ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Olivier Steiger

Smart camera for MPEG-7

Diploma project

Advisors: Andrea Cavallaro
Professor Touradj Ebrahimi

Lausanne, EPFL
2001

Acknowledgments

First of all, I would like to thank Andrea Cavallaro for the competent and stimulating advices he gave me during this whole diploma project. He somehow managed to assist me efficiently without “controlling” my work too much. Humanly and technically, it was a great pleasure to work with him.

I am also grateful to Prof. Touradj Ebrahimi for the technical insight he gave me before and during this project; our regular meetings helped me to keep track while inspiring new ways to improve the project.

To the professor Murat Kunt and his staff at the EPFL Signal Processing Laboratory, I own my presence there. Many thanks go to all of them.

A special thank goes to Professor Tsuhan Chen at Carnegie Mellon University, Pittsburgh. He was the first one who showed me the fascinating world of image processing. Maybe I would not have followed this path without him.

The biggest contribution to this work however comes from my parents, family and friends: they gave me the love and support that made this and many other things possible. Thank you!

Contents

List of Figures

List of Tables

Abbreviated terms

CD: Committee Draft
D: Descriptor
DCT: Discrete Cosine Transform
DDL: Description Definition Language
DS: Description Scheme
DVD: Digital Versatile Disc
FCM: Fuzzy C-means clustering
HDTV: High-Definition Television
MDS: Multimedia Description Schemes
MPEG: Moving Picture Experts Group
V: Visual
VO: Video Object
VOP: Video Object Plane
WD: Working Draft
XM: eXperimentation Model
XML: Extensible Markup Language

Abstract

While a first generation of video coding techniques proposed to remove the redundancies in and between image frames to get smaller bitstreams, second generation schemes like MPEG-4 and MPEG-7 aim at doing content-based coding and interactivity. To reach this goal, tools for the extraction and description of semantic objects need to be developed. In this work, we propose an algorithm for the extraction and tracking of semantic objects and an MPEG-7 compliant descriptor set for generic objects; together, they can be seen like a *smart camera* for automatic scene description. Some parts of the proposed system will be tested by software.

The tracking algorithm has been laid out so as to follow generic objects in scenes including partial occlusions and merging. To do this, we first localize each moving object of the scene using a change-detection mask. Then, a certain number of representative points called *centroids* is given to the objects by a fuzzy C-means algorithm. For each centroid of some current frame, we try to find the closest centroid in the previous frame. Once we found these pairs, each object can be labelled according to its corresponding previous centroids.

The description structure is a subset of the DDL language used in MPEG-7. The main concern was to find a simple, but flexible descriptor set for generic objects. A corresponding C-structure for software implementations is also proposed and partially tested.

Chapter 1

Introduction: the MPEG-7 camera

During the whole second part of the twentieth century, mankind proceeded to digitize whatever it had written, composed or painted once. But while no particular effort was required to store even huge text documents, sound and still images remained a challenge for a long time, and it was hardly possible to dream about digital video thirty years ago. Around 1970, however, there was an explosion of interest in the field of image processing and analysis which lent, about a decade later, to the first mature image data compression techniques. But while still-image algorithms like JPEG performed rather well, the first video compressors were rudimentary since they processed the stream like a bunch of still images (MJPEG). It was soon noticed how big the savings would be if one was concerned about the links between the images of a sequences. This funded a first generation of video compression techniques, including the well-known H.261 / H.263 and the first two MPEG standards. It is thanks to these techniques that we can access virtually any video document on digital media like DVD's or the internet today.

However, the storing and processing efficiency can be further increased if one adds some scene understanding to the video system; also, new possibilities for automatic image classification and retrieval emerge. This is why a second coding generation has been studied since 1985 and is still under standardization. The keyword for those new techniques is *content-based interactivity*, which means that video is not anymore processed in terms of pixels and frames, but of its content. In a news sequence for example, we would encode the background only once and then send a "talking head", or even just its moving parts like the mouth and the eyes. This does not only save bits, but it facilitates compositing and other visual effects too. For these techniques to be efficient however, semantically meaningful information has to be extracted from the input stream with as few human input as possible; therefore, advanced image analysis and machine vision tools are needed. Today, the automatic description of even simple scenes remains an unanswered challenge.

Starting with this existing coding knowledge, we would like to build a system to describe generic video scenes in an MPEG-7 compliant format. This means that semantic objects have to be extracted from the input stream, tracked over time and finally described and stored textually. The whole system forms a smart camera which could be used for automatic video surveillance, as an interface with virtual realities or for coding.

1.1 Objectives and starting point

Some months ago, a camera which is able to describe the position, color and motion of simple objects moving in front of a uniform background has been presented to the public. Interacting objects and camera movements were not supported, but the processing happened in real-time. While the applications field for such a system is restricted, it is an interesting example of a machine that “understands” what it sees to some degree. If complex situations including interacting or occluded objects and camera movements could be handled, the applications would be countless. Such a surveillance camera may detect and identify fire, intrusions or speeding automatically; a similar system may also serve as an interface between actors and computer-generated realities. And of course, TV studios could generate low-bitrate video streams in an automated way. The generated descriptions can then be stored and retrieved efficiently using some usual text-based search engine…

In our work, we would like to go one step away from the existing real-time camera towards a system capable of describing generic scenes with some object deformations and interactions like split/merge cycles and partial occlusions. We will not however support moving cameras. Such improvements may be part of a future work and will be briefly addressed at the end of this report. The camera must describe the scenes in an MPEG-7 compliant way (XML text document) and has to be fully automatic. While we would like to keep its theoretical conception as general as possible, some specific aspects will be validated by software, with processing speed not an issue. The developed algorithms are mainly intended to be enhanced by semantic databases, self-learning algorithms, human input or other technologies to form a truly generic smart camera.

Since this subject has already been explored at the EPFL Signal Processing Laboratory, we will start with a software that was specifically designed for highway surveillance [1]. This program detects and tracks cars very accurately, but is not able to handle merging or deforming objects in a satisfying way, because it processes objects as one solid entity without any subparts. However, its optical flow based image segmentation and the object description by centroids makes it very interesting for our own work. Many of the underlying theoretical aspects like fuzzy C-means clustering and optical flow estimation were studied by Roberto Castagno and published in 1998 [2, 3]. To get an insight into the traditional image and signal processing techniques that he may encounter in this report (mathematical morphology, filtering), the interested reader should refer to standard works like [4, 5].

1.2 Organization of the report

Since our work is closely related to the emerging video coding techniques, we begin with a historical overview of the MPEG standards and show how they may serve our camera. Then, we go into the localization and tracking of the objects; after a detailed summary of the existing highway algorithm, the new improvements as well as their potential limitations will be described. Since most of the physical features like color, texture or motion are extracted at this stage, this will also be addressed. The aim of the improvements is to propose a system for the localization and tracking of generic data filmed by a non-moving camera; since the software implementation may bring some difficulties of its own, those will be discussed.

The next chapter is about the textual description of video scenes. Since we know our target application, a subset of the standardized MPEG-7 descriptors will be proposed to fit the data provided by the localization & tracking stage, but this set is again chosen in a way allowing easy extensions or generalizations. To facilitate the software implementation, a computer specific data-storage structure will be proposed; the sharp separation between the data storage and the textual description allows the replacement of our MPEG-7 output module by any other language.

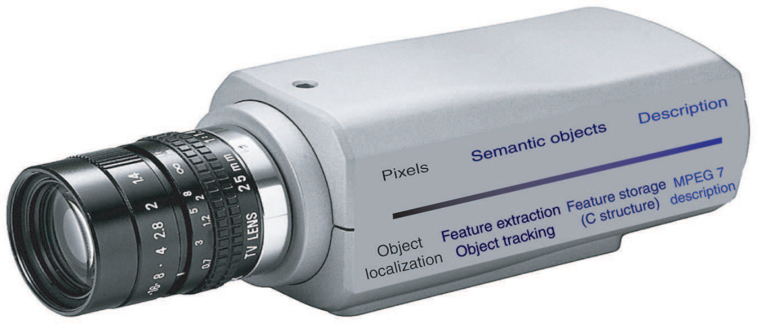


Figure 1.1: The MPEG-7 camera

Finally, our proposition will be validated with different video sequences. Limitations and further improvements will be discussed.

This structure follows the natural path of the visual information through the camera: after its digitization, which is not part of the present work, semantic objects are identified and tracked over time. Their physical description is stored in a specific C-structure and finally translated to DDL or some other description language by an output stage. This data flow is shown in a detailed block diagram of the camera which can be found in appendix A.

Chapter 2

Video coding standards

By standardizing video decoders for diverse applications and making them widely available, the ITU and ISO groups, with their respective H. and MPEG standards, contributed notably to the spread of digital video. An apposite by-product of this process is the development of new image analysis and machine vision techniques. In deed, the accompanying core experiments often ask for innovations to argue in favor, or against, some contribution; they are mostly performed with a long-term vision that is potentially beneficial for many industrial products. Therefore, a short compression history allows us to recall some important video processing milestones and outlines some links between our camera and the actual coding techniques.

2.1 Pixel based coding: the first generation

Even though compression ratios of 200 can be achieved by first generation techniques, the underlying idea is very simple: minimize spatial and temporal redundancies by a combination of pixel-based coding schemes like transform coding, predictive coding, vector quantization or subband coding. The widely used **MPEG-1** (low-quality video, e.g., internet) standard provides a comprehensive example of such techniques.

In a first step, the video stream is subdivided into spatially encoded intra-frames (I), predicted (P) and interpolated (B) frames. Each frame gets further subdivided into 16 x 16 pixel blocks. The intra frames are encoded exactly like a JPEG image: each block is decorrelated by the *discrete cosine transform* (DCT). The resulting coefficients are then quantized according to the human perception: more for high frequencies, not much for low ones. This is the stage where data get lost. Afterwards, the coefficients are zig-zag scanned and entropy encoded. Since I–frames do not depend on other frames, they can be fully reconstructed even if some data loss affected the previous frames; therefore, they are often called *keyframes*.

On the other hand, P–frames are partially reconstructed by moving each 16 x

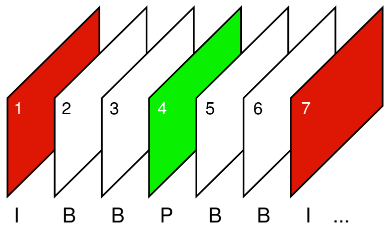


Figure 2.1: MPEG image frames

16 block of the previous I-frame according to some *motion vector*. The difference between the motion-compensated and the actual frame is intra-coded again and usually results in a very short stream. To get the motion vectors, each I–frame block is compared to the current frame in some limited range. The vector with the shortest mean-square distance is kept.

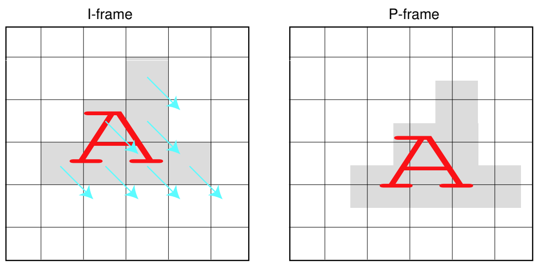


Figure 2.2: Motion compensation

B–frames finally are bidirectionally interpolated between the P– and the I–frames. They achieve the biggest compression ratio, but require the prior decoding of the I– as well as P–frame. While the exclusive use of I-frames results in MJPEG,

$$(IBBPBBPBB)(IBBPBBPBB)$$

has proved to be an efficient sequence in many situations.

Even though they were designed for higher image quality/lower bit rates and support features like half-pixel motion compensation, interlaced video or the 4:4:4 color format, the newer **MPEG-2** (HDTV, DVD) or **H.263** (very low bit rate coding) standards are essentially similar to the previous example.

2.2 Content-based coding

Second generation coding differs from the previous technology in that it does not just decompose the images in square blocks, but rather tries to extract *semantically* *meaningful* information from them. The resulting content-based video streams are not only smaller usually, but also easier to process than the pixel-based ones. We can imagine a scene with some static background and moving actors: if the actors are encoded separately, it is easy to remove one of them or to change the background. Also, when block-based motion compensation is used, the background pixels that fall in a moving block have to be re-encoded. This is not the case with content-based video. To illustrate some of the new visual concepts, we will have a closer look at **MPEG-4** [6].

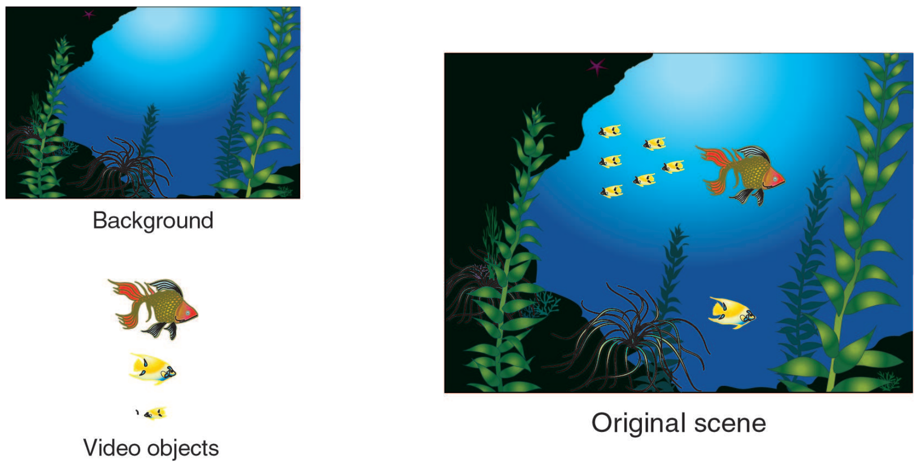


Figure 2.3: Content-based video (example)

As before, the video stream is divided into I–, P– and B–frames. But instead of forming square blocks, we try to get semantically meaningful entities called *video object planes*; each one of those VOP's is a snapshot of the corresponding *video object* (s. figure 2.3). Since such objects can be arbitrarily shaped, the usual DCT-Huffmann encoding cannot be used anymore. Instead, shapes are encoded using context-based arithmetic encoding and textures by a shape-adaptative DCT. The detail of those techniques is beyond the scope of this work, but is thoroughly explained in the MPEG-4 video verification model [7]. Motion is again encoded by motion vectors, with the block-based matching techniques having been extended to arbitrary shapes. The full image is finally reconstructed by VO composition.

But while it is easy for a video decoder to reconstruct an MPEG-4 stream, no encoder gets the right objects out of *any* input sequence so far. In fact, this requires full scene understanding by the machine with limited human support. The purpose of one part of our work is precisely to get semantic objects out of generic video scenes, so it can be thought of as of a partial MPEG-4 encoder. It has to be reminded here that MPEG-4 includes many other aspects which will not be addressed.

The second part of our work addresses the description of the scene, which can again be illustrated by the means of an emerging standard.

2.3 Multimedia databases and MPEG-7

The permanent growth of publicly available data makes it increasingly difficult to find the wanted information. Powerful search engines have been developed for text documents, but there is no such thing for images or audio so far. In deed, it is necessary not only to extract, but also to be able to describe the image content for retrieval. The recent MPEG-7 work aims at a standardized description of moving pictures and other image documents. To fulfill this goal, the MPEG has standardized a set of *descriptors* and *description schemes* which are independent of the actual video encoding format (MPEG, JPEG, analogue movie, ...), but can be extended at will thanks to the description definition language DDL. Essential object characteristics like shape, texture or motion as well as many general features (sequence name, location, links, etc.) can be described textually and retrieved or matched with other descriptions, for example one of a sketch, by usual text-based search engines. Image analysis is also facilitated a lot since it is not necessary anymore to go back to the actual video data to get some visual information.

This is precisely the interesting aspect of content description for our project. In deed, if a camera is able to describe fire, intruders or speeding cars so that they can be matched with a generic description of a similar object, fully automated surveillance becomes possible. And in a gaming application, the relevant features of the player could be filtered automatically and sent alone, sparing the costly video transmission. So we will try to select an appropriate subset of the wide descriptor selection proposed by the MPEG for our application. The goal is to describe any kind of object without too much overhead and with no necessity for computationally intensive choices, for example of *which* color descriptor to choose.

So the smart camera is an illustrative example of recent video coding techniques and of their practical applications.

Chapter 3

Localization and tracking of objects

For a video scene to be described, its basic building blocks, the semantic objects, have to be localized and tracked over time. This is a challenging task, because physical objects are normally not homogeneous with respect to low-level features such as color, texture or optical flow. In this chapter, we propose an algorithm that aims at the automatic segmentation and tracking of scenes including partial occlusions and interacting objects. Since object features have to be extracted for the chosen method to work, this issue will be addressed too.

Over the last years and in part thanks to MPEG-4, many segmentation and tracking techniques have been proposed. Classical methods that are based on change detection masks combined with an affine or perspective motion model generally fail to produce semantically meaningful objects and do not provide local motion since a single parametric motion model is used. Meier and Ngan [8] successfully got round those difficulties by using a morphological motion filter together with a Hausdorff object tracker. However, their work does not track interacting objects side-by-side. Even more sophisticated algorithms such as the semi-automatic segmentation presented in [9] use 2-D mesh-based object tracking to follow the boundary, motion and intensity variations of mildly deforming objects. But while this method accounts for occlusion, the object boundaries have to be marked manually on some keyframes.

Our work combines an existing change detection mask and object clustering system [1] with a new centroid tracker. The goal is to track partially occluded objects accurately. Also, we would like to follow objects that merge and evolve together independently. The new algorithm will be validated with video scenes showing people crossing each other or playing volleyball together in front of a simple background. The highway sequence used in the previous work will also be used.

3.1 The existing algorithm

As mentioned, we start with a program that was used to find and track cars in highway scenes. This system shows good performance on the scenes it was made for, but looses track on more complicated videos. Its functionality can be subdivided into three parts: the object-oriented segmentation, the clustering and the object tracking. Each one of those parts shall be detailed now.

3.1.1 Object oriented segmentation: the object mask

The purpose of object-oriented image segmentation is to distinguish objects that could be of interest from the background. To do so, the software uses a change-detection mask: the current frame is compared with the previous as well as with the next one. Pixels that change their value between those frames are considered to be part of an interesting object and adequately labelled. Each closed region of the change detection mask is then given a unique number in an *object mask*. Obviously, this technique is limited to moving objects, but it could be easily replaced by some other algorithm relying on an object database for example.

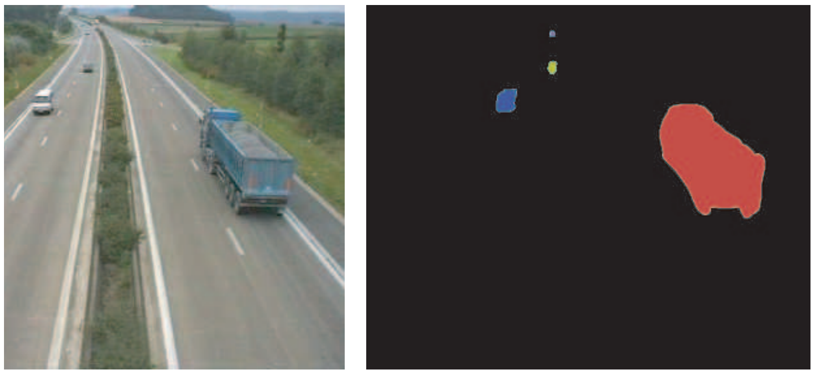


Figure 3.1: A highway scene and its object mask

Also, some pixels may change their value even though they do not belong to an interesting image part, so post-processing techniques are applied to reduce the errors. First, **shadows** are removed by a simple threshold method (s. figure 3.2): each object is scanned horizontally; the gray levels of the first and the last scanned pixels are used to form a threshold line. The pixels located between the reference pixels which have a lower luminance value than the threshold line (and some security margin δ) are then considered as shadow pixels. This operation is repeated vertically and pixels marked as shadows in both scans are removed from the object mask. Unfortunately, object pixels which are darker than the threshold line are removed from the mask too. These problems can be avoided with more sophisticated methods. Stauder et al. for example make assumptions on the unknown

3-D geometry of the scene to model and track shadows independently [10]. Such improvements however are beyond the scope of this work.

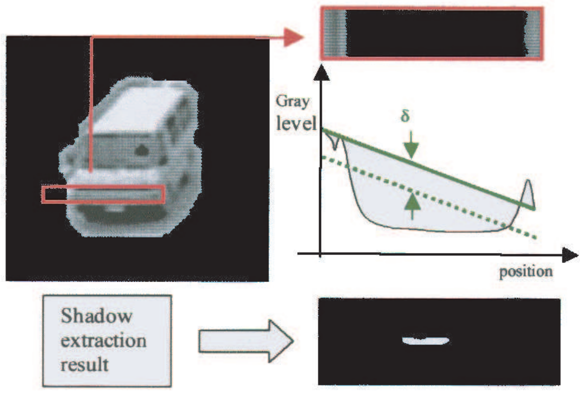


Figure 3.2: Shadow extraction principle

A **background** image can also help to get better results. If it is compared with the extracted objects, similar parts can be removed from the object mask. Also, objects that are smaller than some predefined threshold size may be suppressed.

3.1.2 Feature extraction and centroids: the object clustering

To get a representative description of the objects, the highway software uses centroid-based clustering. Its purpose is to subdivide each object in one or more subregions called *clusters*, which have homogeneous characteristics with respect to some predefined features. Each cluster is represented by a *centroid vector* $\mathbf{v}_j$, with j the index related to the features, representing its "gravity center". The number of clusters c per object was chosen so as to assign more centroids to large objects than to small ones. The empirically chosen formula is

$$\boxed{c = M(1 - e^{\frac{N}{300}})} \tag{3.1}$$

where M is the maximum admissible amount of centroids and N the number of pixels for one object.

To get the clusters, a variant of the hard C-means algorithm called **Fuzzy C-Means clustering** (FCM) is used [3]. This algorithm aims at evaluating the partition that minimizes the functional J expressed by

$$\boxed{J_{FCM}(U, \mathbf{v}) = \sum_{k=1}^{N} \sum_{i=1}^{c} u_{ik}^{m} (d_{ik})^2} \tag{3.2}$$

with N the number of pixels in the object and c the desired number of clusters. U is called the *membership matrix* and u_{ik} represents the degree of belongingness of the feature vector $\mathbf{f}_k$ to the class i; $u_{ik} \in [0;1]\ \forall i,k$ and $\sum_{i=1}^{c} u_{ck} = 1\ \forall k$. The weighting exponent $m \in [1,\infty)$ controls the amount of fuzziness, $m = 1$ corresponding to the hard algorithm. The distance between the i^{th} centroid vector $\mathbf{v}_i$ and the feature vector corresponding to the k^{th} pixel $\mathbf{f}_k$ is measured by the Mahalanobis distance, with F the total number of features:

$$\boxed{d_{ik} = \sqrt{\sum_{j=0}^{F-1} \frac{(f_{kj} - v_{ij})^2}{\sigma_j^2}}} \tag{3.3}$$

σ_j^2 is the standard deviation of feature j over the entire image. The fuzzy C-means algorithm iterates, evaluating new centroids and a new fuzzy partition at each step, until stability is reached.

In the actual clustering system, the fuzzy algorithm is run two times using *a priori* as well as *a posteriori* feature reliability measures for its initialization. A flowchart is given in [1].

3.1.3 Object tracking

The tracking has three main functions:

1. assign the same label to a particular object over the whole sequence
2. detect and take care of new objects
3. detect and properly process disappearing objects.

Even though this looks simple, there are some difficulties. First, an object may grow or shrink and change its shape over time; the tracking must nevertheless perform properly. Then, the splitting of an object may form new regions in the object mask; we have to distinguish this particular situation from an object that enters the scene. Also, objects may “disappear” because they merge with other objects or get partially occluded; it is essential to detect this.

The car tracking algorithm projects each object of the previous frame $(n - m)$ to the current frame (n) by motion prediction. To get the motion vectors, an estimation method developed by Lucas and Kanade is used. Then, the Euclidean distance between the objects in frame (n) and the motion compensated objects is calculated. The object with the shortest distance to the prediction of an object A in frame $(n - m)$ is considered to be A in (n).

New objects are detected using the object mask. In fact, if there are more objects in the current than in the previous mask, one of them must be new; it is considered to be the one with its gravity center most away of the projected gravity centers. There is, however, no distinction between a splitting and a real new object.

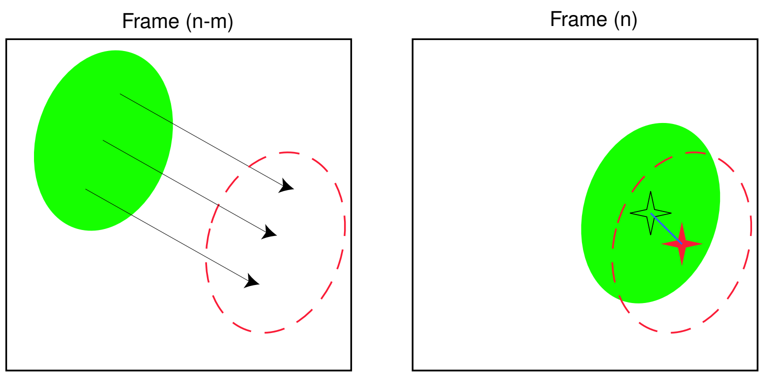


Figure 3.3: Object tracking

Disappearing objects are also detected with the mask: when an object is missing in frame (n), it is considered to be the one with its prediction most away from the remaining objects.

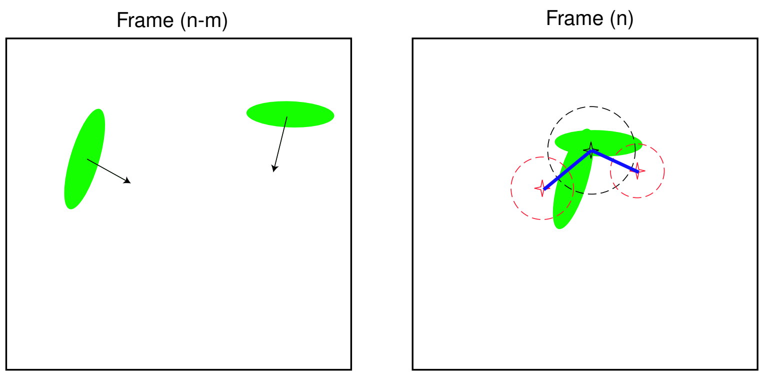


Figure 3.4: Merging detection

Merging is localized using a tricky, but somewhat restricted way. Again, the objects are projected to the current frame. There, the predicted (red on fig. 3.4) as well as the actual objects (in black) get an imaginary circle that has the surface of the object and its origin on the gravity center. If the sum of the actual and the predicted radii is larger than the Euclidean distance between them, and this for all the predicted objects, then they have merged. In order to try to track merged objects side-by-side and not as one unit, a **post-segmentation** technique based on feature comparison was considered. It basically looks for the old centroids that look “closest” to the new ones. This will be the basic idea for the improved tracking scheme.

3.2 The improved tracking algorithm

As explained earlier, we would like to track objects accurately in generic scenes; therefore, the tracking scheme has to be adapted. To further improve its efficiency, we will also refine the object mask and the clustering somewhat. However, different segmentation or clustering algorithms may be substituted to them whenever needed. The new tracking acts at a different level than the old one: instead of find-

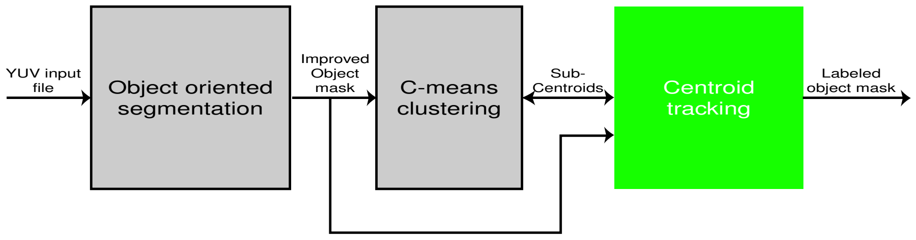


Figure 3.5: Overview of the improved tracking system

ing directly the (n) object corresponding to some $(n-m)$ object using their gravity centers, it tries to track the *centroids* from the previous to the current frame. If we know to which object some centroids belonged in the previous frame and were able to track them correctly, then we also know where the object is currently located, because we have labelled its centroids.

3.2.1 The enhanced object mask

Until now, the clustering was performed on each object mask region, so merged objects could have common clusters. Since one of our primary objectives is to track merged objects side-by-side, we need to adapt the mask in order to be able to give independent clusters and centroids to each *subobject*. To do this, we will try to anticipate the position of merged objects prior to the clustering. This information is approximately available if each previous cluster is projected to the current frame. If projected clusters belonging to different previous regions are projected on a unique object, this one gets further divided into subobjects. Also, the predicted label of each object and subobject should be stored because it may be reused to give some reliability to the tracking results. Since motion projection does not necessarily produce an accurate result, the holes that may be left in the object mask are filled out by recursive dilatation. The C-means clustering is then performed on each subregion.

3.2.2 Refined C-means clustering: the subclusters

With the C-means algorithm, it can be that one cluster spans many disparate parts of an object. This may be problematic when those parts move into disparate directions, like the parts of a person spreading his arms; in such cases, a centroid could have null mean motion while the corresponding parts actually move. To avoid such situations, we will further subdivide the clusters into homogeneous *subclusters*. Each one of those then gets its own *subcentroid*. Doing so, we do not only decrease the area described by one cluster, but we also increase their number, which may be helpful for the tracking too.

3.2.3 Centroid tracking

For textual video description, it is of main importance not just to find semantic regions in one frame, but to be able to follow them during the sequence. Since we decided to extract some object features by centroid clustering, it seems reasonable to look for similarities between the (sub)centroids of the previous and the current frame to track them. Unlike the mesh-based approach, centroid tracking is not bound to the actual shape of an object. In compensation, the hidden shape of occluded objects is not known, but this is irrelevant for our application.

Before previous and current centroids can be paired, we need to define what **features** should be used. At first sight, one may think that the more features are considered, the better the tracking performs. For example, it would be impossible to distinguish two similar cars just on their color; position and speed may certainly improve the result. However, too many features are misleading: in our car example, it is useless to match the color because it is the same in different objects anyway and does not help to differentiate them. So features that are similar up to some threshold in different objects have to be ignored.

For our camera, we propose to select some or all of the following features depending on the application: position, motion, color and texture. The position helps to find small disparate objects, but may not be a support for deforming or overlapping ones. Motion is often very useful, especially to distinguish parts of an object moving into opposite directions. It is of less help for solid objects going the same way. As we will see later, the color may be used when the motion cannot be accurately determined; it is worth to point out here that the used color space may also affect the result: in the ordinary video color space YUV, perceptually similar colors result in very disparate numerical component values, which are difficult to match. Perceptual color spaces like Lab help to avoid these problems. The texture is less important, but may serve as an indicator for the motion reliability. Of course, other features can be considered when required.

Then, the previous centroids have to be projected to the current frame. For the position, we use motion compensation once again. The other features however keep their ancient value, because it is difficult to plan the evolution of color or motion without any high-level knowledge of the scene (3-D perspective model).

To form the **pairs**, it is necessary to find a way to measure the distance between centroids despite their disparate feature values[1]. The Mahalanobis distance

[1]the x and y coordinates are limited by the size of the picture, the color components usually lie

provides a widely used solution for distance normalization. In our case, it would also be recommendable to weight the features since one feature may not be able to differentiate some objects as good as another one. The weighted Mahalanobis distance between two centroids u and v is:

$$\boxed{d_{uv} = \sqrt{\sum_{j=0}^{F-1} (w_{uj} w_{vj})^{1/2} * \frac{(u_j - v_j)^2}{\sigma_j^2}}} \tag{3.4}$$

with

$$\sigma_j^2 = \frac{\sum_{k=0}^{N-1} (f_{kj} - \bar{f}_j)^2}{N} \qquad \bar{f}_j = \frac{\sum_{k=0}^{N-1} f_{kj}}{N}$$

w_{kj} represents the relative weight of the j^{th} feature in centroid k, F is the maximal number of features and N stands for the number of pixels in the image. $\mathbf{f}_k$ is again the feature vector with respect to the k^{th} pixel.

To define the **weights**, we suggest a method that was first proposed by Roberto Castagno in [2]. He noticed that most motion-prediction algorithms perform very poorly on uniform surfaces, while they show better results on textured parts. Therefore, the motion should be weighted according to its reliability, which is given by the eigenvalue product $\sqrt{\lambda_1 \lambda_2}$ in the Lucas & Kanade method. Since on the other hand color information defines uniform surfaces well, it may just be weighted against motion as shown in fig. 3.6. There is no apparent reason to change the weights of the position and texture, so they just get some constant value. The actual values for those weights have to be determined experimentally; Castagno found that weights of 0.1 for position and texture and a maximal weight of 0.675 for motion showed good results. For the Mahalanobis distance to be accurate, the sum of all weights has to be 1.

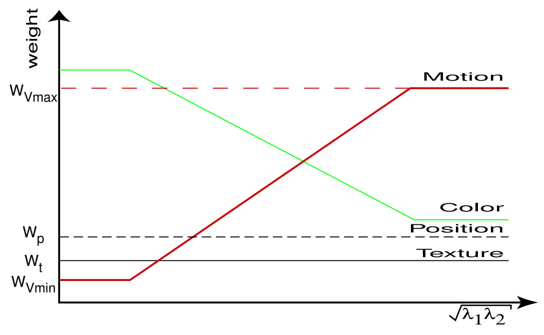


Figure 3.6: Relative weights of the available features

between 0 and 255, while the motion may span any interval, depending on the sequence

During the pairing, it is not sufficient just to attribute the closest previous centroid to each current one, because this would force the appearing centroids of some new object to be paired with old ones. One possible way, which we use in our software, is to first take the best one-to-one pairs (current centroid with closest previous one and vice versa), which are removed from the list of centroids to pair. The remaining ones are paired again, etc. , until no pair closer than a given threshold d_{max} can be formed. The so-formed pairs are stored in a table with the related previous centroid as well as the corresponding Mahalanobis distance for each paired current centroid, as shown in 3.1. From those pairs, it has to be determined which objects belong together.

Current centroid	0	1	2	4	...
Closest previous centroid	0	1	3	4	...
Mahalanobis distance	0.18	0.20	0.34	0.12	...

Table 3.1: Table for centroid pairs (example)

Relating centroids to objects and subobjects

The purpose of this essential part of the tracking is to follow individual objects through the scene, based on our centroid pairs. The main idea here is that, once we know to which object some centroids belonged in frame $(n-m)$ and where those same centroids are in frame (n), we should be able to find the object corresponding to the (n) centroids and therefore its label from $(n-m)$.

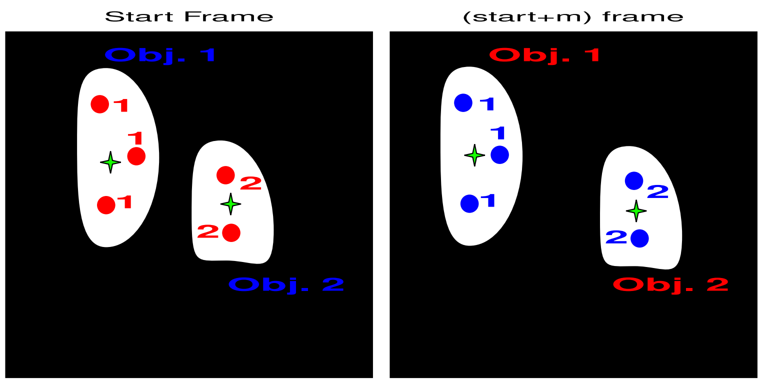


Figure 3.7: Initialization and first tracking step

This procedure has to be initialized somewhere so, in the first processed frame (start frame), each centroid gets a label corresponding to its closest object. Then, the centroids are paired with the ones of the next frame as explained in the previous section. In this next frame, we want to label the object mask regions according to the start frame. To do so, we only consider the centroids that were successfully paired and therefore got an object label. Each region of the current object mask gets the label of the centroids that are related to it. Of course, some regions may not get any centroids so far. The figure 3.7 may help to understand this first step. In this figure, blue means some given input element while red are values found by the tracking.

Sometimes, centroids with different labels coexist in a same region. In this case, objects have either **merged** together, or one object got **partially occluded** by another one. This case can be handled in two different ways: either the resulting object gets the “majority” or some new label, but then the merging is not visible and must be stored in the object history; or we try to track the objects ”side by side”. To do this, we can use the subobjects formed prior to the clustering (s. section 3.2.1). As we did for the objects, each subobject gets the label from its centroids (fig. 3.8). Note that it is possible to know which object overlaps which one because the overlapping part, known thanks to the motion prediction, gets labelled accordingly too. Total occlusion results in the loss of the occluded object, but it could be recovered when it reappears if some memory of its characteristics (storing of its centroids for example) was added.

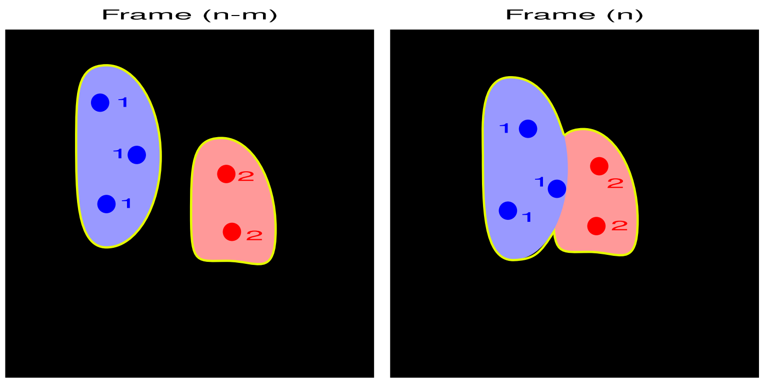


Figure 3.8: Merging centroids

On the other hand, if centroids with the same label are present in two distinct regions, the original object has **split**. The event has to be stored in the object history in order to allow them to re-merge together and get back their old label. The motion predicted subclusters may help to indicate a splitting.

Thanks to the predicted labels, we can also measure the **reliability** of the labelling. In deed, a label is likely to be reliable if the centroid tracking and the predicted label are in accordance. On the other hand, less reliability should be given for contradictory results. To get some numbers, we could look at the percentage of centroids in an object that were paired accordingly to the prediction: if almost none are, the tracking may be really wrong.

Updating unpaired centroids

Once all objects and subobjects have been labelled, we need to update the labels of all unpaired centroids so that they can be reused for the next iteration. If some unpaired centroids are alone in an object mask region, we consider them to belong to a **new object**, which gets, as well as the related centroid(s), an unused label. If on the other hand the region been labelled already, the unpaired centroid is considered part of a growing object and labelled correspondingly.

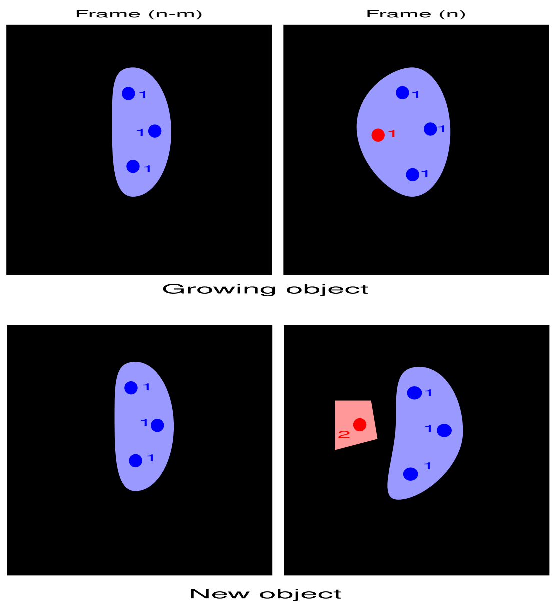


Figure 3.9: Processing appearing centroids

Once all the current clusters and object mask regions have been labelled, the algorithm iterates.

3.2.4 Limitations and software problems

The principles explained so far seem rather simple and should provide better results than the old tracking in many cases. However, there are limitations which are inherent to the principle as well as to its implementation. First, the algorithm does not have any knowledge of the shape of an object, so it may be, for an inexact object mask or if the cluster projection performs badly for merging objects, that the resulting regions become meaningless. Also, there is no real way to catch errors once they occurred. An object database would help out in those cases.

Parameters such as the weights also bring their problems, since centroids may be paired wrongly. This sometimes results in erroneously labelled (sub)objects. Therefore, the system has to be carefully calibrated.

Chapter 4

Sequence description

Now that the semantic objects have been localized and their main features extracted, we have to describe them textually. The descriptors proposed in this chapter try to be as general as possible. Also, instead of translating them directly into some high-level language (XML), we store the extracted parameters in a C structure that will be discussed in the second part of this chapter.

4.1 Descriptors

Rather than to reinvent descriptors for our purpose, we tried to stay as close as possible to the MPEG-7 syntax. General information about DDL can be found in [11], while the visual descriptors are listed in [12, 13, 14]. Following features shall be supported (optional descriptors in italic):

FEATURE	DESCRIPTOR/DS	PURPOSE
Label	`Object Identifier`	Label each object in a unique way
	`Media Locator`	Specify the location of the object
Shape	`Region locator`	Box or polygon shape description
	`Contour shape`	Describe a closed contour shape
Color	`Dominant color`	Specify a set of dominant colors
	`Color layout`	Spatial distribution of colors
Texture	`Texture browsing`	Perceptual texture description
	`Homogeneous texture`	Structural texture description
Motion	`Motion trajectory`	One point motion
	`Spatio-temp. locator`	Describes moving regions
	`Motion activity`	Qualitative motion descriptor

Table 4.1: Descriptor list

4.1.1 Main cluster

Since the purpose of the descriptors listed in table 4.1 is to characterize an arbitrary image region, we group them together in an entity called *object*. MPEG does not provide description schemes for generic objects, so we use the `descriptor collection` DS to define them. In fact, the descriptor collection allows to specify an unordered, unbounded set of descriptor instances. Note that this object can, but must not correspond to a physical object.

Syntax: s. MDSWD § 14.1.3

4.1.2 Label

Object identifier

The function of the identifier is to

- label each object in a unique way.
- provide some information about the past of the object (merging and splitting).

In order to allow maximal flexibility (multiple merge-split cycles, tracking of side-by-side objects) with a simple naming scheme, we proceed as follows:

- when a new object appears, for the first time as well as after merging/splitting, it is named `o#`, where # is the smallest number that has not already been used. This can be done using a simple counter. However, objects that were already merged and are re-split or vice versa will get back their old names.
- when some objects merge together, the resulting object gets the mention `_m(l1,l2,...)` in addition to its new name, where `lx` are the full labels of the original objects (e.g., `o5_m(o2_s(o1),o4))`.
- when an object splits in some parts, the resulting objects get the mention `_s(l)` in addition to their new names, where `l` is the full label of the original object (e.g., `o2_s(o1)` and `o3_s(o1))`.

Those rules may result in rather long labels (multiple merge / split cycles), but they allow to fully reconstruct each object's history (for database applications) and are independent of the implemented tracking scheme, since for example merge and split detection can simply be left out. When we display an object, we may only keep it's name without the "historical" part, for example just `o2`. Some of those rules are illustrated in fig. 4.1.

Syntax: the identifier goes into the object header

```
<Object id="ObjectIdentifier">
    ...
</Object>
```

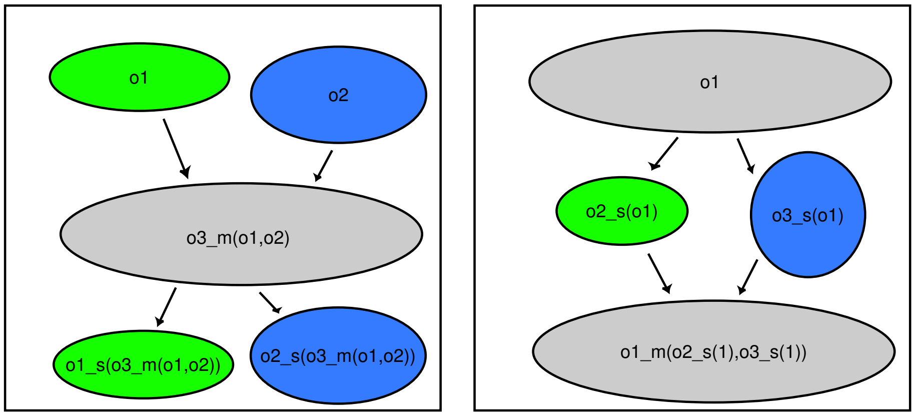


Figure 4.1: Some naming rules

Media locator

Media locators are used to specify the location of an object; such information is essential for scene reconstruction and databases, since it tells which objects belong together and what scene they come from. In our application, where objects are usually extracted of a video file, we will use the `video segment locator`. However, depending on the origin (e.g., internet), other media locators may be considered.

Syntax: s. MDSWD § 6.4

4.1.3 Shape

As shown in fig. 4.2, we propose three different shape descriptions: as a box, as a polygon and contour shape. The first two will mainly be used to get some idea about an object's size or dimensions, while the third one allows its reconstruction. Those descriptors include information about the shape as well as about its absolute position in the image.

Region locator

The region locator enables a brief and scalable representation of a box or a polygon; the polygon extraction can be performed by the preexisting EPFL tool [16]. It should be noticed that the region locator may be redundant with the `spatio-temporal locator` when both specify a polygonal shape; however, the last one has more general functionality since it can also describe the deformations of a contour shape, so we keep them both[1].

Syntax: s. VCD § 10.1

[1]The encoder will have to prevent conflicts

Contour shape

The contour-based shape descriptor represents closed shapes in the Curvature Scale Space (CSS). Holes are not supported, but they can be encoded as another contour shape if required.

Syntax: s. VCD § 8.2

Unused descriptors

Even though `region shape` can describe complex shapes made of several disjoint regions, we will not implement it since its purpose, the full reconstruction of a shape, is mainly the same as for the simpler contour shape. We will also not use the `object bounding box`[2], whose purpose is perfectly the same as the one of a box region locator.

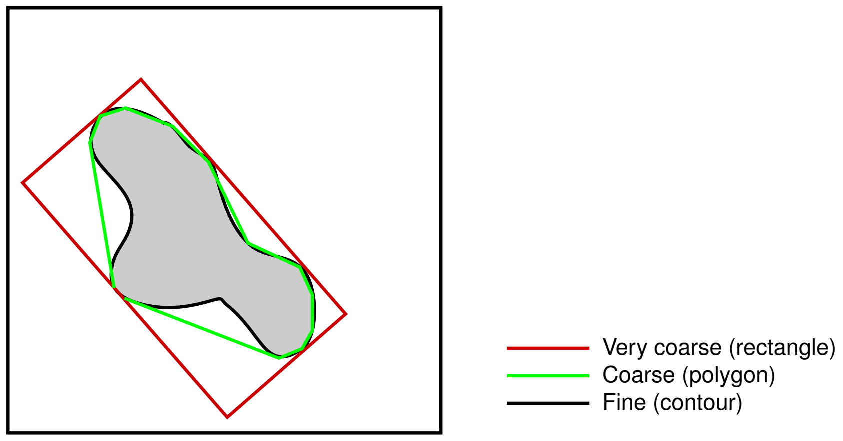


Figure 4.2: Scalable shape description

4.1.4 Color

Dominant colors

The MPEG standard allows up to 8 dominant colors for this descriptor. This should be enough for an adequate description of most natural objects, and it permits scalable encoding (e.g., when we transmit a scene containing a fire, we can first send red, then the secondary colors. A video surveillance program may identify the fire before it receives all the colors). Notice that `color space` and `color quantization` have to be specified in this descriptor. Since it is often used in

[2]VXM § 7.1

video files, we will usually define the YUV color space by its linear matrix transformation [15]:

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Syntax: s. VCD § 6.3 and § 6.1–6.2 for `ColorSpace`/`ColorQuantization`

Color layout (*optional*)

For complex or large objects, it may be desirable to store some information about the spatial distribution of colors (e.g., for a "person" object with different clothes). That is the purpose of this descriptor, which can be added if necessary.

Syntax: s. VWD § 6.5

Unused descriptors

The MPEG documents provide many color descriptors. `Scalable color` and `color structure` are based on histograms, which we will not use since we consider that 8 colors are enough for our application.

4.1.5 Texture

For scalable data transmission, we want to have a qualitative as well as a quantitative texture descriptor.

Texture browsing

This perceptual texture descriptor allows fast image retrieval and very compact encoding. However, a qualitative description may not be sufficient to represent natural images (for example when the texture contains some essential information, like age rings of trees).

Syntax: s. VCD § 7.2

Homogeneous texture (*optional*)

This can be added if a more precise texture description is wanted; as pointed out in the VXM[3], it performs well together with `texture browsing` for database retrieval. However, we will not implement it since it addresses pure texture images rather than object textures. For implementation, one has to decide which part of an object's texture he wants to extract.

Syntax: s. VCD § 7.1

[3]VXM § 4.1.2.3

Unused descriptors

`Edge histograms` can be very effective for natural image mapping based on texture[4], but, since they use square sub-blocks, they are not of much help to describe object textures.

4.1.6 Motion

We try again to provide some scalability by using a motion descriptor for single points as well as a more complex one. Additionally, a qualitative motion description can be used.

Motion trajectory

Its purpose is to describe the movement of a single point, for example the gravity center of an object, through temporal interpolation. Since it includes no information about the spatio-temporal location of the trajectory, we need some other means for this, for example the `media time` description scheme[5]. Notice the presence of `CoordinateSystem` and `TemporalInterpolation` fields.

Syntax: s. VCD § 9.2 and § 5.5 for `TemporalInterpolation` as well as VXM § 2.2 for `CoordinateSystem`

Spatio-temporal locator

Localization functionality in space and time as well as shape deformations in time can be addressed by this description scheme. For the later, it provides two distinct methods: `FigureTrajectory` for non-rigid moving object regions, and `ParameterTrajectory` for rigid regions. `Media time` as well `region locator` and `temporal interpolation` are used.

Syntax: s. VCD § 10.2 as well as § 10.1 for `RegionLocator`, § 5.5 about `TemporalInterpolation` and MDSWD § 6.3.8 for `MediaTime`

[4]VXM § 4.3.2.3
[5]MDSWD § 6.3.8

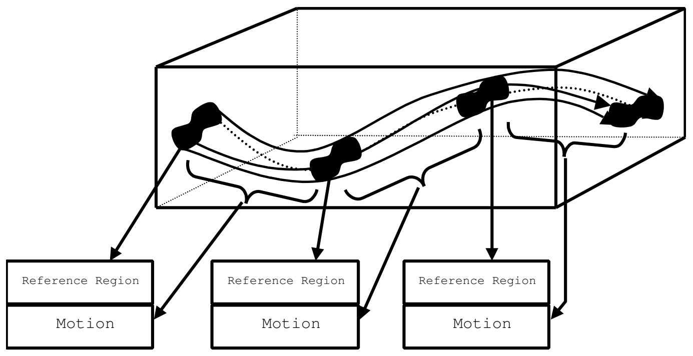


Figure 4.3: Spatio-temporal regions

Motion activity (*optional*)

As for texture, it can prove helpful to extract a qualitative description for database applications. Motion activity describes motion in terms of Intensity of Activity, Direction of Activity, etc. Those qualifiers rely on relative criteria and need human input.

Syntax: s. VWD § 9.4

Unused descriptors:

Even though `parametric motion` is more specific than the `ParameterTrajectory` in a spatio-temporal descriptor, their functionality seems to be very similar (same parametric models). We keep the more general one.

4.1.7 Reliability and confidence

Feature extraction and object tracking are difficult tasks for a computer and can produce very rough results. A decoder has therefore to know what importance the received parameters have, because, if we are looking for a picture containing a tall man, it would not be advisable to keep an object whose shape extraction was very uncertain…
MPEG provides the `model` description scheme for confidence as well as for reliability. Here, `reliability` refers to the accuracy of the values of the model parameters, which may take into account the method of extraction, while `confidence` refers to the values of the model parameters statistically by indicating the degree of membership in a confidence interval (for example how many percent of a rectangular shape descriptor contain the actual shape). Both are optional.

Syntax: s. MDSWD § 14.2.1

4.2 C structure for objects

Rather than translating the extracted features into high-level descriptors directly, we will previously store them in a computer-friendly C structure. Doing so, we increase flexibility because the translator module can be modified easily (different languages, ...) without affecting the feature extraction or storage. The proposed structure must store all the variables required by the descriptors as well as their reliability. For scalable transmission, we will provide another structure with a descriptor priority list.

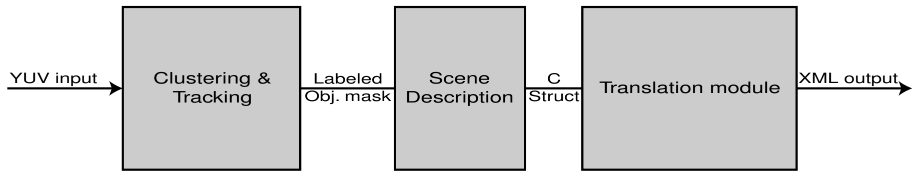


Figure 4.4: System overview

The data is organized as follows:

- the `Object` cluster (s. table 4.2) includes one structure for each physical feature, like color, texture, ...
- each one of those structures holds a substructure for every MPEG descriptor listed in subsection 4.1 (the *structname***D** substructures).
- in those substructures, there is one variable (scalar, vector, table, ...) for each parameter. Associated to those variables is a table which stores the **reliability** (between 0–1) and the **confidence** (0–1) of the corresponding parameter.

parameter	*reliability*	*confidence*

- Finally, a separate and optional `Priority` structure lists the data in the order we want it to be transmitted, down to any needed level (e.g., can give a priority to a physical feature, a descriptor or even a single parameter).

Note: each feature can be used independently of the other ones, since it contains all the required information (no “differential coding”). Also, we will not mention the reliability/confidence table in the subsequent structure descriptions, since its format will always be the same.

According to those rules, the main object cluster looks as follows (optional descriptors are *not* considered here):

```
struct object
{ Label label;
  Shape shape;
  Color color;
  Texture texture;
  Motion motion;
};
```

Table 4.2: C structure for objects

4.2.1 Label

Identifier

This is the object name, as explained in 4.1.2. As for the other parameters, a reliability measure can be associated to the name, since the given names may be uncertain in complex scenes (after merging or when an object disappears and reappears later).

Media locator

The proposed structure is laid out for the `VideoSegmentLocator` DS, but it can easily be modified for other media locators.

```
struct MediaLoc
{ VideoSegmLocD videoSegmLoc;
};
```

Video segment locator

```
struct VideoSegmLocD
{ Char *video_name;
};
```

4.2.2 Shape

The shape structure includes two substructures, for the `region locator` and for `contour shape`.

```
struct Shape
{ RegionLocD regionLoc;
  ContourShapeD contourShape;
};
```

Region locator

```
struct RegionLocD
{ Unsigned short box_Poly;          // 0=box, 1=poly
  Int coordref                      // Optional
  Int *coords[2];                   // (x,y)
                                    coordinates
  Unsigned short inverse;           // Inner/outer
                                    region
};
```

Contour shape

```
struct ContourShapeD
{ Int numberOfPeaks;
  CurvatureVector                   // Defined below
  globalCurvVector;
  CurvatureVector
  protoCurvVector;
  Float heighestPeak;
  Float *peak[2];                   // (x,y)
                                    coordinates
};
```

The `CurvatureVector` is defined as follows:

```
struct CurvatureVector
{ Float circularity;
  Float eccentricity;
};
```

4.2.3 Color

Stores the dominant colors.

```
struct Color
{ DomColorsD domColors;
};
```

Dominant colors

```
struct DomColorsD
{ ColorSpaceD colorSpace;          // Defined below
  ColorQuantD colorQuant;          // Defined below
  Unsigned short size;             // # of
                                   dom. colors
  Unsigned short
  variancePresent;
  Unsigned short                   // 0--31
  spatialCoherency;
  DomColorValues *values;          // Defined below
};
```

`ColorSpaceD`, which is itself a MPEG descriptor[6], has this structure:

```
struct ColorSpaceD
{ Unsigned short colorSpaceType;
  Float colorTransMat[3][3];       //
                                   Transform. matrix
};
```

`ColorQuantizationD`. This descriptor defines the uniform quantization of a color space.

```
struct ColorQuantD
{ Unsigned short component;
  Unsigned binNumber;
};
```

The `DomColorValues`, finally, are defined as follows:

[6]S. VCD § 6.1

```
struct DomColorValues
{  Unsigned short percentage;      // % of object
   Int colorValueIndex;
   Int colorVariance[3];           // Dim(color
                                   space)
};
```

Low and high spatial coherency are illustrated in the following figure:

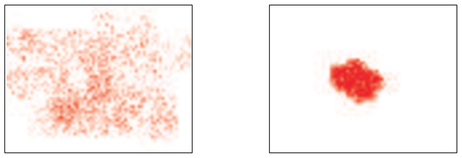


Figure 4.5: Examples of spatial coherency for dominant color

4.2.4 Texture

```
struct Texture
{  TextureBrowsD textureBrows;
};
```

Texture browsing

This descriptor uses subjective perceptual criteria like “Slightly regular coarse texture”.

```
struct TextureBrowsD
{  Unsigned short regularity;
   Unsigned short direction[2];
   Unsigned short scale[2];
};
```

4.2.5 Motion

The motion description may look rather complex, since it contains many other MPEG descriptors. We tried to organize it similarly to the other features.

```
struct Motion
{  MotionTraj motionTraj;
   SpatioTempLocD spatioTempLoc;
};
```

Motion trajectory

Since the system must know the time context for this descriptor, we add `MediaTime` to this structure; however, we want to keep it as a self-contained descriptor, so we put it in another `MotionTrajD` substructure.

```
struct MotionTraj
{  MotionTrajD motionTraj;
   MediaTimeD mediaTime;
};
```

The main structure is

```
struct MotionTrajD
{  CoordRef coordRef;
   SpatialRef spatialRef;           // Optional
   InterpolationD interpolation;
   Unsigned short cameraFollows;    // Here not
};
```

Following substructures have to be defined:

```
struct MediaTimeD
{  MediaRelIncrTimept timept;       // syntax:  MDSWD
   MediaIncrDuration mediaDur;      // ''
};

struct CoordRef // s. VCD 5.4
{  Unit unit=pixel;
   Int pictureHeight;
   Int pictureWidth;
};
```

Note that only the “unit” part of the coordinate system is required; `CoordinateMapping` and `GlobalImageMotion` are not used.

```
struct InterpolationD
{ WholeInterval interval;
  KeyTime *keyTime;
  KeyValue *keyValue;
  Unsigned short functionID;
  Float functionParam;
  Unsigned short numOfKeyPoints;
  Unsigned short size;
  Unsigned short constTimeInt;
  Unsigned short defaultFunct;
};
```

Spatio-temporal locator

```
struct SpatioTempLoc
{ FigureTrajectory *figTraj;
  ParameterTrajectory
  *parameterTraj;
};
```

```
struct FigureTrajectory
{ MediaTimeD mediaTime;
  Unsigned short figureType;
  TemporalIntD temporalInt;
};
```

```
struct ParameterTrajectory
{ MediaTimeD mediaTime;
  Unsigned short motionModel;
  Unsigned short ellipseFlag;
  RegionLocD regionLoc;
};
```

The `region locator` has already been used to describe shape; its definition can be found in 4.2.2.

4.2.6 Priority list

As an optional feature, we provide a structure for priority lists corresponding to different applications or hardware. A slow network could for example give high priorities to shape and motion descriptions but low ones for texture, while a fast network would not attribute any priority.

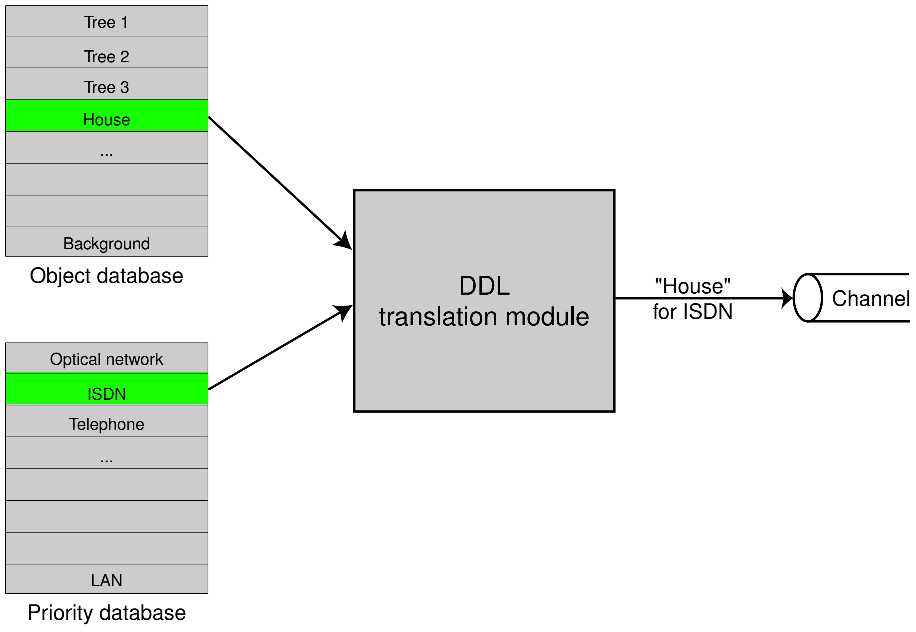


Figure 4.6: Priority list application (example)

```
struct Priority
{  PriorityField *priorityField;
};
```

with

```
struct PriorityField
{  Char *feature;
   Char *descriptor;                  // Optional
   Char *parameter;                   // Optional
};
```

4.2.7 Notes about the software

The previous sections shall give a comprehensive overview of the data organization in our project. For simplicity however, some of the actual coding techniques used in the software do not appear in this document. For example, parameter chains like the coordinates in the `regionLoc` descriptor (§ 4.2.2) are noted `*coords` in this document; in the software, we use a dynamic structure for them. However, those simplifications should not affect the understanding of this work in any way.

Chapter 5

Experimental results

To evaluate the proposed tracking and description schemes, we have added centroid tracking as well as a data structure for the storage and DDL output of scene descriptions to the existing highway program [1]. The resulting package has been tested on different sequences, each one of them containing some specific tracking challenges.

After an overview of our actual implementation (s. also appendix A), we will describe the behavior of our system with each test sequences. While we generally focus on the tracking performance, one full result (from the input file to the DDL output) will be given.

5.1 Implementation choices and parameters

One of the core requirements for our system was the ability to track reliably a wide range of sequences without any human interaction. Therefore, we had to find a parameter set which does not need to be adapted to each sequence while keeping errors rare. The performance is mainly influenced by:

1. the features used for tracking,
2. the number of clusters given to each object and
3. the method used for their extraction,
4. the weights of the Mahalanobis distance,
5. the conflict resolution between the prediction and the centroid tracking.

In our program, the user can choose to track based on position, motion, color and/or motion. However, it would not be wise to fix the **features** in advance, because the performance may vary depending on the scene composition (new objects, different lightning, etc.). To avoid this, our software measures the distance between the centroids belonging to different objects for each feature; features which are too close[1] and therefore unrepresentative are not used for pairing. The Lab **color space** always performed better than YUV in our tests, and was therefore used systematically. The reason of its superiority lies in the adequation between Lab components and the human color perception: similar colors have similar components.

As mentioned in section 3.2.2, the **subclusters** are obtained by isolating the homogeneous parts of each FCM cluster. The mean values of the features over the subclusters provide the corresponding subcentroids. This method allows to take care of possible local specificities of an object even if its FCM clusters are spread all over it. In order to limit the number of subclusters, we also tried to group close ones together, but the tracking results were not affected by this.

The **weights** used in the Mahalanobis distance are given according to Castagno [2] as shown in figure 3.6. He experimentally got a set of numbers that shows good performances for centroid pairing:

$$W_{Vmax} = 0.675 \qquad W_{Vmin} = 0.025 \qquad W_p = W_t = 0.1$$

with W_{Vmax} and W_{Vmin} the maximum and minimum weights for the motion, W_p the weight for position and W_t for texture. In our experiments, it was sometimes possible to resolve a local problem, for example an undetected splitting, by changing those weights drastically, but those numbers worked well for the general case.

The most tricky part was to process the **conflicts** between the centroid tracking and the predicted labels. In deed, the tracking must give the *most probable* label to each object. Based on our experiments, we decided to keep the predicted result in conflicting cases. In deed, it sometimes happens that the tracking attributes the same label to different objects even though no splitting was predicted; after a while, we may have many objects with the same label. Since we rarely observed the reverse situation, our choice was made. Another, more complex solution would have been to compare the pairing and the projection reliability to make the final decision.

To get the **descriptors**, we take advantage of the features that were extracted for the FCM algorithm by calculating their mean value for each object when it appears. The mean position is then calculated on each frame to form a trajectory. This provides some useful information about the object's position, speed, color and texture. To describe other features such as the shape, external algorithms like the one proposed in [16] may easily be added.

[1]Distance lower than given threshold

5.2 The test sequences

Our system has been tested on four different video sequences. To provide a clear idea of its possibilities, we will give the full result, from the input file to the DDL output, of one sequence. Then, we analyze some specific tracking challenges aid of the other sequences. This way, we will get some insight into the strengths and weaknesses of our algorithm.

5.2.1 Full example: the highway sequence

This sequence shows cars and trucks of different kinds and sizes driving on a highway, and is a nice example of what a smart surveillance camera may have to describe.

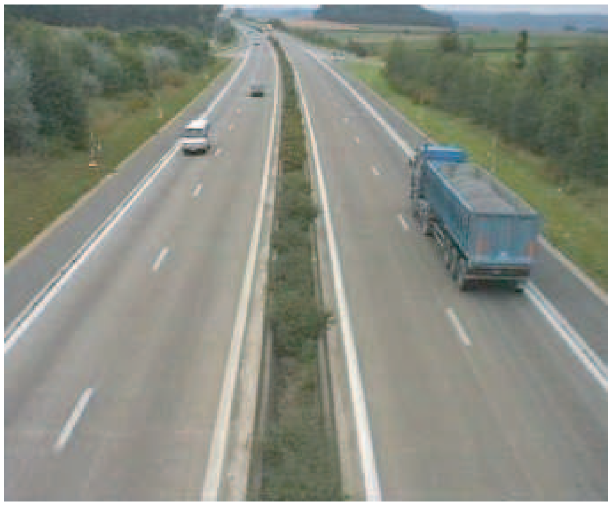


Figure 5.1: Highway.YUV

To thoroughly test our software, we ran the sequence between the frames 110 and 320. The figure 5.2 shows the trajectories of some of the cars that are detected these frames. The overall performance on this test was very good since all cars have been tracked accurately, and both mergings (frame 114 and frame 256) processed in a satisfying way. One of them will be detailed later.

The DDL output of the truck shall help to understand how a the scene is described. Each time a new object appears, it gets a name which, in our software, is just a number. The truck, which was already present in frame 110, got the label 1. Then, the average feature values of the new objects are calculated; they will be used as *key values* for the description. For the truck, we got the following numbers:

The object position is updated each frame, and new key values may be stored when the old ones differ too much from the actual values. On the next page, we

Feature	X pos.	Y pos.	X speed	Y speed	L	a	b	Texture
Value	236.4	92.9	0.4	1.1	148	85	79	5.7

Table 5.1: Feature values of the truck

show the DDL[2] output generated for the truck. We suppose that a `descriptor collection` for objects as well as the Lab `color space` have been previously defined. For any details about DDL, the reader should refer to chapter 4.

[2]In fact, we use a simplified “DDL-like” language for our purpose.

```
<!-- ##################################################### --!>
<!--  DDL output for object 1                              --!>
<!-- ##################################################### --!>

<Object id="1">

    <!-- First key values --!>

    <DominantColor>
        <ColorSpace>  Lab </ColorSpace>
        <ColorValue1> 148 </ColorValue1>
        <ColorValue2>  85 </ColorValue2>
        <ColorValue3>  79 </ColorValue3>
    </DominantColor>
    <HomogeneousTexture>
        <TextureValue> 5.7 </TextureValue>
    <MotionTrajectory>
        <TemporalInterpolation>
            <KeyFrame>   110    </KeyFrame>
            <KeyPos> 236.4 92.9 </KeyPos>
            <KeyFrame>   112    </KeyFrame>
            <KeyPos> 234.0 90.9 </KeyPos>
            <KeyFrame>   114    </KeyFrame>
            <KeyPos> 230.3 88.9 </KeyPos>
            <KeyFrame>   116    </KeyFrame>
            <KeyPos> 227.4 87.1 </KeyPos>
                     ...
        </TemporalInterpolation>
    </MotionTrajectory>

    <!-- New key values --!>

    <DominantColor>
        <ColorSpace>  Lab </ColorSpace>

                        ...


</Object>
```

Of course, this description can be refined with the full DDL syntax or some more features, but the example shows well how a video scene can be translated into a text. If one added an object database with key values of some known object, the camera could ideally track and identify objects automatically.

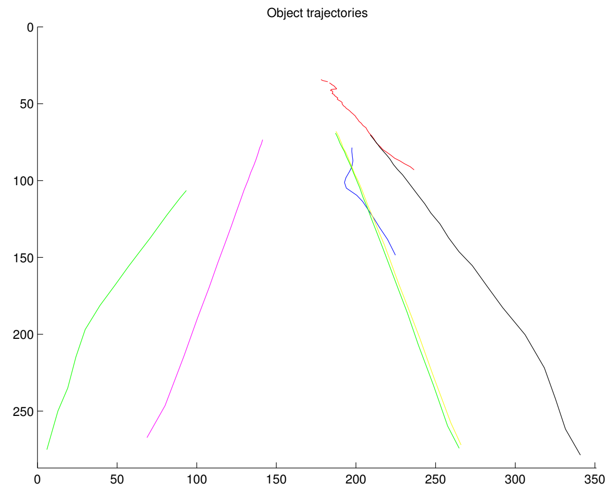


Figure 5.2: Some trajectories of the full highway test (frames 110–320)

The second point that we want to illustrate with this sequence is the tracking of side-by-side objects. Between the frames 114–160, a van passes the blue truck. The tracked object masks in figure 5.3 show what happens: until frame 112, the van and the truck are in separate mask objects. In frame 114, both merge to become one object, split into two subobjects by the preprocessing according to section 3.2.1. At this stage, the subobjects seem to fit the vehicles well. However, as the sequence continues, not all subclusters are projected accurately, and the van is less and less present in the object. In fact, the speed of most clusters is under-estimated, and only a few are still remaining 30 frames after the merging. But even like this, the separation of both objects is a real improvement over the old object-based tracking, which was only able to follow separate objects and, in simple cases, the individual centroids of merged objects.

The progressive loosing of the van's shape is also not necessarily a problem for the scene description: if we use the gravity centers of the subobjects as their position descriptor, we are able to track the van and the car separately nearly until they leave the image. This is shown in graph 5.4, were each object is given the same color than in the tracked mask images.

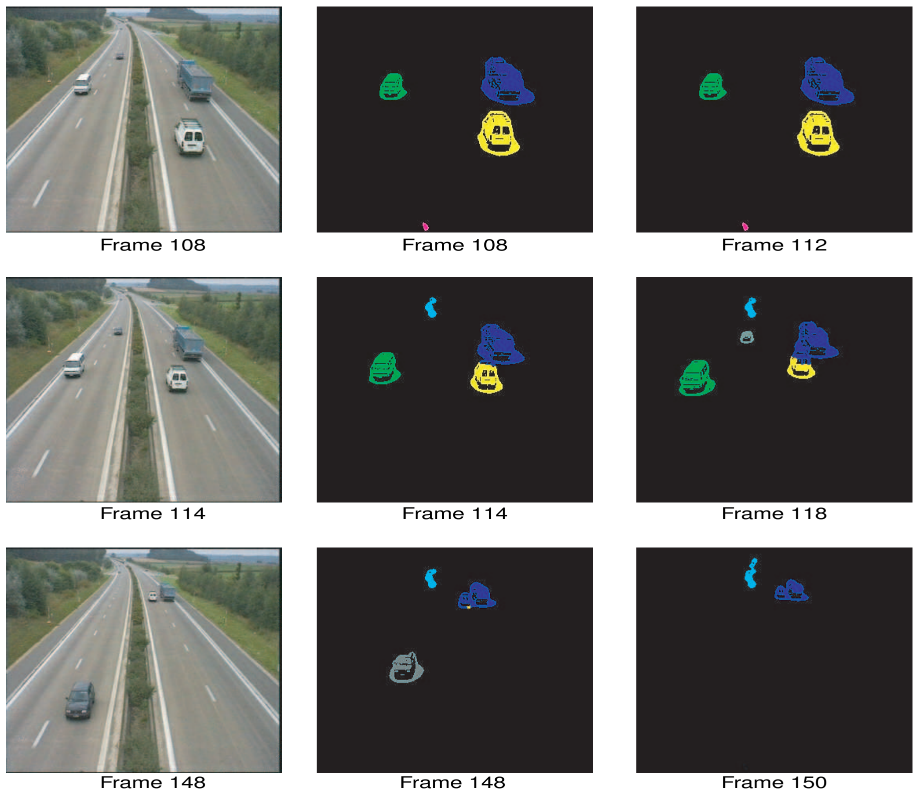


Figure 5.3: Side-by-side tracking

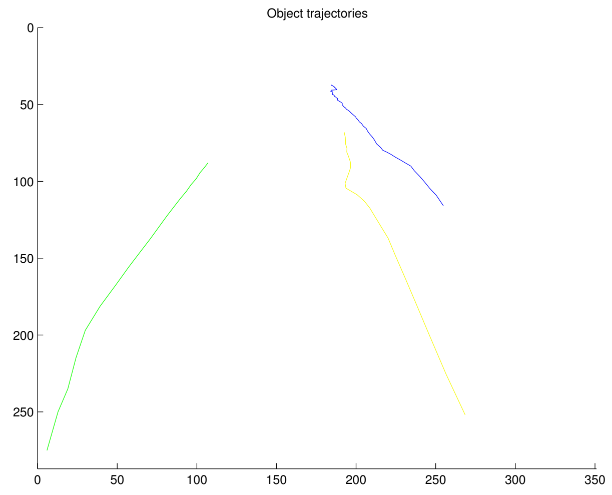


Figure 5.4: Trajectories of side-by-side objects (frames 100–160)

5.2.2 Software stability and object splitting: the hall sequence

Hall is a simple, 300 frames long sequence showing somebody coming out of an office, putting his document case on a small table, meeting somebody else and going into another room. Since we did not expect major difficulties on this scene, we mainly used it to get an idea of the system's stability. This aspect is of main importance in our application since an automatic camera must be able to film without interruptions under any circumstances.

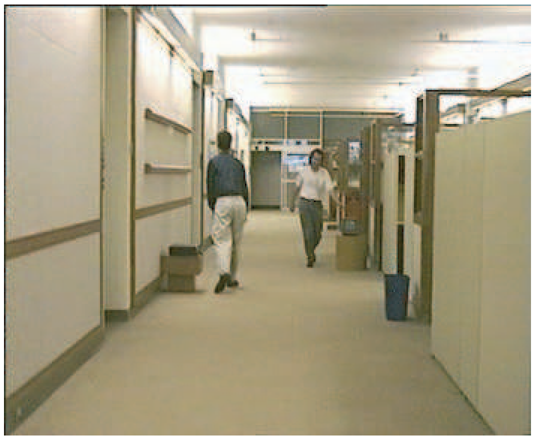


Figure 5.5: The hall sequence

The test ran crash-free over the whole sequence and did not loose track on any object. In the trajectory graph (fig. 5.6), one can clearly recognize each person's path, including the rest of the one that picks up a small TV set.

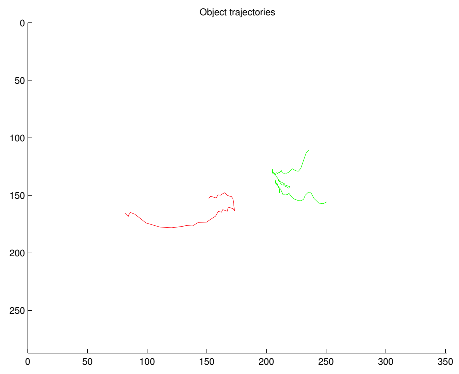


Figure 5.6: Trajectory of both people int the hall sequence (frames 40–300)

Another interesting part of this sequence is when the left person (in red on graph 5.7) puts his document case on the small table. Once the person moves along, the case keeps its label, as it should.

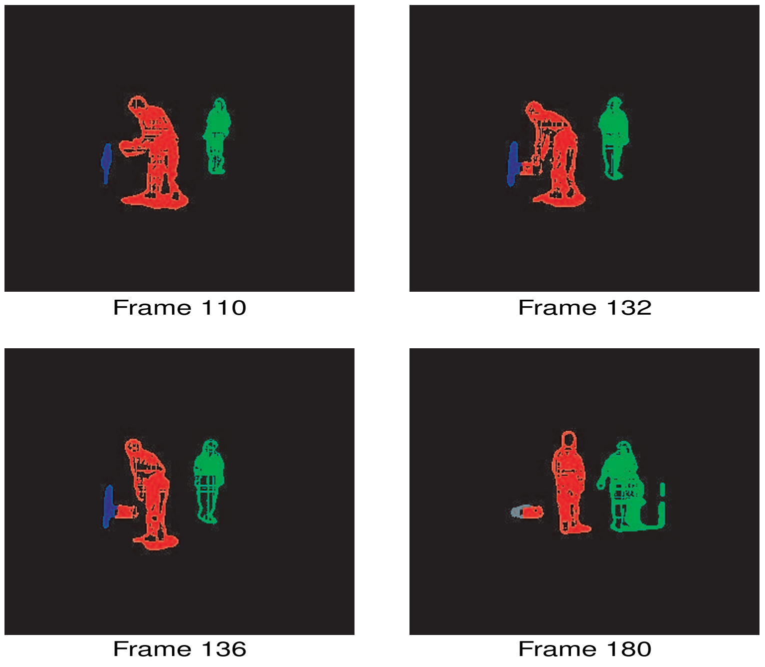


Figure 5.7: Splitting object

5.2.3 The volleyball sequence: split/merge cycles

This sequence includes much more difficulties than the previous ones: it shows two people playing volleyball with some small stuff pet. Since the lightning is very dark, it is sometimes difficult to distinguish the pet from the background. Also, its image is slurred when it moves fast, due to the long aperture time of the camera. This makes the sequence a good test for the change detection algorithm, which has to find an only slightly visible object, as well as for the tracking, since it must pair centroids based on features which are hardly different from the background.

We ran this sequence from frame 310 to 510. In this part, both people are seen playing together while passing one in front of the other. The tracking of the players was accurate as long as they did not exchange their positions (partial occlusion). Once this happened, the algorithm lost track on one of them, and the subobjects did not correspond to people anymore (the same problem will be addressed in detail in the group sequence). The ball tracking highlighted some inaccuracies that sometimes happen during merge/split cycles. To illustrate this, we shall have a look at the frames 133–147.

On both the tracked object mask and the trajectories graph (figures 5.9 and

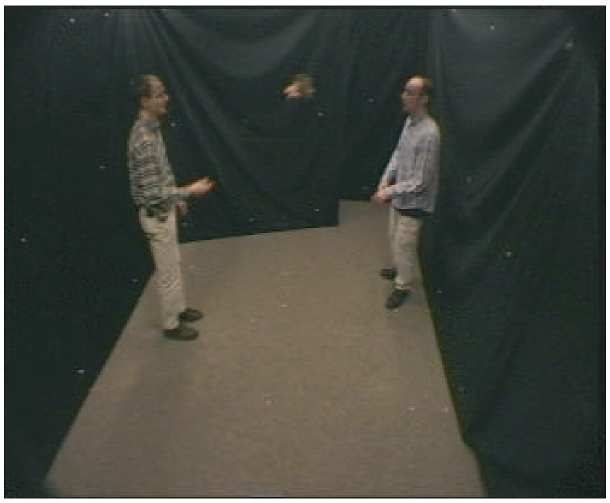


Figure 5.8: The volleyball sequence

5.10), we can see that the ball is tracked accurately up to frame 140. But then, the direction change is not anticipated correctly, and the ball therefore not projected out of the player's mask, so there is no predicted label for the ball in frame 141. Unfortunately, in frame 141, the right player's head has the same color and the same motion (!) than the ball, and gets paired with it erroneously. Since unpredicted splitting is not allowed, the ball finally gets a new label. Between the frames 146 and 147, a similar scenario happens: the prediction fails, and no subobject is formed at the end of the player's arm. So the ball is lost.

The Lucas & Kanade algorithm cannot predict the changing ball direction, because it analyzes five consecutive frames and gets the average motion out of them. Therefore, for changing directions, the average may be zero. So it would be up to the tracking to find the ball. This is difficult here because of the describes feature similarities between the head and the ball, but may be done if more weight was given to the position in such cases. In most common scenes however, it should be easier to distinguish objects.

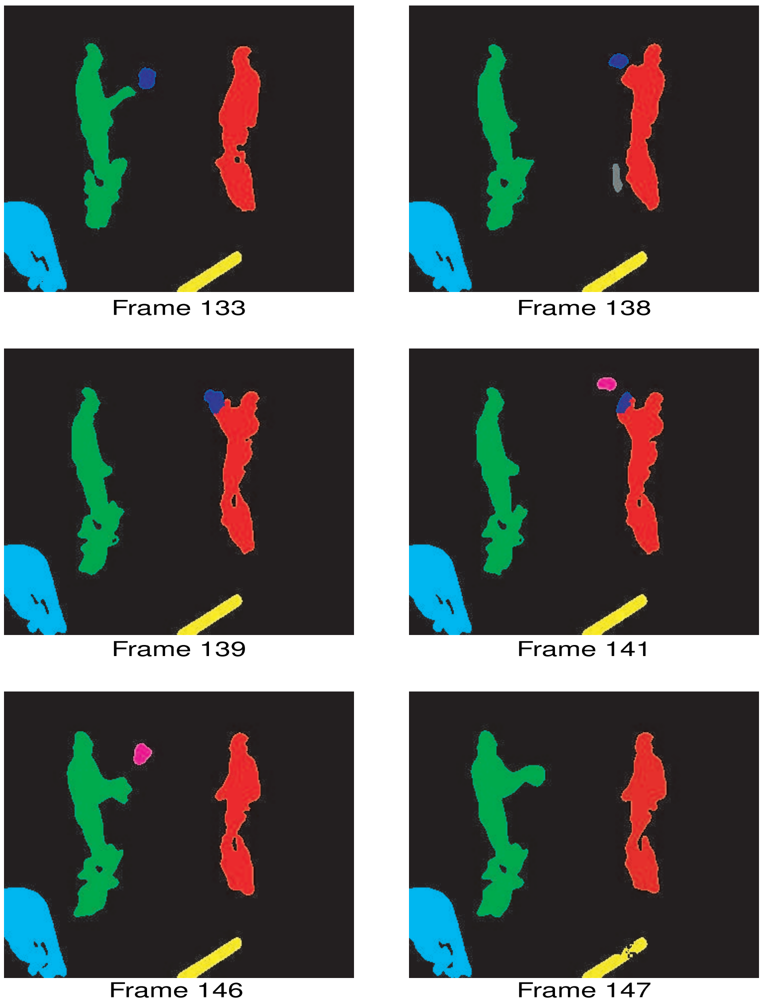


Figure 5.9: Split/merge cycles

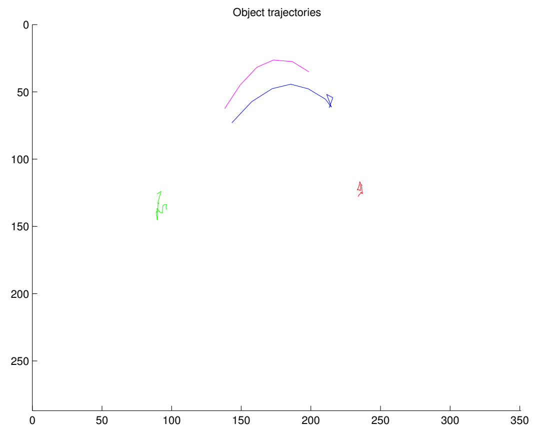


Figure 5.10: Ball and players trajectories in the volley sequence (frames 133–147)

5.2.4 Group: a complex sequence

The group sequence is very difficult to track, because it shows people coming together, passing in front of each other and making various movements. Since the people are grouped together in one object mask region most of the time, it relies a lot on the motion estimation to get correct subobjects. In order to create ideal testing conditions, we used a 20 frames long hand-made object mask for this sequence.

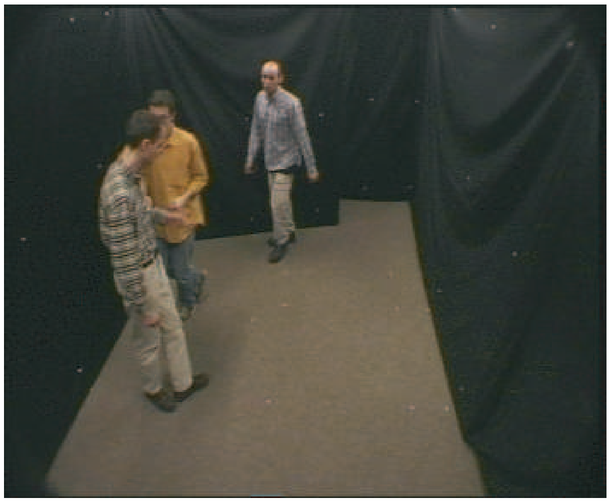


Figure 5.11: The group sequence

On the tracked mask (fig. 5.12), we see that the person who entered the scene first is tracked correctly, because it never merges with another one. The problem with the red person is that the subclusters of its body and head are not projected accurately; only the feet go to the right place. Therefore, the red object gets soon split into two parts, and at the end (frame 105), the blue-yellow and the red person are nearly reversed (they actually are if one goes further into the sequence).

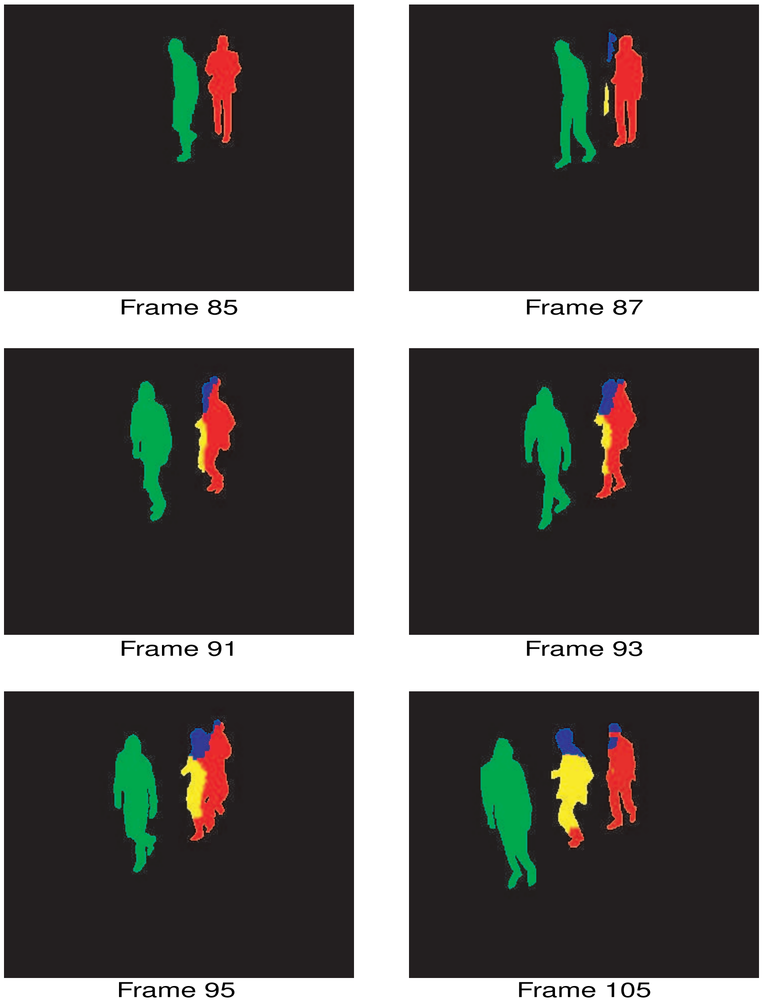


Figure 5.12: Motion estimation errors

This experiment suggests to find a way to get less dependent on the motion estimation results. In the next chapter, we will propose some possible solution to this problem and show how the system may be further enhanced.

5.3 Summary of the results

Even though it was not possible to track some semantic objects in very difficult cases, the overall performance of our system seems promising. Not only does it almost never loose track on independent objects, but there is some support for merged ones as long as the motion is not too complex. Another positive point is the stability of the system: it never crashed during our tests (on a Unix computer).

The implementation of the scene description has been kept very simple because we just wanted to show that it is in fact possible to extract descriptors automatically with our method. Of course, the complete proposed description set should be used whenever necessary.

Chapter 6

Further developments and conclusions

The camera that we have studied in this work proved to be instructive in many ways. First, it gave us an interesting insight into recent image processing techniques while showing their limitations in a clear way. Then, the integration of two different topics, object tracking and scene description, into one application, taught us how to fit systems in a practical way. Finally, our software showed how many details one has to consider to make theory work... In this final chapter, we propose some possible extensions and improvements for our camera and conclude with a summary of the achieved results.

6.1 Possible extensions and improvements

Many of the possible extensions of the camera have already been addressed briefly in the text. There are at least three points that should be investigated for the tracking.

First, the experimental results showed that the tracking accuracy depends a lot on the motion estimation. This is why we lost the object shape in complex merging cases. One trivial method to avoid this would be to replace the Lucas & Kanade algorithm by a more performing one; however, this one may not be easy to find, and there is surely no estimation method that performs well in *any* cases. We could also use some other input to get the subobjects. One possible way is to store object shapes as long as the object is alone and to correct the projected clusters according to the expected shape. Or the shapes can be previously stored in an object database. However, such a system would not be generic anymore.

The Mahalanobis pairing had some difficulties with similar features belonging to different objects. We may look out for possible better thresholds for the disconnection of similar features.

Also, conflicts were resolved by always giving priority to the predicted result over the tracking. Even though this was no problem in our tests, the use of reliability for the labelling decision should be investigated. In fact, the reliability is already available for prediction and can be defined for the pairing.

Another interesting improvement would be to support camera movements or tracking without any object mask. Theoretically, this is already possible if we rely only on the centroids and not on the mask. However, a practical implementation asks for future work.

The descriptor set that we proposed seems adequate for generic object description, but we did not implement and test wide parts of it. So it would be interesting to validate our set on multiple objects, possibly taken out of random video sequences. Also, we provided no tools for scene description, only for its composing objects. So one may add other MPEG-7 descriptors to our subset for this purpose.

6.2 Conclusions

In this work, we put together a centroid-based tracking algorithm and a descriptor set which were both laid out for generic objects. The result has been partially implemented and tested on different video sequences. Even though there were errors, the system showed to be performing on a realistic highway video-surveillance sequence as well as, up to some degree, for the tracking of people interacting with objects. More important, the system does not depend on the actual objects: as long as their interactions are not too complicated, they can be tracked and described.

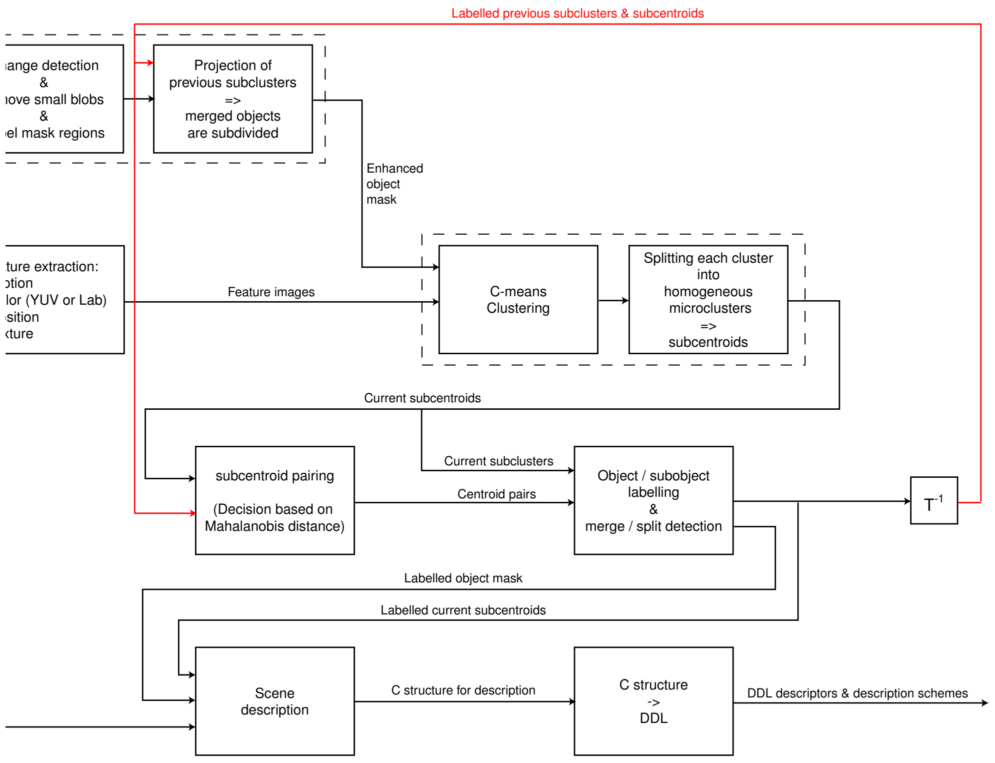
But as impressive as machines “understanding” their surrounding world may be, it will probably take some more time and work to come close to human performance in this field. For sure, there are many open challenges!

Olivier Steiger

Lausanne, 22th February 2001

Appendix A

Block diagram of the camera



Labelled previous subclusters & subcentroids
ange detection & ove small blobs & el mask regions
Projection of previous subclusters => merged objects are subdivided
Enhanced object mask
ture extraction: otion lor (YUV or Lab) sition xture
Feature images
C-means Clustering
Splitting each cluster into homogeneous microclusters => subcentroids
Current subcentroids
Current subclusters
subcentroid pairing (Decision based on Mahalanobis distance)
Centroid pairs
Object / subobject labelling & merge / split detection
T^{-1}
Labelled object mask
Labelled current subcentroids
Scene description
C structure for description
C structure -> DDL
DDL descriptors & description schemes

Bibliography

[1] J. Y. Bernard and A. Cavallaro, “Trajectory of a Video Object in an Outdoor Scene,” Diploma Project, Swiss Federal Institute of Technology, Lausanne, Switzerland, June 1999.

[2] R. Castagno, “Video Segmentation Based on Multiple Features for Interactive and Automatic Multimedia Applications,” PhD thesis, Swiss Federal Institute of Technology, Lausanne, Switzerland, December 1998.

[3] R. Castagno, T. Ebrahimi and M. Kunt, “Video Segmentation Based on Multiple Features for Interactive Multimedia Applications,” in *IEEE trans. on circuits and systems for video technology*, vol. 8, pp. 562–571, September 1998.

[4] M. Sonka, V. Hlavac and R. Boyle, *Image Processing, Analysis, and Machine Vision*. Pacific Grove, CA: PWS Publishing, 1998.

[5] A. V. Oppenheim and R. W. Schafer, *Discrete-time Signal Processing*. Englewood Cliffs, NJ: Prentice Hall, 1989.

[6] T. Sikora, “The MPEG-4 Video Standard Verification Model,” in *IEEE trans. on circuits and systems for video technology*, vol. 7, pp. 19–31, February 1997.

[7] T. Ebrahimi, “MPEG-4 Video Verification Model,” Tech. Rep. N1992, ISO/IEC JTC1/SC29/WG11, San Jose, CA, February 1998.

[8] T. Meier and K. N. Ngan, “Segmentation and Tracking of Moving Objects for Content-based Video Coding,” in *IEEE proc. on Vis. Image Signal Process.*, vol. 146, pp. 144–150, June 1999.

[9] C. Toklu, A. M. Tekalp and A. T. Erdem, “Semi-Automatic Video Object Segmentation in the Presence of Occlusion,” in *IEEE trans. on circuits and systems for video technology*, vol. 10, pp. 624–529, June 2000.

[10] J. Stauder, R. Mech and J. Ostermann, “Detection of Moving Cast Shadows for Object Segmentation,” in *IEEE trans. on multimedia*, vol. 8, pp. 65–76, March 1999.

[11] E. Wan, C. Seyrat, C. Thiénot and F. Nack, “CD 15938-2 – MPEG-7 Multimedia Content Description Interface – Part 2 Description Definition Language,” Tech. Rep. N3702, ISO/IEC JTC1/SC29/WG11, La Baule, France, October 2000.

[12] L. Cieplinski, M. Kim, J.-R. Ohm, M. Pickering and A. Yamada, “CD 15938-3 – MPEG-7 Multimedia Content Description Interface – Part 3 Visual,” Tech. Rep. W3703, ISO/IEC JTC1/SC29/WG11, La Baule, France, October 2000.

[13] L. Cieplinski, S. Jeannin, M. Kim and J.-R. Ohm, “MPEG-7 Visual Part of eXperimentation Model Version 7.0,” Tech. Rep. N3521, ISO/IEC JTC1/SC29/WG11, Beijing, China, July 2000.

[14] P. van Beek, A. B. Benitez, J. Heuer, J. Martinez, P. Salembier, J. Smith and T. Walker, “MPEG-7 Multimedia Description Schemes Working Draft (Version 4.1),” Tech. Rep. M6477, ISO/IEC JTC 1/SC 29/WG 11, La Baule, France, October 2000.

[15] C.A. Poynton, “A Technical Introduction to Digital Video,” John Wiley, New-York, NY, 1996.

[16] C. Secrétan and A. Cavallaro, “Representation of Shapes for Multimedia Applications,” Semester Project, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1999.