frequently-used visualization technique for panoramic images are projections onto a cylinder surface which is unwrapped into a rectangular image. However, the disadvantage of this approach is that the viewer has no good orientation in the panoramic image because he looks into all directions at the same time. In order to provide
a more intuitive presentation of wide-angle views, we have developed a visualization technique specialized for the case of indoor environments. We present an algorithm to determine the 3-D shape of the room in which the image was captured, or, more generally, to compute a complete floor plan if several panoramic images captured in each of the rooms are provided. Based on the obtained 3-D geometry, a graphical model of the rooms is constructed, where the walls are displayed with textures that are extracted from the panoramic images. This representation enables to conduct virtual walk-throughs in the reconstructed room and therefore, provides a better orientation for the user.

Summarizing, we can conclude that all segmentation techniques employ some definition of foreground objects. These definitions are either explicit, using object models like in Part II of this thesis, or they are implicitly defined like in the background synthetization in Part I. The results of this thesis show that implicit descriptions, which extract their definition from video content, work well when the sequence is long enough to extract this information reliably. However, high-level semantics are difficult to integrate into the segmentation approaches that are based on implicit models. Intead, those semantics should be added as postprocessing steps. On the other hand, explicit object models apply semantic pre-knowledge at early stages of the segmentation. Moreover, they can be applied to short video sequences or even still pictures since no background model has to be extracted from the video. The definition of a general object-modeling technique that is widely applicable and that also enables an accurate segmentation remains an important yet challenging problem for further research.

## Samenvatting

De huidige praktisch bewezen videocompressie- en opslagtechnieken bewerken videosequenties nog steeds als rechthoekige beelden zonder enige semantische structuur. Echter, mensen die naar sequenties van videobeelden kijken, nemen onmiddellijk de daarin optredende objecten als semantisch relevante eenheden waar. Deze semantische objectherkenning wordt niet gereflecteerd in de technische implementatie, zodat het moeilijk is om videobeelden te manipuleren op objectniveau. De realisatie van objectmanipulatie zal veel nieuwe mogelijkheden introduceren om met videobeelden te werken, zoals het samenstellen van nieuwe scènes van reeds bestaande video-objecten en speciale gebruikersinteractie met de gerepresenteerde scène.

Daarnaast kan objectgebaseerde videocompressie zoals gedefinieerd in de MPEG-4 standaard, tot hoge compressiefactoren leiden, omdat de objecten op de voorgrond onafhankelijk van de achtergrond kunnen worden verzonden. In het geval van een statische achtergrond in de scène kunnen de verschillende achtergrondbeelden worden gecombineerd in een groot panoramisch beeld, genaamd sprite-beeld, waarvan het actuele camerablikveld kan worden geëxtraheerd. Dit concept resulteert in een hogere compressiefactor, omdat het sprite-beeld voor elke scène slechts eenmaal hoeft te worden verzonden.

Een voorwaarde voor het gebruiken van objectgebaseerde videobewerking is automatische (of op zijn minst met hulp van de gebruiker semiautomatische) segmentatie van de videobeelden aan de ingang in semantische eenheden, ook wel video-objecten genoemd. Deze segmentatie is een complex probleem, omdat een computer niet de enorme voorkennis heeft, die mensen onbewust gebruiken voor het detecteren van objecten. Zelfs een eenvoudige definitie van het gewenste uitgangsresultaat van het segmentatiesysteem is moeilijk. Het onderwerp van dit proefschrift is om algoritmen te ontwikkelen voor segmentatie die toepasbaar zijn voor gebruikelijk videomateriaal en die rekenkundig gezien efficiënt zijn.

Het proefschrift is conceptueel gesplitst in drie delen. In Deel I wordt
een automatisch segmentatiesysteem voor generieke beeldinhoud in detail beschreven. Deel II introduceert objectmodellen als gereedschap om voorkennis van de gebruiker toe te voegen over de te extraheren objecten in het segmentatieproces. Deel III concentreert zich op het modelleren van camerabeweging om de geobserveerde camerabeweging te relateren aan de werkelijke, fysische cameraparameters.

Het segmentatiesysteem dat wordt beschreven in Deel I is gebaseerd op een techniek met achtergrond-subtractie. Het pure achtergrondbeeld dat nodig is voor deze techniek, is gesynthetiseerd van het ingangsvideosignaal zelf. Sequenties van videobeelden die een draaiende camerabeweging bevatten kunnen ook worden bewerkt, omdat de camerabeweging wordt geschat en de ingangsbeelden in een panoramische achtergrond van de scène worden samengesteld. Deze benadering is volledig compatibel met de MPEG-4 videocodering, zodat het segmentatiesysteem probleemloos kan worden gecombineerd met een objectgebaseerde MPEG-4 videocoder.

Na een introductie in de theorie van projectieve geometrie in Hoofdstuk 2 , die nodig is voor de afleiding van camerabewegingsmodellen, wordt de schatting van camerabeweging besproken in de Hoofdstukken 3 en 4. Het is belangrijk dat de schatting van de camerabeweging niet wordt beïnvloed door de objectbeweging op de voorgrond van de scène. Tegelijkertijd moet de schatting tot nauwkeurige bewegingsparameters leiden, zodanig dat alle ingangsbeelden naadloos kunnen worden samengevoegd in het achtergrondsbeeld. De bewegingsschatting is in de kern een featuregebaseerde benadering, waarin de bewegingsparameters worden bepaald met een robuust schattingsalgoritme (RANSAC), om een onderscheid te maken tussen camerabeweging en de gelijktijdig zichtbare objectbeweging. Experimenten hebben aangetoond dat het originele RANSAC-algoritme de theoretische voorspelde robuustheid in de praktijk niet realiseert. Een analyse van dit probleem heeft opgeleverd dat dit door numerieke instabiliteiten wordt veroorzaakt. Deze kunnen significant worden gereduceerd door een algoritmemodificatie die in Hoofdstuk 4 wordt beschreven.

De synthetisatie van beelden met statische achtergrond wordt beschreven in Hoofdstuk 5. Een bijzondere bijdrage is een nieuw algoritme voor het verwijderen van objecten op de voorgrond in het achtergrondbeeld, zodat een pure scène-achtergrond overblijft. Het voorgestelde algoritme is geoptimaliseerd om de achtergrond reconstrueren, zelfs voor moeilijke scènes waarin de achtergrond slechts korte tijd zichtbaar is. Dit probleem is opgelost door de beeldinhoud van een gebied temporeel zodanig te klusteren dat elk kluster een statische beeldinhoud heeft. Tevens wordt benut dat de tijden waarin voorgrondobjecten in een gebied zichtbaar zijn gelijkwaardig zijn aan de corresponderende tijden van naburige beeldgebieden.

De gereconstrueerde achtergrond zou direct kunnen worden gebruikt als sprite beeld in een MPEG-4 videocoder. Het is echter een interessante ontdekking dat een anti-intuïtieve benadering, om de achtergrond te splitsen in verscheidene onafhankelijke delen, de totale hoeveelheid beelddata kan verminderen. In het geval van generieke camerabeweging is de constructie van een enkel sprite-beeld zelfs onmogelijk. In Hoofdstuk 6 wordt een multi-sprite partitioneringsalgoritme gepresenteerd, dat de sequentie van videobeelden verdeeld in een aantal segmenten waarvoor onafhankelijke sprites worden opgebouwd. De partitionering wordt zodanig berekend, dat de totale oppervlakte van de resulterende sprites wordt geminimaliseerd, terwijl gelijktijdig extra voorwaarden worden gerealiseerd. Deze voorwaarden zijn een beperkte sprite buffergrootte in de decoder en de beperking dat de beeldresolutie in de sprite nooit lager mag zijn dan de ingangsbeeldresolutie. De beschreven multi-sprite benadering is volledig compatibel met de MPEG-4 standaard, maar heeft desondanks drie voordelen. Ten eerste kan elke draaiende camerabeweging worden gebruikt. Ten tweede zijn de coderingskosten voor het overdragen van de sprite-beelden lager en ten derde, is de kwaliteit van de gedecodeerde sprite-beelden beter dan dat van eerdere algoritmen voor spritegeneratie.

Segmentatiemaskers voor de voorgrondobjecten worden bepaald met een algoritme voor het detecteren van veranderingen, dat het pure achtergrondbeeld vergelijkt met de ingangsbeelden. Een speciaal effect dat optreedt in de veranderingsdetectie is het probleem van foutieve beeldpositionering. Omdat de veranderingsdetectie overeenkomstige beeldelementen in de camerabewegingsgecompenseerde beelden vergelijkt, kan een kleine fout in de bewegingsschatting leiden tot segmentatiefouten. De reden hiervoor is dat niet-corresponderende beeldelementen worden vergeleken. In Hoofdstuk 7 wordt dit opgelost door zogenaamde risicomaskers in het segmentatie-algoritme te integreren, die beeldelementen identificeren waarvoor foutieve beeldpositionering waarschijnlijk zal resulteren in fouten. Voor deze beeldgebieden is het veranderingsdetectie-algoritme gemodificeerd zodanig dat de beeldverschillen van deze beeldelementen niet worden gebruikt. Deze modificatie vermindert het aantal verkeerde objectdetecties aanzienlijk in beeldgebieden met veel detailinformatie.

De hierboven beschreven algoritmemodules kunnen op verschillende manieren in het segmentatiesysteem worden gecombineerd, afhankelijk van of camerabeweging moet worden geïntegreerd of wanneer real-time executie noodzakelijk is. Deze verschillende systemen en voorbeeldtoepassingen worden bediscussiëerd in Hoofdstuk 8.

Deel II van het proefschrift verbreedt het segmentatiesysteem door objectmodellen mede in de analyse te betrekken. Objectmodellen maken
het mogelijk voor de gebruiker om te specificeren welke objecten uit het videosignaal moeten worden onttrokken. In de Hoofdstukken 9 en 10 wordt een graafgebaseerd objectmodel gepresenteerd, waarin de eigenschappen van de belangrjkste objectgebieden worden samengevat in de knooppunten van de graaf en de spatiële relaties tussen deze gebieden worden uitgedrukt door de verbindingen van de graaf. Het segmentatie-algoritme is uitgebreid met een objectdetectie-algoritme dat in het ingangsbeeld zoekt naar het door de gebruiker gedefiniëerde objectmodel. Twee algoritmen voor objectdetectie zijn ontwikkeld. Het eerste is specifiek geschikt voor tekenfilmbeelden en gebruikt een efficiënt deelgraaf-zoekalgoritme, het tweede objectdetectie-algoritme kan daarentegen algemene videobeelden bewerken. Door de uitbreiding met het objectmodel kan het segmentatiesysteem worden gecontroleerd om individuele objecten te extraheren, zelfs wanneer de ingangsbeeldsequentie veel objecten bevat.

Hoofdstuk 11 stelt een alternatieve benadering voor om objectmodellen te integreren in een segmentatie-algoritme. Dit hoofdstuk beschrijft een semi-automatisch segmentatie-algoritme, waarin de gebruiker globaal het object markeert en de computer dit verfijnt tot de exacte objectcontour. Hierna wordt het object gevolgd gedurende de beeldsequentie. In dit algoritme is het objectmodel gedefinieerd als de beeldstructuur (textuur) langs de objectcontour. Deze textuur wordt onttrokken in het eerste beeld en dan gebruikt gedurende het volgen van het object om het originele object te localiseren. De kern van het algoritme gebruikt een graafrepresentatie van het beeld en een nieuw ontwikkeld algoritme voor het berekenen van de kortste rondgaande paden (cykels) in planaire graven. Het voorgestelde algoritme is sneller dan de algemeen bekende algoritmen voor dit probleem en het kan ook worden toegepast voor veel alternatieve problemen, zoals het vergelijken van objectvormen.

Deel III van het proefschrift gaat dieper in op verschillende technieken om informatie over de fysische 3-D wereld af te leiden van de camerabeweging. In het segmentatiesysteem gebruiken we camerabewegingsschatting, maar de verkregen parameters hebben geen directe fysische betekenis. Hoofdstuk 12 behandelt een uitbreiding naar camerabewegingsschatting om de bewegingsparameters te factoriseren naar fysisch zinvolle parameters (draaihoek, brandpuntsafstand), die zijn gebaseerd op zelf-calibratie. Het speciale element in het algoritme is dat sequenties kunnen worden bewerkt met een camerabeweging die zich over verscheidene sprites uitstrekt, wanneer de eerder genoemde multi-sprite techniek wordt gebruikt. De consequentie is dat het algoritme kan worden toegepast voor generiek draaiende camerabewegingen.

Voor de analyse van videosequenties is het vaak nodig om de object-
posities te bepalen en te volgen. Het is duidelijk dat de objectpositie in beeldcoördinaten weinig informatie geeft wanneer het blikveld van de camera onbekend is. Hoofdstuk 13 bespreekt een nieuw algoritme om de transformatie tussen beeldcoördinaten en de wereldcoördinaten af te leiden voor de speciale toepassing van video-analyse van sportwedstrijden. In sportbeelden kan het blikveld van de camera worden bepaald via markeringen op het speelveld. Om deze reden is een model van het speelveld gebruikt, dat de inrichting van de veldlijnen beschrijft. Nadat de significante lijnen in het beeld zijn gedetecteerd, wordt een combinatorische zoekstrategie uitgevoerd om overeenkomsten tussen lijnen in het beeld en veldlijnen in het model te vinden. Het algoritme heeft geen informatie nodig over de specifieke kleur van het speelveld en het is zeer robuust tegen afdekkingen van objecten of slechte belichtingscondities. Bovendien is het algoritme generiek toepasbaar voor elke andere sport door eenvoudigweg het speelveldmodel te verwisselen.

In Hoofdstuk 14 beschouwen we opnieuw panoramische achtergrondbeelden en focusseren in het bijzonder op hun visualisatie. Behalve de eerder besproken vlakke achtergrond sprites, is het projecteren op een cilinderoppervlak een gebruikelijke visualisatietechniek, waarbij het oppervlak wordt afgerold tot een vlak rechthoekig beeld. Het nadeel van deze techniek is echter dat de kijker geen goede oriëntatie heeft in het panoramische beeld, omdat hij alle richtingen tegelijk observeert. Om te kunnen voorzien in een meer gebruikersvriendelijke visualisatie van panoramische beelden, is een techniek ontwikkeld die speciaal geschikt is voor inpandige ruimtes. We presenteren een algoritme om de 3-D vorm van de kamer waar het beeld was opgenomen te bepalen, of meer algemeen, het berekenen van het complete vloerplan wanneer panoramische beelden van elke ruimte ter beschikking staan. Gebaseerd op de verkregen 3-D geometrie wordt een grafisch model geconstrueerd, waarbij de muren worden getoond met de beeldstructuur die is geëxtraheerd van de panoramische beelden. Deze visualisatie maakt het mogelijk om virtuele wandelingen in de gereconstrueerde kamer te maken en voorziet daardoor in een betere oriëntatie voor de gebruiker.

Samenvattend kan worden geconcludeerd dat alle segmentatietechnieken een zekere definitie van voorgrondobjecten toepassen. Deze definities zijn ofwel expliciet, gebruik makend van objectmodellen zoals in Deel II van dit proefschrift of zij zijn impliciet gedefiniëerd, zoals bijvoorbeeld de achtergrondsynthetisatie in Deel I. De resultaten van dit proefschrift tonen aan dat impliciete modellen die hun definitie onttrekken aan de video-inhoud, goed werken wanneer de beeldsequentie lang genoeg is om deze informatie betrouwbaar te extraheren. Semantiek op hoog niveau is echter moeilijk te integreren in segmentatiebenaderingen die gebaseerd zijn op impliciete
modellen. In plaats daarvan moet deze semantiek in nabewerkingsstappen worden toegevoegd. Expliciete objectmodellen passen daarentegen semantische voorkennis toe in de aanvangsstappen van de segmentatie. Bovendien kunnen deze modellen worden toegepast voor korte videosequenties of zelfs individuele beelden, omdat geen achtergrondmodel hoeft te worden geëxtraheerd van het videosignaal. De definitie van een algemene objectmodelleringstechniek die breed toepasbaar is en die ook een nauwkeurige segmentatie mogelijk maakt, blijft een belangrijk doch uitdagend probleem voor verder onderzoek.

## Zusammenfassung

Die gegenwärtig in der Praxis verwendeten Videokompressions- und Speichertechniken verarbeiten die Videosequenzen nach wie vor als rechteckige Bilder ohne weitere semantische Struktur. Andererseits nehmen wir als Menschen sofort die agierenden Objekte als semantische Einheiten wahr. Diese Zerlegung in semantische Objekte wird momentan auf der technischen Seite nicht durchgeführt, was die Manipulation des Videos auf Objektebene erschwert. Die Realisierung objektbasierter Manipulation wird neue Möglichkeiten für die Verarbeitung von Videos erlauben, wie z.B. das Zusammensetzen neuer Szenen aus vorgefertigten Videoobjekten oder die Interaktion des Benutzers mit der dargestellten Szene.

Desweiteren kann objektbasierte Videokompression, wie sie im MPEG-4Standard definiert wurde, hohe Kompressionsfaktoren erreichen, da die Objekte im Vordergrund unabhängig vom Hintergrund übertragen werden können. Für den Fall dass der Szenenhintergrund statisch ist, können die Hintergrundansichten sogar in ein großes Panoramabild (Sprite) zusammengefügt werden, vom dem die aktuelle Kameraansicht wieder extrahiert wird. Dies resultiert in einem erhöhten Kompressionsfaktor, da das SpriteBild für jede Szene nur einmal gesendet werden muss.

Eine Voraussetzung für objektbasierte Videoverarbeitung ist die automatische (oder zumindest benutzerunterstützte, halbautomatische) Segmentierung des Eingabevideos in semantische Einheiten; den Videoobjekten. Diese Segmentierung ist ein schwieriges Problem, da der Computer nicht das unermessliche Vorwissen zur Verfügung hat, das Menschen unterbewusst für die Objekterkennung benutzen. Daher ist schon eine einfache Definition der erwünschten Ausgabe eines Segmentierungssystems schwierig. Das Thema dieser Dissertation ist es, Algorithmen für die Segmentierung zu entwickeln, die für gewöhnliches Videomaterial geeignet und effizient in der Berechnung sind.

Diese Dissertation ist konzeptuell in drei Teile gegliedert. In Teil I wird ein automatischen Segmentierungssystem für allgemeine Videoinhalte detailliert beschrieben. Teil II führt Objektmodelle als ein Werkzeug ein,
um benutzerdefiniertes Wissen über die zu extrahierenden Objekte in den Segmentierungsprozess einfließen zu lassen. Teil III beschreibt die Modellierung der Kamerabewegung, um die beobachtete Kamerabewegung mit den realen physischen Kameraparametern in Zusammenhang zu bringen.

Das in Teil I beschriebene Segmentierungssystem basiert auf der Technik der Hintergrundssubtraktion. Das reine Hintergrundbild, welches für diese Technik benötigt wird, wird aus dem Eingabevideo selbst synthetisiert. Sequenzen, in denen drehende Kamerabewegungen enthalten sind, können auch verarbeitet werden, da die Kamerabewegung geschätzt wird und die Eingabebilder in ein Panoramabild des Szenenhintergrunds zusammengesetzt werden. Dieser Ansatz ist voll kompatible zum MPEG-4 Videokompressionsverfahren, so dass das Segmentierungssystem problemlos mit einem objektbasierten MPEG-4 Videocodec kombiniert werden kann.

Nach einer Einführung in die Theorie der projektiven Geometrie in Kapitel 2, was für die Herleitung der Kamerabewegungsmodelle benötigt wird, wird die Schätzung der Kamerabewegung in den Kapiteln 3 und 3 diskutiert. Es ist wichtig, dass die Schätzung der Kamerabewegung nicht durch gleichzeitig vorhandene Bewegungen von Vordergrundobjekten beeinflußt wird. Andererseits sollte sie präzise Bewegungsparameter bestimmen, so dass alle Eingabebilder nahtlos in ein Hintergrundbild zusammengefügt werden können. Der Kern der Bewegungsschätzung verwendet einen featurebasierten Ansatz, bei dem die Bewegungsparameter mit einem robusten Schätzalgorithmus (RANSAC) bestimmt werden, um die Kamerabewegung von gleichzeitig sichtbarer Objektbewegung unterscheiden zu können. Unsere Experimente zeigten, dass die Robustheit des ursprünglichen RANSAC-Algorithmus in der Praxis nicht die theoretisch vorausgesagte Leistung erreicht. Eine Analyse des Problems ergab, dass dies in numerischen Instabilitäten begründet liegt, die durch eine in Kapitel 4 beschriebene Modifikation des Algorithmus erheblich reduziert werden können.

Die Synthese statischer Hintergrundbilder wird in Kapitel 5 diskutiert. Dabei präsentieren wir im speziellen einen neuen Algorithmus für das Entfernen von Vordergrundobjekten aus dem Hintergrundbild, so dass der reine Szenenhintergrund verbleibt. Der vorgeschlagene Algorithmus ist daraufhin optimiert, den Hintergrund auch in schwierigen Szenen rekonstruieren zu können, in denen er nur für kurze Zeiträume sichtbar ist. Das Problem wird gelöst, indem die Bildinhalte einer Region zeitlich so zu Clustern gruppiert werden, dass die Cluster jeweils einem statischen Bildinhalt entsprechen. Desweiteren wird ausgenutzt, dass die Zeiten, in denen in einer Region Vordergrundobjekte sichtbar sind, ähnlich sind wie die entsprechenden Zeiten der benachbarten Bildregionen.

Der rekonstruierte Hintergrund könnte direkt als Sprite-Bild in einem MPEG-4 Videocoder verwendet werden. Allerdings haben wir herausgefunden, dass der unintuitive Ansatz, den Hintergrund in mehrere unabhängige Teile zu zerlegen, die gesamte Datenmenge reduzieren kann. Im allgemeinen Fall unbeschränkter Kamerabewegung ist die Konstruktion eines einzelnen Sprite-Bildes sogar unmöglich. In Kapitel 6 wird ein Algorithmus zur Multi-Sprite Zerlegung präsentiert, welcher die Videosequenz in eine Anzahl Segmente unterteilt, für die dann unabhängige Sprites erstellt werden. Die Zerlegung wird so bestimmt, dass die Gesamtfläche des resultierenden Sprites minimiert wird, während gleichzeitig zusätzliche Nebenbedingungen erfüllt werden müssen. Dazu zählt eine Limitierung der Größe des Sprite-Bildspeichers im Decoder und die Einschränkung, dass die Bildauflösung im Sprite niemals unter die Auflösung des Eingabebildes sinken darf. Der beschriebene Multi-Sprite Ansatz ist vollständig kompatibel zum MPEG-4 Standard, aber bietet drei Vorteile. Erstens erlaubt er die Verarbeitung beliebiger drehender Kamerabewegungen. Zweitens sind die Kodierungskosten für die Übertragung des Sprite-Bildes geringer, und schliesslich ist die Qualität des dekodierten Sprite-Bildes besser als in früheren Algorithmen zur Spritegenerierung.

Die Segmentierungsmasken der Vordergrundobjekte werden mit einem Algorithmus zur Detektion von Änderungen zwischen dem reinen Hintergrundbild und den Eingabebildern bestimmt. Ein spezieller Effekt, der in der Änderungsdetektion auftritt, ist das Problem der Fehlausrichtung der Bilder. Da die Änderungsdetektion Bildpunkte an korrespondierenden Bildpositionen vergleicht, kann ein kleiner Fehler in der Bewegungsschätzung zu Segmentierungsfehlern führen, falls Pixel verglichen werden, die nicht korrespondieren. Wir gehen dieses Problem in Kapitel 7 dadurch an, Risikomasken in den Segmentierungsalgorithmus einzuführen, welche diejenigen Bildpunkte markieren, für welche eine Fehlausrichtung der Bilder wahrscheinlich zu Fehlern führen würde. Für diese Bildbereiche wird der Algorithmus zur Änderungsdetektion so modifiziert, dass er die Bilddifferenzen für diese Bildpunkte nicht beachtet. Diese Modifikation reduziert die Anzahl der Fehldetektionen von Objekten in feintexturierten Bildbereichen erheblich.

Die oben beschriebenen Algorithmenmodule können auf verschiedene Weise in ein Segmentierungssystem kombiniert werden, abhängig davon, ob ggf. Kamerabewegungen beachtet werden müssen oder ob eine Ausführung in Echtzeit benötigt wird. Diese unterschiedlichen Systeme und Beispielanwendungen werden in Kapitel 8 diskutiert.

Teil II der Arbeit erweitert das beschriebene Segmentierungssystem so, dass Objektmodelle in die Analyse einbezogen werden. Objektmodelle erlauben es dem Benutzer, die Objekte, die aus dem Video extrahiert werden sollen, zu spezifizieren. In den Kapiteln 9 und 10 wird ein graphenbasiertes Objektmodell präsentiert, in dem die Eigenschaften der elementaren Objektregionen in den Knoten des Graphen zusammengefasst sind und die räumlichen Beziehungen zwischen den Regionen mit Kanten im Graph repräsentiert werden. Der Segmentierungsalgorithmus wird mit einer Objektdetektion erweitert, welche im Eingabebild nach dem benutzerdefinierten Objektmodell sucht. Wir präsentieren zwei Algorithmen zur Objektdetektion. Der erste ist spezialisiert auf Zeichentricksequenzen und benutzt einen Algorithmus zur effizienten Suche von Teilgraphen, wohingegen der zweite reale Videosequenzen verarbeitet. Mit der Erweiterung um Objektmodelle kann das Segmentierungssystem so kontrolliert werden, dass es individuelle Objekte extrahiert, selbst wenn die Eingabesequenz mehrere Objekte enthält.

Kapitel 11 schlägt einen alternativen Ansatz vor um Objektmodelle in einen Segmentierungsalgorithm zu integrieren. Das Kapitel beschreibt einen halbautomatischen Segmentierungsalgorithmus, bei dem der Benutzer das Objekt grob markiert und der Computer dies zur exakten Objektkontur verfeinert. Anschliessend wird das Objekt automatisch durch die Sequenz verfolgt. In diesem Algorithmus wird das Objektmodell als die Textur entlang der Objektkontur definiert. Diese Textur wird im ersten Bild extrahiert und dann während der Objektverfolgung benutzt, um das ursprüngliche Objekt wiederzufinden. Der Kern des Algorithmus benutzt eine Graphdarstellung des Bildes und einen neu entwickelten Algorithmus zur Berechnung kürzester zirkulärer Pfade in planaren Graphen. Der vorgeschlagene Algorithmus ist schneller als die derzeit bekannten Algorithmus für dieses Problem und er kann ebenso für viele andere Probleme benutzt werden, wie z.B. dem Vergleich von Objektformen.

Teil III der Arbeit widmet sich verschiedenen Techniken um Informationen über die physische 3-D Welt aus der Kamerabewegung abzuleiten. Im Segmentierungssystem haben wir die Bewegung der Kamera geschätzt, allerdings hatten die berechneten Parameter keine direkte physikalische Bedeutung. Kapitel 12 diskutiert eine Erweiterung für die Schätzung der Kamerabewegung, um die Bewegungsparameter mit Techniken der Selbstkalibrierung in physikalisch bedeutungsvolle Parameter (wie Drehwinkel oder Brennweite) zu faktorisieren. Die Spezialität des Algorithmus ist, dass er mit Hilfe der Multi-Sprite-Technik Kamerabewegungen verarbeiten kann, die sich über mehrere Sprites erstrecken. Folglich kann der Algorithmus für beliebige drehende Kamerabewegungen angewendet werden.

Für die Analyse von Videosequenzen ist es oft erforderlich, die Position von Objekten zu bestimmen und zu verfolgen. Natürlich liefert die Objektposition in Bildkoordinaten wenig Informationen falls die Blickrichtung der Kamera unbekannt ist. Kapitel 13 beschreibt einen neuen Algorithmus um die Transformation zwischen Bildkoordinaten und Weltkoordinaten für die Spezialanwendung der Sportvideoanalyse zu bestimmen. In Sportvideos kann die Kameraansicht von Markierungen auf dem Spielfeld abgeleitet werden. In diesem Sinne benutzen wir ein Modell des Spielfeldes, welches die Anordnung der Linien beschreibt. Nach der Extraktion der wesentlichen Linien im Eingabebild wird eine kombinatorische Suche durchgeführt um Korrespondenzen zwischen den Linien im Eingabebild und den Linien im Modell herzustellen. Der Algorithmus benötigt keine Information über die spezifische Spielfeldfarbe und ist sehr robust gegenüber Verdeckungen oder ungünstigen Beleuchtungsverhältnissen. Des weiteren ist der Algorithmus generisch in dem Sinne, dass er an jede Sportart angepasst werden kann, indem lediglich das Spielfeldmodell ausgetauscht wird.

In Kapitel 14 betrachten wir wieder Hintergrundpanoramas and konzentrieren uns dabei speziell auf deren Visualisierung. Abgesehen von den ebenen Hintergrund-Sprites, die oben diskutiert wurden, sind Projektionen auf Zylinderoberflächen, die danach zu einem rechteckigen Bild ausgerollt werden, eine gebräuchliche Darstellungstechnik. Der Nachteil dieses Ansatzes ist jedoch, dass der Betrachter sich im Panoramabild nicht gut orientieren kann, da er gleichzeitig in alle Richtungen schaut. Um eine intuitivere Darstellung für weitwinklige Ansichten bereitzustellen, haben wir eine Darstellungstechnik entwickelt, die für Innenraumansichten spezialisiert ist. Wir präsentieren einen Algorithmus, um die 3-D Form des Raumes zu bestimmen, in dem das Bild aufgenommen wurde, oder, allgemeiner, um den kompletten Grundriss zu berechnen, falls Panoramabilder von jedem der Räume zur Verfügung stehen. Basierend auf der ermittelten 3-D Geometrie wird ein graphisches Modell des Raumes erstellt, wobei die Wände Texturen aus den Panoramabildern zugewiesen bekommen. Diese Darstellung erlaubt es, virtuelle Begehungen im rekonstruierten Raum durchzuführen und ermöglicht dadurch dem Betrachter eine verbesserte Orientierung.

Zusammenfassend können wir feststellen, dass sämtliche Segmentierungstechniken eine gewisse Definition für Vordergrundobjekte benutzen. Diese Definitionen sind entweder explizit durch Objektmodelle gegeben, wie in Teil II der Arbeit, oder sie sind implizit definiert, wie z.B. durch die Hintergrundssynthese aus Teil I. Die Ergebnisse dieser Arbeit zeigen, dass implizite Beschreibungen, die ihre Definition aus dem Videoinhalt selbst ableiten, gut funktionieren, wenn die Sequenz lang genug ist, diese Information zuverlässig zu extrahieren. Es ist jedoch schwierig, höhere Se-
mantik in Segmentierungsansätze zu integrieren, die auf impliziten Modellen aufbauen. In diesem Fall sollte die Semantik stattdessen in Nachverarbeitungsschritten hinzugefügt werden. Explizite Objektmodelle bringen dagegen das semantische Vorwissen früh in den Segmentierungsvorgang ein. Desweiteren können sie auf kurze Videosequenzen oder sogar Standbilder angewendet werden, da kein Hintergrundmodell aus dem Video extrahiert werden muss. Die Definition einer allgemeinen Objektmodellierungstechnik, die breit anwendbar ist und die auch eine genaue Segmentierung ermöglicht, bleibt ein wichtiges aber anspruchsvolles Problem für die weitere Forschung.

## Acknowledgments

The research for this thesis was carried out in three research groups, starting in Mannheim at the Circuitry and Simulation group, headed by Prof. Peter de With. After Prof. de With changed his position to the Technical University of Eindhoven, I could continue my research at the Computer Science IV group in Mannheim, headed by Prof. Wolfgang Effelsberg. Finally, I finished the research at the Video Coding and Architectures group at the Technical University of Eindhoven.

However, seen in a more global picture, the story begins much earlier. First, I want to thank my parents for always supporting me and opening the possibility to receive a good education. The basis for my technical education was provided by the University of Stuttgart, where I appreciate especially the time I spent at the Image Understanding group of Prof. Levy for writing my study thesis and master thesis. In particular, I also thank Niels Mache, who supervised these works and with whom I still have a continuing friendship and exchange of ideas.

After the studies in Stuttgart, I joined the Circuitry and Simulation group at the University of Mannheim. I am thankful to Peter de With, who opened the opportunity for me to work towards a PhD on an exciting topic. I also thank Wolfgang Effelsberg for inviting me to join his group and continue my research there without having to change the topic. In both groups I enjoyed to have the full freedom of research and to be allowed to explore any topic that I was interested in. Special thanks go to Peter de With for the detailed reviewing of every paper that we have published, for his endless support and confidence. Finally, I want to express thanks to all my former colleagues in Mannheim and my current colleagues in Eindhoven for the very friendly working environment.

In 2003, Wolfgang Effelsberg and Roy Pea from the Stanford University made it possible that I could spend some time at the Stanford Center for Innovations in Learning (SCIL) to work on panoramic-video processing. I thank both of them for this unique opportunity. This stay has been a valuable experience for me, and I enjoyed the warm working atmosphere
in the project team, which included Roy Pea, Mike Mills, Joe Rosen, and many others. The ideas developed there have lead to Chapter 14.

In the past years, numerous students worked on their study and master thesis under my supervision. Thereby, they helped substantially in the implementation and development of new ideas. The valuable student work is too numerous to be listed exhaustively, but in particular, I want to thank (in temporal order) Alexander Staller, Thomas Brox [12], Michael Käsemann [64], Susanne Krabbe [107], Stefan Birringer [10], Stefan Seedorf [165], Tobias Gleixner [75], Holger Peinsipp [143], Magnus Pfeffer [146], Christian Thiel [181], and Sascha Finke [71].

It is also noteworthy that all research and writing of this thesis was carried out exclusively with open-source software. I am indebted to everyone working on these wonderful software projects, especially the programs gcc, emacs, the Linux kernel, LaTeX, and tgif.

Finally, I want to thank the promotion committee for reviewing the thesis. In particular, the very detailed reading and endless flow of comments provided by Peter de With and Wolfgang Effelsberg were very helpful. I take full responsibility for any remaining non-conform hyphenation and misplaced comma.

## Biography



Dirk Farin was born in Tübingen, Germany in 1973. He graduated in computer science and electrical engineering from the University of Stuttgart, Germany, in 1999. Subsequently, he became research assistant at the Department of Circuitry and Simulation at the University of Mannheim, where he started his research on video-object segmentation. He joined the Department of Computer Science IV at the University of Mannheim in 2001. Since 2004, he has a post-doc position in the Video Coding and Architecture group at the Technical University of Eindhoven, Netherlands. Apart from video-object segmentation, his research interests include video compression, content analysis, and 3-D reconstruction. Currently, he is involved in a joint project of Philips and the Technical University of Eindhoven about the development of video capturing and compression systems for 3-D television. He received a best student paper award at the SPIE Visual Communications and Image Processing conference in 2004 for his work on multi-sprites, and two best student paper awards at the Symposium on Information Theory in the Benelux in 2001 and 2003. He is member of the program committee of the IEEE International Conference on Image Processing and reviewer for several journals including IEEE Multimedia and IEEE Circuits and Systems for Video Technology. In 2005, he organized a special session about sports-video analysis at the IEEE International Conference on Multimedia and Expo. Mr. Farin developed popular open-source and commercial software including an MPEG-2 decoder, two MPEG-2 encoders, libraries with computer-vision algorithms, and image-format conversion software.

## STELLINGEN

behorende bij het proefschrift

Automatic Video Segmentation Employing Object/Camera Modeling
door Dirk Sven Farin

Since a segmentation system for natural video is based on various algorithms applying different mathematical techniques at the same time, such as optimization, clustering, or statistics, the main problem in segmentation is not the ultimate cleverness of any of these algorithms, but the complexity to combine them into a robust system.

Thesis Part I and Chapter 13.

## II

Although coding people have adopted the projective motion model for sprite generation to cover rotational camera motion, it is surprising to see that it was not noticed that the model implicitly imposes a restriction on the maximum rotation angle between two frames.

$$
\text { See [122] and Chapter } 6 .
$$

## III

A desired segmentation result can be defined either explicitly by describing the object itself with application preknowledge, or it can be found as the difference to a background model that is usually extracted from the input itself. The shorter the input sequence, the more preknowledge should be added to the object model, in order to compensate for the reduced amount of information obtained from the input data.

Chapters 5 and 7 on background models versus Chapters 9 and 10 on object models.

## IV

While most people work on optimizing optimization algorithms, it is often possible and more successful to optimize the optimization goal.

For example, Chapter 14, where the observation angle is redefined to improve the convergence behaviour.

## V

Whenever content analysis is used in order to raise the level of semantic understanding of the input data, the user should be aware that the analysis will partly fail in finding the correct semantics. In this aspect, the robustness of the analysis can only be increased by further constraining the application domain.

Chapter 8 and Chapter 13

## VI

The accuracy of analysis is typically increased by including more model knowledge and more advanced processing techniques, but it can be more effective to simply adopt a more suitable input sensor.
E.g., using radar instead of computer vision for measuring the distance to close vehicles.

VII
The most insightful ideas for image understanding can be obtained when one is fooled by the human visual system, as this is the unique moment in which unconscious processes become visible.

## VIII

Many algorithms feature the adjustability of a plurality of algorithm parameters (e.g., thresholds) for adapting to varying input conditions. However, well-designed algorithms should either apply parameters that are derived from measurable input properties, or the setting should be non-critical within a wide range, independent of the input.

At best, any well-designed program code should be based only on fundamental constants and the numbers 0 (for initializations), 1 (for iterations), and 2 (for decisions).

## IX

Commercial software is of inherently lower quality than comparable open-source software, because it is developed by people who are paid for writing software, instead of people who are fascinated about finding the best solution.

## X

While mathematics is the art of denoting different things with the same name, research funding can be raised more easily by giving new names to existing concepts.

Extending a quote of Jules Henri Poincaré.

## XI

The query-by-humming algorithms are a nice tool to search music databases, but it has not been solved yet how one should hum the Scriabin sonatas.

## XII

In computer graphics and electronic music, we should not strive to imitate the real, but instead explore the creation of new forms of art.

## XIII

The Dutch lunchrooms show that the food supply may be optimized with respect to preparation time or walking distance, but they also show that more subjective properties, like taste, are not so easy to integrate into the optimization.

## XIV

Noticing that you do not miss something can be more worrying than actually missing it.


[^0]:    ${ }^{1}$ Note that this definition implies that shadows and reflections are extracted as part of the object and that "background objects" like the audience in a sports broadcast are also considered as foreground when they are moving.

[^1]:    Chapter Publication title and contribution
    Early work about content-adaptive MPEG-2 encoding
    1 "A Software-Based High-Quality MPEG-2 Encoder Employing Scene Change Detection and Adaptive Quantization", IEEE Trans. on Consumer Electronics, 2002, [67] and
    1 "SAMPEG, a Scene Adaptive Parallel MPEG-2 Software Encoder", SPIE VCIP, 2001, [66]

    Preparatory work: implementation of a parallel MPEG-2 encoder. Includes video content analysis to control the adaptive quantization separately for different types of content. Also includes scene-change detection for adapting the GOP pattern.
    1 "Rate-Distortion Optimal Adaptive Quantization and Coefficient Thresholding for MPEG Coding", 23rd Symposium on Information Theory in the Benelux, 2002, [64] Development of a theoretically optimal encoder for MPEG-2 I-frames, yielding the highest possible PSNR for quality comparison to adaptive quantization approaches.

    |  | Part I - An Automatic Video-Object Segmentation System |
    | :---: | :--- |
    | $2-8, \mathrm{~F}$ | Book chapter "Segmentation and Classification of Mov- <br> ing Video Objects", in "CRC Handbook of Video <br> Databases", 2003, [62] |
    | Overview of the core segmentation system, excluding multi- |  |
    | sprite segmentation. Presents different motion-models, |  |
    | feature-based and dense motion-estimation, Markov Random |  |
    | Field segmentation. Also discusses object recognition based on |  |
    | the object shape. |  |

[^2]:    ${ }^{1}$ Rigid: shape and size preserving. Motion of non-deformable solid objects.

[^3]:    ${ }^{1}$ See also Annex B for an alternative way to obtain the motion parameters with lower computation cost.

[^4]:    ${ }^{2}$ Note that we use the three terms feature-point, interest-point, and corner-point almost synonymically. The term feature-point originates from the motion estimation area, where the generation process of these points is not considered. The terms interest and corner point are used for detection algorithms, depending on whether the algorithm is designed to identify just arbitrary interest points, or corners in the image.

[^5]:    ${ }^{3}$ See Appendix D for a description of the sequence contents.

[^6]:    ${ }^{4}$ Since the latter detectors do not show that clear saturation point, we fix $\epsilon$ to a defensive value of 2.5 for the successive sections and set $\mathcal{R}=\mathcal{R}_{2.5}$.

[^7]:    ${ }^{1}$ Also visible in the table is the unusually high error for the opera 4 sequence. The reason for this is the fact that the sequence shows very little texture (see Fig. $4.5(\mathrm{~b})$ ), so that only few feature-points are generated. Moreover, these features are not distributed equally over the image. A motion model that is derived from only these features will have a larger error at positions that are distant to the detected features. We will evaluate the problem of parameter estimation from a poor set of features in detail in Section 4.3.3. In most cases, these errors in the feature-based motion estimator can be corrected in the direct estimation algorithm that will be described in Chapter 5.

[^8]:    ${ }^{2}$ See Appendix C for a description of alternative algorithms.

[^9]:    ${ }^{1}$ In fact, it is also possible to generate sprites for arbitrary camera motion including translation provided that the background is planar.

[^10]:    ${ }^{2}$ At first view, both approaches seem to be symmetric. If the cost of non-overlapping pixels is high, the optimization will move the image onto the defined area to decrease the error. On the other hand, if the cost of non-overlapping pixels is zero, the optimization will move the image off the defined area to decrease the error. However, the difference is that we start with a good initialization where most of the residuals are low. Hence, if we set the cost in undefined areas to zero, the error does not change much if pixels move off the defined area.

[^11]:    ${ }^{3}$ Bi-linear interpolation was chosen because this complies to the sprite-warping process as defined in the MPEG-4 standard.

[^12]:    ${ }^{4}$ We will omit the superscript $(u, v)$ to simplify notation when the meaning is clear. Since we are only considering single blocks in this section, no ambiguities will occur.

[^13]:    ${ }^{1}$ But with the optimal selection of the reference frame.

[^14]:    ${ }^{2}$ For the affine motion model, which is a special case of the projective transformation, $p_{x}, p_{y}$ are zero and, hence, $D=1$. The pixel scale is then simply the determinant of the affine matrix and independent of $x, y$.

[^15]:    ${ }^{1}$ In [1] it is derived that for the Laplace distribution, a test-statistic using the sum of absolute differences should be used, while for the Gaussian distribution, the sum of squared differences is optimal.

[^16]:    ${ }^{1}$ However, the information from the camera control can provide a good initialization for the motion estimation.

[^17]:    ${ }^{1}$ These ellipses have already been shown in the visualizations in the last chapter, but they were not actually used in the matching algorithm.

[^18]:    ${ }^{1}$ In fact, minimum-cost paths can cross more than once, but there is always a path of similar cost that does not cross more than once. Hence, when searching for a minimumcost path, we can assume that they do not cross more than once.

[^19]:    ${ }^{1}$ Note that the original article [180] contains several typing errors in the equations.

[^20]:    ${ }^{2}$ The Cholesky decomposition $\boldsymbol{\omega}=\mathbf{A}^{\top} \mathbf{A}$ is often implemented to give lower triangular matrices $\mathbf{A}$. We use the trick of transposing the matrix $\omega$ along the upward diagonal $\left(\omega_{00} \leftrightarrow \omega_{22}, \omega_{01} \leftrightarrow \omega_{12}, \omega_{10} \leftrightarrow \omega_{21}\right)$ before and after the decomposition to find a decomposition into upper triangular matrices.

[^21]:    ${ }^{3}$ Note that the constant principal-point constraint was not applied in the linear calibration algorithm.

[^22]:    ${ }^{1}$ We use the word court as an umbrella term for playfields with clearly defined geometric marks, like tennis courts, soccer fields, volleyball courts.

[^23]:    ${ }^{1}$ The algorithm considers small blocks in the image independently. For each block, it computes the sum of luminance differences between adjacent same-parity field lines and the sum between adjacent differing-parity lines. If motion is visible in the block, the differences between lines of differing parity is larger, since the object moved during the

[^24]:    ${ }^{2}$ This presentation has become popular with Apple's Quicktime VR standard.

[^25]:    ${ }^{1}$ See section E.4.1 for more information on the watershed-presegmentation column.

