Robust and Efficient Camera-based Scene Reconstruction

Robust and Efficient Camera-based Scene Reconstruction

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Dipl.-Inform. Benjamin Josef Resch
aus Heidenheim an der Brenz

Tübingen
2017

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	19.09.2017
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Hendrik P. A. Lensch
2. Berichterstatter:	Prof. Dr. Andreas Schilling

für Maria

Abstract

For the simultaneous reconstruction of 3D scene geometry and camera poses from images or videos, there are two major approaches: On the one hand it is possible to perform a sparse reconstruction by extracting recognizable features from multiple images which correspond to the same 3D points in the scene. With those features, the positions of the 3D points as well as the camera poses can be obtained such that they explain the positions of the features in the images best. On the other hand, on video data, a dense reconstruction can be obtained by alternating between the tracking of the camera pose and updating a depth map representing the scene per frame of the video. In this dissertation, we introduce several improvements to both reconstruction strategies. We start from improving the reliability of image feature matches which leads to faster and more robust subsequent processing. Then, we present a sparse reconstruction pipeline completely optimized for high resolution and high frame rate video, exploiting the redundancy in the data to gain more efficiency. For (semi-)dense reconstruction on camera rigs which is prone to calibration inaccuracies, we show how to model and recover the rig calibration online in the reconstruction process. Finally, we explore the applicability of machine learning based on neural networks to the relative camera pose problem, focusing mainly on generating optimal training data. Robust and fast 3D reconstruction of the environment is demanded in several currently emerging applications ranging from set scanning for movies and computer games over inside-out tracking based augmented reality devices to autonomous robots and drones as well as self-driving cars.

Kurzfassung

Für die gemeinsame Rekonstruktion von 3D Szenengeometrie und Kamera-Posen aus Bildern oder Videos gibt es zwei grundsätzliche Ansätze: Auf der einen Seite kann eine aus wenigen Oberflächen-Punkten bestehende Rekonstruktion erstellt werden, indem einzelne wiedererkennbare Features, die zum selben 3D-Punkt der Szene gehören, aus Bildern extrahiert werden. Mit diesen Features können die Position der 3D-Punkte sowie die Posen der Kameras so bestimmt werden, dass sie die Positionen der Features in den Bildern bestmöglich erklären. Auf der anderen Seite können bei Videos dichter gesampelte Oberflächen rekonstruiert werden, indem für jedes Einzelbild zuerst die Kamera-Pose bestimmt und dann die Szenengeometrie, die als Tiefenkarte vorhanden ist, verbessert wird. In dieser Dissertation werden verschiedene Verbesserungen für beide Rekonstruktionsstrategien vorgestellt. Wir beginnen damit, die Zuverlässigkeit beim Finden von Bildfeature-Paaren zu erhöhen, was zu einer robusteren und schnelleren Verarbeitung in den weiteren Rekonstruktionsschritten führt. Außerdem präsentieren wir eine Rekonstruktions-Pipeline für die Feature-basierte Rekonstruktion, die auf hohe Auflösungen und Bildwiederholraten optimiert ist und die Redundanz in entsprechenden Daten für eine effizientere Verarbeitung ausnutzt. Für die dichte Rekonstruktion von Oberflächen mit Multi-Kamera-Rigs, welche anfällig für Kalibrierungsungenauigkeiten ist, beschreiben wir, wie die Posen der Kameras innerhalb des Rigs modelliert und im Rekonstruktionsprozess laufend bestimmt werden können. Schließlich untersuchen wir die Anwendbarkeit von maschinellem Lernen basierend auf neuralen Netzen auf das Problem der Bestimmung der relativen Kamera-Pose. Unser Hauptaugenmerk liegt dabei auf dem Generieren möglichst optimaler Trainingsdaten. Eine robuste und schnelle 3D-Rekonstruktion der Umgebung wird in vielen zur Zeit aufstrebenden Anwendungsgebieten benötigt: Beim Erzeugen virtueller Abbilder realer Umgebungen für Filme und Computerspiele, bei inside-out Tracking basierten Augmented Reality Geräten, für autonome Roboter und Drohnen sowie bei selbstfahrenden Autos.

Acknowledgments

First I would like to thank Prof. Dr.-Ing. Hendrik P.A. Lensch for the possibility to write this thesis during my work in the Computer Graphics Group in Tübingen. This includes providing the technical and social environment for working on my research and many fruitful discussions. Due to his expertise in different fields from computer graphics over computer vision to computational photography and machine learning, he often could provide insights from a different perspective onto upcoming problems.

I would also like to thank the other members of the Computer Graphics Group and people who stayed with us for sabbaticals or interns, namely (in alphabetical order) Boris Ajdin, Dennis Bukenberger, Manuel Finckh, Christian Fuchs, Fabian Groh, Sebastian Herholz, Chen Jieen, Jochen Lang, Takahiro Okabe, Katharina Schwarz, Beatriz Trinchão Andrade, Jian Wei and Patrick Wieschollek, for lots of short-term discussions and answered questions and the flexibility everybody provided in scheduling research, teaching and administration tasks.

Another thanks goes to my secondary supervisor Prof. Dr. Andreas Schilling for our discussions and his eye for details that made me rethink specific aspects of my work.

I am also thankful to Dr. Alexander Sorkine-Hornung and the former Image and Video Processing Group at Disney Research Zürich for providing me the opportunity for an internship during which a part of the research for this thesis could be done.

Last but maybe most important, I'd like to thank my family: my mom and dad who managed to let my sister and me grow up in a protected and supporting environment while providing us with the attitude that life is what you make it and my loving wife for relieving me from regular tasks in stressful times and her motivational support.

Contents

Chapter 1

Introduction

The reconstruction of the 3D geometry of a scene and the corresponding camera poses only from images has its origins in the field of photogrammetry which goes back to Leonardo da Vinci [Gho05]. Due to the lack of automated computing, only very basic, mostly planar scenes and projections were studied at that time. This changed in the 1980s when sufficient compute power was available for processing more complex tasks like handling arbitrary scene geometry or more advanced projection models. Finally, in the 1990, affordable digital cameras became available which enabled easy and cheap capturing of images and videos from real world scenes which are subject to geometrical reconstruction in this thesis.

To reconstruct 3D geometry from images, the first step usually is to establish some kind of mapping between images in the sense of recovering which positions on a reference image correspond to which positions on one or more secondary images. This can either be done on distinctive keypoints which are easily detectable and recognizable such as corners, or by establishing a dense mapping which aims on optimally finding a mapping for all pixels of one image to another. Such mappings can be found without any previous knowledge about the scene structure or camera poses by analyzing the image contents for similarities only. However, correctly mapped keypoints from two images should show the same 3D surface point in space (see Figure 1.1).

Geometrical Reconstruction. As a next step, the camera parameters and especially the poses from which the images were taken have to be reconstructed. In general, we start from few, most often two camera poses and append more and more camera poses to the reconstruction later. There are several methods to recover the relative pose of a pair of cameras, e.g. analytic, closed form methods that rely on at least five keypoint correspondences between two images.

As soon as at least two camera poses are geometrically registered against each other, the scene geometry can be recovered. A simple approach that suits well enough as initialization of the scene geometry is triangulation (see Figure 1.1): given a corresponding pair of keypoints from two images, we can represent both of the image positions as rays in the 3D space, originating from the camera centers and intersecting the keypoints' positions on the image planes. As both keypoints show the same 3D surface point as stated above,

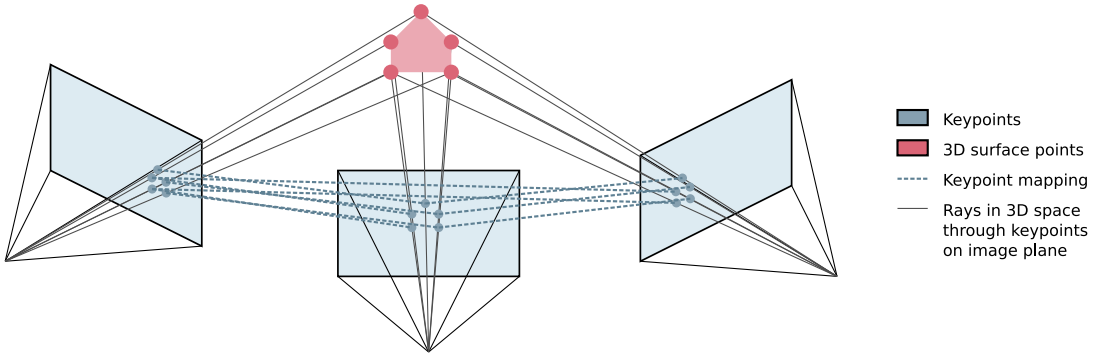


Figure 1.1: *Entities envolved in reconstruction 3D geometry and camera poses. Based on mappings between the images, relative camera poses can be obtained. Based on those, 3D surface points are triangulated which serve as anchors for additional camera poses to be attached to the reconstruction.*

the corresponding rays must intersect at exactly this point in space. By triangulating positions for all keypoint correspondences of two images or even for every pixel with a dense image-to-image mapping, we can obtain a representation of the scene's structure.

Given this structure, additional camera poses are appended to the reconstruction. Note that by generating all 3D surface points based on image positions, we can transfer the previously constructed image-to-image mappings to image-to-surface mappings. Given at least three surface points that map to image positions for a camera that is not yet geometrically registered, we can again find its relative pose. From here on, the scene expands iteratively by alternating between updating the scene structure based on all geometrically registered cameras.

Obviously, the geometrical errors in such an iterative reconstruction are not distributed equally: errors are lower between the camera poses and surface points that are directly reconstructed from each other but may be higher otherwise. This leads to unnecessarily high accumulation of errors when cameras are added iteratively to the reconstruction and should therefore be avoided. A trivial but quite compute expensive approach is to iteratively re-register camera poses and surface points according to all matched points / camera poses until they converge. A more practical approach is Bundle Adjustment (BA): The refinement of the camera poses and point positions is treated as a nonlinear optimization problem whose residual (which is to be minimized) is based on the error between surface points projected back into camera images and their corresponding keypoints. Specialized solvers for sparse nonlinear problems can optimize all camera pose and point parameters efficiently in common, lowering the residual and thus finding a geometrical scene representation that fits all observed image keypoint correspondences best.

Photometrical Reconstruction. As an alternative to the geometrical registration of camera poses and scene structure based on observed positions on the images as described

above, a photometric approach can be chosen for densely reconstructed scene structures: The whole reconstructed scene structure can be projected into the image of a synthetic, arbitrarily placed camera, thus creating an artificial rendering of the scene from a novel viewport. We can optimize this synthetic camera's pose to generate an image that is photometrically similar to a certain captured image, thus recovering the camera pose parameters (see Figure 2.17 left).

We can also use a photometric approach for obtaining the scene structure: Given a pixel's ray in 3D space, we can project all points on that ray to all other images and compare the image patches at the projected positions for photometric consistency (see Figure 2.17 right). The most consistent point on the ray usually corresponds to the 3D surface position.

Machine Learning based Reconstruction. Instead of formulating residuals for estimating a camera pose explicitly as described above, we can also apply machine learning to the problem: artificial neural networks (ANNs) can be seen as large functions mapping an input (e.g. two video frames) to an output (e.g. the relative camera pose) based on many parameters. Training the ANN corresponds to optimizing for parameters that accomplish the task of the neural network correctly.

1.1 Main Contributions

In this thesis, we develop improvements to sparse, feature based as well as dense 3D reconstruction algorithms and examine the applicability of ANNs to the task of relative camera pose tracking. In particular, the contributions are:

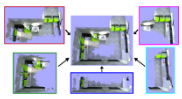
- An image feature matching refinement strategy for omnidirectional cameras. Given matches between dinstinct keypoints of two images, our method is able to analyze all matches for rotational consistency even on omnidirectional images where a camera rotation can cause the image content to rotate differently depending on the image position. Our method can be seamlessly integrated into any image feature matching framework that extracts orientations from its keypoints. [RLL14].

- A feature based 3D reconstruction pipeline which is designed for fast but precise processing of high resolution, high framerate video footage. We achieve unprecedented performance on this task by using a window-based, piecewise processing of video streams, robust subsampling of scene points and camera poses during reconstruction and a careful, cost-value based selection of frame pairs for loop closing to reduce the data that has to be processed with BA at once. BA-like global consistency is established based on a linear optimization that does not concern scene points. [RLW+15].

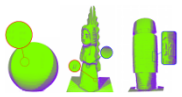
- The extension of a monocular, photometry based reconstruction and camera tracking framework to flexible multi-camera rigs. Modelling and reconstructing the rig's dynamic intrinsics per frame is crucial because larger rigs are hard to manufacture perfectly rigidly and even small 1-pixel calibration errors can render the photometric error useless. [RWL16].

- A game engine based framework for training a neural network on relative camera pose tracking. Here we focus on analyzing which training data works best concerning the number of input frames, the camera motion presented for training and others.

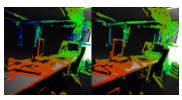
Some of these topics have already been published in the following bold-faced conference papers. In addition, this thesis may have some overlapping content with the other listed papers.



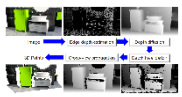
Dense and Scalable Reconstruction from Unstructured Videos with Occlusions.
Jian Wei, Benjamin Resch, Hendrik P. A. Lensch.
VMV 2017. [WRL17]



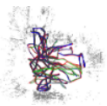
Multi-View Continuous Structured Light Scanning.
Fabian Groh, Benjamin Resch, Hendrik P. A. Lensch.
GCPR 2017. [GRL17]



Real Time Direct Visual Odometry for Flexible Multi-Camera Rigs.
Benjamin Resch, Jian Wei and Hendrik P. A. Lensch.
ACCV 2016. [RWL16]



Dense and Occlusion-Robust Multi-View Stereo for Unstructured Videos.
Jian Wei, Benjamin Resch, Hendrik P. A. Lensch.
CRV 2016 (Oral). [WRL16]



Scalable Structure from Motion for Densely Sampled Videos.
Benjamin Resch, Hendrik Lensch, Oliver Wang, Marc Pollefeys, Alexander Solkine-Hornung.
IEEE CVPR 2015. [RLW+15]

Local Image Feature Matching Improvements for Omnidirectional Camera Systems.
Benjamin Resch, Jochen Lang, Hendrik P. A. Lensch.
ICPR 2014, pp. 918-923. [RLL14]

Multi-View Depth Map Estimation With Cross-View Consistency.
Jian Wei, Benjamin Resch, Hendrik P. A. Lensch.
BMVC 2014. [WRL14]

Please note that all research, data generation, data analysis and paper writing in the bold-faced publications that are handled in this thesis is my contribution, while the other authors were mainly involved in discussions and proof reading. An exception is [RLW+15] were some paragraphs were contributed or refined by other authors while writing up the paper and generating the viewport interpolation data in Section 5.4.4 in which Dennis Bukenberger was involved.

1.2 Funding and External Contributions

The research topics presented in this thesis have received the following funding or other external support:

- The work in Chapter 3 has been partially funded by the DFG Emmy Noether fellowship (Le 1341/1-1).
- In Chapter 4, frame matching cost matrices as illustrated in Figure 4.13 were kindly provided by Oliver Wang [WSZ+14].
- The work presented Chapter 5 was supported by Daimler AG, Germany. Real-world flexible stereo rig datasets were kindly provided by Dr. Senya Polikovsky, OSLab, Max Planck Institute for Intelligent Systems Tübingen.

1.3 Notation

Please note that vectors appearing in equations throughout this thesis are denoted with boldfaced small letters while Matrices are denoted with bold capital letters. All scalar values appear in normal letters.

Any modifiers like sub- or superscripts, hats, bars and so on which result in a vector or matrix (e.g. transposing, inversion, selecting a submatrix, ...) are set boldfaced. Modifiers which result in a scalar (e.g. selecting a scalar entry of a vector) appear in normal letters.

1.4 Structure of this Thesis

This thesis is structured as follows: In Chapter 2, important and common background for the remainder of this thesis is given. Chapter 3 describes our contributions to feature matching on omnidirectional images while Chapter 4 presents our sparse reconstruction

framework for highly redundant data. The Chapters 5 and 6 concentrate on dense methods, presenting our extension for flexible multi-camera rigs and our approach to learn tracking the camera pose with neural networks. Chapter 7 presents our visualization framework. Finally, Chapter 8 concludes this work.

Chapter 2

Background

This chapter provides the basics necessary to understand the following chapters containing our contributions. The order of its content corresponds to the order of tasks that have to be performed for 3D scene reconstruction: Section 2.1 handles the description, tracking, detection and matching of image features. Section 2.2 contains a short introduction to geometrical optics covering transformations between affine spaces as well as transformations from or to images. Section 2.3 provides the mathematical background on optimization techniques and Sections 2.4 and 2.5 illustrate sparse geometrical and dense photometrical scene reconstruction.

2.1 Image Features

For reconstructing 3D scenes, a crucial and often first step is to find correspondences between images. This can potentially be done by brute force searching for similar patches in two images. Alternatives are the tracking or the matching of distinct feature points.

In the next sections, we will first examine measures for the similarity of patches followed by the tracking, detection and matching of distinct point features.

2.1.1 Similarity of Patches

A measure for patch similarity as presented in this section can be used to verify if two image regions are similar, to find the most similar patch for a query patch from another image or for feature matching.

The easiest approach to compute patch similarity in the sense of judging if the same scene surface is seen is to compare the image intensities only.

Intensity Based Similarity

The dissimilarity of two patches with pixels P from a reference image I_r and at least one secondary image I_s can be expressed as

$$\epsilon = \sum_{s \in S} \sum_{(u\ v) \in P} \Delta(I_r(u,v), I_s(\boldsymbol{\omega}(u,v;s))) \cdot w(u,v). \tag{2.1}$$

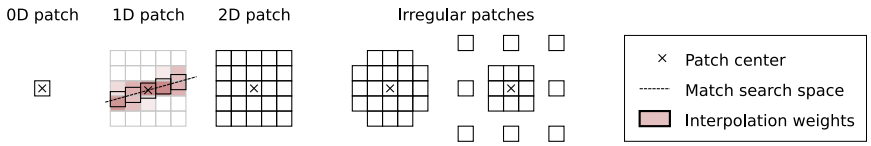


Figure 2.1: *Visualization of different patch shapes. 1D patches are usually aligned to the direction along which the correct match is expected. Therefore its intensities always are interpolated from the neighboring pixels.*

For every pixel in the patch, the intensity of the images is looked up at position $(u\ v)$ for the reference image and at a transformed position $\boldsymbol{\omega}(u, v; s)$ (patches are usually not at the same position in different images) for the secondary images. A difference operator $\Delta()$ is applied to the intensities and its result is weighted by $w()$ and summed up. If $(u\ v)$ denotes a position between pixels, bilinear interpolation is applied usually.

This basic formulation is used in many different specializations concerning the patch shape P, the parameter space of transformations $\boldsymbol{\omega}$, the difference operator Δ, the weighting function w and the number of secondary images $|S|$. Below, we introduce some of the possibilities:

Patch Shapes We list several patch shapes (see Figure 2.1) on which similarity can be evaluated. Larger patches most often lead to more discriminative similarity scores and can be used to recover more complex transformations $\boldsymbol{\omega}$. However, larger patches also increase the probability of covering non-flat surfaces although the patch is assumed to be flat. Smaller patches result in less discriminative similarity scores but do not suffer from the flat surface assumption.

Depending on the size and shape of a patch, the obtained similarity metrics are more or less discriminative or sensitive to certain properties of an image.

- 2D patches are most typically used in a quadratic shape with about 3-15 pixels in width and height. They perform well on local but otherwise unconstrained search tasks, i.e. the patch might undergo an arbitrary but small transformation from one image to the next.

- 1D patches (stripes) are commonly used when the search direction is linear. They are aligned parallel to the direction of the expected displacement and capture gradients orthogonal to this displacement direction very well.

- 0D patches (single pixels) usually require a greater amount of secondary images and a consistent set of transformations $\boldsymbol{\omega}(u, v, d; s)$, e.g. the geometrical transformation of a pixel with a certain depth to another known camera pose (see Equation 2.27). Given that, 0D patches can be as discriminative as 1D or 2D patches when only one transformation parameter (such as the distance to a reference camera) has to be recovered.

- Other irregular patches include all other thinkable sampling patterns on the image, e.g. circular patterns for rotational symmetry or patterns which exhibit lower sampling frequencies further away from the patch's center which reduces the effect of small differences in the periphery of the patch.

Transformations To model the warping of a patch from one image to another correctly, it might not be sufficient to treat it as purely translational. If the camera is rotated or the perspective on a surface is changed, more advanced patch transformation models should be used. Here we list the most important transformation models, from simple to more complex:

- Purely translational transformations

$$\boldsymbol{\omega}(u, v; t_u, t_v) = \begin{pmatrix} u + t_u \\ v + t_v \end{pmatrix} \tag{2.2}$$

  can be used for frontoparallel camera movements or when the rotation of the camera around the scene can be neglected.

- Translation + rotation + scale transformations

$$\boldsymbol{\omega}(u, v; t_u, t_v, \alpha, s) = \begin{pmatrix} s\cos\alpha & -s\sin\alpha & t_u \\ s\sin\alpha & s\cos\alpha & t_v \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.3}$$

  can model all patch transformations resulting from a camera that moves in arbitrary directions and rolls around its forward axis. Of course there are also variants which model only rotation ($s = 1$) or scaling ($r = 0$).

- Homographies

$$\boldsymbol{\omega}(u, v; m_i | i \in \{1, ..., 9\}) = \begin{pmatrix} m_1 & m_2 & m_3 \\ m_4 & m_5 & m_6 \\ m_7 & m_8 & m_9 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{2.4}$$

  cover all deformations that a planar surface patch might undergo in an image. Note that the result of the transformation has to be interpreted as homogeneous coordinate. If perspective effects like becoming smaller in the distance do not have to be concerned, a limited, affine homography can be used which has $m_7 = m_8 = 0$ and $m_9 = 1$.

If the camera geometry of a scene is known beforehand, the patch transformation parameters can often be derived from it. The parameters can however also be recovered purely from the images, e.g. by brute force searching or by iterative optimization (see

Sections 2.1.2 and 2.3). In general, the more transformation parameters have to be recovered, the more $\Delta(...)$ terms of Equation 2.1 are required. For recovering rotations and homographies, 2D patches are mandatory.

Intensity Differences Given a specification on the pixels that should be compared (patch shape) and a way of mapping them to another image (transformations), it is possible to look up corresponding intensity values from two images which are to be compared. This can be done in many different ways, concerning the influence of differences and the invariance against certain effects. Some important difference methods are listed below.

- The absolute difference of intensities I_1 and I_2,

$$\Delta(I_1, I_2) = |I_1 - I_2| \tag{2.5}$$

  is the most easy difference metric and results in the L1 norm on the sample pairs form the patches.

- The squared difference

$$\Delta(I_1, I_2) = (I_1 - I_2)^2 \tag{2.6}$$

  emphasizes strong outliers while not being sensitive to decent noise or small shading changes and is a reasonable choice for many applications. With squared differences, the dissimilarity ϵ evaluates to the squared L2 norm.

- To calculate the normalized cross correlation,

$$\Delta(I_1, I_2) = \frac{(I_1 - \bar{I_1})(I_2 - \bar{I_2})}{\sigma_1 \sigma_2} \tag{2.7}$$

  has to be used. This difference measure does not compare the absolute image values but analyzes the correlation of the differences to the average patch intensities $\bar{I_1}$ and $\bar{I_2}$. The magnitude is normalized by the standard deviations in the individual patches σ_1 and σ_1. This makes the operator invariant against brightness changes that affect the whole patch. Note that the this in fact calculates a correlation instead of a difference which means that ϵ must be maximized to find the most similar patches.

In principle, arbitrary loss functions like the Huber Loss [H+64] can be applied to make the dissimilarity robust against outlier pixels or to tweak it for certain image characteristics.

Gradient Images. The similarity of image patches is often determined on gradient images which could be Sobel or Laplace filtered versions of the intensity image. This can be beneficial because (1) gradient information only encodes brightness differences in a

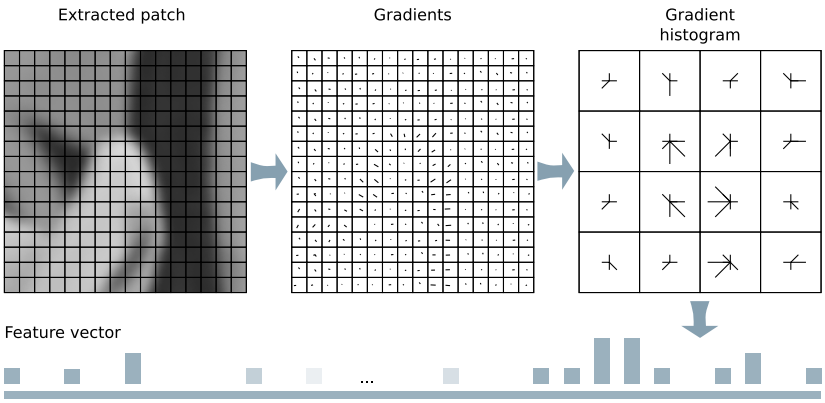


Figure 2.2: *Histograms of oriented gradients. To describe an image patch, gradients are determined per pixel and summed into histogram bins depending on the gradient's orientation. The concatenated values of all histogram's bins is used as the feature's description vector.*

local patch and is not influenced by image-wide biases and (2) derivating the image content to obtain a gradient image emphasizes high frequencies in the image which makes the precise localization of patches easier.

Feature Based Similarity

In contrast to intensity based patch similarity, a completely different and more robust and high-level approach for comparing image patches is the generation of a feature descriptor from the image patch close to a keypoint. Feature descriptors are usually hand-crafted transformations of image patches to lower dimensional representations which should be as discriminative as possible on the surface appearance but as invariant as possible to all other effects. This includes illumination, perspective, image noise, atmospheric effects and others. A frequently used feature descriptor for which many enhancements exist is the Histogram of Oriented Gradients (HOG).

Histogram of Oriented Gradients. The steps for generating a HOG from an image patch are illustrated in Figure 2.2. Gradient information is obtained for every pixel, e.g. by using the Sobel operator. Then, the image patch is spatially sectioned and for each section a histogram with bins corresponding to a range of gradient orientations is established. Usually, the histograms' bins are overlapping by 50% on each side. The gradient magnitudes of all pixels are summed into two histogram bins each, depending on the pixel position and gradient orientation. Finally, the sequence of all values of the histograms' bins forms a feature vector which is normalized in the end. Then, the feature vector can be compared with the feature vectors of other image patches to find similar ones.

Due to the aggregation of information from multiple pixels, the descriptor becomes invariant against small perspective transformations. By working on image gradients and by normalizing the feature vector in the end, invariance against brightness and contrast changes of the features is gained.

2.1.2 Tracking

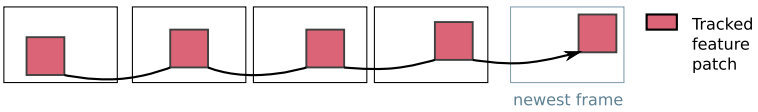


Figure 2.3: *Illustration of feature tracking. An image feature is tracked from one frame to the next based on the local gradient of the feature's patch.*

One possibility to obtain correspondences between frames with very small disparities, e.g. on videos, is feature tracking. Feature tracking is the process of keeping track of a feature's location from one video frame to the next. Under the assumption of only minor changes from frame to frame, a trivial, brute force approach would be to evaluate the dissimilarity ϵ (see Equation 2.1) for every combination of Transformation ω parameters (see Equations 2.2, 2.3, 2.4) in a small range. While this is feasible for simple, e.g. purely translational transformations and small, e.g. 1D patches, the course of dimensionality prevents us from applying this approach to more complex cases.

For faster convergence to the correct transformation, the search for the lowest dissimilarity can be guided by the gradients in the image. We illustrate the approach on the simple case of a 1D function and a purely translational transformation of the image content. The intensity at an image position shifted slightly by h

$$I(u+h) \approx I(u) + hI'(u) \tag{2.8}$$

can be approximated based on the image gradient I' (see Figure 2.4) which can be obtained for example by the Sobel operator. This approximation corresponds to a truncated Taylor series. To find the minimum dissimilarity, we set its derivative equal to zero and

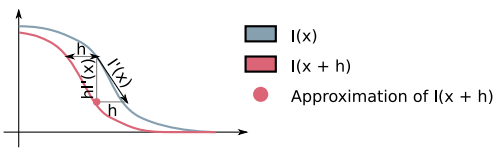


Figure 2.4: *Gradient based approximation of shifted function. The value of a shifted function can be approximated by multiplying the local gradient $I'(x)$ with the shift's offset h.*

apply the approximation above:

$$
\begin{aligned}
0 &= \frac{\partial \epsilon}{\partial h} && (2.9)\\
&= \frac{\partial}{\partial h} \sum_{u \in P} (I_2(u+h) - I_1(u))^2 && (2.10)\\
&\approx \frac{\partial}{\partial h} \sum_{u \in P} (I_2(u) + hI_2'(u) - I_1(u))^2 && (2.11)\\
&= \sum_{u \in P} 2I_2'(u)(I_2(u) + hI_2'(u) - I_1(u)) && (2.12)\\
\Leftrightarrow h &= \frac{\sum_{u \in P} I_2'(u)(I_1(u) - I_2(u))}{\sum_{u \in P} I_2'(u)^2} && (2.13)
\end{aligned}
$$

This approximation of h can be embedded in an iterative scheme to obtain the exact solution $\lim_{i \to \infty} h_i$:

$$
h_i = \begin{cases} i = 0: & 0 \\ i > 0: & h_{i-1} + \frac{\sum_{u \in P} I_2'(u+h_{i-1})(I_1(u) - I_2(u+h_{i-1}))}{\sum_{u \in P} I_2'(u+h_{i-1})^2} \end{cases} \tag{2.14}
$$

The concept of approximating a nonlinear function linearly and solving for the root iteratively is the Newton-Rhapson method. Its generalization is the Gauss-Newton method which can solve overdetermined problems with multiple parameters (e.g. all parameters of a homography transformation) for a local minimum given a function evaluation operator and a function derivative operator for all free parameters. For details on this method, refer to Section 2.3.2.

Multi-Scale Tracking

Feature tracking as described above converges to the closest minimum in the dissimilarity function ϵ which means that its spatial convergence radius is limited by the highest prominent frequency component in the image signal which can introduce local minima closer than the global optimum. A common approach to circumvent this problem is to low-pass filter the images for a larger convergence radius at the cost of tracking precision.

To combine both the advantages from low and high frequencies, it is beneficial to start tracking on strongly filtered images and to refine the tracked positions gradually by tracking on less and less filtered image versions. Since low-pass filtered images can be stored with smaller resolution without loosing information, it is reasonable to perform tracking on an image pyramid [Bou01].

2.1.3 Good Feature Candidates

While, in principle, arbitrary image patches can be compared or matched as described in the previous sections, the results might not be expressive in terms of describing if the same scene surface structure is seen. Negative examples include homogeneous areas or edges which look similar everywhere along the structure. Therefore, methods have to be used that extract reliable keypoints from an image which are likely to be tracked or matched to keypoints of other images that show the same surface point.

Good Features to Track

Shi and Tomasi [S$^+$94] have developed a method that finding optimal image patches for tracking by design. Note that the offset h is determined by division through $\sum_{u \in P} I'(u)^2$ in Equation 2.9. In the 2D case, this corresponds to a multiplication with the inverse of the matrix

$$\sum_{(u,v) \in P} \left(\boldsymbol{\nabla I}(u,v) \cdot \boldsymbol{\nabla I^T}(u,v) \right) = \sum_{(u,v) \in P} \begin{pmatrix} \left(\frac{\partial I}{\partial u}\right)^2 & \frac{\partial I}{\partial u}\frac{\partial I}{\partial v} \\ \frac{\partial I}{\partial u}\frac{\partial I}{\partial v} & \left(\frac{\partial I}{\partial v}\right)^2 \end{pmatrix}. \tag{2.15}$$

To be invertible in a stable way, this matrix (1) should have values above the image's noise level and (2) has to be well conditioned. This means that the matrix has to have (1) two relatively large eigenvalues with (2) approximately the same order of magnitude. This observation matches the intuitive interpretation of the matrix: small eigenvalues indicate a relatively homogeneous area while very differently scaled eigenvalues indicate a one-dimensional structure like an edge where features cannot be localized in one dimension - this is called the aperture problem. Two large eigenvalues are able to represent two-dimensional structures like corners or dots.

Good Features to Match

When the camera poses become more different from image to image, tracking the features is usually not possible. In this case, keypoints are extracted on every image individually and matched later. Since the image transformations are unknown and potentially large, keypoints for matching usually include a scale and an orientation attribute in addition to the position so that the feature descriptor can be extracted relative to all those attributes which results in a very similar descriptor even when the camera was rotated or moved away from or towards the feature.

Laplace Keypoint Detection A robust way for finding features on multiple scales employs a Laplace pyramid or stack. As illustrated in Figure 2.5, a Laplace stack is created by subtracting consecutive images from a Gauss stack. Laplace filtering is the difference of two low-pass filters and therefore band-pass filtering: increasing Laplace levels reveal

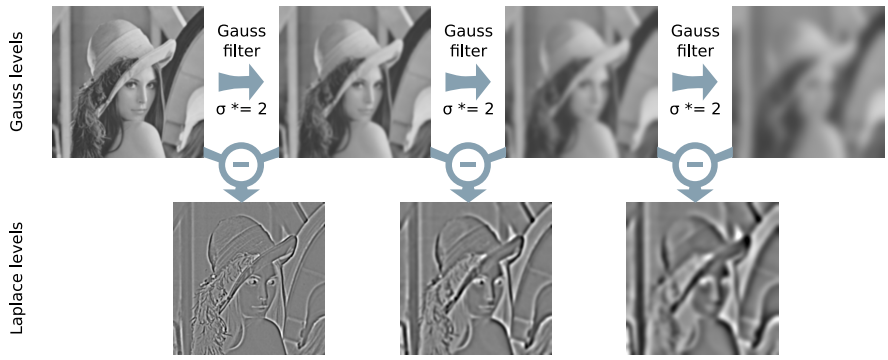


Figure 2.5: *Comparison of Gauss and Laplace levels. Laplace levels can be created as difference of two Gauss filtered images. Higher Laplace levels reveal lower frequency content.*

lower frequency content, so it is reasonable to assign the standard deviation σ of the corresponding Gauss level as scale to the Laplace level. In a Laplace stack, keypoints can be found by searching for local extremes in the (u, v, σ) domain, i.e. a keypoint is defined as a pixel which has a greater or smaller value than all of its neighbors on the same and the neighboring Laplace levels. The keypoint's position and scale are refined by finding the extreme in a 2nd order, 3D Taylor expansion in the position / scale space. Harris corner detection (see next paragraph) is applied to filter keypoints on edges. Finally, an orientation can be assigned to the keypoint based on the gradient on the Gauss level of its scale.

Typically, such features are found in corners, at irregularities along edges and on the inner side of curves where the curvature is maximal.

Harris Corner Detection Harris and Stephens [HS88] developed a more simple feature detector. They propose to search for blob like structures whose image patches exhibit a relevant curvature in multiple directions. This can be done by calculating the determinant of the second order derivative matrix, the Hessian matrix

$$\boldsymbol{H}(u, v) = \begin{pmatrix} I_{uu}(u, v) & I_{uv}(u, v) \\ I_{uv}(u, v) & I_{vv}(u, v) \end{pmatrix} \tag{2.16}$$

for each pixel. The second order derivatives I_{**} for each pixel can be obtained by finite differences. Two large eigenvalues of the Hessian matrix indicate that there is a two dimensional curvature present at $I(u, v)$. Mikolajczyk et al. [MTS+05] conclude that it is sufficient to apply a threshold on the Hessian's determinant. Since the determinant is proportional to the eigenvalues and therefore to the curvature, it can be used as a measure for the scale of the keypoint.

2.1.4 Matching

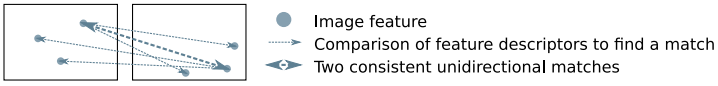


Figure 2.6: *Finding feature matches on a pair of images. For each feature of one image, the best unidirectional match to the other image is searched. Only matches that are consistent in both directions are used.*

Finding correspondences between images is more difficult when the keypoints move too far for tracking from one image to the next. In this case, keypoints are detected on each image individually as described in the previous section and corresponding feature descriptors are extracted. In a next step, the feature descriptors can be matched.

Matching is usually done first for all image pairs. When pairwise matches are available, they are transitively extended to tracks spanning multiple images.

Matching Image Pairs. When matches between features of two images should be found (see Figure 2.6), a common approach is to

1. find a good unidirectional match $(f_i^1, f_j^2)_{1\to 2}$ for each feature f_i^1 of the first image I_1 in the second image I_2,

2. find a good unidirectional match $(f_j^2, f_i^1)_{2\to 1}$ for each feature f_i^2 of the second image I_2 in the first image I_1 and

3. extract consistent matches $\{f_i^1, f_j^2\}$: $\quad \exists(f_i^1, f_j^2)_{1\to 2} \wedge \exists(f_j^2, f_i^1)_{2\to 1}$.

This reduces the amount of false matches significantly because consistency is much more unlikely for false matches than for correct ones. In addition, it ensures a bijective mapping between the matched features of two images.

To find unidirectional feature matches, the dissimilarity ϵ or the euclidean distance between two HOG feature vectors can be utilized. For each feature in a reference image, the best and the second best fitting features are searched in the secondary image. The best fitting feature is considered a match if

$$\frac{\epsilon_{best}}{\epsilon_{second}} < t \tag{2.17}$$

where t is a user defined threshold. This ensures that no match is recorded when there are multiple similar features which cannot be differentiated clearly.

Note that finding the two most similar features of a secondary image for each feature in the reference image is one of the most compute intensive tasks in feature matching because of its runtime in $O(n^2)$ where n is proportional to the number of features

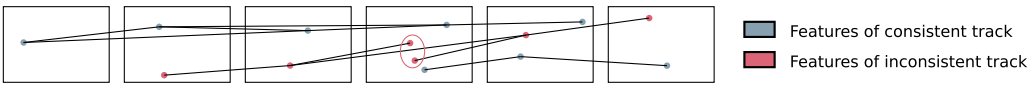


Figure 2.7: *Illustration of consistent and inconsistent tracks from pairwise image feature matches.*

or pixels in the image. The runtime can be reduced by using (approximate) spatial acceleration structures for euclidean distance based matching including randomized KD-forests and K-Means trees [ML14], Locality Sensitive Hashing [DIIM04], Vector Quantization [LBG80], Product Quantization [KS99] or Product Quantization Trees [WWSHL16]. If the image transformation $\boldsymbol{\omega}$ is (partially) known, the search can also be restricted to certain regions in the secondary image.

Matching Sets of Images. When features should be matched among multiple images, one usually establishes feature tracks (inspired by the term from feature tracking as described in Section 2.1.2) which should correspond to the same 3D surface point. One track contains all features that are transitively coupled by image pair feature matches $\{f_i^m, f_j^n\}$.

To find optimal tracks in an image set, all pairwise matches between all images would be needed which has a runtime in $O(n^2)$ with n representing the number of images. Large scale reconstruction methods deal with this problem by thinning out the image pairs that are actually worth being matched based on methods that determine the similarity of whole images based on Fourier transforms [OT02], Gabor filters [RM04], wavelet decompositions [TMFR03], analysis of biological systems [SI07] or with learning based methods [KGC15]. Alternatively, the next image pairs to be matched can be determined based on number of feature matches that have been found on previously matched image pairs [KTT+12].

For further processing, we are only interested in *consistent* tracks, i.e. tracks which contain not more than one feature per image. The development of inconsistent tracks is illustrated in Figure 2.7 and would intuitively represent a surface point seen in one image that corresponds to two surface points in another image (which is what we want to avoid).

2.1.5 Common Feature Matching and Tracking Frameworks

Many commonly used point feature matching and tracking frameworks are assembled of the building blocks described in the previous subsections or some modified (simplified or specialized) versions. We give an overview of the most important ones:

KLT The Kanade-Lucas-Tomasi feature tracker [TK91] is the de-facto standard for tracking sparse point features. Good features to track are detected as described in Section 2.1.3 and Tracking is performed as described in Section 2.1.2. Concerning patch similarity (Section 2.1.1), it uses 2D image patches, optimizes for the squared intensity difference and tracks from frame to frame based on a purely translational

image patch transformation. The tracking is verified based on an affine homography transformation model. There exist extensions for image pyramid based KLT tracking [Bra00].

SIFT Scale-invariant feature transform [Low04] is a feature detection, description and matching framework. Feature keypoints are detected in a Laplace pyramid as described in Section 2.1.3. The features are described as a vector from a HOG (Section 2.1.1) which is extracted from a patch sampled relative to the keypoint's orientation and scale. Feature matching for pairs of images is performed as described in Section 2.1.4. In addition, the features' rotations are checked for consistency and inconsistent matches are removed (see Chapter 3 for more details on that part).

SURF Speeded up robust features [BTVG06] is inspired by SIFT but tuned for higher speed. Feature keypoints are detected with Harris corner detection as described in Section 2.1.3. The feature description is very similar to SIFT as well but obtained differently: while SIFT creates a Gauss pyramid first by convolution with a Gauss kernel and extracts gradients from there by convolution with a Sobel kernel, SURF convolves the gradient operator itself with a Gaussian kernel and applies it to the whole image:

$$\overbrace{\underbrace{I * \mathit{Gauss\ kernel}}_{\text{Gauss pyramid}} * \mathit{Sobel\ kernel}}^{\text{SIFT}} \cong \overbrace{I * \underbrace{\mathit{Gauss\ kernel} * \mathit{Sobel\ kernel}}_{\text{Filtered gradient operator}}}^{\text{SURF}} \tag{2.18}$$

However, the responses from the filtered gradient operators are not evaluated by convolution but approximated by evaluating Haar wavelets of a corresponding size on an integral image (which can be done in constant time, independently of the Haar wavelet's size).

GLOH Gradient location-orientation histogram [MS05] is an extension to SIFT which uses a different sampling pattern for the described feature patch: it uses a log polar sampling distribution, i.e. samples are placed on concentric circles around the feature center with increasing distance. In addition, a PCA is used to reduce the dimensionality of the feature vectors.

Binary Algorithms There are several feature detectors (e.g. FAST [RD06]) and descriptors (e.g. BRIEF [CLSF10], BRISK [LCS11] or FREAK [AOV12]) or combinations of both (ORB [RRKB11]) which encode the comparisons of image intensities from certain areas of a patch into a binary vector and use this representation to determine if a keypoint has been found or to match different features. Such algorithms are usually much faster than the ones described previously but do not localize with sub-pixel precision which makes them less useful for high quality camera pose and geometry reconstruction.

CNN-based Feature Descriptions are obtained by training a convolutional neural network to transform image patches to feature vectors in a way so that the distance between the feature vectors describes if two patches show the same surface point. One the one hand, such algorithms usually outperform hand-crafted feature descriptors [FDB14]. On the other hand, while hand-crafted feature descriptors are mostly inspired by cognitive processing, they can be expected to be able to process the patches similar to humans by design while the performance of learned feature descriptors depends mainly on the training data. In fact it has been shown that neural networks can be fooled by virtually invisible but specific changes to an image patch [MDFF16].

By using one of the frameworks described above, one can extract image positions corresponding to the same 3D scene points from image collections and videos. This information can be used together with findings from geometrical optics as described in the next section to recover camera calibration parameters and poses as well as scene structure.

2.2 Geometry and Transformations

In this section we give an overview over geometrical properties of the setups we use for reconstruction. We show how to transform points and directions from images to 3D view space and back as well as between euclidean spaces. Concatenating such transformations enables us to project 3D surface points back into the images that captured the scene by transforming them first to the view space of the camera and then to the image. It also enables us to project points with depth information from one image into another by transforming it to its camera's view space, then to the other camera's view space and finally to the other image (see Figure 2.8).

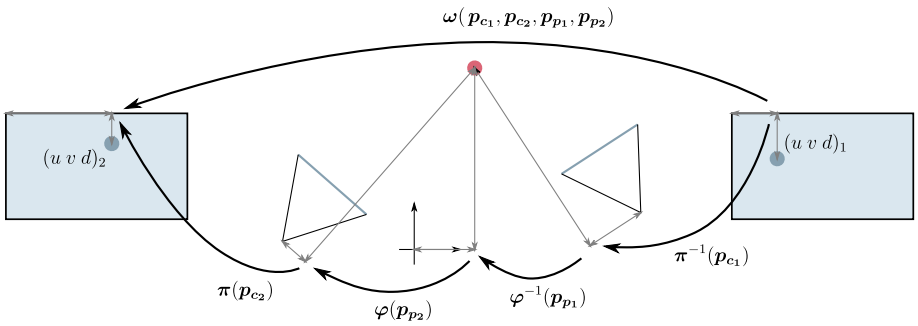


Figure 2.8: *Transformation chain for transforming an image point with depth from one image to another.*

2.2.1 Camera Transformations in Euclidean Spaces

We discern between different euclidean spaces when we deal with transformations between the images of cameras:

World space This is the reference space for all objects and all cameras. Its origin is most often placed at the position of the first camera pose that is used in the reconstruction.

View space In the view space that belongs to a certain camera pose, the camera is located in the origin and the space is oriented according to the principal axes of the camera. In this work, the axes are aligned as illustrated in Figure 2.9.

Rig space We introduce the rig space as common space of a multi-camera rig with all of its attached cameras. The origin of the rig space may be placed at a distinguished position on the rig, e.g. the mount point. Its orientation is usually similar to the orientation of the most important camera.

Between those spaces, only transformations from the Lie group SE(3) are concerned. They consider arbitrary rotations and translations, but no scaling or other effects. The

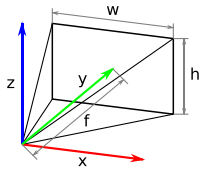


Figure 2.9: *Axis directions in view space and parameters for the pinhole camera model.*

SE(3) group can be expressed in matrix from:

$$SE(3) = \left\{ \boldsymbol{M} \mid \quad \boldsymbol{M} = \begin{pmatrix} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0} & 1 \end{pmatrix}, \boldsymbol{R} \in \mathbb{R}^{3x3}, \boldsymbol{t} \in \mathbb{R}^3, \boldsymbol{R^T R} = \boldsymbol{R R^T} = \boldsymbol{I}, |\boldsymbol{R}| = 1 \right\} \tag{2.19}$$

The transformation matrix $\boldsymbol{M}$ for homogeneous coordinates $(x\ y\ z\ 1)^T$ contains an orthonormal rotational part $\boldsymbol{R}$ and a translation vector $\boldsymbol{t}$.

Note that we favor rotation matrices throughout this thesis over quaternions, angel/axis representations and Euler angles, because rotation matrices are easy to parse, integrate trivially into transformation matrices which describe rotation and translation in a uniform representation and they are, in contrast to e.g. Euler angles, singularity free.

Commonly, the transformation matrix which transforms points from world space to view space is called view matrix. Intuitively, it encodes the pose of the world space origin in the view space in $\boldsymbol{R}$ and $\boldsymbol{t}$. This is however difficult to conceive. Instead, we prefer to interpret the camera pose's *alignment* matrix which is the inverse of the view matrix and encodes the pose of the camera in the world space in $\boldsymbol{R}$ and $\boldsymbol{t}$, transforming points from view space to world space.

2.2.2 Camera Image Transformations

Cameras can be seen as devices that measure radiance along certain rays and map them to pixels. Consequently, the most generic geometrical camera model [GN01] is a mapping from arbitrary rays $\boldsymbol{p}(t) = \boldsymbol{o} + t\boldsymbol{d}$ to pixels of an image $(u\ v)$.

In this thesis we focus - as it is done for the whole field of geometry reconstruction from images - to camera models which are bijective, continuously differentiable and strictly monotonic. Such models can easily be used in non-linear optimization frameworks which are utilized to improve camera and scene structure parameters in common. A camera model which has these properties is the pin-hole camera model. In the next paragraphs, we introduce the classical pinhole camera and some extensions to capture real cameras more realistically.

Linear Camera Models

The most simple perspective camera model, the pinhole camera, allows to map 3D view space points $(x\ y\ z)$ to image coordinates $(u\ v)$ and depth d:

$$\begin{aligned} u_s &= x \cdot {}^{f}/_{y} + {}^{w}/_{2} \\ v_s &= -z \cdot {}^{f}/_{y} + {}^{h}/_{2} \\ d &= y \end{aligned} \tag{2.20}$$

As shown in Figure 2.9, w and h denote the width and height of the image plane or the image resolution and f represents the focal length: the distance between image plane and the optical center of the camera.

In digital cameras, there are two sources of linear error: It might happen that a image sensor has non-square pixels which results in a different horizontal and vertical focal length. In addition, the image sensor might not be aligned perfectly centered on the optical axis of the camera. A camera model which includes such linear effects is given by

$$\begin{aligned} u_l &= x \cdot {}^{f_u}/_{y} + c_u \\ v_l &= -z \cdot {}^{f_v}/_{y} + c_v \end{aligned} \tag{2.21}$$

where c_u and c_v encode the pixel position on the image that is intersected by the optical axis, the principal point. The two camera models shown so far are linear functions and thus easily invertible. This means that given image coordinates and depth, a point in view space can be recovered, or: given the image coordinates only, an arbitrary depth can be assumed to obtain a view space direction vector.

Nonlinear Camera Models

The models above result in rectified images in the sense that straight view space lines remain straight when transformed to the image. On real cameras, one can observe different lens effects that lead to bending of straight lines. There is (1) radial distortion which is a rotational symmetric effect originating from imperfect lens shapes and (2) tangential distortion which has its origin in a slightly non-planar alignment of lens and image sensor. The second linear model above can be extended for radial and tangential distortion

as follows:

$$
\begin{aligned}
u_r &= {}^{x}/_{y} \\
v_r &= -{}^{z}/_{y} \\
u_d &= u_r \overbrace{\frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6}}^{\text{radial distortion}} + \overbrace{2p_1 u_r v_r + p_2 (r^2 + 2u_r^2)}^{\text{tangential distortion}} \\
v_d &= v_r \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_1 (r^2 + 2v_r^2) + 2p_2 u_r v_r \\
\text{where} \quad r^2 &= u_r^2 + v_r^2 \\
u &= f_u \cdot u_d + c_u \\
v &= f_v \cdot v_d + c_v
\end{aligned}
\tag{2.22}
$$

At first, the perspective division is applied to the view space coordinates. Then, the radial distortion (three symmetric polynomial terms with coefficients k_1, k_2, k_3 and three symmetric inverse polynomial terms with coefficients k_4, k_5, k_6) and the tangential distortion (one vertical and one horizontal term with coefficients p_1, p_2) are applied. Finally, the correct focal length is multiplied and the image is shifted according to the principal point.

This polynomial mapping cannot by inverted trivially anymore. In practice, a lookup map is created which maps from image coordinates according to Equation 2.22 to image coordinates according to Equation 2.21 and back. This way, a keypoint found in an image can first be transformed to its rectified position via lookup and then be converted to view space with Equation 2.21. Since the relative distortion of two pixels is usually very small compared to the pixel offset, linear interpolation on the lookup map can be applied to handle sub-pixel precise image positions.

2.2.3 Transformation Functions

Based on the content of this chapter, we define several transformation functions for an easier representation of transformations (see Figure 2.8), their concatenations and their parameters throughout the thesis. The transformation from view space to image space is defined as

$$
\boldsymbol{\pi} : \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_c}|} \to \mathbb{R}^3, \quad (\boldsymbol{x}; \boldsymbol{p_c}) \mapsto \boldsymbol{\pi}(\boldsymbol{x}; \boldsymbol{p_c}) = (u\ v\ d) \tag{2.23}
$$

which maps a view space point $\boldsymbol{x}$ to an image point + depth depending on a set of camera parameters $\boldsymbol{p_c}$ (e.g. w, h, f, k_i, ...). The corresponding inverse is defined as

$$
\boldsymbol{\pi^{-1}} : \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_c}|} \to \mathbb{R}^3, \quad ((u\ v\ d); \boldsymbol{p_c}) \mapsto \boldsymbol{\pi^{-1}}((u\ v\ d); \boldsymbol{p_c}) = \boldsymbol{x} \tag{2.24}
$$

Similarly, we define the transformation to view space and its inverse which depend on a set of camera pose parameters $\boldsymbol{p_p}$:

$$\boldsymbol{\varphi}\colon \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_p}|} \to \mathbb{R}^3, \qquad (\boldsymbol{x}; \boldsymbol{p_p}) \mapsto \boldsymbol{\varphi}(\boldsymbol{x}; \boldsymbol{p_p}) = \boldsymbol{x_v} \tag{2.25}$$

$$\boldsymbol{\varphi^{-1}}\colon \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_p}|} \to \mathbb{R}^3, \qquad (\boldsymbol{x_v}; \boldsymbol{p_p}) \mapsto \boldsymbol{\varphi^{-1}}(\boldsymbol{x_v}; \boldsymbol{p_p}) = \boldsymbol{x} \tag{2.26}$$

Given the definitions above, we can easily express or calculate the transformation or warping from one image to another as

$$\boldsymbol{\omega}((u\ v\ d)_2; \boldsymbol{p_{c_1}}, \boldsymbol{p_{c_2}}, \boldsymbol{p_{p_1}}, \boldsymbol{p_{p_2}}) = [\boldsymbol{\pi}(\boldsymbol{p_{c_2}}) \circ \boldsymbol{\varphi}(\boldsymbol{p_{p_2}}) \circ \boldsymbol{\varphi}^{-1}(\boldsymbol{p_{p_1}}) \circ \boldsymbol{\pi}^{-1}(\boldsymbol{p_{c_1}})](u\ v\ d)_1. \tag{2.27}$$

The parameters $\boldsymbol{p_c}$ are usually called the *intrinsic* parameters of a camera while $\boldsymbol{p_p}$ are called *extrinsic* parameters.

Note that all parts of this warping function as well as the function en block are continuously differentiable usually, except for some singularities that should never be captured like a camera trajectory intersecting a scene point's position (which would result in $y = 0$ in view space and lead to a division by zero). This makes the warping function and all of its parameters a convenient subject for the optimization techniques presented in the next section.

Epipolar Geometry

Given a linear camera model without polynomial distortions, a pixel $(u\ v)$ in one view can be transformed to a line in another view which consists of the points

$$\{\boldsymbol{\omega}((u\ v\ d)_2; \boldsymbol{p_{c_1}}, \boldsymbol{p_{c_2}}, \boldsymbol{p_{p_1}}, \boldsymbol{p_{p_2}}) | d \in \mathbb{R}\} \tag{2.28}$$

and is called epipolar line (EPL) (see Figure 2.10). The epipolar lines of all pixels have one common point for $d = 0$ which is called epipole. It represents the optical center of one camera in the image of the other.

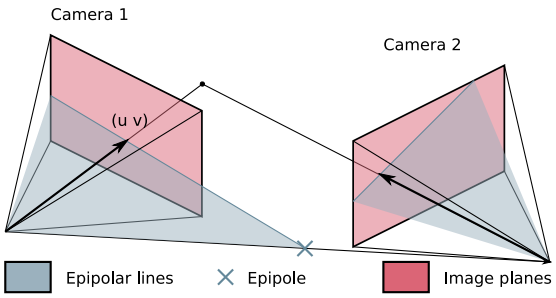


Figure 2.10: *Epipolar geometry with epipolar lines and the epipole of camera 2 in the image of camera 1.*

2.3 Optimization Techniques

There are several solvers involved in 3D reconstruction. The most prominent example is Bundle Adjustment which refines a scene reconstruction by distributing the reprojection errors equally between all points and camera poses (see Section 2.4), but optimization is also involved in dense camera tracking (see Section 2.5), linear camera pose estimation (see Section 4.3.1) or image feature tracking (see Section 2.1.2).

In this section we introduce the optimization algorithms utilized in this thesis. We start from simple linear solvers in Section 2.3.1 and build upon those to explain how nonlinear problems can be solved and how sparse data can be exploited in Sections 2.3.2 and 2.3.3.

2.3.1 Linear Solvers

Linear problems have the form

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad | \quad \boldsymbol{A} \in \mathbb{R}^{m \times n}, \quad \boldsymbol{x} \in \mathbb{R}^{n}, \quad \boldsymbol{b} \in \mathbb{R}^{m} \tag{2.29}$$

where a set of parameters $\boldsymbol{x}$ has to be found so that the equation holds. If $m = n$ and $\boldsymbol{A}$ has full rank, the problem is determined and can be solved by inverting $\boldsymbol{A}$. If $n > m$, the problem is underdetermined which means that there are less constraints on $\boldsymbol{b}$ than parameters in $\boldsymbol{x}$ which means that there is often not only a single solution but a whole space of solutions.

The most common case in the field of 3D reconstruction from images is that $m > n$, i.e. that the problem is overdetermined: there are more constraints on $\boldsymbol{b}$ than parameters in $\boldsymbol{x}$ to generally fulfill the constraints and most commonly, the constraints are not perfectly consistent due to imprecisions. In this case, it is intended to find a least squares solution where the residual

$$R = \sum_{i=1}^{m} \boldsymbol{\epsilon}_i^2 \qquad\qquad \boldsymbol{\epsilon}_i = \boldsymbol{b}_i - \sum_{j=1}^{n} \boldsymbol{A}_{ij}\boldsymbol{x}_j \tag{2.30}$$

should be minimized. It can be shown ([GL96], Section 5.3), that finding the global minimum by

$$\frac{\partial R}{\partial \boldsymbol{x}} = \boldsymbol{0} \tag{2.31}$$

leads to the normal equations

$$\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{b} \tag{2.32}$$

$$\Leftrightarrow \quad \boldsymbol{x} = (\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A})^{-1}\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{b}. \tag{2.33}$$

This can intuitively be seen as replacing the inversion of $\boldsymbol{A}$ which is not possible by the pseudo-inverse $(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A})^{-1}$. To guaranty invertibility, $\lim_{\epsilon \to 0}(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A} + \epsilon \boldsymbol{I})^{-1}$ can be used.

2.3.2 Nonlinear Solvers

Nonlinear least squares problems have the form

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{b}, \qquad \boldsymbol{f} : \mathbb{R}^n \mapsto \mathbb{R}^m, \qquad m >> n, \tag{2.34}$$

so they are essentially an overdetermined system of equations. We cannot follow the same approach as for linear overdetermined systems and just minimize the residual error (compare to Equation 2.30)

$$R = \sum_{i=1}^{m} \boldsymbol{\epsilon}_i^2 \qquad\qquad \boldsymbol{\epsilon} = \boldsymbol{b} - \boldsymbol{f}(\boldsymbol{x}) \tag{2.35}$$

again because setting the differentiation of the general function $\boldsymbol{f}$ can be arbitrarily complicated if $\boldsymbol{f}$ can be differentiated everywhere at all. One can instead approximate $\boldsymbol{f}$ locally and linearly at $\boldsymbol{x}$ (compare to Figure 2.4 and Equation 2.8):

$$\boldsymbol{f}(\boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}}) \approx \boldsymbol{f}(\boldsymbol{x}) + \boldsymbol{J}\boldsymbol{\delta}_{\boldsymbol{x}} \tag{2.36}$$

$\boldsymbol{J}$ is the Jacobian of $\boldsymbol{f}$ at $\boldsymbol{x}$. With this approximation, the minimization problem can be reformulated to find the offset $\boldsymbol{\delta}_{\boldsymbol{x}}$ for the current parameter set $\boldsymbol{x}$ that minimizes R most:

$$R = \|\boldsymbol{b} - \boldsymbol{f}(\boldsymbol{x} + \boldsymbol{\delta}_{\boldsymbol{x}})\|^2 \approx \|\boldsymbol{b} - \boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{J}\boldsymbol{\delta}_{\boldsymbol{x}}\|^2 = \|\boldsymbol{\epsilon} - \boldsymbol{J}\boldsymbol{\delta}_{\boldsymbol{x}}\|^2 \tag{2.37}$$

Now the quantity that has to be minimized is the parameter vector of a linear overdetermined system

$$\boldsymbol{0} = \boldsymbol{\epsilon} - \boldsymbol{J}\boldsymbol{\delta}_{\boldsymbol{x}} \qquad \Leftrightarrow \qquad \boldsymbol{J}\boldsymbol{\delta}_{\boldsymbol{x}} = \boldsymbol{\epsilon} \tag{2.38}$$

which can be solved according to Section 2.3.1. Since the calculated residual is only based on an approximation, the correct solution has to be found iteratively.

Trust Region

When Equation 2.38 is solved, $\boldsymbol{\delta}_{\boldsymbol{x}}$ represents the update step that would lead to the optimal solution for a linear function. Depending on the shape of the actual function $\boldsymbol{f}$, this step can however be too large or too small. While too small steps at least reduce the magnitude of the residual, too large steps might lead to chaotic behavior during the optimization. Therefore it is reasonable to approximate how well the linear approximation fits the optimized function and define a trust region for $\boldsymbol{\delta}_{\boldsymbol{x}}$ in which a decrease of the optimized residual is guaranteed.

Levenberg Marquard Algorithm. The Levenberg Marquard (LM) algorithm [Lev44] is a method for adapting the step size to guaranty beneficial updates which do converge

reasonably fast. It uses the augmented normal equations (compare to Equation 2.32)

$$(\boldsymbol{J^T J} + \mu \boldsymbol{I})\boldsymbol{\delta_x} = \boldsymbol{J^T \epsilon}. \tag{2.39}$$

where μ serves as damping parameter. When the equation

$$\boldsymbol{\delta_x} = (\boldsymbol{J^T J} + \mu \boldsymbol{I})^{-1} \boldsymbol{J^T \epsilon} \tag{2.40}$$

is evaluated with $\mu = 0$, the usual normal equations are evaluated. In this case, $\boldsymbol{\delta_x}$ corresponds to a Gauss Newton step: a linearly extrapolated guess of the minimum's parameter set. When μ is raised, the $(\boldsymbol{J^T J} + \mu \boldsymbol{I})^{-1}$ matrix converges to $(1/\mu) \cdot \boldsymbol{I}$ making $\boldsymbol{\delta_x} \to (1/\mu)\boldsymbol{J^T \epsilon}$. Since it can be shown that the gradient of $\boldsymbol{f}$ at $\boldsymbol{x}$ points towards $-\boldsymbol{J^T \epsilon}$, high damping factors lead to small optimization steps towards the direction of the steepest descent: towards the negative gradient.

There are several heuristics for adjusting the damping factor. Usually, an updated parameter set is calculated to check the new residual R. If the residual increased, the update is discarded and μ is increased by a certain factor. If the residual decreased, the update is accepted and μ is decreased by a certain factor.

Constraint Certainty

It may happen that there is information about the certainty of the constraints $\boldsymbol{b}$ on which the optimization is applied, e.g. $\boldsymbol{b}$ might contain feature keypoint positions from images which can be determined more precisely on high gradients or lower scales (Section 2.1.3). This confidence information can be respected by extending the augmented normal equations by a confidence matrix $\boldsymbol{\Sigma_\epsilon}$ which encodes the confidence or inverse variance of every $\boldsymbol{b}_i$ on its diagonal elements:

$$(\boldsymbol{J^T \Sigma_\epsilon^{-1} J} + \mu \boldsymbol{I})\boldsymbol{\delta_x} = \boldsymbol{J^T \Sigma_\epsilon^{-1} \epsilon}. \tag{2.41}$$

2.3.3 Exploiting Structured Data

The nonlinear solver and especially the LM algorithm is the common choice for optimizing camera calibration, camera pose or other geometrical scene properties at once because it approximates the local slope of the loss function very precisely by differentiating for every parameter that should be optimized and proposes therefore close-to-optimal improvements $\boldsymbol{\delta_x}$ on smooth functions. However, this comes with the high computational cost of calculating all necessary derivatives and, more important, the cost of inverting $(\boldsymbol{J^T J} + \mu \boldsymbol{I})$. Fortunately, certain structures in the data used for optimization can be exploited to reduce those costs.

The main idea in exploiting structure is to analyze the derivatives encoded in the Jacobian $\boldsymbol{J}$ for zero values which appear if a constraint (row of $\boldsymbol{J}$) is independent of a parameter (column of $\boldsymbol{J}$).

i	Indexes points		
j	Indexes camera poses		
l	Indexes calibration / global parameter sets		
$\|\boldsymbol{A}\|$	Number of parameters per camera pose		
$\|\boldsymbol{B}\|$	Number of parameters per point		
$\|\boldsymbol{C}\|$	Number of parameters per calibration set		
$\boldsymbol{A_{ij}}$	Error residual derivative with respect to camera pose parameters $\boldsymbol{p_p}$ ($1 \times \|\boldsymbol{A}\|$ Matrix)		
$\boldsymbol{B_{ij}}$	Error residual derivative with respect to point parameters $\boldsymbol{x}$ ($1 \times \|\boldsymbol{B}\|$ Matrix)		
$\boldsymbol{C_{lij}}$	Error residual derivative with respect to calibration parameters $\boldsymbol{p_c}$ ($1 \times \|\boldsymbol{C}\|$ Matrix)		
$\boldsymbol{D_{l_1 l_2}}$	$\|C\| \times \|C\|$ Matrix	$\boldsymbol{U_j}$	$\|A\| \times \|A\|$ Matrix
$\boldsymbol{E_{lj}}$	$\|C\| \times \|A\|$ Matrix	$\boldsymbol{V_i}$	$\|B\| \times \|B\|$ Matrix
$\boldsymbol{F_{li}}$	$\|C\| \times \|B\|$ Matrix	$\boldsymbol{W_{ij}}$	$\|C\| \times \|B\|$ Matrix

Table 2.1: *Notation for Subsection 2.3.3.*

In this section, we aim on exploiting the structure of a Sparse Bundle Adjustment (SBA) problem. It is based on the work of Triggs et al. [TMHF99] concerning the mathematical derivations and Lourakis and Argyros [LA09] concerning an efficient implementation. Given a model of a scene consisting of cameras which have taken images from several camera poses and the 3D scene points they have observed, we want to minimize the distance between (1) the keypoint positions $(u\, v)$ as they have been observed in the images and (2) the synthetic projection of the 3D scene points $\boldsymbol{x}$ to the images based on camera parameters $\boldsymbol{p_c}$ and camera pose parameters $\boldsymbol{p_p}$:

$$R = \sum_{i=1}^{m} \|(u\, v) - (\pi(\boldsymbol{p_{c_l}}) \circ \varphi(\boldsymbol{p_{p_j}}))(\boldsymbol{x_i})\|^2 \tag{2.42}$$

Clearly, the value of one constraint ($\|...\|^2$ term) depends only on the ith point $\boldsymbol{x_i}$, the jth camera pose p_j and the lth camera c_l. In our following example with one camera, two camera poses and three points, this results in a Jacobian exhibiting the following structure:

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{C_{111}} & \boldsymbol{A_{11}} & \boldsymbol{0} & \boldsymbol{B_{11}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C_{112}} & \boldsymbol{0} & \boldsymbol{A_{12}} & \boldsymbol{B_{12}} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{C_{121}} & \boldsymbol{A_{21}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B_{21}} & \boldsymbol{0} \\ \boldsymbol{C_{122}} & \boldsymbol{0} & \boldsymbol{A_{22}} & \boldsymbol{0} & \boldsymbol{B_{22}} & \boldsymbol{0} \\ \boldsymbol{C_{131}} & \boldsymbol{A_{31}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B_{31}} \\ \boldsymbol{C_{132}} & \boldsymbol{0} & \boldsymbol{A_{32}} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{B_{32}} \end{pmatrix}. \tag{2.43}$$

Given this Jacobian, the matrix $\boldsymbol{J^T \Sigma_\epsilon^{-1} J} + \mu \boldsymbol{I}$ in Equation 2.41 has the following structure:

$$\left(\begin{array}{ccc|ccc} \boldsymbol{D_{11}^*} & \boldsymbol{E_{11}} & \boldsymbol{E_{12}} & \boldsymbol{F_{11}} & \boldsymbol{F_{12}} & \boldsymbol{F_{13}} \\ \boldsymbol{E_{11}^T} & \boldsymbol{U_1^*} & \boldsymbol{0} & \boldsymbol{W_{11}} & \boldsymbol{W_{21}} & \boldsymbol{W_{31}} \\ \boldsymbol{E_{12}^T} & \boldsymbol{0} & \boldsymbol{U_2^*} & \boldsymbol{W_{12}} & \boldsymbol{W_{22}} & \boldsymbol{W_{32}} \\ \hline \boldsymbol{F_{11}^T} & \boldsymbol{W_{11}^T} & \boldsymbol{W_{12}^T} & \boldsymbol{V_1^*} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{F_{12}^T} & \boldsymbol{W_{21}^T} & \boldsymbol{W_{22}^T} & \boldsymbol{0} & \boldsymbol{V_2^*} & \boldsymbol{0} \\ \boldsymbol{F_{13}^T} & \boldsymbol{W_{31}^T} & \boldsymbol{W_{32}^T} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{V_3^*} \end{array}\right) \tag{2.44}$$

$$\boldsymbol{D_{l_1 l_2}} = \sum_{j=1}^{2}\sum_{i=1}^{3} \boldsymbol{C_{l_1 ij}^T \Sigma_{\epsilon_{ij}}^{-1} C_{l_2 ij}} \qquad \boldsymbol{E_{lj}} = \sum_{i=1}^{3} \boldsymbol{C_{lij}^T \Sigma_{\epsilon_{ij}}^{-1} A_{ij}}$$

$$\boldsymbol{F_{li}} = \sum_{j=1}^{2} \boldsymbol{C_{lij}^T \Sigma_{\epsilon_{ij}}^{-1} B_{ij}} \qquad \boldsymbol{U_j} = \sum_{i=1}^{3} \boldsymbol{A_{ij}^T \Sigma_{\epsilon_{ij}}^{-1} A_{ij}}$$

$$\boldsymbol{V_i} = \sum_{j=1}^{2} \boldsymbol{B_{ij}^T \Sigma_{\epsilon_{ij}}^{-1} B_{ij}} \qquad \boldsymbol{W_{ij}} = \boldsymbol{A_{ij}^T \Sigma_{\epsilon_{ij}}^{-1} B_{ij}}$$

The "$*$" represents the augmentation of the diagonal elements by the $\mu\boldsymbol{I}$ term.

The right hand side of Equation 2.41 can be expressed as

$$\boldsymbol{J^T \Sigma_\epsilon^{-1} \epsilon} = (\boldsymbol{\epsilon_{ca}}, \boldsymbol{\epsilon_b})^T = (\boldsymbol{\epsilon_{c1}^T}, \boldsymbol{\epsilon_{a1}^T}, \boldsymbol{\epsilon_{a2}^T}, \boldsymbol{\epsilon_{b1}^T}, \boldsymbol{\epsilon_{b2}^T}, \boldsymbol{\epsilon_{b3}^T})^T. \tag{2.45}$$

$$\boldsymbol{\epsilon_{c_l}} = \sum_{j=1}^{2}\sum_{i=1}^{3} \boldsymbol{C_{lij}^T \Sigma_{\epsilon_{ij}}^{-1} \epsilon_{ij}} \qquad \boldsymbol{\epsilon_{a_j}} = \sum_{i=1}^{3} \boldsymbol{A_{ij}^T \Sigma_{\epsilon_{ij}}^{-1} \epsilon_{ij}} \qquad \boldsymbol{\epsilon_{b_i}} = \sum_{j=1}^{2} \boldsymbol{B_{ij}^T \Sigma_{x_{ij}}^{-1} \epsilon_{ij}} \tag{2.46}$$

In short, the whole Equation 2.41 can be represented as

$$\begin{pmatrix} \boldsymbol{U^*} & \boldsymbol{W} \\ \boldsymbol{W^T} & \boldsymbol{V^*} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_{ca}} \\ \boldsymbol{\delta_b} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon_{ca}} \\ \boldsymbol{\epsilon_b} \end{pmatrix} \tag{2.47}$$

with the block matrices $\boldsymbol{U^*}$, $\boldsymbol{W}$, $\boldsymbol{W^T}$ and $\boldsymbol{V^*}$ corresponding to the blocks outlined in Equation 2.44. $\boldsymbol{\delta_{ca}}$ and $\boldsymbol{\delta_b}$ are the calibration/camera part and the point part of $\boldsymbol{\delta}$, respectively. By left multiplying Equation 2.47 with the block matrix

$$\begin{pmatrix} \boldsymbol{I} & \boldsymbol{-W V^{*^{-1}}} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix}, \tag{2.48}$$

its first line can be made independent from the second line:

$$\begin{pmatrix} \boldsymbol{U^* - W V^{*^{-1}} W^T} & \boldsymbol{0} \\ \boldsymbol{W^T} & \boldsymbol{V^*} \end{pmatrix} \begin{pmatrix} \boldsymbol{\delta_{ca}} \\ \boldsymbol{\delta_b} \end{pmatrix} = \begin{pmatrix} \boldsymbol{\epsilon_{ca} - W V^{*^{-1}} \epsilon_b} \\ \boldsymbol{\epsilon_b} \end{pmatrix}. \tag{2.49}$$

This way, $\boldsymbol{\delta_{ca}}$ and $\boldsymbol{\delta_b}$ can be computed independently:

$$\boldsymbol{\delta_{ca}} = \quad (\boldsymbol{U^* - W V^{*^{-1}} W^T})^{-1}(\boldsymbol{\epsilon_{ca} - W V^{*^{-1}} \epsilon_b}) \tag{2.50}$$

$$\boldsymbol{\delta_b} = \quad \boldsymbol{V^{*^{-1}}(\epsilon_b - W^T \delta_{ca})} \tag{2.51}$$

The above deductions show that there is no need for inverting the whole matrix $\boldsymbol{J^T \Sigma_\epsilon^{-1} J} + \mu\boldsymbol{I}$ to calculate $\boldsymbol{\delta}$. It is sufficient to invert $\boldsymbol{V^*}$ which is trivial since there are only small nonzero blocks on the diagonal and $(\boldsymbol{U^* - W V^{*^{-1}} W^T})$ which has only the size of $\boldsymbol{U^*}$. In addition, $(\boldsymbol{U^* - W V^{*^{-1}} W^T})$ is symmetric and positive definite, so it can be inverted efficiently using Cholesky Factorization.

Reusing Intermediate Results

Similar to [LA09], we introduce the helper matrix

$$\boldsymbol{Y} = \boldsymbol{W}{\boldsymbol{V}^*}^{-1} = \left(\begin{array}{ccc} \boldsymbol{Y1_{11}} & \boldsymbol{Y1_{12}} & \boldsymbol{Y1_{13}} \\ \hline \boldsymbol{Y2_{11}} & \boldsymbol{Y2_{12}} & \boldsymbol{Y2_{13}} \\ \boldsymbol{Y2_{21}} & \boldsymbol{Y2_{22}} & \boldsymbol{Y2_{23}} \end{array}\right) \tag{2.52}$$

$$\boldsymbol{Y1_{li}} = \boldsymbol{F_{li}}{\boldsymbol{V_i}^*}^{-1} \qquad \boldsymbol{Y2_{ji}} = \boldsymbol{W_{ij}}{\boldsymbol{V_i}^*}^{-1}$$

which will be evaluated just once and can be used to calculate $\boldsymbol{S} = (\boldsymbol{U}^* - \boldsymbol{Y}\boldsymbol{W^T})$ and $\boldsymbol{r} = (\boldsymbol{\epsilon_{ca}} - \boldsymbol{Y}\boldsymbol{\epsilon_b})$:

$$\boldsymbol{S} = \left(\begin{array}{c|cc} \boldsymbol{S1_{11}} & \boldsymbol{S2_{11}} & \boldsymbol{S2_{12}} \\ \hline \boldsymbol{S3_{11}} & \boldsymbol{S4_{11}} & \boldsymbol{S4_{12}} \\ \boldsymbol{S3_{21}} & \boldsymbol{S4_{21}} & \boldsymbol{S4_{22}} \end{array}\right) \tag{2.53}$$

$$\boldsymbol{S1_{l_1l_2}} = \boldsymbol{D^*_{l_1l_2}} - \sum_{i=1}^{3} \boldsymbol{Y1_{l_1i}}\boldsymbol{F^T_{l_2i}} \qquad \boldsymbol{S2_{lj}} = \boldsymbol{E_{lj}} - \sum_{i=1}^{3} \boldsymbol{Y1_{li}}\boldsymbol{W^T_{ij}}$$

$$\boldsymbol{S3_{jl}} = {\boldsymbol{E_{lj}}}^{\boldsymbol{T}} - \sum_{i=1}^{3} \boldsymbol{Y2_{ji}}\boldsymbol{F^T_{li}} \qquad \boldsymbol{S4_{j_1j_2}} = \begin{cases} \boldsymbol{U^*_{j_1}} - \sum_{i=1}^{3} \boldsymbol{Y2_{j_1i}}\boldsymbol{W^T_{ij_2}} & \text{if } j_1 = j_2 \\ -\sum_{i=1}^{3} \boldsymbol{Y2_{j_1i}}\boldsymbol{W^T_{ij_2}} & \text{else} \end{cases}$$

$$\boldsymbol{r} = (\boldsymbol{r1_1^T}, \boldsymbol{r2_1^T}, \boldsymbol{r2_2^T})^{\boldsymbol{T}} \tag{2.54}$$

$$\boldsymbol{r1_l} = \boldsymbol{\epsilon_{cl}} - \sum_{i=1}^{3} \boldsymbol{Y1_{li}}\boldsymbol{\epsilon_{bi}} \qquad \boldsymbol{r2_j} = \boldsymbol{\epsilon_{aj}} - \sum_{i=1}^{3} \boldsymbol{Y2_{ji}}\boldsymbol{\epsilon_{bi}}$$

Then Equation 2.50 can be solved by applying Cholesky Factorization to

$$\boldsymbol{S}\boldsymbol{\delta_{ca}} = \boldsymbol{r}. \tag{2.55}$$

More Efficient Implementation

Most of the matrices and vectors involved in SBA, namely $\boldsymbol{D}$, $\boldsymbol{E}$, $\boldsymbol{F}$, $\boldsymbol{U}$, $\boldsymbol{V}$, $\boldsymbol{W}$, $\boldsymbol{\epsilon}$, $\boldsymbol{Y}$, $\boldsymbol{S}$ and $\boldsymbol{r}$ are sparse by themselves. E.g.: $\boldsymbol{W_{ij}}$ as well as $\boldsymbol{Y_{ij}}$ are only non-zero if point i was observed from camera pose j.

Memory consumption can be reduced by storing all nonzero elements consecutively in an array and by creating an index matrix which stores the offsets into this array for each element. In fact, all submatrices with the same indices can share the same index matrix.

By exploiting the sparsity in the matrices listed above, we can also reduce the computational cost for setting up Equation 2.55: we have to make sure that we never touch (or iterate over) zero elements of a matrix during optimization. This corresponds to han-

dling only observed reprojections (which grow linearly with the number of camera poses) instead of handling all camera pose-point pairs with all global parameters (which grow quadratically). To this end we create arrays of the index combinations we have to concern (which correspond to the nonzero matrix elements) for assigning values or summing them up. Lets illustrate this for the calculation of $\boldsymbol{S2}$ in Equation 2.53: We prepare an array of all "existing" lj index sets to initialize $\boldsymbol{S2_{lj}} = \boldsymbol{E_{lj}}$. In addition, we prepare an array of all existing lij index sets to evaluate $\boldsymbol{S2}$ completely by subtracting $\boldsymbol{Y1_{li}W_{ij}^{T}}$ from $\boldsymbol{S2_{lj}}$ for each element of that array.

By implementing Section 2.3.2, one can solve non-linear problems such as dense camera pose tracking in a stable and fast way. We have implemented a solver corresponding to the content of Section 2.3.3 which is state of the art for Bundle Adjustment problems in the sense that it achieves comparable results to Google's Ceres solver [AMO17], which is a popular choice for such problems nowadays. Due to a higher flexibility and better support, we opted to use Ceres in favor of our own solution for most of our work.

2.4 Sparse Scene Reconstruction

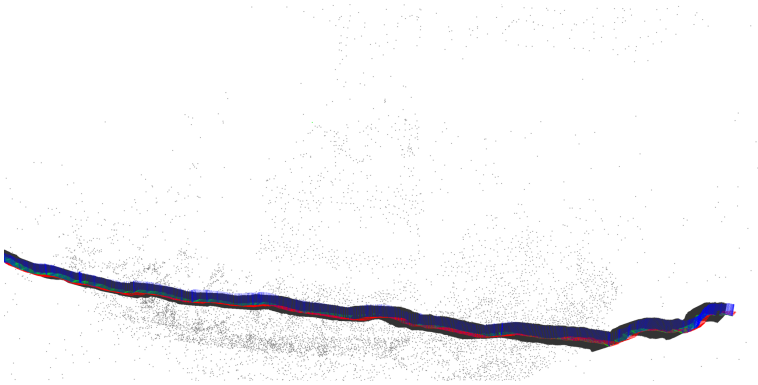


Figure 2.11: *Sparse scene reconstruction example of a statue. Compare to Figure 2.15 for a corresponding dense reconstruction.*

In this section we discuss the data structures and algorithms necessary to perform a scene reconstruction from distinct image features which leads to a sparse representation of the scene's structure. On a high level, sparse reconstructions represent cameras, camera poses and 3D surface points of a scene.

This kind of algorithm is usually chosen when precise camera poses and parameters should be obtained while the scene structure plays only a minor role (e.g. for initializing other algorithms). Advantages of sparse reconstructions are a relatively high processing speed as a result of the limited data that needs to be processed due to the sparse nature of the algorithm on the one hand and a typically high reconstruction precision on the other hand. This results from the fact that the image features which are used as constraints in the reconstruction process are only extracted at distinctive, high gradient locations which can be localized in the images robustly and precisely.

2.4.1 Data Structures

As illustrated in Figure 2.12, the `Scene` is captured by potentially multiple `Cameras` while each `Camera` takes images from multiple `CameraPoses`. Sparse methods rely on distinctive, local image `Features` which are detected on the images of all `CameraPoses` of the scene. Given similar `Features` from different images, feature `Tracks` can be constructed of which each corresponds to a surface `Point` of the scene when feature matching works optimally. Errors in feature matching lead to `Tracks` whose `Features` cannot be triangulated to the same surface `Point`, thus producing `Points` which are not attached to all `Features` of the corresponding `Track`.

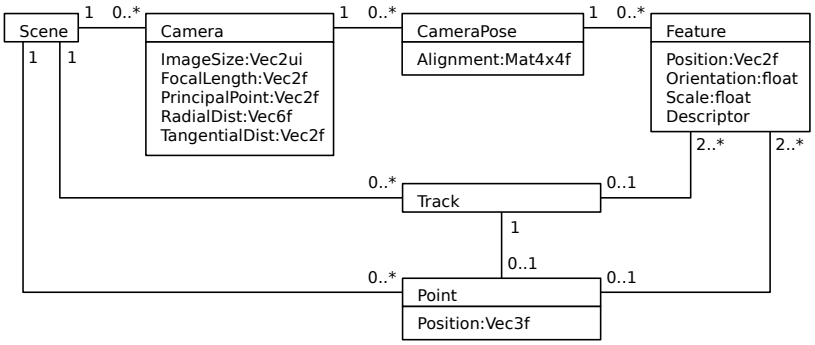


Figure 2.12: *Datastructures involved in sparse scene reconstruction.*

2.4.2 Bootstrapping a Scene

Given all the data that we want to recover in Figure 2.12, we describe the reconstruction process. The key idea behind the process is to start by the geometrical registration of just two camera poses with an overlapping field of view and to add all other camera poses iteratively later. Below we give a step by step algorithm which describes the order in which different objects from Figure 2.12 are initialized.

1. Create one `Camera` object for every camera that was used for capturing and initialize its attributes roughly based on Exif tags or manual measurements.

2. Create one `CameraPose` object for every captured frame. Either
   - use a feature tracker (Section 2.1.2) to initialize the `Features` of every frame and create `Tracks` from tracked features or
   - extract `Features` for matching and create consistent `Tracks` as described in Section 2.1.4.

3. Initialize the first two `CameraPoses` which should see the same scene content by using one of the analytical methods listed in Table 2.2. Use RANSAC (see Section 2.4.3) for robustness against outliers. For small camera baselines, it might be sufficient to initialize the first two `CameraPoses` with a small random displacement.

4. For every `Track` that contains a `Feature` for both initialized `CameraPoses`, triangulate a `Point`. Then, remove all `Features` from `Points` which have a greater reprojection error than the average (and also the `Points` which end up with less than 2 `Features`) for robust outlier handling.

5. Refine the reconstruction with SBA (see Section 2.3.3).

6. Add the next `CameraPose` with an overlapping view to the already reconstructed scene by using the 3-point algorithm (see Table 2.2). Use RANSAC (see Section 2.4.3) for robustness against outliers. For small camera baselines, it might be

Name	Subject	Relative to	Constraints	Output
3-point algo. [FB81]	camera pose	scene points	3 keypoint / scene point pairs	Relative pose
5-point algo. [Nis04]	camera pose	camera pose	5 keypoint / keypoint pairs	Relative pose
6-point algo. [Li06]	camera pose	camera pose	6 keypoint / keypoint pairs	Relative pose joint focal length
7-point algo. [Har92]	camera pose	camera pose	7 keypoint / keypoint pairs	Relative pose two focal lengths
8-point algo. [LH81]	camera pose	camera pose	8 keypoint / keypoint pairs	Relative pose

Table 2.2: *List of n-point algorithms for relative camera pose estimation with important properties. Note that the 3-, 5-, 6- and 7-point algorithms use the minimum number of constraints for recovering their output while the 8-point algorithm has an easier implementation and higher robustness on noisy but otherwise correct input data. However, on input data with outliers, it has a higher probability to fail because of its additional input constraints.*

sufficient to initialize the `CameraPose` similarly to the closest already initialized `CameraPose`.

7. For every `Track` that contains a `Feature` for at least three `CameraPoses`, triangulate a `Point`. Then, remove all `Features` from `Points` which have a greater reprojection error than the average (and also the `Points` which end up with less than 3 `Features`) for robust outlier handling.

8. Refine the reconstruction with SBA (see Section 2.3.3).

9. If there are still uninitialized `CameraPoses`, continue with 6.

Given enough memory and time, this algorithm is in principle able to reconstruct arbitrarily large, static 3D scenes consistently as long as there is a sufficient coverage of the whole scene with images. Research focuses mainly on computationally critical tasks like finding the next camera pose to add to a scene (Steps 3 and 6) or on exploiting the redundancy in the data to avoid having to solve an increasingly large SBA problem per added camera pose in Step 8 (refer to our solution in Chapter 4).

2.4.3 Handling Uncertainty

Since feature tracking and matching are prone to errors, it is very important to handle and keep track of uncertainty during reconstruction. This can be done by avoiding outliers in the tracking or matching data that is used as input (see Chapter 3) or as described in this section.

RANSAC

Random Sample Consensus [FB81] is a model fitting approach that estimates, in contrast to least squares, many models in parallel, each based on the minimum number of data points, to estimate the model parameters. Every model is rated by the number of inliers and the best model is selected in the end (see Figure 2.13). It is typically used in combination with the n-point algorithms (see Table 2.2).

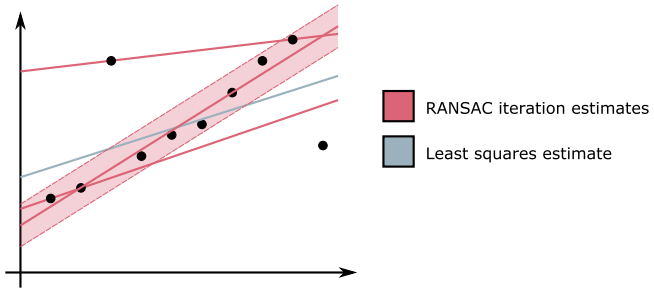


Figure 2.13: *Least squares vs. RANSAC model fitting*

Neumann and Bayerl [NB10] have derived an estimation of the number of iterations which have to be made - assuming that the dataset is very large compared with the amount of samples which are chosen so that the probability of choosing an outlier does not become significantly higher with the number of selected samples.

Let a_o be the maximum expected outlier percentage in the dataset. Further, let $P(C(c))$ be the probability that the c samples necessary to create a model which are randomly chosen from the dataset are all correct. Then, the probability P for choosing c correct samples can be expressed as

$$P(C(c)) \geq (1 - a_o)^c \tag{2.56}$$

$$\Rightarrow P(\overline{C(c)}) \leq 1 - (1 - a_o)^c. \tag{2.57}$$

$P(\overline{C(c)})$ is the probability that at least one sample is an outlier. So the probability for choosing t times a set of c samples of which each contains at least one outlier is:

$$P((\overline{C_0(c)}, \overline{C_1(c)}, ..., \overline{C_n(c)})) \leq (1 - (1 - a_o)^c)^t \tag{2.58}$$

We want this probability to be lower than p_{max}:

$$(1 - (1 - a_o)^c)^t \leq p_{max} \tag{2.59}$$

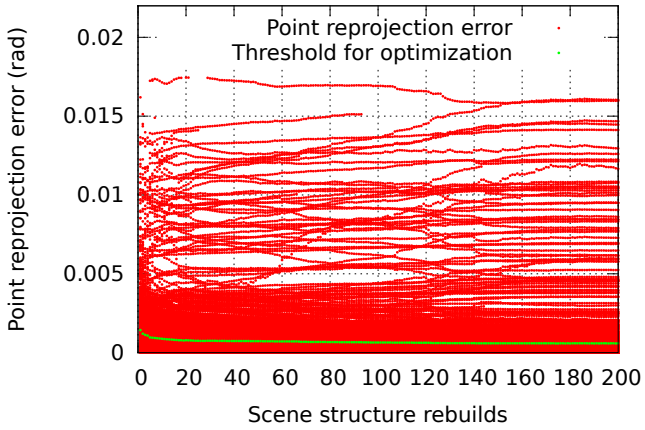


Figure 2.14: *Behavior of reprojection errors over many scene structure rebuilds with outlier elimination. Points with a reprojection error above the average (green dots) are behaving inconsistently with the residual's trend because they are mainly outliers and won't be used for optimization therefore.*

Now this equation can be solved for t to get the number of iterations which have to be made to get a good model with a probability of $(1 - p_{max})$:

$$(1 - (1 - a_o)^c)^t \leq p_{max} \tag{2.60}$$

$$\Leftrightarrow \quad t \log(1 - (1 - a_o)^c) \leq \log(p_{max}) \tag{2.61}$$

$$\Leftrightarrow \quad p \geq \frac{\log(p_{max})}{\log(1 - (1 - a_o)^c)} \tag{2.62}$$

When RANSAC is used with an n-point algorithm, it calculates a camera pose estimate using n random point pairs in each RANSAC iteration. All points which have a reprojection error below a threshold for the estimated camera pose of a certain RANSAC iteration are counted as inliers for this iteration.

Point Outlier Removal

In the presence of only few outliers, SBA will optimize towards the correct solution, reducing the residual of the majority of inlier data points but the optimization will not converge completely due to the inconsistent outliers (see Figure 2.14). Therefore, we always remove image features from points when their reprojection error is larger than the average reprojection error in the scene in Steps 4 and 7, Section 2.4.2.

Loss Functions

Even after RANSAC for initialization and outlier removal whenever the scene is updated, there might be outlier reprojections that reach a noticeable and thus disturbingly high re-

projection error during the SBA. This may for example happen when a tracking keypoint is erroneously placed on an edge which leads to slowly increasing tracking errors from frame to frame which become only visible when the reconstruction of the scene is already very precise. To ensure convergence anyway, the residual (Equation 2.42) is extended by a loss function, e.g.

$$\rho(s)_h = \begin{cases} s & s \leq 1 \\ 2\sqrt{s} - 1 & s > 1 \end{cases} \quad \text{or} \tag{2.63}$$

$$\rho(s)_c = \log(1 + s), \tag{2.64}$$

that reduces large residuals to have a sub-squared influence:

$$R = \sum_{i=1}^{m} \rho(\|(u\ v) - (\boldsymbol{\pi}(\boldsymbol{p_{c_l}}) \circ \boldsymbol{\varphi}(\boldsymbol{p_{p_j}}))(\boldsymbol{x_i})\|^2) \tag{2.65}$$

The most usual choices are the Huber loss function (Equation 2.63) which lowers the residual from squared to linear for reprojection errors for $s > 1$ and the Cauchy loss function (Equation 2.64) which behaves logarithmically.

2.5 (Semi-)Dense Scene Reconstructions

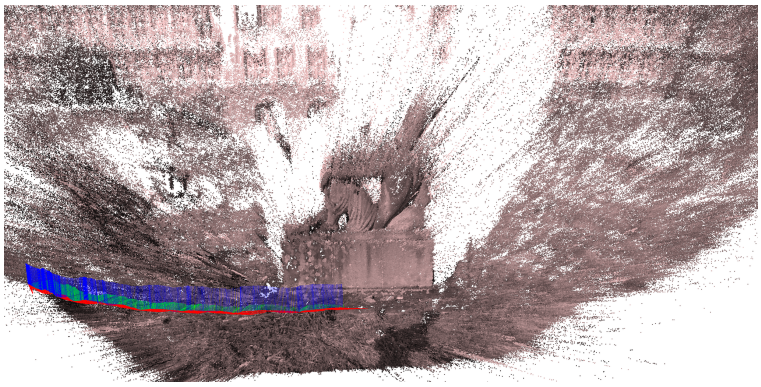


Figure 2.15: *Dense scene reconstruction example of a statue. Compare to Figure 2.11 for a corresponding sparse reconstruction.*

In this section we discuss the data structures and algorithms necessary to perform a semi-dense scene reconstruction directly from the frames of a video. This reconstruction method is based on a dense scene structure representation in the form of depth maps for certain keyframes. Instead of optimizing for geometric errors such as the deviations between point reprojections and the corresponding feature positions in sparse methods (see Section 2.4), we utilize the photometric error for both tracking new camera poses and obtaining depth information (see Figure 2.17). Camera pose tracking is achieved by rendering the previously recovered scene structure into a novel view while optimizing its camera pose parameters to obtain an image that is similar to the frame to be tracked. The per pixel depth is recovered by searching for a depth value that projects the surrounding image patch to a similarly looking patch in another image.

(Semi-)dense methods have advantages over sparse methods on low resolution input data or images with few 2D structures which lack sufficient distinct point features for reconstruction: dense methods are able to utilize all the intensities of an image for tracking and scene structure estimation.

Below, we present a representative algorithm in detail.

2.5.1 LSD-SLAM

We describe LSD-SLAM by Engel et al. [ESC14] which is a direct, semi-dense, monocular Self Localization And Mapping approach. It consists of the following steps (see Fig. 2.16):

1. Tracking has the goal of finding the parameters of the relative transformation $\boldsymbol{p}_{\boldsymbol{p}_i \to \boldsymbol{p}_j}$ of the camera from the last keyframe $\mathcal{K}_i = \{I_i, D_i, V_i\}$ to the most current image I_j as

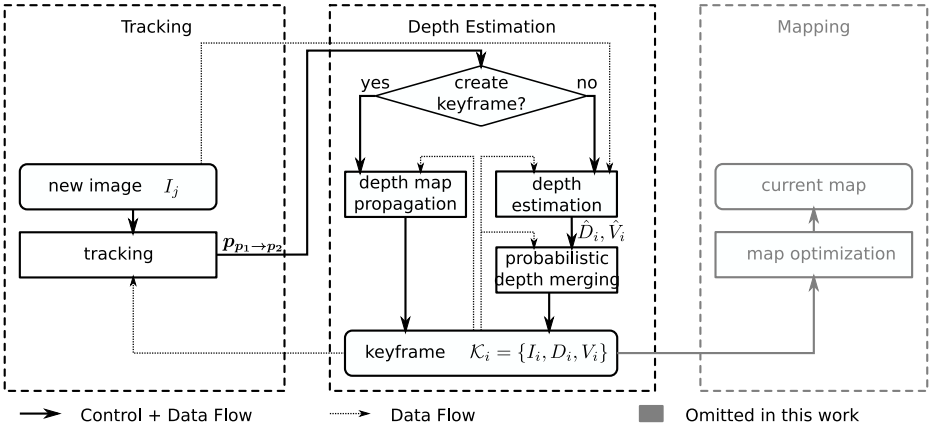


Figure 2.16: *Overview over the LSD-SLAM algorithm.*

illustrated in Figure 2.17 on the left. For keyframes, a semi-dense, inverse depth map D_i as well as the variances V_i of the inverse depth values are assumed to be available beside the pixel intensities I_i.

A new image is aligned to the previous keyframe by Levenberg-Marquard minimization (see Section 2.3.2) of the photometric error

$$E_c(\boldsymbol{p}_{\boldsymbol{p_i} \to \boldsymbol{p_j}}) = \sum_{(u\ v)} [I_i(u, v) - I_j(\boldsymbol{\omega}(u, v, D_i(u, v); \boldsymbol{p}_{\boldsymbol{p_i} \to \boldsymbol{p_j}}))]^2 \tag{2.66}$$

between every pixel $(u\ v)$ in the keyframe's image I_i and the intensity at the corresponding location in the new image I_j, obtained by warping $(u\ v)$ based on its depth $D_i(u, v)$ and the camera pose transformation parameters $\boldsymbol{p}_{\boldsymbol{p_1} \to \boldsymbol{p_2}}$. The warping function (see Section 2.2.2) is given by

$$\begin{aligned} \boldsymbol{\omega}(u, v, d; \boldsymbol{p}_{\boldsymbol{p_1} \to \boldsymbol{p_2}}) &= [\boldsymbol{\pi}(\boldsymbol{p}_{\boldsymbol{c_2}}) \circ \boldsymbol{\varphi}_{\boldsymbol{1} \to \boldsymbol{2}}(\boldsymbol{p}_{\boldsymbol{p_1} \to \boldsymbol{p_2}}) \circ \boldsymbol{\pi}^{-\mathbf{1}}(\boldsymbol{p}_{\boldsymbol{c_1}})](u\ v\ d) \\ \boldsymbol{\varphi}_{\boldsymbol{1} \to \boldsymbol{2}}(\boldsymbol{p}_{\boldsymbol{p_1} \to \boldsymbol{p_2}}) &\cong \boldsymbol{\varphi}(\boldsymbol{p}_{\boldsymbol{p_2}}) \circ \boldsymbol{\varphi}^{-\mathbf{1}}(\boldsymbol{p}_{\boldsymbol{p_1}}) \end{aligned} \tag{2.67}$$

(compare to Equation 2.27) and contains only one view space to view space transformation $\boldsymbol{\varphi}_{\boldsymbol{1} \to \boldsymbol{2}}$ since we are optimizing for camera pose relative to the last frame instead of the scene's world space. Note that in LSD-SLAM, the intrinsic camera parameters $\boldsymbol{p}_{\boldsymbol{c_1}}$ and $\boldsymbol{p}_{\boldsymbol{c_2}}$ are not subject to optimization.

Tracking is performed on an image pyramid in a coarse-to-fine manner: A coarse representation helps to converge even with high disparities, fine details provide exact tracking in the end.

After tracking, a decision about creating a new keyframe is made based upon the distance that the camera moved since the last keyframe and the number of pixels that could be used for tracking. Depending on the decision, steps 2 and 3 or step 4 are executed.

2. Depth Estimation produces a new inverse depth map $\hat{D}_i$ and a corresponding variance map $\hat{V}_i$, using a stereo method on the image intensities I_i and I_j by using the previously obtained camera pose transformation parameters $\boldsymbol{p}_{\boldsymbol{p_1 \to p_2}}$. Depth estimation is illustrated in Figure 2.17 on the right.

Depth $\hat{D}_i$ is obtained for all pixels that exhibit a sufficiently large gradient $\boldsymbol{\nabla} I_i(p)$ along the Epipolar Line (EPL) by brute force evaluating patch similarity (see Section 2.1.1) based on a 1D, 5 pixel EPL aligned patch (see Figure 2.1) and a squared distance metric. The depth is further refined by second order Taylor approximation. The search range for each pixel $(u\ v)$ on the EPL corresponds to the depth range $(D_i(p) \pm 3V_i(p))$.

The variance values $\hat{V}_i(p)$ are estimated based on the photometric disparity error ($\approx \frac{\textit{camera noise}}{|\boldsymbol{\nabla} I_i(p)|}$) and the geometric disparity error ($\approx 1/\langle \frac{\boldsymbol{\nabla} I_i(p)}{|\boldsymbol{\nabla} I_i(p)|}, \frac{\textbf{EPL}}{|\textbf{EPL}|} \rangle$, i.e. it is large if the image gradient is orthogonal to the EPL direction), both determined in the image domain and transformed to the inverse depth domain.

3. Probabilistic Depth Merging is used to update the previous keyframe depths D_i and variances V_i with the new estimations $\hat{D}_i$ and $\hat{V}_i$. This is done by multiplying the distributions according to the update step in a Kalman filter [K$^+$60]: Given a prior distribution $\mathcal{N}_{\textit{prior}} = \mathcal{N}(D_i(\begin{smallmatrix} u \\ v \end{smallmatrix}), V_i(\begin{smallmatrix} u \\ v \end{smallmatrix}))$ and a new observation $\mathcal{N}_{\textit{new}} = \mathcal{N}(\hat{D}_i(\begin{smallmatrix} u \\ v \end{smallmatrix}), \hat{V}_i(\begin{smallmatrix} u \\ v \end{smallmatrix}))$, the posterior is given by

$$\mathcal{N}_{\textit{post}} = \mathcal{N}_{\textit{prior}} \cdot \mathcal{N}_{\textit{new}} = \mathcal{N}\left(\frac{V_i(\begin{smallmatrix} u \\ v \end{smallmatrix})\hat{D}_i(\begin{smallmatrix} u \\ v \end{smallmatrix}) + \hat{V}_i(\begin{smallmatrix} u \\ v \end{smallmatrix})D_i(\begin{smallmatrix} u \\ v \end{smallmatrix})}{V_i(\begin{smallmatrix} u \\ v \end{smallmatrix}) + \hat{V}_i(\begin{smallmatrix} u \\ v \end{smallmatrix})}, \frac{V_i(\begin{smallmatrix} u \\ v \end{smallmatrix})\hat{V}_i(\begin{smallmatrix} u \\ v \end{smallmatrix})}{V_i(\begin{smallmatrix} u \\ v \end{smallmatrix}) + \hat{V}_i(\begin{smallmatrix} u \\ v \end{smallmatrix})} \right). \quad (2.68)$$

After depth and variance have been updated, smoothing and hole filling are applied to D_i and V_i.

4. Depth Map Propagation establishes a new keyframe by propagating depth and variance information from the previous keyframe to the currently processed frame. A forward mapping from the keyframe to the new frame is established by using $\boldsymbol{\omega}$ (see Equation 2.27) and depths and variances are assigned to the closest pixel in the new frame's image.

After the depth propagation, the new depth and variance maps are subject to removal of occluded pixels, hole filling and smoothing.

5. Map Optimization Every keyframe is added to the mapping component of LSD-SLAM which integrates it into a complete, globally consistent map of the reconstructed scene, caring about loop closing and tracking error distribution over the whole scene. This is just mentioned for completeness of the LSD-SLAM algorithm: We handle only sparse loop closing in the context of this thesis in Chapter 4.

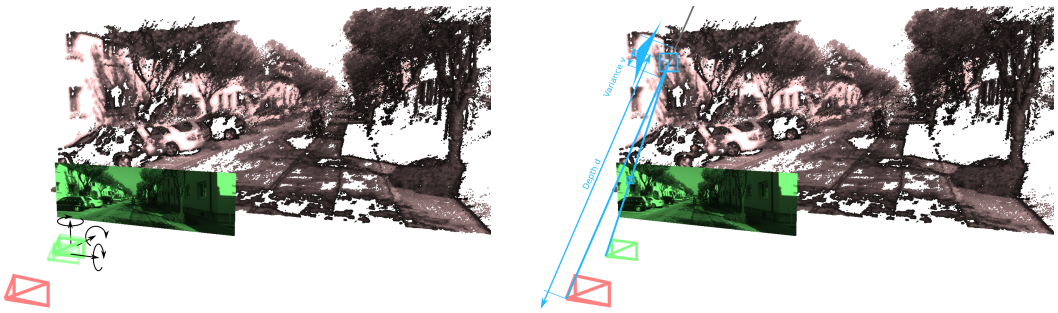


Figure 2.17: *Tracking and mapping in LSD-SLAM. Tracking (left) aims at aligning the new frame(green) so that it matches the reprojections from the last keyframe's (red) data. Mapping (right) finds a depth for every keyframe (red) pixel so that it is reprojected to a similar patch on the new frame (green).*

The LSD-SLAM algorithm is initialized with a random inverse depth map and an identity camera pose transformation for tracking and converges over multiple tracking / depth map update iterations. However, it is crucial to avoid propagating errors during reconstruction.

2.5.2 Handling uncertainty

LSD-SLAM applies several techniques to keep track of uncertainty and handling it appropriately:

Probabilistic depth: All per pixel depth values are represented as normal distribution and updated probabilistically similar to a Kalman filter. The variances of the inverse depths influence the importance for tracking (Σ_ϵ^{-1} in Equation 2.41) and the search range on the epipolar line during stereo depth estimation.

Pixel validity tracking: The validity of pixels is monitored during depth estimation: A validity score is increased when a depth is found successfully and decreased if it is not, e.g. when no good depth was found or multiple depths produced equally good patch similarities. This information is used when an a priori depth map is merged with a new observation.

Blacklisting: While validity can increase again when better depths are found over time, keyframe pixels can also become blacklisted until the next depth map propagation if depth from stereo fails drastically.

Cleanup: When the keyframe data is propagated to another frame, occluded pixels are removed, small holes are filled and depth maps are smoothed to reduce errors.

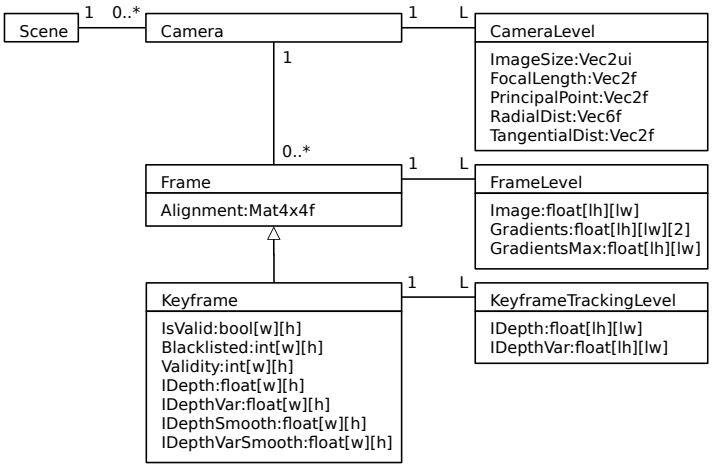


Figure 2.18: *Data structures involved in dense scene reconstruction. All data that is used for tracking has to be available in L scale levels. Only keyframes contain depth data and all necessary maps for its creation.*

The requirements described above for the processing and handling of uncertainty lead to data structures shown in Figure 2.18 to perform dense camera tracking and scene reconstruction.

2.6 Related Research Areas

After the introduction of the important background to our contributions, we'd like to clearly distinguish different classes of algorithms that are related to camera pose and scene reconstruction.

Visual Odometry (VO) is the visual counterpart to classical odometry from robotics which aims on estimating a robot's pose based wheel ticks, steering angle etc. The objective in VO is to estimate the relative camera pose from one frame of a video stream to the next as precisely as possible with a low latency, i.e. by considering only temporally nearby information [NNB04].

Self Localization and Mapping (SLAM) extends VO by a mapping component. Beside tracking the camera pose from frame to frame, a consistent and complete representation of the environment should be reconstructed, including the recognition of already observed parts of the scene and loop closing. The location of the camera within this environment should be provided with low latency [ESC14].

Structure from Motion (SfM) refers to an offline, usually sparse scene reconstruction including camera poses and intrinsic camera parameters often. It aims at an efficient but precise reconstruction. Most algorithms are not restricted to video streams but can also work on image collections.

Dense reconstructions feature scene surface samples in the order of magnitude of the pixels in the images used to reconstruct the scene. This can either be achieved by using a dense reconstruction method like [ESC14, UZU+17] directly or by using a multi-view stereo method on top of a SfM reconstruction [FP10, BFL12, WRL14, WRL16, WRL17]. In more controlled environments, surfaces can also be captured with active light sources, e.g. by projecting patterns [SFPL10, GRL17] or with Time of Flight cameras [IKH+11].

Appearance reconstruction refers to capturing models of the scene that represent the captured radiances correctly. Model parameters can include color and textures, surface reflectance properties and the properties of light sources as well as volumetric effects. Appearance reconstruction is often seen as the last step of scene reconstruction after obtaining sparse and dense geometry. However, depending on the scene, obtaining geometry might be arbitrarily hard in the absence of appearance information, so iterative or joint approaches for reconstruction seem to be more promising.

2.6.1 Embedding of this Thesis

We would like to classify the work in this thesis according to the described areas of research.

Our method on improving image feature matches based on feature orientations in Chapter 3 [RLL14] is potentially applicable to any sparse, feature based reconstruction method, may it be VO, SLAM or SfM. The largest benefits can however be expected in SfM algorithms which exhibit potentially significant camera and image feature rotations among the input images, while SLAM and VO methods are inherently video based and exhibit only small differences on consecutive frames.

The pipeline for scalable SfM on highly redundant, high resolution video data presented in Chapter 4 [RLW+15] is obviously a SfM method which is, on the one hand, closely related to SLAM due to the processing of video data at high speed and the reconstruction of camera trajectories as well as a globally consistent scene structure representation. On the other hand it differs from other SLAM methods as it is much more precise at the cost of offline processing: the result of the reconstruction is only available at the end of the process.

Our work on dense reconstruction methods for flexible camera rigs in Chapter 5 [RWL16] is primarily a VO method as it aims on reconstructing a locally consistent scene representation. Here, we focus on solving the critical problem of handling slightly deformable camera rigs which can break the photoconsistency based tracking and the integration of data from multiple cameras in the reconstruction for improved precision and stability. Although we propose a VO method, the presented improvements can be integrated in other SLAM in a straight-forward way.

Chapter 6 handles the training of convolutional neural networks (CNNs) for VO with special attention to analyzing the effects of different training data on the results. This can be seen as fundamental research on machine learning based VO and is not yet able to produce results on par with other VO methods.

Chapter 3

Matching Sparse Features on Omnidirectional Images

Figure 3.1: *Omnidirectional images with level (top) and rolled (bottom) camera. Note how the image areas in the back facing part rotate inversely to the front, so there is no common 2D feature rotation.*

In this chapter we present a method for refining the matches obtained between image features with descriptors like SIFT, SURF or ORB (see Section 2.1.5). In addition to matching as described in Section 2.1.4, many feature matching frameworks utilize the relative orientation of the features to filter out inconsistent matches. This is important since the computational cost for subsequent tasks like camera pose estimation or object detection increases dramatically with the number of outliers. For omnidirectional images, simple 2D orientation descriptions are unsuitable because of image distortions and non-monotonic mapping from camera rotations to image rotations as shown in Figures 3.3 and 3.1. Therefore we introduce 3D orientation descriptors which, unlike 2D descriptors, are suitable for match refinement on omnidirectional images and improve matching results on images from cameras and camera rigs with a wide field of view. We propose different match refinement strategies based on which we show for 2D and 3D orientations the fundamental advantages of our approach.

Figure 3.2: *Extremely wide FoV image with pinhole camera projection. Note that the periphery appears stretched. The sampling of the image cannot be optimal everywhere in the image with this projection.*

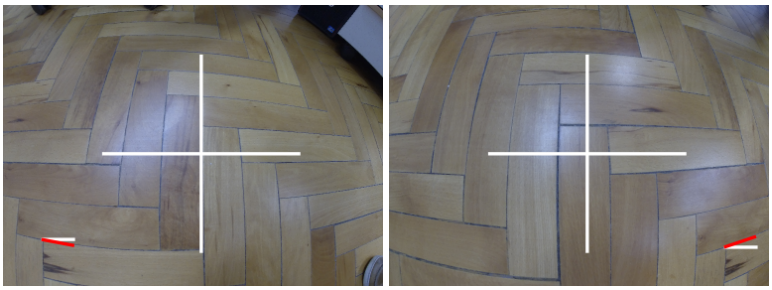


Figure 3.3: *Inconsistent feature rotations on wide field of view images with strong distortions. Note how the orientation of the feature at the bottom left/right of the left/right image changes while the central image areas keep their orientation.*

3.1 Introduction

The extraction of local image features is a key task in many algorithms such as image classification, object recognition or sparse scene geometry reconstruction as described in Section 2.4. The usual way of achieving rotation invariance of the feature descriptor is to assign an orientation to each keypoint before extracting the feature descriptor relative to this orientation [Low04, BTVG06, MS05].

The orientations are not only useful for the rotational invariance of the feature descriptors but can also be exploited to improve matching that is solely based on the descriptors (see Section 2.1.4) by ensuring rotational consistency: For "normal" 2D images with a decent field of view (FoV) (usually $< 40°$), it is a legitimate assumption that all features in an image pair have the same rotation behavior: the inverse of the camera roll.

For wider FoVs, a mapping to a flat image plane as described in Section 2.2.2 is not beneficial (Figure 3.2) or even impossible (Figure 3.1). Therefore, non-monotonic mappings are applied which results in mapping the scene to the image quite differently in different regions (see Figure 3.3). Omnidirectional camera rigs can provide FoVs of virtually 360°, which are usually handled as a set of individual images or mapped to a cylindrical, spherical or other representation for feature extraction. In all those types of images, the rotation of matched features does not uniformly map to the camera's orientation (see Figure 3.1): Tilting and rolling of the rig results in features rotating in opposite directions.

To overcome the issues, we propose a 3D orientation descriptor that can be inferred from its 2D counterpart and the intrinsic camera calibration (per pixel view direction in view space): In principle, our orientation descriptor defines a triplet of orthonormal vectors for each feature whose base axes are given by (1) the 2D feature orientation transformed to view space, (2) the view direction towards the feature in view space and (3) the cross product of (1) and (2) (see Figure 3.4).

Since this 3D orientation descriptor is constructed in view space, it is independent of the camera type and parameters and therefore allows for significantly more robust matching of features between omnidirectional and wide-angle camera images. The number of spurious matches is significantly reduced using our 3D orientation while positive matches are essentially detected unhindered. Subsequent processing tasks such as sparse geometry reconstruction (see Section 2.4) become significantly more robust and more efficient after this simple outlier removal based on 3D orientation.

In the remaining part of this chapter, we introduce related work to the handling of wide FoVs as well as omnidirectional cameras with local image features. We present our 3D orientation descriptor in detail, describe our match refinement strategies and evaluate and compare our work to the common use of 2D feature orientations.

3.2 Related Work

Finding point correspondences in image pairs is usually done in four steps: First, distinctive keypoints are found with position, orientation and scale (Section 2.1.3). Their local image content appearance is stored in a feature descriptor (Section 2.1.1) which is matched based on some distance norm (Section 2.1.4). While the best match is found for each feature *individually*, the set of all found matches can be refined *in common* based on geometrical consistency of the keypoints as a final step: the SIFT framework [Low04] introduced analyzing keypoint orientations of all matches for consistency. By dropping inconsistent matches, the number of outliers is reduced. This match refinement stage is what we analyze and improve in this chapter.

Match refinement can be applied to any feature matching framework that extracts feature orientations such as the already presented SIFT [Low04], SURF [BTVG06], GLOH [MS05], Oriented FAST [RD06, RRKB11], BRISK [LCS11] and FREAK [AOV12] (Summary in Section 2.1.5). While SIFT, SURF and GLOH assign orientations based on the local gradient, newer descriptors like Oriented FAST uses intensity centroids [Ros99]: it calculates moments

$$m_{pq} = \sum_{u,v} u^p v^q I(u,v) \qquad p,q \in \{0,1\} \tag{3.1}$$

for all pixels of a patch and obtains the orientation based on their gradients. BRISK [LCS11] and FREAK [AOV12] use the sum of gradients obtained from special

sample pairs of patches around a keypoint. Our method is not only applicable to point features: Scaramuzza et al. [SCMS08] extract vertical line features which are oriented by definition. Lu and Wu [LW08] do quasi-dense matching of omnidirectional and perspective images using an affine model for local transformations which can be reduced to a rotation.

There are several SIFT extensions which improve matching wide FoV and omnidirectional camera image features by better feature extraction instead of refining the matches in the end: Arican and Frossard [AF10] focus on calculating the scale space pyramid used for keypoint detection correctly for parabolic mirrors by solving the heat equation using Riemannian geometry. Hansen et al. [HCB10] and Cruz-Mota et al. [CMBP+12] make features invariant against distortions by projecting the images to a sphere and then back to planes which are tangential to the sphere at the feature's location. Lourenço et al. [LBV12] implement scale space construction and feature extraction without image resampling by using adaptive filtering that compensates image distortion. Our approach of match refinement is orthogonal to all the publications focusing on better feature extraction and can be applied on top of those for further improvement.

3.2.1 Applications

Our match refinement provides advantages on all cameras with a large field of view or distorted images, e.g., currently popular action cams like the GoPro Hero Series.

The greatest improvements can however be observed on omnidirectional camera systems, e.g. custom camera rigs, professional devices such as the various Point Grey's Ladybugs, commercial omnidirectional camera systems like the 360° camera in the new Mercedes E-Class or the Panono Panoramic Ball Camera. Beside one shot solutions, many smartphones are able to capture panoramas, e.g. by using Photo Sphere (Google) or Photosynth (Microsoft).

3.3 3D Orientation Descriptor

Previously, orientations of local 2D image features were simply represented by an orientation angle o.

We represent orientations with our 3D orientation descriptor $\boldsymbol{O}$ which can be calculated for each feature and is made up of three orthonormal vectors which define an Euclidean 3D space. For each match of two features with their 3D orientation descriptors $\boldsymbol{O_1}, \boldsymbol{O_2}$, one can find the rotation of the match $\boldsymbol{R^{3D}} = \boldsymbol{O_2} \cdot \boldsymbol{O_1^{-1}}$ which transforms from one feature space to the other. The orientation descriptors are designed in a way that the match rotation $\boldsymbol{R^{3D}}$ is roughly the inverse of the camera rotation and therefore similar for all correct matches of an omnidirectional image pair.

The 3D orientation descriptor is calculated based on the 2D orientation o of a feature in the image plane, its position $(u\ v) \in \mathbb{R}^2$ in the image and the intrinsic calibration of

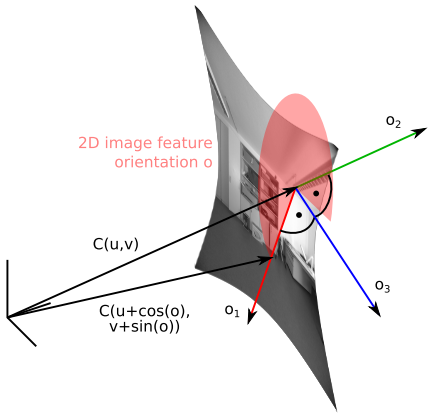


Figure 3.4: *3D orientation descriptor. The red, green and blue vectors represent the axes of the orthonormal descriptor frame defined by the orientation and position of each feature.*

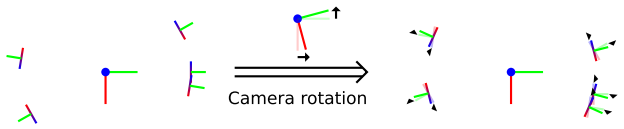


Figure 3.5: *Descriptor behavior under camera rotation. If the camera is rotated, all 3D feature orientations perform an inverse rotation.*

the camera (or camera rig)

$$\boldsymbol{C}(u,v) = \frac{\boldsymbol{\pi^{-1}}((u\ v\ 1); \boldsymbol{p_c})}{\|\boldsymbol{\pi^{-1}}((u\ v\ 1); \boldsymbol{p_c})\|} \quad \in \quad \{\mathbb{R}^2 \rightarrow \mathbb{R}^3\} \tag{3.2}$$

that maps image coordinates to directions in camera space (compare to Section 2.2.3). Note that calculating the 3D descriptor works the same way for single images, spherical or cylindrical mappings or sets of images that represent an omnidirectional view.

The 3D orientation descriptor is defined by $\boldsymbol{O} = (\vec{\boldsymbol{o_1}}\ \vec{\boldsymbol{o_2}}\ \vec{\boldsymbol{o_3}})$ (see also Figure 3.4):

$$\vec{\boldsymbol{o_1}} = \frac{\boldsymbol{C}(u+\cos(o), v+\sin(o)) - \boldsymbol{C}(u,v)}{\|\boldsymbol{C}(u+\cos(o), v+\sin(o)) - \boldsymbol{C}(u,v)\|} \tag{3.3}$$

$$\vec{\boldsymbol{o_2}} = \boldsymbol{C}(u,v) \tag{3.4}$$

$$\vec{\boldsymbol{o_3}} = \vec{\boldsymbol{o_1}} \times \vec{\boldsymbol{o_2}} \tag{3.5}$$

3.3.1 Properties

By design, the rotation $\boldsymbol{R^{3D}}$ obtained from two matched features is the inverse of the camera (or camera rig) rotation (see Figure 3.5). In contrast to 2D orientations, all our 3D orientation descriptors rotate the same way under arbitrary camera transformations.

This allows for analyzing a set of feature matches for consistent rotations and discarding inconsistent matches.

Another benefit is that we incorporate the features' viewing directions $\boldsymbol{d_c}$ into the orientation descriptors. This enables us to distinguish between features with similar appearance and 2D orientation but different viewing direction.

3.4 Feature Matching

The proposed 3D orientation helps in refining matches by analyzing the rotations between their features for consistency and removing inconsistent matches. We do not propose any new method on how to find matches. Instead our approach can work with any preselection of matches.

The general procedure for analyzing matches given 3D feature orientations is the same as for 2D feature orientations. The steps are as follows:

1. Calculate a rotation r_i from the first to the second feature's orientation of each match.
2. Find a representative rotation $\hat{r}$ for the rotations r_i of all matches.
3. Remove all matches whose feature's rotations r_i differ from the representative $\hat{r}$ by more than a certain threshold.

The next two sections for 2D and 3D descriptor match refinement first describe how to obtain a rotation from an orientation (step 1) and how to calculate differences in rotation (step 3). Then we present different strategies for finding a representative rotation $\hat{r}$ (step 2) in both subsections.

3.4.1 2D Descriptor Match Refinement

For 2D feature orientations o_1, o_2 which are scalar rotation angles, we obtain the scalar rotation for a match $r^{2D} = o_2 - o_1$ by simple subtraction. The results are mapped to the range $(-\pi, \pi)$ to have the identity rotation in the middle of the interval.

We implemented and tested the following strategies for obtaining $\hat{r}^{2D}$ from a set of rotations r_i^{2D}:

Histograms This idea is borrowed from SIFT [Low04] and sorts all r_i^{2D} into histogram bins. To avoid problems at the bin boundaries, the bins overlap by 50%, so each r_i^{2D} is assigned to two bins. The center of the bin with the most rotations is used as $\hat{r}^{2D}$.

Mean Using the average of all rotations is expected to lead to good results for evenly distributed outlier rotations but is prone to errors for multimodal distributions which can arise from rotated omnidirectional cameras (see Figure 3.10).

Cosine fit We select the $\hat{r}^{2D}$ that fits a shifted $\cos^2$ function to the relative rotations r_i^{2D} by maximizing

$$E(\hat{r}^{2D}) = \sum_i \left(\frac{1}{2} \cos(r_i^{2D} - \hat{r}^{2D}) + \frac{1}{2} \right)^2 \tag{3.6}$$

with a gradient ascent method using a Levenberg-Marquardt [Lev44] inspired step size. This method should nicely fit to a peak in the rotation distribution - it might however not be the global optimum.

K-Means K-Means clustering is applied to all r_i^{2D}. K-Means cluster centers will converge to the peaks in the rotation distribution, so the center of the largest cluster can be selected as $\hat{r}^{2D}$. Based on tests with 2-10 clusters we decided that three clusters lead to the best k-Means results for all of our test scenes. This corresponds to up to trimodal 2D rotation distributions of our data (see Figure 3.10).

3.4.2 3D Descriptor Match Refinement

For a match's 3D feature orientations $\boldsymbol{O_1}, \boldsymbol{O_2}$ which are orthonormal matrices, we obtain the match's rotation matrix $\boldsymbol{R^{3D}} = \boldsymbol{O_2} \cdot \boldsymbol{O_1^{-1}}$. This $\boldsymbol{R^{3D}}$ can be interpreted as the inverse camera rotation matrix (compare with Section 3.3.1).

As distance associated with a rotation we will use the shortest rotation angle $\alpha(\boldsymbol{R^{3D}}) = \arccos(\mathrm{Tr}(\boldsymbol{R^{3D}}) - 1)$ according to [Gla90].

We implemented the following strategies for obtaining $\hat{\boldsymbol{R}}^{\boldsymbol{3D}}$ from a set of rotations $\boldsymbol{R_i^{3D}}$:

Mean Using the average of all matches' rotations is promising for 3D orientations since the rotation distribution won't get multimodal for omnidirectional setups by design (compare to Section 3.4.1) and outlier rotations can be expected to be more randomly distributed since we incorporate scene information *plus* viewing direction information into the feature orientations. The mean of rotations for all matches is obtained by the orthonormalized sum of all $\boldsymbol{R_i^{3D}}$:

$$\begin{aligned} (\boldsymbol{U}, \boldsymbol{\Sigma}, \boldsymbol{V^T}) &= \mathrm{SVD}\left(\sum_i \boldsymbol{R_i^{3D}} \right) \\ \hat{\boldsymbol{R}}^{\boldsymbol{3D}} &= \boldsymbol{U} \cdot \boldsymbol{V^T} \end{aligned} \tag{3.7}$$

Singular Value Decomposition (SVD) is employed for orthonormalization to keep the rotational parts $\boldsymbol{U}, \boldsymbol{V^T}$ while setting the scaling part $\boldsymbol{\Sigma}$ to identity. This generates the optimal solution in a least squares sense [CJZ93].

Cosine fit $\hat{\boldsymbol{R}}^{\boldsymbol{3D}}$ is selected like in the 2D case (see Section 3.4.1) by maximizing

$$E(\hat{\boldsymbol{R}}^{\boldsymbol{3D}}) = \sum_i \left(\frac{1}{2} \cos(\alpha((\hat{\boldsymbol{R}}^{\boldsymbol{3D}})^{-1} \cdot \boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{3D}})) + \frac{1}{2} \right)^2 \tag{3.8}$$

where $\alpha((\hat{\boldsymbol{R}}^{\boldsymbol{3D}})^{-1} \cdot \boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{3D}})$ is the angular difference between the representative rotation $\hat{\boldsymbol{R}}^{\boldsymbol{3D}}$ and the rotation $\boldsymbol{R}_{\boldsymbol{i}}^{\boldsymbol{3D}}$ of a match. As for the 2D refinement, this will fit to a peak in the rotation distribution which will typically be the global optimum since we expect an unimodal distribution.

K-Means We proceed as in the 2D case (see Section 3.4.1) and use $\alpha(\boldsymbol{R}_{\boldsymbol{j}}^{\boldsymbol{3D}} \cdot (\boldsymbol{R}_{\boldsymbol{k}}^{\boldsymbol{3D}})^{-1})$ as distance metric for a pair of rotations. As before, three clusters already lead to the best k-Means results for all our test scenes. Actually we don't expect multimodal distributions, so this strategy was mainly implemented to have a counterpart to the 2D version of k-Means for evaluation.

3.4.3 Match Selection

All matches whose deviation from $\hat{r}^{2D}$ or $\hat{\boldsymbol{R}}^{\boldsymbol{3D}}$ is smaller than a certain error threshold e are *selected*; the others are *discarded*. For a fair comparison, we selected a common threshold e. To select a good e, we sampled the thresholds in steps of five degrees on our test scenes using the cosine fit strategies from Section 3.4.1 and 3.4.2 (see Figures 3.6 and 3.7). It turns out that setting $e = 35°$ is high enough to select most good matches while a higher threshold gives overproportional rise to the outliers.

Note that the original SIFT matching [Low04] uses 30° bins for 2D features. For the 2D case, our higher threshold is justified by the fact that our cameras have more radial distortion. The 3D match refinement strategies are invariant to image distortion because we use the intrinsic camera calibration, but we incorporate the viewing direction into the descriptors which will cause more variance (Section 3.3).

3.5 Evaluation

Our proposed 3D orientation descriptor and all match refinement strategies are evaluated on datasets captured with a Point Grey Ladybug 3 omnidirectional camera system and a GoPro Hero 3 wide angle camera.

The Ladybug 3 system has 5 equatorially aligned and one up-facing cameras, each shipped with calibration data. The six individual images have a resolution of 1616x1232 pixels each. Six scenes were captured with the omnidirectional camera by taking two shots of each (see Figure 3.8): for *stairway*, *cafe1* and *cafe2* the camera was moved but not rotated. In *office**, *forest** and *parking**, the camera was rolled and tilted, so that

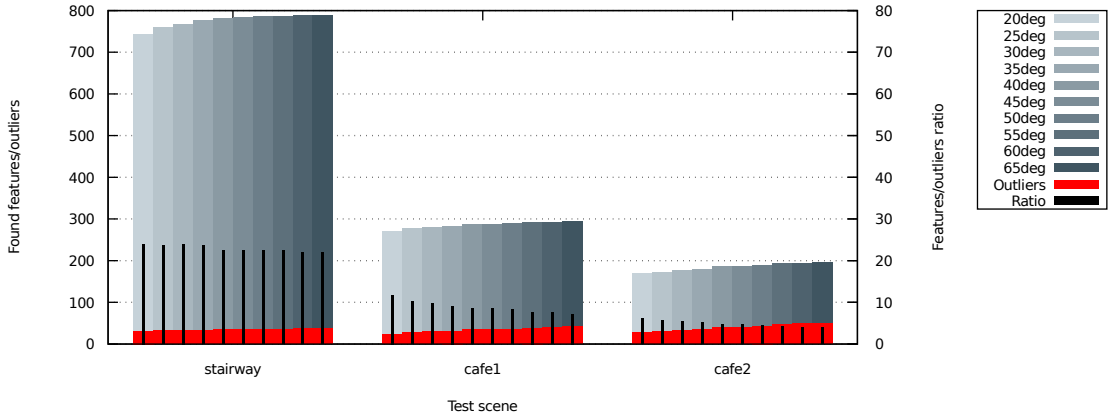


Figure 3.6: *Selecting a good rotation threshold for 2D features using the cosine fit strategy. We chose $e^{2D} = 35°$ because this threshold selects almost all correct feature matches while the match/outlier ratio is still high.*

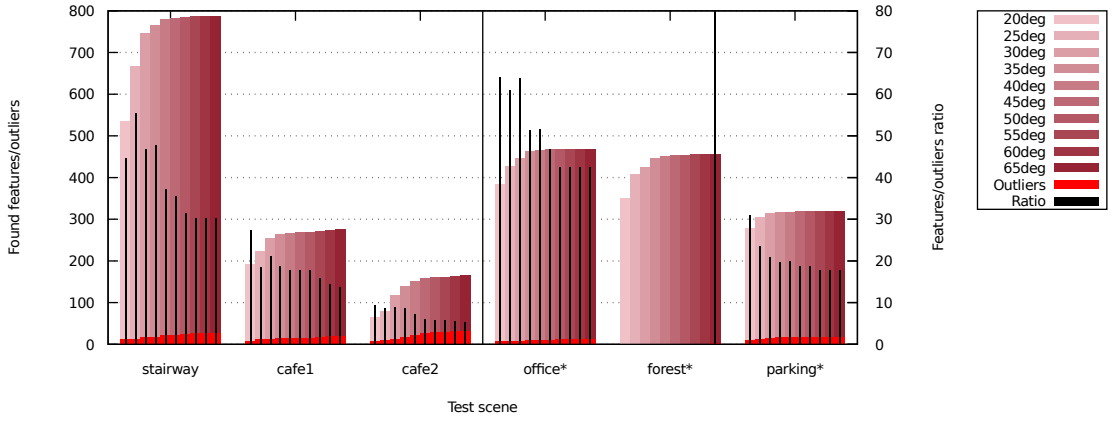


Figure 3.7: *Selecting a good rotation threshold for 3D features using the cosine fit strategy. We chose $e^{3D} = 35°$ because this threshold selects almost all correct feature matches while the match/outlier ratio is still high.*

Figure 3.8: *Ladybug test shots used for evaluation. From left to right: stairway, cafe1 (small baseline), cafe2 (wider baseline), office*, forest* and parking* (cloudy non static sky blackened). * scenes include camera roll and tilt.*

there is no uniform 2D image feature rotation. Features were extracted by a standard SIFT-like implementation [Low04] as the FoV for each camera is moderate.

The two scenes *office'* and *parking'* were acquired with a wide angle GoPro Hero 3 camera (Figures 3.13 and 3.14 right part only) for which we performed intrinsic calibration. Images were scaled down to 1600x1200 pixels. For dealing with the wide FoV, SIFTS [CMBP+12] was applied for feature extraction.

For all scenes, the feature descriptor based matches were classified manually to obtain ground truth for refinement.

3.5.1 Runtime Discussion

Let us first discuss the runtime to be expected by the different refinement strategies, namely histogram, mean, cosine fit and k-Means clustering.

The non-iterative histogram based and the mean strategies have $O(n)$ runtime complexity: Histogram bin assignment transforms each rotation to two bin indices (because we have overlapping bins) while the mean strategies basically sum up all rotations.

The iterative methods of cosine fitting and k-Means clustering exhibit $O(n)$ runtime *per iteration*: The cosine fit strategies compute the gradient contribution of each match's rotation with finite differences while the k-Means strategies obtain the distance of each rotation to the cluster centers.

Given that cosine fitting and k-Means clustering need a few iterations to converge, we can expect the mean strategy to be the fastest among the 3D orientation based methods.

3.5.2 Results

We first compare results of the de-facto standard SIFT-like histogram-based 2D refinement (Section 3.4.1) with the fast mean-based 3D refinement (Section 3.4.2). Figure 3.12 shows selected and discarded matches in the *office** scene. Note how our approach discards most false matches while keeping almost all inliers. Similar rotations of 2D feature orientations can only be found in a small image region. Therefore the 2D approach can-

not discern positive and false matches except for one small image region. Figures 3.9 and 3.11 show a random set of selected and discarded feature patches for both strategies. Note that the 2D strategy discards too many features because of inconsistent 2D image (and patch) rotations while our approach almost always classifies matches correctly.

In Figure 3.10, we visualize the rotations r_i or $\boldsymbol{R_i}$ of the matches for all scenes with all strategies. It is obvious that with the 2D strategies, the rotations are only similar for the first three scenes where the camera orientations don't change. The 3D strategies can handle all scenes correctly. In particular, the 2D mean strategy must fail by design in the * scenes (see Figure 3.1), while the 3D mean strategy produces similar rotations given the 3D feature orientations.

We present the number of the selected matches compared to the number of included outliers for all our test sets in Figure 3.13. As a baseline, we also present all found matches including all outliers without refinement. It can be clearly seen that our 3D refinement strategies perform better in terms of outliers per match in all scenes. In the rolled and tilted * scenes almost all correct matches survive while the 2D strategies fail as they only select about 50% of the correct inlier matches. In the other scenes, the 2D strategies keep most inliers whereas our 3D approach removes significantly more outliers.

One possible subsequent task for feature matching is relative camera pose estimation using the 8-point algorithm [HZ03] together with RANSAC [FB81] for robustness against the remaining outliers as described in Section 2.4.3. Figure 3.14 shows the number of RANSAC iterations according to Equation 2.62 that would theoretically be necessary for 8-point relative pose estimation with a tolerated error probability $p_{max} = 0.001$, based on the refined matches: our 3D orientation descriptor reduces the costs for that task by more than 50% on average by improving the input data.

3.6 Summary

In this chapter we proposed a novel 3D orientation descriptor for local image feature matching which is particularly designed for omnidirectional and wide FoV camera systems. We represent 3D feature orientations in the view space of the cameras, which makes them rotating inversely to the camera's motion. In viewspace, our orientation descriptor is invariant to image distortions or non-monotonic (e.g. cylindrical) projections from view space to image space. Simple match refinement based on our 3D orientation descriptors outperforms refinement based on 2D, image based orientation angles not only in cases where image features rotate differently in different parts of omnidirectional images, but also for wide FoV and distorted images where we can exploit intrinsic camera calibration and viewing direction.

Our method can be used with all local image feature frameworks that assign orientation angles to the keypoints because we can translate those to our 3D orientation representation based on the camera's intrinsic calibration in a straight-forward way. The need for a rough camera calibration is the only major limitation for our method. This can however

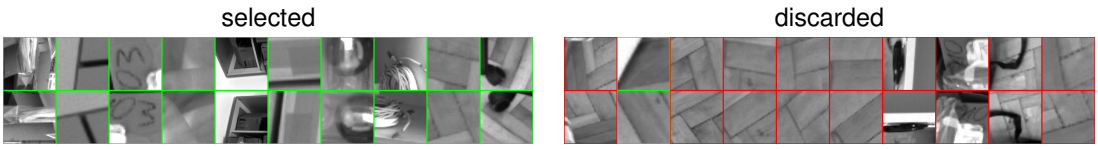


Figure 3.9: *Poor match refinement performance with 2D orientation based strategy. Randomly selected matched patches of the office* dataset refined with the 2D histogram approach. Selected (left) and discarded (right) patches. Green borders indicate correct classification. Many correct matches are discarded because they don't exhibit a similar 2D rotation behavior.*

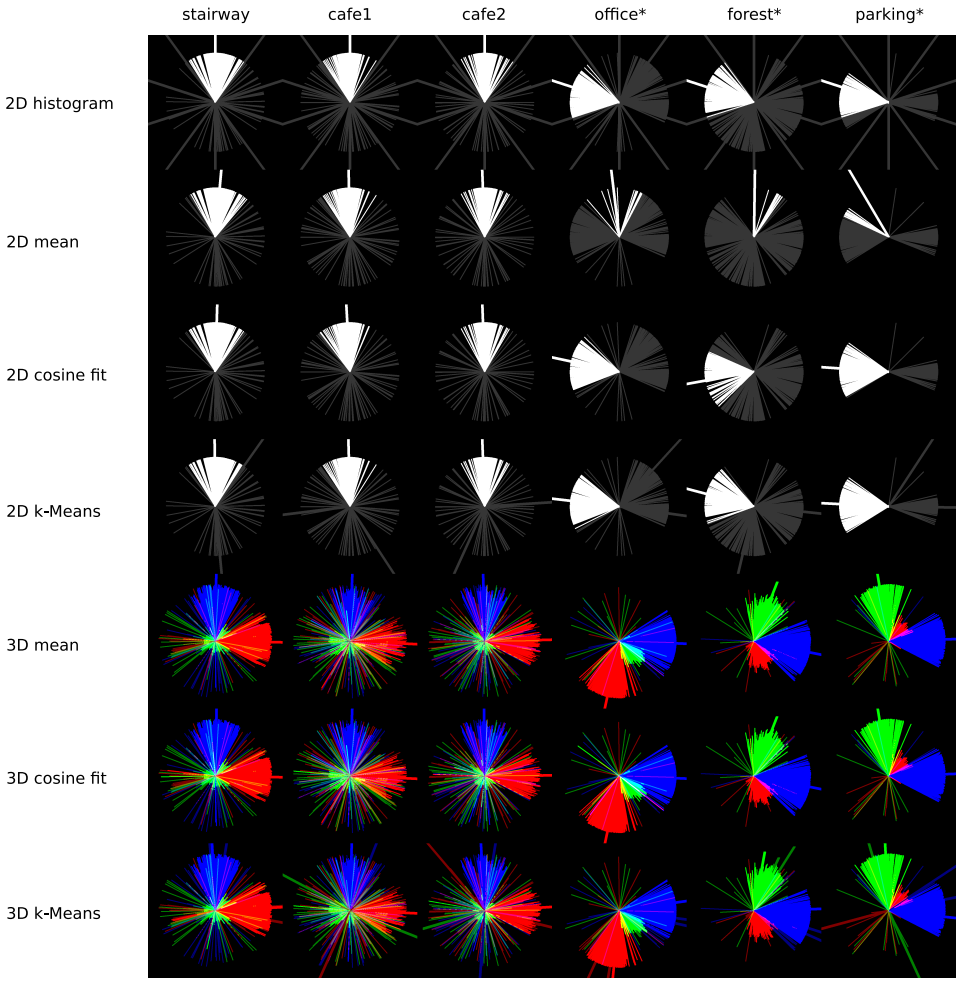


Figure 3.10: *Visualization of the relative feature rotations of matches. Long bars visualize the representatives $\hat{r}$ or $\hat{\boldsymbol{R}}$. Bright bars show selected matches and representatives while dark bars show discarded ones. 3D visualizations show the axes of the 1^{st} camera in the 2^{nd} camera's coordinate system based on the feature rotations of the matches. Omnidirectional camera rotations lead to multimodal 2D rotation distributions that cannot be separated correctly. 3D rotations distribute unimodal and are therefore easy to separate into inliers and outliers.*

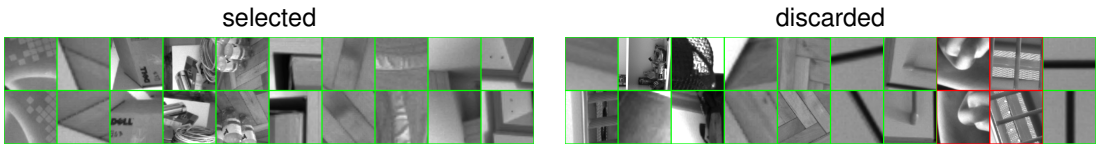


Figure 3.11: *Good match refinement performance with 3D orientation based strategy. Randomly selected matched patches of the office* dataset refined with the 3D mean approach. Selected (left) and discarded (right) patches. Green borders indicate correct classification. This refinement classifies similarly looking patches correctly even if they don't exhibit a similar 2D rotation.*

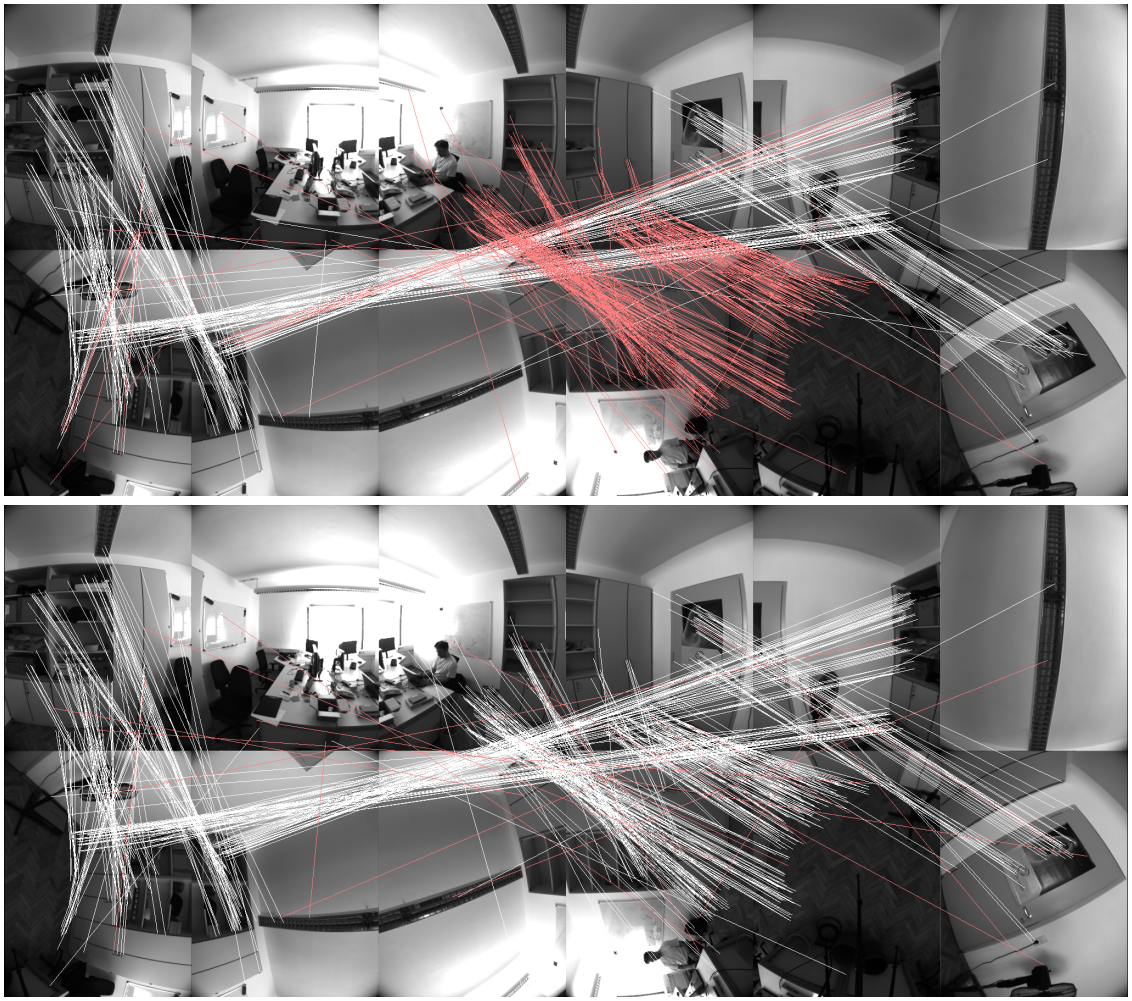


Figure 3.12: *Failing 2D orientation vs. nicely separating 3D orientation based match refinement. 2D histogram based (top, Section 3.4.1) vs. 3D mean (bottom, Section 3.4.2) strategy. White lines represent selected, red lines discarded features. Our 3D refinement discards false matches selectively in the entire omnidirectional image while the 2D refinement only succeeds for a compact region of viewing directions, discarding lots of inliers.*

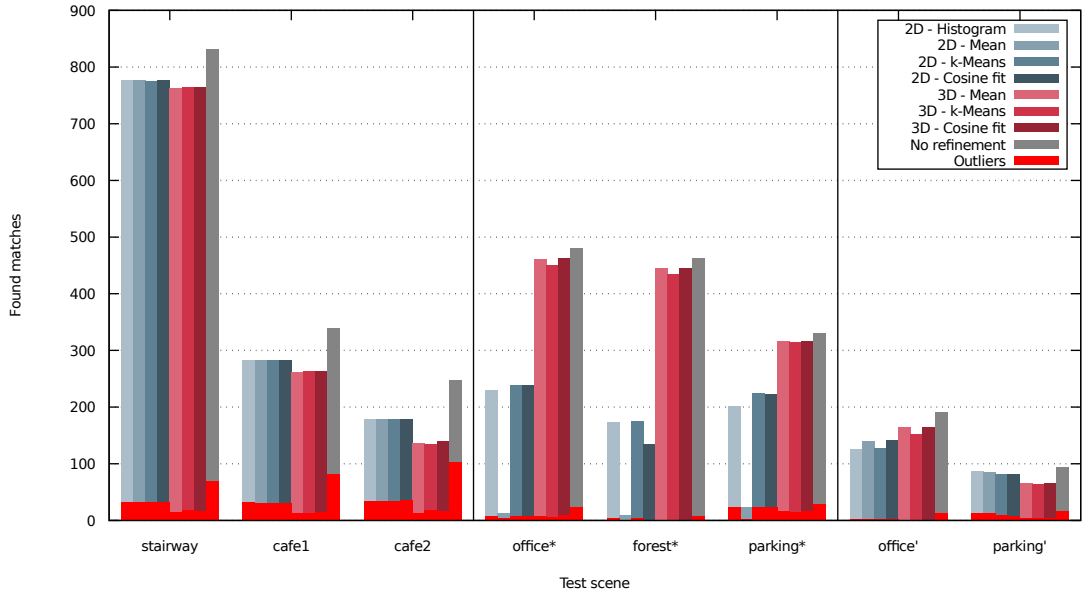


Figure 3.13: *Improvements from 3D orientation descriptors for outlier/match ratio and number of matches. 3D orientations lead to the fewest outliers per match in all test scenes. 3D strategies often keep less matches for fixed orientation omnidirectional image pairs (left part) and wide FoV image pairs (right part), which can mainly be explained by better outlier elimination. In the rolled/tilted * scenes, our 3D descriptor helps to discard false matches selectively while finding most of the correct matches which is about 100% more than the 2D orientation based methods can find.*

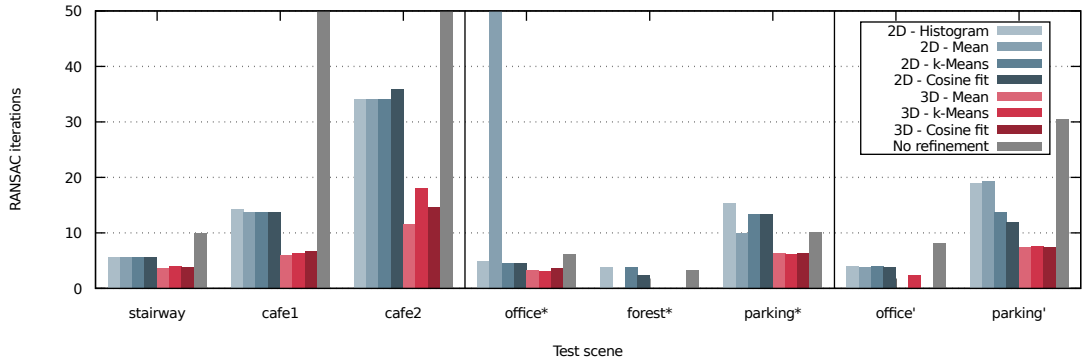


Figure 3.14: *Computational cost reduction by 3D orientations for exemplary subsequent RANSAC task. We show the number of iterations theoretically necessary for 8-point relative pose estimation desiring 99.9% success rate. 3D orientation based match refinement reduces the computational cost of this subsequent task by about 50%.*

be solved by pre-calibrating the cameras. Due to the fact that we work with rotation tolerances of 35°, it is often sufficient to simply guess the calibration parameters.

Our match refinement's computational overhead is low compared to other tasks in feature extraction and matching, but the use of the 3D orientation descriptor can dramatically speed up subsequent tasks because of the reduced number of outliers while keeping almost all correct inliers.

Given very reliable feature tracks, the next step is the actual reconstruction of the 3D geometry. In contrast to baseline method described in Section 2.4, we propose a method that works very efficiently on highly redundant data in the next Chapter.

Chapter 4

Scalable Structure from Motion on Highly Redundant Data

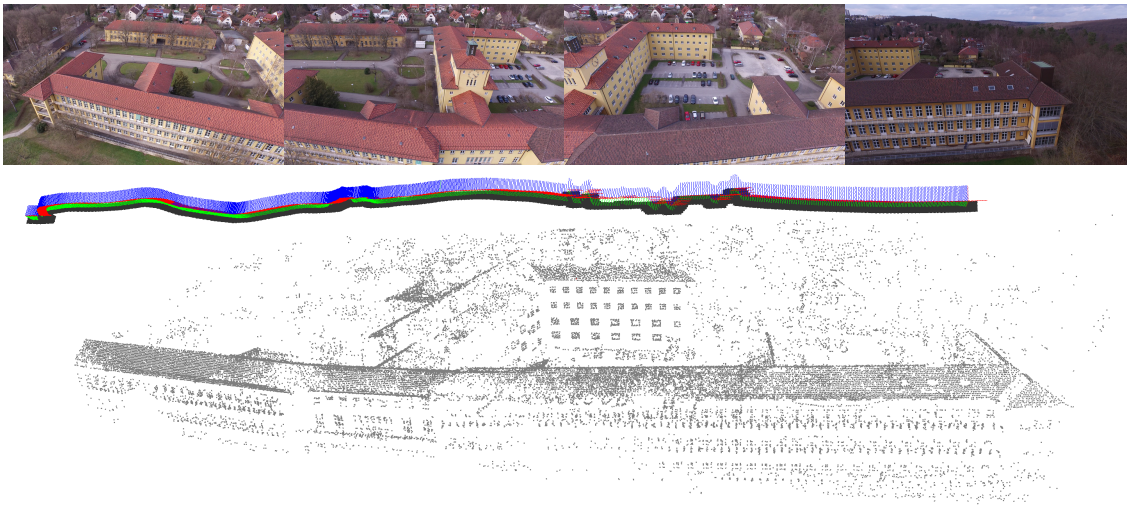


Figure 4.1: *Reconstruction of the main building of our computer science department, based on a 4k drone video sequence.*

High frame rate videos consisting of thousands of high resolution frames are challenging for existing SfM and SLAM algorithms. We present a new approach for simultaneously computing extrinsic camera poses and 3D scene structure that is capable of handling such large volumes of image data. The key insight in this chapter is to effectively exploit coherence in densely sampled video input. Our technical contributions include robust tracking and selection of confident video frames, a novel window bundle adjustment with content- and pose-adaptive keyframe selection, frame-to-structure verification for globally consistent reconstructions with multi-loop closing, and utilizing efficient global linear camera pose estimation in order to link both consecutive and distant bundle adjustment windows. To our knowledge we describe the first system that is capable of handling high resolution, high frame-rate video data with close to real-time performance. In addition, our approach can robustly integrate data from different video sequences, allowing

multiple video streams to be simultaneously calibrated in an efficient and globally optimal way. We demonstrate high quality alignment on large scale challenging datasets, e.g., 2-20 megapixel resolution at frame rates of 25-120 Hz with thousands of frames.

4.1 Introduction

Structure from Motion (SfM), i.e., reconstructing camera (pose) parameters and sparse scene structure from images, has a long history in computer vision. Early approaches concentrated on reconstruction from videos based on feature tracking techniques (see, e.g., [SK94, HZ03] for an overview). With the advent of robust feature descriptors like SIFT, SURF or ORB (for an overview refer to Section 2.1.5), larger view differences could be matched, and SfM techniques were successfully extended to very large scale, unstructured sets of images [SSS08, ASSS10]. Coupled with the availability of online photo collections, these approaches have become very popular, enabling a wide range of novel applications such as 3D reconstruction from thousands of photographs [AFS+11, ZGW+14].

However, for individual users, and in personalized application domains, it is often much simpler and more practical to capture video instead of photographs to ensure a sufficiently broad as well as dense coverage of a scene. At the same time, video resolution and frame rate are constantly increasing. Mobile cameras such as the iPhone can record more than 120 frames per second, yielding thousands of images in just a few seconds of capture. Older approaches based on feature tracking as well as modern sparse feature point and SLAM-based approaches do not scale well to such densely-sampled data, both due to numerical inaccuracies arising from small baselines, and computational tractability associated with the sheer quantity of frames and pixels.

But while densely sampled, high quality video data presents many challenges, it also provides opportunities. For instance, [YG14, KZP+13] have demonstrated how spatiotemporal coherence can be exploited in the context of 3D reconstruction. In the remainder of this chapter, we show that densely sampled data also enables novel, more efficient strategies for achieving globally consistent geometric calibrations.

4.1.1 Contributions

The main technical contributions that we propose are: a modification to KLT that allows for drift free tracking over thousands of frames, a robust selection of confident frames, a novel *interleaved* window bundle adjustment (BA) that makes optimizing large windows more efficient, uniform image coverage based point subsampling, robust frame-to-structure verification to obtain global, wide baseline anchors between camera poses, and the utilization of an efficient linear camera pose estimation (LCPE) method that integrates information from both BA windows and global anchors in a unified way.

As opposed to prior work, our approach does not rely on analytical, fixed input size n-Point methods (see Table 2.2), which we observed to be not good enough due the fact that they use less data than BA, and therefore yield less precise results for the small baselines of densely sampled video input. Instead, we demonstrate that it is possible to rely solely on nonlinear optimization for initialization, loop closing, and linking of multiple independent video sequences. We address the problem of small baselines using a point parameterization approach from [YG14].

When all of the contributions are combined, our method is able to obtain a globally consistent extrinsic camera calibration in substantially less time then previous approaches, and on datasets with thousands of high resolution frames. Furthermore, our approach generalizes to an arbitrary number of input video sequences, allowing for rapid, globally consistent calibration and scene reconstruction across multiple capture devices. In this chapter we describe all of these contributions.

In our system we leverage established existing approaches, such as the commonly used KLT tracker [TK91] (implementation from [Bra00]), SIFT histogram based frame similarity cost matrices [WSZ+14], a global linear solver that integrates relative camera pose constraints [JCT13], and a robust depth-based point parameterization [YG14].

4.2 Related Work

The method proposed in this chapter is mainly related to two different fields of research (compare also to Section 2.6): Structure from Motion (SfM) concentrates mainly on unstructured, sparse input like photo collections, while Visual Odometry (VO) / Simultaneous Localization and Mapping (SLAM) is more related to coherent, densely sampled input.

4.2.1 Unstructured, Sparsely Sampled Input

A key challenge for methods focusing on sparsely sampled input such as photo collections is that the data is generally unstructured and heterogeneous, with significant appearance changes between images. This means on the one hand that it is difficult to find out which images show the same scene content, on the other hand the scene has to be build up iteratively as outlined in Section 2.4 and quite some effort has to be put into keeping the reconstruction consistent, usually by running BA regularly. This approach has been successfully extended to massive, very large scale datasets [SSS08, AFS+11, FFGG+10], with various publicly available implementations [SSS06, W+11].

Bootstrapping can be simplified by concerning only relative camera poses instead of the full BA reconstruction: Martinec and Pajdla [MP07] present a robust solution for finding global camera poses, concentrating on the camera orientation, while Wilson and Snavely [WS14] and Jiang et al. [JCT13] show how to find global camera positions given known orientations.

Another approach is to add images to the reconstruction in a way that ensures consistency over the whole scale of the scene early, e.g. by starting from few but well connected camera poses, which cover the greatest part of the scene. To this end, techniques such as skeletal graphs [SSS08] have been proposed, which first remove unnecessary data by focusing on stable subsets of camera poses. Agarwal et al. [ASSS10] showed that it is necessary to reconsider well established strategies in order to tackle large datasets consisting of 10s of thousands of images.

A further alternative is to perform an incremental, piecewise reconstruction of a scene [PS12], and later assemble individual fragments based, e.g., on extracted scene point descriptors.

All these methods are tuned towards heterogeneous, unstructured data, and as a consequence have difficulty when applied to densely sampled, coherent image sequences. This is due to per-frame feature point detection and pose estimation using n-point algorithms which cause unstable reconstructions, as well as computational inefficiency. In our experiments, we show that by explicitly considering coherence in the data, it is possible to achieve high quality reconstructions at significantly faster convergence and computation times.

4.2.2 Coherent, Densely Sampled Input

In contrast to above, most techniques for densely sampled input such as video sequences are based on continuously tracking feature points (see Section 2.1.2) throughout image sequences and iterative pose optimization techniques [SK94, HZ03, PVGV+04]. These original methods were designed for short, low resolution video sequences and did not consider multi-loop closing.

Particularly related are SLAM approaches and their variants, as their aim is to compute accurate camera poses of a dynamically moving camera from a video stream. Often, however, such techniques are limited with respect to the supported scene size [KM07] or require additional sensor modalities [LM13]. Real-time methods based on feature points [DRMS07] or dense, per-pixel tracking [NLD11] are generally designed to provide as good as possible results with a small input lag, rather than a final, fully consistent and high quality reconstruction that globally optimizes the poses of all input frames. CoSLAM [ZT13] combines data from cooperatively acquired videos as long as some of the cameras see the same content at the same time. Other approaches achieve real time performance, but only on preconstructed scenes [LSC+15], i.e., with known geometry.

With LSD-SLAM [ESC14] (outlined in Section 2.5), a direct method was proposed, which does not require detection and tracking of feature points, but instead recovers sparse depth maps based directly on epipolar line scanning. However, such an approach does not scale well to high image resolutions, as it requires depth estimates for many pixels. In addition, depth recovery is very sensitive to accurate intrinsic calibration (more details in Section 5.1). Our approach instead focuses on a subset of reliable features tracks, which is more efficient and less sensitive to image distortions, especially for high

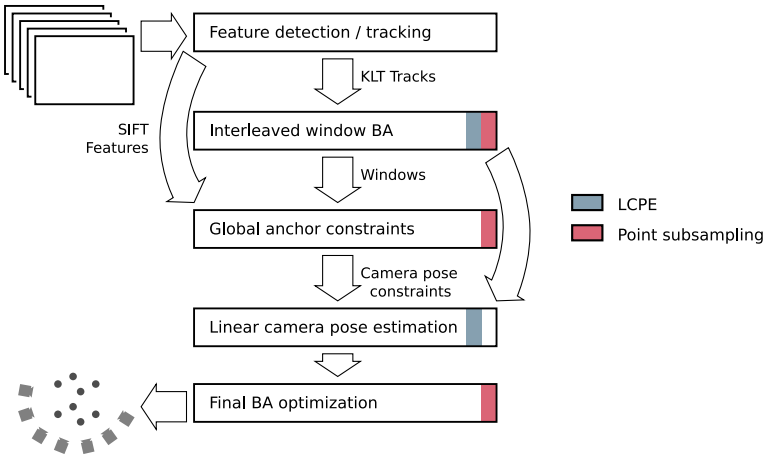


Figure 4.2: *Algorithm overview. The feature detection stage computes KLT tracks for window BA and SIFT features for wide baseline handling. The window BA sweeps through the input, selects confident frames and computes camera pose constraints between interleaving sets of the selected frames. The global anchor stage uses the SIFT features to establish global links between different sections of the sequence. A linear camera pose estimation (LCPE) produces an initial arrangement of camera poses which is further refined by bundle adjustment steps. Note that the LCPE as well as our point subsampling strategy are reused in several parts of the algorithm.*

resolution input. As a result, our method is able to find good solutions on datasets where LSD-SLAM fails to converge and produces results faster, especially for high resolution input.

Specific light field calibration techniques have been proposed for dense spatio-temporal-angular sampling using camera arrays [WJV+05, KZP+13, HHA+10] and plenoptic cameras [Ng06]. However, these methods generally focus on the static geometric calibration of a light field, rather than computing both structure and motion, and hence cannot be applied to the acquisition scenarios we discuss in this chapter. The work on unstructured light field acquisition [DLD12] explores this to some extent, but only supports small scale scenes and focuses on an interactive interface for guiding the user during the acquisition process.

Our method focuses on densely sampled image sequences and overcomes several of the previously mentioned limitations. This results in a SfM approach that is stable and globally consistent over long, high resolution sequences, while still being able to robustly handle wide baseline matches.

4.3 Method

The input to our method is one or more image sequences. We focus on extrinsic calibration and assume the intrinsics to be fixed and known (in practice they can be computed from a few frames of the image sequences by using Bundler [SSS06]).

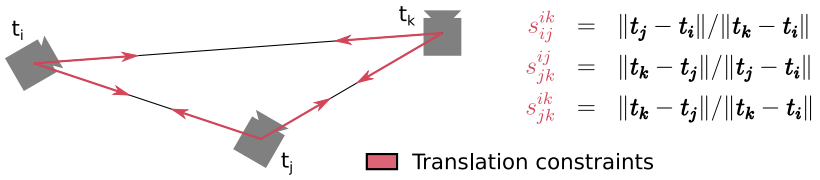


Figure 4.3: *Translation constraints for the global linear camera pose solver. Global space direction vectors between camera positions and distance ratios s_{ij}^{ik}, s_{jk}^{ij} and s_{jk}^{ik}.*

On a high level our strategy is as follows. First, we perform a modified 2D tracking of feature points utilizing data coherence to reduce drift. Next, we apply a window BA strategy on a set of *confident* frames only. These are frames that are well connected via continuous tracks. To incorporate loop closing, we further establish global anchor links between carefully selected frame pairs of different parts of the video or even different video streams altogether. In addition to these global constraints, relative camera pose constraints from the window BA are integrated with an efficient linear camera pose estimation [JCT13]. We then perform global BA, and finally add all the less confident images by interpolation and BA of their poses. During this step, we keep the scene structure fixed as determined by the confident images. The final result is a globally consistent calibration of all input frames from all input sequences.

Our reconstruction method features two reoccurring building blocks which we describe first: the linear camera pose estimator (LCPE) and our method for point subsampling in Sections 4.3.1 and 4.3.2. Then we continue by describing the key steps of our algorithm in the following sections in the order shown in Figure 4.2.

4.3.1 Linear Camera Pose Estimation

To avoid running BA unnecessarily often, we use the linear camera pose estimation technique proposed by Jiang et al. [JCT13]. This linear (and therefore fast) algorithm solves for global camera rotations and positions in a least squares sense based on pairwise camera rotation constraints and triplet camera translation constraints. Since it is not available as an out-of-the-box implementation, we describe it here briefly.

Input and Output

For solving for the global camera rotations, relative rotations for pairs of camera poses have to be supplied so that all poses are connected transitively by such constraints.

For solving for the global camera positions, constraints for triplets of camera poses have to be given which contain the ratios of the distances between the camera positions in the triplet and the direction vectors from each camera's position to the others (see Figure 4.3).

It should be clear that both rotation and translation constrains can be extracted from a BA reconstruction with at least three camera poses. According to [JCT13], an (over-)de-

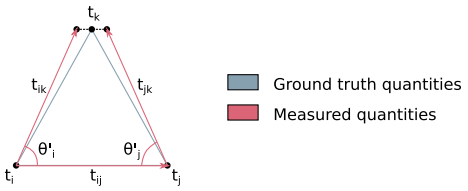


Figure 4.4: *Illustration of the variables used for recovering globally consistent camera positions.*

termined problem can be obtained when all camera poses are connected transitively by triplets which have at least two camera poses in common (to transport scale information throughout the whole optimization problem). In fact, we use all possible camera pose pairs / triplets from our BA reconstructions to generate constraints for this linear method.

The result of Jiang et al.'s method is a globally consistent rotation matrix $\boldsymbol{R_i}$ and a translation vector $\boldsymbol{t_i}$ per camera pose, optimized in a least squares sense.

Recovering Rotation

For recovering globally consistent rotations, the method of Martinec and Pajdla [MP07] is employed: Given the relative rotations $\boldsymbol{R_{ij}}$, the following must hold for every column vector $\boldsymbol{r_i^k}$ of a resulting global rotation matrix $\boldsymbol{R_i} = [\boldsymbol{r_i^1 r_i^2 r_i^3}]$:

$$\boldsymbol{r_j^k} - \boldsymbol{R_{ij}} \boldsymbol{r_i^k} = \boldsymbol{0} \tag{4.1}$$

By concatenating all such constraints from all the camera pose pairs given as input, one can obtain an overdetermined linear system of equations which can be solved for all $\boldsymbol{r_i}$ according to Section 2.3.1. Since orthonormality of the recovered $\boldsymbol{R_i}$ matrices is not enforced in the linear equations, the resulting matrices have to be projected to the space of orthonormal matrices. This can be done in a least squares sense by using a SVD as shown in Equation 3.7.

Recovering Translation

Given the distance ratios and direction vectors between a triplet of camera poses as shown in Figure 4.3, we can express one of the three camera translation vectors $\boldsymbol{t_k}$ based on the other two $\boldsymbol{t_i}$, $\boldsymbol{t_j}$ like illustrated in Figure 4.4 as:

$$\boldsymbol{t_k} \approx \frac{1}{2}\Big((\boldsymbol{t_i} + \boldsymbol{R_i}(\theta_i')s_{ij}^{ik}(\boldsymbol{t_j} - \boldsymbol{t_i})) + (\boldsymbol{t_j} + \boldsymbol{R_j}(-\theta_j')s_{ij}^{jk}(\boldsymbol{t_i} - \boldsymbol{t_j}))\Big) \tag{4.2}$$

Here, $\boldsymbol{R_i}()$ is a rotation matrix around the axis $\boldsymbol{c_{ij}} \times \boldsymbol{c_{ik}}$ and can be obtained from the direction vectors between the triplet of camera poses. By assuming that either $\boldsymbol{t_{ij}}$, $\boldsymbol{t_{ik}}$ or

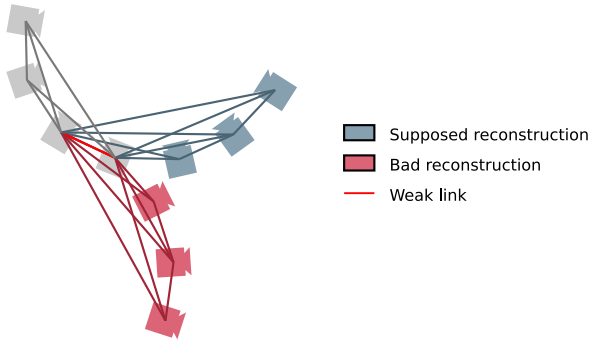


Figure 4.5: *Example for partially rotated camera pose reconstruction because of underconstrained linking of scene parts. The blue and gray parts can rotate against each other around the weak link's axis.*

$\boldsymbol{t_{jk}}$ are free from error, one can define the following three linear constraints:

$$2\boldsymbol{t_k} - \boldsymbol{t_i} - \boldsymbol{t_j} = \boldsymbol{R_i}(\theta_i')s_{ij}^{ik}(\boldsymbol{t_j} - \boldsymbol{t_i})) + \boldsymbol{R_j}(-\theta_j')s_{ij}^{jk}(\boldsymbol{t_i} - \boldsymbol{t_j})) \quad (4.3)$$

$$2\boldsymbol{t_j} - \boldsymbol{t_i} - \boldsymbol{t_k} = \boldsymbol{R_i}(-\theta_i')s_{ik}^{ij}(\boldsymbol{t_k} - \boldsymbol{t_i})) + \boldsymbol{R_k}(\theta_k')s_{ik}^{jk}(\boldsymbol{t_i} - \boldsymbol{t_k})) \quad (4.4)$$

$$2\boldsymbol{t_i} - \boldsymbol{t_j} - \boldsymbol{t_k} = \boldsymbol{R_j}(\theta_j')s_{jk}^{ij}(\boldsymbol{t_k} - \boldsymbol{t_j})) + \boldsymbol{R_k}(-\theta_k')s_{jk}^{ik}(\boldsymbol{t_j} - \boldsymbol{t_k})) \quad (4.5)$$

Given the constraints, a linear system $\boldsymbol{At} = \boldsymbol{0}$ can be set up where $\boldsymbol{t}$ is the concatenation of all camera translations and $\boldsymbol{A}$ contains all coefficients from the three linear equations for all camera pose triplets. The solution can be obtained by calculating the eigenvector associated to the fourth smallest eigenvalue of $\boldsymbol{A^T A}$. The eigenvectors corresponding to the three zero eigenvalues correspond to the three degrees of freedom from the origin of the coordinate system.

Issues

Beside several ambiguities described in [JCT13], we observed two issues with the algorithm which needed special treatment:

Partial Rotation. The scene reconstruction may be partially rotated. This happens when there are two subsets of camera poses to be obtained and all the triplets that overlap with both of the subsets contain the same "weak" link. As illustrated in Figure 4.5, false reconstructions are possible in this case which satisfy all the angular and ratio constraints applied.

Such configurations must be avoided, e.g. by using only BA reconstructions with at least four camera poses as input for the linear camera pose estimation and extract all possible triplets from them.

Numerical Instabilities. Due to inaccuracies in the input and numerical instabilities, small eigenvalues of the matrix $\boldsymbol{A}^T\boldsymbol{A}$ may switch which leads to selecting a wrong eigenvector as solution to the camera translation problem.

To compensate for that, we take the eigenvectors corresponding to a range of eigenvalues and re-check the constraints (Equations 4.3, 4.4 and 4.5). Finally, we pick the eigenvector that fits the constraints best.

4.3.2 Point Subsampling

Another reoccurring building block in our algorithm is point subsampling. Following the observation that BA requires a certain minimum number of scene points but does not improve significantly with many more points, we employ a subsampling scheme on the available scene points in all BA steps. We found that sampling points randomly can lead to unstable or underconstrained optimization. We therefore choose point samples according to three rules explained below, achieving similar reconstruction quality to the full set of points at a fraction of the optimization time. In fact, our point subsampling makes subsequent BA independent of the image resolution as the number of sampled points is only depending on the number of frames but independent from the number of features in the frames.

- First, a minimum number of points should be visible from each camera pose.
- Second, the reprojected 2D positions of the points should be uniformly distributed in all camera images.
- Finally, the points should be visible in "sufficiently many" images, since in general a point provides more reliable constraints when seen from more camera poses. At the same time, however, we observed that points visible in a very large number of frames, i.e. points with very long 2D tracks, are more likely to be affected by tracking errors along object silhouettes, corrupting the result.

Selecting points according to these rules is not trivial: e.g., a good spatial distribution in one image might lead to a particularly bad one in another. Therefore, we propose a multi-image stratified sampling approach. We separate each image into $U \cdot V$ regularly spaced, rectangular strata and assign the corresponding image features. Then we select points as follows:

1. Among all images, we choose the stratum showing the fewest selected points.
2. From this stratum, we select the 3D point of the feature with the highest score

$$\Lambda(f) = max(10, |f(p(f)))| \tag{4.6}$$

which corresponds to the number of features that 3D point of the feature has, cropped at 10.

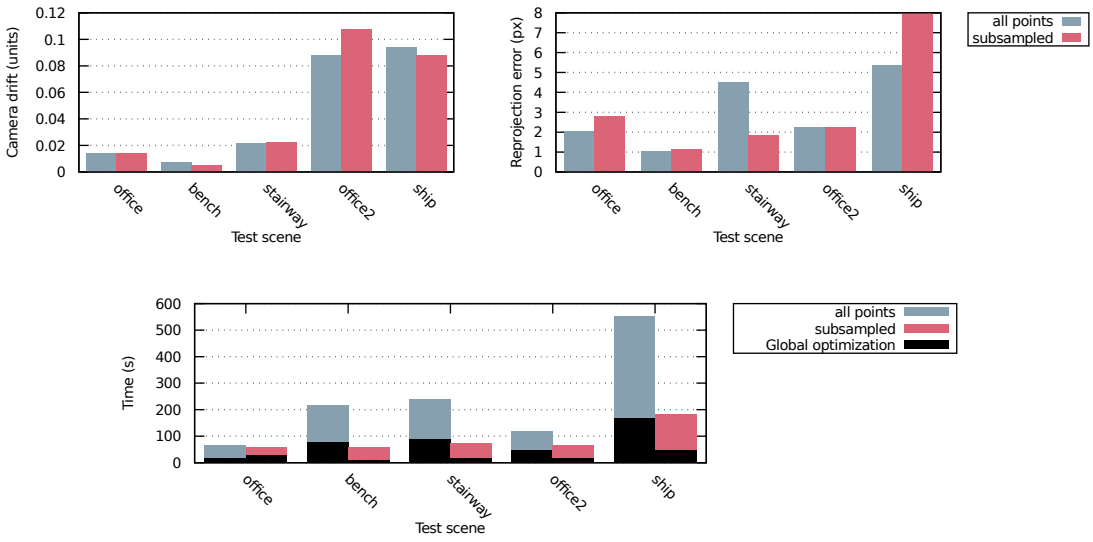


Figure 4.6: *Comparison of reconstruction with and without point subsampling. Our subsampling strategy leads to comparable reconstruction results while reducing the computation time by roughly a factor of 3. This factor is assumed to increase on larger image resolutions since more points can be excluded there.*

3. We continue at 1. until the stratum showing with the fewest selected points is above a certain threshold τ.

For all our experiments, we used the values $U = 6, V = 4$ and $\tau = 2$. The subsampling resulted in a 3-fold speedup without sacrificing result quality (see Figure 4.6).

In the next sections, we will describe the individual parts of our framework as shown in Figure 4.2.

4.3.3 Drift Reduced Feature Tracking

Detectors optimized for wide baseline matching such as SIFT [Low04] compute incoherent feature point sets even between neighboring video frames. For continuous sequences, feature point tracking produces more reliable and efficient results. We build on the standard KLT tracker implemented in OpenCV [Bra00], which is also the basis for earlier video-centered SfM techniques [SK94]. There are, however, two limitations of standard continuous KLT that have to be addressed in our application setting.

Firstly, we observed that for densely sampled video sequences, feature tracks that are visible for hundreds of frames exhibit noticeable drift. Note that for high frame rate cameras, this often corresponds to just a *second* of video. We therefore modify the basic tracking to perform a simple drift correction: when adding a frame, we track each fea-

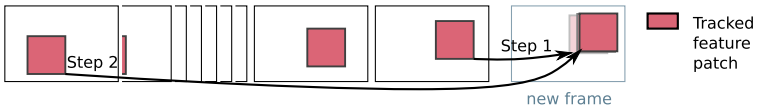


Figure 4.7: *Illustration of our KLT drift correction. After tracking a feature from the previous frame, we refine its position based on the patch on the image where the feature was detected originally.*

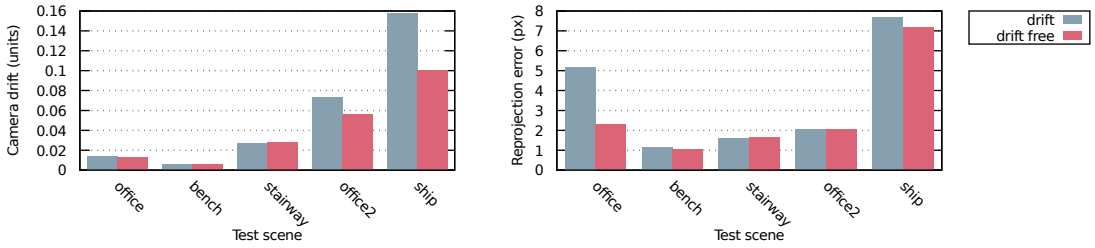


Figure 4.8: *Influence of KLT drift on the reconstruction result. Note how drift free KLT reduces the drift of the camera positions as well as the average reprojection error.*

ture from the previous frame to the new one, and then refine the feature position in the new frame using the *original* frame where the feature was detected (see Figure 4.7). In our experiments this simple modification led to considerably reduced drift and higher reconstruction quality (see Figure 4.8).

Secondly, simply tracking points over an image sequence cannot guarantee any form of global consistency of the reconstructed camera poses and scene. For example, when the camera revisits the same scene elements multiple times over a longer image sequence with intermediate occlusions, a single scene point will be represented by multiple, individually tracked and reconstructed points. This problem is known as the so called loop-closing problem in SLAM. For each feature track we therefore extract SIFT descriptors [Low04] in confident frames after the window BA, which are later used to re-identify points and for the generation of global anchor constraints.

4.3.4 Interleaved Window Bundle Adjustment

Given the feature tracks, the goal of our window BA is to efficiently reconstruct camera poses for image sub-sequences without considering global consistency or jointly processing multiple individual sequences, both of which will be addressed later.

Window Initialization

Initializing camera pose geometry is usually accomplished using n-point algorithms (see Table 2.2), which are (in contrast to BA) limited in the number of constraints and therefore

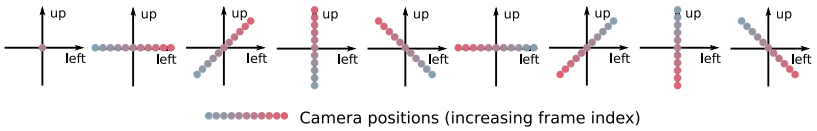


Figure 4.9: *Illustration of different camera pose initializations. Poses are arranged uniformly along a line orthogonal to the viewing direction in 45° steps.*

generally not sufficiently accurate given the very small camera baselines encountered in high frame rate video sequences. Our method is inspired by the approach of Yu and Gallup [YG14] designed for accidental small baseline camera motion.

We initialize a window by picking the first N consecutive images from an image sequence and immediately perform a BA step using the parameterization proposed in [YG14], where points are represented by inverse depth values projected from a reference frame (we use the center image in the window). We found, however, that identical initialization of all camera poses [YG14] may cause BA to get stuck in local minima. According to our observations, this can reliably be avoided by starting from different linearly displaced configurations (see Figure 4.9) and optimizing first for the camera orientation and then for all extrinsics. Finally we pick the best result in terms of reprojection error. Moreover, we observed more robust results when initializing scene points with uniform instead of random depth [YG14]. The original method of Yu and Gallup requires a comparably large number of images for robust convergence. With our above modifications we observed stable convergence already with $N = 11$. For high frame rate handheld video, spacing between frames (e.g., 3 in our experiments) for slightly increased baselines led to improved convergence.

The next step is to grow this window. For selecting further frames to be added to the window and for their later removal, we specify confidence criteria.

Confidence Criteria

For efficiency reasons, and to improve result quality, we compute scene structure and camera poses initially only for a sparse set of confident frames. In order to find this set, we test camera poses with a linearly increasing step size and add the furthest possible frame fulfilling a set of confidence criteria. We define the following three confidence criteria ξ_1, ξ_2, ξ_3 for measuring whether a tested camera pose c_t is suitable for window BA:

- The first term ξ_1 measures the number of features of the camera pose c_t that can be matched to the points p_i of the window with a low reprojection error. This ensures that there are sufficiently many constraints for BA.

- ξ_2 represents how far the camera has moved around the scene points. We use the median of all points' angular differences $\widetilde{\phi}(\overrightarrow{\boldsymbol{pc_p}}, \overrightarrow{\boldsymbol{pc_n}})$ between the vectors to the

tested c_t and the previous camera pose c_p. This term makes sure that two camera poses are not too far apart from each other and ensures that the visual appearance of the feature points doesn't change to much so that the next confident frame has mostly the same feature tracks.

- The last term ξ_3 is set to the median reprojection error $\widetilde{e}$ of the tested camera pose c_t and its visible points p_t: $\xi_3 = \widetilde{e}(c_t, p_t)$. This ensures that no camera poses are added to the optimization which are too inconsistent with the content of the window.

We label a camera pose as sufficiently confident when the following criteria are fulfilled: $\xi_1 \geq 30$, $\xi_2 \leq 5°$, $\xi_3 \leq 5$px, i.e., the camera pose must be linked to at least 30 points, must not rotate more than five degrees around at least half of these points, and at least half of the points must have less than five pixels reprojection error. Similarly, a camera pose is labeled as candidate for removal from the current window as soon as it does not satisfy the following confidence constraints anymore: $\xi_1 \geq 70$, $\xi_2 \leq 10°$, i.e., at least 70 points and less than ten degrees of camera rotation around at least 50% of the points. We always keep at least 5 camera poses in our window, which avoids directly dropping camera poses after adding them because of the more strict ξ_1 removal constraints. Instead, camera poses usually stabilize fast in the window, gaining additional links to points from subsequently added frames.

Window Processing

Given the confidence criteria for addition and removal, in each iteration of the window bundle adjustment, we first remove images labeled for removal from the current window, keeping a minimum of 5 camera poses in the window at all times. Then, we add a new frame to the window according to our confidence criteria. After this step, the current window contains usually about five to ten confident camera poses.

However, for some camera (sub-)trajectories, stable windows can be much larger. To retain the efficiency of BA while keeping as much information as possible, we select a subset of the camera poses in the window on which to perform the actual BA. We pick camera poses with increased spacing for older images (see Figure 4.10). This subset is then optimized using standard BA.

After BA, all camera poses in the current window are made consistent with a linear camera pose estimation technique described in Section 4.3.1, using the relative camera pose constraints of former windows. This solver works on the camera poses only, producing faster results than BA in comparable quality as long as the input is consistent.

We experimented with various offsetting strategies besides the growth strategy described above (see Figure 4.12). Beside the selection pattern with linearly growing offsets, we also tried exponentially growing offsets. This however leads to significantly worse results because the offsets are greater in general and because this pattern generates more redundant information: it happens more often that the same cameras end up in BA

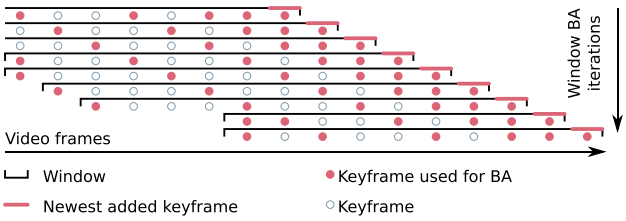


Figure 4.10: *Selecting keyframes for interleaved window BA. Offsets between the keyframes selected for BA are linearly increasing towards older frames. To determine a consistent pose for a camera which was not part of the BA, we use the relative pose constraints that were generated in previous windows where the camera pose was part of the BA.*

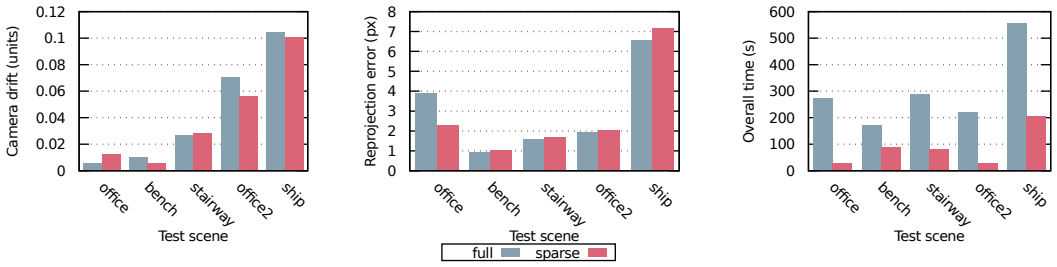


Figure 4.11: *Keyframe selection evaluation. We compare our implementation optimizing only keyframes together with the points (sparse) to an implementation that does a full global BA (full). Using a sparse camera pose set yields results comparable to the full optimization but is 2-10x faster. This factor is assumed to increase with higher frame rates since less keyframes are selected for the sparse set.*

together. We also tried using at most seven consecutive or all available cameras in the window for BA which leads to worse results and/or increased runtime.

The output of this stage are camera pose constraints from each window, which we will later use for initializing the global scene optimization. We also keep all the windows for finding global anchor constraints as described in the following section.

In order to demonstrate the robustness of our confident frame selection process we compare the results of just using those frames in the final BA optimization to a full BA reconstruction using all frames. Figure 4.11 shows that there is on average no quality penalty compared to the benefit of considerably decreased compute time with our frame selection process.

4.3.5 Global Anchor Constraints

The goal of these constraints is to establish global links between different parts (possibly different subsequences) of a video that have shared scene content. These links can later be used in the linear camera pose estimation stage (see Figure 4.2) to obtain a good global initialization.

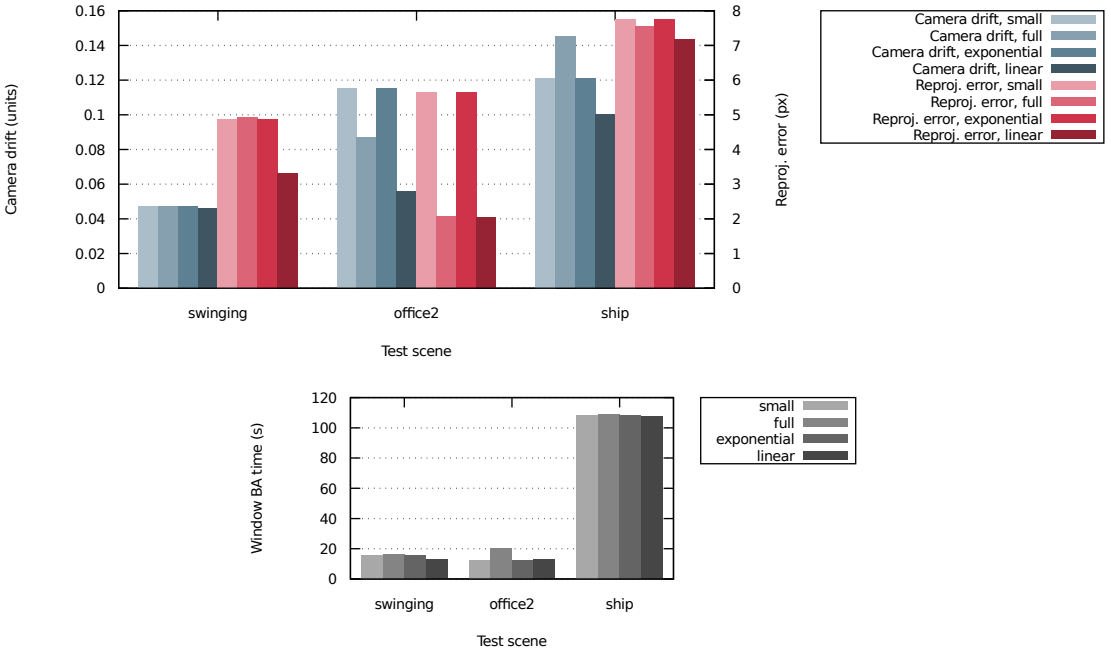
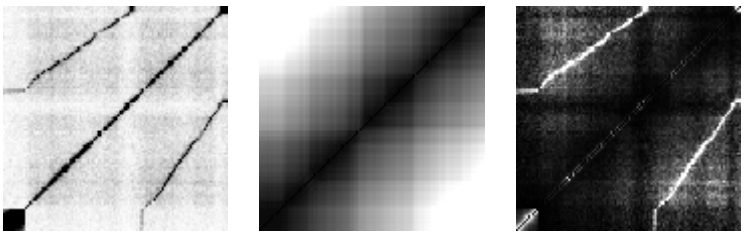


Figure 4.12: *Comparison of the reconstruction results with different keyframe spacings in the window. We compare small (first 7 keyframes), full (all keyframes in the window), exponentially spaced and linear increasingly spaced reconstruction windows. Linear spacing reveals the lowest camera drift and reprojection errors at comparable reconstruction times.*



$\mathbf{C}$ $\mathbf{V}$ $\mathbf{S} = (\mathbf{1} - \mathbf{C}) \circ \mathbf{V}$

Figure 4.13: *Cost, variance and sampling matrices for wide baseline candidate picking. The camera circled an object twice. Dark parts of* **C** *indicate regions where good global anchor constraints are likely to be found.* **S** *shows where we sample for global anchor constraints.*

We establish these constraints by importance sampling frame pairs from the set of confident frames and by joining them based on SIFT features and the previously reconstructed window scene structure.

For finding good frame pairs to be used as global anchors as well as for geometrical verification, we propose several camera stability measures.

Camera Stability

The stability of camera poses for being used as global anchor constraints is based on the following measures ζ:

- The number of remaining points attached to a camera pose: $\zeta_1 = n$. This makes sure that there are enough constraints for optimization.
- The distribution of the point projections $(u\ v)$ in the image, specifically the minimum standard deviation σ in vertical and horizontal direction: $\zeta_2 = \min(\sigma(u), \sigma(v))$. This avoids unstable configurations with very localized feature positions.
- The ratio between the smallest and the largest principal component $PCA(\boldsymbol{x})_{\text{min}}$, $PCA(\boldsymbol{x})_{\text{max}}$ of the scene point positions $\boldsymbol{x}$: $\zeta_3 = \frac{PCA(\boldsymbol{x})_{\text{min}}}{PCA(\boldsymbol{x})_{\text{max}}}$. This avoids using two-dimensional scenes which tend to be ambiguous (e.g. camera in plane) or unstable (e.g. frontoparallel plane).

We will use these measures to identify specifically unstable camera pairs that would benefit from a global anchor and for verifying global anchors candidates geometrically.

Anchor Selection

To find good anchor candidates for wide baseline links we use two measures: The cost for matching two frames, and the uncertainty of relative camera poses computed during the window BA.

Cost Estimation. For robust estimation of the basic linking cost, we compute frame similarity based on histograms of SIFT features [WSZ+14]. The output is a cost matrix **C** representing the cost for matching two frames of a video sequence (see Figure 4.13).

Uncertainty Estimation. For uncertainty estimation, we approximate the variance of the camera poses for every pair of confident camera poses in the following three steps:

1. Estimate the variance of each camera's pose c_i relative to each window's structure, i.e., the 3D points computed from camera poses in window w_j:

$$Var(c_i, w_j) \propto 1/(\min(25, \zeta_1) \cdot \zeta_2 \cdot \zeta_3)^2 \tag{4.7}$$

   We assume that 25 reprojections are sufficiently many constraints.

2. Use this information to estimate the variance between windows by averaging the summed variances to common camera poses:

$$Var(w_{j_1}, w_{j_2}) = \sum_{i=1}^{n} \frac{Var(c_i, w_{j_1}) + Var(c_i, w_{j_2})}{n^2} \tag{4.8}$$

3. Find the camera pose→window→...→window→camera pose path with the lowest summed variance for each camera pose pair. While step 2 only considers variances for windows that share a camera pose, this step propagates the variance information to arbitrary indirectly connected camera pose pairs.

This results in a variance matrix **V** (see Figure 4.13). We can now estimate a matrix **S** representing potential anchor frames to be used as global links:

$$\mathbf{S} = (\mathbf{1} - \mathbf{C}) \circ \mathbf{V} \tag{4.9}$$

Note that $\mathbf{C}_{ij} \in (0, 1)$ and $\circ$ is the element-wise product of matrices. We importance sample **S** to get frame pairs (f_1, f_2) that represent useful anchor constraints.

Geometrical Verification

To ensure that a global anchor constraint is truly useful, we perform a geometrical verification. We pick the window with the most available scene points that contains f_1 and BA for the pose of f_2's camera based on the points of that window, utilizing SIFT matches for linking f_2's features to f_1's points.

In our experiments we observed that up to 40% of the matches were outliers when SIFT features extracted from KLT keypoints were matched. Therefore, we exploit the already known scene geometry to gain robustness in this process. We apply four passes of BA for the camera pose parameters while removing all the points with reprojection errors worse

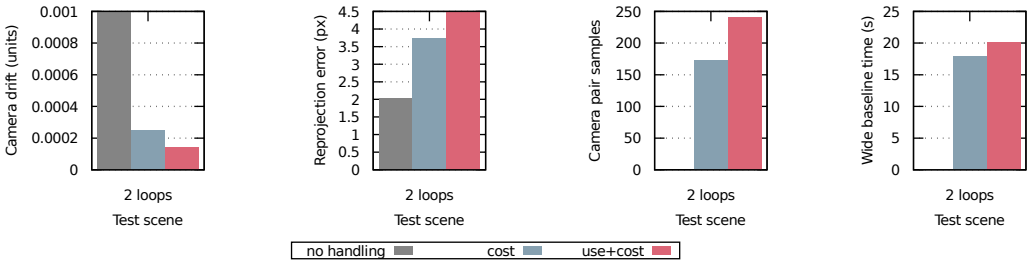


Figure 4.14: *Comparison of loop closing strategies. Beside camera drift and reprojection error, we also show the number of samples needed to get 20 verified wide baseline links and the computation time for wide baseline handling. Without loop closing, the camera drift is quite high (out of scale: 0.08). Cost based frame selection for wide baseline handling reduces the drift drastically (use). Choosing frames also based on their value for wide baseline handling (encoded in* **V***) reduces the drift by another 40% for a fair amount of extra samples/runtime (use+cost). Note that the reprojection error increases because of the extra constraints that have to be fulfilled for closing the loop.*

than the average between the passes. Since BA tends to prefer consistent constraints, inconsistent reprojections are removed by this procedure. If there is not enough consistent data, BA diverges which leads to a violation of our stability constraints.

We consider the geometric verification successful if it passes the following stability thresholds: $\zeta_1 \geq 25$, $\zeta_2 \geq 0.075 \cdot \mathit{ImageSize}$ and $\zeta_3 \geq 0.1$, which worked well in all our experiments. When a pair of frames representing a global anchor constraint fulfills these thresholds, we add the respective relative camera pose constraints to the existing set of constraints which are then used for the initialization of camera poses before the final optimization.

Figure 4.14 illustrates the effect of using the anchor constraints, based on sampling costs **C** and **S**, which additionally takes into account variance matrix **V**. Using anchor constraints considerably reduces camera drift. By concerning **V** in addition to the basic matching cost, drift can be reduced by another 40%.

4.3.6 Final Optimization

The window BA and the global anchors now provide a large set of pose constraints. Using all these constraints we again apply the linear camera pose estimation (Section 4.3.1) globally in order to compute consistent camera poses for all confident frames.

We then apply a series of nonlinear least squares optimization passes based on the following three strategies:

A No Field of View (FoV) optimization, no bad point removal.

B No FoV optimization, bad point removal.

C FoV optimization, bad point removal.

We run the following sequence: `ABABABCCC`. Skipping bad point removal (`A`) at the beginning avoids the removal of reliable points because of a bad initialization, thus loosing valuable information for optimization. FoV optimization is added at the very end only (`C`), because it tends to converge to singularities in small or badly initialized scenes.

When all confident camera poses are optimized and corresponding stable scene points are reconstructed, we initialize the poses of all remaining, non-confident frames by linear interpolation followed by a BA step constrained by the fixed, stable scene points that were obtained based on the confident frames before.

4.4 Evaluation

In the addition to the quantitative evaluations shown throughout the chapter, we provide additional results on captured datasets and on ground truth data. Most captured test scenes (office 1333 frames, bench 1055 frames, stairway 867 frames, office2 1180 frames, swinging 1257 frames, 2loops 1715 frames) were recorded with a GoPro Hero 3 in Wide Angle 1080p 60fps mode. The ship scene (4411 frames) was recorded with a DSLR mounted on a slowly moving crane to simulate high frame rate footage. Detailed information about our captured datasets can be found in the appendix Chapter B.

Our algorithm is implemented completely on CPU. All timings were evaluated based on a system with two Xeon X5660 6 core CPUs. We assume that our algorithm is limited by the system bus bandwidth since occasional tests on single, newer 6 core CPUs such as a Xeon E5-1650 v3 reveal similar timings.

For KLT tracking, we utilized OpenCV's CPU implementation. We noticed large runtime differences between the CPU and the OpenCL version ranging from neglectable runtimes that could be hidden by IO latency to runtimes that made up the major part of processing. Because of this extremely variable behavior of the tracking algorithm that is used as plug-in black box in our method, we partially exclude it from the runtime evaluations.

4.4.1 Ground Truth Comparison

We have constructed a 2MP, 60 fps synthetic ground truth sequence from the Open Movie Project "Sintel" [Roo10] containing rich scenery, motion blur, glare and camera shakes. Our method is more of a SfM approach than SLAM, as it features global BA steps typical to SfM. However, Figure 4.15 shows that it runs orders of magnitude faster than other SfM systems, at comparable speed to current SLAM systems, while producing results which are an order of magnitude more accurate than SLAM systems.

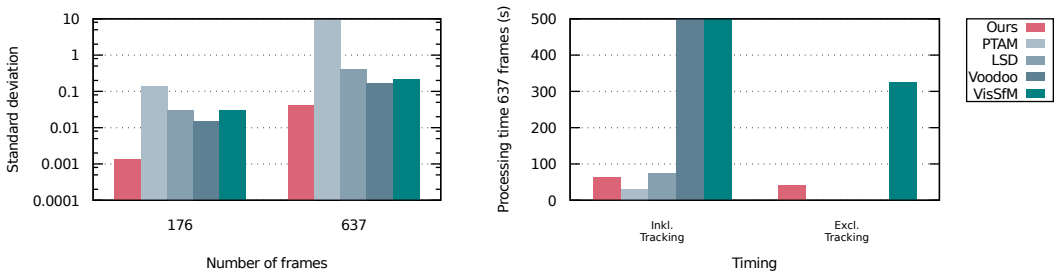


Figure 4.15: *Evaluation based on synthetic ground truth. We give the standard deviation of the reconstructions fitted to the ground truth with an affine transformation plus timings. Our approach runs orders of magnitude faster than other SfM systems while producing results which are an order of magnitude more accurate than SLAM systems. PTAM failed after 176 frames due to too slow map update.*

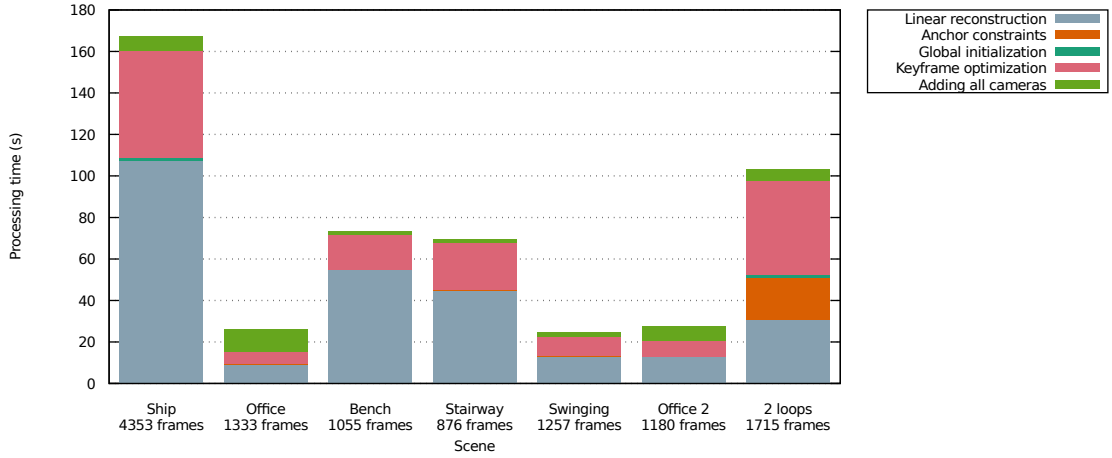


Figure 4.16: *Breakdown of reconstruction timings to the individual pipeline parts.*

4.4.2 Timings

Figure 4.16 breaks down the reconstruction timings for the captured scenes used in this chapter to the individual parts of our pipeline.

Reconstruction timings of our approach are further compared with several other techniques in Figure 4.17 analyzing timings for varying numbers of frames of a FullHD outdoor video sequence. We compare to two approaches designed for handling images (Bundler [SSS06] and VisualSfM [W+11] (GPU accelerated, parallelized)) as well as two approaches for video sequences (Voodoo Camera Tracker and LSD-SLAM [ESC14]). For a fair comparison, we show results with and without feature processing as this step cannot be excluded from all algorithms.

Methods intended for sparse, unstructured data suffer from n^2 runtime for searching corresponding images. The Voodoo Camera Tracker performs well for small tracks but

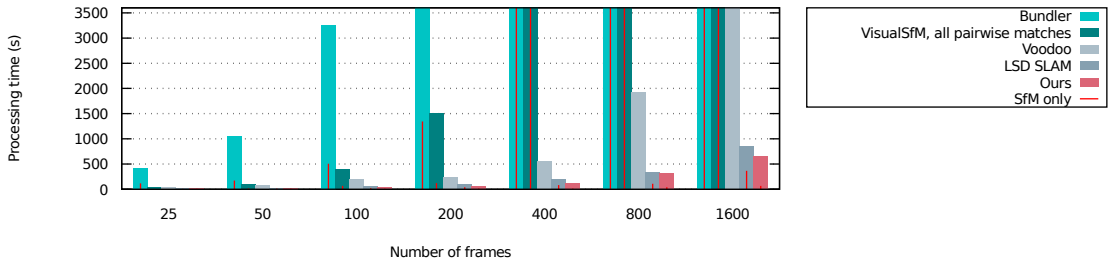


Figure 4.17: *Computation time comparison for a FullHD (1080p) image sequence. Our technique exhibits the lowest computation time of all tested approaches. Note that if only the SfM part without feature processing is considered, we are about 6x faster than the nearest competitor. For 400, 800 and 1600 frames, we did not obtain reconstruction timings from all methods. Separate timings for IO/features and SfM reconstruction could not be obtained with Voodoo.*

becomes much slower when BA has to correct accumulated drift in the end. Even in comparison to recent efficient SLAM approaches such as LSD SLAM our method is faster. We also observed that increased image resolution can lead to significant drops in performance for the tested competitors, whereas our method scales well due to the proposed subsampling. We expect further significant speed gains by improved preprocessing such as GPU-based feature detection and tracking, as the majority of computing time is spent on these steps, and not our core optimization procedure (Figure 4.17).

4.4.3 Stanford Lightfields

We reconstructed some of the Stanford Light Field Archive datasets by treating the individual 10 megapixel raw images in zigzag order as video stream. Each dataset contains an array of 17x17=289 camera poses. For timings refer to Figure 4.18. An exemplary reconstruction visualization of the Lego bulldozer scene can be found in Figure 4.19.

4.4.4 5k High Resolution Video

We reconstructed a scene from very high resolution video with 5120x2700 pixels and 901 frames. Timings and a visualization of the reconstruction can be found in Figure 4.20.

Note that the major bottleneck in this reconstruction was the triangulation and subsampling of points.

4.4.5 Cooperative Room Capturing

We reconstructed an office room from video footage with 14254 frames that consists of several video sequences recorded with three different cameras. We used a GoPro Hero3,

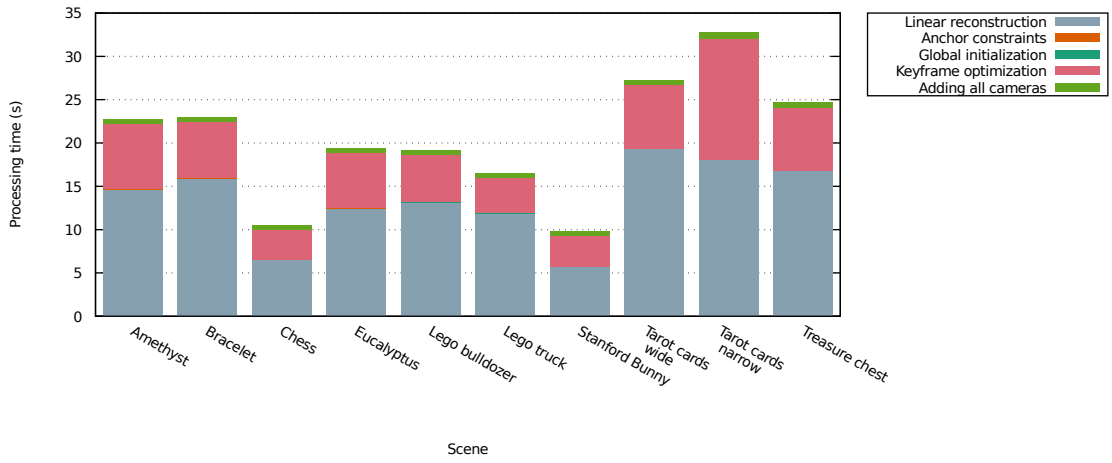


Figure 4.18: *Timings for the stanford dataset reconstruction. Each scene contains 289 10MPixel images.*

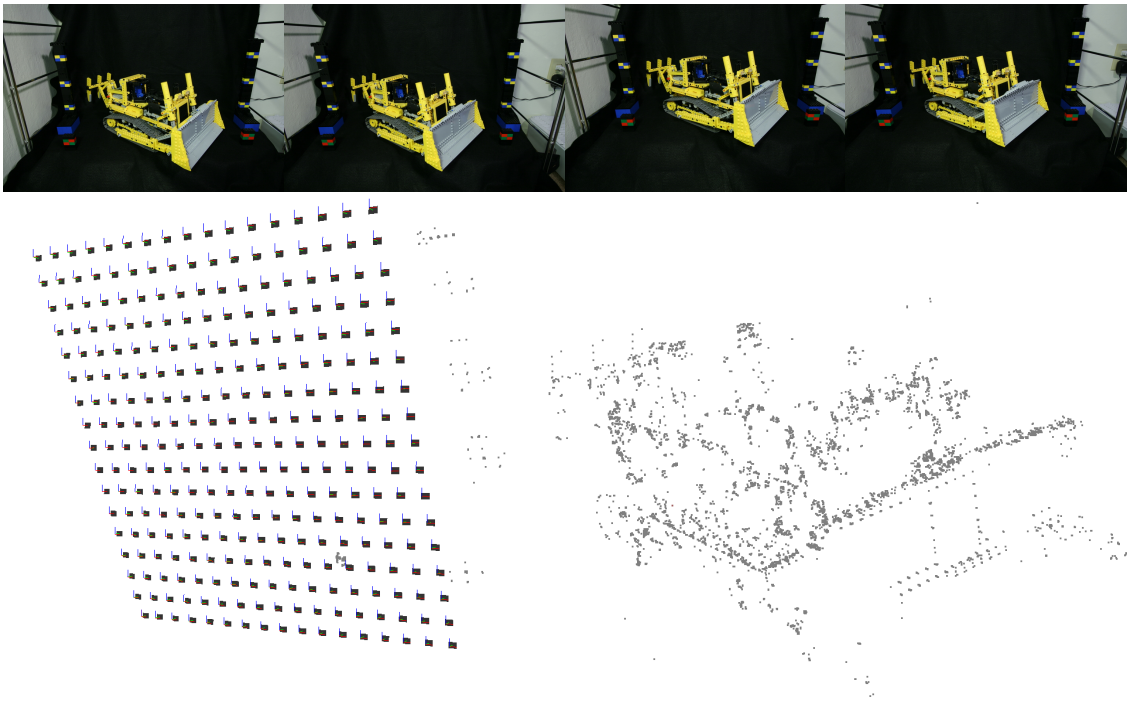


Figure 4.19: *Reconstruction of the Stanford Light Field Archive Lego bulldozer scene.*

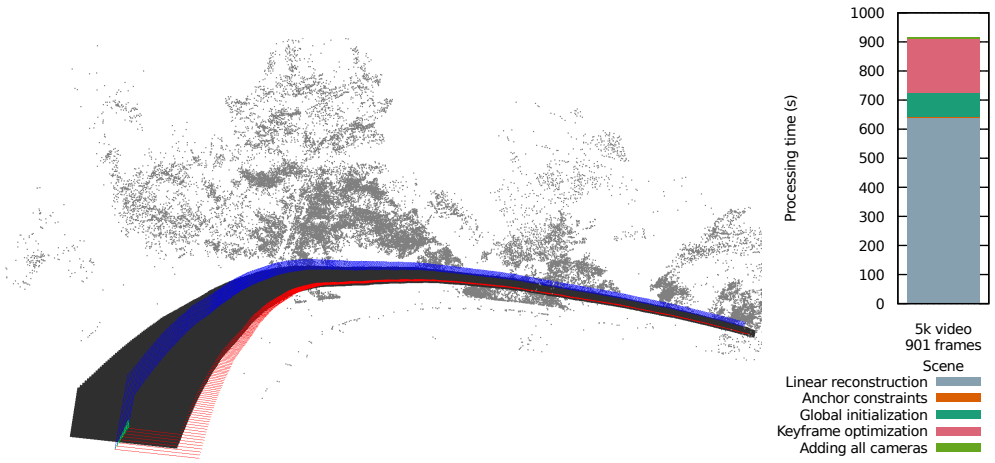


Figure 4.20: *Reconstruction visualization and timings and reconstruction from a very high resolution (5k) video. Most time is spend on point triangulation and subsampling during window BA and the final BA.*

a budged consumer Canon S100 photo camera and a Sony camcorder and recorded all video sequences in FullHD with frame rates between 25 and 60 Hz.

Example images, a visualization of the reconstructed scene and camera tracks as well as timing information can be found in Figure 4.21. Refer to Figure 4.22 for an illustration of the effect of different frame rates on the camera tracks.

4.5 Summary

We introduced a novel pipeline that enables efficient computation of extrinsic camera poses and scene structure on high spatiotemporal resolution, i.e. high resolution, high frame rate video sequences. One of the key insights in this work is that the coherence of such data enables the use of modified tracking, subsampling, and global optimization schemes, which in combination allow for considerably faster and more robust computation, similar to observations made in previous works [YG14, KZP+13] in the context of dense surface reconstruction.

In particular we found that common choices in SfM such as n-point algorithms for initialization are problematic in this context and can be entirely replaced by BA-based approaches.

A large part of this work is however spent on making BA more efficient: We start by optimizing the input data for better and faster convergence by extending KLT to drift free tracking. An important finding throughout the whole chapter is that, similar to [SSS06] and [ASSS10], stability in BA can be achieved with only a small subset of the data. This manifests in our method of consequent partitioning and subsampling: We use a sliding window approach based on only few important keyframes for processing the video stream

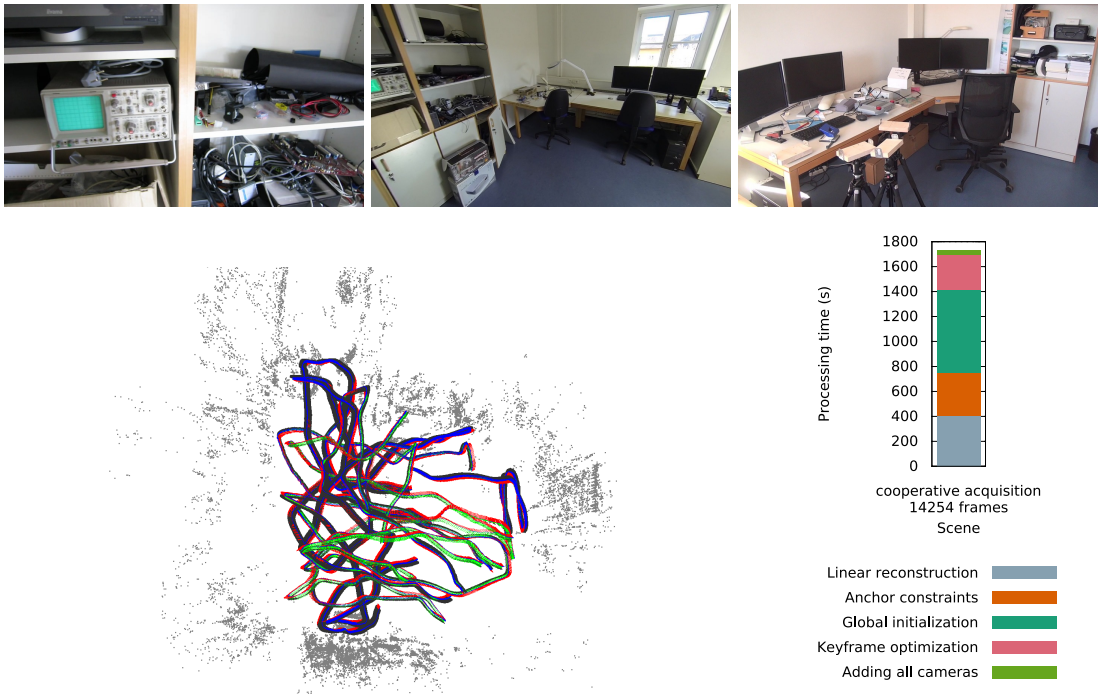


Figure 4.21: *Example images, reconstruction visualization and timings from cooperatively captured video sequence set containing 14254 frames from three different cameras. Examples images from left to right: Canon S100, GoPro Hero3, Sony camcorder. All reconstructed camera trajectories are linked into a common global context by the anchor constraints.*

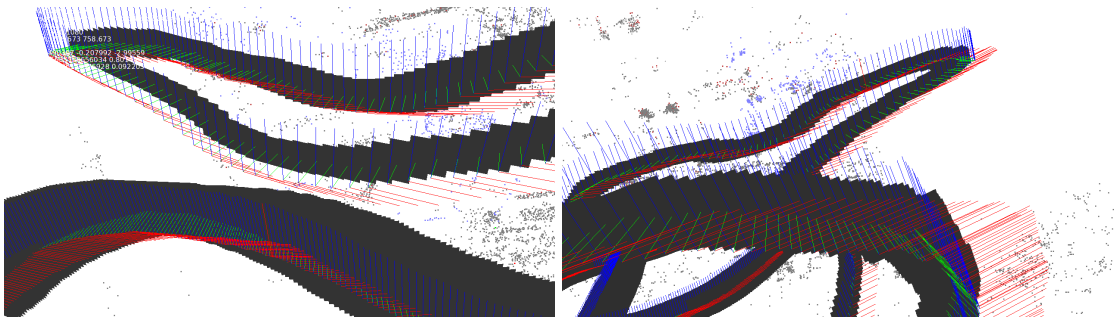


Figure 4.22: *Effect of different frame rates on camera trajectories. Left: Canon S100 (25 fps) vs. Gopro Hero 3 (60 fps). Right: Canon S100 (25 fps) vs. Sony camcorder (50 fps).*

and reduce the number of 3D scene points for optimization to a small but robust set. Even for loop closing we only use a small set of frame pairs sampled based on a cost/benefit evaluation. To integrate all information from our piecewise, subsampled reconstruction and loop closing in a unified way, we employ a fast, linear method for global camera pose registration.

Given the constant increase of camera resolution and frame rate, and the advent of light field sensors by companies such as Lytro or Pelican Imaging, we believe that algorithms specifically designed for densely sampled input represent a great opportunity for future research in this area.

4.5.1 Limitations and Future Work

Our method currently focuses on computing extrinsic camera parameters. As future work it would be interesting to support uncalibrated cameras with changing intrinsics.

Since high framerate cameras are easier to realize based on CMOS sensor technology, they usually exhibit rolling shutter. Our framework could be extended to model this effect, e.g. by modelling the intra-frame camera trajectory by using polynomials or splines [OFKS13].

Chapter 5

Semi-Dense, Direct Visual Odometry for Flexible Multi-Camera Rigs

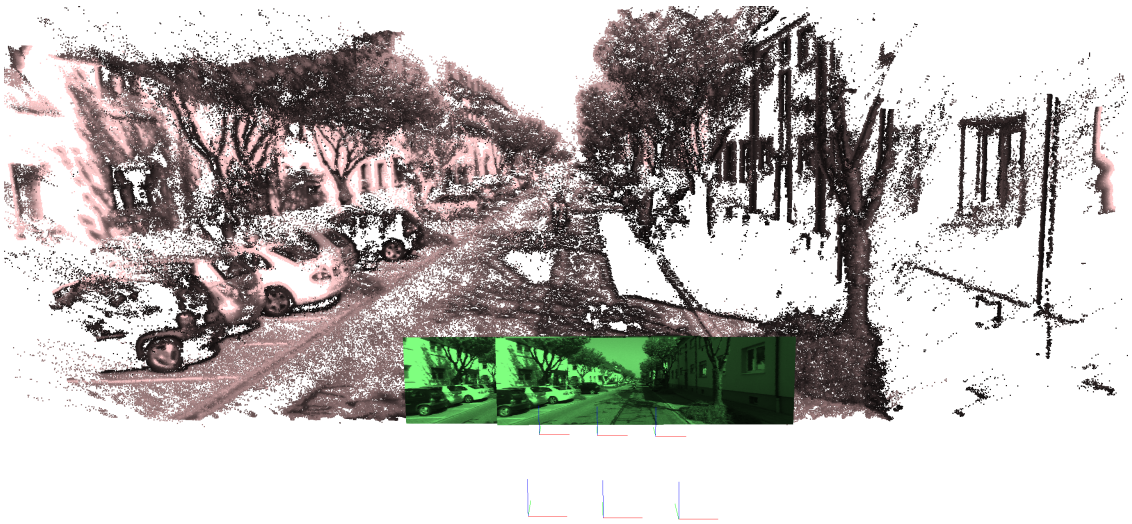


Figure 5.1: *Reconstruction from a KITTI odometry benchmark dataset scene with our dense algorithm for multi-camera rigs.*

In contrast to the previous chapters which handle improvements on sparse reconstructions (compare to Section 2.4), we now focus on improving a dense algorithm (see Section 2.5).

We present a direct Visual Odometry (VO) algorithm for multi-camera rigs, that allows for flexible connections between cameras and runs in real-time at high frame rate on GPU for stereo setups. By using a multi-camera setup we can introduce an absolute scale into our reconstruction. Multiple views also allow us to obtain depth from multiple disparity sources: temporal disparity by exploiting rig motion and static disparity between the different cameras of the rig, which allows for handling dynamic scene parts. We propose a joint optimization of the rig poses and the camera poses within the rig which enables working with flexible rigs. We show that sub-pixel rigidity is difficult to manufacture for $\geq$720p cameras which makes this feature important, particularly in current and future

(semi-)autonomous cars or drones. Consequently, we evaluate our approach on own, real-world and synthetic datasets that exhibit flexibility in the rig beside sequences from the established KITTI dataset.

5.1 Introduction

Visual Odometry (VO) and Self Localization And Mapping (SLAM) systems traditionally are feature-based approaches: Distinct features are tracked over many frames. The scene reconstruction, i.e. determining for all frames the parameters of the camera poses and the positions of all scene points, is usually formulated as a least squares optimization problem that aims at minimizing the difference between the observed feature positions in the images and the reprojections of the scene points into the frames (see Section 2.4).

Direct methods (see Section 2.5) instead maintain per pixel depth information for several keyframes which allows the keyframe's image to be projected into other frames, given the camera's intrinsic parameters and its relative pose to the keyframe. The camera poses of new frames can be found by minimizing the photometric error between the keyframe projected into the new frame and the new frame itself. When the relative camera transformation between the keyframe and another frame is known, depth information of the keyframes is improved with stereo depth estimation techniques.

With a stereo camera rig, the quality of the depth maps of the keyframes can be improved by applying not only the *dynamic* stereo between a keyframe and its subsequent frames, but also by applying *static* stereo between two images that were recorded by the rig at the same time with known rig intrinsics (poses of the cameras relative to the rig, compare to Section 2.2) [ESC15].

It turns out that direct VO methods are much less forgiving to bad calibrations as feature-based VO methods. Feature-based methods optimize based on distances on the image which increase gradually when the calibration quality decreases. Direct methods optimize based on photoconsistency in a small window which can be arbitrarily bad if the calibration is just one pixel off - except when low-pass filtered images are used at the cost of loosing high frequency content.

While the intrinsic parameters of cameras can usually be controlled with pixel precision and the tracking of the rig's pose works just as precise, the rig intrinsics can't be assumed to be static enough for sub pixel precision over time, especially for higher resolution cameras, large baselines and/or material/cost/weight limitations of the rig. Deflection theory shows that two 0.5 kg cameras mounted on a 300x25x4 mm stainless steel carrier are rotated by 0.56° against each other when an acceleration of ± 1 g is applied (see appendix Chapter C). This already introduces significant errors if the rig flexibility is not concerned (see Figure 5.2).

In this chapter, we propose a method that extends existing, direct monocular or stereo VO approaches with per frame rig intrinsics tracking for multi-camera rigs. We target on high-frequency rotations of the cameras in the rig up to a few degrees which are hard

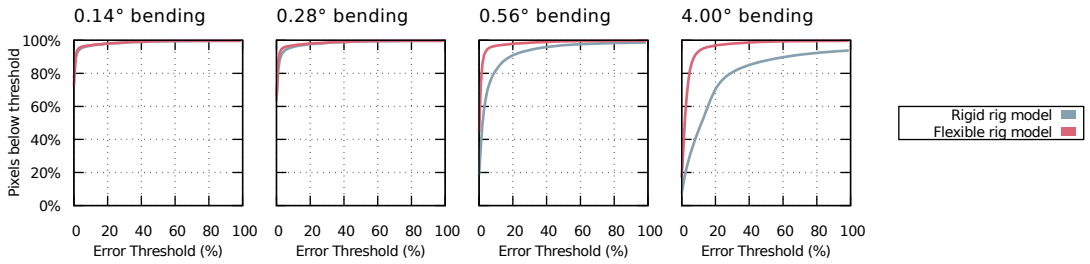


Figure 5.2: *ROC evaluation of the depth quality on a synthetic dataset (720p, 90° FoV camera). Error thresholds are given relative to the pixels' depths. Note that even for small camera rotations inside the rig of 0.56°, taking the rig flexibility into account improves the results very significantly: For a 3% threshold, the recall is 49.4% (rigid model) vs. 85.9% (flexible model).*

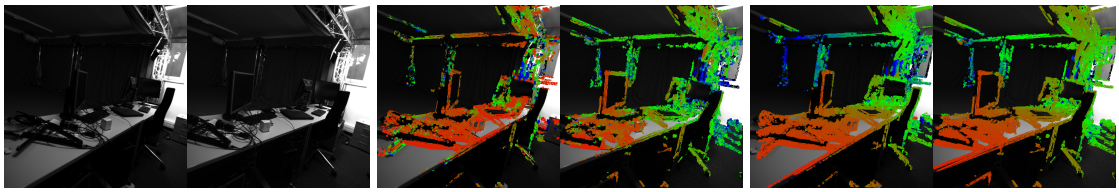


Figure 5.3: *Comparison of the depth maps reconstructed without (center) and with (right) flexible rig optimization. The original frames are shown on the left. Without flexible tracking, tracked frames cannot be aligned consistently to the last keyframe. This leads to a bad alignment of projected depths onto image gradients and therefore to stereo errors which result in incomplete and noisy depth maps (center), while flexible tracking does not suffer from those issues (right).*

to avoid but harmful to conventional direct VO methods (see Figures 5.2 and 5.3). For each keyframe, we maintain per pixel depth information for the images of all cameras. For every new frame, we optimize for the rig's pose as well as for its intrinsics based on the photometric error between every image of the keyframe and every new image. This way, we find consistent poses for all cameras of the rig by utilizing the information that is available in all images.

In fact, we can even choose (1) the pairs of keyframe / new frame images that should be used for tracking and (2) the new frames' or keyframes' images that should be used to improve the depth information of the keyframe images. This flexibility can be used to balance between computation speed and reconstruction quality as well as to adjust the processing to the rig configuration, e.g. avoid direct interaction of cameras that never see the same field of view (FoV).

We implemented most of our algorithm on the GPU using CUDA to address the increased computational demands compared to a monocular setup.

Please note that building a globally consistent reconstruction of a scene including loop closing and global error distribution is out of the scope of this document. However, our technique can be used as a plugin replacement for the VO part of other SLAM systems.

5.2 Related Work

The proposed approach reconstructs camera poses as well as the scene structure, so we consider VO / SLAM methods as well as Structure from Motion (SfM) and Multi View Stereo (MVS) techniques in this section.

5.2.1 Visual Odometry and SLAM

Most VO and SLAM methods are feature-based. Chiuso et al. [CFJS02] presented one of the first real time capable, monocular, feature-based SLAM systems. Nistér et al. [NNB04] established the term ”Visual Odometry” by presenting a feature tracking based system for frame-to-frame monocular or stereo camera pose estimation. Davison et al. [DRMS07] proposed MonoSLAM which is an Extended Kalman Filter (EKF) based method that creates a probabilistic but persistent map of natural landmarks and is therefore drift free in small workspaces. PTAM [KM07] is designed to avoid the iterative tracking and mapping per frame by separation into two concurrent threads: The mapping thread integrates all previously tracked camera poses and features into a consistent representation in the background while the tracking thread tracks every new frame against the newest available map. Paz et al. [PPTN08] combine an EKF-based framework with a stereo camera setup.

Direct methods for VO and SLAM are available for some years now and have two major benefits over the feature-based methods: First, there is no need to craft a feature detector. Second, not only the information from sparse feature points but virtually any gradient information in the images can be used for reconstruction. Direct monocular methods first appeared in the RGBD domain [KSC13, MC13] where the per pixel depth information comes directly from the sensor and only the tracking has to be performed. The first direct real-time method relying solely on color images is LSD-SLAM [ESC14] which is outlined in Section 2.5. LSD-SLAM was extended to stereo cameras [ESC15] where tracking only happens on the left camera. This is in contrast to our approach which allows for tracking between multiple cameras. Consequently, in [ESC15], a depth map is maintained only for the left camera; the right images are just used for static stereo with the corresponding left images which improves the keyframe’s left depth map and introduces an absolute scale to the reconstruction. This is however not feasible if the rig intrinsics change over time. In addition, we show that we can reconstruct datasets from rigs with few overlap between the fields of view of the cameras by utilizing the information from all cameras where methods like [ESC15] fail (see Section 5.4.5).

Pillai et al. [PRL16] accelerates the depth estimation for stereo pairs by starting from few sparse, Delaunay triangulated piecewise planar surfaces which can be refined in multiple iterations in areas where the matching cost is high, e.g. due to non-planarity. A substantially different approach is the method of Comport et al. [CMR07] which avoids the explicit reconstruction of scene depth and uses the quadrifocal constraints from two pairs of stereo images to obtain the transformation of the camera rig instead.

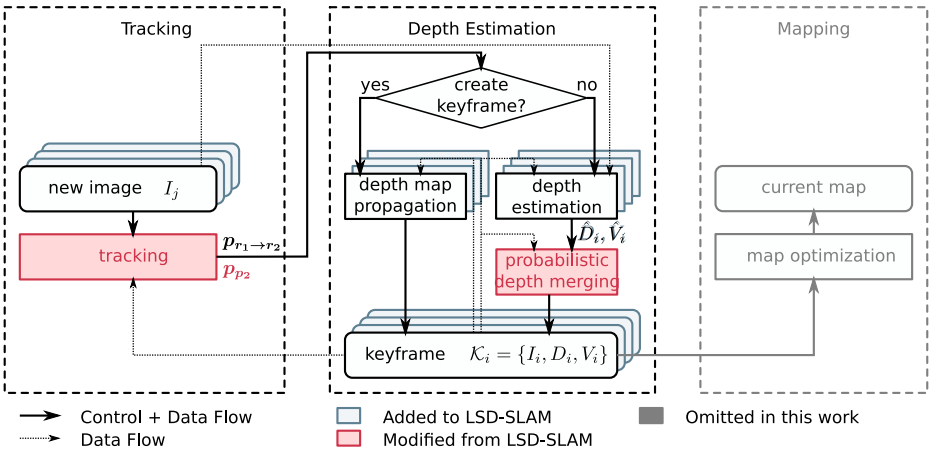


Figure 5.4: *Overview over the LSD-SLAM algorithm and our extensions.*

5.2.2 Structure from Motion and Multi-View Stereo

SfM (usually feature based) with subsequent MVS (dense, direct surface reconstruction) produces output similar to dense SLAM methods since both reconstruct consistent, dense 3D models and camera poses from images. However, SfM+MVS methods traditionally aim more at image collections instead of videos and reconstruction quality instead of real time processing with low latency. Recently, there have been efforts to develop SfM / MVS methods that utilize the redundancy of video streams to achieve close-to-real-time performance. As shown in Chapter 4, we have accelerated SfM by the consequent subsampling of frames, features, scene points and loop closure candidates in a Bundle Adjustment framework and employing a linear solver to keep unsampled frames consistent. The MVS methods in [KZP+13, WRL16] compute depth maps based on the photoconsistency with up to 100 other images but compare for photoconsistency based on 1x1 pixel masks only.

Delaunoy and Pollefeys propose a Photometric Bundle Adjustment [DP14] which is designed to recover camera pose parameters and a dense surface mesh based on the photometric error between observed and model-based, generated images.

5.3 Multi-View VO for Flexible Camera Rigs

In this section we'd like to introduce our method. Figure 5.4 shows an overview of the originally monocular LSD-SLAM algorithm [ESC14] and our extensions.

LSD-SLAM tracks the camera poses of new images against a previous, depth enhanced keyframe. If the current keyframe is still good for tracking (content similar to the new frame), new depth information is estimated with a stereo method between the new image and the keyframe and is merged into the keyframe. If the keyframe is not good

enough anymore, it gets propagated to the current image. A mapping component finally cares about global consistency of all keyframes. For a more detailed description of LSD-SLAM, please refer to Section 2.5.

Our Method allows for multiple input images from multiple cameras per frame and stores depths for all cameras of a keyframe, consequently. We extended the tracking to optimize for the rig pose and the rig intrinsics (camera poses within the rig) jointly (Section 5.3.1). For stereo cameras, we propose a reliable rig parameterization in Section 5.3.2. Tracking gives us the transformation from every keyframe image to every new frame's image, so we can generate multiple stereo observations per keyframe (Section 5.3.3). In order to utilize all depth information, we extended the Bayesian depth merging approach of LSD-SLAM (Section 5.3.4). On top of the reconstruction of static scene parts, we can utilize the tracked transformations between every new frame's images (which are captured at the same time) to obtain depth for dynamically moving objects (Section 5.3.6).

5.3.1 Tracking Flexible Multi-Camera Rigs

The following sections describe our contributions for enabling multi-camera, flexible rig VO, starting with the tracking part where we optimize for the rig pose and the rig intrinsics concurrently:

For multi-camera rigs, if the ith frame is a keyframe $\mathcal{K}_i^l = \{I_i^l, D_i^l, V_i^l\}$, it contains image intensities I, inverse depths D and inverse depth's variances V for every camera l of the rig. For tracking, we extend the image and view space transformation functions from Section 2.2.2 by a rig space transformation function and its inverse, both depending on a set of rig pose parameters $\boldsymbol{p_r}$:

$$\boldsymbol{\varsigma}\colon \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_r}|} \to \mathbb{R}^3, \qquad (\boldsymbol{x}; \boldsymbol{p_r}) \mapsto \boldsymbol{\varsigma}(\boldsymbol{x}; \boldsymbol{p_r}) = \boldsymbol{x_r} \tag{5.1}$$

$$\boldsymbol{\varsigma^{-1}}\colon \mathbb{R}^3 \times \mathbb{R}^{|\boldsymbol{p_r}|} \to \mathbb{R}^3, \qquad (\boldsymbol{x_r}; \boldsymbol{p_r}) \mapsto \boldsymbol{\varsigma^{-1}}(\boldsymbol{x_r}; \boldsymbol{p_r}) = \boldsymbol{x} \tag{5.2}$$

This allows us to define a warping function of monoscopic LSD-SLAM (Equation 2.67 to a rig based environment:

$$\begin{aligned} & \boldsymbol{\omega}(u, v, d; \boldsymbol{p_{r_1 \to r_2}}, \boldsymbol{p_{p_1}}, \boldsymbol{p_{p_2}}) \\ = \; & [\boldsymbol{\pi}(\boldsymbol{p_{c_2}}) \circ \boldsymbol{\varphi}(\boldsymbol{p_{p_2}}) \circ \boldsymbol{\varsigma_{1\to 2}}(\boldsymbol{p_{r_1 \to r_2}}) \circ \boldsymbol{\varphi^{-1}}(\boldsymbol{p_{p_1}}) \circ \boldsymbol{\pi^{-1}}(\boldsymbol{p_{c_1}})](u\ v\ d) \end{aligned} \tag{5.3}$$

$$\boldsymbol{\varsigma_{1\to 2}}(\boldsymbol{p_{r_1 \to r_2}}) \cong \boldsymbol{\varsigma}(\boldsymbol{p_{r_2}}) \circ \boldsymbol{\varsigma^{-1}}(\boldsymbol{p_{r_1}})$$

This warping function respects parameters for the rig pose $\boldsymbol{p_r}$ as well as parameters for the camera poses $\boldsymbol{p_p}$ inside the rig (see Figure 5.5) and allows us to expand the photometric error of the original LSD-SLAM algorithm to use all available information from all

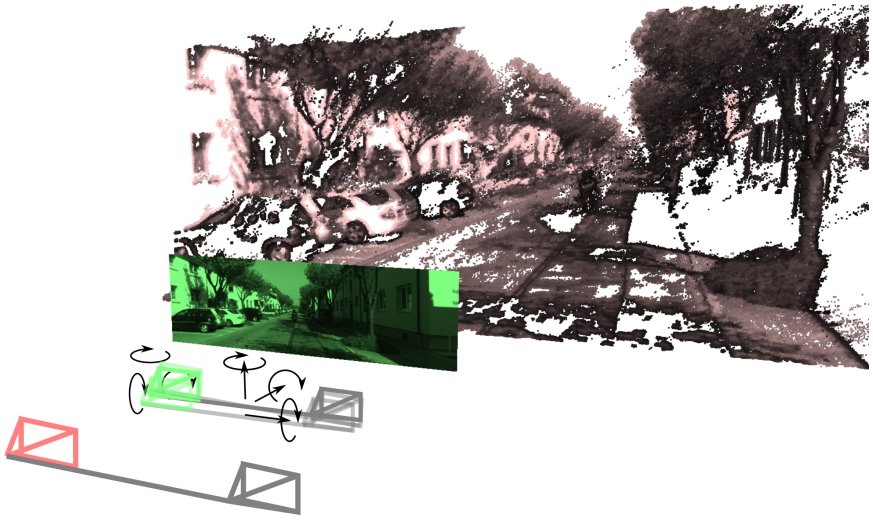


Figure 5.5: *Tracking camera rigs with rig intrinsics. Tracking aims on finding a rig pose and rig intrinsics that align each camera of the new frame (green) so that it matches the reprojections from the last keyframe's (red) data.*

frames:

$$E_r(\boldsymbol{p}_{\boldsymbol{r_i}\to\boldsymbol{r_j}}, \boldsymbol{p}_{\boldsymbol{p_i}}) \tag{5.4}$$
$$= \sum_{c_i}\sum_{c_j}\left(f_t(c_i,c_j)\sum_{(u\ v)}(I_i^{c_i}(u,v) - I_j^{c_j}(\boldsymbol{\omega}(u,v,D_i^{c_i}(u,v);\boldsymbol{p}_{\boldsymbol{r_i}\to\boldsymbol{r_j}},\boldsymbol{p}_{\boldsymbol{p_i}},\boldsymbol{p}_{\boldsymbol{p_j}})))^2\right)$$

c_i and c_j are iterating over the individual cameras of the keyframe's or the tracked frame's rig. The mapping $f_t : \mathbb{N}^2 \to \{0,1\}$ is user defined and can be used to disable pairs of cameras for tracking, e.g. if their fields of view do not overlap at all or for performance reasons (see also Figure 5.9).

Rig Parameterization

Note that our implementation features only one set of rig intrinsic parameters $\boldsymbol{p}_{\boldsymbol{p}}$ (camera pose parameters inside the rig) per frame which is mapped by $\boldsymbol{\varphi}$ to a different rig-to-view-space transformation for each camera in the rig. Consequently, for introducing a new rig configuration to our system, it is sufficient to specify a parameter set $\boldsymbol{p}_{\boldsymbol{p}}$ and a mapping $\boldsymbol{\varphi}$ from the rig intrinsic parameters $\boldsymbol{p}_{\boldsymbol{p}}$ to a rig-to-view-space transformation for each camera as described in the next section.

For solving the minimization problem in Equation 5.4 with the LM algorithm, we have to solve the augmented normal equations as described in Section 2.3.2

$$(\boldsymbol{J}^T\boldsymbol{J} + \mu\boldsymbol{I})\boldsymbol{\delta}_{\boldsymbol{p_r p_p}} = \boldsymbol{J}^T\boldsymbol{\epsilon} \tag{5.5}$$
$$\boldsymbol{J} = (\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{r1}}}, ..., \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{r6}}}, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{p1}}}, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{p2}}}, ...)$$

for $\boldsymbol{\delta}_{\boldsymbol{p_r p_p}} = (\boldsymbol{\delta}_{\boldsymbol{p_r}}, \boldsymbol{\delta}_{\boldsymbol{p_p}})$ to improve our parameters iteratively with $\boldsymbol{p_r^{n+1}} = \boldsymbol{p_r^n} + \boldsymbol{\delta}_{\boldsymbol{p_r}}$ and $\boldsymbol{p_p^{n+1}} = \boldsymbol{p_p^n} + \boldsymbol{\delta}_{\boldsymbol{p_p}}$. $\boldsymbol{J}$ is the Jacobian matrix, containing the derivatives of the error function to all rig pose and rig intrinsics parameters. Note that each $\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial ...}}$ is a column vector with one element for each $()^2$ term of Equation 5.4. We know that the pose of one camera in world space only has 6 degrees of freedom (DoF), so we prefer to evaluate the derivatives for them only.

To achieve this, we replace the rig based error residual by the single camera residual

$$E_r(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}}) = E_c(\boldsymbol{T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})), \tag{5.6}$$

employing a transformation function $\boldsymbol{T}$ that maps rig based parameters to camera pose parameters. To obtain the derivative of E_r, we can according to the chain rule also compute:

$$\frac{\boldsymbol{\partial E_r}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})}{\boldsymbol{\partial}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} = \frac{\boldsymbol{\partial E_c}(\boldsymbol{T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}}))}{\boldsymbol{\partial T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} \frac{\boldsymbol{\partial T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})}{\boldsymbol{\partial}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} \tag{5.7}$$

$$\begin{aligned}
\frac{\boldsymbol{\partial E_r}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})}{\boldsymbol{\partial}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} &\cong \boldsymbol{J} = (\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{r1}}}, ..., \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{r6}}}, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{p1}}}, \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{p2}}}, ...) \\
\frac{\boldsymbol{\partial E_c}(\boldsymbol{T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}}))}{\boldsymbol{\partial T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} &\cong \boldsymbol{J_W} = (\frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{w1}}}, ..., \frac{\boldsymbol{\partial E}}{\boldsymbol{\partial p_{w6}}}) \\
\frac{\boldsymbol{\partial T}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})}{\boldsymbol{\partial}(\boldsymbol{p_{r_i \to r_j}}, \boldsymbol{p_{p_i}})} &\cong \boldsymbol{J_T} = (\frac{\boldsymbol{\partial T}}{\boldsymbol{\partial p_{r1}}}, ..., \frac{\boldsymbol{\partial T}}{\boldsymbol{\partial p_{r6}}}, \frac{\boldsymbol{\partial T}}{\boldsymbol{\partial p_{p1}}}, \frac{\boldsymbol{\partial T}}{\boldsymbol{\partial p_{p2}}}, ...)
\end{aligned}$$

$\boldsymbol{J_W}$ contains the per pixel derivatives to the world space degrees of freedom. $\boldsymbol{J_T}$ consists of column vectors with 6 elements, one for each world space degree of freedom and can easily be obtained analytically or with finite differences.

Besides having to evaluate the minimum number of derivatives only, this substitution makes the GPU code which calculates the per pixel components of $\boldsymbol{J_W}$ - instead of a rig dependent $\boldsymbol{J}$ - independent from the rig parameterization and allows for implementing new rigs with different camera configurations easily at one place.

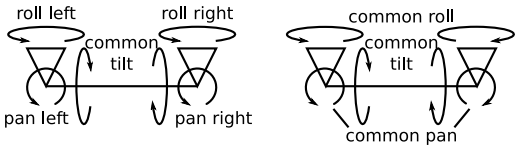


Figure 5.6: *Comparison of symmetric stereo rig models. The left model exhibits the theoretical minimum number of DoFs for fixed baselines. The right model is the one that we finally used due to its clearly distinct DoFs.*

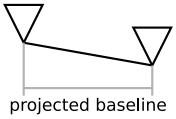


Figure 5.7: *Degeneration of the 5 DoF rig model. Cameras may shift in front of each other to reduce the projected baseline of the rig, overriding the scale fixation that it should actually introduce.*

5.3.2 A Flexible Stereo Rig Model

Generally, a stereo rig model has 6 DoF: trivially one would use an identity camera-to-rig transformation for one of the cameras in the rig while the other one can be placed freely in the rig space. To avoid scale drifting, one might want to set the distance between the cameras fixed which leaves 5 DoF remaining. We found that this is a valid assumption even for flexible camera rigs because bending a rig by small angles leads to negligible changes in the distance between the cameras (e.g. 0.5° bending $\Rightarrow 10^{-5}\%$ closer).

In most stereo camera rigs, both cameras are attached to the rig in the same way and the rig is mounted or held symmetrically, so we prefer a symmetrical rig model as well to have a more natural mapping from real world effects to parameter space.

For our first experiments we used the rig model shown in Figure 5.6 on the left. It exhibits individual parameters for the cameras' roll and pan, but a common tilt parameter. Using individual tilt parameters would introduce 6 DoF which is above the minimum and introduces an ambiguity since tilting the rig can also be accomplished by tilting both cameras.

We observed that this rig model tends to shear as shown in Figure 5.7 when the scene scale starts to drift instead of fixing the scale. Therefore we switched to the 3 DoF model shown in Figure 5.6 on the right which ensures that the rig's baseline stays orthogonal to the average camera's viewing direction. Although this model is not able to represent asymmetric camera rotations correctly, it provides superior reconstruction quality on exactly such input data.

Depth / Panning Ambiguity

While using the 3 DoF stereo rig model, we observed that the depth tends to get compressed in front of the cameras while the cameras start panning towards each other after

a few hundred frames. Although depth and panning are not really ambiguous (i.e. the photometric error $E_r(\boldsymbol{p}_{\boldsymbol{r_i}\rightarrow\boldsymbol{r_j}}, \boldsymbol{p}_{\boldsymbol{p_i}})$ is always smaller with the correct panning), it seems like the constraints in the images are not strong enough to discern them clearly.

To resolve this issue, we implemented a low frequency panning correction: We assume that the camera rotations relative to the rig are high frequent and on average corresponding to the initial rig parameterization. Therefore, we keep track of the low pass filtered panning pan_{avg} and correct each optimized panning parameter for the difference between the initial and the averaged panning:

$$pan_{avg}^{n+1} = (1-\lambda)pan_{avg}^{n} + \lambda pan^{n+1} \tag{5.8}$$

$$pan_{correction} = pan_{avg}^{n+1} - pan_{init} \tag{5.9}$$

$$pan_{corrected}^{n+1} = pan^{n+1} - pan_{correction} \tag{5.10}$$

This ensures that the panning parameter is unable to drift away for many subsequent frames. We got good results on all of our datasets with $\lambda = 0.1$.

5.3.3 Depth Estimation

We use the depth estimation component of LSD-SLAM as explained in Section 2.5 as it is but apply it potentially once for each possible pair of a keyframe camera pose and the tracked frame's camera pose. We can simply obtain the camera-to-camera transformation

$$\boldsymbol{\varphi_{1\rightarrow 2}}(\boldsymbol{p}_{\boldsymbol{p_1}\rightarrow\boldsymbol{p_2}}) \cong \boldsymbol{\varphi}(\boldsymbol{p}_{\boldsymbol{p_2}}) \circ \boldsymbol{\varsigma_{1\rightarrow 2}}(\boldsymbol{p}_{\boldsymbol{r_1}\rightarrow\boldsymbol{r_2}}) \circ \boldsymbol{\varphi}^{\boldsymbol{-1}}(\boldsymbol{p}_{\boldsymbol{p_1}}) \tag{5.11}$$

that is necessary for stereo depth reconstruction by concatenating the rig intrinsics and rig transformation obtained during tracking.

5.3.4 Probabilistic Depth Merging with Multiple Observations

In this subsection we examine how to merge multiple depth observations that come from stereo with the different images from the tracked frame into the depth and variance maps of *one* of the cameras c of the keyframe. Let's consider one pixel p of the keyframe whose depth is described as a normal distribution $\mathcal{N}_{prior} = \mathcal{N}(D_i^c(p), V_i^c(p))$ and the depth distributions $\mathcal{N}_{c_j \in C} = \mathcal{N}(\hat{D}_i^{cc_j}(p), \hat{V}_i^{cc_j}(p))$ of the corresponding pixels in the stereo observations with the tracked frame's cameras c_j. To perform a Bayes filter update step, we decided to sum up the depth observations from all the cameras of the tracked frame and treat them as one single sensor observation

$$\mathcal{D}_{new} = \sum_{c_j \in C} \mathcal{N}_{c_j}. \tag{5.12}$$

The update step can then be expressed by

$$\mathcal{D}_{post} = \mathcal{N}_{prior} \cdot \mathcal{D}_{new} = \mathcal{N}_{prior} \cdot \sum_{c_j \in C} \mathcal{N}_{c_j}. \tag{5.13}$$

Now $\mathcal{D}_{new}$ and $\mathcal{D}_{post}$ are not normal distributions but we will have to represent them as one to update the keyframe's maps. Therefore we rewrite our formulation to apply the multiplications of the distributions first (the product of two normal distributions is again a normal distribution) and approximate the sum by one single normal distribution, i.e. we do the approximation of our distribution as normal distribution as late as possible:

$$\mathcal{N}_{post} \approx \sum_{c_j \in C} \left(\mathcal{N}_{prior} \cdot \mathcal{N}_{c_j} \right) \tag{5.14}$$

To calculate $\mathcal{N}_{post}$, we have to extend our model of normal distributions by a weight parameter: $\mathcal{N}(d, v, w)$. It was previously omitted because we were dealing with probability distributions whose weights and integrals evaluate to one always. In the sum above, however, we need this weight since we want its terms to contribute differently to the resulting distribution, depending on how much $\mathcal{N}_{prior}$ and $\mathcal{N}_{c_j}$ overlap.

Formulas for computing the product or the merge of normal distributions can be found in [CWPS11]:

Product of normal distributions: $\mathcal{N}(d, v, w) = \mathcal{N}(d_1, v_1, w_1) \cdot \mathcal{N}(d_2, v_2, w_2)$

$$d = \frac{d_1 v_2 + d_2 v_1}{v_1 + v_2} \qquad v = \frac{v_1 v_2}{v_1 + v_2} \qquad w = \frac{\mathcal{N}(d; d_1, v_1, w_1) \cdot \mathcal{N}(d; d_2, v_2, w_2)}{\mathcal{N}(d; d, v, 1)} \tag{5.15}$$

Merge of normal distributions: $\mathcal{N}(d, v, w) \approx \sum_i \mathcal{N}(d_i, v_i, w_i)$

$$w = \sum_i w_i \qquad d = \frac{1}{w} \sum_i w_i d_i \qquad v = \sum_i \frac{w_i}{w} (v_i + (d_i - d)^2) \tag{5.16}$$

Note that our implementation contains a dynamic mapping selection function $f_{m_d}(c_i, c_j) : \mathbb{N}^2 \to \{0, 1\}$ similar to f_t from Section 5.3.1 to allow the user to control which tracked frame cameras c_j are used to update the the maps of keyframe camera c_i (see also Figure 5.9).

Alternative Approach

The approach described above (Equation 5.14) is engineered in a way that makes sure that the a priori distribution is propagated if it is supported by at least one of the cameras.

We illustrate an alternative approach that treats the depth estimations from the tracked frame's cameras as individual observations on the example of a stereo rig. In this case,

the a posteriori distribution can be described by

$$\mathcal{N}_{post} = \mathcal{N}_{prior} \cdot \mathcal{N}_{c_1} \cdot \mathcal{N}_{c_2} \tag{5.17}$$

which allows single wrong observations to extinguish the probably correct $\mathcal{N}_{prior}$. To avoid this, we can model a uniform outlier probability $\mathcal{U}$, which leads to

$$\begin{aligned} \mathcal{N}_{post} &= \mathcal{N}_{prior} \cdot (\mathcal{N}_{c_1} + \mathcal{U}) \cdot (\mathcal{N}_{c_2} + \mathcal{U}) && (5.18) \\ &= \mathcal{N}_{prior}\mathcal{N}_{c_1}\mathcal{N}_{c_2} + \mathcal{N}_{prior}\mathcal{N}_{c_1}\mathcal{U} + \mathcal{N}_{prior}\mathcal{N}_{c_2}\mathcal{U} + \mathcal{N}_{prior}\mathcal{U}^2. && (5.19) \end{aligned}$$

In the particularly interesting case of $\mathcal{N}_{prior} \neq \mathcal{N}_{c_1}$ or $\mathcal{N}_{prior} \neq \mathcal{N}_{c_2}$, the first term becomes small which results in

$$\mathcal{N}_{post} \approx \mathcal{N}_{prior}\mathcal{N}_{c_1}\mathcal{U} + \mathcal{N}_{prior}\mathcal{N}_{c_2}\mathcal{U} + \mathcal{N}_{prior}\mathcal{U}^2. \tag{5.20}$$

When we further assume the outlier probability to be small (which is the case in our experiments), we can approximate

$$\mathcal{N}_{post} \approx \mathcal{N}_{prior}\mathcal{N}_{c_1}\mathcal{U} + \mathcal{N}_{prior}\mathcal{N}_{c_2}\mathcal{U}. \tag{5.21}$$

The probability distribution $\mathcal{N}_{post}$ is normalized, so the scaling factor $\mathcal{U}$ can be omitted which makes the results of this approach similar to the previous one in Equation 5.14.

5.3.5 Depth Map Propagation

We propagate depths and variances from one keyframe to the next within the same camera. When a new keyframe is created, we potentially apply one iteration of static stereo, i.e. update each camera's depth and variance maps with the stereo observations from the other cameras of the same frame. This can be user-controlled with $f_{m_s}(c_i, c_j) : \mathbb{N}^2 \rightarrow \{0, 1\}$.

5.3.6 Dynamic Depth Estimation

LSD-SLAM is originally based on a static scene assumption: the warping function (Equation 5.3) used in the photometric tracking residual (Equation 5.4) only considers camera and rig motion but ignores scene changes. With our multi-camera setup, we can obtain the depth of dynamic objects from static stereo which only considers images that were recorded at the same time, i.e. the images of one frame from different cameras.

Figure 5.8 compares the depth that can be obtained from static stereo with the depth based on all disparity sources: static and dynamic stereo. The static stereo depth map is clearly less dense and less precise, because no information is aggregated over time. However, it exhibits correct depths even for moving objects like the biker. To blend between the high-quality, dynamic+static stereo depth map and the static stereo depth map, LSD-SLAM's per pixel tracking score can be utilized. It represents the per pixel

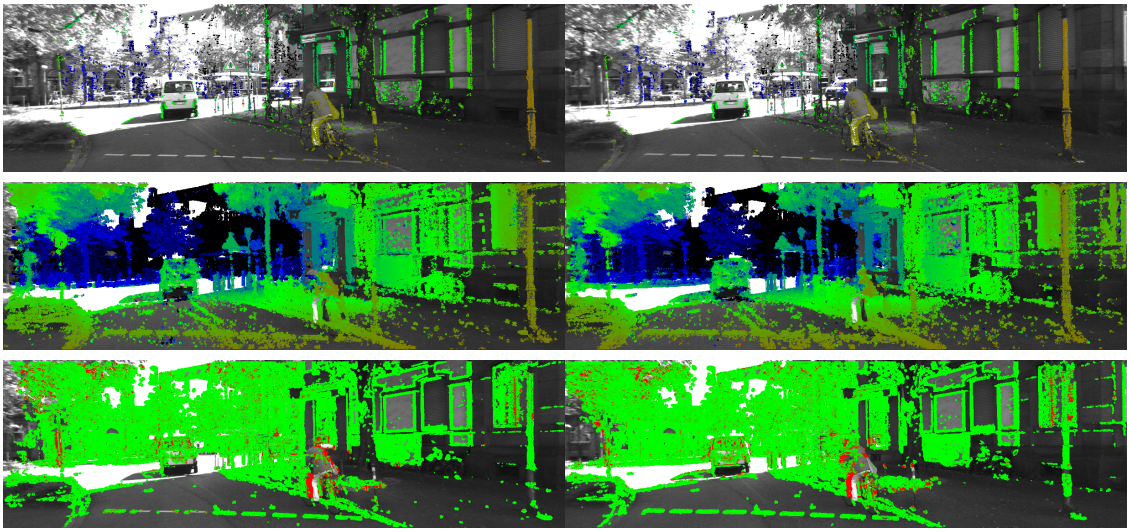


Figure 5.8: *Comparison of static stereo for dynamic objects (top) and joined stereo from all disparity sources (center). Note that depth from static stereo is much more sparse but correct on moving objects. The bottom image shows the tracking score which is low, when the disparities can be explained by camera motion in a static scene. Depths are color coded from red (close) to black (far).*

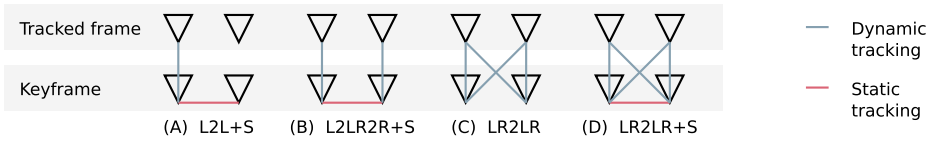


Figure 5.9: *Different camera tracking and mapping configurations. Dynamic tracking is either done within a single stream, within all streams or across streams. Static tracking is performed between two images captured at the same time.*

photometric residual after tracking and is therefore a good measure on how well the image intensities can be explained under a static scene assumption.

5.4 Evaluation

We evaluated different configurations (see Figure 5.9) of our algorithm on various datasets. Please note that this is a frame-to-frame VO algorithm that does not perform full SLAM including loop closing and global optimization and is therefore not directly comparable to such methods.

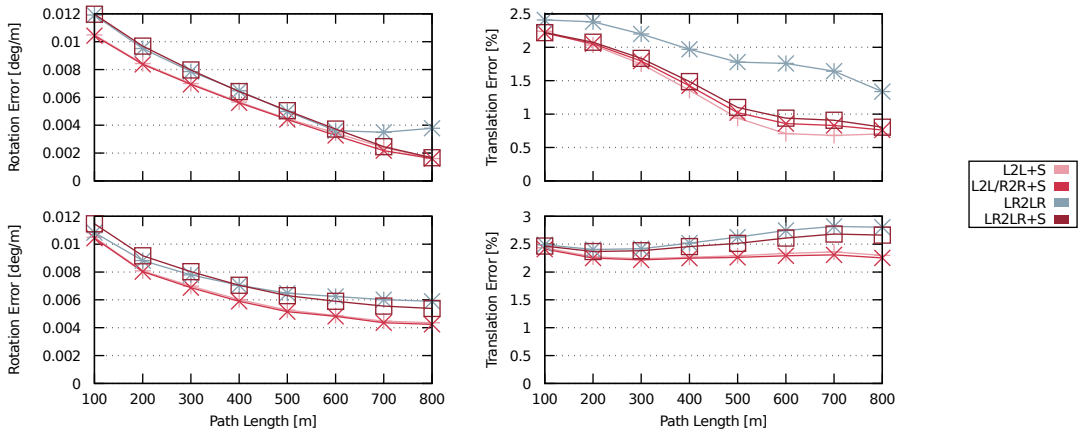


Figure 5.10: *Results from the KITTI odometry benchmark scenes 06 (top) and 08 (bottom). Note that static stereo seems to improve quality significantly. Tracking and depth estimation between different cameras from different frames produces worse results in the scenes because of errors that are introduced by more occlusions on longer baselines.*

5.4.1 KITTI Dataset

Results for two scenes of the KITTI odometry benchmark [GLU12] based on a stereo setup are shown in Figure 5.10 using the configurations displayed in Figure 5.9.

Dynamic tracking and mapping on the left camera only is analog to [ESC15] and produces similar results. Worse results with other configurations of our algorithm, e.g. LR2LR+S, have also been observed by [ESC15] and seem to be related to more outliers due to obstructions for large baselines between different cameras. However, those other configurations are crucial for flexible camera rigs (Section 5.4.2) or rigs with barely overlapping FoVs (Section 5.4.5).

We have also reconstructed odometry by using all four cameras (2x color, 2x grayscale) from the KITTI dataset, comparing it to the results from two cameras (grayscale) in Figure 5.11. The results from four cameras are better than from two cameras for short paths which can be explained by short-term, local scene dependent effects like better handling of occlusions. Over long distances, results with four cameras are worse which might be caused by calibration issues or a worse resolution of the additionally used color cameras due to Bayer patterns.

We assume that a non-colinear rig setup would yield better results because it guaranties that there are non-orthogonal epipolar lines for all gradients (see Section 2.5, ”Depth Estimation”).

5.4.2 Flexible Rigs

We evaluate our estimation of intrinsic rig parameters based on synthetic and real-world data. Figure 5.12 compares the intrinsic rig parameters to the ground truth of a synthetic

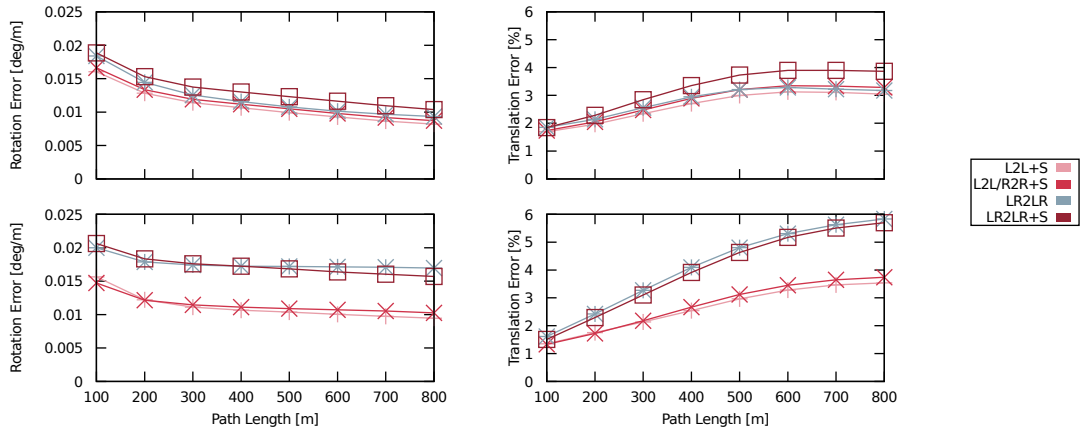


Figure 5.11: *Results from the KITTI odometry benchmark scene 00 with two (top) and four (bottom) cameras.*

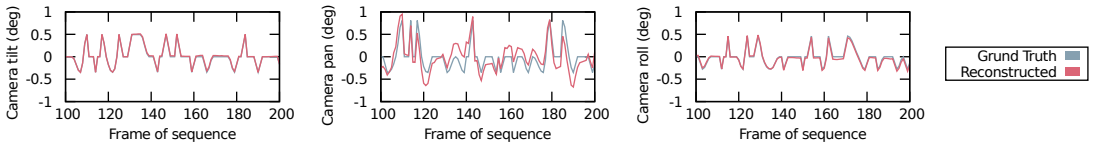


Figure 5.12: *Evaluation of the rig intrinsics reconstruction on synthetic data with ground truth. Note that roll and tilt are reconstructed nicely while the panning exhibits some artifacts due to our correction (Section 5.3.2) which we accept in favor of drifts or a collapsing reconstruction.*

scene (highly textured, 720p, with symmetric rig intrinsics as in Section 5.3.2) and shows that our approach clearly produces correct results.

For evaluation on real world data, we recorded three scenes with a flexible stereo camera rig with a FoV of 110 degrees and 2048x2048 pixels per image downsampled to 512x512 pixel images with 30 fps. The rig was bent and twisted by about 3 degrees while recording. Figure 5.13 compares the drift of the camera pose when sequences are reconstructed based on a rigid or a flexible rig model. To evaluate the drift, we moved the camera back to its origin at the end of the sequence and compare the first and last reconstructed rig pose. Reconstructed depth maps for stereo pairs are shown in Figure 5.3. Our results show that dense algorithms that don't keep track of the rig intrinsics perform significantly worse and introduce jittering to the rig pose since the tracking cannot converge on all cameras consistently. Reconstructions for this section were performed using LR2LR+S (see Figure 5.9), because we observed unstable reconstructions without cross-camera tracking on some of our flexible rig scenes.

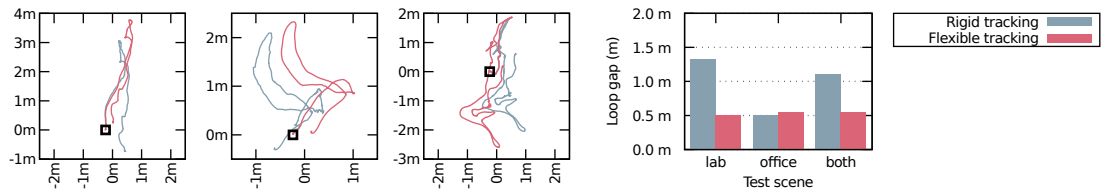


Figure 5.13: *Loop gap evaluation on a flexible rig dataset. Tracking with a flexible rig model clearly improves the drift in the tracking. A closer look to the plotted paths reveals that tracking a flexible rig with a rigid model leads to jittering in the reconstructed camera positions, most likely caused by convergence to one of the cameras when the cameras don't agree due to a deformed rig.*

5.4.3 Timings

Figure 5.14 compares the per frame processing times for different resolutions, different configurations and different steps of the algorithm. It shows that there is just small overhead for nonrigid tracking. It also shows that our GPU implementation is able to process even 4 Megapixel images in real-time.

5.4.4 Viewport Interpolation

We evaluate our method for viewport interpolation based on the obtained depth maps. For a stereo video sequence, the goal is to generate a center, cyclopean view from the left and right images. As illustrated in Figure 5.15, depth is obtained from the stereo video sequence with LSD-SLAM. From the depth, disparity maps are generated, filled and median filtered. The disparity maps are used to warp the left and the right image to the central viewport and both images are combined. This section therefore evaluates the usability of our LSD-SLAM extension for re-rendering of images where exact depth values are most important in high-gradient areas showing close surfaces and less relevant on homogeneous areas where warping errors hare hard to perceive and in the distance where depth errors have only low impact on the disparities.

We evaluate the results against ground truth renderings based on the squared intensity differences and the VDP-2 error [MKRH11] between the interpolated and the ground truth images. VDP-2 is a perception-based evaluation algorithm from which we visualize the ”probability of detection” map for the differences in images.

In the Figures 5.16 and 5.17 we compare the images reconstructed based on LSD-SLAM with images reconstructed based on disparity from the OpenCV implementation [Bra00] of the SGBM [Hir08] stereo algorithm. LSD-SLAM has clear advantages

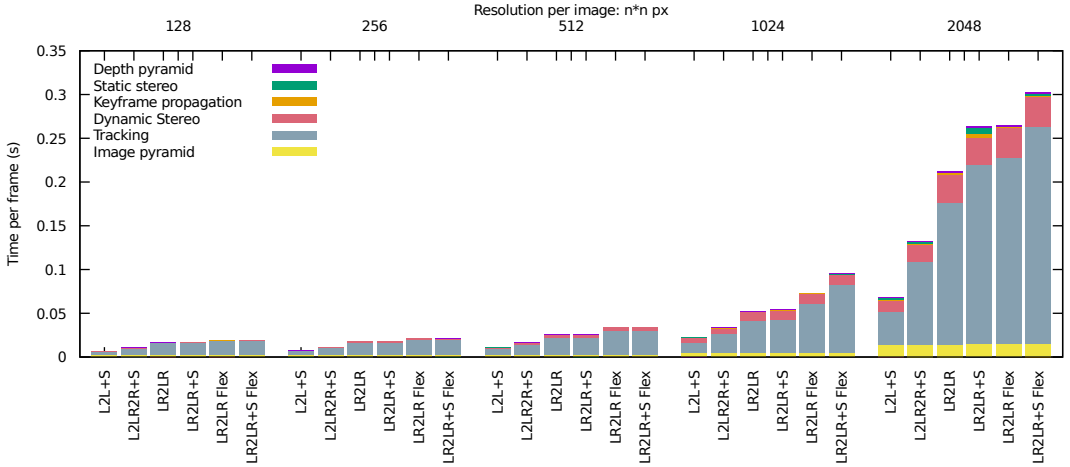


Figure 5.14: *Timing evaluation. The L2L+S configuration which works best for rigid stereo rigs runs at more than 15 fps even for 4 Megapixel images. There is just small computational overhead introduced by static stereo and nonrigid tracking. Note that there is some overhead for CPU/GPU synchronization which consumes significant time for smaller resolutions. Also note that the three tasks at the top that are only performed when the keyframe is updated have a very small average per frame contribution.*

Figure 5.15: *LSD-SLAM based viewport interpolation pipeline, from left to right: Input images, depth from LSD SLAM, cleaned and filled disparity, warped images, stitched result. Scene (without BMW model): CC-BY (`https://creativecommons.org/licenses/by/4.0/legalcode`) eMirage.*

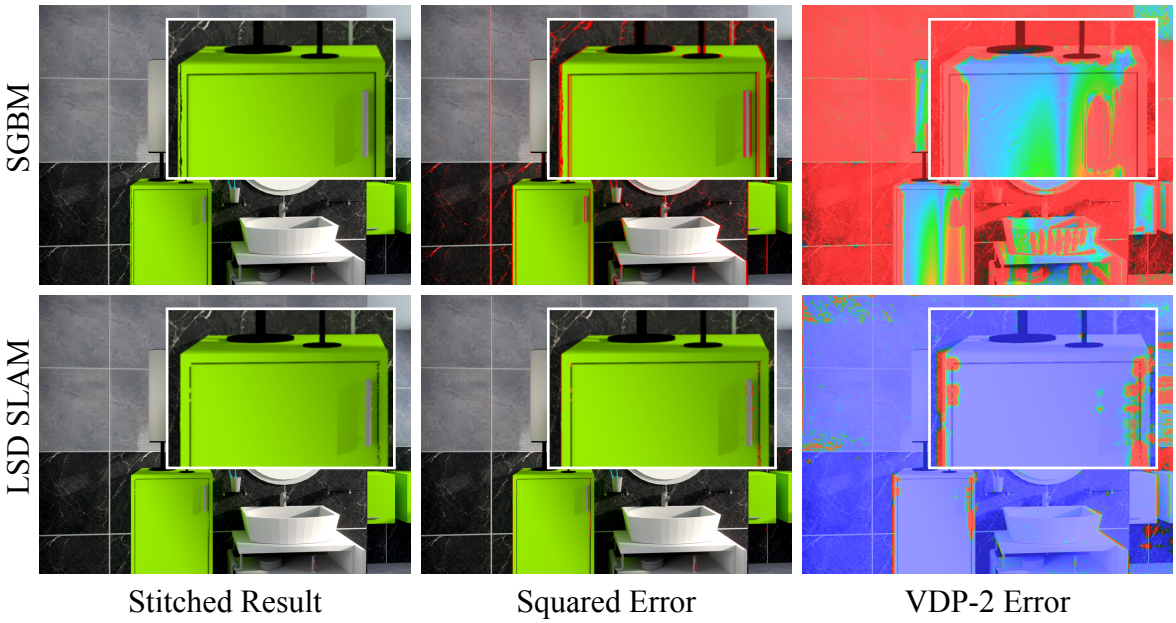


Figure 5.16: *Comparison of viewport interpolation based on SGBM and LSD-SLAM on a bathroom scene. We compare based on a squared image intensity error and the VDP-2 error norm. Scene: CC-BY 3.0 (https://creativecommons.org/licenses/by/3.0/legalcode), cenobi / blendswap.com.*

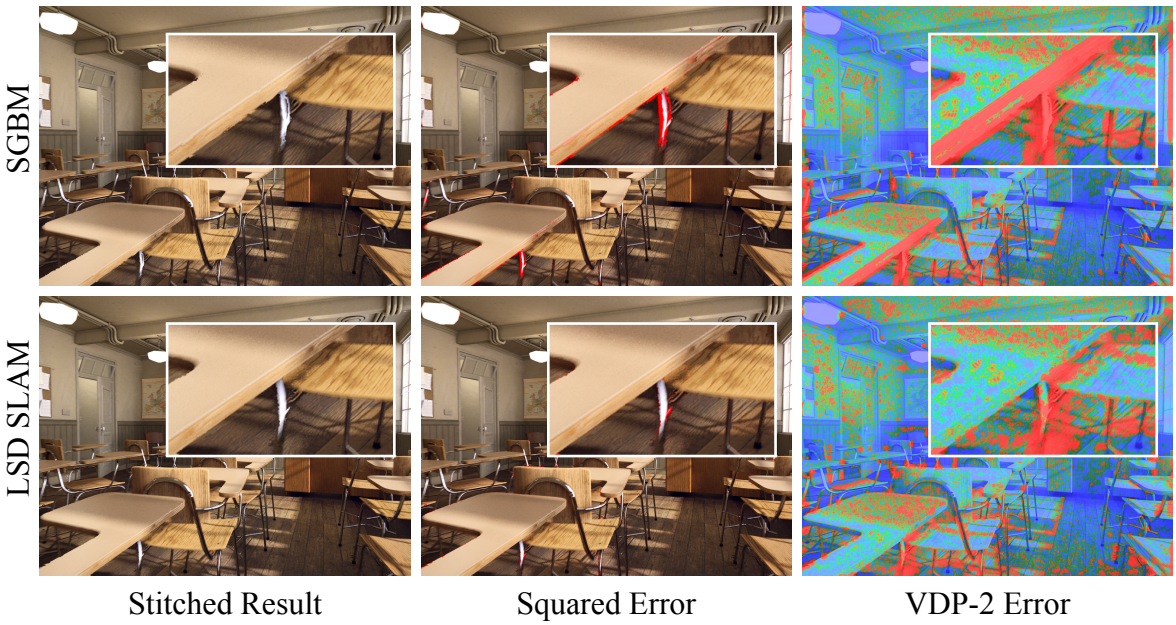


Figure 5.17: *Comparison of viewport interpolation based on SGBM and LSD-SLAM on a classroom scene. We compare based on a squared image intensity error and the VDP-2 error norm.*

Figure 5.18: *Test scenes for multi-camera rig evaluation: corridor (left) exhibits few gradients, repetitive ceiling texture and glossy highlights; office (center) shows many easily trackable gradients; veranda (right) is an outdoor scene with large depth variations.*

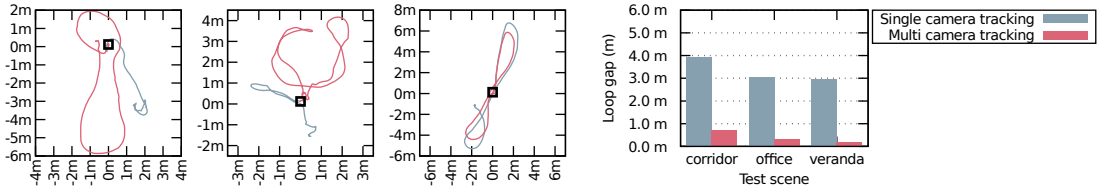


Figure 5.19: *Loop gap evaluation on the Point Grey Ladybug 3 which is an omnidirectional, 6 camera rig. We used static and dynamic intra-camera tracking (compare to Figure 5.9, B) on a rigid rig model. Note that in contrast to stereo data with overlapping fields of view (see Figure 5.10), the tracking benefits from taking all available data into account and optimizing for one consistent rig pose. The multi camera tracking reduces the error by one order of magnitude. The average processing time per frame (6 x 736x1136px) was 59ms for single camera and 120ms for multi camera tracking.*

because it can integrate depth information over many frames which enables it to sort out false depth values which seem ambiguous on the single image pair that is processed by SGBM (e.g. the marble tiles structure on the bathroom wall). Integrating data from multiple frames also leads to higher precision (vertical lines in the bathroom) and better results on smooth gradients originating, e.g., from motion blur (metal rod of the classroom chair).

5.4.5 Multi-Camera Rigs

To show the multi-camera capabilities of our algorithm, we evaluated it on video streams from a Point Grey Ladybug 3 which is a 6 camera, omnidirectional camera rig with slightly overlapping FoVs. We used three test scenes shown in Figure 5.18. The results in Figure 5.19 indicate that our tracking approach can reconstruct valid camera paths based on the data of all cameras where methods that use just one camera for tracking like [ESC15] fail, e.g. because of few or EPL-aligned high gradient pixels or a fixed point of expansion position in the images with leads to low disparities and therefore bad depth estimations in the surrounding.

5.5 Summary

We present a direct, semi-dense visual odometry method for flexible multi-camera rigs. Key features of our method are (1) extending the photometric error based tracking to optimize for consistent, relative rig poses as well as the rig intrinsics at once, (2) the update of the semi-dense depth with stereo information from multiple cameras in a Bayesian framework and (3) the support for dynamically moving objects in the scene.

For tracking rig intrinsics, we propose a stereo rig model which is free from ambiguous parameters and extend the tracking residual function for those parameters as well es for utilizing data from all available images. We also show how to limit the per pixel linear approximation of the tracking residual for LM optimization to the minimum set of six degrees of freedom per camera.

For combining the depth information from many measurements in a Bayesian framework, we utilize Gaussian mixture models to obtain a single normally distributed depth information per pixel that can be combined with the a priori information in a probabilistic way. By reordering the operations, we minimize the error introduced by Gaussian mixture approximation.

While monocular SLAM methods heavily rely on a static scene that can be triangulated over time, multi-camera methods can recover depth from images that were captured at the same time without having to rely on surfaces that stay at the same place. We show that LSD-SLAM's per pixel static scene tracking residual clearly identifies non-static scene parts and allows for switching to a classical stereo algorithm to reconstruct moving objects.

Our method achieves state of the art reconstruction quality for scenes recorded with completely rigid rigs. But it even supports the reconstruction with non-rigidly connected cameras. This reduces reconstruction errors dramatically, even if only small relative camera motion is present. Additionally, we show that our method is not just suitable for stereo cameras but also for other configurations like omnidirectional camera rigs with barely overlapping views.

In the future, this work could be extended to allow for the reconstruction of dynamic camera intrinsics as well by making the camera intrinsics a controllable parameter in the warping function (Equation 5.3). Another interesting direction for research would be the subsampling during tracking and mapping - either by dynamically omitting camera pairs or by subsampling the pixels that are evaluated during tracking and updated during depth estimation.

Chapter 6

Learning Visual Odometry

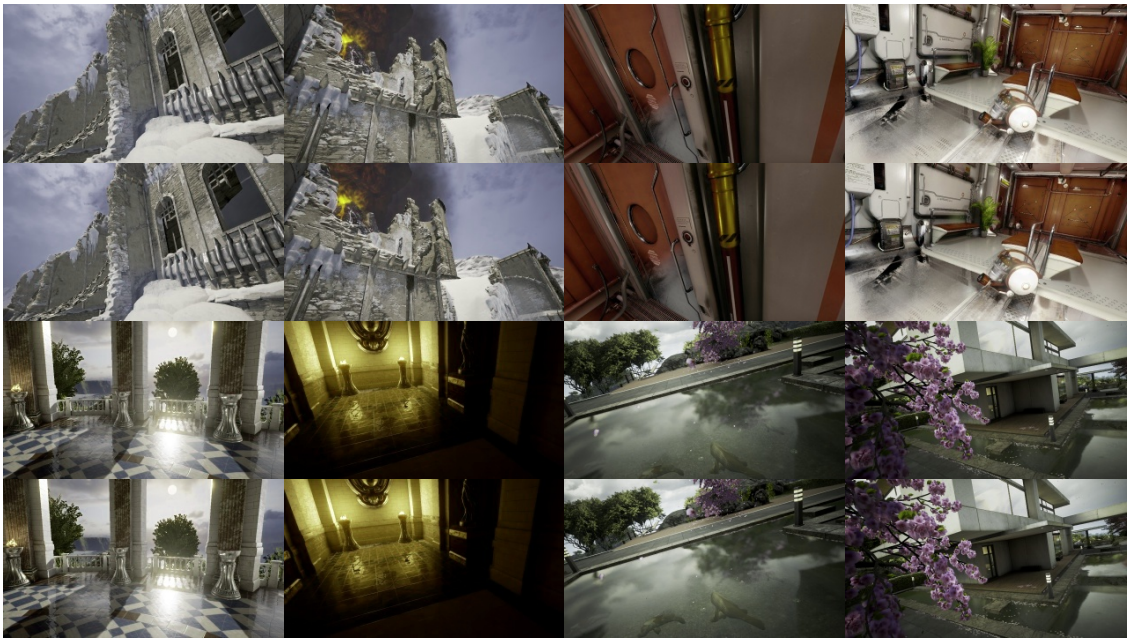


Figure 6.1: *Online rendered image pairs used to train our CNN for Visual Odometry. We show two frame pairs per scene: Elemental (top left), SciFi Hallway (top right), Sun Temple (bottom left) and Zen Garden (bottom right).*

After determining camera trajectories based on sparse features and with a semi-dense method in the previous chapters, we analyze the utilization of machine learning, specifically convolutional neural networks (CNNs), for this task. To train such a network, we present a method for generating unlimited, high quality training data with the Unreal Game Engine 4. Our approach can generate training data on one machine as fast as it is consumed by the training of our CNN, making it unnecessary to prepare the training data offline or to store it. Furthermore, this allows us to do an in-depth analysis on how properties of the training data, e.g. the number of frames or the distribution of labels, affect the quality of the trained neural network. We evaluate our results on test data generated with our approach as well as real data captured with a moving camera.

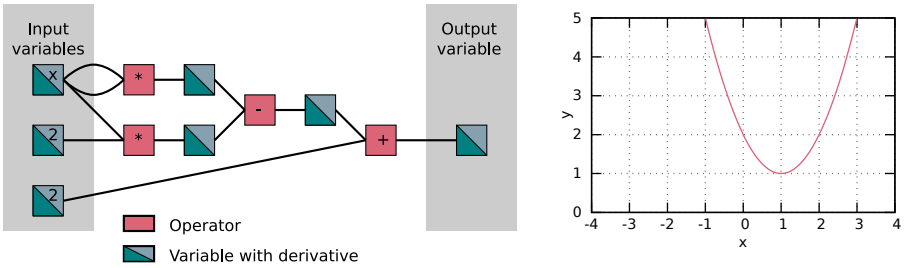


Figure 6.2: *Equation 6.1 visualized as graph and as plot. Note that the graph contains variables for all intermediate results. In addition, all variables are split: They contain the values evaluated on the forward pass (left to right) and the derivatives evaluated on the backward pass.*

6.1 Introduction to Artificial Neural Networks

Artificial neural networks can be seen as graphs representing mathematical functions which are to be minimized in the broadest sense. We will illustrate this on the following equation

$$f(x) = x^2 - 2x + 2 \tag{6.1}$$

which is visualized in Figure 6.2 as graph and as plot. Such nonlinear functions can be minimized with the gradient descend method (see Section 2.3.2) for which the derivative of the function to every parameter (here only $\partial f(x)/\partial x$) has to be obtained. For functions which have reoccurring elements, this can be done very efficiently by backpropagation.

6.1.1 Backpropagation

The idea behind backpropagation is to process the equation's graph first in a forward pass (from left to right) to obtain the function value $f(x)$ given initial parameter values x. Then, derivatives are calculated in a backward pass (from right to left) until the input variables are reached. At every operator, the chain rule

$$\frac{\partial a(b(x))}{\partial x} = \frac{\partial a(b(x))}{\partial b(x)} \frac{\partial b(x)}{\partial x} \tag{6.2}$$

can be applied: In order to calculate the derivative for the input (graph: left hand side) of an operator b, its local derivative $\frac{\partial b(x)}{\partial x}$ can be multiplied with the derivative of the encapsulating operator $\frac{\partial a(b(x))}{\partial b(x)}$ at its output (graph: right hand side).

Given this background, backpropagation on a graph as shown in Figure 6.2 boils down to the following steps:

Forward Pass:

- Initialization:
  Mark all input values as available.

- For each operator that has all input values available:
  Compute output values.

- For each computed value:
  Mark as available.

Backward Pass:

- Initialization:
  Set output derivative to 1 and mark as available.

- For each operator that has its output derivative available:
  Compute all input derivatives according to the chain rule (Equation 6.2). If the derivative's variable contributes to multiple operators, add up the derivative contributions of all operators.

- For each derivative hat has been added up by all its output operators:
  Mark as available.

After one forward and one backward pass, the parameters must be updated with a fraction of their negated derivative to implement a gradient descent step and reduce the function value in the next iteration.

6.1.2 Artificial Neural Networks

Backpropagation can now be used to evaluate parameters on graphs which mimic neuronal behaviour as it can be found in the visual cortex of the brain [TD11]. A single neuron which collects information from its input variables $\boldsymbol{\psi}$ which are then weighted ($\boldsymbol{w}$) and lead to transmitting a signal to further connected neurons if above a threshold can be expressed by

$$N(\boldsymbol{\psi}; \boldsymbol{w}) = f(\sum_i \boldsymbol{\psi}_i \boldsymbol{w}_i). \tag{6.3}$$

Instead of a discrete signal, the mathematical formulation models the continuous probability for transmitting a signal, which is modified by an activation function f that introduces nonlinearity into the otherwise linear function as it can be observed in real neurons. Typical activation functions that are used in ANNs are shown in Figure 6.3.

To form a network, artificial neurons are structured in layers as shown in Figure 6.4. Typically, such layers are fully connected: each neuron is connected to all neurons from the previous layer.

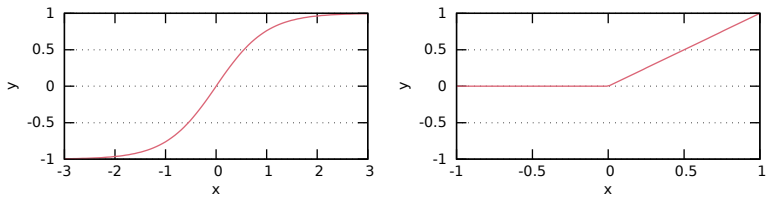


Figure 6.3: *Typical activation functions for artificial neurons:* $\tanh(x)$ *(left) and* $\mathrm{ReLU}(x)$ *(right).*

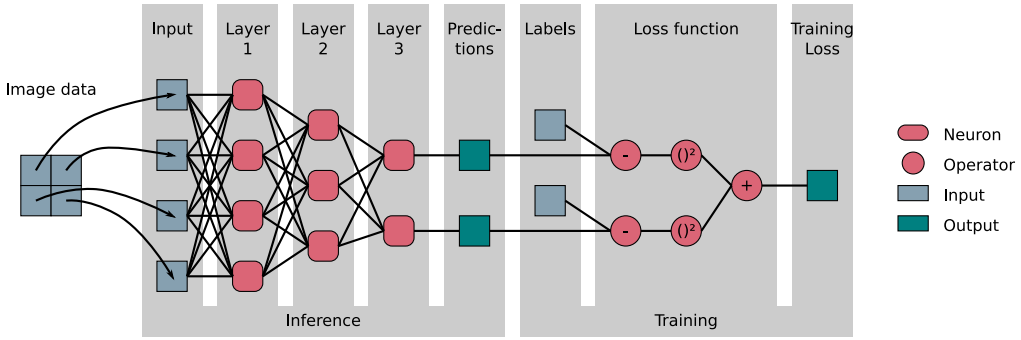


Figure 6.4: *Artificial neural network with multiple layers. Input for training may be given as images + labels. As loss function, the predictions of a neural network can be compared in a least squares sense with known labels.*

Training

For supervised training, the predictions of a neural network can be compared in a least squares sense with known labels (see Figure 6.4). Training then aims on adjusting the weights of the neurons to minimize this squared difference. In contrast, unsupervised learning judges the quality of the predictions ad-hoc, e.g. based on their consistence, (physical) constraints that must be met or other algorithms that solve related problems.

In the simplest case of supervised training, one training sample consisting of the input $\boldsymbol{\psi}$ and the corresponding label $\boldsymbol{l}$ is shown to the neural network per training iteration. All weights in the neural network are slightly adjusted as described in Section 6.1.1 and the training continues with the next iteration. This stochastic gradient descend scheme eventually converges to weights which allow the neural network to predict labels with a certain quality. For better stability in the training process (and better utilization of massively parallel GPUs), training samples can be combined in batches which are propagated through the network at once. This way, the weights are more likely to adjust to consistent properties of all samples of the batch.

There are several methods for choosing an optimal step size for the parameter update during optimization. Beside simple gradient descend with a fixed step size, Adam [KB14] has become very popular which updates all parameters based on momentum vectors which are only slightly influenced by the gradients in every iteration.

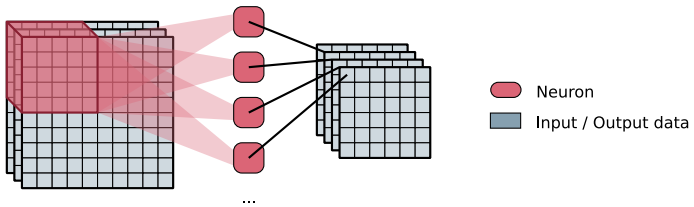


Figure 6.5: *Convolutional layer of a neural network with multiple input maps and multiple convolution kernels (weights) which generate an output map each. All neurons calculating the pixels of one output feature map share the same convolution kernel.*

6.1.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) usually consist of several convolutional and several fully connected layers (also compare to Figure 6.7). Convolutional layers apply a set of convolution kernels to the input images as illustrated in Figure 6.5, generating multiple output feature maps. A convolutional layer contains one neuron per output pixel which has a receptive field corresponding to the output pixel's position $(u\ v)$. All neurons that compute pixels of one output map share the same weights as all output pixels of a convolution are computed with the same convolution kernel.

Often, deeper layers in a CNN become smaller in terms of width and height but have more feature maps (see Figure 6.7). Shrinking the images can be achieved by applying a stride on the read input pixels or by pooling layers. Pooling layers reduce the size of the input by combining $n \cdot n$ pixels of their input by propagating only the pixel with the greatest input value.

6.1.4 Deeper ANNs

Deeper neural networks are in general able to represent more complex functions. Therefore, many established neural network models [SLJ+15, UZU+17] feature 30 or more layers. Since each layer usually includes a multiplication with weights and a range-limiting activation function, output values tend to get smaller with every additional layer which leads to numerical problems for deep networks. Luckily, most of the information that is transported between layers is encoded in the distribution or the neuronal responses rather than in their absolute magnitudes. This allows for normalization of a layer's output which prevents the output values from vanishing completely. One example for normalization is the local response normalization (lrn) [KSH12] that normalizes every pixel $(u\ v\ m)$ by the L2 norm of all equally positioned pixels of the other feature maps $(u\ v\ \{1..M\})$.

There are multiple open source ANN libraries available such as Theano, Torch, Caffe, TensorFlow or MXNet. They all support the operations and layers which are necessary to reimplement popular NN architectures such as AlexNet [KSH12] or GoogLeNet [SLJ+15].

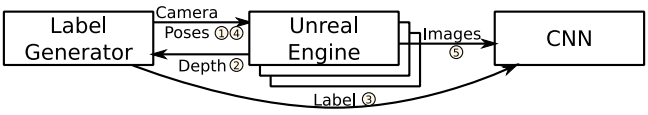


Figure 6.6: *Framework overview. Per training sample, (1) the label generator sends a start camera pose to one engine instance, (2) receives the corresponding depth image, (3) generates a label for training and (4) sends corresponding camera poses with depth-normalized translation to the engine to (5) generate the corresponding image sequence.*

Choosing one of them is mainly a matter of preference for programming languages, the specific APIs and of support for / performance on the available compute infrastructure.

6.2 Motivation

Visual Odometry (VO) refers to recovering egomotion from visual input and is a long standing task in computer vision. Early methods [HJ92] factored out rotation and translation of a camera from the biologically inspired optical flow between a pair of frames. Another approach is to track feature points from one frame to the next and to calculate the relative pose from at least five points [Nis04]. More recent approaches yield by far better results by building up a local scene representation, either by using a sparse set of 3d scene features [DRMS07, KM07, SF11] or by using a depth map [ESC14] (see also Section 2.5 and Chapter 5).

During the last two years there have been attempts to apply learning-based approaches with neural networks to the VO problem: earlier approaches learn the relative pose directly on images while newer ones utilize optical flow or depth as input for training and inference. The most recent publications jointly estimate depth and the relative camera pose, yielding the best results so far at the cost of increasing network depths, complexities and computational needs further and further.

In this work, we avoid to build larger and more complex network structures. Instead, we focus on the generation of training data and analyze how the training data should be set up to improve the quality of the results. However, most likely larger networks will benefit from our framework as well.

As shown in Figure 6.6, we first define the labels on which we want to train our network. Then we utilize the Epic Games Unreal Engine 4 (UE4) to render corresponding images which we use for training our TensorFlow based CNN.

Rendering the images used for training with a game engine has several advantages: Beside having perfect ground truth for all training data, we can generate limitless data for training and we are able to generate data at training speed. We use Epic's carefully crafted Engine feature demos as foundation for our renderings which contain realistically modeled scenes and effects including glossy surfaces and reflections, moving particles

and objects, motion blur, depth of field, fog, haze, brightness adaption, bloom and lens flares. This results in stunningly realistic images (see Figure 6.1).

Our results (Section 6.5) show that a network trained on such data generalizes to captured video frames. Furthermore, we found some interesting insights on which training data should be chosen to improve VO results significantly, e.g. supplying multiple frames from perturbed camera poses to the network during training (Section 6.5.1) and adapting the training label distribution to the expected testing label distribution (Section 6.5.4).

6.3 Related Work

In this section we give an overview over methods that employ machine learning to estimate a camera's pose. We also introduce some publications that incorporate game engines and machine learning. For a survey on conventional VO techniques, please refer to Section 5.2.1.

6.3.1 Neural Network based Localization and VO

Allthough this chapter is about VO, we will first have a look at some publications about localization in a known environment where a NN is trained on pose-labeled images of an environment and challenged to find the pose of previously unseen images of the *same* environment. This is an easier problem than VO where the NN has to predict the relative pose of a camera given few images of an *unseen* environment and was therefore treated first. Kendall et al. created PoseNet [KGC15] by removing the softmax classification layers from GoogLeNet [SLJ+15] and adding some fully connected layers for regression, yielding significantly better results than a nearest neighbour classification CNN. This approach was further improved to model uncertainty [KC16], to balance rotation and translation error loss automatically [KC17] and to incorporate depth data in addition [LLG+17]. Clark et al. [CWM+17] improve localization results significantly for video sequences by propagating information to previous and subsequent frames with LSTM layers.

Konda and Memisevic [KM15] as well as Costante et al. [CMVC16] achieve very good results for VO on the KITTI [MG15] benchmark, by training on KITTI data and either by using stereo data or optical flow as input to their neural networks. However, the KITTI dataset exhibits few vertical movement and tilting as well as a tight coupling of horizontal movement and panning of the camera due to the car's steering mechanism, thus simplifying the VO problem significantly. Zhou et al. [ZBSL17] provides an approach that promises to be more invariant to the scenery: Similarly to conventional dense VO methods, the relative camera poses and depths for one target view are estimated simultaneously by two CNNs. Then, the target view is warped into all other views (compare to Section 2.5.1) and the resulting photoconsistency error is used as loss for training the CNNs. Explicit modelling of depth in the network is also the key for wider baselines with

DeMoN [UZU+17] where encoder-decoder networks transform forth and back between a depth+pose and a optical flow representation, improving the results after every iteration.

All the NN based VO approaches shown above have either limited potential for generalization [KM15, CMVC16] or they have very deep and complex network structures and explicitly modelled depth in the network [ZBSL17, UZU+17] which leads to high computational costs. In contrast, we use a very simple CNN structure and concentrate on evaluating how to improve the VO quality by using better training data.

6.3.2 Playing for Data

Some recent publications use games and machine learning in a common framework to learn to play games. Mnih et al. [MKS+13] feed the screen output of Atari games to a neural network that produces user input to play the game. Bhatti et a. [BDM+16] process the screen output of the game Doom with a SLAM algorithm and feed its output to a NN which plays the game. Richter et al. [RVRK16] analyze the communication of computer games with graphics hardware to extract associations between image patches. This allows them to easily propagate semantic labels from one rendered frame to the next for creating semantic object labeling training datasets.

6.4 Framework Description

As shown in Figure 6.6, our framework consists of a label generator that sends per-image camera poses to the Unreal Engine (UE) and per-training-sample (sequence of 2-5 images) labels to the CNN for training. The label generator receives depth maps from the UE to be able to normalize translation labels by the distance to the observed structures. Finally, images corresponding to the generated labels are rendered by the UE and sent to the CNN for training.

6.4.1 Label Definition

We use 6 dimensional labels l consisting of three Euler rotation angles and three translation vector components:

$$\boldsymbol{l} = (r_x, r_y, r_z, t_x, t_y, t_z) \tag{6.4}$$

Our label generator creates labels with equal variance for all components which ensures that the training loss is balanced for rotation and translation and that the CNN respects both equally well during training (this has also been found to be important by [KGC15, KC17]).

Rotation: For larger rotation angles, the singularity-free quaternion or axis/angle representations are preferable [UZU+17] but need additional projections to rotation space or

handling of multiple representations for the same rotation. For small rotations as in our VO setting we found that there are no drawbacks from using Euler angles.

Translation: A common problem for reconstructions from monocular cameras is the scale ambiguity: The absolute, metric scale of the scene cannot be recovered from the input frames only and therefore it is also not possible to recover the metric magnitude of the translation vector just from the images, which makes training a NN on it senseless. However, we can render our training data in a way such that the translation label magnitude is proportional to the disparity in the images. Technically, we

1. define a target translation (t_x, t_y, t_z),
2. render the depth $D(u, v)$ of the first frame of the sequence,
3. calculate the average inverse depth

$$\bar{d}^{-1} = \sum_{u,v} \frac{1}{D(u, v)} \tag{6.5}$$

   which is proportional to the average disparity and

4. use a corrected translation

$$(\hat{t}_x, \hat{t}_y, \hat{t}_z) = \frac{(t_x, t_y, t_z)c_t}{\bar{d}^{-1}} \tag{6.6}$$

   for rendering the training sample.

This ensures that the translation which is used for rendering is scaled with the images' depths so that the rendered disparity is related to the translation magnitude no matter how distant the scene is to the camera.

We introduced a correction factor c_t which ensures that similar rotation and translation label magnitudes lead to similar disparity in the images. For all our experiments we used $c_t = 0.2$.

6.4.2 Image / Depth Rendering

For rendering the images, we use the Epic Games Unreal Engine 4.14 which is one of the major game engines that is frequently used for AAA titles. It is running on different platforms and its source code is available completely which makes it a natural choice for a research project.

As starting point we used several engine feature demos which feature realistically modeled environments, namely the Elemental Demo, the Sun Temple, the Epic Zen Garden and the SciFi Hallway (see Figure 6.1). We disabled the in-engine temporal Anti-aliasing:

it is implemented by jittering the whole image plane of the camera with sub-pixel magnitude and accumulating the results only when the image content is still, thus introducing random but consistent shifts on whole images when the camera moves like on our renderings. We also disabled in-engine multisampling because we obtained better results by rendering everything in 4x resolution and scaling down the results by ourselves. We fixed the frame rate of the engine to 30 fps which means that the in-game time advances by 1/30 s per frame which results in images most similar to a 30 fps capturing concerning object and particle motion and motion blur. Since UE is a pure online rendering engine, it won't render more than 30 fps with these settings. To achieve a higher rendering speed, we run up to 12 UE instances concurrently, distributed over 3 GPUs, which can in total render 360 fps.

Feeding Camera Poses into UE

We transfer data from and to the UE via named pipes. The UE features actors which can be thought of as empty 3D objects from which one can derive a custom actor that can be implemented in C++. Our custom actor reads one camera pose per frame from a named pipe and adjusts itself accordingly. We attach a camera to our actor which follows its movements, generating the desired images.

Sending Frames from UE

We transfer the rendered images and depth maps via named pipes as well. Although the engine's game developer API provides a method for taking screenshots and processing them programatically, we opted not to use it because of its slowness and the inability to retrieve high quality depth maps. Instead, we modified the engine itself, specifically the deferred shading renderer as well as the graphics hardware abstraction layer to extract the rendered images and depths from the GPU as soon as they are rendered and to transfer them to the CNN.

6.4.3 Learning

Figure 6.7 shows the CNN that we implemented with TensorFlow. Input to our network are grayscale images of a sequence and their Sobel (X/Y) filtered gradient images. Our network contains five convolutional layers, three fully connected layers and one regression layer. We weight the resulting error based on an image dependent importance (see Section 6.5.5 for details) and use an L2-norm loss. In addition, we limit the magnitude of the fully connected layer weights by adding them to the loss function.

In our experiments we found that the tanh activation performs superior to other activation functions like the currently popular ReLU. We expect this to be a result of tanh's similarity to the atan function which is directly involved in deriving rotations from disparities.

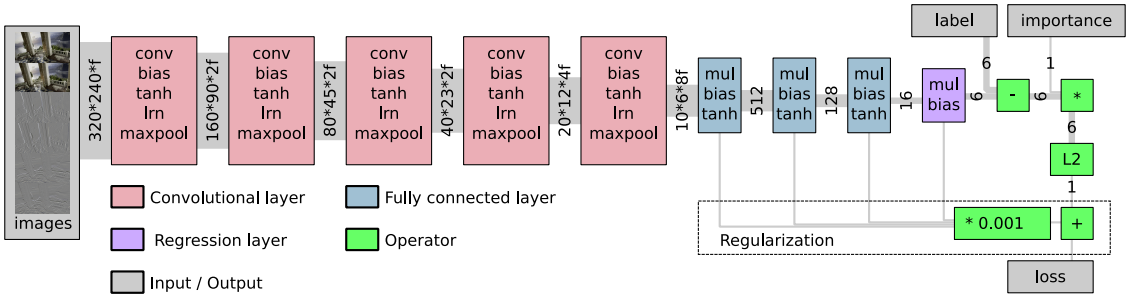


Figure 6.7: *CNN structure used for training. The f input layers (grayscale images + Sobel (X/Y) filtered versions) are reduced by several convolutional layers and then processed by three fully connected layers until a final, fully connected regression layer predicts the relative camera pose which is compared with the label, weighted and its L2-norm is added to the loss function. In addition, we regularize the weights of all fully connected layers by multiplying them with a small factor and adding them to the final loss.*

Figure 6.8: *Exemplary images from our captured real world dataset, featuring various indoor and outdoor scenery.*

We train our model from scratch with Adam [KB14] using a learning rate of 0.0001 with a batch size of 100 samples.

6.5 Evaluation

In the following experiments, we use images with 320x180 pixels and train our network for 5000 iterations if not mentioned otherwise. We evaluate on different datasets (compare to Figure 6.1,6.8):

E Images rendered from the Elemental scene

A Images rendered from all synthetic scenes

O Images rendered from all synthetic scenes except Elemental

R Real world captured images

We express training on certain data with the Tr() operator and testing with the Te() operator. For evaluating on real world captured images, we utilized the algorithm from Chapter 4 to generate ground truth labels.

	Tr(E)xTe(E)		Tr(A)xTe(A)		Tr(A)xTe(E)		Tr(O)xTe(E)		Tr(A)xTe(R)	
	Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran
2 frames	0.60	33.45	0.68	38.06	0.70	39.14	0.72	38.61	0.86	47.23
3 frames	0.54	30.07	0.63	34.58	0.66	36.27	0.67	35.77	0.93	50.20
5 frames	0.50	29.48	0.55	30.56	0.57	33.56	0.61	34.33	1.00	50.20
5 frames perturbed	0.49	29.12	0.54	29.99	0.57	34.46	0.61	35.08	0.88	46.06

Table 6.1: *Quality of the results related to the number of frames used per sample. We give the average rotational and translational errors on the test set in degrees (for translations: between translation directions). We compare different input data combinations for training and testing (columns) and different numbers of input frames (rows).*

.

6.5.1 Varying Number of Frames

We evaluated our framework on 2, 3 and 5 frames per training and test sample, hoping that the network utilizes the additional information (and larger baseline) from the additional frames.

For more than 3 frames, a simple but unnatural way for *training* data generation is to apply the same camera motion to all frames, hoping for the network to regress a good approximation when a test sample exhibits different motions from frame to frame. Another approach is to perturb the camera motion during *training* for every but the last pair of frames in a training sample, thus providing the network with all information of the whole sequence but training it only for the relative camera pose of the last two frames.

Our synthetic *test* samples are always generated with perturbed camera motion where the label corresponds exactly only to the motion of the last pair of frames in the sequence.

Our results (Table 6.1) show that using more frames improves the results indeed on our synthetic training set, both for networks which were trained on previously seen or unseen scenery. However, an increase of the errors can be observed from more frames with constant camera movement between successive frames of a training sample on real data. We assume that this is the result of the real camera's movement being more irregular than the movement of our synthetic camera. When we perturb the camera motion during training, our CNN performs better on real data.

Figure 6.9 shows a typical development of training loss over time. Our CNN converges better when more frames per sample are involved in training.

In Figure 6.10 we show the cumulative error functions for rotation and translation. One can clearly see that results get slightly worse when the CNN is tested on (partially) unseen scenery but that it performs well even on completely unseen data (Tr(O) x Te(E)) compared to the baseline which is predicting zero labels always.

Figure 6.11 plots error magnitude over the label magnitude and gives an insight on how the loss function is optimized. At the beginning of the training, zero labels are predicted which makes the rotational and translational error equal to the label magnitude. Then errors start to drop. Note that samples with lower rotation or translation magnitude get good predictions earlier than samples with higher label magnitudes.

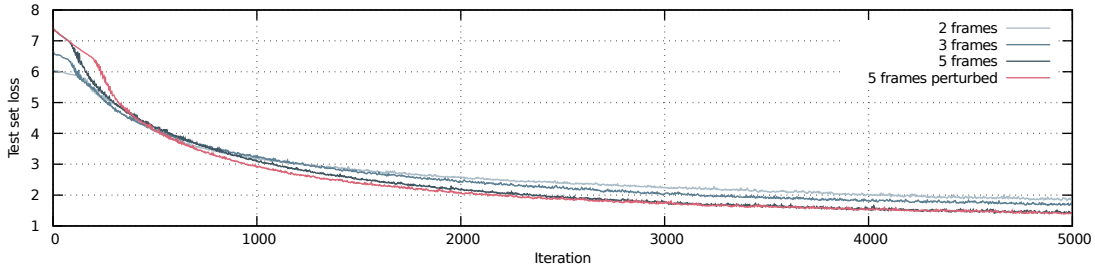


Figure 6.9: *Test loss of CNNs trained with a different number of frames per sequence.*

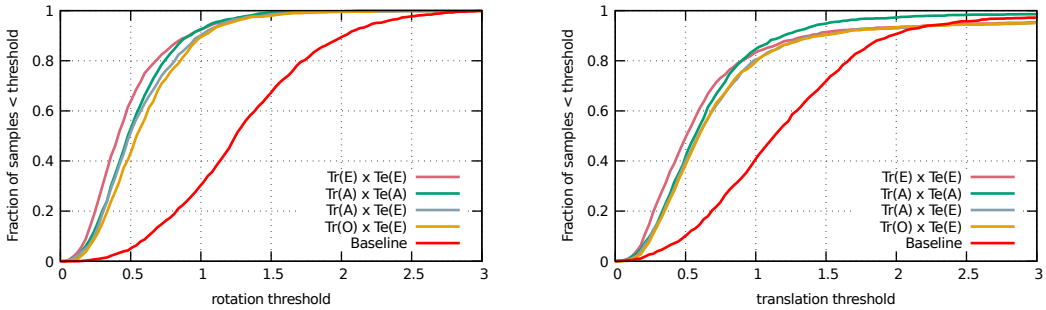


Figure 6.10: *Cumulative error functions over rotation (left) and translation (right) labels for training data with 5 perturbed frames per sample. Rotation labels correspond to degrees, translation labels are normalized by the average inverse depth and rescaled to match the magnitude of the rotation loss (Section 6.4.1).*

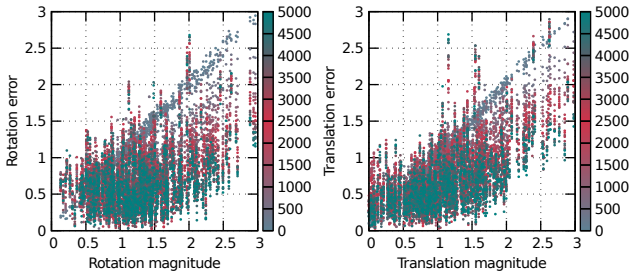


Figure 6.11: *Errors over label magnitude over time for rotation (left) and (translation). Note how the errors drop from the main diagonal at the beginning of the training (predicting zeros leads to errors similar to the labels) closer to zero over time.*

		Tr(E)xTe(E)		Tr(A)xTe(A)		Tr(A)xTe(E)		Tr(O)xTe(E)		Tr(A)xTe(R)	
		Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran
2 frames	160x90	0.64	34.56	0.67	36.31	0.70	37.55	0.75	39.77	0.85	44.82
	320x180	0.60	33.45	0.68	38.06	0.70	39.14	0.72	38.61	0.86	47.23
5 frames perturbed	160x90	0.52	29.76	0.56	30.25	0.60	34.87	0.62	36.80	1.00	44.39
	320x180	0.49	29.12	0.54	29.99	0.57	34.46	0.61	35.08	0.88	46.06

Table 6.2: *Resolution dependence of the training results. We give the average rotational and translational errors on the test set in degrees (for translations: between translation directions). Tests are performed for different training / testing data (columns) on different frame configurations (row blocks) in full and half resolution (rows). Increased input resolution leads to slightly but consistently better results.*

6.5.2 Resolution Dependence

We evaluated our framework on its dependence on image resolution. To train and test with half resolution, we scaled down the first convolutional layer of our network by 50% and dropped the second one (compare to Figure 6.7). Results are presented in Table 6.2. Obviously, results are almost similar with half resolution. This indicates that the quality of our results is currently not limited by the information in the images but by either the complexity of the network or by insufficient training.

6.5.3 Correlation of Rotation / Translation Error

Tilting the camera vs. moving it up and down as well as panning it vs. moving it to the left and the right have very similar effects in the camera's image. Therefore it is important to see if our CNN can distinguish such motions and rotations. We analyze the covariance of rotation and translation errors in Table 6.3. It shows that the rotation / motion combinations described above are indeed the critical ones. However, the table as well as Figure 6.12 show that the error covariance can be reduced by using more frames per sample to train and test the network. This shows (1) that even a simple CNN is in principle able to distinguish such motion and rotation and (2) that it exploits the information from the additional frames, even when it is only trained on the camera movement between the last two frames of the sample (see Section 6.5.1).

6.5.4 Influence of Training / Testing Data Distributions

Since we generate all our data ad-hoc and according to previously specified labels, we can easily experiment with different label distributions for training and testing data. We select three substantially different distributions for specifying the Euler rotation angles and translation vectors:

Normal A Normal distribution with $\sigma = 0.8$

	Tr(E) x Te(E)				Tr(O) x Te(E)			
2 frames		$E(t_x)$	$E(t_y)$	$E(t_z)$		$E(t_x)$	$E(t_y)$	$E(t_z)$
	$E(r_x)$	-0.01	-0.02	0.13	$E(r_x)$	-0.00	0.08	0.28
	$E(r_x)$	-0.02	0.06	0.04	$E(r_x)$	-0.04	0.13	-0.04
	$E(r_z)$	-0.09	0.07	-0.05	$E(r_z)$	-0.15	0.08	-0.07
5 frames perturbed		$E(t_x)$	$E(t_y)$	$E(t_z)$		$E(t_x)$	$E(t_y)$	$E(t_z)$
	$E(r_x)$	-0.00	-0.01	0.10	$E(r_x)$	-0.00	0.02	0.13
	$E(r_x)$	-0.01	0.00	-0.04	$E(r_x)$	-0.01	0.00	-0.02
	$E(r_z)$	-0.08	0.01	-0.02	$E(r_z)$	-0.11	0.03	-0.01

Table 6.3: *Covariance of rotation and translation errors. We show a subset of the covariance matrix of the label errors E which shows the covariances between rotational and translational label components. Results are shown for different training / testing data (columns) and frame configurations (rows). The highest covariances are present for rotation / translation combinations which have similar effects on the images, e.g. tilting and moving the camera up/down. However, more frames per sample reduce this problem.*

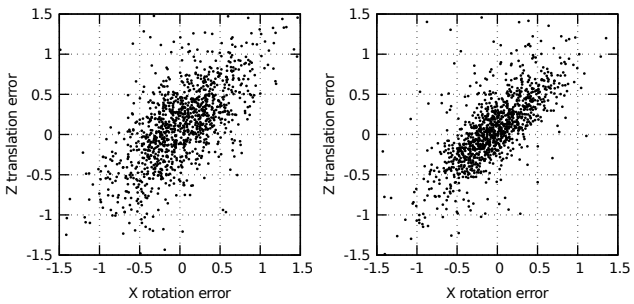


Figure 6.12: *Error correlation for tilting and moving up/down for training on 2 (left) and 5 perturbed (right) frames. Note that the network can distinguish rotation and translation better with more images.*

5 frames perturbed	Tr(E) x Te(E)			Tr(A) x Te(A)			Tr(A) x Te(E)			Tr(O) x Te(E)		
	TeN	TeU	TeB	TeN	TeU	TeB	TeN	TeU	TeB	TeN	TeU	TeB
TrN	0.49	0.50	0.70	0.54	0.56	0.82	0.57	0.58	0.84	0.61	0.65	0.87
TrU	0.53	0.49	0.66	0.58	0.53	0.70	0.60	0.56	0.73	0.65	0.62	0.80
TrB	1.01	0.89	0.43	1.00	0.84	0.45	1.03	0.89	0.50	1.08	0.94	0.56

Table 6.4: *Dependence of the training results on the label distribution. We created the labels for training (rows) and testing (columns) according to different distributions, namely a normal distribution, a uniform distribution with equal variance and a double beta distribution. We give the average training errors for rotations in degrees for different combinations.*

Uniform A uniform distribution with $\sigma = 0.8$

Double Beta Two end-to-end attached beta functions forming an U-shape distribution with a width similar to the uniform distribution.

Table 6.4 shows the results on training and testing with different combinations of label distributions. One can clearly see that best results are obtained when we train and test on the same distribution. Overall, normal and uniformly distributed training seems to provide the best results while U-shaped distributions only seem to be beneficial for use cases where no slow camera movement is expected. There is no clear evidence that the training converges faster in general when the one or the other training label distribution is present.

6.5.5 Weighting of Training Samples

In theory it might be beneficial to exclude or down-weight training samples from which the camera motion cannot be deduced, e.g. because the camera does not see any structure or because the VO problem is ill-posed on the seen structure, e.g. when it is a one-dimensional line. We use the average image gradient as an estimate for the probability that the VO problem can be solved and weight the training samples accordingly (see "Importance" in Figure 6.7). The results are compared to weighting all training samples equally in Table 6.5. Although such preselection of samples is a common technique for CNNs, it does not seem to improve the results for the VO problem on our network. We assume that the inherent invariance of the stochastic training of the CNN to inconsistent samples provides a better filtering of the training samples than the simple gradient heuristic.

6.5.6 Long Training Results

We trained our CNN for 5000 iterations in the previous experiments because we observed that this is usually enough to see how well different framework configurations perform. To give an insight on how longer training affects our results, we repeated training on the A dataset (Section 6.5) with 5 perturbed frames over 22000 iterations. Compared to 5000

		Tr(E)xTe(E)		Tr(A)xTe(A)		Tr(A)xTe(E)		Tr(O)xTe(E)		Tr(A)xTe(R)	
		Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran	Rot	Tran
2 frames	Equal	0.63	34.82	0.64	35.25	0.67	37.03	0.70	38.14	0.79	46.89
	∅ grad.	0.60	33.45	0.68	38.06	0.70	39.14	0.72	38.61	0.86	47.23
5 frames perturbed	Equal	0.48	29.59	0.51	27.80	0.53	32.04	0.61	34.96	0.95	43.91
	∅ grad.	0.49	29.12	0.54	29.99	0.57	34.46	0.61	35.08	0.88	46.06

Table 6.5: *Dependence of the training results on the weighting of images during training. We trained our network once by weighting all images equally and once by weighting images based on their average image gradient (rows) and compare the results for different frame configurations (row blocks) and training / testing data (columns).*

iterations, we observed the testing loss on the E dataset to drop from 1.03 to 0.58 (-44%) while the rotation error dropped from 0.57 to 0.42 degrees (-26%) and the error of the translation direction improved from 34.46 to 24.91 degrees (-28%).

6.6 Summary

In this chapter, we present a framework to train a CNN for Visual Odometry based on online-rendered, realistic images from the Unreal Game Engine with high speed. We describe how we integrated the UE in a machine learning framework. Our results show that our synthetically generated data generalizes to other rendered scenery and real world captures. The online training data generation enables us to run several special evaluations like training with different numbers of frames per training sample or different label distributions. Such evaluations are not easily achievable with pre-generated or captured training data. Based on our framework, we can show how the number of frames given for tracking, the resolution and the distribution of camera rotations and translations on the training samples influence the quality of the results. We also give detailed insights on the behavior of our CNN on difficult, confusable camera motions and show how different tracking errors behave over time during training.

In the future, this evaluation could be extended to more recent and more complex NN structures. Another interesting component that could easily be integrated is learning the camera intrinsics.

Chapter 7

Visualization

Examining our reconstruction results based on raw numbers or rendered images is in many cases very cumbersome because it is difficult to get an impression of the 3D structure of the observed content and of effects distributed in a scene such as the relationships between scene surface points and corresponding features on different camera images. We therefore developed an interactive, 3D, OpenGL based visualization framework for the reconstructed scenes which works quite similarly for dense, feature based and sparse, depth map based methods.

Our visualization framework (see Figures 7.1 and 7.2) runs in a separate thread and opens its own OpenGL window. This enables us to update the presented content virtually everywhere in our reconstruction process: We can either just represent the final reconstruction result at the end or update the presentation on every tiny reconstruction step such as on each newly triangulated point, each additional camera that was added or even on each separate LM iteration which allows us to get a very detailed insight in the process of the reconstruction. Furthermore, due to the fact that the visualization runs in its own thread, it stays fully responsible while the reconstruction is performed in the background and even interoperates nicely with debuggers such as GDB in non-stop mode: it is possible to stop the reconstruction thread exclusively which enables us to fly through our 3D reconstruction visually while debugging the processing on it in the code of the same program, without having to restart it.

Our visualizer shows the reconstructed scene in one big window which can be explored in free flight, WASD-style mode. It is possible to select cameras or scene points to obtain a numerical representation of selected entities. Moreover, the user can seamlessly move the observer camera to a reconstructed camera pose, seeing the captured image or features as well as the related scene reconstruction parts as captured from that camera pose. The user may also zoom in to specific image regions to see the sub-pixel precise alignment of image feature points and the corresponding 3D scene point reprojections or the subpixel alignment of warped keyframe and tracked frame image content.

There are some differences in the representation of the scene structure and the image content for sparse, feature based and for dense, depth map based methods:

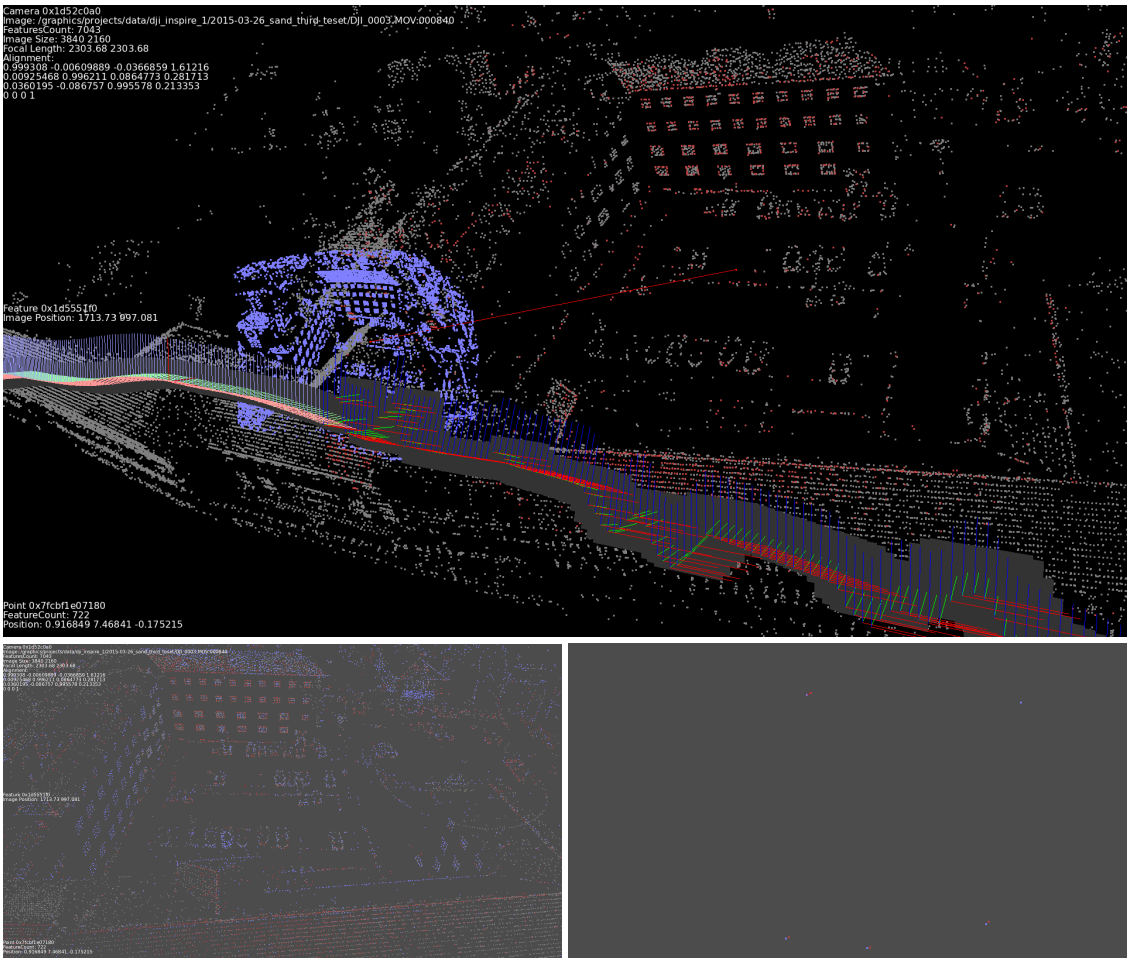


Figure 7.1: *Visualization of a sparse surface, feature based reconstruction. 3D scene points are represented in light gray or, if seen by the currently selected camera, in red. For the selected camera, the image feature points are represented in blue. Image planes are shown in dark gray. The top image shows the reconstruction in free flight mode while the bottom row contains two images from a camera's perspective, closely zoomed in on the right so that reprojection errors become visible as offset between a red scene point and its corresponding blue image feature.*

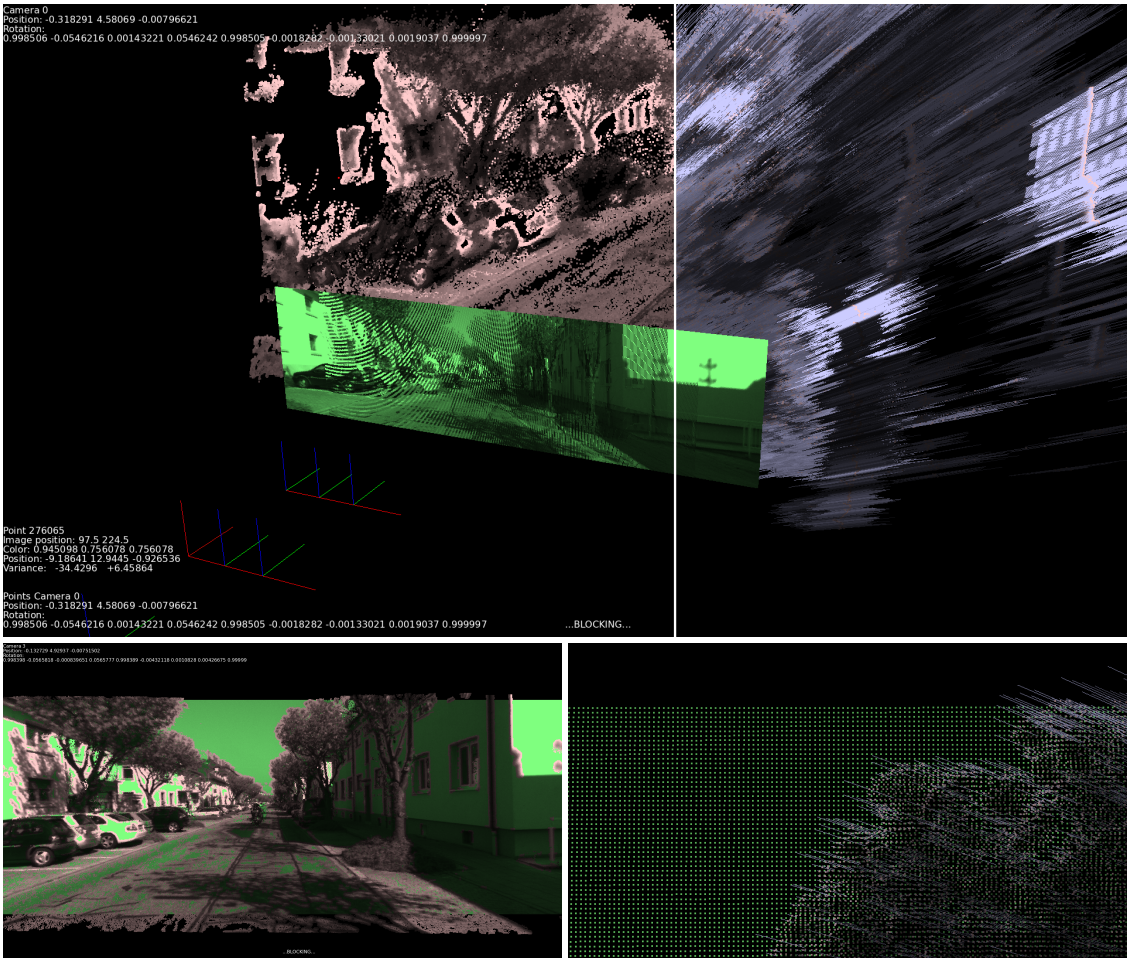


Figure 7.2: *Visualization of a dense, depth map based reconstruction. The depth maps of the keyframes are represented in red while their variances are shown in blue. The images from the tracked frames which have no depth information are rendered in green color. The top split image shows the reconstruction in free flight mode with (right) and without (left) visualized variances of the depth values. The bottom row contains two images from a camera's perspective. The left view shows that the new, green frame is nicely aligned with the warped keyframe data (red) after tracking. The closeup view on the right reveals where the warped pixels are placed exactly on the tracked frame. In addition, one can see that the variances are in the order of magnitude of the pixel size when projected to the image plane while being much larger in the pixel's direction in space (top right).*

7.1 Feature Based Visualization

For the visualization of sparse scenes (see Figure 7.1), the dimensions of the image plane are shown in front of every camera pose and the direction towards all features detected for a camera pose are visualized only when the camera is selected (to avoid cluttering the screen). In addition, all scene points are shown and marked red if they are attached to the currently selected camera pose. This enables us to quickly see the points based on which a camera's pose is determined. If a camera pose and an attached point are selected, our tool visualizes the reprojection error from point to feature as a line which can be analyzed in 3D or image space.

7.2 Depth Map Based Visualization

If depth map based reconstructions are to be visualized (see Figure 7.2), our framework allows the user to switch on or of the image content, depth content and depth variance visualization for each camera pose separately. This way one can nicely observe how the warped depth of a keyframe aligns with the content of the newest frame after tracking. Seamlessly moving forth and back between a free flight and a camera pose perspective shows impressively how small disparity variances in the image can translate to huge depth uncertainties in the 3D model.

Our visualization framework was not only a valuable tool for understanding reconstruction problems and the debugging of our algorithms. It also provided a great and interactive base for discussions about further development and ideas for improvements of our algorithms.

Chapter 8

Conclusion

In this thesis, we address the problem of robust and efficient scene geometry and camera trajectory reconstruction from different perspectives, specifically based on sparse features, dense depth map based reconstruction and machine learning. Obtaining such reconstructions based on cameras and compute power only has the potential to replace several conventional techniques.

Previous methods for 3D scene capturing or more precisely for measuring the sensor-to-surface distance in many directions include radar, lidar, active cameras based on pattern projection or time of flight, supersonic imaging and others. Camera trajectories can alternatively be estimated with classical odometry methods based on wheel ticks or robot joint angles, combinations of GPS and inertial measurement units (IMUs), external tracking of the cameras by triangulation or trilateration or by inside-out tracking based on depth data obtained as mentioned above. All of the listed methods need specialized, and partially expensive sensors while the methods we propose can be based on cellphone camera modules for a few tens of dollars and sufficient compute power. In contrast to lidar or active cameras, we can also fully utilize the resolution of today's camera hardware, producing precise camera pose estimates or depth maps with several megapixels on the fly.

With the increasing compute power of embedded platforms such as the NVIDIA Jetson PX2 and falling prices, algorithms like the ones described in this thesis will be available ubiquitously everywhere. This will enable the robust answering of questions like ”are there obstacles on the road?”, ”where is a plain, free area?”, ”would this new couch fit into my flat?”, ”how many liters has this package?”, ”what is the user looking at?” and many more robustly by cars, drones, smartphones, AR devices and others.

In this thesis, we have worked on several classes of algorithms in the field for improving their robustness, stability and efficiency which results in higher processing speed.

We investigate sparse feature matching in Chapter 3 to provide higher quality input for scene reconstruction and present an effective yet simple and fast technique for removing outliers from matches between local image features than can be applied on omnidirectional cameras and cameras producing heavily distorted images. By comparing feature orientations in view space instead of image space, we can demonstrate significant advantages in outlier removal, especially for omnidirectional cameras, which leads to speeded up and more robust 3D reconstructions.

A large part of this work is the development of a sparse scene reconstruction algorithm that can efficiently handle highly redundant input data from high resolution, high frame rate cameras (Chapter 4). We achieve this by consequent but careful subsampling and partitioning of the data that we use for Bundle Adjustment at once, joining the results by using an efficient linear camera pose estimation method.

Our work on dense reconstruction algorithms in Chapter 5 provides a robust solution for handling camera rigs which are prone to small deformations during capturing. Handling those is crucial because dense methods rely on a photometric error residual for camera tracking which is intolerant even to small, pixel-scale image offsets. Our solution constantly tracks the pose of the camera rig as well as the cameras inside the rig which allows for handling rig deformation effects reasonably during depth estimation.

Finally, we explore machine learning based reconstruction of the camera trajectory in Chapter 6. We propose a simple convolutional neural network that is able to learn Visual Odometry and train it based on ad-hoc rendered images from a game engine, focusing on the analysis of the relationship between training data properties and the resulting tracking quality.

In Chapter 7 we present our visualization framework which was helpful for visually examining and debugging our work.

8.1 Future Work

In addition to the potential improvements for each individual method proposed in summaries of the previous chapters, there are quite some possibilities to combine some of the methods: A promising candidate seems to be the integration of learned Visual Odometry into our dense reconstruction method as initialization for tracking which might allow for larger frame-to-frame disparities. It might also be beneficial to integrate the support for camera rigs into our sparse reconstruction method for densely sampled videos to gain additional stability and precision on environments with few features.

Another possible direction of research is the extension of the proposed algorithms to appearance acquisition. On the one hand, this could happen by appending full and dense geometry reconstruction followed by an acquisition of the photometric surface properties to the pipelines. Another interesting possibility is to integrate photometric parameters directly in a (semi-)dense SLAM algorithm, e.g. by handling diffuse and specular color (and other) maps side by side with the depth maps to aggregate photometric information from every frame.

Finally, a lot of work can be done on the interpretation of the data produced by our algorithms, which is in principle a huge set of camera pose and surface point samples. To make the data useful for other tasks than representing it, domain specific high level processing of the data has to be enabled. While there are already quite some publications about labeling and detection on 3D data, we believe that making such data available

with comparable ease and within the time of a simple video shot can significantly boost research on the way to many new applications.

Appendix A

Algorithm for Scalable Structure from Motion

For easier reimplementation, we give the pseudo code of our method in Algorithm 1.

Algorithm 1 High level pseudo code of the proposed method. This algorithm covers the main function "reconstruct" and a selected subset its called functions.

1: **function** reconstruct($images$, $costMatrix$)
2: $windows \leftarrow \emptyset$ ▷ *windows* only contains keyframes
3: **for all** $imageSeq \subseteq images$ **do**
4: $tracks \leftarrow$ tracking($imageSeq$)
5: $startPoses \leftarrow$ initWindowBA($imageSeq, tracks$) ▷ See Section 4.3.4
6: $windows \leftarrow windows \cup$ windowBA($startPoses, imageSeq, tracks$)
7: **end for**
8: $windows \leftarrow windows \cup$ anchorHandling($windows, costMatrix$)
9: $constraints \leftarrow$ ExtractConstraints($windows$) ▷ *constraints* only contains information about keyframes
10: $scene \leftarrow$ initPosesLinear($constraints$) ▷ *scene* only contains keyframes poses here
11: $scene \leftarrow$ optimize($scene$)
12: $scene \leftarrow$ interpolateNonKeyframePoses($scene$)
13: $scene \leftarrow$ optimize($scene$)
14: **end function**
15: ▷ ...

16: **function** tracking($images$) ▷ See Section 4.3.3
17: $allTracks \leftarrow \emptyset$
18: $activeTracks \leftarrow \emptyset$
19: **for** $image \in images$ **do**
20: **for all** $track \in activeTracks$ **do**
21: $feature \leftarrow$ findFeature($image, track.previousImagePattern$)
22: $feature \leftarrow$ refineFeature($feature, image, track.firstImagePattern$)
23: **if** $feature \neq$ **null then**
24: extendTrack($track, feature$)
25: **else**
26: $activeTracks \leftarrow activeTracks \setminus \{track\}$
27: **end if**
28: **end for**
29: $newTracks \leftarrow$ findNewKeypoints($image$)
30: $newTracks \leftarrow$ filterTooClose($newTracks, activeTracks$)
31: $activeTracks \leftarrow activeTracks \cup newTracks$
32: $allTracks \leftarrow allTracks \cup newTracks$
33: **end for**
34: **return** $allTracks$
35: **end function**
36:
37: **function** windowBA($startPoses, images, tracks$) ▷ See Section 4.3.4
38: $currentWindow \leftarrow$ createScene($startPoses$)
39: $windows \leftarrow \{currentWindow\}$
40: **while** last image not processed **do**
41: $currentWindow \leftarrow$ removeUnconfidentPoses($currentWindow$)
42: $baPoses \leftarrow$ SelectBaSubset($currentWindow$)
43: $lastConfidentPose \leftarrow$ **null**
44: **while true do**
45: $candidate \leftarrow$ pickImageLinIncreasingOffset($images$)
46: **if** IsConfident($baPoses, candidate$) **then** ▷ Includes optimizing for the candidate's pose against baPoses
47: $lastConfidentPose \leftarrow candidate$
48: **else**
49: **break**
50: **end if**
51: **end while**

▷ ...

```
52:         currentWindow ← addPose(currentWindow, lastConfidentPose)
53:         windows ← windows ∪ {currentWindow}
54:         constraints = extractConstraints(windows)
55:         constraints = filterByPoses(currentWindow.cameraPoses)
56:         currentWindow ← initPosesLinear(constraints)
57:     end while
58:     return windows
59: end function
60:
61: function anchorHandling(windows, costMatrix)          ▷ See Section 4.3.5
62:     anchorWindows ← ∅
63:     varianceMatrix ← estimateVariances(windows)
64:     samplingMatrix ← (1 − costMatrix) ∘ varianceMatrix
65:     while |anchorWindows| < maxGlobalAnchors do
66:         cameraPair ← ImportanceSampledPair(samplingMatrix)
67:         window ← bestWindowContaining(windows, cameraPair.first)
68:         anchorWindow ← addPose(window, cameraPair.second)
69:         anchorWindow ← optimize(anchorWindow)
70:         if isStable(anchorWindow, cameraPair.second) then
71:             anchorWindows ← anchorWindows ∪ {anchorWindow}
72:         end if
73:     end while
74:     return anchorWindows
75: end function
76:
77: function optimize(scene)
78:     scene.points ← findPoints(scene.cameraPoses)
79:     scene.points ← subsamplePoints(scene.points)          ▷ See Section 4.3.2
80:     scene ← BA(scene)
81:     while config says so do
82:         scene.points ← findPoints(scene.cameraPoses)
83:         scene.points ← filterBadPoints(scene.points)
84:         scene.points ← subsamplePoints(scene.points)      ▷ See Section 4.3.2
85:         scene ← BA(scene)
86:     end while
87:     return scene
88: end function
```

Appendix B

Scalable Structure from Motion - Datasets

We used the following datasets in the paper for which we show some example images and illustrations of the reconstructions.

Datasets

Ship The ship dataset consists of a video sequence with 4353 frames, recorded with a DSLR camera in 1080p. The camera was mounted on a slowly moving crane to simulate high frame rates: the camera translation from frame to frame is about 1/4 of the other test scenes in the paper, so it corresponds to the same camera velocity at about 240 fps. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.1 for some example images and a 3D visualization of the reconstruction.

Office The office dataset consists of a video sequence with 1333 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.2 for some example images and a 3D visualization of the reconstruction.

Bench The bench dataset consists of a video sequence with 1055 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.3 for some example images and a 3D visualization of the reconstruction.

Stairway The stairway dataset consists of a video sequence with 876 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.4 for some example images and a 3D visualization of the reconstruction.

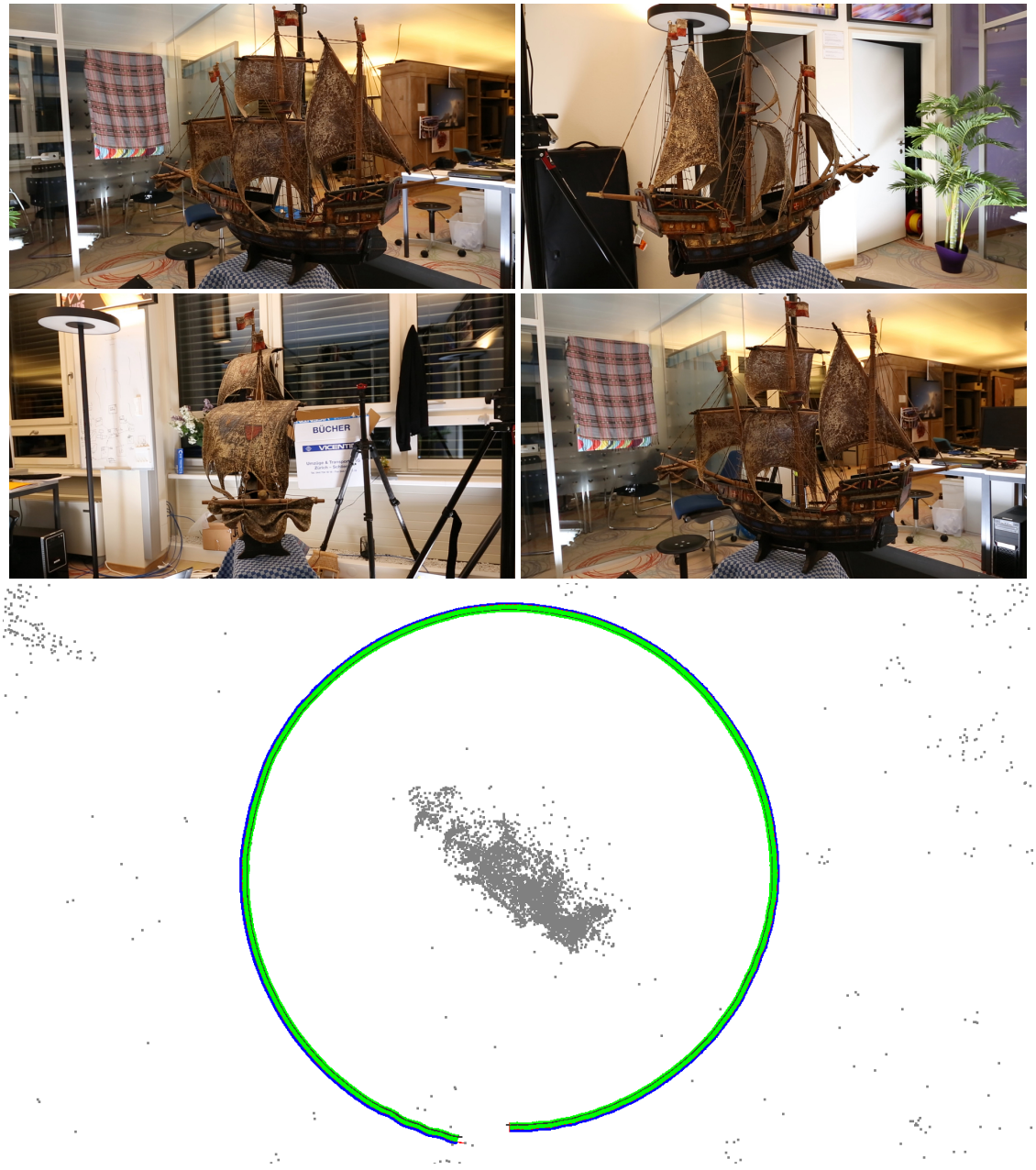


Figure B.1: *Ship scene: Input images and visualization - top-down view. Note that the loop is not closed because this scene was reconstructed without global anchor constraints to measure camera drift.*

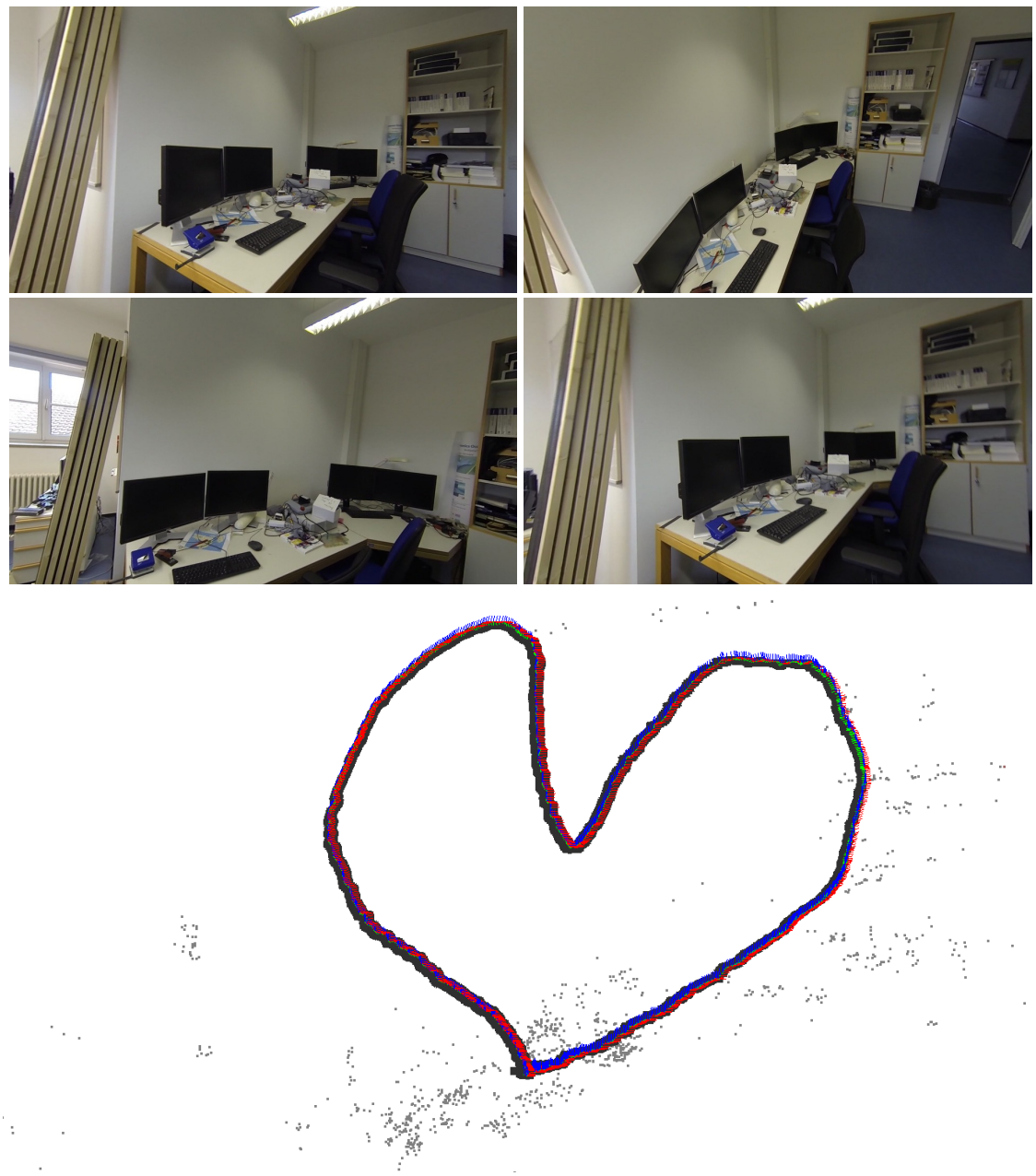


Figure B.2: *Office scene: Input images and visualization - viewing in camera direction.*

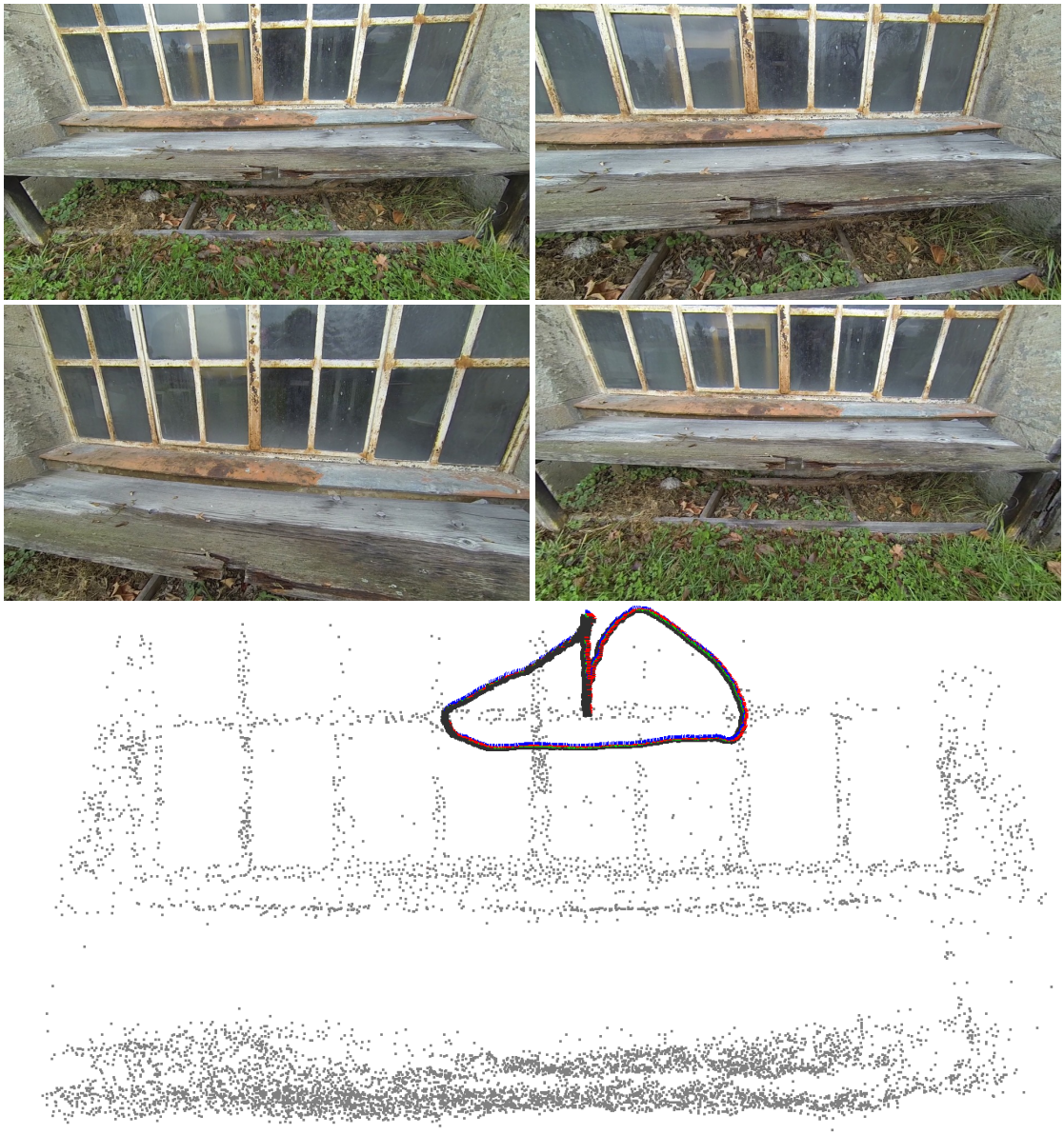


Figure B.3: *Bench scene: Input images and visualization - frontal view.*

Figure B.4: *Stairway scene: Input images and visualization - top-down view.*

Office 2 The office 2 dataset consists of a video sequence with 1180 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.5 for some example images and a 3D visualization of the reconstruction.

Swinging The swinging dataset consists of a video sequence with 1257 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. Refer to Figure B.6 for some example images and a 3D visualization of the reconstruction.

2 loops The 2 loops dataset consists of a video sequence with 1715 frames, recorded with a GoPro Hero 3 camera in 1080p 60 fps wide angle mode. The trajectory starts and stops at the same position which allows camera drift to be measured. The camera circles an object twice which allows to compare our algorithm with and without global anchor constraints between the two loops. Refer to Figure B.7 for some example images and a 3D visualization of the reconstruction.

Figure B.5: *Office 2 scene: Input images and visualization - top-down view. Note that the loop is not closed because this scene was reconstructed without global anchor constraints to measure camera drift.*

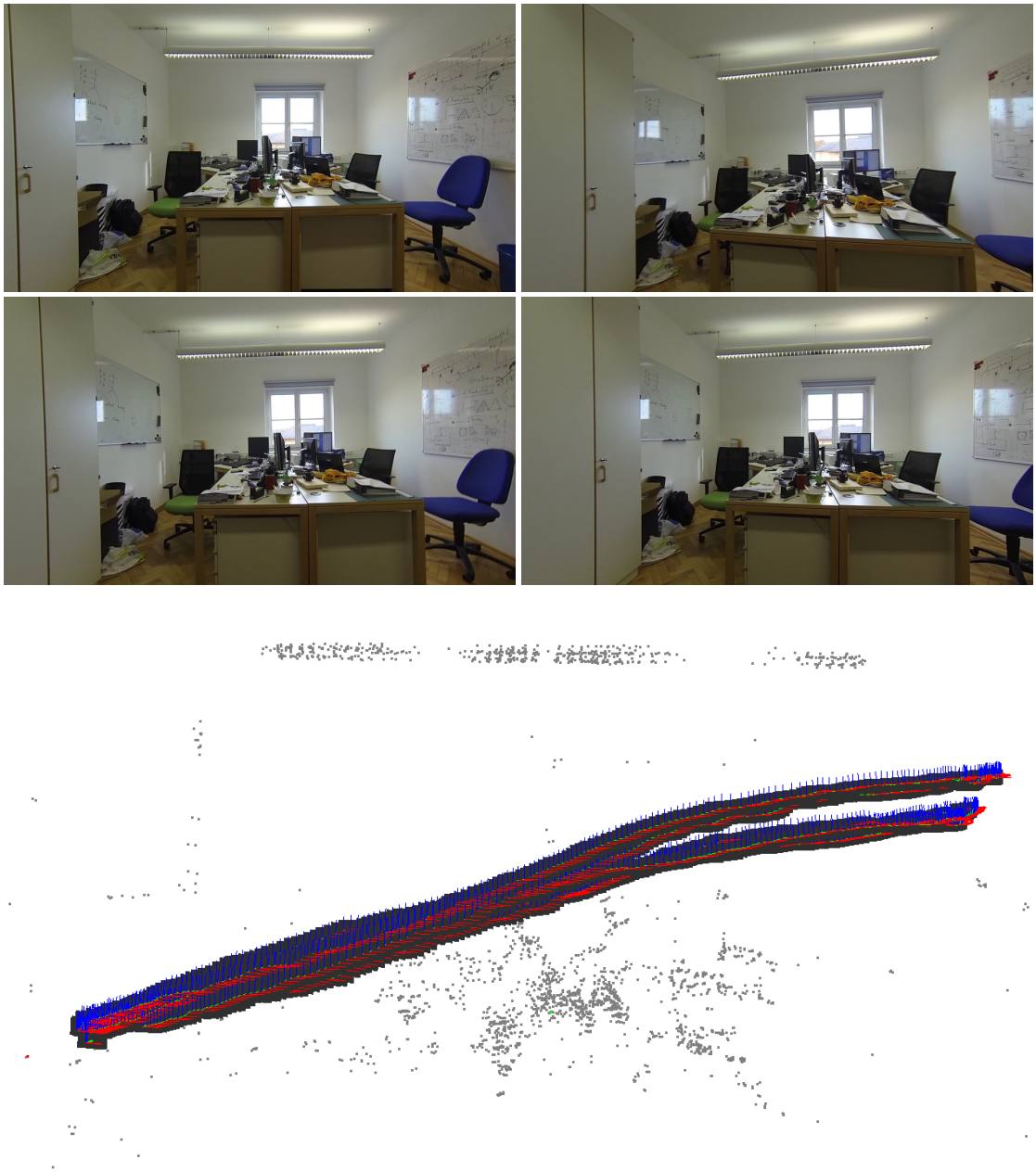


Figure B.6: *Swinging scene: Input images and visualization - viewing in camera direction.*

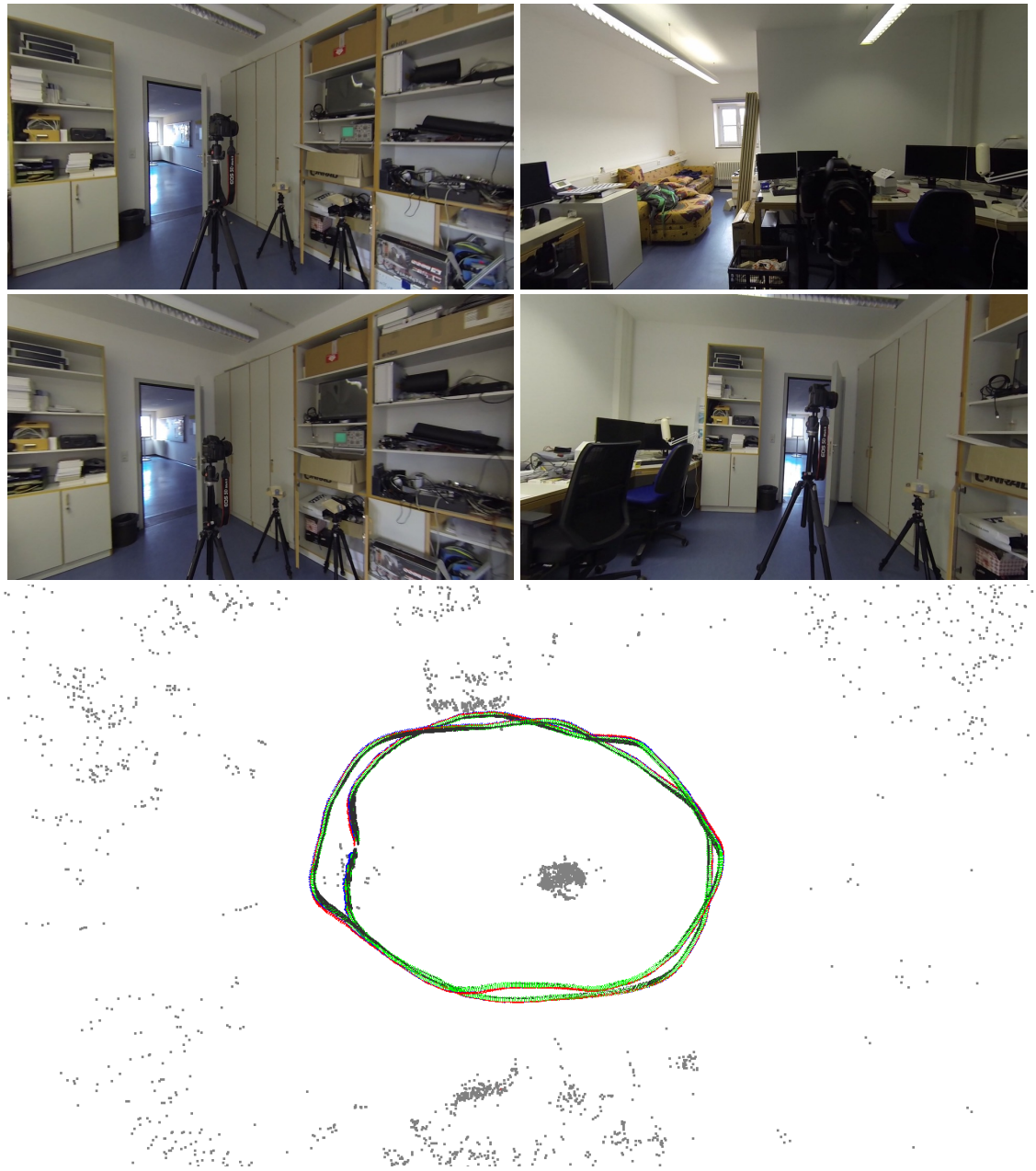


Figure B.7: *2 loops scene: Input images and visualization - top-down view.*

Appendix C

Mechanical Deflection of Camera Rigs

In this section we describe our calculations of the mechanical deflection of a camera rig which we use as motivation in our introduction.

For evaluating the rotation angle of stereo cameras against each other on a carrier (dimensions $w \times h \times d = 300 \times 4 \times 25$ mm), we treat the carrier as two cantilever beams of half the length of the carrier (since we assume it to be fixed in the middle). There are two types of force applied to the carrier when an acceleration is applied to the rig:

1. The force of the cameras connected to the endpoints of the rig and
2. the force of the mass of the rig itself.

We will treat both forces separately in the following Subsections.

Deformation Because of The Camera's Mass

Let's first have a look at the cameras. According to [Wik16], the angle of deflection ϕ_C for end-loaded cantilever beams is

$$\phi_C = \frac{FL^2}{2E_S I}. \tag{C.1}$$

F is the force acting at the tip of the beam which is according to newton's second law

$$F = ma = mg = m_C \cdot 9,81[m/s^2] \tag{C.2}$$

where a is the acceleration for which we use $1g$ which is the gravity of the earth and m_C is the mass of the camera.

$L = w/2$ is half the width w of the carrier (fixation point to end point) and E_S is the modulus of elasticity (in our case the one of stainless steel) which is a material constant. I is the area moment of inertia which represents the stiffness of a certain profile and is given for rectangular profiles as

$$I = dh^3 \tag{C.3}$$

where d is the depth of the carrier (width of the profile) and h is the height of the carrier.

Putting all equations together into Equation C.1 yields

$$\phi_C = \frac{3w^2 m_C g}{2dh^3 E_S} \tag{C.4}$$

Deformation Because of the Carrier's Mass

For the mass of the carrier itself, the equations for uniformly-loaded cantilever beams [Wik16] hold which give the angle of deflection as

$$\phi_R = \frac{qL^3}{8EI}. \tag{C.5}$$

Here, q is the force per unit length

$$q = F/L \tag{C.6}$$

with

$$F = ma = mg = m_R \cdot 9.81[m/s^2]. \tag{C.7}$$

We can calculate the mass of half of the carrier by its volume V and the density ρ_S of its material:

$$m_R = V\rho_S = Lhd\rho_S \tag{C.8}$$

Putting everything into Equation C.5 yields

$$\phi_R = \frac{g\rho_S w^3}{4h^2 E_S} \tag{C.9}$$

Summing Up Rotations

For obtaining the cameras' rotations (in radians) against each other, we sum up the contributions of the previous subsections and double the result since the cameras rotate in different directions when the whole rig is accelerated.

$$\phi = 2(\phi_C + \phi_R) \tag{C.10}$$

For our example in the introduction if the paper we use the following constants

$$w = 0,3[m] \tag{C.11}$$
$$h = 0,025[m] \tag{C.12}$$
$$d = 0,004[m] \tag{C.13}$$
$$m_C = 0.5[kg] \tag{C.14}$$
$$E_S = 180 \cdot 10^9[Pa] \tag{C.15}$$
$$\rho_S = 7900[kg/m^3] \tag{C.16}$$

which leads to

$$\phi_C = 0.0022992187 \tag{C.17}$$
$$\phi_R = 0.0001816382 \tag{C.18}$$
$$\phi = 2(\phi_C + \phi_R) = 0.0049617140 \approx 0.28°. \tag{C.19}$$

When the acceleration is applied once in positive and once in negative direction, the maximum relative rotation doubles to 0.56°.

Bibliography

[AF10] Zafer Arican and Pascal Frossard. Omnisift: Scale invariant features in omnidirectional images. In *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pages 3505–3508. IEEE, 2010.

[AFS+11] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.

[AMO17] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. `http://ceres-solver.org`, 2017.

[AOV12] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pages 510–517. Ieee, 2012.

[ASSS10] Sameer Agarwal, Noah Snavely, Steven M Seitz, and Richard Szeliski. Bundle adjustment in the large. In *European conference on computer vision*, pages 29–42. Springer, 2010.

[BDM+16] Shehroze Bhatti, Alban Desmaison, Ondrej Miksik, Nanthas Nardelli, N. Siddharth, and Philip H.S. Torr. Playing doom with slam-augmented deep reinforcement learning. In *arXiv preprint arXiv:1612.00380*, 2016.

[BFL12] Christian Bailer, Manuel Finckh, and Hendrik Lensch. Scale robust multi view stereo. *Computer Vision–ECCV 2012*, pages 398–411, 2012.

[Bou01] Jean-Yves Bouguet. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.

[Bra00] G. Bradski. Opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[BTVG06] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, pages 404–417, 2006.

[CFJS02] Alessandro Chiuso, Paolo Favaro, Hailin Jin, and Stefano Soatto. Structure from motion causally integrated over time. *IEEE transactions on pattern analysis and machine intelligence*, 24(4):523–535, 2002.

[CJZ93] W Dan Curtis, Adam L Janin, and Karel Zikan. A note on averaging rotations. In *Virtual Reality Annual International Symposium, 1993., 1993 IEEE*, pages 377–385. IEEE, 1993.

[CLSF10] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. *Computer Vision–ECCV 2010*, pages 778–792, 2010.

[CMBP+12] Javier Cruz-Mota, Iva Bogdanova, Benoît Paquier, Michel Bierlaire, and Jean-Philippe Thiran. Scale invariant feature transform on the sphere: Theory and applications. *International journal of computer vision*, 98(2):217–241, 2012.

[CMR07] Andrew I Comport, Ezio Malis, and Patrick Rives. Accurate quadrifocal tracking for robust 3d visual odometry. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 40–45. IEEE, 2007.

[CMVC16] G. Costante, M. Mancini, P. Valigi, and T. A. Ciarfuglia. Exploring representation learning with cnns for frame-to-frame ego-motion estimation. *IEEE Robotics and Automation Letters*, 1(1):18–25, Jan 2016.

[CWM+17] Ronald Clark, Sen Wang, Andrew Markham, Niki Trigoni, and Hongkai Wen. A deep spatio-temporal model for 6-dof videoclip relocalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[CWPS11] D. F. Crouse, P. Willett, K. Pattipati, and L. Svensson. A look at gaussian mixture reduction algorithms. In *Information Fusion (FUSION), 2011 Proceedings of the 14th International Conference on*, pages 1–8, July 2011.

[DIIM04] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[DLD12] Abe Davis, Marc Levoy, and Fredo Durand. Unstructured light fields. In *Computer Graphics Forum*, volume 31, pages 305–314. Wiley Online Library, 2012.

[DP14] Amaël Delaunoy and Marc Pollefeys. Photometric bundle adjustment for dense multi-view 3d modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1486–1493, 2014.

[DRMS07] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE transactions on pattern analysis and machine intelligence*, 29(6):1052–1067, 2007.

[ESC14] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[ESC15] Jakob Engel, Jörg Stückler, and Daniel Cremers. Large-scale direct slam with stereo cameras. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, pages 1935–1942. IEEE, 2015.

[FB81] Martin A Fischler and Robert C Bolles. A paradigm for model fitting with applications to image analysis and automated cartography". *Comm. ACM*, 24(6):381–395, 1981.

[FDB14] Philipp Fischer, Alexey Dosovitskiy, and Thomas Brox. Descriptor matching with convolutional neural networks: a comparison to sift. *arXiv preprint arXiv:1405.5769*, 2014.

[FFGG+10] Jan-Michael Frahm, Pierre Fite-Georgel, David Gallup, Tim Johnson, Rahul Raguram, Changchang Wu, Yi-Hung Jen, Enrique Dunn, Brian Clipp, Svetlana Lazebnik, et al. Building rome on a cloudless day. In *European Conference on Computer Vision*, pages 368–381. Springer, 2010.

[FP10] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2010.

[Gho05] Sanjib Kumar Ghosh. *Fundamentals of computational photogrammetry*. Concept Publishing Company, 2005.

[GL96] Gene Howard Golub and Charles F. Van Loan. *Matrix computations*. JHU Press, October 1996.

[Gla90] Andrew S. Glassner. *Graphics Gems*. Academic Press, Inc., Orlando, FL, USA, 1990.

[GLU12] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 3354–3361. IEEE, 2012.

[GN01] Michael D Grossberg and Shree K Nayar. A general imaging model and a method for finding its parameters. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 2, pages 108–115. IEEE, 2001.

[GRL17] Fabian Groh, Benjamin Resch, and Hendrik PA Lensch. Multi-view continuous structured light scanning. In *German Conference on Pattern Recognition*. Springer, 2017.

[H+64] Peter J Huber et al. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[Har92] Richard Hartley. Estimation of relative camera positions for uncalibrated cameras. In *Computer Vision—ECCV'92*, pages 579–587. Springer, 1992.

[HCB10] Peter Hansen, Peter Corke, and Wageeh Boles. Wide-angle visual feature matching for outdoor localization. *The International Journal of Robotics Research*, 29(2-3):267–297, 2010.

[HHA+10] Matthias B Hullin, Johannes Hanika, Boris Ajdin, Hans-Peter Seidel, Jan Kautz, and Hendrik Lensch. *Acquisition and analysis of bispectral bidirectional reflectance and reradiation distribution functions*, volume 29. ACM, 2010.

[Hir08] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2008.

[HJ92] David J Heeger and Allan D Jepson. Subspace methods for recovering rigid motion i: Algorithm and implementation. *International Journal of Computer Vision*, 7(2):95–117, 1992.

[HS88] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988.

[HZ03] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[IKH+11] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[JCT13] Nianjuan Jiang, Zhaopeng Cui, and Ping Tan. A global linear method for camera pose registration. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 481–488, 2013.

[K+60] Rudolph Emil Kalman et al. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.

[KB14] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KC16] Alex Kendall and Roberto Cipolla. Modelling uncertainty in deep learning for camera relocalization. *Proceedings of the International Conference on Robotics and Automation (ICRA)*, 2016.

[KC17] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[KGC15] Alex Kendall, Matthew Grimes, and Roberto Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.

[KM07] Georg Klein and David Murray. Parallel tracking and mapping for small ar workspaces. In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pages 225–234. IEEE, 2007.

[KM15] Kishore Reddy Konda and Roland Memisevic. Learning visual odometry with a convolutional network. In *VISAPP (1)*, pages 486–490, 2015.

[KS99] Dong Sik Kim and Ness B Shroff. Quantization based on a novel sample-adaptive product quantizer (sapq). *IEEE Transactions on Information Theory*, 45(7):2306–2320, 1999.

[KSC13] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 3748–3754. IEEE, 2013.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[KTT+12] Kwang In Kim, James Tompkin, Martin Theobald, Jan Kautz, and Christian Theobalt. Match graph construction for large image databases. In *European Conference on Computer Vision*, pages 272–285. Springer, 2012.

[KZP+13] Changil Kim, Henning Zimmer, Yael Pritch, Alexander Sorkine-Hornung, and Markus H Gross. Scene reconstruction from high spatio-angular resolution light fields. *ACM Trans. Graph.*, 32(4):73–1, 2013.

[LA09] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009.

[LBG80] Yoseph Linde, Andres Buzo, and Robert Gray. An algorithm for vector quantizer design. *IEEE Transactions on communications*, 28(1):84–95, 1980.

[LBV12] Miguel Lourenço, Joao P Barreto, and Francisco Vasconcelos. srd-sift: Keypoint detection and matching in images with radial distortion. *IEEE Transactions on Robotics*, 28(3):752–760, 2012.

[LCS11] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[Lev44] K. Levenberg. A method for the solution of certain non-linear problems in least squares. *Quarterly Journal of Applied Mathmatics*, II(2):164–168, 1944.

[LH81] H Christopher Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293(5828):133–135, 1981.

[Li06] Hongdong Li. A simple solution to the six-point two-view focal-length problem. In *European Conference on Computer Vision*, pages 200–213. Springer, 2006.

[LLG+17] R. Li, Q. Liu, J. Gui, D. Gu, and H. Hu. Indoor relocalization in challenging environments with dual-stream convolutional neural networks. *IEEE Transactions on Automation Science and Engineering*, PP(99):1–12, 2017.

[LM13] Mingyang Li and Anastasios I Mourikis. High-precision, consistent ekf-based visual-inertial odometry. *The International Journal of Robotics Research*, 32(6):690–711, 2013.

[Low04] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[LSC+15] Hyon Lim, Sudipta N Sinha, Michael F Cohen, Matt Uyttendaele, and H Jin Kim. Real-time monocular image-based 6-dof localization. *The International Journal of Robotics Research*, 34(4-5):476–492, 2015.

[LW08] Lingling Lu and Yihong Wu. Quasi-dense matching between perspective and omnidirectional images. In *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications-M2SFA2 2008*, 2008.

[MC13] Maxime Meilland and Andrew I Comport. On unifying key-frame and voxel-based dense visual slam at large scales. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 3677–3683. IEEE, 2013.

[MDFF16] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2574–2582, 2016.

[MG15] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.

[MKRH11] Rafat Mantiuk, Kil Joong Kim, Allan G Rempel, and Wolfgang Heidrich. Hdr-vdp-2: A calibrated visual metric for visibility and quality predictions in all luminance conditions. In *ACM Transactions on Graphics (TOG)*, volume 30, page 40. ACM, 2011.

[MKS+13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*, 2013.

[ML14] Marius Muja and David G Lowe. Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11):2227–2240, 2014.

[MP07] Daniel Martinec and Tomas Pajdla. Robust rotation and translation estimation in multiview reconstruction. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE, 2007.

[MS05] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10):1615–1630, 2005.

[MTS+05] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, Jiri Matas, Frederik Schaffalitzky, Timor Kadir, and Luc Van Gool. A comparison of affine region detectors. *International journal of computer vision*, 65(1-2):43–72, 2005.

[NB10] Heiko Neumann and Pierre Bayerl. Bildverarbeitung und bewegtbildanalyse. Seminar, February 2010.

[Ng06] Ren Ng. *Digital light field photography*. Stanford University California, 2006.

[Nis04] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):756–770, 2004.

[NLD11] Richard A Newcombe, Steven J Lovegrove, and Andrew J Davison. Dtam: Dense tracking and mapping in real-time. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2320–2327. IEEE, 2011.

[NNB04] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee, 2004.

[OFKS13] Luc Oth, Paul Furgale, Laurent Kneip, and Roland Siegwart. Rolling shutter camera calibration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1360–1367, 2013.

[OT02] Aude Oliva and Antonio Torralba. Scene-centered description from spatial envelope properties. In *Biologically motivated computer vision*, pages 263–272. Springer, 2002.

[PPTN08] Lina M Paz, Pedro Piniés, Juan D Tardós, and José Neira. Large-scale 6-dof slam with stereo-in-hand. *IEEE transactions on robotics*, 24(5):946–957, 2008.

[PRL16] Sudeep Pillai, Srikumar Ramalingam, and John J Leonard. High-performance and tunable stereo reconstruction. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 3188–3195. IEEE, 2016.

[PS12] Roman Parys and Andreas Schilling. Incremental large scale 3d reconstruction. In *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2012 Second International Conference on*, pages 416–423. IEEE, 2012.

[PVGV+04] Marc Pollefeys, Luc Van Gool, Maarten Vergauwen, Frank Verbiest, Kurt Cornelis, Jan Tops, and Reinhard Koch. Visual modeling with a hand-held camera. *International Journal of Computer Vision*, 59(3):207–232, 2004.

[RD06] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, pages 430–443, 2006.

[RLL14] Benjamin Resch, Jochen Lang, and Hendrik PA Lensch. Local image feature matching improvements for omnidirectional camera systems. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 918–923. IEEE, 2014.

[RLW+15] Benjamin Resch, Hendrik Lensch, Oliver Wang, Marc Pollefeys, and Alexander Sorkine-Hornung. Scalable structure from motion for densely sampled videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3936–3944, 2015.

[RM04] Laura Walker Renninger and Jitendra Malik. When is scene identification just texture recognition? *Vision research*, 44(19):2301–2311, 2004.

[Roo10] Roosendaal, Ton. Sintel. Blender Foundation, Durian Open Movie Project. `http://www.sintel.org/`, 2010.

[Ros99] Paul L Rosin. Measuring corner properties. *Computer Vision and Image Understanding*, 73(2):291–307, 1999.

[RRKB11] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE international conference on*, pages 2564–2571. IEEE, 2011.

[RVRK16] Stephan R. Richter, Vibhav Vineet, Stefan Roth, and Vladlen Koltun. Playing for data: Ground truth from computer games. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *European Conference on Computer Vision (ECCV)*, volume 9906 of *LNCS*, pages 102–118. Springer International Publishing, 2016.

[RWL16] Benjamin Resch, Jian Wei, and Hendrik PA Lensch. Real time direct visual odometry for flexible multi-camera rigs. In *Asian Conference on Computer Vision*, pages 503–518. Springer, 2016.

[S+94] Jianbo Shi et al. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[SCMS08] Davide Scaramuzza, Nicolas Criblez, Agostino Martinelli, and Roland Siegwart. Robust feature extraction and matching for omnidirectional images. In *Field and Service Robotics*, pages 71–81. Springer, 2008.

[SF11] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE robotics & automation magazine*, 18(4):80–92, 2011.

[SFPL10] Joaquim Salvi, Sergio Fernandez, Tomislav Pribanic, and Xavier Llado. A state of the art in structured light patterns for surface profilometry. *Pattern recognition*, 43(8):2666–2680, 2010.

[SI07] Christian Siagian and Laurent Itti. Rapid biologically-inspired scene classification using features shared with visual attention. *IEEE transactions on pattern analysis and machine intelligence*, 29(2):300–312, 2007.

[SK94] Richard Szeliski and Sing Bing Kang. Recovering 3d shape and motion from image streams using nonlinear least squares. *Journal of Visual Communication and Image Representation*, 5(1):10–28, 1994.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[SSS06] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[SSS08] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[TD11] M. Trepel and K. Dalkowski. *Neuroanatomie: Struktur und Funktion*. Elsevier Health Sciences Germany, 2011.

[TK91] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical report, International Journal of Computer Vision, 1991.

[TMFR03] A. Torralba, K. P. Murphy, W. T. Freeman, and M. A. Rubin. Context-based vision system for place and object recognition. In *Proceedings Ninth IEEE International Conference on Computer Vision*, pages 273–280 vol.1, Oct 2003.

[TMHF99] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms*, pages 298–372. Springer, 1999.

[UZU+17] B. Ummenhofer, H. Zhou, J. Uhrig, N. Mayer, E. Ilg, A. Dosovitskiy, and T. Brox. Demon: Depth and motion network for learning monocular stereo. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[W+11] Changchang Wu et al. Visualsfm: A visual structure from motion system. 2011.

[Wik16] Wikipedia. Deflection — Wikipedia, the free encyclopedia, 2016. [Online; accessed 06-September-2016].

[WJV+05] Bennett Wilburn, Neel Joshi, Vaibhav Vaish, Eino-Ville Talvala, Emilio Antunez, Adam Barth, Andrew Adams, Mark Horowitz, and Marc Levoy.

High performance imaging using large camera arrays. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 765–776. ACM, 2005.

[WRL14] Jian Wei, Benjamin Resch, and Hendrik PA Lensch. Multi-view depth map estimation with cross-view consistency. In *BMVC*, 2014.

[WRL16] Jian Wei, Benjamin Resch, and Hendrik PA Lensch. Dense and occlusion-robust multi-view stereo for unstructured videos. In *Computer and Robot Vision (CRV), 2016 13th Conference on*, pages 69–76. IEEE, 2016.

[WRL17] Jian Wei, Benjamin Resch, and Hendrik PA Lensch. Dense and scalable reconstruction from unstructured videos with occlusions. In *VMV*, 2017.

[WS14] Kyle Wilson and Noah Snavely. Robust global translations with 1dsfm. In *European Conference on Computer Vision*, pages 61–75. Springer, 2014.

[WSZ+14] Oliver Wang, Christopher Schroers, Henning Zimmer, Markus Gross, and Alexander Sorkine-Hornung. Videosnapping: Interactive synchronization of multiple videos. *ACM Transactions on Graphics (TOG)*, 33(4):77, 2014.

[WWSHL16] Patrick Wieschollek, Oliver Wang, Alexander Sorkine-Hornung, and Hendrik Lensch. Efficient large-scale approximate nearest neighbor search on the gpu. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2027–2035, 2016.

[YG14] Fisher Yu and David Gallup. 3d reconstruction from accidental motion. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3986–3993, 2014.

[ZBSL17] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G. Lowe. Unsupervised learning of depth and ego-motion from video. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[ZGW+14] Chenxi Zhang, Jizhou Gao, Oliver Wang, Pierre Georgel, Ruigang Yang, James Davis, Jan-Michael Frahm, and Marc Pollefeys. Personal photograph enhancement using internet photo collections. *IEEE transactions on visualization and computer graphics*, 20(2):262–275, 2014.

[ZT13] Danping Zou and Ping Tan. Coslam: Collaborative visual slam in dynamic environments. *IEEE transactions on pattern analysis and machine intelligence*, 35(2):354–366, 2013.