Linköping Studies in Science and Technology.
Ph.D. Thesis. Dissertation No. 733

# Model-Based Head Tracking and Coding

**Jacob Ström**


**INSTITUTE OF TECHNOLOGY**
**LINKÖPINGS UNIVERSITET**

Department of Electrical Engineering
Linköping University, S-581 83 Linköping, Sweden

Linköping 2002

ISBN 91-7373-260-5
ISSN 0345-7524
Printed in Sweden by Linköpings Tryckeri AB (LTAB) 2002.

# Abstract

This thesis treats two topics in model-based coding; coding of facial textures (Part 1) and real-time head tracking (Part 2).

In Part 1, it is shown that a face image can be efficiently coded using a parameterization based on geometrical normalization followed by Karhunen-Loève transformation (KLT). The resulting parameterization is shown to be convex, and can be used for coding: It improves both the measured and perceived quality of face images compared to only using eye matching normalization followed by KLT. A block based version of the coder improves the performance dramatically both in terms of quality and complexity. By distributing the bits unevenly over the face, quality can be further improved. Comparisons with JPEG show an improvement of 8 dB.

In Part 2, a real-time head tracker that is robust to large rotations is described. The system uses a large number of automatically selected feature points, constrained by dynamically estimated structure. The structure from motion (SfM) algorithm described by Azarbayejani and Pentland in 1995 is used. The algorithm is first examined for planar objects, and then extended to manage points not visible in the first frame. The extended SfM method is the basis for the head tracker: A texture mapped three-dimensional head model is created, and 24 feature points on the surface of this model are automatically selected and tracked. The trajectories of the tracked feature points are forwarded to the SfM algorithm, which in turn provides estimates of the location of the feature points in the next frame. By adaptively updating the texture, the extended SfM algorithm can be used to track points on, e.g., the side of the head, which improves the range of the tracker. A reinitialization procedure that uses data from the original initialization is also presented. The complete system runs at full frame rate (25/30 Hz) and is evaluated on both real and synthetic data.

# Acknowledgements

First of all I would like to thank my supervisor Dr. Robert Forchheimer for his help and hospitality. I am especially grateful for his informal manners, which have made this process very pleasant. I would also like to thank Prof. Sandy Pentland for accepting me as a visiting student in his group. The year at MIT was stimulating, rewarding and most of all fun. A thank to all my colleagues, especially to Prof. Haibo Li who acted as an extra supervisor to me before he moved to Umeå, and to Jörgen Ahlberg with whom I shared both office and an interest in facial image processing. My thanks also go to Dr. Tomas Möller for endless proofreading and to Sumit Basu for stimulating feedback. I would also like to thank Ericsson, where I have completed the editing of this thesis. Finally I would like to thank all my supporting friends and family members, and especially my sisters Anna and Karin and my parents Helge and Eva.

# Contents

# Chapter 1

# Introduction

## 1.1 Motivation

This work deals with two topics; compression of facial textures and real-time head tracking. The starting point for this work has been video telephony for low bandwidth channels, such as the 56 kbit/s modem channel provided by the public switched telephone network. One technique that promises video communication over such low bit rates is *model-based coding*, shown in Figure 1.1. The encoder extracts the motion of the head (global motion), the motion of the

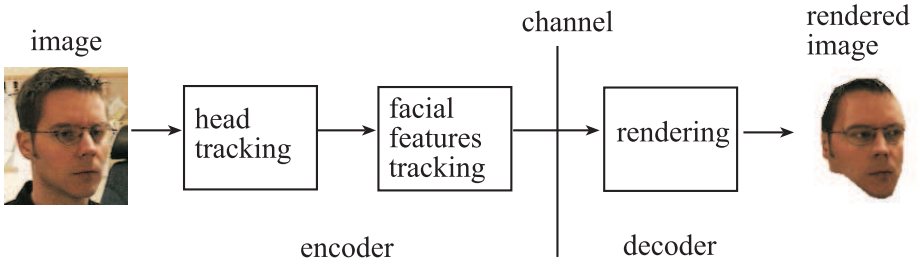



Figure 1.1: *Simplified diagram of a model-based coder.*

facial features (local motion) as well as the texture of the head model. These parameters are sent over the channel to the decoder that uses them to animate a three-dimensional model of the head. The idea is that the parameters can be sent with significantly fewer bits than the images themselves. The first part of this thesis will deal with the efficient transmission of facial textures, whereas the second part will treat the real-time extraction of the (global) motion of the head.

## 1.2 Problem Formulation

In this thesis, the *facial texture compression problem* is formulated as:

DEFINITION 1 *Given a gray-scale image of a human face and the locations of certain feature points within that image (such as the position of the left corner of the mouth), find a representation that is good in terms of coding efficiency, measured in Peak Signal to Noise Ratio (PSNR) and bits per pixel (bpp).*

The study is limited to gray-scale images, but it is assumed that color images can be treated the same way. The main emphasis is to find a good representation in terms of coding efficiency, but issues such as computational complexity and possible quality range will also be treated.

The following problem, called the *head tracking problem*, will also be addressed:

DEFINITION 2 *Given a monocular image sequence of a moving head, estimate in real-time the three-dimensional pose (position and orientation) of the head, given that the pose is known in the first image (the head is in a prespecified position, facing the camera and has a certain size on the screen).*

*Monocular* means that the image sequence is taken from a single video camera, and not from a stereo camera rig. Video telephony is potentially a low cost product and should not require expensive two-camera systems. *Real-time* here means that the average time to process each image is less than 40 ms (33 ms for an NTSC system) which means that the system runs at 25 Hz (30 Hz). The desired output is three-dimensional pose, i.e., the simple two-dimensional position of the head in the image is not sufficient, but the three-dimensional coordinates and angles of the head should be estimated. This is needed for correct animation of the computer model. The pose is assumed to be known in the first image, hence it is a *tracking* system that is considered, rather than a *face finding* system.

## 1.3 Contributions

The first part of this thesis deals with the compression of facial texture images using the Karhunen-Loève transform (KLT, also called principal component analysis, PCA, see Appendix A). The contributions are:

- The usage of geometrical normalization for face image coding.
- The usage of modular eigenspaces for face image coding.
- Global bit allocation for face images.

The two first results were presented in an international workshop paper [63], and all three items were included in the licentiate thesis[1] of the author [64]. Since then, the analysis and representation of face images using PCA have received much attention due to the stellar work on active appearance models by Tim Cootes et al. [17]. This can make the results in the first part of the thesis look dated, especially Chapter 5 which deals with automatic extraction and refinement of facial feature locations.

The second part of the thesis deals with head tracking. The proposed system is illustrated in Figure 1.2. It starts with an initialization procedure, (*init*

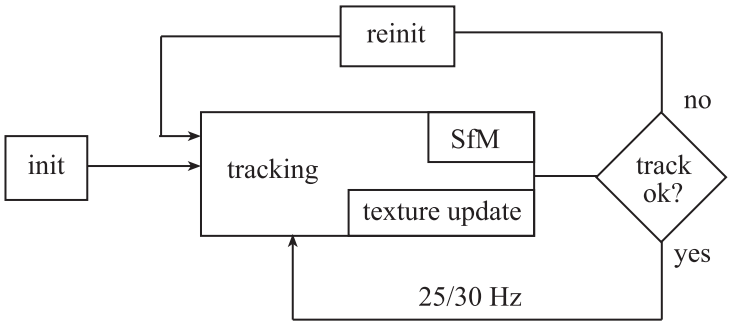



Figure 1.2: *Proposed head tracking system.*

box in Figure 1.2), which assumes that the head is in a prespecified starting position, roughly covering a projected head model on the screen, and facing the camera. In the *tracking* block the head is tracked. The *SfM* (structure from motion) algorithm and the *texture update* scheme are vital ingredients here. They are described in Chapters 7 and 10 respectively. The algorithm then decides whether the tracking was successful (the *track ok* diamond). If so, it continues tracking at full frame rate (25/30 Hz). Otherwise, the reinitialization procedure is invoked (*reinit*) before the tracking is continued.

There are four contributions in this part.

- The system itself is a major contribution. The basic parts of it were published at a workshop at ICCV 1999 [66].
- Many SfM algorithms break down for planar surfaces. Since the points on the face can be close to coplanar, Chapter 8 treats how the well-known algorithm by Azarbayejani and Pentland [8] behaves in this case. This has previously not been reported in the literature. The planar case is found to converge to the correct solution, albeit at a slower pace and not quite as reliably as in the general case. Tentative results have been published by the author at a national conference [69] but only for the noise free case.

[1]The licentiate thesis is a Swedish intermediate degree between M.Sc. and Ph.D.

- The method of adding feature points to the Structure from Motion algorithm of [8] that is described in Chapter 9 is novel. This makes it possible to do texture updates on the side of the head and to track points on, for instance, the ear. This greatly extends the range of the tracker, as shown in Chapter 12. The method is presented in a technical report [67] but is otherwise unpublished.

- The reinitialization system component presented in Chapter 11 is novel. It was published at the refereed international conference EuroImage ICAV3D 2001 [68].

Part of the work on head tracking was carried out when the author was a visiting student in Alex Pentland's group at the MIT Media Lab in Cambridge, MA.

Outside the scope of this thesis, the author has also produced a journal paper on combined lossless/lossy coding for medical images [65]. This work was done while at P. Cosman's group at University of California, San Diego.

## 1.4 Outline

The thesis is arranged as follows: The rest of this chapter is devoted to a brief introduction to image and video coding in general, and to model-based coding in particular.

The first part of the thesis treats facial texture compression, and starts with a chapter that discusses how face images can be coded using the Karhunen-Loève transform. This chapter also introduces a geometrical normalization step and investigates how this step can improve the coding. Chapter 3 presents a way to enhance image quality significantly, and at the same time lower complexity, by tiling the image into blocks prior to the Karhunen-Loève transform. The resulting method is compared to JPEG. The following chapter discusses the benefits of distributing the bits non-uniformly among the blocks. The resulting improvement in quality is measured. In Chapter 5, an automatic way to refine the feature point positions used by the geometrical normalization is treated. Complexity and gain in quality are discussed.

The second part of the thesis deals with head tracking, and starts with an overview of the different head trackers in the literature and how they work. It ends with a motivation for the proposed solution of the head tracking problem. Since the Structure from Motion processing is a vital part of the system, the following chapter treats this problem, first with the help of extended Kalman filtering and then using multilinear constraints. The methods are compared for degenerate motion and degenerate objects in Chapter 8. The Kalman technique is then extended in Chapter 9 so that it can handle new points that are not visible in the first frame. Chapter 10 presents the tracking algorithm, including details on how the confidence of the point matchings is estimated and fed into

the Kalman filter. This is followed by a chapter on how to reinitialize the tracker when it fails. In Chapter 12, the complete system is evaluated on both real and synthetic data. The last chapter contains a conclusion of both Part I and Part II.

Appendix A is included to quickly update the reader on the theory behind and the relations between the Karhunen-Loève transform, principal component analysis and singular value decomposition. Appendix B goes through the theory of optical flow that is used in Chapter 6. Appendix C contains an introduction to Kalman filtering and its extension to the non-linear case. Finally, Appendix D goes through some of the proofs of Chapter 10.

## 1.5 Model-Based Video Coding

A standard TV set deceives the eye to perceive motion by displaying a large number of static images each second. This results in huge amounts of data. The raw data rate of the PAL TV signal exceeds 230 Mbit/s, about 4100 times more than what is currently possible to send over a normal PSTN modem channel (56 kbit/s). The objective of video coding is to reduce the number of bits needed to transmit the image sequence. Currently, hybrid coding is the most widely used technique for video coding, named after the hybrid combination of motion compensated predictive coding and discrete cosine transform. One example is digital broadcast TV, where the PAL signal mentioned above is sent using MPEG-2, which is based on hybrid coding. The rate is reduced to around 4 Mbits/s, a compression of about 60 times.

Although the compression performance of hybrid coding is impressive, the bit rate produced is still about 70 times larger than what can be sent over a telephone modem. As pointed out by Li et al. [41], it is generally impossible to code a full TV signal with high quality at such low rates. The contents of a video telephony sequence however, are known roughly à priori; such a sequence is likely to contain a head-and-shoulder shot of a talking person. This information can be used to enhance the compression efficiency of the coder.

Model-based coding takes advantage of the fact that the image to encode is a two-dimensional projection of a three-dimensional object. Among other model-based techniques, *semantic coding* assumes that the image depicts a specific three-dimensional object, typically the head-and-shoulder part of a person. (For other model-based techniques, see [51].) The coder tries to describe the image sequence by animating a computer model that resembles the person in the scene. By transmitting the animation parameters instead of the images themselves, extremely low bit rates can be achieved. Different studies [3, 41] show that the motion information needed to animate a human face can be transmitted at a rate of about 30 bits per frame, which at 25 frames per second requires a channel capacity of 750 bits per second. This is about 1.3% of the 56 kbit/s capacity and a proof of concept of video telephony for modem type channels.

Figure 1.3 shows the basic idea of a model-based coder. The top half (above the channel box) represents the encoder. In the first image, a wire frame model is adapted to the face, and the texture of the face is also extracted. These two pieces of information constitute the parameter vector $\varphi$ in Figure 1.3. Since $\varphi$ will not change during the course of the image sequence, it needs only be sent once. During the sequence, the head will rotate and translate (global motion) and the face will deform, e.g., when the person smiles (local motion). Therefore, motion parameters (animation parameters) $\phi_k$ are extracted for each frame $k$, and sent over the channel. The decoder is illustrated in the lower part of Figure 1.3. The information from the two parameter vectors $\varphi$ and $\phi_k$ is used to render frame $k$. Note that all the parts of the image for which there is no model are absent — for example the background and the inside of the mouth.

A scheme such as the one described above was presented in 1983 [23] and refined in 1984 [24] by Forchheimer et al., but suffered from the inability to portray realistically the skin and other parts of the face since the model was made up of single colored polygons (see left part of Figure 1.4). In 1987 Welsh [74] and Aizawa et al. [5] introduced texture mapping, which profoundly improved realism. The development of computer graphics has been very rapid and today realistic rendering of such a complex object as a head can be performed by inexpensive graphics cards for the PC and even by video game platforms. Recent work in MPEG-4 [33] has put forward a standard way of representing the animation parameters. Thus it is fair to say that the decoder part of model-based coding is more or less solved. On the encoder part however, several components can be identified.

- *Face detection* is needed in order to know if there is a face in the image and where it is located.
- *Model initialization* places the facial model correctly in three dimensions and possibly also changes the shape of the three-dimensional model in order to make it conform with the geometry of the face.
- *Facial texture coding* is needed to compress and send the texture of the face to the decoder.
- *Head tracking* (or global motion tracking) makes sure that the model mimics the three-dimensional rigid body motion of the head.
- *Facial gestures tracking* (or local motion tracking) extracts facial motion such as smiles and eye blinks.
- *Model refinement* continuously refines the three-dimensional model by adding texture and changing the shape.

The goal of this thesis is to solve the problem of facial texture compression and the problem of head tracking. The methods that have been developed to do

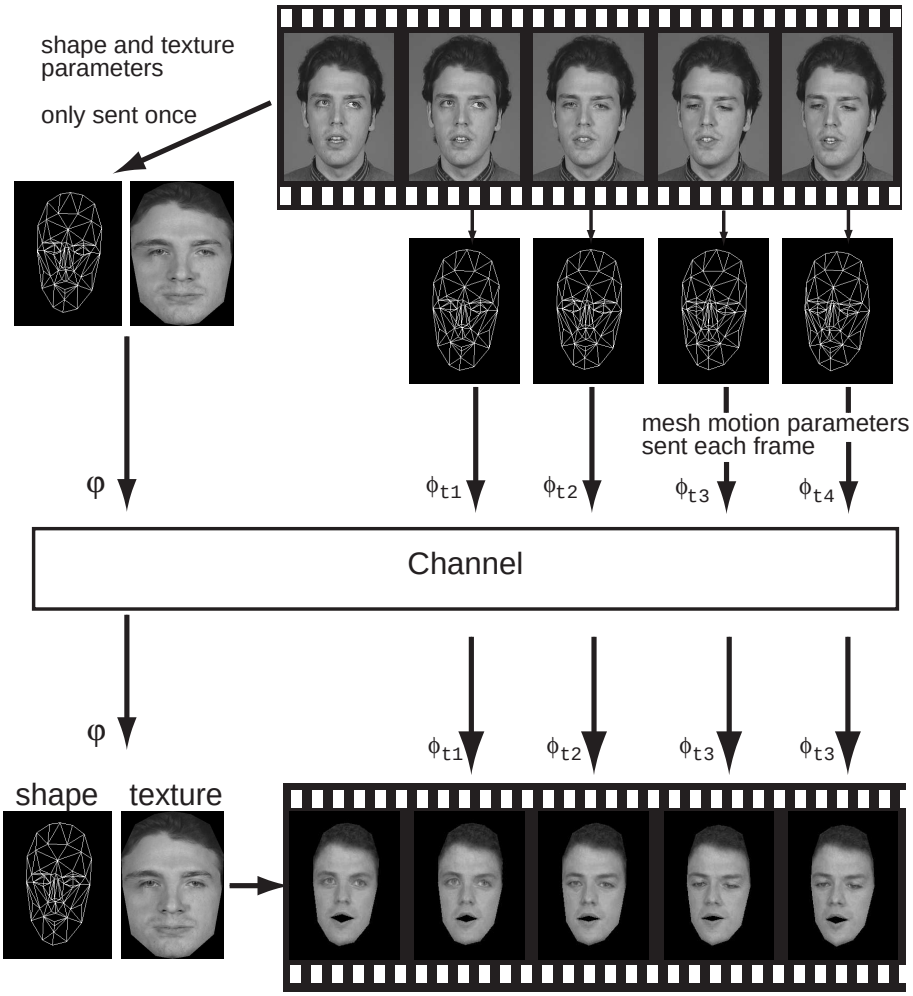



Figure 1.3: *Basic idea of a model-based coding system. The image is analyzed, and the shape and texture parameters are sent only once. For each frame, motion parameters are sent that tell the decoder how to animate the face.*

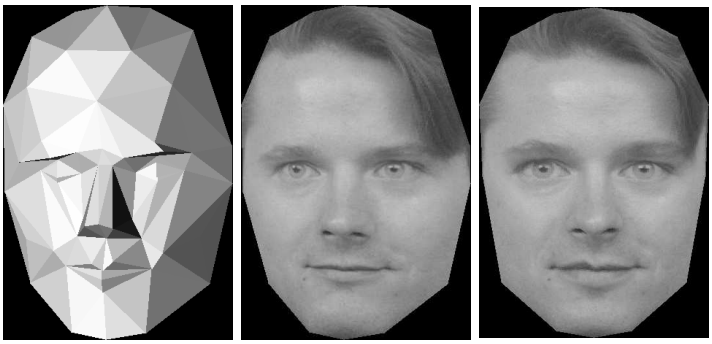


Figure 1.4: *Texture mapping significantly improves the photorealism of the decoded image in model-based coding. Left: without texture mapping. Middle: with texture mapping. Right: the texture.*

this, however, touch upon several of the other above-mentioned problems. For instance, the reinitialization method in Chapter 11 solves the problem of face detection and model initialization under the assumption that a good model of the head already exists. The adaptive texture update mentioned in Chapter 10 and the depth estimation of the feature points in Chapter 7 are examples of model refinement.

# Part I

# Facial Texture Coding

# Chapter 2

# Frame Based Texture Compression

## 2.1 The Face Image Set

We will regard an image $x[i,j]$ supported on, say $[512\times512]$ pixels, as a vector $\bar{x}$ in an $n = 262144$ dimensional vector space called *image space*. Each face image corresponds to a point in image space, and the set of all possible face images will form some region in image space, called the *face image set*. If this set is a convex sub-region of low dimension, a small number of basis vectors $(\phi_k)_1^M$ will suffice to represent well any face image $\bar{x}$ by

$$\bar{x} \approx \hat{x}_M = \sum_{k=1}^{M} \alpha_k \bar{\phi}_k. \tag{2.1}$$

Then the basis vectors $(\bar{\phi})_1^M$ can be stored in both the encoder and the decoder, and to transmit the face image, only the $M$ coefficients $(\alpha_k)_1^M$ need to be sent, instead of $n$ pixels. If the rate is measured as the number of values that are sent, then since $M \ll n$, a considerable compression has taken place. If the $\bar{\phi}_k$s are made orthonormal, it will also be easy to calculate the coefficients $\alpha_k = \bar{\phi}_k^T \bar{x}$. The system is shown in Figure 2.1. Note here that the *database*, i.e., the set of basis vectors $(\bar{\phi}_k)_1^M$, is general for all face images and can therefore be prestored in both the encoder and the decoder; it need not be transmitted. The two interesting questions are now: Is the face region set convex and of low dimensionality, and if so, how does one find a suitable database of basis vectors $(\bar{\phi}_k)_1^M$? These two questions will be answered in the following two sections, in reversed order.

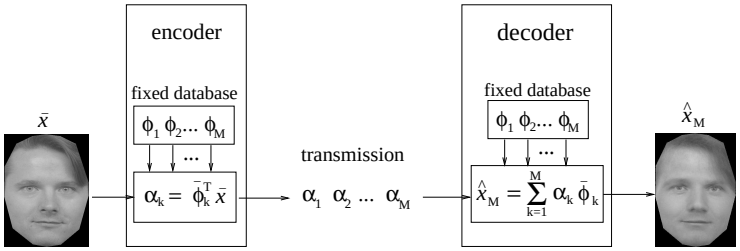



Figure 2.1: *Structure of a face image coding system for very low bit rates: The inner product is formed between the image and the basis vectors $\bar{\phi}_k$, yielding the $M$ coefficients $(\alpha_k)_1^M$, which are transmitted. Since $M$ is much smaller than the number of pixels, the result is a considerable compression. The decoder linearly combines the basis vectors $\bar{\phi}_k$ with the $\alpha_k$ as weights. The result is the reconstructed image $\hat{x}_M$.*

## 2.2 The Karhunen-Loève Transform

In Appendix A, the image is treated as a vector valued random variable. Using the *Karhunen-Loève transform*, it is possible to obtain a set of basis vectors $(\bar{\varphi}_k)_1^n$ with the attractive property of minimizing the expected mean-square error[1] (MSE): If $\bar{x}$ is expressed in this basis, $\bar{x} = \sum_{k=1}^{n} \alpha_k \bar{\varphi}_k$, the error $E\{|\bar{x} - \hat{x}_M|^2\}$ due to the truncation $\hat{x}_M = \sum_{k=1}^{M} \alpha_k \bar{\varphi}_k$ of the sum is smaller than (or equal to) the error achieved with any other orthonormal set of basis vectors $(\bar{\phi}_k)_1^n$, for all truncations $0 \leq M \leq n$. This should make $(\bar{\varphi}_k)_1^M$ suitable as the basis vectors in Equation 2.1.

The $\bar{\varphi}_k$s can be calculated using principal component analysis (PCA), which in essence is an eigenvalue decomposition of the covariance matrix of the pixels in the image. The covariance matrix is estimated from a large collection of images, called the *training set*. Since this training set is finite, the covariance matrix, and hence the basis vectors derived from it, will be approximate. To indicate this, the basis vectors are written $\hat{\varphi}_k$ rather than $\bar{\varphi}_k$. The difference between theory and practice is illustrated in Figure 2.2. The basis vectors $\bar{\varphi}_k$ (and their approximations $\hat{\varphi}_k$) can be called either principal components, eigenvectors, eigenimages or eigenfaces. Throughout this thesis they will be called eigenimages or, more generally, basis vectors. Note that if there are $N$ images in the training set, it is only possible to estimate $N$ basis vectors $(\hat{\varphi})_1^N$. Since $N < n$, these vectors will not span the entire image space. (The term basis vector is thus somewhat inaccurate but will still be used in this thesis.) Still, if the face image set is convex and of sufficiently low dimension, all face images will be well represented. Note also that the theory in Appendix A is valid under the assumption that the images have zero mean. This is achieved by calculating the average image of the training set and subtracting this average image from all the images prior to the calculation of the basis vectors.

[1] This means that if the rate is measured as the number of values sent, then the Karhunen-Loève transform is optimal in a rate-distortion sense.

| | | | | | | |
|---|---|---|---|---|---|---|
| THEORY: | | | | | | |
| $\bar{\mathbf{X}}$ | $\rightarrow$ | $C_{\bar{\mathbf{X}}}$ | $\rightarrow$ | $(\bar{\varphi}_k)_1^n$ | $\rightarrow$ | $(\bar{\varphi}_k)_1^M$ |
| random vector | | covariance matrix | | basis vectors | | data base |
| PRACTICE: | | | | | | |
| $(\bar{x}_j)_1^N$ | $\rightarrow$ | $\hat{C}_{\bar{\mathbf{X}}}$ | $\rightarrow$ | $(\hat{\varphi}_k)_1^N$ | $\rightarrow$ | $(\hat{\varphi}_k)_1^M$ |
| training set | | covariance matrix estimate | | basis vectors | | data base |

Figure 2.2: *The difference in notation to distinguish theory from practice.*

## 2.3 The Structure of the Face Image Set

As pointed out by Bichsel and Pentland [12], large translations or scale-changes make the face image set highly non-convex and hence not simply shaped. Therefore, previous efforts to encode face images with eigenimages have included a normalization step that is performed on all images in the training set and on the image to be encoded. (Additional information must then be transmitted to the decoder to enable it to undo this normalization before displaying the image.) Kirby and Sirovich, who presented the first work [39] on the usage of Karhunen-Loève techniques for face image coding, normalized their images by fixing the eyes to a template. This will be referred to as *eye matching* normalization. Moghaddam and Pentland, who used the technique for automatic model-based coding of face images [47], used an affine coordinate transform, (referred to here as *affine normalization*), allowing for the additional fixing of a third point. However, this section will show that even after normalization with eye matching, the face image set will be non-convex, and the same is true for the images after affine normalization using analogous reasoning.

In the example in Figure 2.3 we have chosen to fix the eyes. Since the eye-mouth distance varies among people, it will be possible to find two images, $\bar{x}_A$ and $\bar{x}_B$, in the normalized face image set with the mouths at different positions. If the normalized face image set were convex, all convex combinations $\bar{x}_C = \alpha\bar{x}_A + (1-\alpha)\bar{x}_B, \quad \alpha \in [0,1]$, (i.e., all points on the dotted line in the diagram in Figure 2.3) must belong to the set. But letting $\alpha = \frac{1}{2}$ means that the image $\bar{x}_C$ would result in an image with two mouths, which is clearly not a face, and thus the set cannot be convex. Affine normalization allows for the fixation of a third point, and will consequently cope with different ratios between eye-eye distances and eye-mouth distances. However, the ratio between the eye-mouth distance and the eye-nose distance also varies among people, and this cannot be compensated for by an affine normalization, since it involves a fourth point. Hence, the face image set after affine normalization will also be non-convex.

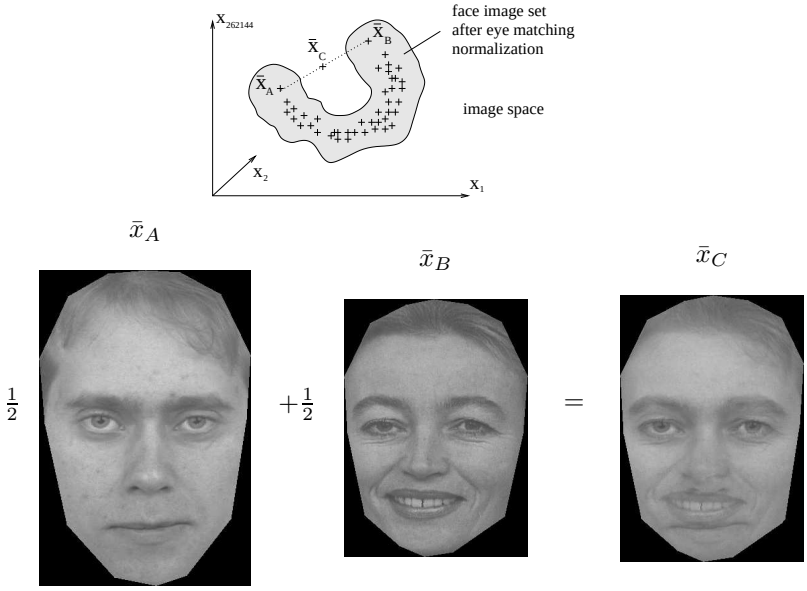



Figure 2.3: *In the face image set after eye matching normalization, depicted in the top diagram, two images, $\bar{x}_A$ and $\bar{x}_B$, with different eye-mouth distances are found. If the face image set is convex, then all convex combinations $\bar{x}_C = \alpha\bar{x}_A + (1-\alpha)\bar{x}_B, \quad \alpha \in [0,1]$, (corresponding to all points on the dotted line), must belong to the set. But letting $\alpha = \frac{1}{2}$ will result in an image $\bar{x}_C$ that is not a face (since it has two mouths, as illustrated in the lower part of the figure). Thus this convex combination is not part of the set and the set cannot be convex.*

The non-convexity implies that a linear combination, such as the one in Equation (2.1), cannot represent the data satisfactorily. Either it will not be able to represent both small and large eye-mouth distances, or the base will be over-complete, capable of representing two-mouthed creatures that do not exist. Different facial expressions, such as eye blinks and open/closed mouths, will reveal the same kind of sub-optimality in the representation.

This problem can also be illustrated as follows; since, according to Appendix A, each basis vector is a linear combination of the images in the training set, each small area of the face image is coded as a linear combination of what is in the same area in the images of the training set. Typically, a specific part of the face, such as the lower lip, should correlate stronger with the lower lips of the images in the training set than with other parts of those images. This calls for an alignment procedure that is more powerful than the one in Figure 2.3, where the lower lip in image $A$ is in the same spot as the cheek in image $B$.

## 2.4 Geometrical Normalization

An approach for a refined normalization can be seen in Figure 2.4. In the origi-

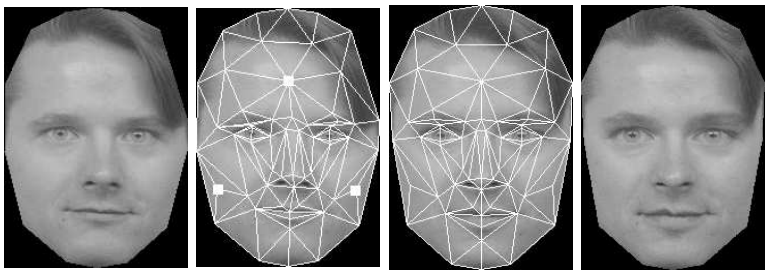


Figure 2.4: *In the original image (first image), a triangular mesh is fitted to feature points in the face (second image). The triangle-vertices are then moved to predetermined positions, and the texture is mapped accordingly (third image). The result is a geometrically normalized image (fourth image).*

nal image (first image), a triangular mesh (CANDIDE [54]) is fitted to the face by manually moving each vertex in the CANDIDE-mesh to a specific feature point in the face (second image). Examples of such feature points include the corners of the eyes, the nostrils etc, but also more loosely defined points such as points on the hairline or on the outline of the face. Three vertices (marked with boxes in the second image) are not fitted to any particular points in the face, but are positioned roughly in the center of the surrounding triangles. The manual operator also makes sure that the fitted wireframe has the same topology as the original CANDIDE-mesh. Next, each triangle is texture mapped, using the part of the image that is directly underneath the triangle as texture. Each vertex is then moved to a predetermined position, namely the average position of this

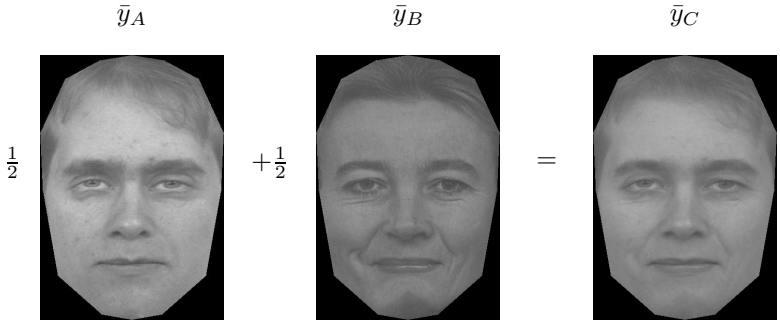



Figure 2.5: *$\bar{y}_A$ and $\bar{y}_B$ are the geometrically normalized versions of the images $\bar{x}_A$ and $\bar{x}_B$ in Figure 2.3. Note that the image $\bar{y}_C = \frac{1}{2}\bar{y}_A + \frac{1}{2}\bar{y}_B$ is a valid face image.*

vertex over a large set of images (third image). This changes the shape of the triangles, and their texture is therefore remapped using nearest neighbor texture mapping. The result is a geometrically normalized image (fourth image), where each facial feature has a predetermined position. Henceforth, this normalization method is called *geometrical normalization* since it normalizes the geometrical distances (e.g., nose/mouth distance) in a face image. In fact, the geometrical normalization normalizes (fully or to some extent) differences due to three different things: Firstly, it normalizes global motion, such as head rotation, translation and scaling. Secondly, it normalizes local motion due to facial expression etc, so that the face will look neutral after the normalization. Thirdly, it also normalizes the geometrical differences between individuals; some people have a short distance between their eyes and their nose, others have a shorter upper lip instead but the same length of the face. After the normalization, they will all have the same geometry.

The following notation will be used: $\bar{x}$ will denote an image before normalization and $\bar{\theta}$ will denote the vector containing the coordinates of the feature points in $\bar{x}$. The mapping of an image $\bar{x}$ to the geometrically normalized version $\bar{y}$ will be denoted $\bar{y} = GN(\bar{x}, \bar{\theta})$. The Karhunen-Loève transform will thus be performed on the vector $\bar{y}$; $\hat{y} = \sum_{k=1}^{M} \alpha_k \hat{\varphi}_k$, where $\alpha_k = \bar{y}^T \hat{\varphi}_k$.

Figure 2.5 shows geometrically normalized versions $\bar{y}_A$ and $\bar{y}_B$ of the images $\bar{x}_A$ and $\bar{x}_B$ from Figure 2.3. The open mouth of image $\bar{x}_B$ is now closed, which makes it more sensible to superimpose the two mouths[2]. Note how the image $\bar{y}_C = \frac{1}{2}\bar{y}_A + \frac{1}{2}\bar{y}_B$ is now a valid face image, and that the convexity condition then seems to hold. This experiment has been repeated 100 times for different images, each time resulting in a valid face image. Thus there is reason to believe that the face image set is more convex after geometrical normalization than after eye matching, and that it will be easier to find linear basis functions

[2]The interior of the mouth is thus not coded. However, it is possible to encode the interior of the mouth with a separate set of basis vectors.

that can well represent images from this set. The top row in Figure 2.6 shows five images from the training set after they have been geometrically normalized. The bottom row shows the average image followed by the first four eigenfaces.

Figure 2.6: *Top: some images in the training set (geometrically normalized). Bottom: the mean image (left) followed by the first four eigenfaces.*

## 2.5 Quality Improvement

If the geometrical normalization step makes it easier to find basis vectors that can span the face image set, it should be possible to measure an increase in quality in the encoded images. Of course, the geometrical normalization also involves a rate penalty for the transmission of the feature point data that is needed to undo the normalization in the decoder. Whether the rate penalty is worth the increase in quality is discussed in Section 2.6.

To evaluate how much can be gained by using geometrical normalization instead of eye matching normalization, the following experiment was conducted: An original image was first geometrically normalized, then coded as a linear combination of $M = 198$ eigenimages calculated from a geometrically normalized training set. (The image to be coded was not part of the training set.) For simplicity, the size of the database, $M$, was equal to the size of the training set, $N$, and the coefficients were not quantized. Finally, the coded image was inversely normalized and compared to the original image by calculating the Peak Signal to Noise Ratio, ($\text{PSNR} = 10\log(255^2/\text{MSE})$). The image was then coded a second time using the same training set, this time with eye matching normalization replacing geometrical normalization in all steps, resulting in a second PSNR value. The gain from using geometrical normalization could be calculated

by forming the difference between the two PSNR measures. This process was repeated for a set of 100 different images, and the result can be seen in Figure 2.7. The average gain in PSNR is rather modest (only 0.5 dB), but the distortion in

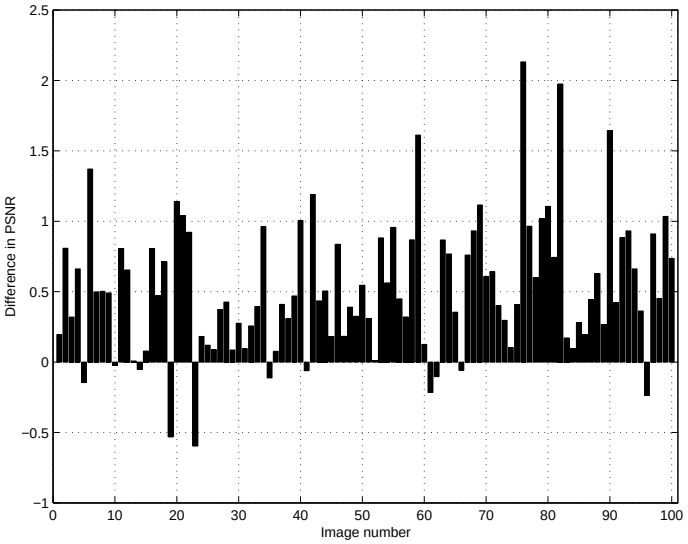



Figure 2.7: *The gain in* PSNR *from using geometrical normalization as opposed to eye matching normalization, measured over a set of 100 images.*

the geometrical normalization case is different; the facial features such as the nostrils and the mouth are of higher contrast, and typically the ringing around edges such as the face contour and the mouth is diminished. The hair line is less distinct, but since this is not of such vital perceptual value the result indicate a clear increase in visual quality despite the modest gain in PSNR as shown in Figure 2.8. The original image (left) is coded using eye matching normalization (middle) and geometrical normalization (right). The geometrical normalization improves this image by about 0.8 dB, but the difference is perceptually visible when comparing, for instance, the nose and the mouth regions.

## 2.6 Quantization Results

To investigate whether this quality increase is worth the extra cost of sending the feature points needed for the geometrical normalization, quantized data must be considered. Figure 2.9 shows examples of three images that are represented with quantized coefficients. The original images in the first column (not part of the training set) are geometrically normalized (second column) and then projected onto a database of $M = 77$ eigenfaces calculated from a training set of $N = 200$ images. Each coefficient is divided by its standard deviation (the square-root of the corresponding eigenvalue from the KL calculation) and

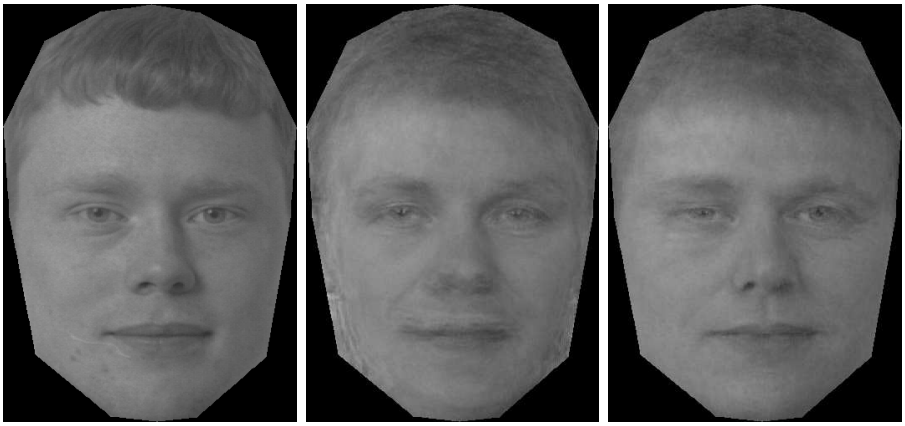


Figure 2.8: *The original image to the left (number two in the diagram in Figure 2.7) is coded using eye matching normalization (middle) and geometrical normalization (right). The right image is 0.8 dB higher in* PSNR. *Note the differences in the distortion of the nose and mouth regions.*

then scalar quantized, using a Lloyd-Max quantizer for a Gaussian source. The number of bits allocated to a specific coefficient is chosen according to

$$b_k = b + \log_2 \frac{\sigma_k}{\rho}, \tag{2.2}$$

where $b_k$ is the number of bits to quantize coefficient $\alpha_k$, $b$ is the average number of bits per coefficient, $\sigma_k$ is the square-root of the corresponding eigenvalue and $\rho$ is the geometrical mean of all $\sigma_k$s. The bit assignment vector is processed in a number of steps to remove irregularities such as negative bit assignments, single bits, etc. The same procedure is conducted in the decoder, which is possible since the eigenvalues are prestored in the decoder. In the example in Figure 2.9, 300 bits were distributed among the 200 coefficients. For this rate, only 77 of the 200 coefficients were assigned bits, which means that the size of the database, $M$, equals 77. Another 154 10-bit integers of feature point data (1540 bits) are sent to perform the inverse geometrical normalization. The images in the fourth column can thus be represented with a total of 1840 bits. At a first glance, it seems that the expected quality gain of 0.5 dB is not worth the extra 1490 bits[3] needed for the improved normalization, but there are several reasons why the situation in fact favors geometrical normalization. First, in the current scheme, no effort has been made to compress the feature point data; 1490 bits should be considered to be an upper bound rather than a realistic estimate of what is required to send the data. Second, the feature point data can be derived from the three-dimensional shape of the head and the motion data. The three-

[3]The positions of the eyes would require four 10-bit integers = 50 bits. Extra bits needed to replace eye matching normalization with geometrical normalization are thus $1540 - 50 = 1490$ bits.

Figure 2.9: *Examples of quantized data: The original images (first column), not included in the training set) are geometrically normalized (second column), these textures are then coded using* $N = 200$, $M = 77$ *at 300 bits (third column). By performing inverse geometrical normalization, the fourth column is obtained. Another 1540 bits are needed for this step.* PSNR *are (from top to bottom) 31 dB, 29 dB and 29 dB, rates 0.018 bpp, 0.015 bpp and 0.016 bpp, respectively, counting only foreground pixels. Total size for each of the three images is 1840 bits, or 230 bytes, which is about half of the number of letters in this figure text.*

dimensional shape data will not change over time and need only be sent once; its rate penalty can thus be neglected. The motion information can be transmitted losslessly at around 400 bits per frame [3], and at lossy quality at such low rates as 30 bits per frame.
Third, and perhaps most important, if this algorithm is to be incorporated in a model-based coding scheme, it will be used for face *texture* compression rather than face *image* compression. Only the texture map of the triangles needs to be transmitted, a task that is fulfilled already in the third image in Figure 2.9. This means that facial texture coding can be added to a model-based coding scheme at an extra cost of only 300 bits per texture. Henceforth, the coding of texture plus geometry information will be called face image coding while coding of only the texture will be called face texture coding.

## 2.7 Inverse Geometrical Normalization

On the decoder side, the geometrically normalized image $\bar{y}$ is reconstructed as a linear combination of basis vectors $\hat{\varphi}_k$ using the transmitted coefficients as weights: $\hat{y} = \sum_{k=1}^{M} \alpha_k \hat{\varphi}_k$. The image is then inversely geometrically normalized, going through the steps of Figure 2.4 in reverse order. It is important to point out that the normalization step itself is likely to introduce errors in the image. This is because some of the triangles are shrunk during the forward geometrical normalization. The information that is lost hereby cannot be regained in the inverse geometrical normalization. To indicate this, the inverse geometrical normalization mapping is denoted $GN^{+}(\bar{y}, \bar{\theta})$ rather than $GN^{-1}(\bar{y}, \bar{\theta})$. The error introduced by the geometrical normalization is a lower bound on the total error of the system. In order to estimate this lower bound, the error $\varepsilon^2 = \|\bar{x}_k - GN^{+}(GN(\bar{x}_k, \bar{\theta}), \bar{\theta})\|^2$ has been measured for a set of 50 images. The resulting mean-square error was 1.3346, which is equal to an image quality of around 47 dB. It should thus be of little use to code a normalized image $\bar{y}$ to a quality higher than this. The expected mean-square error introduced by the eye matching normalization was also measured for 50 images, to a value of 0.7020, which is equal to a PSNR of around 50 dB.

The texture mapping technique used both in the geometrical normalization and in the eye matching normalization in this thesis is based on a simple nearest neighbor technique. This yields rather crude results and can rather easily be improved by, e.g., bilinear interpolation. The MSE would then probably decrease significantly but the principle would not change — the expected error would still be larger than zero.

# Chapter 3

# A Block-Based Algorithm

In this chapter, we will argue that if high quality levels are to be supported, the full frame basis vector strategy is too high in complexity to be practical.

This complexity problem is addressed by a block-based algorithm where each block is described with its own set of basis vectors. The basis vectors are obtained through a separate PCA in each block. This technique has previously been used for face recognition by Pentland et al. [52]. It was first described for face texture representation by the author in 1997 [63] and later by Jebara et al. [36], who also used it for extracting depth information. The eigenspaces found by the different PCAs are called *modular eigenspaces*.

The use of modular eigenspaces to represent the image results in a decrease in storage demands for the database and a simultaneous increase in image quality. The technique proposed in this chapter is compared to JPEG, showing an improvement of 8 dB over this coder.

## 3.1 Complexity

The usefulness of any image compression system is dependent upon the capability to provide a range of different quality levels. The average quality of the 100 images coded with geometrical normalization in the diagram in Figure 2.7 was 30.5 dB. Admittedly, the scope of a coder unable to compress images to higher quality levels than this is rather limited. The upper bound on the quality imposed by the geometrical normalization (see Section 2.7) is at about 47 dB, so there is headroom for another 18 dB of quality enhancement.

One way to enhance the quality is to increase the size of the training set, $N$, while holding the size of the database, $M$, constant. The basis vectors $(\hat{\varphi}_k)_1^M$ will then become closer to the optimal $(\bar{\varphi}_k)_1^M$, with an increase in expected PSNR as a result. By increasing $N$ it is possible to augment the quality without

penalizing either the rate or the complexity of the encoder/decoder. (The time needed to calculate the database is of course affected — it grows proportionately to $N^2 n$, but this operation only has to be performed once and is therefore not considered.) As long as there is a significant quality difference in the coding results of images from inside and outside the training set, quality can be raised by increasing $N$, but eventually a point will be reached when increasing $N$ will not affect the $M$ basis vectors in the database much, and thus only result in a negligible increase in quality.

Another way to augment the quality is to increase the size of the database, $M$. This way, more degrees of freedom are given to the image representation and a better fit to the original image is possible. However, complexity problems arise when $M$ is increased. For instance, a system with $M = 5000$ coefficients will (for $[512 \times 512]$ pixel images) require over a Gigabyte of memory only to store the basis vectors. To code (or decode) a single image requires $1.2 \cdot 10^9$ floating point operations and, arguably worse, the same number of memory transfers. Even if such a workload may not be impractical with future technology, the complexity is still disproportionately huge compared to other parts of the model-based coding scheme. Moreover, the size of the database, $M$, cannot be larger than the size of the training set, $N$, which in practice is a problem since training data is currently obtained manually and is thus scarce.

## 3.2 Block-Tiling of Eigenspace

Roughly speaking, the expected image quality is determined by the number of coefficients used to describe the image. The time to compute these coefficients is directly proportional to the size of the support[1] of the corresponding basis vectors. Thus, reducing the support of the basis vectors will lower complexity for a certain quality level. This can be done by tiling the image to encode $\bar{y}$ into disjoint blocks $(\bar{y}^j)_{j=1}^B$ as seen in Figure 3.1. Each block $\bar{y}^j$ can then be seen as an $n_b$-dimensional vector, where $n_b$ equals the number of pixels inside the block (e.g., $32 \times 32 = 1024$ pixels as in the example in Figure 3.1). In analogy with Equation 2.1 in the frame based case, each vector $\bar{y}^j$ representing a block can be approximated by a linear combination of $M$ basis vectors,

$$\bar{y}^j \approx \hat{y}^j = \sum_{i=1}^{M} \alpha_i^j \bar{\varphi}_i^j. \tag{3.1}$$

An important note to make here is that the basis vectors $\bar{\varphi}_i^j$ depend on the block position $j$, i.e., a different set of basis vectors is used for the coding of each block. Hence, the block marked with a rectangle in the left eye-region in Figure 3.1 will be described using a different set of basis vectors than the block from the mouth-region marked with a dashed rectangle.

[1] In this context, the support of a vector is the set of all elements in the vector that can be non-zero.

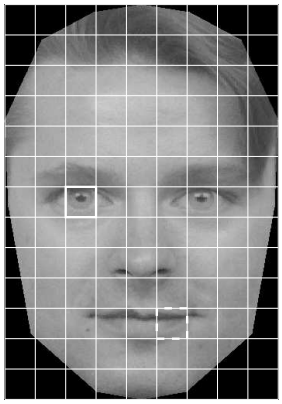


Figure 3.1: *All the images in the training set are tiled into blocks, and a separate* PCA *is performed at each block-position. Thus, a special set of basis vectors is obtained for the eye block marked with a rectangle, and a different set is obtained for the mouth block marked with a dashed rectangle.*

The basis vectors $\bar{\varphi}_i^j$ can be obtained by performing principal component analysis on blocks at this particular position in the images of the training set. As in the full frame case, the finite size of the training set will mean that the calculated basis vectors will only be an approximation of the true Karhunen-Loève basis vectors. They will therefore be denoted by $\hat{\varphi}_i^j$.

The idea of performing a principal component analysis on only a part of the image was introduced by Pentland et al. [52] for face recognition purposes. In its current form, it was first described for coding purposes by the author [63].

The technique has several advantages compared to the full frame case. Firstly, as already mentioned, the complexity is lowered, which will be investigated in more detail shortly. Secondly, since the blocks are disjoint, the training set used to find the basis vectors for one block can be reused for the next block. This can be very important in practice where training data is often scarce. Thirdly, as pointed out by Jebara et al. [36], the modular eigenspace can be a superset of the single eigenspace. This is illustrated in Figure 3.2: Here the training set consists of two faces, one with open eyes and mustache, and another with closed eyes and no mustache. It is clear that it is impossible to construct a face with closed eyes and mustache as a linear combination of these two faces. With a modular eigenspace, on the other hand, this is possible.

Let $f_f$ be the number of floating point operations[2] needed to calculate the $KL$ coefficients $\{\alpha_k\}_1^M$ in the full frame case. Each coefficient $\alpha_k$ is obtained by calculating the inner product between the original image $\bar{y}$ and the corresponding

[2]One floating point operation is here defined as either a multiplication or an addition.

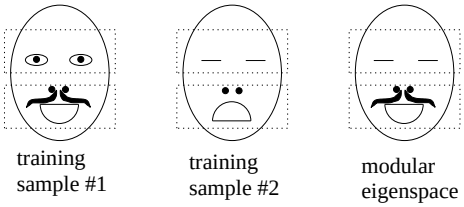



Figure 3.2: *Using a block-based approach, it is possible to create combinations of textures that are not present in the training set.*

basis vector $\hat{\varphi}_k$, $\alpha_k = \langle \bar{y} \,|\, \hat{\varphi}_k \rangle$. For each pixel in the inner product, one multiplication and one addition is performed. If $w$ and $h$ are the width and height of the image, then $f_f = 2whM$, since $M$ such inner products need to be formed. Analogously, let $m_f$ denote storage complexity for the full frame case. Since $M$ basis vectors of size $wh$ need to be stored, this results in $m_f = whM$. Now the same sort of analysis is performed in the block-based case. Each block $\bar{y}^j$ is to be described by a linear combination of $M$ basis vectors, $\bar{y}^j = \sum_{k=1}^{M} \alpha_k^j \hat{\varphi}_k^j$, where $\hat{\varphi}_k^j$ is a $[w_b \times h_b]$ sized block. Each coefficient $\alpha_k^j$ is obtained by computing the inner product $\left\langle \bar{y}^j \,|\, \hat{\varphi}_k^j \right\rangle$ which involves $2w_bh_b$ floating point operations. The total number of floating point operations is thus $f_b = 2w_bh_bBM$, where $B$ is the number of blocks in the image. In the same manner, the storage complexity for the block-based case is $m_b = w_bh_bBM$. Since the tiling is non-overlapping, the number of pixels is independent of the tiling, and $wh$ thus equals $w_bh_bB$. Hence, for a certain $M$, $f_f = f_b$ and $m_f = m_b$. However, as will be demonstrated shortly, for a certain quality level, the block-based scheme will need a smaller $M$ in order to reach the same quality level. For this reason, complexity is reduced.

## 3.3 Quantization

After the Karhunen-Loève transformation has been performed on the block, $M$ coefficients are obtained. The standard deviation $\sigma_{\alpha_i^j}$ for each coefficient is estimated off-line using the square root of the corresponding eigenvalue. Each coefficient $\alpha_i^j$ is then divided by this estimate in order to normalize it. It is then quantized with a Lloyd-Max quantizer that is optimized for a Gaussian source. Each block gets the same number of bits to distribute among the coefficients of the block. However, inside a block bits are distributed non-uniformly according to Equation 2.2. The image is decoded using the C-like pseudocode in Figure 3.3. Here, `nextbitsINT(a,b)` is a function that fetches an (unsigned) integer of `b` bits from the bit stream `a`.

Figure 3.4 shows an example of an image coded with block-tiling. The original image to the left (not included in the training set) is geometrically normalized and coded with the block-tile approach using $N = 200$, $M = 24$. Note that even

```
b = bit_budget / nbr_of_blocks;
a = get_allocation_vector(b);
for( each block j )
{
  for( each pixel k in reconstructed block )
     block[j,k] = meanblock[j,k];

  for( i=1 to M )
  {
     quant_tab = quanttables[a[i]];
     c = quant_tab[nextbitsINT(stream,a[i])];
     c = c * sqrt( eigenvalue[j,i]);
     for( each pixel k in the block )
        block[j,k] = block[j,k] + c * eigenimage[j,k,i];
  }
}
```

Figure 3.3: *The decoding algorithm (pseudocode).*

Figure 3.4: *The original image (left) is geometrically normalized, coded block-wise with M=24 coefficients per block, quantized with about 61 bits per block, and inverse geometrically normalized resulting in the right image.* PSNR *= 36 dB, 165 blocks of* $[32 \times 32]$ *pixels, total size = 11585 bits including information for geometrical normalization.*

though the quality has improved from 28 dB in the frame based case (Figure 2.9) to 36 dB, the size of the database which determines complexity has lowered from $M = 77$ (19 Mbytes) to $M = 24$ (6 Mbytes).

## 3.4 JPEG Comparison

The image quality is now in a range where comparison to JPEG [73] is worthwhile. To avoid edge effects that would hamper the JPEG method unfairly, a rectangular image is cut out, as depicted in the first column of Figure 3.5. The dimensions of the images are chosen as multiples of 16 to facilitate JPEG compression of both the original and the subsampled version of the images. The images are first geometrically normalized, which makes them slightly bigger. The normalized images are then divided into $[32 \times 32]$ pixel blocks and each

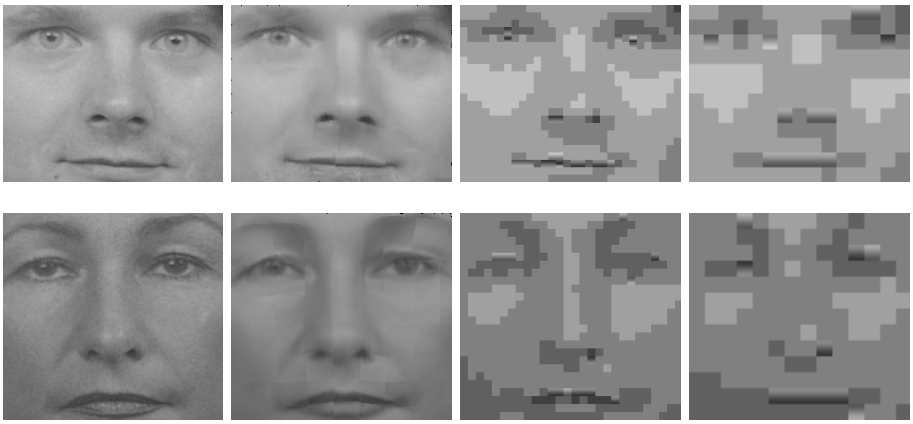


Figure 3.5: *First column: original images (not included in training set). Second column: block-tiling with block size* $[32 \times 32]$ *pixels,* PSNR *= 33.6 dB and 33.3 dB (top and bottom, respectively), rate = 0.07 bpp. Third column:* JPEG *encoded,* PSNR *= 27.0 dB and 26.6 dB, respectively, rate = 0.18 bpp. Fourth column: subsampled* JPEG*:* PSNR *= 25.4 dB and 25.3 dB, rate = 0.12 bpp*

block is then Karhunen-Loève transformed. The coefficients are coded as described above using an overall bit rate of around 3000 bits per image (or 0.07 bits per pixel (bpp)), including the 1360[3] bits to perform the inverse geometrical normalization. This corresponds to the second column in Figure 3.5, with PSNR around 33 dB. The block is also coded with JPEG (third column), resulting in PSNR = 27 dB, rate = 0.18 bpp (8000 bits). Note that this is more than two times the number of bits needed by the proposed algorithm, and the quality is 6 dB worse. Since the JPEG coder could not reach lower rates than this, the images were subsampled, JPEG coded and upsampled. This resulted in the right-most image, PSNR = 25 dB, rate = 0.12 bpp (4500 bits), about 8 dB worse than the block-tile algorithm.

## 3.5 Size and Shape of the Regions

By tiling the images into regions, two new parameters have been introduced, namely the size and the shape of the regions. The diagram in Figure 3.6 shows the quality as a function of rate for the same image[4] compressed with different region sizes; $[8 \times 8]$ pixel blocks (triangles), $[32 \times 32]$ pixel blocks (squares), and the full frame size (circles).

[3]Since the image does not cover the entire face, nine points of the mask in Figure 2.4 are not needed in the inverse geometrical normalization. With 18 10-bit integers fewer, the result is 1360 bits.

[4]The same experiment was conducted for a small set of images, and the result was similar in all cases.

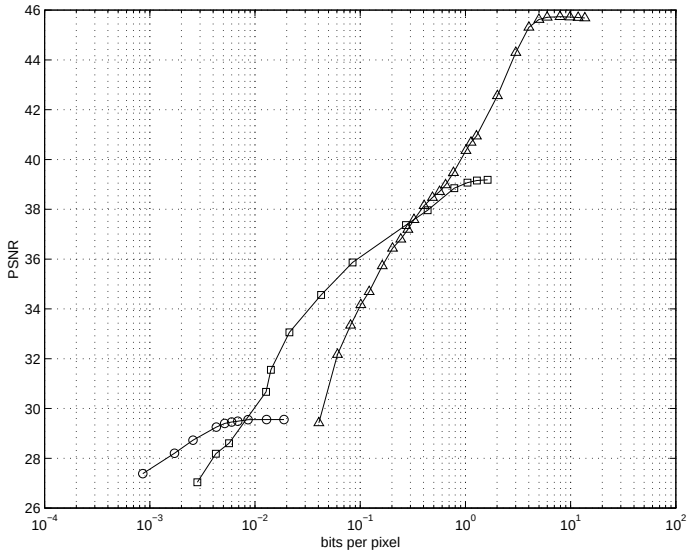



Figure 3.6: *Circles: full frame. Squares:* $32 \times 32$ *pixel block. Triangles:* $8 \times 8$ *pixel block.*

It is interesting to note that the full frame coder is outperformed by the $[32 \times 32]$ pixel-block-coder at bit rates above 0.01 bpp. This might at first seem peculiar: The full frame case is equivalent to the block-based case but without restrictions on the support of the basis vectors. Given a large enough training set, the full frame PCA will find the best set of linear basis vectors. In other words, if the block-based scheme were optimal, the basis vectors in the full frame case would converge towards rectangular support. Since the full frame method in this sense is more general, it ought to outperform the block-based scheme at all bit rates. One possible explanation why it is not so is that for large region sizes, such as the full frame case, the size of the training set, $N$, is too small in relation to the dimensionality of the region, $n_b$. This means that the $\hat{\varphi}_k$s will be a bad estimate of the $\bar{\varphi}_k$s, particularly for large regions.

If the region size becomes too small, on the other hand, the penalty for not being able to exploit pixel correlation across block boundaries will be high. Thus we have identified two problems coupled to the size of the regions:

1. Too large a block size $\Rightarrow$ $\hat{\varphi}_k$ bad estimate of $\bar{\varphi}_k$ if $N$ too small.

2. Too small a block size $\Rightarrow$ The inability to exploit across-block correlation hampers performance seriously.

Different parts of the face texture may have different characteristics, and may thus be more or less influenced by the two problems stated above. For instance, the smooth cheek areas with high inter-pixel correlation might suffer more from

(2) than from (1) and should thus be tiled into larger regions than, e.g., the eye area. This could lead to the conclusion to use the CANDIDE triangles (depicted in Figure 2.4) as regions. However, some of the CANDIDE triangles are small even though they cover smooth areas of the face, and others are probably larger than they should. In addition, not only the sizes of the regions but the placing of them is more related to the 3D structure of the face. Recently, a more appealing approach was presented by Jebara et al. [36], where an iterative algorithm finds the shape and the relative size of the regions by minimizing the MSE over a set of face images different from the ones in the training set.

In this work, square-shaped regions of equal size were chosen, primarily because it makes the coder easy to implement. Since inter-pixel correlation decreases with distance in normal images, a more compact region-shape such as the hexagon would be a possible improvement. An even more obvious improvement would be to make use of the strong correlation of the left and right halves of the face by letting each region consist of two sub-regions, one on each side of the symmetry axis. Figure 3.7 illustrates this. Here the region A in the image

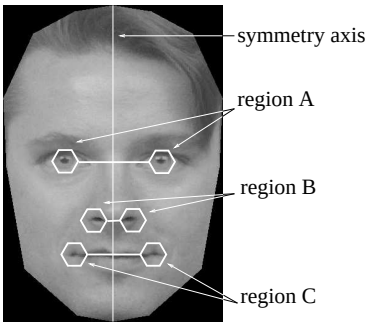



Figure 3.7: *The symmetry axis of the face suggests that the regions should be non-connective and made up from two sub-regions from each side of the axis.*

is made up from the two subregions of each eye.

# Chapter 4

# Global Bit Allocation

In the block-based algorithm described in Chapter 3, each block is handled independently from the others. More specifically, the same number of bits is allocated to all blocks, regardless of the image contents. Most transform based image coding algorithms make the quantization dependent of the image content in order to allocate bits to parts of the image that are hard to code. Some wavelet algorithms [57, 55], use zerotrees (or similar structures) to localize large coefficients, corresponding to areas that contain a lot of structure. Another example is the JPEG algorithm which run length codes the coefficients; blocks of simple structure will contain a lot of zero coefficients and use up few bits. However, in both of the above-mentioned methods, all parts of the image are treated the same way, e.g., the zerotree algorithm does not favor the left hand side of the image over the right hand side. Similarly the JPEG method treats each block the same way, regardless of its position within the image. This approach makes sense when compressing general-purpose images, since any pattern may occur anywhere in the image. However, this is not true for the class of geometrically normalized face images where, e.g., the nose is always in the center of the image.

To clarify the difference, consider for a moment the collection of all images accessible on the Internet, $I^{web}[i,j]$. The expected luminance level in each pixel, $E\{I^{web}[i,j]\}$, will probably not depend much on $[i,j]$. Likewise, the correlation between pixels in the image will probably not depend much on where it is measured; two neighboring pixels in the center of the image will correlate approximately as strongly as two neighboring pixels near the corner of the image will. Accordingly, the JPEG algorithm, which is widely used for Internet images today, is built to treat each block in the image the same way, without considering from where in the image the block emanates.

For the source of geometrically normalized face images $\bar{y}^F[i,j]$, on the other hand, none of the above is true. The expected pixel intensity, for instance, will depend heavily on where in the image it is measured. This can be seen in the image in Figure 4.1, which is the average of 100 geometrically normalized

face images and thus an approximation of $E\{\bar{y}^F[i,j]\}$. This image is far from

Figure 4.1: *This image is the average of 100 geometrically normalized images.*

having the same graylevel everywhere; instead the intensity varies significantly over the image area. Similarly it is probable that the correlation between pixels in the smooth regions of the face (such as the cheek region) is higher than in other regions. Since each region, due to the normalization, will always be in a prespecified part of the image, it is possible to know à priori which parts of the image will be easy to code. Some regions will, on average, be well described using just a few bits, whereas others will need more bits in order to achieve the same quality.

## 4.1 Global Allocation of Bits

This section proposes to allocate the bits globally, by collecting the eigenvalues corresponding to the basis vectors from all blocks, and performing the bit allocation mentioned in Section 2.6 on all coefficients simultaneously. Note that this operation does not depend on the particular image, but rather on statistics common to all face images, namely the eigenvalues from the PCA. These are prestored in the decoder and thus no extra bits need to be transmitted in order to resolve the bit allocation for a certain bit rate. The table in Figure 4.2 shows the distribution after global bit allocation. Note that some blocks, such as the one marked with a box, are assigned several times the number of bits than some of the other blocks.

To evaluate how much is gained in PSNR by this uneven distribution of the bits, the following experiment has been conducted: A second set of images, called the *evaluation set*, disjoint from the training set, is used. Each image in the evaluation set is coded, first with the bit distribution from Figure 4.2, and then with an even distribution, but with the same total number of bits. The

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| – | – | 12 | 68 | 97 | 114 | 99 | 72 | 15 | – | – |
| – | 41 | 104 | 119 | 120 | 117 | 120 | 116 | 107 | 51 | – |
| 08 | 96 | 95 | 95 | 93 | 85 | 92 | 97 | 94 | 103 | 12 |
| 40 | 92 | 78 | 68 | 66 | 67 | 66 | 69 | 74 | 88 | 51 |
| 70 | 71 | 60 | 52 | 52 | 49 | 52 | 49 | 58 | 67 | 77 |
| 77 | 54 | 46 | 50 | 43 | 46 | 46 | 48 | 46 | 52 | 73 |
| 77 | 69 | 79 | 73 | 59 | 47 | 56 | 71 | 79 | 73 | 73 |
| 68 | 79 | 166 | 206 | 92 | 55 | 85 | 192 | 179 | 84 | 70 |
| 45 | 65 | 118 | 113 | 82 | 43 | 77 | 113 | 118 | 70 | 51 |
| 25 | 28 | 36 | 39 | 43 | 42 | 40 | 39 | 37 | 28 | 25 |
| 14 | 22 | 27 | 46 | 99 | 86 | 100 | 50 | 27 | 24 | 16 |
| 11 | 27 | 38 | 42 | 52 | 65 | 54 | 41 | 44 | 28 | 12 |
| 05 | 33 | 50 | 100 | 101 | 114 | 105 | 104 | 56 | 33 | 09 |
| – | 26 | 41 | 42 | 59 | 68 | 60 | 42 | 43 | 26 | 04 |
| – | 4 | 19 | 36 | 41 | 48 | 44 | 36 | 22 | 05 | – |

Figure 4.2: *Left: the number of bits per block with global assignment. An average of 61 bits per block is used. Right: the left table visualized; the darker the area, the more bits per block.*

result is presented in Figure 4.3. In all of the 33 images, the PSNR count is higher with the uneven bit distribution, and the average gain is 0.5 dB.

The image to the right in Figure 4.2 shows the same data as the table, using dark shades of gray for high bit concentrations, and light shades for low. Note that the optimization procedure is concentrating the bits to the visually important features. Thus the perceived quality gain should be visible despite the modest PSNR-gain. This is confirmed in Figure 4.4. It is evident that it is suboptimal to spend the same number of bits on edge blocks, which contain few foreground pixels, as on interior blocks. Therefore, the edge blocks were removed from the image in Figure 4.4 prior to the coding, so that a fair comparison of the two methods could be conducted. The original image (left) is compressed with a uniform bit distribution (middle) and with the bit distribution from Figure 4.2 (right). The uneven coefficient distribution improves the quality about 0.42 dB. Note the substantial difference of image quality in the eye region. A blow-up of the right eye can be seen in the lower row of Figure 4.4.

## 4.2 Possible Improvements

This chapter has concluded that, on average, it is wise to assign more bits to certain regions at the expense of others. However, for a particular image, the best bit allocation may be different. Thus, an improved bit allocation algorithm can include an image dependent component in addition to the image class dependent component presented in this chapter.

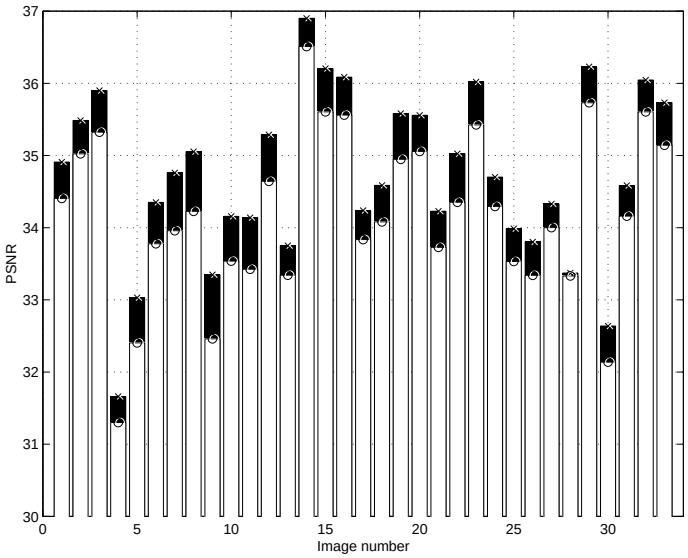



Figure 4.3: *The diagram shows the benefit of allocating bits globally. The white bars with circles show the* PSNR *when the same number (61) of bits is used in each block. The black bars with crosses show the* PSNR *for the same images when the bits are globally allocated.*

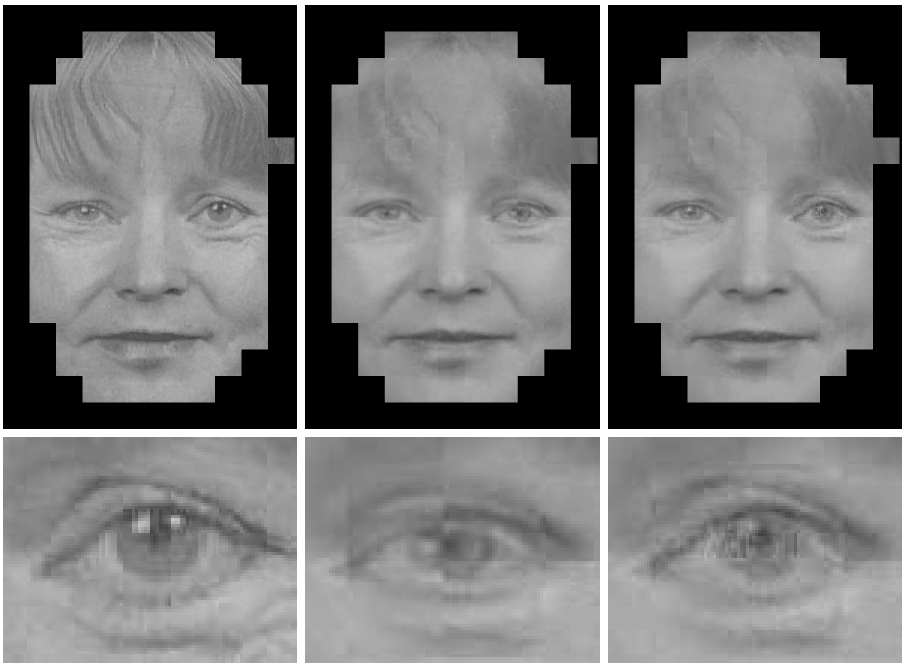


Figure 4.4: *Top row: the original image (left) is compressed with a uniform bit distribution (middle). Using the uneven distribution from Figure 4.2 improves the quality 0.42 dB (right) for the same rate. Bottom row: a blow-up of the right eye region.*

# Chapter 5

# Refined Normalization

In Chapter 2 the feature points needed for the geometrical normalization are extracted manually. The exact placing of the feature points is thus rather ad hoc. This chapter investigates how much can be gained if the feature point positions are refined. This refinement problem can be formulated as a minimization problem, where the mean-squared error (MSE) is minimized over possible feature point positions, leading to an increase in quality of around 1.6 dB. The chapter is concluded with a speculation about the possibilities of using the method as a step in an automatic feature extractor.

## 5.1 Errors in Manually Extracted Feature Point Data

On scrutinizing the training set after geometrical normalization, it becomes evident that the (manual) extraction of the feature points can be improved. For instance, the eye-sizes should be the same for all images in the training set after geometrical normalization has been carried out; still, as shown in Figure 5.1, they vary considerably. Due to erroneous feature point data, the geometrical normalization step fails in this case to compensate for different eye-sizes. Worse still, it can introduce differences that are not present in the original data. Figure 5.2 shows a face image before (left) and after (right) geometrical normalization. Before the normalization the eyes are roughly of the same height, whereas after the normalization, the right eye is clearly more vertically elongated than the left. This means that there is room for improvement of the feature point data.

Figure 5.1: *Two images from the geometrically normalized training set. Note that the left eye of the left person is smaller than the left eye of right person. Geometrical normalization is supposed to remove such differences between images, so this erroneous result is due to bad feature point data.*

Figure 5.2: *With bad feature point data the geometrical normalization can worsen the alignment: The left face image (before the geometrical normalization) has roughly the same height of the eyes, whereas in the right image (after), the person's left eye is clearly more vertically elongated than the right.*

## 5.2 Feature Point Refinement as an Optimization Problem

A possible means of refining the feature point positions is to vary the feature points around their current positions and choose the position that gives the lowest MSE between the original and compressed face image. Let $\bar{\alpha} = \text{prj}(\bar{y})$ be the function that projects a (normalized) face image $\bar{y}$ on the basis vectors of the database, and let $\hat{y} = \text{rec}(\bar{\alpha})$ be the function that reconstructs the image $\hat{y}$ from the coefficients $\bar{\alpha}$. Analogous with Chapter 2, $\bar{y} = GN(\bar{x}, \bar{\theta})$ and $\hat{x} = GN^{+}(\hat{y}, \bar{\theta})$, respectively, will denote the geometrical normalization and its inverse[1]. It is then possible to define a function $\varepsilon = f(\bar{x}, \bar{\theta}) = \|\bar{x} - \hat{x}\|^2 =$

[1]The geometrical normalization is not reversible (see Section 2.7). The function $GN^{+}$ is still called inverse geometrical normalization, even though this is somewhat incorrect.

$\|\bar{x} - GN^{+}(\text{rec}(\text{prj}(GN(\bar{x},\bar{\theta}))),\bar{\theta})\|^2$. The function is illustrated in Figure 5.3. The problem of refining the feature point positions $\bar{\theta}$ can thus be seen as an

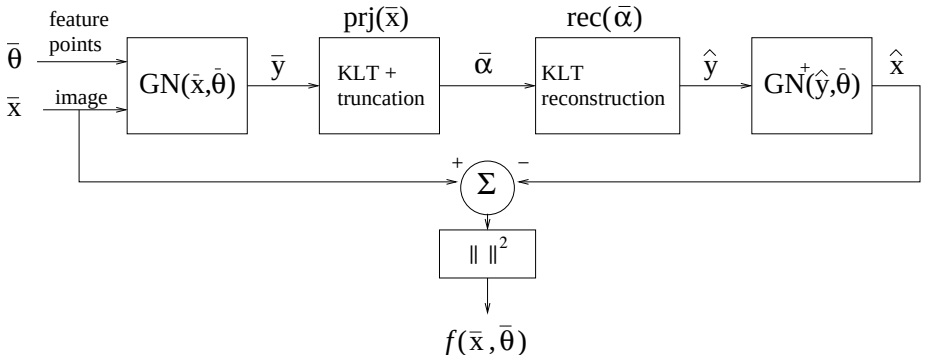



Figure 5.3: *The function to be optimized.*

optimization problem to find the $\bar{\theta}$ that minimizes the error $\varepsilon$;

$$\min_{\bar{\theta}} f(\bar{x},\bar{\theta}). \tag{5.1}$$

Such an approach was used by Jebara et al. [36] but for finding the global motion of a rigid 3D mask rather than adapting a flexible wireframe to a face.

For simplicity, a simple coordinate search algorithm is selected: each parameter $\theta_k$ is examined separately. First a direction $d$ is chosen by evaluating $f_0 = f(\bar{x},\bar{\theta})$ and $f_1 = f(\bar{x},\bar{\theta}+\bar{\delta}_k)$, where $\bar{\delta}_k = [00\ldots010\ldots0]^T$ with a one in the $k$th position. If $f_1 < f_0$, a positive direction $d = 1$ is chosen, otherwise a negative direction $d = -1$ is chosen. The $k$th element of $\bar{\theta}$ is now incremented (or decremented) in pixel-wise steps and the function $f_i = f(\bar{x},\bar{\theta}+d\bar{\delta}_k i)$ is evaluated in each step. The process halts as soon as the function values start to increase. The parameter $\theta_k$ is then updated to the best value, and the process continues by examining the next parameter, $\theta_{k+1}$.

When moving the feature points, it is important to preserve the topology of the wire frame mesh. In Figure 5.4 the point $M$ is moved to the right. The

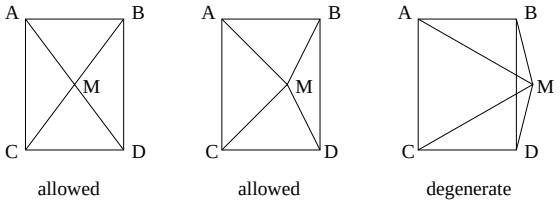



Figure 5.4: *Moving a point $M$ can alter the topology of the mesh. The two first meshes are of the same topology, but the rightmost one is of a different topology.*

topology is not changed for small changes of $M$, when the point is still left of the line $\overline{BD}$ (middle image), but for large translations of $M$ (rightmost image), the mesh is no longer of the same topology. Thus Equation 5.1 must be completed with a condition that the topology should remain unchanged. This can be done by simply rejecting changes in $\bar{\theta}$ that make triangles face the wrong way. Such a test would reject the rightmost wire frame in Figure 5.4, since the triangle $\triangle BMD$ is showing the wrong side to the viewer.

## 5.3 Results

The diagram in Figure 5.5 shows, for six images, the increase in PSNR due to the optimization procedure. These images were not part of the training set that

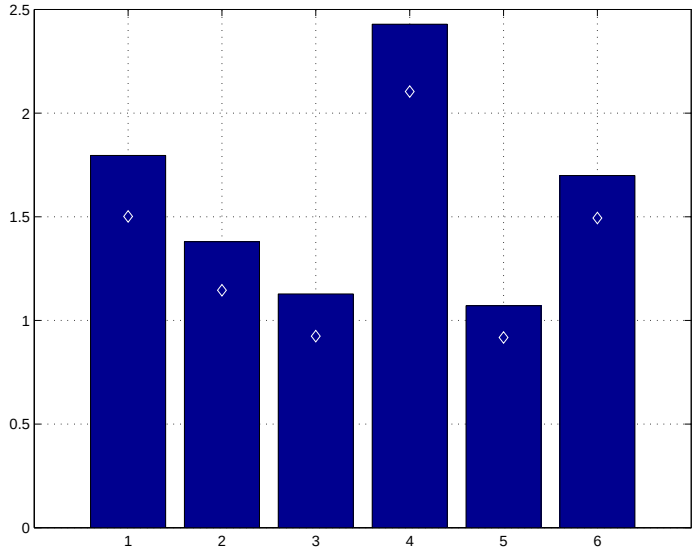



Figure 5.5: *The gain in* PSNR *due to optimized placing of the feature points. Database size $M$ used in* prj() *and* rec() *equals 50. The diamonds indicate the quality improvement after the first pass of the algorithm.*

was used to calculate the database used in prj() and rec(). The full frame basis vectors were used in prj() and rec() as opposed to the block based versions. The increase in PSNR varies between 1.1 dB and 2.4 dB, and the average increase is 1.6 dB. In Figure 5.6, three of these six images are shown before and after the optimization. The first column contains the original images, the second contains the images coded using the original feature point data, and the last column contains the images that are coded using the refined feature points. The images (from top to bottom) correspond to the first, third and the fourth image in the diagram in Figure 5.5. Hence the images in the third column have increased their PSNR by 1.8 dB, 1.1 dB and 2.4 dB, respectively, compared to the

Figure 5.6: *First column: the original images. Second column: the coded images without feature point refinement. Last column: the coded images after feature point refinement. The images (from top to bottom) correspond to the first, third and fourth bar in the diagram in Figure 5.5, respectively. The* PSNR *improvement figures are 1.8 dB, 1.1 dB and 2.4 dB, respectively. In the two top rows, note the improvement in the eyebrow region. Also, note that in the top and bottom rows, the nose and the mouth regions are better represented. In the third row the hairline has also improved significantly.*

images in the second column. Note the following improvements in image fidelity which are typical of this type of optimization: In the female faces of the two top rows, the thin eyebrows are much better preserved. Two-thirds of the images in the training set are men, who usually have thicker eyebrows. Thanks to the geometrical normalization it is possible to construct a pair of thin eyebrows from a thicker pair. This is tricky to do manually, but is performed elegantly by the optimization step. This is also true for the mouth region, which is seen in the top and bottom row of Figure 5.6. The mouths in the refined versions are more accurately shaped. Moreover, the nose region was improved in all of the six images tested. The refined feature points make a better job of putting the nostrils of the normalized images in the right place. Hence, they are better aligned with the nostrils in the database and become darker, which improves the contrast in the nose region. This is especially apparent in the bottom row of Figure 5.6, where not only the nostrils are darker, but the shape of the nose is also more accurate. In the same way the irises and the pupils are more distinct in the normalized images. Finally, in images with a well-defined hairline, like the bottom image in Figure 5.6, the hairline is more accurately modeled, the hair is darker and the border between the hair and the forehead is more distinct.

## 5.4 Complexity

In order to optimize the wireframes for the images in Figure 5.6, the object function $f$ was evaluated around 800 times (around 540 in the first pass and 260 in the second pass). Since each evaluation of the function involves the projection of the original image on 50 eigenimages, the computation time is rather heavy; 22 seconds per function evaluation, i.e., about 5 hours for the entire optimization. This is acceptable for off-line purposes, such as refining the normalization for the images in the training set, but cannot, at least with the computing power of today, be used for real-time applications.

One way to reduce the complexity is to reduce the number of images in the database used by prj() and rec(). The resulting parameter vector $\bar{\theta}$ can then be used to compress the image using a larger database. Unfortunately, this does not work very well. The idea was tried out using a database of five eigenimages in prj() and rec(), but 50 eigenimages in the final version comparison. As can be seen in the diagram in Figure 5.7, the gain in PSNR is not as high as in Figure 5.5 where 50 eigenimages were used in prj() and rec(); it varies between 0.2 dB and 1.4 dB and averages around 0.7 dB, only half of the 1.6 dB achieved with $M = 50$ in prj() and rec(). One might consider this rather good — 0.7 is still better than nothing — until taking a look at the images depicted in Figure 5.8. The images here suffer from "edge artifacts" (sharp edges that appear in the face) and the skin has a somewhat plastic appearance (this might not be visible in the printed copy). These two artifacts each have a possible explanation: In Figure 5.9, the wireframe for the first image in Figure 5.6 is shown unoptimized (left), optimized with $M = 50$ in prj() and rec() (middle)

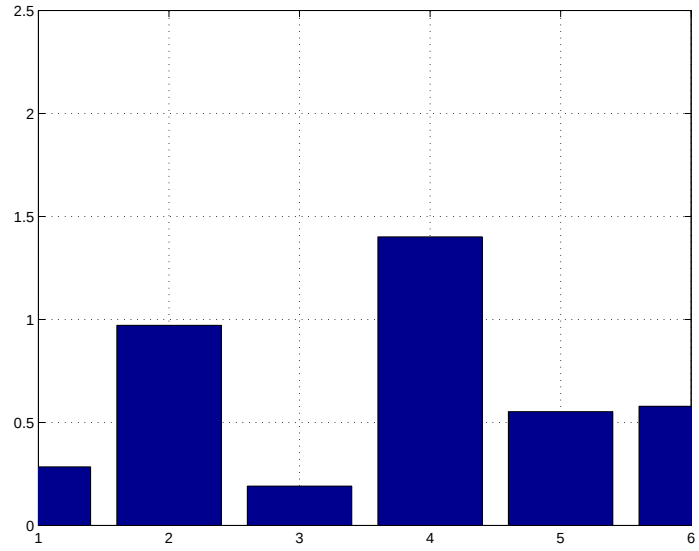



Figure 5.7: *The gain in* PSNR *due to optimized placing of the feature points. Database size $M$ used in* prj() *and* rec() *equals 5, database size for evaluation equals 50.*

and finally optimized with $M = 5$ (right). The rightmost wireframe in this figure differs from the two others in that it contains many triangles of almost zero area. This means that two triangles that are separated by such a triangle in the normalized image appear next to each other in the inversely normalized image, producing discontinuities in the intensity. This explains the "edge artifacts" which, e.g., can be seen on the left side of the nose in the middle image in Figure 5.8. Worse still, in the right eye-region in the rightmost wireframe, the topology has changed (in analogy with Figure 5.4). This can create black spots in the decoded image, such as the black dots above the left corner of the mouth in the rightmost image in Figure 5.8. Moreover, by optimizing on only the first five eigenimages in the database, the result is that a lot of the energy in the image is transferred to the first five coefficients from the others, compared to the case where no optimization has taken place. More specifically, the six images used in the diagram in Figure 5.5 were compressed with a database size of $M = 5$ first with the original feature points, and later with the feature points that had been refined using $M = 5$ in prj() and rec(). The average gain from this refinement was measured as 2.3 dB. This means that, due to the refinement, some extra error-energy corresponding to 2.3 dB was reduced by the first five eigenimages, ending up in the five first coefficients. Now, by changing $M$ to 50, the gain compared to the non-optimized case is only 0.7 dB. Since the first five eigenimages are the same in both coders, the first five coefficients must again have improved their energy by 2.3 dB. This means that the remaining 45 coefficients have lost energy corresponding to 2.3 dB-0.7 dB = 1.6 dB. Thus, by optimizing over only the five first eigenimages, we have transferred energy

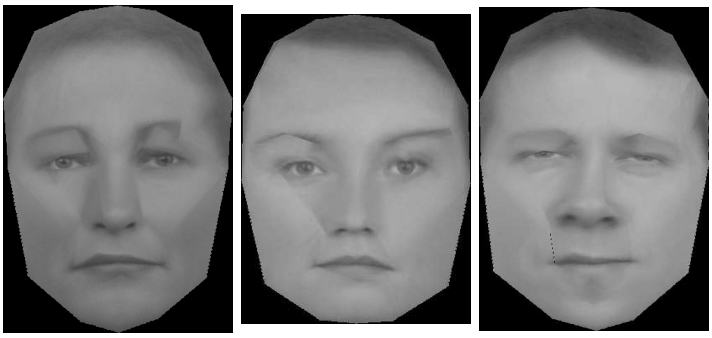


Figure 5.8: *Optimized with M=5 in* prj() *and* rec(). *Note how this causes edge artifacts (all three images), and topology errors (visible as black spots in the rightmost image).*

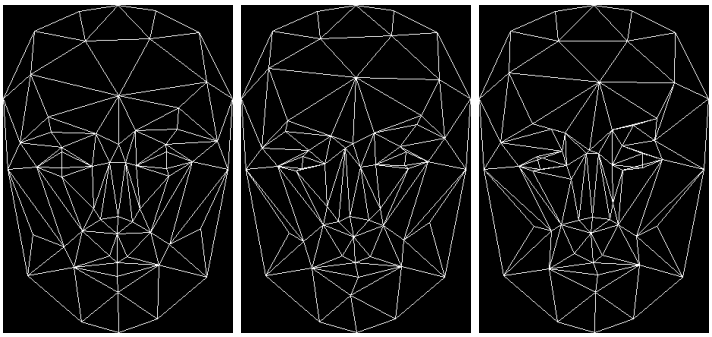


Figure 5.9: *The wireframe mesh for the first face image in Figure 5.8. Left: before refinement. Middle: refined using* $M = 50$ *in* prj() *and* rec(). *Right: ditto for* $M = 5$.

from the 45 last coefficients to the five first. Since basis vectors of lower order usually show low-pass behavior, the result should be an image with smoother, more plastic-looking skin.

## 5.5 Gradient Descent — a Possible Improvement

Instead of reducing the time to evaluate the object function, complexity can be decreased by reducing the number of times the function is evaluated. This can be done by estimating the gradient of the object function with respect to the vertex points.

Assume that $I(x, y)$ is the original image. Since the geometrical normalization is carried out by moving vertices of a triangle mesh, the mapping of the coordinates from the original image to the normalized image will be piecewise affine. Assume that $I'(x, y)$ is the normalized image, and $g$ the affine coordinate

transform

$$g_x(x,y) = a_{11}x + a_{12}y + a_{13}$$
$$g_y(x,y) = a_{21}x + a_{22}y + a_{23}, \tag{5.2}$$

so that $I'(x,y) = I(g_x, g_y)$. The coefficients $a_{ij}$ in $g$ can be determined from the position of the vertices $(v_x^l, v_y^l)_{l=1}^3$ of the triangle in the original image (for details, see [40]). Further assume that $\hat{I}'(x,y)$ is the approximated version of $I'(x,y)$: $\hat{I}'(x,y) = \sum_{k=1}^{p} \alpha_k \varphi_k(x_i, y_i)$, where $\alpha_k = \sum_{\forall i} I'(x_i, y_i)\varphi_k(x_i, y_i)$. By using the affine transform backwards, the original shape can be recovered: $\hat{I}(g_x, g_y) = \hat{I}'(x,y)$. $\hat{I}(x,y)$ is thus the approximation of $I(x,y)$.

When choosing object function to minimize, both $(I - \hat{I})^2$ and $(I' - \hat{I}')^2$ are possible. While the former is more correct, since it considers the original and approximated image as opposed to the original and approximated texture, the latter leads to simpler equations and is hence selected. The object function $f$ thus becomes

$$f = \sum_{i=1}^{p} \left[ I'(x_i, y_i) - \hat{I}'(x_i, y_i) \right]^2 = \sum_{i=1}^{p} \left[ I'(x_i, y_i) - \left( \sum_{k=1}^{p} \alpha_k \varphi_k(x_i, y_i) \right) \right]^2. \tag{5.3}$$

Let $(v_x^l, v_y^l)$ be the position of the $l$th vertex in the original image. $f$ can now be differentiated with respect to the vertex movements through

$$\frac{\partial f}{\partial v_x^l} = \sum_{i=1}^{p} 2 \left( I'(x_i, y_i) - \hat{I}'(x_i, y_i) \right) \left[ \frac{\partial I'(x_i, y_i)}{\partial v_x^l} - \sum_{k=1}^{p} \frac{\partial \alpha_k}{\partial v_x^l} \varphi_k(x_i, y_i) \right], \tag{5.4}$$

where

$$\frac{\partial I'(x_i, y_i)}{\partial v_x^l} = \frac{\partial I(g_x, g_y)}{\partial v_x^l} = \frac{\partial I}{\partial g_x}\frac{\partial g_x}{\partial v_x^l} + \frac{\partial I}{\partial g_y}\frac{\partial g_y}{\partial v_x^l} \tag{5.5}$$

and

$$\frac{\partial \alpha_k}{\partial v_x^l} = \frac{\partial}{\partial v_x^l} \left( \sum_{j=1}^{p} I'(x_j, y_j)\varphi_k(x_j, y_j) \right) = \sum_{j=1}^{p} \frac{\partial I'(x_j, y_j)}{\partial v_x^l} \varphi_k(x_j, y_j). \tag{5.6}$$

In Equation (5.5), $\frac{\partial I}{\partial g_x}$ and $\frac{\partial I}{\partial g_y}$ are simply the gradient images of the original image $I(x,y)$ in the $x-$ and $y-$ directions respectively. The affine function $g_x(x,y)$ depends on the coefficients $a_{ij}$, which in turn depend on $v_x^l$. $\frac{\partial g_x}{\partial v_x^l}$ is therefore easily obtained. $\frac{\partial f}{\partial v_y^l}$ is calculated similarly.

The calculation of $\frac{\partial f}{\partial v_x^l}$ in Equation 5.4 can be sped up by approximating $\frac{\partial \alpha_k}{\partial v_x^l}$ with 0. The scheme will then alternate between finding the best normalization for a specific set of $\alpha_k$ and finding the best $\alpha_k$ for a specific normalization.

The vertices in the triangle mesh can now be updated using gradient descent:

$$v_x^l = v_x^l - \delta \frac{\partial f}{\partial v_x^l}, \tag{5.7}$$

where $\delta$ is a suitable small constant. $v_y^l$ is updated in the same way. Note that each vertex may influence more than one triangle. In that case, the differential $\frac{\partial f}{\partial v_x^l}$ should be summed over all affected triangles.

## 5.6 Feature Extraction

Potentially, the feature point refinement technique presented in this chapter can be used for feature point extraction. Combined with an algorithm that can localize the position and boundary of the face, a wireframe with the feature points in their average position can be put on the face and used as an initialization to the feature point refinement procedure. No experiments have been made to see if this works. One experiment could be to investigate how much the vertices of the wireframe can deviate from the true feature point positions before the algorithm ceases to converge. If it proves to be robust, this could be the solution to the texture extraction problem. As seen in Figure 5.10, this could be the block providing the first texture and the first set of parameters in a model-based coder.

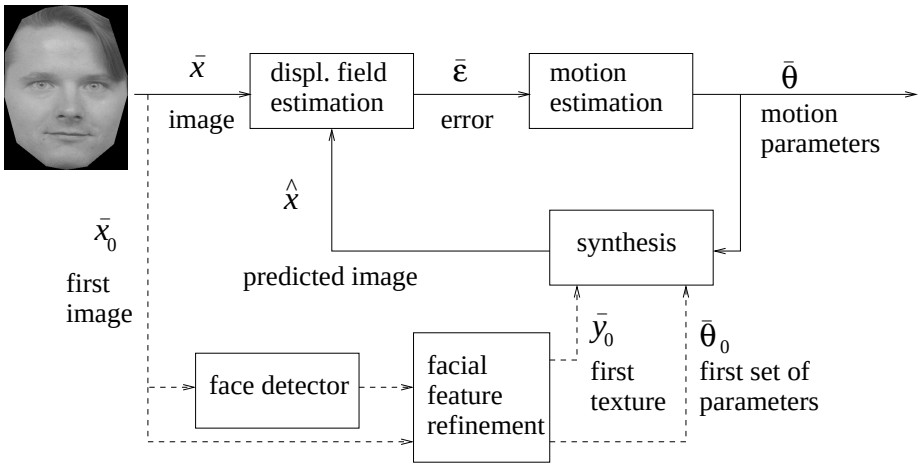



Figure 5.10: *Possible scheme for a model-based coder, which includes the normalization refinement block.*

# Part II

# Real-Time Model Based Head Tracking

# Chapter 6

# Related Work

This chapter will serve as an overview of the different head tracking systems that can be found in the literature. Only systems that track true three-dimensional motion (translation and rotation in three-dimensional space) will be considered. Notation has been changed to keep it consistent throughout the chapter. Some trackers resolve not only head pose but also local motion in the form of facial movements, as well as three-dimensional structure information. In this chapter however, only the head tracking part of the algorithms is considered. The chapter will end with a description of a proposed system that both operates in real time and is robust to large rotations out of the image plane.

## 6.1 Head Tracking Based on Optical Flow Methods

Several of the head tracking algorithms in this survey are based on optical flow, and therefore Appendix B provides a short introduction to the topic. Whenever possible, the notation from the appendix is used in this chapter.

### 6.1.1 Planar Parametric Optical Flow

In their 1995 paper [13], Black and Yacoob model the face as a plane in three-dimensional space. By doing so, the optical flow field from Equation (B.6) can be further simplified [2] to

$$\begin{aligned} u(x,y) &= a_0 + a_1 x + a_2 y + p_0 x^2 + p_1 xy \\ v(x,y) &= a_3 + a_4 x + a_5 y + p_0 xy + p_1 y^2. \end{aligned} \tag{6.1}$$

Equation (6.1) is valid for all pixels in the image that belong to the face, with the same coefficients $\Pi = \begin{bmatrix} a_0 & \dots & a_5 & p_0 & p_1 \end{bmatrix}^T$ for all these points. The tracker starts out with the assumption that the pixels that constitute the face

are given. This area in the first and second frame is now used to estimate the coefficients $\Pi$. By using

$$\mathbf{X}(\mathbf{x}) = \begin{bmatrix} 1 & x & y & 0 & 0 & 0 & x^2 & xy \\ 0 & 0 & 0 & 1 & x & y & xy & y^2 \end{bmatrix}, \tag{6.2}$$

Equation (6.1) can be written as $\begin{bmatrix} u & v \end{bmatrix}^T = \mathbf{X}(\mathbf{x})\Pi$, which, inserted in Equation (B.11), yields

$$\nabla I \mathbf{X}(\mathbf{x})\Pi = -I_t. \tag{6.3}$$

Thus one linear equation for $\Pi$ is obtained per pixel in the face region. A robust least squares regression is used to estimate the coefficients $\Pi$. Once these coefficients are obtained, Equation (6.1) is used to calculate the optical flow to work out where the pixels of the facial area have moved from the first image to the second. This is done repeatedly, and the face can thus be tracked. Since this system is differential, it is susceptible to drift, i.e., tracking errors will accumulate over time. From the coefficients in $\Pi$, information about the three-dimensional motion can be inferred. For instance, if $p_0$ is positive, this is interpreted as the head rotating leftwards around the neck. Black and Yacoob do not attempt to recover the real three-dimensional motion, $(\Omega_x, \Omega_y, \Omega_z, T_x, T_y, T_z)$, (cf. Equation (B.6)), since this is not needed for their application, which is facial expression recognition. About 2 minutes of processing is needed per frame on a Digital Alpha 3000 machine.

### 6.1.2 Elliptical Optical Flow Model

In order to cope with larger sized head motion, Basu et al. use an ellipsoid model instead of the planar model [11]. The first frame is used to adjust the sizes of the major axes of the ellipsoid. A number of three-dimensional points $P_0$ are then sampled from the surface of the ellipsoid, together with their normals $N_0$. The position and orientation of the ellipsoid are parameterized as $a = \begin{bmatrix} \alpha & \beta & \gamma & t_x & t_y & t_z \end{bmatrix}$ where $(\alpha, \beta, \gamma)$ represent the absolute Euler angles relative to the first frame, and $\mathbf{t} = (t_x, t_y, t_z)$ represents the translation, also relative to the first frame. The $k$th three-dimensional point $P_k$ and its normal $N_k$ can be calculated as

$$\begin{aligned} P_k &= T_R P_0 + \mathbf{t} \\ N_k &= T_R N_0, \end{aligned} \tag{6.4}$$

where $T_R$ is a rotation matrix obtained by multiplying the three Euler rotation matrices together. Projection to the image plane is performed using

$$\begin{aligned} x &= \frac{X}{1 - Z/Z_d} \\ y &= \frac{Y}{1 - Z/Z_d}, \end{aligned} \tag{6.5}$$

where $Z_d$, which is assumed to be known, is a constant that depends on the focal length. (This projection formula is different to that of Equation (B.5) due

to a different positioning of the origin.) If the pose is known in frame $n-1$, the image coordinates $(x'_k, y'_k)$ of point $k$ in frame $n-1$ can be calculated, by combining (6.4) and (6.5). The optical flow in this point then follows

$$\begin{aligned} u_k(a) &= x_k(a) - x'_k \\ v_k(a) &= y_k(a) - y'_k, \end{aligned} \tag{6.6}$$

where $(u_k, v_k)$ are functions of the parameters in $a$. A general optical flow algorithm is used to compute the measured flow $(u_{M,k}, v_{M,k})$, and the error between the two is measured;

$$\epsilon_k = (u_k(a) - u_{M,k})^2 + (v_k(a) - v_{M,k})^2. \tag{6.7}$$

In order to decrease the weight of statistical outliers, the error is truncated at $\epsilon_t$ using

$$e_k = \min(\epsilon_k, \epsilon_t). \tag{6.8}$$

The target function $E = \sum_k e_k$ is now minimized over $a$ using simplex search. When the error $E$ is below a certain threshold the procedure is stopped and the next frame is processed. Basu et al. compared their results to a planar model and found that the ellipsoid model gave better results [11]. Due to the simplex search the method is relatively slow, about 30 seconds per frame.

To get a model that fits the head better than what an ellipsoid can do, Zhang and Kambhamettu use an extended superquadric model [76]. They also use a motion segmentation algorithm to provide robustness against partial occlusion. A post-regularization method based on edge flows is also employed to reduce error accumulation.

Harville et al. use depth measurements from a stereo rig to extend the brightness constraint with a depth constraint [30]. This increases robustness for large rotations out of the image plane and for translation in depth, but carries the cost of stereo input.

### 6.1.3 Feed-Back Optical Flow

In his Licentiate Thesis from 1990 [53], Roivainen uses the three-dimensional head model CANDIDE [54] to parameterize the optical flow. Equations B.4 and B.5 are combined to $\begin{bmatrix} x_2 & y_2 \end{bmatrix}^T = F(P_1, U)$, where $U$ is the state vector containing both rotation and translation as well as local motion (face gestures). Taylor expanding around $U$ and neglecting higher order terms, yields

$$\begin{bmatrix} u \\ v \end{bmatrix} = \frac{\partial F}{\partial u_j} \Delta U, \tag{6.9}$$

which is equivalent to Equation (B.8) for $C = \frac{\partial F}{\partial u_j}$ except for the local motion parameters. Roivainen estimates $\frac{\partial F}{\partial u_j}$ numerically. The optical flow $(u_M, v_M)$

is measured, and for each point $k$ in the face, $\begin{bmatrix} u_{Mk} & v_{Mk} \end{bmatrix}^T = C_k \Delta U$ should hold. Gathering 200 such points, the matrix equation

$$V = C\Delta U \tag{6.10}$$

is formed, where $V = \begin{bmatrix} u_{M1} & v_{M1} & u_{M2} & v_{M2} & \dots \end{bmatrix}^T$ and $C = \begin{bmatrix} C_1^T & C_2^T & \dots \end{bmatrix}^T$. $\Delta U$ is then calculated using least squares

$$\Delta U = (C^T C)^{-1} C^T V. \tag{6.11}$$

The point contributing the most to the error is removed and the least squares solution is recalculated. This is done six times in an attempt to remove the worst statistical outliers.

Roivainen also investigates how to update the state parameter $U$, once the differential state $\Delta U$ has been estimated. The first attempt, here denoted *feed-forward* motion estimation, is depicted in Figure 6.1. The two last images $n$ and

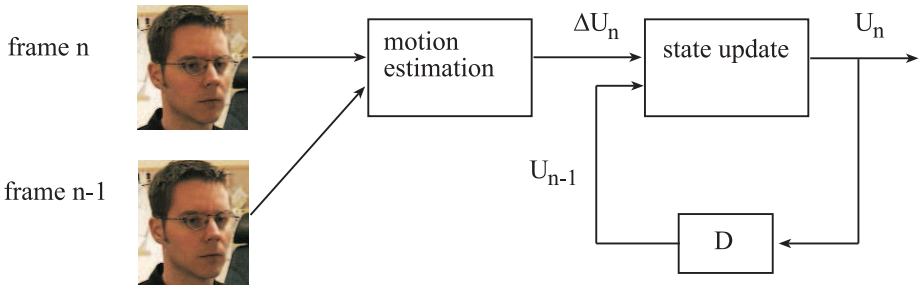



Figure 6.1: *Feed Forward Motion Estimation*

$n-1$ are processed and the differential motion $\Delta U_n$ between these is estimated. This $\Delta U_n$ together with the previous pose $U_{n-1}$ are used to estimate the current pose $U_n$. Since there is no feedback from the pose estimation $U_{n-1}$ to the motion estimation, errors made in the motion estimation stage will accumulate without any chance of recovery. Both the tracker of Black and Yacoob [13] and that of Basu et al. [11] are updated in a similar differential fashion, and will thus be susceptible to drift.

In a *feed-back motion estimation* system, the motion estimation is performed between the image $n$ and a rendered version of the head. This is shown in Figure 6.2. If there is a small error in the estimation of $\Delta U_{n-1}$, this will mean that the pose estimate $U_{n-1}$ will be wrong. This error will influence the rendered image, and can be corrected for by $\Delta U_n$. The feed-back tracker in Figure 6.2 can thus track without error accumulation.

Li et al. [43] extend the work of Roivainen in three important aspects. Firstly, a modified version of Equation (B.9) including local motion is used, instead of

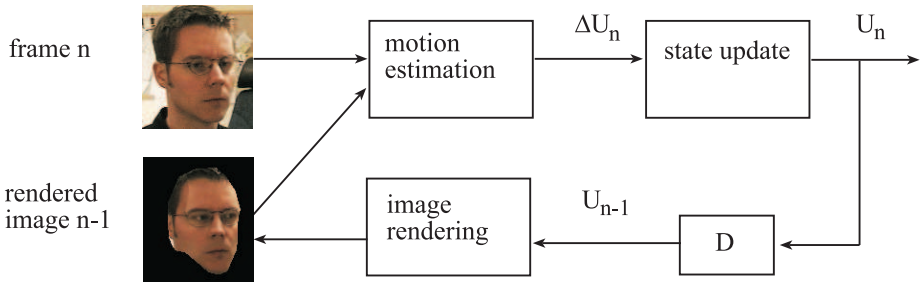



Figure 6.2: *Feed-back motion estimation*

numerically estimating the matrix $C_k$ for each point $k$. Secondly, instead of estimating the optical flow $V$ and using Equation (6.11) to resolve $\Delta U$, the optical flow constraint of Equation (B.12) is used in combination with Equation (B.7):

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} c^u \\ c^v \end{bmatrix} \Delta U = -I_t \tag{6.12}$$

or

$$G\Delta U = -I_t, \tag{6.13}$$

where $G = \begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} c^u \\ c^v \end{bmatrix}$. The motion $\Delta U$ can now be calculated using $\Delta U = -(G^T G)^{-1} G^T I_t$. Avoiding to estimate the optical flow is an advantage, since it is not always possible to do this reliably. The price for this is that the optical flow constraint is only valid for small motions. The third modification by Li et al. is therefore to predict the motion $\Delta\hat{U}_n$ from the previous movements $\Delta U_{n-1}$, $\Delta U_{n-2}$, etc. By linearizing Equation (B.10) around $U_{n-1}+\Delta\hat{U}_n$ instead of around $U_{n-1}$, large motions can be estimated. This changes the constraint from small motions (small $\Delta U$s), to a constraint on the size of the prediction error $\Delta U_n - \Delta\hat{U}_n$.

In a later paper, Li and Forchheimer modify their algorithm to include M-estimation, which is a robust version of the least squares algorithm [42].

### 6.1.4 Deformable Models and Optical Flow

DeCarlo and Metaxas use a very detailed head model constructed from anthropological data [20]. The parameters (around 80) are divided into *basic shape parameters* $q_b$ that are characteristic for each head and do not change over time, and *motion parameters* $q_m$ that drive both global and local motion. The parameters are changed using a deformable model framework, where edges in the image give rise to forces on the parameters according to

$$\dot{q} = f_q, \tag{6.14}$$

where the dot denotes derivative with respect to time. In addition to this, the motion parameters are influenced by the optical flow constraint from Equation (6.13), which in the notation of DeCarlos and Metaxas is written

$$G\dot{q}_m + I_t = 0 \tag{6.15}$$

and can be solved using

$$\dot{q}_m = -G^+ I_t, \tag{6.16}$$

where $G^+$ is the pseudo-inverse $G^+ = (G^T G)^{-1} G^T$. Instead of using Equation (6.16), this equation is used as a hard constraint on Equation (6.14), yielding

$$\dot{q}_m = -G^+ I_t + (I - G^+ G) f_{q_m}. \tag{6.17}$$

The matrix $(I - G^+ G)$ in (6.17) changes $q_m$ according to the part of $f_{q_m}$ that is consistent with the optical flow. The edge force component of (6.17) is designed to combat accumulation error in what is otherwise a differential scheme prone to drift. The system handles large rotations well and runs at about 2.5 seconds per frame on a 175 MHz R10000 SGI O2.

## 6.2 Active Appearance Based Techniques

In their 1998 paper [17], Cootes et al. present a way to analyze images of faces. The starting point for their technique is a representation where both the texture and the shape of an image are modeled using eigenspaces, as shown in Figure 6.3. Similar representations have been used for face recognition [18] and

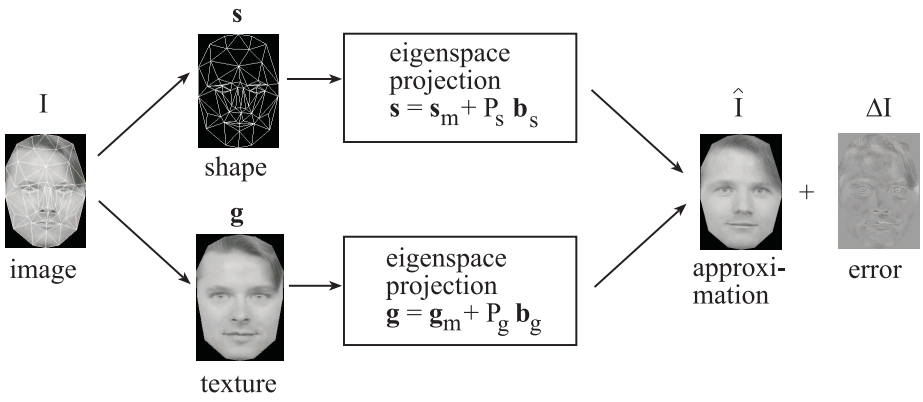



Figure 6.3: *The starting point for an Active Appearance Model of a face — both the shape and the texture are modeled using eigenspaces.*

image compression (Part I of this thesis and [63, 64]). A mesh is fitted to the face image $I$ and the texture is geometrically normalized. The shape $\mathbf{s}$ and the texture $\mathbf{g}$ are projected onto eigenspaces made of similar shapes and textures, and are compressed to a small number of parameters $\mathbf{b}_s$ and $\mathbf{b}_g$. By combining

these two parameter sets and performing a new PCA, a new parameter vector **c** containing *appearance* parameters is created. The appearance parameters contain information about both shape and texture. From this parameter set it is possible to reproduce an approximation $\hat{I}$ of the image, with an error $\Delta I$.

The basic idea of the analysis is to deduce from the error image $\Delta I$ how to change the appearance parameters **c** in order to get a better fit of the model. This is illustrated in Figure 6.4. Here, the shape is misaligned in the vertical

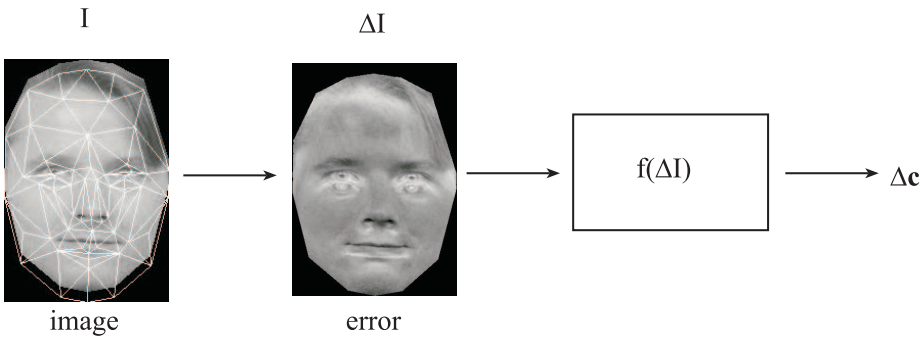



Figure 6.4: *Analysis step of an Active Appearance Model: The slightly mismatched mesh in the image I will generate an error $\Delta I$ between the original image and the model. A function $\Delta \mathbf{c} = f(\Delta I)$ of this difference image tells how to move the mesh to get a better fit.*

direction, which means that the error image $\Delta I$ will contain shadow images of two faces superimposed. For instance, the error image will contain a bright eye shadow below the eye region. A function $\Delta \mathbf{c} = f(\Delta I)$ is found that maps the error image to a suitable correction of the appearance parameter **c**. Cootes et al. use a linear function that can be obtained by setting the Taylor expansion of the error function to zero.

By starting with $\mathbf{c} = 0$ (which gives the average face), **c** can be corrected in the above-described fashion until it converges. It will find the correct shape and texture of the face given that the starting position is not too far from the true solution. Cootes et al. have used their algorithm on image sequences, where the appearance from the previous frame is used as a starting value for **c**.

Even though Active Appearance Models are two-dimensional, they can cope with a certain amount of out-of-plane rotation by deforming the shape. By switching between several two-dimensional models, a wide span of orientations can be covered.

Cascia et al. use a three-dimensional version of the active appearance method [16]. Instead of modeling the shape using a two-dimensional PCA, a cylinder model of the head is used. The shape is parameterized as $b_s = (\alpha, \beta, \gamma, t_x, t_y, t_z)$

where $(\alpha, \beta, \gamma)$ are the absolute Euler angles and $(t_x, t_y, t_z)$ is the three-dimensional translation of the cylinder. The texture modeling is also modified; the mean image $g_m$ of the eigenspace in Figure 6.3 is replaced with the first image of the head in the sequence, which is assumed to be frontal. Given that the shape is correct, any errors should be due to illumination changes, modeled by the eigenspace $P_g$. Cascia et al. reports real time performance (15 Hz) on a SGI O2 machine.

Ahlberg [4] uses the three-dimensional Candide model [54] for face tracking using AAMs. The shape parameter here includes both global parameters (rotation and translation) as well as local deformation. The work is in progress with the goal of achieving real-time performance.

## 6.3 Extended Kalman Filter Based Work

Azarbayejani and Pentland reformulate the Structure from Motion (SfM) problem to a recursive parameter estimation problem that can be solved using an extended Kalman filter [8]. This algorithm will be explained in more detail in Chapter 7. One of their applications is head tracking, where a small number of feature points on the head are tracked using two-dimensional correlation over small patches, and the resulting trajectories are used to compute the three-dimensional pose of the head. Simultaneously, the three-dimensional positions of the tracked feature points are extracted.

Jebara and Pentland [35] extend the work in [8] in several ways: Six feature points are automatically extracted in the face; the eyes, the sides of the nose, and the corners of the mouth. Small patches are extracted and used as templates, and these patches are then tracked using a technique from Hager and Belhumeur [27] that allows for changes in scale and in-plane rotation. The extra two degrees of freedom, as compared to just allowing translation, means that two feature points can be used for each template, i.e., twelve points altogether. The three-dimensional structure that is obtained from the filter is regularized by projecting it on to an eigenspace of depth maps taken from Cyberware[1] scans. The structure is then projected to two dimensions and used as a starting point for the tracking in the next frame. This means that if an individual patch is lost in the tracking, there is a good chance of finding again since the tracker will be given a reasonable starting point. The correlation error for the individual point is fed back to the Kalman filter. Thus the filter will trust bad measurements less. The algorithm runs at 30 Hz on an SGI O2 200 MHz machine.

Matsumoto and Zelinsky have presented a Kalman filter based face tracking system for stereo input [46]. The structure from motion problem is then greatly simplified, but at the cost of having to add a second camera.

[1] Cyberware, Inc. is a commercial company that designs, manufactures, and sells standard and custom 3D scanning systems and software.

## 6.4 Proposed System

Among the different methods mentioned in this overview, only the Adaptive Feedback Tracker (AFT) from Jebara and Pentland achieves full frame rate, i.e., 30 Hz, even though the system by Cascia et al. comes close with 15 Hz. Another appealing feature of the AFT system is that it uses structure from motion to build the model that helps the tracking. Although more advanced adaptation of the shape to the image sequence is done by the system from DeCarlo and Metaxas, their system is far from real-time performance. Another advantage with the AFT tracker is that it estimates also the focal length of the camera. All other trackers assume this camera- and zoom- depending parameter to be known.

Many of the trackers described above use a three-dimensional surface model for the face, such as a plane [13], an ellipsoid [11] or a triangular face model [53, 43, 20, 16, 4]. The model estimated by Jebara and Pentland however, is a point configuration without surface. This is an advantage in that it can be estimated easily, but it also means that the system cannot predict self-occlusion. When the head turns left for instance, the left eye will disappear behind the edge of the head, but the AFT system will keep looking for the eye. The patch tracking method from Hager and Belhumeur allows for in-plane rotation and scaling of the patch, but cannot model the affine and projective transformations that occur when the patch is viewed at a steep angle. Moreover, there is no mechanism in the AFT system for adding new points to track when the old ones are occluded. The inability to handle new points, self occlusion and projective transformation of the patches means that the tracker will not work when there are large rotations out of the image plane.

In this thesis, the tracker by Jebara and Pentland is extended with a three-dimensional model of the head. This has several advantages.

- By rendering the model in the estimated pose, the patches can be obtained from the rendered model. Thus the patches will be transformed projectively.
- Instead of tracking rotation and scale for each individual patch, all the patches will be transformed at once, reducing the cost of adding another point. Thus a dense set of feature points can be tracked, which is essentially equivalent to the optical flow in the most information bearing points.
- The three-dimensional model can be used to predict self-occlusion and thus determine when a feature point measurement ceases to be reliable.
- The normal of the model can be used to predict the angle at which a patch is seen. A patch viewed at a very steep angel is more prone to mismatches than one that is head on, its impact on the solution should therefore be diminished by the tracking algorithm.

- The model can be used to predict the position of the side of the head, which is useful when adding new points.
- By incrementally updating the texture, a full three-dimensional textured head model is recovered. This fits nicely into the model-based coding framework. Moreover, by warping the incoming video back to frontal position, an eigen-representation on the parts of the image can be used. This enables low-bitrate updates of the texture of, e.g., the mouth and the eyes.

For developing purposes, it is also useful to be able to compare the incoming video with the rendered model in order to assess whether the motion of the head is correct and looks natural.

The proposed tracker is presented in Figure 6.5. The three-dimensional sur-

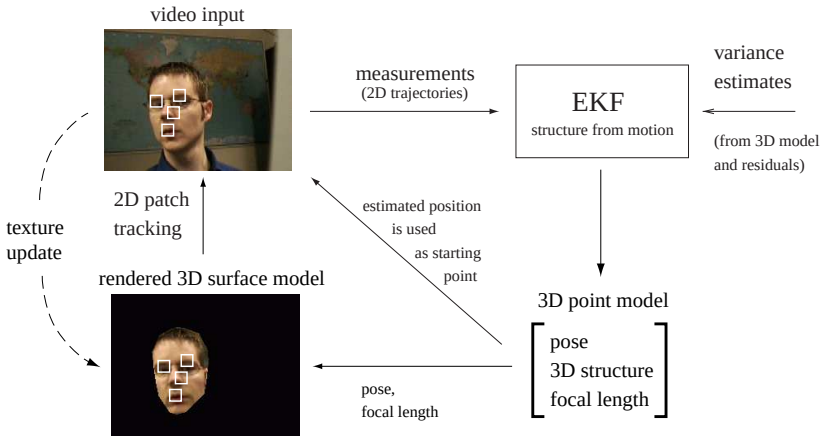



Figure 6.5: *Patches from the rendered image (lower left corner) are matched with the incoming video. The 2D feature point trajectories are fed through the SfM extended Kalman filter that estimates the pose information needed to render the next model view. For clarity, only four patches are shown.*

face model (bottom left) is rendered in the predicted pose. Patches from this rendered image are matched against the incoming video (top left). The two-dimensional measurements are fed into an extended Kalman filter that calculates the three-dimensional structure, pose and focal length for a point configuration defined by the center of each patch. The pose and the focal length are then used to render the surface model, and the structure is used to predict the positions of the patches' centers in the next frame. Whereas the solid arrows represent information flow that occurs every frame, the dashed arrows are only invoked at texture update. When the head has turned sufficiently, texture is grabbed, and both the three-dimensional surface model (bottom left) and the extended Kalman filter (upper right) are updated.

As seen here, the structure from motion (SfM) Kalman filter plays a vital part in the loop above. The next two chapters will therefore describe the SfM problem in some more detail.

# Chapter 7

# SfM for Tracking

Knowing the three-dimensional structure of the tracked feature points can greatly help tracking. By constraining the two-dimensional positions of the tracked features to valid projections of the three-dimensional structure, robustness can be gained. By solving the *structure from motion* (SfM) problem, this three-dimensional structure can be obtained from the two-dimensional trajectories of the feature points. The SfM algorithm used in this thesis is the one from Azarbayejani and Pentland [8] which is based on an extended Kalman filter (EKF). The reasons for choosing this algorithm is mainly that it is recursive and that it provides an intuitive tool for the three-dimensional structure constraint. The most widely used SfM algorithms today are not based on extended Kalman filtering but on multilinear constraints. Therefore, this chapter will derive the simplest of the multilinear constraints, and briefly go through how it can be used for SfM. This will enable a comparison between the EKF- and the multilinear-based methods in the next chapter.

The chapter starts with a motivation for using SfM in tracking, and continues with a description of EKF based SfM. The reader is assumed to know about Kalman filtering and its non-linear extension, but Appendix C is provided as a service to those who want to refresh their knowledge. The chapter will be concluded with a derivation of how the SfM problem can be solved using multilinear constraints.

## 7.1 Using Structure to Help Tracking

To illustrate how SfM can be useful for tracking, a naïve approach to track a single feature point will first be examined. A small patch around the feature point is cropped out and matched with the incoming video image. For computational efficiency, the patch is not matched against the entire image; instead the search is restricted to a small *search area*. The search area is centered around the starting position, which in this first approach is set to be the tracking result

of the previous frame. As illustrated in Figure 7.1, the measured position is fed to the memory element (the block marked with $D$) and used in the next frame

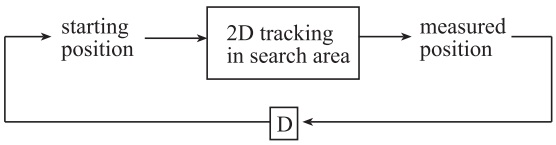



Figure 7.1: *A simple tracking loop — the measured position is fed to the memory element $D$ and used as the starting position for the matching in the next frame.*

as a starting position for the matching.

Such a tracking system has serious stability problems. There are several reasons why the two-dimensional matching might not result in the correct position within the search area: The point can be occluded, and the illumination or the orientation can have changed between the template and the image so that the correct position is no longer the best match. This means that the starting position in the next frame might be wrong, and the search area centered around it might no longer contain the true position. When this happens, the tracker will move around randomly and will not find the feature point again other than by pure chance — the tracking has failed.

One way to get around this problem is to track several feature points that belong to the same rigid object, and attach a confidence value to each match. For a feature point with a low confidence value, rather than using the flawed measurement, the two-dimensional position will be estimated from the known three-dimensional structure and from the other feature points. This is shown in Figure 7.2, where the cube represents the known three-dimensional structure,

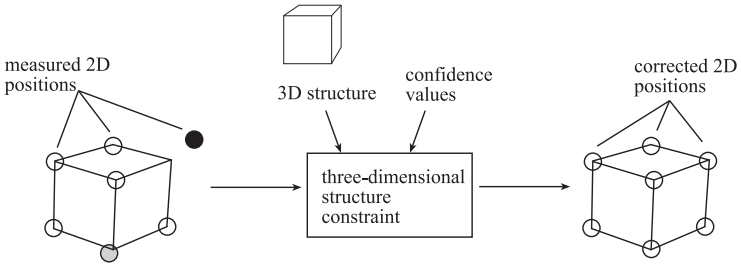



Figure 7.2: *The tracked two-dimensional featue points with associated confidence values (dark circle indicates low confidence value) are fused to a better estimate using the three-dimensional structure constraint.*

and the gray level of a feature point represents its confidence value. The block labeled “three-dimensional structure constraint” fuses the information from the three-dimensional structure, the measurements and their corresponding confidence values to corrected estimates of the two-dimensional positions of the

feature points.

Figure 7.3 shows how this can be used to create a more stable tracking sys-

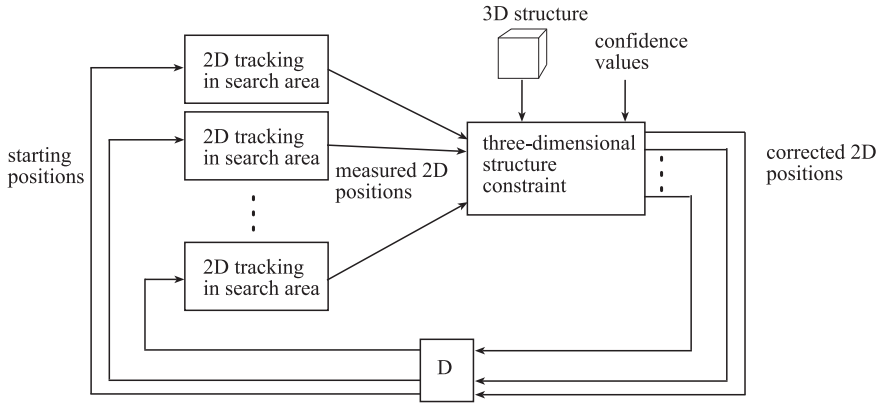



Figure 7.3: *A more stable tracking loop — the three-dimensional structure constraint and the confidence values are used to obtain reasonable two-dimensional starting positions for the tracking in the next frame.*

tem: The noisy two-dimensional measurements are constrained using the three-dimensional structure and the confidence values. The corrected two-dimensional estimates are then used as starting points for the two-dimensional trackers. Note that in this scheme, the mismatch of a single feature point is not fatal since the measurement can be corrected and the starting position in the next frame hence can be made reasonable.

### 7.1.1 Structure from Motion

To impose a structure constraint, the three-dimensional structure is needed. Whereas many systems use generic head models [13, 11, 53, 43, 42, 16, 4], DeCarlo and Metaxas [20] and Jebara and Pentland [35] estimate this structure from the image data.

The general problem of obtaining three-dimensional structure from two-dimensional images involves difficult problems such as segmentation, illumination modeling and correspondence. However, the problem is simplified if it is restricted to the estimation of rigid three-dimensional structure from the two-dimensional motion of feature points — a purely geometrical problem. This is usually referred to as the *structure from motion problem* and has been carefully studied [44, 22, 34, 29]. More formally, the SfM problem can be defined as

Definition 3 *Given the approximate two-dimensional location of a number of*

*features in a number of images of a rigid object, calculate their three-dimensional coordinates, the relative motion of the camera and its internal calibration parameters.*

The error in the two-dimensional location of the feature points can both be small (measurement noise) or large (anomaly errors due to mismatched points), and the algorithm should preferably be robust to both types of error. Note that the definition treats the motion of the camera, as opposed to the motion of the object in the scene. If the object moves rigidly, these two types of motions are equivalent. The attentive reader might oppose that, due to illumination changes, rotating the object gives a different result than rotating the camera around it. In this geometric formulation however, only the positions of the features are considered, and illumination effects are thus ignored. The common practice in the literature is to regard the object as static and ascribe the motion to the camera. This convention is adopted also in this thesis, even though it may seem a bit counterintuitive in the case of head tracking.

In summary, the three-dimensional structure of the feature points can help the tracking. Obtaining this structure is the SfM problem, which is treated in the rest of this chapter.

## 7.2 SfM using EKF

An extended Kalman filter (EKF) is a tool to estimate the dynamic changes of a state vector $\bar{\mathbf{x}}$. The vector $\bar{\mathbf{x}}$ itself cannot be measured; instead a measurement vector $\bar{\mathbf{z}}$, which is a function of $\bar{\mathbf{x}}$, can be observed. The dynamics of the system from time step $k$ to $k+1$ is described as (see Appendix C)

$$\begin{cases} \bar{\mathbf{x}}_{k+1} = f_k(\bar{\mathbf{x}}_k) + \bar{\mathbf{w}}_k, & \bar{\mathbf{w}}_k \sim N(\bar{\mathbf{0}}, Q_k), \\ \bar{\mathbf{z}}_k = h_k(\bar{\mathbf{x}}_k) + \bar{\mathbf{v}}_k, & \bar{\mathbf{v}}_k \sim N(\bar{\mathbf{0}}, R_k), \end{cases} \tag{7.1}$$

where the dynamics function $f_k(\cdot)$, the measurement function $h_k(\cdot)$ and the noise statistics $Q_k$ and $R_k$ are assumed to be known. To use this tool to estimate SfM, the state vector $\bar{\mathbf{x}}$ is identified as the unknowns; the three-dimensional structure and the extrinsic and intrinsic camera parameters relative to the first frame. The measurement vector $\bar{\mathbf{z}}$ will consist of the two-dimensional image coordinate measurements of the feature points. The function $f_k(\bar{\mathbf{x}})$ can be used to create a motion model for the object, e.g., constant velocity prediction. The function $h_k(\bar{\mathbf{x}})$ will contain a projection from the three-dimensional coordinates to the two-dimensional image coordinates.

The camera model and the representation of the three-dimensional points are very important in a recursive SfM algorithm. By using a parameterization that is different from the one commonly used in the literature, a dramatic increase in filter stability is gained. The parameterization and the SfM algorithm in this chapter is the work of Azarbayejani and Pentland [8]. It is repeated here since

it is needed in order to understand the next two chapters dealing with planar surfaces and the addition of new points.

### 7.2.1 Camera Model

The most common way to select the camera coordinate system is to place the origin in the center of projection (COP), so that

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} X_C \\ Y_C \end{pmatrix} \frac{f}{Z_C}, \tag{7.2}$$

where $(u, v)$ are the image coordinates and $(X_C, Y_C, Z_C)$ are the camera coordinates of the point. Instead, Azarbayejani and Pentland choose to place the origin in the image plane using

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} X_C \\ Y_C \end{pmatrix} \frac{1}{1 + Z_C\beta}, \tag{7.3}$$

where $\beta = 1/f$. This is depicted in Figure 7.4. By choosing an image centered coordinate system, a change in focal length has a direct effect on the image

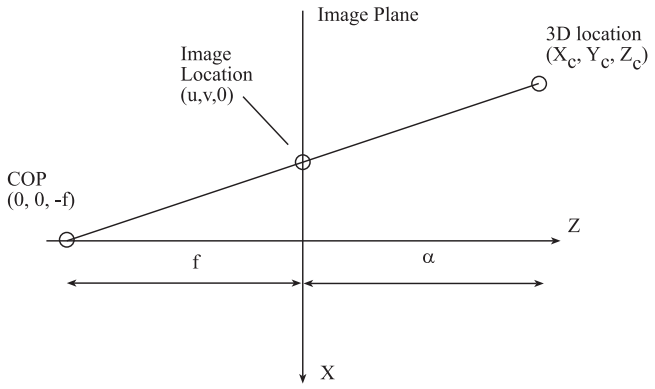



Figure 7.4: *The camera model of Azarbayejani and Pentland.*

geometry, i.e., the relative distances between the projected points in the image plane. In contrast, a change of $f$ in Equation (7.2) will only change the scale of the image coordinates, and the image geometry will in effect be coded into the depths $Z_C$. Using $\beta$ instead of $f$ makes it possible to cover both projective and orthogonal cases with the same algorithm, and avoids ill-conditioning when $f$ is large. Note that this camera model assumes that a number of parameters are known (calibrated), such as camera skew, aspect ratio, and the position of the principal point.

### 7.2.2 Structure Representation

The position of a feature point is represented by its image coordinates in the first frame $(u^0, v^0)$, and its unknown depth $\alpha$, as shown in Figure 7.4. The point

is restricted to lie on the line that emanates from the center of projection and goes through the point $(u^0, v^0)$ in the image plane. If the depth $\alpha$ is known, the cartesian coordinates in the first frame $(X, Y, Z)$ can be computed from $u^0$, $v^0$ and $\alpha$ using

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} u^0 \\ v^0 \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} \beta u^0 \\ \beta v^0 \\ 1 \end{pmatrix}. \tag{7.4}$$

Note that the three-dimensional location thus can be parameterized by the single parameter $\alpha$, since the image coordinates $(u^0, v^0)$ in the first frame are known. This has important implications for the stability of the Kalman filter. Using the standard parameterization $(X, Y, Z)$ yields $3N + 7$ degrees of freedom in the state vector $\bar{\mathbf{x}}$; three for each point, three respectively for translation and rotation and one for focal length. The measurement vector, however, will only contain $2N$ elements (the $x-$ and $y-$ image coordinates for each point), and the Kalman filter will thus be underconstrained. This is the case in the work of Broida et al. [15] where a good starting estimate of the structure is needed for convergence. However, the parameterization of Equation (7.4) with a single parameter per point will result in $N+7$ degrees of freedom compared to $2N$ for the measurement vector. Hence, if $N > 7$, the problem is overconstrained and thus uniquely solvable in each time step. This makes the algorithm useful as a solution to the SfM problem and not merely a way to adjust estimates already obtained from another SfM algorithm such as in [15].

Since structure can only be recovered up to scale, the depth $\alpha$ of the first point is fixed at 1. This is conveniently done by setting the corresponding variance in the Kalman filter to zero.

### 7.2.3 Translation and Rotation

Translation between the first and the current camera frame is parameterized as $(t_X, t_Y, t_Z)$. In the orthogonal projection case, $t_Z$ is not observable. Therefore, $t_Z\beta$ is estimated instead. This is advantageous also in the projective case for long focal lengths, since the sensitivity of the image coordinates for $t_Z\beta$

$$\frac{\partial u}{\partial t_Z \beta} = \frac{-X_C}{(1 + Z_C\beta)^2} \tag{7.5}$$

does not go to zero when $f \to \infty$ $(\beta \to 0)$ as it does for $t_Z$:

$$\frac{\partial u}{\partial t_Z} = \frac{-X_C\beta}{(1 + Z_C\beta)^2}. \tag{7.6}$$

The rotation from the first to the current frame is represented by a unit quaternion $\mathbf{q} = (q_1, q_2, q_3, q_4)$. The rotation matrix can be obtained from the quaternion by

$$R(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_0q_3) & 2(q_1q_3 + q_0q_2) \\ 2(q_1q_2 + q_0q_3) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_0q_1) \\ 2(q_1q_3 - q_0q_2) & 2(q_2q_3 + q_0q_1) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}. \tag{7.7}$$

The unit constraint of the quaternion cannot be enforced in the linearization of the Kalman filter. Therefore the incremental Euler angles $(\omega_X, \omega_Y, \omega_Z)$ are used to estimate interframe rotation. At each time step the global rotation quaternion is updated using $\mathbf{q}_k = \mathbf{q}_{k-1} + \delta\mathbf{q}$, where

$$\delta\mathbf{q} = (\sqrt{1-\epsilon}, \omega_X/2, \omega_Y/2, \omega_Z/2) \tag{7.8}$$

$$\epsilon = (\omega_X^2, \omega_Y^2, \omega_Z^2)/4. \tag{7.9}$$

The coordinate transformation from the first frame to the current then becomes

$$\begin{pmatrix} X_C \\ Y_C \\ \beta Z_C \end{pmatrix} = \begin{bmatrix} 1 & & \\ & 1 & \\ & & \beta \end{bmatrix} R(q) \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \\ \beta t_z \end{pmatrix}. \tag{7.10}$$

### 7.2.4 The Kalman Filter

The state vector $\bar{\mathbf{x}}$ in the Kalman filter will consist of the parameters for translation, rotation, focal length and structure for each of the $N$ points:

$$\bar{\mathbf{x}} = (t_X, t_Y, t_Z\beta, \omega_X, \omega_Y, \omega_Z, \beta, \alpha_1, \alpha_2, \ldots, \alpha_N). \tag{7.11}$$

The function $(\hat{u}_1, \hat{v}_1, \hat{u}_2, \hat{v}_2, \ldots, \hat{u}_N, \hat{v}_N) = h_k(\bar{\mathbf{x}})$ in Equation (7.1) is essentially the concatenation of Equations (7.4), (7.10) and (7.3). This is illustrated in Figure 7.5. The first frame coordinates, $(X, Y, Z)$, are calculated, then rotated

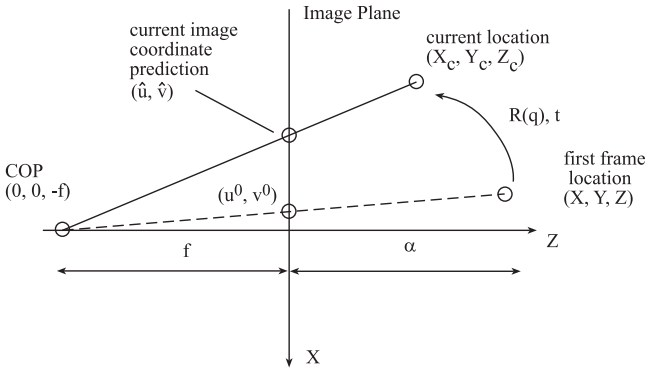



Figure 7.5: *To obtain the current image coordinate prediction $(\hat{u}, \hat{v})$, the following calculation is made: The image coordinates in the original frame $(u^0, v^0)$ and the depth $(\alpha)$ are used to get $(X, Y, Z)$. This point is then moved using $R(q)$ and $\mathbf{t}$ to get the point $(X_c, Y_c, Z_c)$, which is projected using $f = \frac{1}{\beta}$ to $(\hat{u}, \hat{v})$. All these operations make up the function $(\hat{u}, \hat{v}) = h(\bar{\mathbf{x}})$.*

and translated to $(X_C, Y_C, Z_C)$, and finally projected to provide an estimate, $(\hat{u}, \hat{v})$, of the image location of the feature point. The measurement vector $\bar{\mathbf{z}}$ is simply the measured image coordinates of the feature points,

$$\bar{\mathbf{z}} = (u_1, v_1, u_2, v_2, \ldots, u_N, v_N). \tag{7.12}$$

The dynamics function $f_k(\bar{\mathbf{x}})$ is trivially chosen to be the identity $f_k(\bar{\mathbf{x}}) = \bar{\mathbf{x}}$. Thus the motion between frames is modeled as noise. The state estimate $\hat{\mathbf{x}}$ is then updated using

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k-1} + K_k(\bar{\mathbf{z}}_k - h_k(\hat{\mathbf{x}}_k)), \tag{7.13}$$

where $K_k$ is defined as in Appendix C.

## 7.3 Multilinear Constraints

As mentioned in the beginning of this chapter, the most widely used approach for SfM is not based on Kalman filtering, but on multilinear constraints. To be able to compare the methods, a small introduction to the multilinear approach is given here.

Multilinear constraints have proved to be useful tools for solving the SfM problem [22, 7, 31, 58, 29]. They elegantly provide a linear solution to a fundamentally non-linear problem, and can also be used to restrict the search area for additional feature points. Moreover, they can be used on images from uncalibrated cameras, i.e., cameras where intrinsic parameters such as focal length, skew, aspect ratio and principal point position are unknown. The method works along the following lines:

1. The coefficients of the multilinear constraint are calculated from the image coordinates of the points.

2. Possible parameters (both intrinsic and extrinsic) of the cameras are calculated.

3. Triangulation is used to find possible three-dimensional positions for the points.

Note that neither the camera parameters obtained in step 2, nor the three-dimensional positions of the points obtained in step 3, are necessarily the true ones. However, they differ from the correct values only by a projective transformation that is the same for all cameras and points. The terminology used is that the solution is known *up to* a projective transformation. By adding more constraints, e.g., that the intrinsic camera parameters are constant over several images, it is possible to upgrade, or *stratify* the solution up to scale, which is as far as it is possible to come [29].

This section will concentrate on step 1, i.e., the calculation of the coefficients of the multilinear constraint. The investigation will also be limited to the simplest version of the multilinear constraints, the bifocal tensor constraint, which can be conveniently expressed using the *Fundamental matrix*.

Before going into details with the Fundamental matrix, a small introduction to the projective camera model is given.

### 7.3.1 Projective Camera Model

By using homogenous coordinates, Equation (7.2) can be compactly written as

$$\lambda \mathbf{x} = P_C \mathbf{X_C}, \tag{7.14}$$

where $\mathbf{X_C} = (X_C, Y_C, Z_C, 1)^T$ are the homogeneous coordinates of the three-dimensional point in the coordinate system of the camera, and $\mathbf{x} = (x, y, 1)$ are the homogeneous coordinates of its two-dimensional projection in the image plane. The matrix $P_C$, called the camera matrix, equals

$$P_C = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{7.15}$$

which, if used in Equation (7.14), gives the three equations

$$\begin{aligned} \lambda \mathbf{x} &= f X_C \\ \lambda \mathbf{y} &= f Y_C \\ \lambda &= Z_C. \end{aligned} \tag{7.16}$$

With the help of the last equation, the first two equations of Equation (7.16) correspond to Equation (7.2). A rigid transformation moves from the coordinate system of the camera to that of the world:

$$\begin{pmatrix} X_C \\ Y_C \\ Z_C \end{pmatrix} = R \left( \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} - \mathbf{c} \right), \tag{7.17}$$

where $\mathbf{X} = (X, Y, Z)$ are the coordinates in the world coordinate system, $R$ is a rotational matrix and $\mathbf{c}$ is the position of the camera's COP in the world coordinate system. This yields

$$\begin{pmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{pmatrix} = \begin{bmatrix} R & -R\mathbf{c} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}, \tag{7.18}$$

which combined with Equation (7.14) gives

$$\lambda \mathbf{x} = P \mathbf{X}, \tag{7.19}$$

where

$$P = \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} [R| - R\mathbf{c}]. \tag{7.20}$$

For more general cameras, where the intrinsic camera parameters are not known, $P$ can be written

$$P = [A| - A\mathbf{c}], \tag{7.21}$$

or simply

$$P = [A|\mathbf{b}]\,, \tag{7.22}$$

where A is an nonsingular $3 \times 3$ matrix and $\mathbf{b}$ is a $3 \times 1$ vector representing translation.

### 7.3.2 Derivation of the Fundamental matrix

The fundamental matrix was presented by Faugeras [21] as an uncalibrated version of the essential matrix from Longuet-Higgins [44]. The following derivation of the fundamental matrix follows the one by Heyden and Åström [32]. Given a stereo pair of cameras, let $P_l$ represent the "left" camera and $P_r$ represent the "right" camera. Then Equation (7.19) becomes

$$\begin{aligned} \lambda_l \mathbf{x_l} &= P_l \mathbf{X} \\ \lambda_r \mathbf{x_r} &= P_r \mathbf{X} \end{aligned} \tag{7.23}$$

where $\mathbf{x_l}$ and $\mathbf{x_r}$ are the homogenous image coordinates. These equations can be rewritten as

$$\lambda_l \mathbf{x_l} = [A_l|\mathbf{b}_l]\,\mathbf{X} = A_l \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{b}_l \tag{7.24}$$

$$\lambda_r \mathbf{x_r} = [A_r|\mathbf{b}_r]\,\mathbf{X} = A_r \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + \mathbf{b}_r. \tag{7.25}$$

Solving Equation (7.24) for $(XYZ)^T$ yields $(XYZ)^T = A_l^{-1}(\lambda_l \mathbf{x}_l - \mathbf{b}_l)$ which is inserted into Equation (7.25)

$$\lambda_r \mathbf{x}_r = A_r A_l^{-1}(\lambda_l \mathbf{x}_l - \mathbf{b}_l) + \mathbf{b}_r. \tag{7.26}$$

Using

$$\mathbf{a} = A_r A_l^{-1} \mathbf{x}_l \tag{7.27}$$

and

$$\mathbf{t} = \mathbf{b}_r - A_r A_l^{-1} \mathbf{b}_l, \tag{7.28}$$

yields

$$\lambda_r \mathbf{x}_r = \lambda_l \mathbf{a} + \mathbf{t}. \tag{7.29}$$

Thus the vector $\lambda_r \mathbf{x}_r$ is a linear combination of the two other vectors, i.e., the three vectors are contained in the same plane as illustrated in Figure 7.6. The normal $\mathbf{n}$ of the plane can be obtained by taking the cross-product between $\mathbf{t}$ and $\mathbf{x}_r$, $\mathbf{n} = \mathbf{t} \times \mathbf{x}_r$ or

$$\mathbf{n} = T_{\mathbf{t}} \mathbf{x}_r, \tag{7.30}$$

where $T_{\mathbf{t}}$ is a $3 \times 3$ matrix that performs the cross-product. The condition that $\mathbf{a}$ is contained in the plane can now be written

$$\mathbf{a}^T \mathbf{n} = 0. \tag{7.31}$$

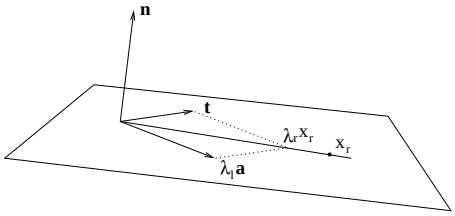



Figure 7.6: *The three vectors* $\mathbf{a}$*,* $\mathbf{t}$ *and* $\mathbf{x}_r$ *are all contained in the same plane.*

Substituting $\mathbf{a} = A_r A_l^{-1}\mathbf{x}_l$ and $\mathbf{n} = T_\mathbf{t}\mathbf{x}_r$ gives

$$(A_r A_l^{-1}\mathbf{x}_l)^T T_\mathbf{t}\mathbf{x}_r = 0. \tag{7.32}$$

Rearranging yields

$$\mathbf{x}_l^T A_l^{-T} A_r^T T_\mathbf{t}\mathbf{x}_r = 0, \tag{7.33}$$

which can be written

$$\mathbf{x}_l^T F\mathbf{x}_r = 0, \tag{7.34}$$

where $F = A_l^{-T} A_r^T T_\mathbf{t}$ is the fundamental matrix. Since the right hand side of Equation (7.34) is zero, the fundamental matrix is only defined up to scale. Adding the constraint

$$||F||_F = 1 \tag{7.35}$$

gives a unique solution[1].

### 7.3.3 Estimation of $F$, $A$ and b

Continuing along the lines of [32]; Equation (7.34) is linear in the coefficients of $F$, and can hence be rewritten as $\mathbf{m}_k F_{vec} = 0$, where

$$\mathbf{m}_k = \begin{pmatrix} x_l x_r & x_l y_r & x_l & y_l x_r & y_l y_r & y_l & x_r & y_r & 1 \end{pmatrix} \tag{7.36}$$

for point $k$ and $F_{vec}$ is a vector containing the elements of $F$. Using $(\mathbf{m}_k)_{k=1}^8$ from eight different point correspondences as rows in the matrix $M$ yields

$$MF_{vec} = 0, \tag{7.37}$$

which can be solved for $F_{vec}$ using singular value decomposition (SVD). Since the world coordinate system in unknown, it is possible to choose $A_l = I$ and $b_l = 0$. $F$ then becomes $A_r^T T_\mathbf{t}$, and a valid $A_r$ and $T_\mathbf{t}$ can be found by decomposing $F$ into one non-singular and one skew-symmetric matrix. Decomposing $F$ with SVD gives $F = U\Sigma V$, were $\Sigma = \mathrm{diag}(a, b, 0)$. By using the two decompositions $\Sigma = \Sigma_1 T_1$ and $I = QQ^{-1}$, where

$$\Sigma_1 T_1 = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \text{and} \quad QQ^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{7.38}$$

[1]The subscript $F$ here stands for the Frobenius norm, $||A||_F = \sqrt{\sum_{i,j} a_{ij}^2}$.

$F$ can be rewritten

$$\begin{aligned} F =& U\Sigma_1 T_1 V \\ F =& U\Sigma_1 Q Q^{-1} T_1 V \\ F =& U\Sigma_1 Q V V^T Q^{-1} T_1 V. \end{aligned} \tag{7.39}$$

If $c$ is selected so that $\det(\Sigma_1) = 1$, $U\Sigma_1 QV$ is a non-singular matrix and can be identified as $A_r^T$. Likewise, $V^T Q^{-1} T_1 V$ is a skew-symmetric matrix that can be used as $T_t$, which in turn is used to obtain $\mathbf{b}_r$.

### 7.3.4 Image-to-Image Homography

As shown in Figure 7.7, a point $(XYZ)^T$ belonging to plane $\pi$ can be expressed

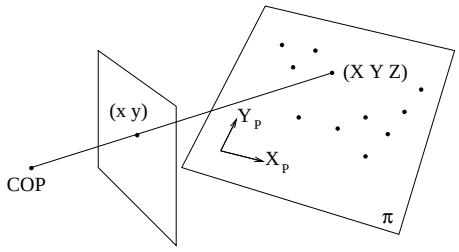



Figure 7.7: *Coplanar points; all points $(XYZ)^T$ belong to the plane $\pi$.*

through its plane coordinates $\mathbf{X}_P = (X_P, Y_P, 1)$ as

$$(XYZ)^T = X_P \bar{e}_{xp} + Y_P \bar{e}_{yp} + \mathbf{c}_p, \tag{7.40}$$

where $\bar{e}_{xp}$ and $\bar{e}_{yp}$ are two non-parallel vectors in $\pi$ and $\mathbf{c}_p$ is a point in $\pi$. Representing again the left and right camera with $\lambda_l \mathbf{x}_l = P_l \mathbf{X}_l$ and $\lambda_r \mathbf{x}_r = P_r \mathbf{X}_r$ respectively and using Equation (7.24), gives

$$\lambda_l \mathbf{x}_l = A_l \left\{ \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} + A_l^{-1} \mathbf{b}_l \right\}. \tag{7.41}$$

Inserting (7.40) gives $\lambda_l \mathbf{x}_l = A_l \left\{ X_P \bar{e}_{xp} + Y_P \bar{e}_{yp} + \mathbf{c}_p + A_l^{-1} \mathbf{b}_l \right\}$, which can be written on matrix form

$$\lambda_l \mathbf{x}_l = A_l \begin{bmatrix} | & | & | \\ \bar{e}_{xp} & \bar{e}_{yp} & \mathbf{c}_p + A_l^{-1} \mathbf{b}_l \\ | & | & | \end{bmatrix} \begin{pmatrix} X_P \\ Y_P \\ 1 \end{pmatrix} = A_l B_l \mathbf{X}_P = H_l \mathbf{X}_P. \tag{7.42}$$

Here $B_l$ (and thus $H_l$) is non-singular as long as the COP is outside the plane $\pi$. The relation $\lambda_l \mathbf{x}_l = H_l \mathbf{X}_P$ between the plane coordinates and the (left) image coordinates is called a homography. A similar relationship $\lambda_r \mathbf{x}_r = H_r \mathbf{X}_P$ is available for the right camera. Combining these yields

$$\lambda \mathbf{x}_l = H \mathbf{x}_r, \tag{7.43}$$

where $H = H_l H_r^{-1}$ and $\lambda = \frac{\lambda_l}{\lambda_r}$. Thus there exists a homography also between the coordinates of the left and right image, here called the *image-to-image homography*.

# Chapter 8

# Degeneracies

In this chapter, the behavior of the SfM algorithms presented in the last chapter will be examined for degenerate situations, more specifically when the motion is purely rotational or when the object is planar. Both these degenerate cases are important for head tracking: The motion of the head might be close to a pure rotation (just keeping the head still is a special case of this), and random points on the surface of a face might be close to co-planar. The behavior of algorithms based on multilinear constraints for these cases is well documented [61, 28, 70, 22], but will be repeated here as a background. The fact that the EKF-based SfM algorithm can recover motion from sequences with pure rotation is also known [8]. It is less well known how the EKF-based algorithm behaves in the case of planar objects, and this investigation is the main goal of this chapter.

The first section will go through the behavior of multilinear constraint based algorithms for pure rotation and planar structure. After this, the EKF-based SfM algorithm will undergo the same investigation.

## 8.1 Pure Rotation and Multilinear Constraints

Recovering three-dimensional structure from two images taken with cameras too close to each other is inherently an ill-posed problem. As the base line, i.e., the distance between the two cameras' centers of projection decreases, triangulation gets increasingly numerically unstable. However, as pointed out by Faugeras [22], it is still possible to calculate the relative *motion* (in this case, the rotation) between the two cameras. Figure 8.1 illustrates a situation where there is only in-plane rotation between the two cameras, i.e., there is no translatory motion and the cameras both have the same COP and principal point (the projection of the COP onto the image plane). From this figure it is evident that it is impossible to obtain depth information about any point $\mathbf{X}$ on the object. However, it is equally evident that it should be possible to estimate the motion (in this case, the rotation angle $\theta$) between the two cameras. Examining

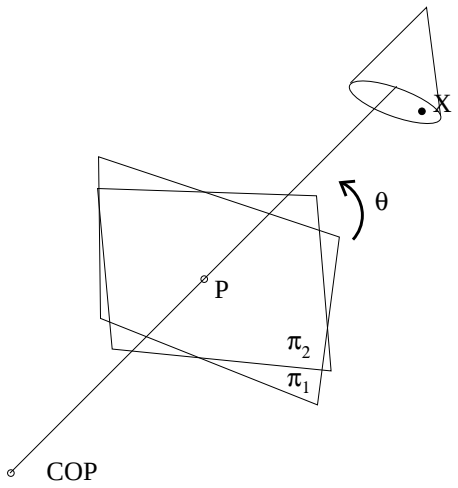



Figure 8.1: *A degenerate motion case — the cameras COPs overlap. Evidently, it is not possible to obtain any depth estimate of any point* $\mathbf{X}$ *on the object, but it should be possible to estimate the relative rotation* $\theta$ *between the two cameras.*

Equation (7.28)

$$\mathbf{t} = \mathbf{b}_r - A_r A_l^{-1} \mathbf{b}_l, \tag{8.1}$$

gives that $\mathbf{t}$ is in fact the base line vector, expressed in the coordinate system of the right camera. Thus, if the base line $\mathbf{t}$ is zero, Equation (7.29) becomes

$$\lambda_r \mathbf{x}_r = \lambda_l \mathbf{a}. \tag{8.2}$$

That is, $\mathbf{a}$ and $\mathbf{x}$ lie on the same line, as illustrated in Figure 8.2. Since a line

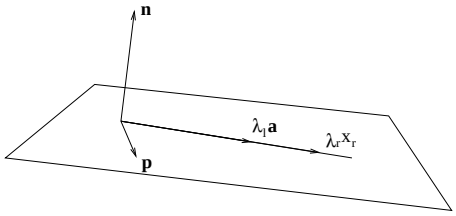



Figure 8.2: *When the base line* $\mathbf{t}$ *is zero, the vectors* $\mathbf{a}$ *and* $\mathbf{x}_r$ *are parallel. Thus an arbitrary vector* $\mathbf{p}$ *can be used to construct the plane.*

and a vector always lie in a plane, an arbitrary vector $\mathbf{p}$ can be used to create the normal $\mathbf{n} = \mathbf{p} \times \mathbf{x}_r = P_{\mathbf{p}} \mathbf{x}_r$. Equation (7.33) now becomes

$$\mathbf{x}_l^T A_l^{-T} A_r^T P_{\mathbf{p}} \mathbf{x}_r = 0, \tag{8.3}$$

and $F = A_l^{-T} A_r^T P_{\mathbf{p}}$. Of course another vector $\mathbf{p}_2$ can be used — yielding a different matrix $F_2 = A_l^{-T} A_r^T P_{\mathbf{p}_2}$ — and there are thus in fact an infinite number of matrices $F$ that satisfy the expression in Equation (7.34). Since $\mathbf{p}_2$ has

three degrees of freedom, $F_2$ is not simply a scaled version of $F$. Adding the constraint (7.35) will thus not be sufficient for uniqueness. Therefore it is not a well-posed problem to identify the elements of $F$ when the COPs overlap. In practice, numerical instabilities will occur not only when $\mathbf{t}$ equals zero but for all small $\mathbf{t}$.

### 8.1.1 Avoiding Ill-Conditioning

In a real head-tracking scenario, it is very plausible that the head sometimes moves insignificantly or not at all. Since $\mathbf{t} = 0$ in this case, this common situation is in fact degenerate. A robust SfM system should recognize when the motion is degenerate and avoid trying to estimate the coefficients of the multilinear constraint in those cases. For the case of pure rotation between the left and right image, it turns out that, just as in the case of coplanar points, there exists a homography between the coordinates,

$$\lambda \mathbf{x}_l = H \mathbf{x}_r. \tag{8.4}$$

The matrix $H$ is the same for any pair of corresponding image points $\mathbf{x}_l$ and $\mathbf{x}_r$. If the object is known to be non-planar, Equation (8.4) can be used as a test on the motion. If it holds, the motion is regarded degenerate and the rotation is calculated from the homography. If it is violated, the motion is non-degenerate and the fundamental matrix can be estimated. It is also possible to do global motion estimation based on Equation (8.4): If the motion estimation provides a good estimation of the next frame, motion is deemed to be degenerate, whereas if it is a poor estimation, motion is non-degenerate. Such a method is used in [49], but is batch oriented and thus not directly applicable to the recursive case.

## 8.2 Planar Objects and Multilinear Constraints

Planar objects are important for two reasons. First, man-made structures contain many planar objects such as walls, floors, desks and so forth. Second, some objects might be close to planar. In the case of face tracking for instance, a small number of randomly picked points on the surface of the face might be close to coplanar. If the SfM algorithm fails for planar objects, it is likely to do poor on such nearly planar objects. It is a well-known fact that planar objects constitute a singular case that SfM algorithms have problems with. For methods based on multilinear constraints, these problems manifest themselves in the estimation of the multilinear coefficients [70, 61, 28]. In the case of the fundamental matrix for instance, Equation (7.34) will have an infinite number of solutions and will not be solvable without additional constraints, as shown below:

THEOREM 1 *If all points lie on a plane, and there exists a solution $F_0$ to the fundamental matrix equation $\mathbf{x}_l^T F_0 \mathbf{x}_r = 0$, then $F_0 + F_s$ is also a solution, where*

*$F_s = G_s H$, $H$ is the image-to-image homography ($\lambda \mathbf{x}_l = H\mathbf{x}_r$), and $G_s$ is any antisymmetric matrix $G_s = -G_s^T$.*

PROOF 1 Starting with the fundamental matrix equation,

$$\mathbf{x}_l^T F_0 \mathbf{x}_r = 0, \tag{8.5}$$

(7.43) is used to obtain

$$\mathbf{x}_l^T F_0 H^{-1} \mathbf{x}_l = 0. \tag{8.6}$$

The lambda is excluded since it does not affect the right hand side. Now, take an arbitrary vector $\mathbf{s}$ to create the cross-product matrix $G_s|G_s y = \mathbf{s} \times y$. $G_s \mathbf{x}_l$ is thus orthogonal to $\mathbf{x}_l$, so

$$\mathbf{x}_l^T G_s \mathbf{x}_l = 0, \tag{8.7}$$

which is added to (8.6):

$$\mathbf{x}_l^T F_0 H^{-1} \mathbf{x}_l + \mathbf{x}_l^T G_s \mathbf{x}_l = 0. \tag{8.8}$$

Using $F_s = G_s H$ (and thus $F_s H^{-1} = G_s$), this can be rearranged to

$$\mathbf{x}_l^T F_0 H^{-1} \mathbf{x}_l + \mathbf{x}_l^T F_s H^{-1} \mathbf{x}_l = 0, \tag{8.9}$$

and further to

$$\mathbf{x}_l^T (F_0 + F_s) H^{-1} \mathbf{x}_l = 0, \tag{8.10}$$

which, by using (7.43), can be simplified to

$$\mathbf{x}_l^T (F_0 + F_s) \mathbf{x}_r = 0, \tag{8.11}$$

and that completes the proof.

Again, since $G_s$ has three degrees of freedom, it is easy to see that $F_0 + F_s \nsim F_0$, and adding the constraint (7.35) is not sufficient for uniqueness. Thus, when all points lie on a plane, it is not possible to identify the elements of the fundamental matrix from Equation (7.34). As in the case with overlapping COPs, numerical instabilities will occur when points are close to coplanar. Stein and Shashua show that this is also true for objects made up of several planes that intersect in a single line, and also point to several real world objects that fit this description [61].

### 8.2.1 Adding Calibration Constraints

If constraints on the camera parameters are added to the multilinear constraints, the solution becomes unique. In the calibrated case for instance, $F$ can be written $F = R^T T_t$, where $R$ is a rotation matrix and $T_t$ is an antisymmetric matrix representing translation. This additional constraint in combination with Equation (7.34) is sufficient for the estimation of $F$. However, instead of the appealing linear simplicity of Equation (7.37), the equations become non-linear.

A similar solution should also be possible in the trilinear tensor case. Since one more image is used, it should be possible to also recover one camera parameter, e.g., the focal length. However, the equations will be even more complicated than in the two-view case. The method is also a deviation from the strategy of first finding a projective solution and then stratifying it to a Euclidean one.

### 8.2.2 Known Planar Structure

If the object is known a priori to be planar, specific techniques can be used to recover the structure. Instead of estimating the fundamental matrix, Wunderlich estimates the image-to-image homography between two images in the fully calibrated case [75]. The homography is then used to calculate the structure and motion. Triggs generalizes this to the non-calibrated case [71], where more images are needed. Using several image-to-image homographies, Triggs estimates structure, motion and camera parameters using non-linear optimization. Triggs also shows that, when $n$ intrinsic camera parameters are unknown (but constant),

$$m = \lceil \frac{n+4}{2} \rceil \tag{8.12}$$

images are needed for reconstruction of the Euclidean solution [71]. This is valid for any SfM algorithm in the case of planar objects.

A solution to the SfM problem can be to use the multilinear constraints only if dealing with a general three-dimensional object, and use a specific technique if the object is planar. If the motion is known not to be purely rotational, it is possible to distinguish planar objects from general ones by using the image-to-image homography

$$\lambda \mathbf{x}_l = H \mathbf{x}_r \tag{8.13}$$

as a test. If it is violated, the structure is general and the normal multilinear approach can be used. If it is satisfied, the object is planar, and techniques such as the one by Triggs [71] can be used.

## 8.3 Pure Rotation and EKF-based SfM

Azarbayejani and Pentland evaluate their algorithm for rotational, parallel and axial motion using a Monte Carlo experiment [8]. Pure rotation (rotation around COP), which is a degenerate form of motion, is also investigated using the same methodology. The rotational component is found to be correctly recovered, whereas the structure obviously cannot be recovered (see Section 8.1 and Figure 8.1).

## 8.4 Planar Objects and EKF-based SfM

The following part of this chapter analyzes the extended Kalman filter approach to the Structure from Motion problem for the case when all points of the object

belong to a plane. It is shown that, even though the Kalman filter is under-constrained in each time step, it will most often converge to the correct solution. A tentative version of this work was published by the author in [69] but treated only the noise free case.

Three ways to analyze the problem are used. The first way is to look at the SfM problem as a minimization problem and investigate numerically whether the true solution corresponds to a valid minimum. The second way is to use Triggs' formula (Equation (8.12)), and the third way is to do Monte Carlo experiments on synthetic data to see how often the algorithm converges.

### 8.4.1 Analysis

If only the first and the $k^{th}$ views are considered, the SfM problem can be formulated as the problem of finding the state vector $\mathbf{x}$ which minimizes the (scalar) error function

$$J_k(\mathbf{x}) = (\mathbf{z}_k - \mathbf{h}_k(\mathbf{x}))^T(\mathbf{z}_k - \mathbf{h}_k(\mathbf{x})) \tag{8.14}$$

where $\mathbf{h}_k(\mathbf{x})$ consists of the Kalman filter's estimate of the projected points $(\hat{u}_1, \hat{v}_1, \hat{u}_2, \hat{v}_2, \ldots, \hat{u}_N, \hat{v}_N)$. If the correct solution $\mathbf{x} = \mathbf{x}^*$ is known, $J_k(\mathbf{x})$ can be Taylor expanded around $\mathbf{x}^*$ according to

$$\begin{aligned} J_k(\mathbf{x}) \approx J_k(\mathbf{x}^*) + \frac{\partial J_k(\mathbf{x}^*)}{\partial \mathbf{x}}(\mathbf{x} - \mathbf{x}^*) + \\ + \frac{1}{2}(\mathbf{x} - \mathbf{x}^*)^T \frac{\partial^2 J_k(\mathbf{x}^*)}{\partial \mathbf{x}^2}(\mathbf{x} - \mathbf{x}^*). \end{aligned} \tag{8.15}$$

If the Hessian $\frac{\partial^2 J_k(\mathbf{x}^*)}{\partial \mathbf{x}^2}$ is positive definite, $\mathbf{x}^*$ will be a local minimum to $J_k(\mathbf{x}^*)$, which means that, locally, there is no $\mathbf{x}$ that minimizes $J_k(\mathbf{x})$ as well as $\mathbf{x}^*$ does. The goal of the analysis is to calculate the Hessian at $\mathbf{x}^*$ for a number of point constellations, and analyze if it is positive definite or not. A positive definite Hessian indicates that the function has a unique optimum (at least locally), whereas a Hessian which has one or more eigenvalues of zero will allow an entire manifold of solutions $\mathbf{x}$ that will minimize the cost function $J_k(\mathbf{x})$ just as well as the known optimum, $\mathbf{x}^*$, will. Note that a positive definite Hessian in $\mathbf{x}^*$ only shows the uniqueness of the solution locally for a particular point constellation, motion and focal length. Nevertheless, these particular cases can help to understand how it works generally.

Differentiating Equation (8.14) yields

$$\frac{\partial^2 J_k}{\partial \mathbf{x}^2} = 2\frac{\partial h_k}{\partial \mathbf{x}}^T \frac{\partial h_k}{\partial \mathbf{x}} + 2\sum_k \frac{\partial^2 h_k}{\partial \mathbf{x}^2}(h_k(\mathbf{x}) - \mathbf{z}_k). \tag{8.16}$$

Note that if the measurements are perfect, $h_k(\mathbf{x}^*) = \mathbf{z}_k$ and the second term of Equation (8.16) is zero. Hence, in $\mathbf{x}^*$, $\frac{\partial^2 J_k}{\partial \mathbf{x}^2}$ can be conveniently calculated

as $2\frac{\partial h_k}{\partial \mathbf{x}}^T \frac{\partial h_k}{\partial \mathbf{x}}$. Since $J_k(x) \geq 0$, negative eigenvalues are impossible, and the Hessian is thus positive definite unless it has a zero-valued eigenvalue. This combined with the fact that the Hessian is symmetric means that its eigenvalues will equal its singular values. Hence it suffices to investigate if the Hessian has any singular values that are zero. Since $\alpha_1 \equiv 1.0$ for scaling reasons, $J_k(\mathbf{x})$ is no longer a function of $\alpha_1$ and the corresponding column must be removed from $\frac{\partial h_k}{\partial \mathbf{x}}$ before calculating $\frac{\partial^2 J_k}{\partial \mathbf{x}^2}$.

### 8.4.2 Two Views of a Three-Dimensional Object

A random three-dimensional object containing 16 points has been rotated and translated randomly to produce two-dimensional measurements that are fed into a SfM EKF. After rotating and translating the object to a new position, the Hessian $\frac{\partial^2 J_k}{\partial \mathbf{x}^2}$ is calculated in the known optimum point $\mathbf{x}^*$, and is decomposed using SVD. The singular values are plotted in the left part of Figure 8.3. The

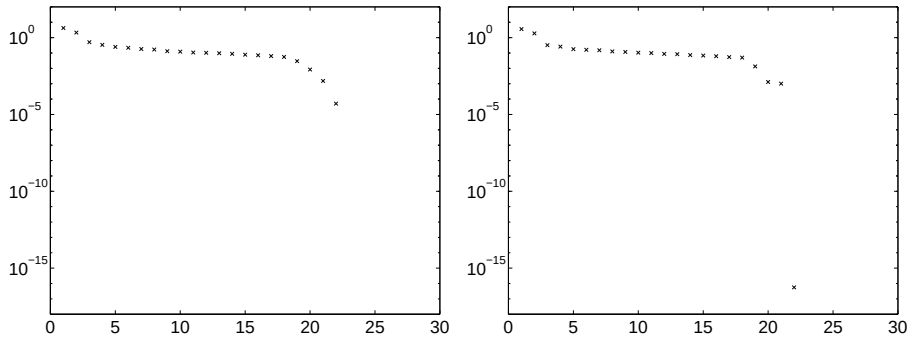



Figure 8.3: *Singular values for the Hessian of $J_k(\mathbf{x})$ in $\mathbf{x}^*$. Left: a 3D object. Right: an object where all points lie in a plane.*

smallest singular value is at around $10^{-5}$, so it is fair to say that the Hessian is positive definite.

### 8.4.3 Two Views of a Planar Object

The right diagram in Figure 8.3 shows the result from repeating the experiment with points that lie in a plane. The smallest singular value is now $10^{-17}$ — several orders of magnitude smaller than the others — practically zero. Thus, locally, there exists a curve $x_k(t)$ for which $J_k(x_k(t)) = 0$. Each point on this curve represents a false solution to the SfM problem. An example of a false solution is shown in Figure 8.4, obtained by letting the filter converge and then deliberately go in the direction corresponding to the zero valued singular value. The two leftmost images are the ones that are part of the optimization function $J_k(\mathbf{x})$, namely the first image and the $k$th image. The crosses (estimated points) fit nicely into the circles (true points). It is not until a third view is seen, (the

rightmost image in Figure 8.4), that it becomes evident that this solution is false.

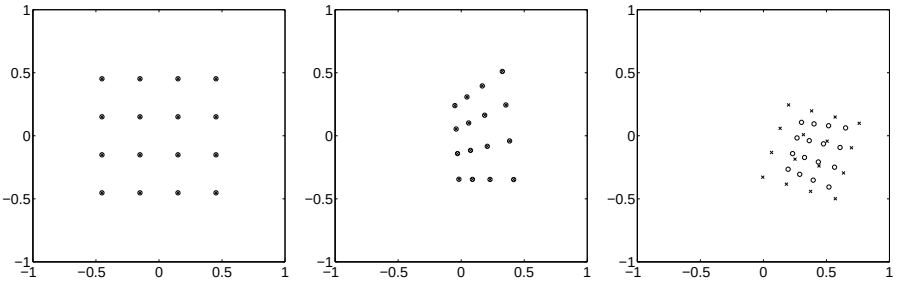


Figure 8.4: *A false solution. The image of the real object is marked with circles, and the reconstructed object is marked with crosses (they appear almost as filled circles in the first two diagrams). In the 1st and the kth image (left, middle), they match ($J_k(\mathbf{x}) = 0$), but from another view (right) it is obvious that the solution is false.*

### 8.4.4 Three Views of a Planar Object

As seen above, two views of a planar object with unknown focal length is not sufficient for a unique solution of the SfM problem. This is consistent with the results by Triggs [71]: Since $n = 1$, (the focal length is the only unknown intrinsic parameter), $\lceil \frac{1+4}{2} \rceil = 3$ images are needed. To investigate the case $m = 3$ for the EKF method, a modified Kalman filter is constructed that considers not only the current image, but also the previous one. The state vector in Equation (8.17) is thus extended with another six degrees of freedom $(t_X^2, t_Y^2, t_Z^2\beta, \omega_X^2, \omega_Y^2, \omega_Z^2)$ for the camera motion of the previous image:

$$\begin{aligned} \mathbf{x}^e = (&t_X^2, t_Y^2, t_Z^2\beta, \omega_X^2, \omega_Y^2, \omega_Z^2, \\ &t_X, t_Y, t_Z\beta, \omega_X, \omega_Y, \omega_Z, \\ &\beta, \alpha_1, \alpha_2, \ldots, \alpha_N). \end{aligned} \tag{8.17}$$

The measurement vector is the concatenation of the two latest measurements,

$$\mathbf{z}_k^e = [\mathbf{z}_k | \mathbf{z}_{k-1}]\,. \tag{8.18}$$

Analogously with Equation (8.14) and Equation (8.16),

$$J_k^e(\mathbf{x}) = (\mathbf{z}_k^e - \mathbf{h}_k^e(\mathbf{x}^e))^T (\mathbf{z}_k^e - \mathbf{h}_k^e(\mathbf{x}^e)) \tag{8.19}$$

and

$$\frac{\partial^2 J_k^e(\mathbf{x}^{e*})}{\partial \mathbf{x}^{e2}} = 2 \frac{\partial h_k^e}{\partial \mathbf{x}^e}^T \frac{\partial h_k^e}{\partial \mathbf{x}^e}. \tag{8.20}$$

An experiment with a planar object using this modified Kalman filter has been

carried out. Rather big steps in the camera motion are taken in order to get reasonably different views. The Hessian is calculated and decomposed using SVD. In Figure 8.5 the singular values for such an experiment is shown. Note that the smallest singular value is about $10^{-4}$, far greater than the $10^{-17}$ obtained in the two view case, and the Hessian in the known optimum $x^{e*}$ is thus positive definite. This means that, locally, there is no other constellation of points and

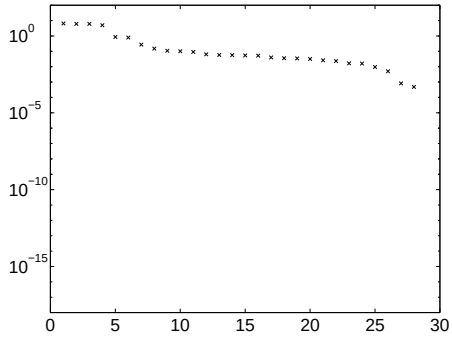



Figure 8.5: *Singular values for the Hessian in* $\mathbf{x}^{e*}$ *of the extended error function* $J_k^e(\mathbf{x})$.

camera positions that will generate zero error.

### 8.4.5 Several Views of a Planar Object

The Extended Kalman filter approach to the SfM problem does not rely on only two or three images, but instead it takes an entire sequence of images into account. Whether the (non-modified, two-view-) filter will converge for a planar object is not obvious: On the one hand, the previous section indicates that three images are enough for uniqueness. On the other hand, the constraints of all the images are not applied simultaneously; at each time step $k$, the problem will be under-determined and a manifold of false structures will be possible. To resolve this a Monte Carlo experiment is conducted. Random points are selected from a randomly oriented planar object of size $0.5 \times 0.5$ area units. The object undergoes random motion and the point measurements are forwarded to the Kalman filter. The filter runs for 1000 frames and if the summed squared error between the estimated depth values and the real ones is smaller than 0.1, convergence is declared. The experiment is repeated 1000 times using two types of motion: Rotation around a random vector in the $z = 0$ plane (to avoid the degenerate rotation around the $z$-axis) and "Brownian motion", i.e., small incremental random steps in translation and rotation. The filter is initialized with $\beta = 2.0$ (the true value is $\beta = 1.3$). Three types of initialization procedures are tried for the depths $\alpha_{1\ldots N-1}$: In the first type, called "prior 1", the $\alpha$-values are set to random values in the interval $\pm 0.5$ around the depth $\alpha_0$ which is fixed to 1.0. In the second type, called "prior 2", the filter is initialized with the true $\alpha$-values plus random noise in the interval $\pm 0.5$. The last one, "prior 3," is equivalent to

| | prior 1 | prior 2 | prior 3 |
|---|---|---|---|
| 2D rot | 0.673 | 0.953 | 1.000 |
| 3D rot | 0.796 | 1.000 | 1.000 |
| 2D brown | 0.751 | 0.972 | 1.000 |
| 3D brown | 0.762 | 1.000 | 1.000 |

Table 8.1: Depth convergence frequency for noise free measurements. $N = 1000$, $\sigma \leq 0.016$.

| | prior 1 | prior 2 | prior 3 |
|---|---|---|---|
| 2D rot | 0.656 | 0.898 | 0.987 |
| 3D rot | 0.802 | 0.995 | 1.000 |
| 2D brown | 0.508 | 0.851 | 0.924 |
| 3D brown | 0.478 | 0.943 | 0.981 |

Table 8.2: Depth convergence frequency for noisy measurements. $N = 1000$, $\sigma \leq 0.016$.

"prior 2" but with noise in the interval $\pm 0.25$.

The entire experiment is then repeated for a three-dimensional object obtained by randomly sampling points from a $0.5 \times 0.5 \times 0.5$ cube and then rotating those points randomly. This time, "prior 1" will mean that all the $\alpha$-values are initialized to the same value 1.0, whereas the "prior 2" and "prior 3" will mean that the $\alpha$-values are initialized to the correct value plus random noise in the interval $\pm 0.5$ and $\pm 0.25$ respectively.

The result is shown in Table 8.1 for the case of noise free measurements. As can be seen, there are differences between the planar (2D) and the general (3D) objects, but as the quality of the depth prior improves, the convergence frequency goes to unity for both types of objects. In the case of noisy measurements, the difference is larger. In Table 8.2, uniformly distributed noise of one pixel variance is added to the measurements (one pixel equals a 1/160th of the distance from the image center to the right hand image border). Still, the difference is quantitative, not qualitative. One example is when tracking a planar object undergoing Brownian motion, with depth prior 3 (third row, third column). In this case one has at least as much to gain by changing the motion to rotational as by changing the planar object to a general three-dimensional one (convergence frequency rises to 0.987 compared to 0.981). Hence, a planar object is not a catastrophic situation that means that the SfM algorithm will unconditionally fail. Rather, planar objects can be seen as something that should be avoided if possible, just like measurement noise, poor depth priors or low excitation in the motion. This is illustrated in Figure 8.6. If the noise-, shape-, prior-

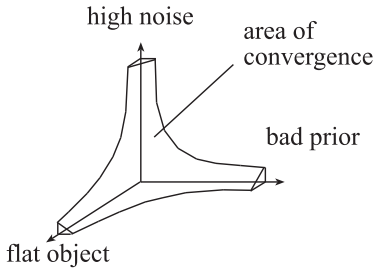



Figure 8.6: *The convergence of the EKF based algorithm depends on the noise- shape-prior- and motion-excitation conditions. For simplicity only the first three axes are drawn in the figure.*

and motion-excitation conditions are good enough, the algorithm will converge. Even a completely planar object will converge if the other conditions are good.

A way to understand how the filter converges for planar views is the following. Each new image $k$ gives a new constraint $J_k(\mathbf{x}) = 0$, which will be satisfied by an entire curve of false solutions $x_k(t)$. If this curve is projected down to only the structure components of $x$, a new curve $s_k(t)$ is obtained, where $\mathbf{s} = (\beta, \alpha_1, \alpha_2 \ldots \alpha_N)$. Since the solution is unique for three views, $s_k(t)$ and $s_{k+1}(t)$ cannot be the same curve, and must meet in the point $\mathbf{s}^*$ that represents the correct structure. This is illustrated in Figure 8.7. The estimate $\hat{s}$

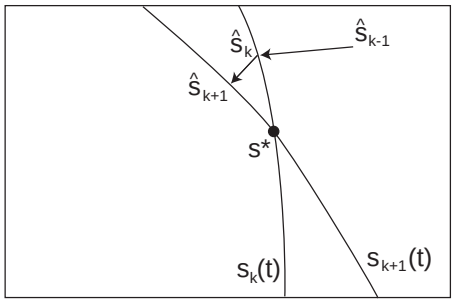



Figure 8.7: *Structure Convergence*

will move towards the curves, but for each new image the curve has moved and $\hat{s}$ will continue moving until it has found the optimum. This should result in a slower convergence than in the over-determined case, and this has also been verified experimentally: The top diagram in Figure 8.8 shows a histogram over how many frames are needed to have a sum squared depth error of 0.05 for rotational motion. In the bottom diagram, a similar histogram is drawn for Brownian motion. It is clear that the general three-dimensional objects (solid curve with crosses) converges faster than the planar ones (dotted curve with circles), but in this case the type of motion seems to have a bigger impact on

the convergence time than has the shape of the object.

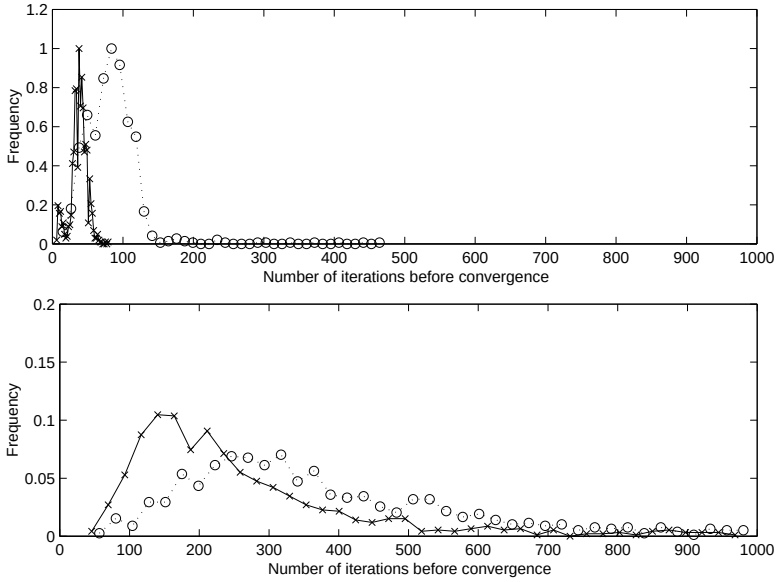



Figure 8.8: *Top: Histogram over convergence time for rotational motion with a general object (solid) and a planar object (dotted). Bottom: Ditto for Brownian motion.*

In fact, the extended Kalman filter does not only try to minimize Equation (8.14) at each frame, but also takes into account the covariances of the measurements and of the state vector from the previous frame. This further constrains the solution $\hat{x}_k$ obtained at time $k$ and prevents it from moving to a point too far away in the state space, even if the error function $J_k(\cdot)$ happens to be zero there.

### 8.4.6 Using the Three-View Filter

The modified three-view filter in Equation (8.17) was introduced to investigate whether three images would be sufficient for a unique solution of the EKF based algorithm. However, it can also be used in lieu of the two-view filter for estimating structure and motion. Since the modified filter fuses the information of three views, it should be over-determined in theory. In practice, however, the two last views are often quite similar to each other, which means that the smallest singular value will still be quite small, and the results should not differ much from those of the two-view filter. This has been validated experimentally: Table 8.3 shows the convergence frequencies for rotational motion in the case of noise for both the two-view filter (top) and the three-view filter (middle). Performance is slightly worse for prior 1, slightly better for prior 2, and roughly equal for prior 3. Increasing the number of frames between the two measurements in Equation (8.18) increases the chance that "enough" motion has occurred. This

| | prior 1 | prior 2 | prior 3 |
|---|---|---|---|
| 2 view filter: frame 0 and $k$ | 0.656 | 0.898 | 0.987 |
| 3 view filter: frame 0, $k-1$ and $k$ | 0.628 | 0.924 | 0.982 |
| 3 view filter: frame 0, $k-8$ and $k$ | 0.627 | 0.856 | 0.964 |

Table 8.3: *Comparison of depth convergence frequency for the two-view filter and two types of three-view filters for noisy measurements.* $N = 1000$, $\sigma \leq 0.016$.

can increase the smallest singular value substantially. In the example in Section 8.4.4 where large motion was used between the measurements, the smallest singular value of $10^{-4}$ was actually bigger than that of the non-planar object in Section 8.4.2. Experimentally however, no gain in convergence frequency has been observed. The last row of Table 8.3 shows results from a three-view filter that considers frame number 0, $k-8$ and $k$. The motion between frame $k-8$ and frame $k$ is much bigger than that between frame $k-1$ and $k$, but that is not reflected in better convergence frequencies. On the contrary; convergence results (bottom row) are worse than for the one-image-difference filter (middle row) for all three priors, and also worse than for the two-view filter (top row).

## 8.5 Conclusions

This chapter has provided an analysis of how different SfM algorithms behave for pure rotation and planar objects. As is known from the literature and has also been seen here, the coefficients of the multilinear constraints cannot be estimated from the linear equations only in these cases [70, 61, 28].

Table 8.4 summarizes SfM algorithms for planar objects. As seen in the first column, projective reconstruction from multilinear constraints is not possible in the uncalibrated case. Note that in all cases where reconstruction is possible (marked with "yes"), the algorithms either only work for planar objects (*) or are non-linear (**). For instance, the methods by Wunderlich [75] and Triggs [71] work only for objects that are known to be planar.

If the object is known not to be planar, the image-to-image homography can be used to determine if the motion is purely rotational. If so, the rotation can be recovered from the homography, otherwise the normal multilinear approach can be used.

If the motion is known not to be purely rotational, the image-to-image homography shows if the structure is planar. If it is, Triggs' method can be used, otherwise normal multilinear constraints will work.

If it is not known whether the structure is planar or the motion is purely rotational, the situation is more complicated. If the homography is satisfied, it can

| | Projective rec. from multilinear constr. | Euclidean reconstruction | | |
|---|---|---|---|---|
| | (uncal.) | uncal. | f unknown | calibr. |
| 2 views | no<br>↑ | no<br>[71] | no<br>[71] | yes<br>(b)**<br>[75]* |
| 3 views | no<br>[28, 61] | no<br>[71] | yes<br>[71]*,** | ↓<br>yes |
| 5 views and more | n/a | yes<br>[71]*,** | yes<br>(a)**<br>[71]*,** | yes<br>(a)**<br>(b)**<br>[75]* |

Table 8.4: *SfM algorithms that work for planar surfaces. A cell is marked with "yes" if reconstruction is possible in this case, and with "no" otherwise, with references in brackets. Symbols: (a) Using EKF as shown in this chapter. (b) Using calibration constraints such as in Section 8.2.1. * Method only works for planar objects. ** Non-linear method. ↑, ↓ Result can be derived from neighboring cell.*

be due either to planar structure or to rotational motion. Torr et al. [70] disambiguates between the two types of degeneracies by using the three-dimensional structure of the points, but this works only if the structure has already been obtained.

The Kalman based method can recover motion parameters from pure rotation [8]. Obviously the structure cannot be estimated if all motion in the sequence is purely rotational, but keeping the object still in the middle of an otherwise normal sequence will not destroy the estimates of structure and motion. Planar objects create a situation where the filter is under-determined in each step, making the SfM algorithm less robust and slower to converge. However, it will still converge if the characteristics of the noise, prior and motion are good enough. The fact that the same algorithm can be used for both planar and non-planar objects means that it will also work for near-planar objects. The recursive structure of the Kalman filter is well suited for tracking, where sequences can be very long. The probabilistic formulation also makes it easy to model measurement noise. The non-linearity of the method is a drawback but, as seen in Table 8.4, no linear methods exist in the planar case except for [75], which only works for planar objects in the fully calibrated case. For these reasons, the Kalman based SfM algorithm is the method of choice in this thesis.

# Chapter 9

# Disappearing and Appearing Points

This chapter treats how to cope with disappearing and appearing points. An extension of the SfM algorithm from Chapter 7 is presented that makes it possible to add points that are not visible in the first frame. This work was published as a technical report [67]. Later, Dell'Acqua et al. have addressed the same problem [1]. Their solution is quite different and will be commented on in the end of this section.

## 9.1 Disappearing Points

In this thesis, the SfM algorithm will be used to help the tracking of points on the surface of the head. As the head turns, some points will disappear. The head is likely to turn back at a later stage, and the points will reappear. An example of this is shown in Figure 9.1, where the left side of the user's head disappears (middle) and reappears later (right). Therefore, rather than remov-

Figure 9.1: *Images number* 50, 150 *and* 200 *from an image sequence. Points on the left hand side of the head will first disappear (middle image) and then reappear later (right-most image)*

ing the disappearing points from the SfM algorithm altogether, it is favorable

if they can be 'put on hold' until they reappear. This can be done conveniently in the Kalman framework by assigning an (effectively) infinite variance to the points that are currently occluded. The three-dimensional head model can be used to decide whether or not a point is occluded.

## 9.2 Appearing Points

If the head turns enough, all points will get occluded. It is thus essential that the algorithm is able to include some of the new points that have appeared. For instance, when the head turns sideways, points on the ear can be used. The rest of this chapter will treat how points can be added to the EKF based SfM algorithm.

As described in Section 7.2.2, the function $h_k(\cdot)$ uses the image coordinates in the first frame, $(u^0, v^0)$, to calculate $(X, Y, Z)$, $(X_C, Y_C, Z_C)$ and then the estimate $(\hat{u}, \hat{v})$. The fact that $(u^0, v^0)$ must be known in order to calculate the filter estimates poses a problem when adding feature points at a later stage. If a new feature point becomes visible first at frame $k$, its image plane projection in the original reference frame $(u^0_{new}, v^0_{new})$ is not known.

### 9.2.1 Old Reference Frame

One way to solve the problem is to rotate and translate back the measurements from the $k^{th}$ frame to the $0^{th}$ frame. As can be seen in Figure 9.2 however, the

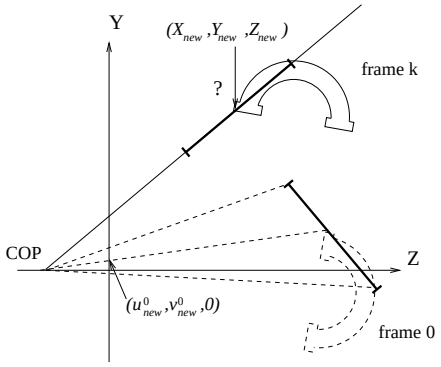



Figure 9.2: *Since the depth of the new point $(X_{new}, Y_{new}, Z_{new})$ is unknown (upper black interval), a large bias in $(u^0_{new}, v^0_{new})$ can be expected.*

lack of knowledge of the depth $\alpha$ will result in a large bias in the estimation of $(u^0_{new}, v^0_{new})$.

### 9.2.2 New Reference Frame

Another solution would be to change the reference frame and restart the Kalman filter at the $k^{th}$ frame. As can be seen in Figure 9.3 a similar problem to that

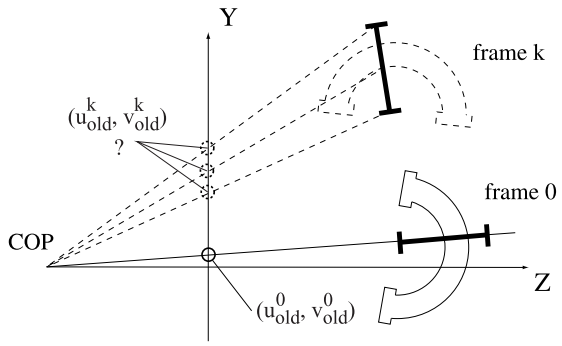



Figure 9.3: *Since the depth of an old point is not known exactly (lower black interval), a bias in the position* $(u^k_{\text{old}}, v^k_{\text{old}})$ *in the new reference frame will occur.*

of Figure 9.2 occurs: For each "old" point, the image coordinates from the first frame $(u^0_{old}, v^0_{old})$ (solid circle) are replaced with estimates $(u^k_{old}, v^k_{old})$ in the $k$th frame (middle dashed circle). The depth $\alpha$ is not known exactly, and this creates a bias in the position of $(u^k_{old}, v^k_{old})$ (other dashed circles). Since the filter has had some time to converge, there is an estimate of $\alpha$, and the bias of $(u^k_{old}, v^k_{old})$ is thus smaller than with the "old reference frame" method of Figure 9.2. On the other hand, bias is added to *all* the old points, compared to only the new points, which are assumed to be fewer.

### 9.2.3 Bias Estimation

Both the "old reference frame"- and the "new reference frame" methods will thus suffer from bias. One solution to this is to include this bias in the state vector and estimate it using the Kalman filter. Bias estimation was proposed in the original paper by Azarbayejani and Pentland [8], but not specifically as a solution to the problem of adding points. Equation (7.4) is then modified to

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} (u^0 + b_u) \\ (v^0 + b_v) \\ 0 \end{pmatrix} + \alpha \begin{pmatrix} \beta(u^0 + b_u) \\ \beta(v^0 + b_v) \\ 1 \end{pmatrix}, \tag{9.1}$$

and $b_u$ and $b_v$ are added to the state vector $\mathbf{x}$. However, introducing these extra degrees of freedom will make the filter less over-determined. In the "old reference frame" case, each new point will subtract one degree of freedom from the surplus created by the parameterization described in Section 7.2.2. In the "new reference frame" case, each of the old points will have the same effect, which is worse,

assuming that they are more numerous. The problem is somewhat alleviated by the fact that the $\alpha$-values are known to a certain extent which means that the corresponding biases will have smaller variances.

### 9.2.4 Two Reference Frames

The method proposed in this thesis is to maintain two reference frames, one for the old points and one for the new ones. As illustrated in Figure 9.3, the old points will continue to be restricted to the line from the COP to $(u^0_{old}, v^0_{old})$,

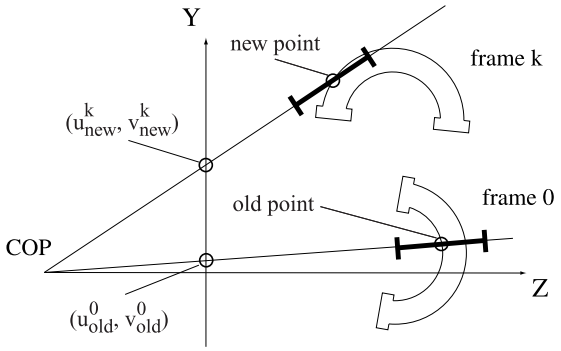



Figure 9.4: *The proposed method — the old points will keep the old reference frame, whereas the new points will use the new reference frame. No bias due to depth will occur.*

whereas then new points will lie on the line from the COP to $(u^k_{new}, v^k_{new})$. The prediction $(\hat{u}, \hat{v})$ for the old points will continue to be calculated using (the projection of) Equation 9.2, whereas the new points will be calculated according to the projection of

$$\begin{bmatrix} \hat{X} \\ \hat{Y} \\ \beta\hat{Z} \end{bmatrix} = BR(q)R_k^T \left( \begin{pmatrix} (1+\alpha\beta)u^k_{new} \\ (1+\alpha\beta)v^k_{new} \\ \alpha \end{pmatrix} - T_k \right) + \begin{bmatrix} t_x \\ t_y \\ \beta t_z \end{bmatrix}, \tag{9.2}$$

where $B = \text{diag}\,[(1, 1, \beta)]$ and $R_k$ and $T_k$ represent the rotation and translation between frame 0 and frame $k$.

This approach is still not bias free — estimation errors in the rotation quaternion and in the translation vector will give rise to a shift in the texture map used for the templates and hence a bias in $(u^k_{new}, v^k_{new})$. However, this problem also occurs with the other methods. Moreover, in the proposed scheme the rotation and translation bias only applies to the new points, as opposed to all the old points in the "new reference frame"-method. As mentioned above, these biases in $u$ and $v$ can be estimated. Alternatively, the motion parameters $Q_k^T$ and $T_k$ can be estimated. This amounts to only 6 extra degrees of freedom for the filter, which is advantageous to estimating $b_u$ and $b_v$ if more than 3 points are added

at once. The implementation of the real system (described in the next chapter), did not seem to suffer from these biases, and the "two reference frames" method could be used without extending the feature vector for bias estimation.

### 9.2.5 Implementational Issues

When new points are added to the extended Kalman filter, the state vector $\mathbf{x}$ must be expanded to include the new $\alpha$-values. Thus all the matrices that make up the Kalman filter must be resized. If fixed-sized matrices are used, this means allocating memory for the new, bigger sized matrices, copying data from the old matrices and then deallocating them. Using variable sized matrices solves this problem, but may result in slower matrix operations.

Implementation-wise, it may instead be easier to include the "new" measurements from the start. By setting the variance of the measurements of these points to (practically) infinite in the beginning, they will not impact the solution of the Kalman filter. Once the new points are included, $(u^k_{new}, v^k_{new})$, $R_k$ and $T_k$ are added to the filter, and the variances are set to normal values. The filter will then start including these measurements in the solution.

### 9.2.6 Related Work

Recently, the problem of adding points to the EKF based SfM algorithm has been investigated by Dell'Acqua et al. [1]. Their solution is to start an independent Kalman filter each time a new point occurs. After collecting data from the entire sequence, a single Kalman filter is stitched together from all the others. Each time a point disappears from the "Master" Kalman filter, all the "Slave" filters that have been created up to that point are examined for replacement candidates. The point that will survive the longest is then used to replace the old point. The "old reference frame" method is used, and the bias of the new point can be reduced since the depth $\alpha$ can be obtained from the "Slave" Kalman filter, that has been converging for a while. The "Master" filter is then continued until a new point disappears, and the procedure is repeated. No attempt is made to reacquire old points — when they reappear they are treated as new, unknown points.

Since the above-mentioned method is of batch type it is not well suited for real-time tracking. It can obviously be modified so that no look-ahead is used, but even so the use of multiple Kalman filters (one filter for each new point) makes it a bit computationally expensive for a real-time scenario. Furthermore, just as in the "new reference frame" method, any remaining error in the estimated depth $\alpha$ will result in some bias. On the other hand, there are advantages to having all points in the same reference system. For instance, the extra matrices for rotation and translation ($R_k$ and $T_k$) of Equation (9.2) are avoided.

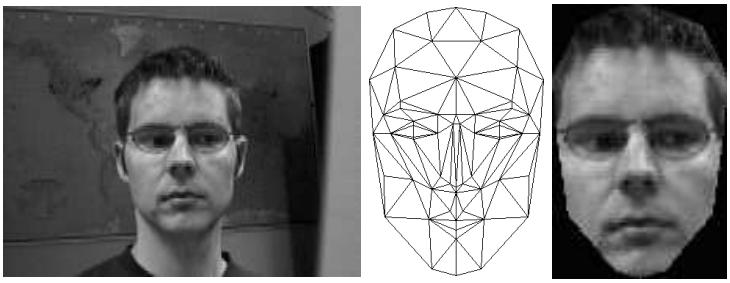
# Chapter 10

# The Tracking System

This chapter will describe the tracking system in detail. First, the initialization procedure is described; how feature points are rated and selected. Then, the matching procedure is treated, followed by a section describing how the variance of the measurements are estimated and fed to the Kalman filter. The chapter is concluded with a description of the adaptive texture update.

## 10.1 Initialization

As mentioned in Chapter 1, this thesis treats the head tracking problem, where it is assumed that the pose of the head is known in the first frame. When implementing a real system, however, this pose must be extracted in some way in order to be able to run the system. In this work, this is done manually. The user is asked to put the head in a specific position, and orient it so that it is in a completely frontal position, as illustrated in Figure 10.1. The head should also occupy a certain area of the screen before tracking is initialized.



(a) (b) (c)

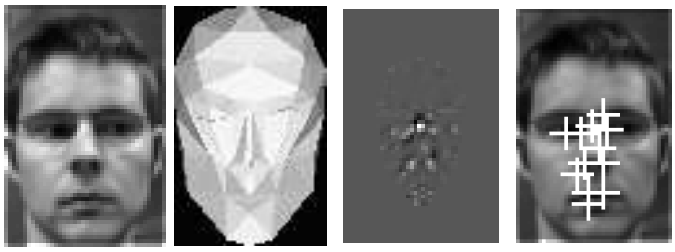
Figure 10.1: *A generic three-dimensional polygonal head model is aligned with a head-on shot of the video sequence, and the corresponding pixels are texture mapped to the surface of the face model.*

### 10.1.1 Head Model

The texture of the face region in the image of Figure 10.1a is texture mapped onto a modified version of the Candide model [54] (Figure 10.1b) to obtain the texture mapped three-dimensional model of Figure 10.1c. The most important task of the three-dimensional model is to deform the tracked patches projectively. If the model is slightly displaced, it is often better to use a smoother model than a very detailed one, even if the latter is more correct [16]. One example is if the head model is accurate, but displaced in the $x$-direction by a distance as large as the width of the nose. Then the depth error in the nose region will cover twice the area of what a model without a nose would give. Therefore, the Candide model has been modified by flattening out the nose. Furthermore, new triangles on the sides of the head have been added in order to allow for texture updates when the head rotates.

### 10.1.2 Selecting Feature Points

After alignment has been performed, the system selects which feature points to use. The idea is to be able to follow a rather dense set of feature points in the face area of the head, but for computational reasons, all pixels in the image cannot be used, and it is important to select only those that are easiest to track. The part of the image in Figure 10.1a containing the face is cropped out, and then lowpass filtered and subsampled once (Figure 10.2a) to avoid selecting features



(a) (b) (c) (d)

Figure 10.2: *(a) The lowpass filtered incoming video. (b) The weighting (the cosine of the angle between the surface normal and the camera direction). (c) The final rating. (d) The first twelve of the 24 extracted feature points.*

that are too vague to be reliably tracked. Denoting the resulting intensity function $I(x, y)$, points where the determinant of the Hessian

$$\det(H) = \begin{vmatrix} I_{xx}(x,y) & I_{xy}(x,y) \\ I_{xy}(x,y) & I_{yy}(x,y) \end{vmatrix} \tag{10.1}$$

is large are of interest. These points are peaks, saddle points and pits in the intensity surface and can be reliably tracked without aperture problems [9, 10]. For each point in the image, the determinant in Formula (10.1) is evaluated and used as a rating of the point. To avoid selecting points on parts of the

face that are perpendicular to the camera, the determinant is weighted with the cosine of the angle between the surface normal and the camera direction. These values can easily be obtained from the computer graphics hardware by rendering a gray-shade version of the three-dimensional model with lighting from the camera direction (Figure 10.2b). The resulting rating of each pixel is shown in the Figure 10.2c, where brighter pixels indicate a higher score. The 24 points with the highest ratings are then selected with a minimum-distance constraint between points. (24 points is the maximum number that the tracker can handle at full frame rate.) In detail, a sorted linked list with the 200 highest-rating pixels are created. The highest scoring pixel is chosen as a feature point, and all elements in the list closer to this point than the minimal distance are removed. This process is repeated 24 times. The fourth image in Figure 10.2 shows the first twelve points selected. Each feature point is then associated with a three-dimensional *model position* on the surface of the head model. The model position will later be used as the center of the template patch. The coordinates of the model position can be obtained by intersecting rays from the camera with the head model. Again, the computer graphics hardware can be used, this time by reading out the value of the depth buffer in the corresponding pixel location, to calculate the depth of the feature point.

### 10.1.3 Initializing the Kalman Filter

The model position of each feature point will not equal the point's true three-dimensional position. However, since the model position is obtained from a head model, it should be a reasonable starting point for further refinement. Hence the depths of the model positions are used as starting values for the $\alpha-$values in the SfM Kalman filter presented in Chapter 7. The three-dimensional position estimated by the Kalman filter is here called the *estimated position*. The image coordinates of the feature points $(u^0, v^0)$ are used to represent the structure according to Equation (7.4). For numerical stability, the coordinates are scaled so that the width of the image equals 1.0 units.

## 10.2 Tracking

As shown in Figure 10.3 (repeated here for the convenience of the reader), the tracking is carried out between the rendered frame and the video input. Both images (or rather, the relevant parts of them) are lowpass filtered and subsampled in order to track larger and more robust features. Since the model positions of the feature points are fixed with respect to the three-dimensional model, their two-dimensional coordinates in the rendered image (bottom left image of Figure 10.3) can be calculated. A $7 \times 7$ pixel patch around each model position is cropped out. This patch will contain the projectively transformed version of the original feature point. The patch is matched with patches from a $11 \times 11$ pixel search window of the incoming video image (top left image of

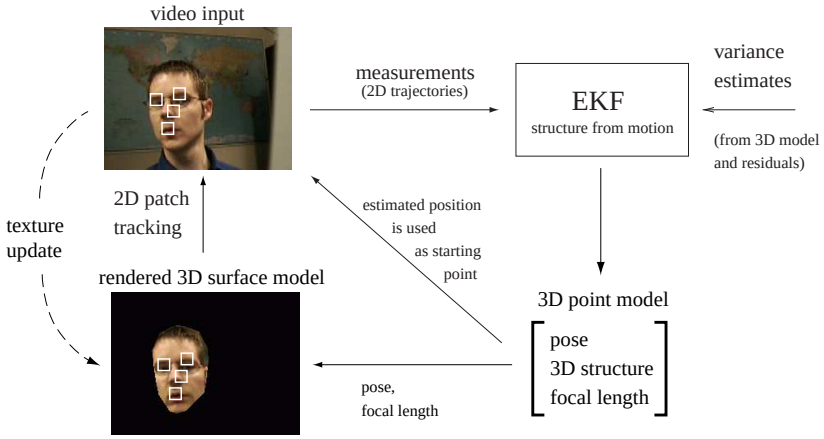



Figure 10.3: *Patches from the rendered image (lower left corner) are matched with the incoming video. The two-dimensional feature point trajectories are fed through the SfM extended Kalman filter that estimates the pose information needed to render the next model view. For clarity, only four patches are shown.*

Figure 10.3). The search window is centered around the estimated position from the SfM algorithm. Normalized correlation is used; if $\mathbf{a}$ and $\mathbf{b}$ are the vectors obtained by raster-scanning the patch in the rendered image and the video input respectively, then the $\mathbf{b}$ that minimizes the angle between them is selected. This is equivalent to maximizing

$$\varrho = \cos(\theta) = \frac{\hat{\mathbf{a}} \cdot \hat{\mathbf{b}}}{\|\hat{\mathbf{a}}\|\|\hat{\mathbf{b}}\|} \tag{10.2}$$

where $\hat{\mathbf{a}} = \mathbf{a} - \mu_{\mathbf{a}}$, $\hat{\mathbf{b}} = \mathbf{b} - \mu_{\mathbf{b}}$. Recall that $\mu_{\mathbf{x}}$ represents the mean of $\mathbf{x}$ and that $\|\cdot\|$ represents the $L_2$ norm. After Equation (10.2) has been used to calculate $\varrho$ for each position in the search window, a Gaussian window is applied to $\varrho$ to favor positions close to the estimated position. An exhaustive search is carried out in the search window and the candidate with the lowest error is selected.

### 10.2.1 Subpixel Refinement

Since the two images were subsampled before matching, the accuracy of the tracking is only $\pm 1$ pixel. However, since an exhaustive search is performed in the search window, the correlation is known in the adjacent positions. By approximating derivatives with central differences, the correlation error surface is approximated by a second degree Taylor polynomial

$$\varrho(\Delta\mathbf{x}) \approx \varrho_0 + \mathbf{c}^T \Delta\mathbf{x} + \Delta\mathbf{x}^T H \Delta\mathbf{x}. \tag{10.3}$$

If the resulting matrix $H$ is negative semidefinite, the (sub-pixel) location of the maximum of the resulting paraboloid is calculated as

$$\Delta \mathbf{x} = -H^{-1}\mathbf{c}, \tag{10.4}$$

and the refined value can then be used as the feature location. If the error surface is very irregular however, the maximum proposed by Equation (10.4) can be outside the $2 \times 2$ pixel area. In this case the Taylor polynomial is a poor approximation of the error surface and the centroid of the $2 \times 2$ pixel area is used instead.

## 10.3 Estimation of the Covariance $R_k$

As described in Appendix C, each measurement vector $\bar{\mathbf{z}}_k$ in a Kalman filter is associated with a covariance matrix $R_k$. Traditionally, this covariance matrix (as well as $Q_k$, and $P_0$) is assumed to be known a priori. If the values of $R_k$ varies from time step to time step, and if these variations can be estimated, then filtering performance can be increased. This is exactly the case in tracking; some points will be occluded, and their position measurements will therefore have large errors — the corresponding elements in $R_k$ will be big. This variation in $R_k$ can be estimated, e.g., by measuring the correlation coefficient $\varrho$; a low $\varrho$ indicates a high $R_k$ element and vice versa.

In the case of template matching, there are basically two types of matching errors. Either the correct match is found, offset only slightly because of small changes in lighting, small errors in the projective transformation of the patch, etc. This is called the *small error case* in this thesis. The other possibility is that the match is completely wrong, for instance due to occlusion or to that the normal of the feature point is almost perpendicular to the camera direction. This is named the *large error case*. In one dimension, the displacement of the

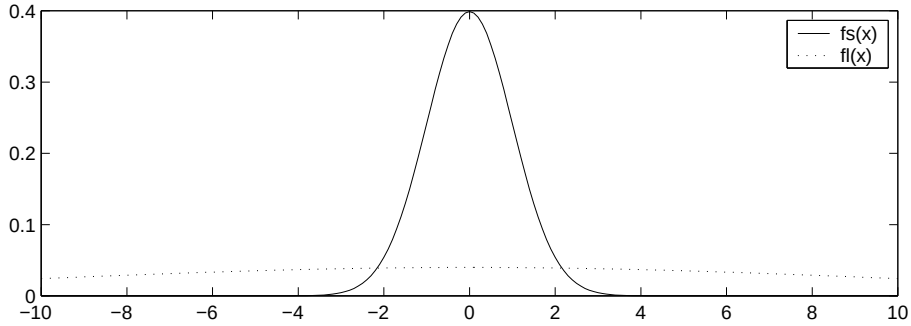



Figure 10.4: *Solid: The probability density function of the matching displacement for the small error case, $f_{X_s}(x)$. Dotted: Ditto for the large error case, $f_{X_l}(x)$.*

measured position to the real one can be distributed as in Figure 10.4. The solid

line shows the probability density function (pdf) in the small error case, $f_{X_s}(x)$, a Gaussian with a standard deviation of one pixel. The dotted line shows ditto for the large error case, $f_{X_l}(x)$, a Gaussian with a standard deviation of ten pixels. Since the error in the $x-$ direction is often correlated with the one in the $y-$ direction, it is advantageous to model the noise for each point as a two-dimensional Gaussian. Instead of just selecting a standard deviation such as in the one-dimensional case, a two-dimensional covariance matrix $\Sigma$ is estimated. The following sections will treat how this is done.

The suitable matrix for the small error case, $\Sigma_s$, and that for the large error case, $\Sigma_l$, are discussed separately. Then a section will follow on how to fuse these to a matrix $\Sigma$ that is used to construct $R_k$, the covariance matrix of the measurement vector.

### 10.3.1 Small Error Case

Jebara and Pentland [35] use the residuals of the correlation-based trackers to estimate the covariance matrix $\Sigma_s$. In the initialization phase, the correlation trackers are perturbed, and the corresponding square root of the sum squared error ($\sqrt{\text{SSD}}$) surface is approximated with a polynomial equation

$$\sqrt{SSD} = \Delta\mathbf{X}^T A \Delta\mathbf{X}. \tag{10.5}$$

Since two points per patch are tracked (cf. Section 6.3), $\Delta\mathbf{X}$ will be four-dimensional, $\Delta\mathbf{X} = (\Delta X_1, \Delta Y_1, \Delta X_2, \Delta Y_2)$. Jebara and Pentland point out that, by making the covariance matrix of $\Delta\mathbf{X}$ proportional to $A^{-1}$, all points $\Delta\mathbf{X}$ that contribute to a certain error $\sqrt{SSD}$ will get the same probability. In other words, the iso-residual ellipsoids will correspond to iso-probability ellipsoids.

Presuming that $\Delta\mathbf{X} \in N(0, \xi A^{-1})$, the measured error residual $\sqrt{SSD}$ can be used to obtain an unbiased Maximum Likelihood estimation of $\xi$, $\xi^* = \frac{1}{4}\sqrt{SSD}$ (cf. Appendix D). Thus

$$\Sigma_s = \frac{1}{4}\sqrt{SSD}\, A^{-1}, \tag{10.6}$$

which is consistent with the formula of Jebara and Pentland

$$\Sigma_s \sim \sqrt{SSD}\, A^{-1}. \tag{10.7}$$

In the proposed tracking system, only one point per patch is tracked, and a similar formulation is possible for this case. The resulting two-dimensional covariance matrix is

$$\Sigma_s = \frac{1}{2}\sqrt{SSD}\, A^{-1}, \tag{10.8}$$

where the only difference is in the estimate of $\xi$, $\xi^* = \frac{1}{2}$. A problem in the two-dimensional case is that Equation (10.8) cannot account for in-plane rotation: For instance, if the patch contains a feature that is elongated in the $x-$direction,

Equation (10.8) will model the error correctly in the beginning; $\Sigma_s$ will be uncertain in the $x-$direction. However, after a 90° in-plane rotation, the feature is now elongated in the $y-$direction, while Equation (10.8) still predicts bigger uncertainty in the $x-$direction. A better estimate of the pdf of the error can be found using the error surface of the correlation matching, $\varrho(\Delta\mathbf{x})$, and its Taylor expansion from Equation (10.3). Using

$$\Sigma_s \sim (-H)^{-1}, \tag{10.9}$$

where $H$ is the Hessian of Equation (10.3), a noise model is obtained that better reflects the actual shape of the error. In Figure 10.5, an error surface is shown, where brighter pixels represents better matches. Superimposed is the ellipse

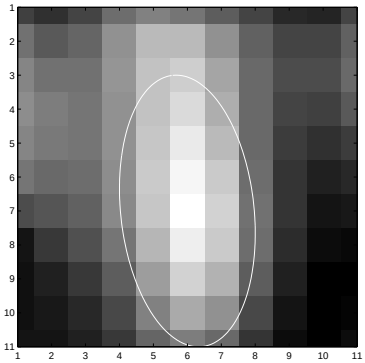



Figure 10.5: *Background: The error surface $\varrho(\mathbf{x})$ of the correlation. Foreground: The ellipse $\mathbf{x}^T(-H)\mathbf{x} = C$ from the Taylor expansion of $\varrho(\mathbf{x})$ around its maximum (Equation (10.3)).*

corresponding to the quadratic form $\mathbf{x}^T(-H)\mathbf{x} = C$. This ellipse represents both constant error and constant probability when Equation (10.9) is used. Loosely speaking, the estimated $\Sigma_s$ should have the same shape as $(-H)^{-1}$, but not necessarily the same magnitude. More precisely, the ellipse formed by $\mathbf{x}^T\Sigma_s^{-1}\mathbf{x} = 1$ should have the same orientation and relative thickness as the one obtained by $\mathbf{x}^T(-H)^{-1}\mathbf{x} = 1$, but not necessarily cover the same area.

### 10.3.2 The Shape of $\Sigma_s$

If $(-H)^{-1}$ is symmetric and positive definite (which it should if a proper maximum has been found), it can be diagonalized into $(-H)^{-1} = RDR^T$ [60], where $R$ is a rotation matrix and $D$ is a diagonal matrix with positive elements. The radii $(a, b)$ of the ellipse $\mathbf{x}^T(-H)\mathbf{x} = 1$ will then correspond to the square roots of the elements in $D$; $a = \sqrt{d_{11}}$, $b = \sqrt{d_{22}}$. If the rotation angle and the thickness of the ellipse $\mathbf{x}^T\Sigma_s^{-1}\mathbf{x} = 1$ should be the same as for $\mathbf{x}^T(-H)\mathbf{x} = 1$, this means that the rotation matrix $R$ and the ratio of the radii $a/b$ should remain

the same. Hence, $\Sigma_s$ can be written

$$\Sigma_s = R \begin{pmatrix} ca & 0 \\ 0 & cb \end{pmatrix}^2 R^T, \tag{10.10}$$

where $c$ is a suitable constant giving the desired magnitude of $\Sigma_s$.

### 10.3.3 The Magnitude of $\Sigma_s$

In the small error case, the assumption will be made that all points are of similar quality, i.e., their deviation from the true positions will be similar. Thus it is natural to give all points a pdf of equal uncertainty, or *entropy*. In his famous 1948 paper [56], Shannon defines entropy and shows that the entropy of a $d-$dimensional gaussian random variable $X$ with covariance matrix $\Sigma$ equals

$$H(X) = \ln\left((2\pi e)^{\frac{d}{2}} |\Sigma|^{\frac{1}{2}}\right). \tag{10.11}$$

For a fixed dimension $d$, the entropy thus depends only on the determinant $|\Sigma|$.

The measure $H(X)$ has two intuitively pleasant properties. First, since a rotation matrix has determinant one, it is rotationally invariant: $\det(R\Sigma R^T) = \det(R)\det(\Sigma)\det(R^T) = \det(\Sigma)$, see Figure 10.6. Second, fixing $H(X)$ will

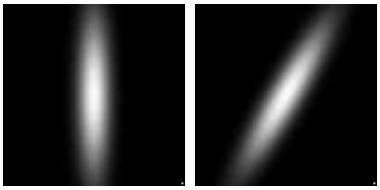


Figure 10.6: *The two Gaussians above with covariance matrices $\Sigma$ (left) and $R\Sigma R^T$ (right) have equal entropy since $|R\Sigma R^T| = |R||\Sigma||R^T| = |\Sigma|$.*

avoid a singular $\Sigma$ which would otherwise lock the Kalman filter solution to a subspace in an unnatural way (see Section C.3). The constant $c$ from Equation (10.10) is selected so that the entropy is equal to that of a Gaussian with $\Sigma_{ref} = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix}$, with a standard deviation of $\sigma = 4$ pixels. The determinant of $\Sigma_s$ in Equation (10.10) is

$$|\Sigma_s| = \left| R \begin{pmatrix} ca & 0 \\ 0 & cb \end{pmatrix}^2 R^T \right| = \left| \begin{pmatrix} ca & 0 \\ 0 & cb \end{pmatrix} \right|^2 = (cacb)^2 = c^4(ab)^2. \tag{10.12}$$

Setting this equal to $|\Sigma_{ref}| = \sigma^4$ gives

$$c = \left(\frac{\sigma^4}{(ab)^2}\right)^{\frac{1}{4}}. \tag{10.13}$$

A check is made so that $cb$ is larger than $\sigma_{min} = \frac{\sigma}{4}$; if not it is set to $\sigma_{min}$ ($a$ is assumed to be larger than $b$). The standard deviation in any direction can thus never be smaller than one pixel. This avoids the situation where the Kalman filter is erroneously locked to a subspace of the solution space (see Section C.3).

In summary, the covariance matrix $\Sigma_s$ is estimated as follows:

1. The matrix $(-H)^{-1}$ is calculated and checked so that it is positive definite
2. Singular value decomposition is used to decompose $(-H)^{-1}$ into $RDR^T$
3. The radii $a \geq b$ are obtained from $D$ and Equation (10.13) is used to calculate $c$.
4. $cb$ is set to $\min(cb, \sigma/4)$, $ca$ is set to $\max(ca, 4\sigma)$
5. $\Sigma_s$ is set to $RD'R^T$, where $D' = \text{diag}\left[(ca)^2, (cb)^2\right]$.

The attentive reader might object that, compared to the above-mentioned method, it is less computationally demanding to use the seemingly equivalent formula

$$\Sigma_s = \frac{|\Sigma_{ref}|^{\frac{1}{2}}}{|-H^{-1}|^{\frac{1}{2}}}(-H^{-1}). \tag{10.14}$$

In this formulation, however, the check of item 4 cannot be performed.

An example of estimation of $\Sigma_s$ can be seen in Figure 10.7, where the radii

Figure 10.7: *Resulting estimation of $\Sigma_s$. The radii of the ellipses $\mathbf{x}^T\Sigma_s\mathbf{x} = 1$ are shown, where long radii correspond to large uncertainties.*

$ca$ and $cb$ have been plotted. Just as expected, the uncertainty is larger along elongated features such as the mouth and the side of the nose.

If $(-H)^{-1}$ is singular or not positive definite, no attempt to extract aperture information from it is made. Instead the uniform covariance matrix

$$\Sigma_s = \begin{pmatrix} \sigma^2 & 0 \\ 0 & \sigma^2 \end{pmatrix} \tag{10.15}$$

is used.

### 10.3.4 Large Error Case

In the large error case, the assumption is that the true position is somewhere in the search area region. A standard deviation of $10\sigma = 40$ pixels is used, which is roughly in the same order of magnitude as the $22 \times 22$ pixel search window. Since the best match is assumed to be in the wrong position, the local Taylor expansion is not valid and no attempt is made to shape the noise. Hence, the covariance matrix

$$\Sigma_l = \begin{pmatrix} 100\sigma^2 & 0 \\ 0 & 100\sigma^2 \end{pmatrix} \tag{10.16}$$

is used for the large error case.

### 10.3.5 Calculating $\Sigma$

Given a point, the next problem is to select a proper covariance matrix given the decision parameters; the correlation error, the visibility of the triangle and its angle to the camera. There are two possibilities; either a *hard* decision is made, selecting either $\Sigma_s$ or $\Sigma_l$, or a *soft* decision is made, using a combination of $\Sigma_s$ and $\Sigma_l$. The hard decision is the simpler one, but has the drawback that a small change in the decision parameters could make the covariance jump instantly from $\Sigma_s$ to $\Sigma_l$, generating a discontinuity in the tracked parameters. The softer version does not have this problem, but requires the design of a reasonable function for combining $\Sigma_s$ and $\Sigma_l$. The following sections will go through both the hard and the soft version. Before that, a model will be presented that is valid for both types of decisions.

### 10.3.6 Confidence Value Model

The problem of deciding which $\Sigma$ to use can be reformulated to estimating a "confidence value", i.e., a probability $p_s$ that the matching belongs to the small error case. More specifically, $p_s$ is the probability of the match belonging to the small error case *given* the value of the decision parameters; $p_s = p(small \mid \varrho, \theta, V)$, where $\varrho$ is the correlation value, $\theta$ the viewing angle of the point's triangle and $V \in \{0, 1\}$ determines visibility. Consequently, the matching belongs to the large error case with probability $(1-p_s)$. The combined pdf of the displacement error will be

$$f_X(x) = p_s f_{X_s}(x) + (1 - p_s) f_{X_l}(x). \tag{10.17}$$

In the hard case, $p_s$ is trivially estimated as being either 0 or 1, whereas in the soft case $p_s \in [0, 1]$. In Figure 10.8 $f_X(x)$ is shown in the one-dimensional case for $p_s = 0.5$. Note that, even though both $f_{X_s}(x)$ and $f_{X_l}(x)$ are Gaussian, the resulting $f_X(x)$ is not; the tails go to zero much slower than for a Gaussian. However, it is still possible to calculate its variance. Since $E[X]$, $E[X_s]$, and

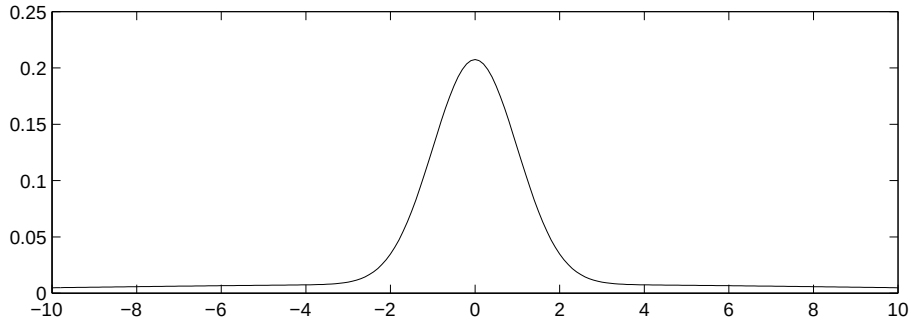



Figure 10.8: *The pdf of Equation (10.17) with equal probabilities of choosing $f_{X_s}(x)$ (with standard deviation $\sigma = 1$) and $f_{X_l}(x)$ (with standard deviation $\sigma = 10$).*

$E[X_l]$ are assumed to be zero, the variance is

$$\begin{aligned}\sigma_X^2 &= \int x^2 f_X(x)\, dx = \\ &= \int x^2 p_s\, f_{X_s}(x)\, dx + \int x^2 (1-p_s) f_{X_l}(x)\, dx = \\ &= p_s\, \sigma_{X_s}^2 + (1-p_s)\, \sigma_{X_l}^2.\end{aligned} \tag{10.18}$$

A similar formula is valid in the two-dimensional case,

$$\Sigma = p_s\, \Sigma_s + (1-p_s)\, \Sigma_l. \tag{10.19}$$

### 10.3.7 Hard Decision

In the case of a hard decision, it is only a question of selecting between the two possibilities $p_s = 0$ and $p_s = 1$. To simplify the problem, the value of $p_s$ is estimated independently for the three decision parameters $\varrho$, $\theta$ and $V$; $p(\text{small} \,|\, \varrho), p(\text{small} \,|\, \theta)$ and $p(\text{small} \,|\, V)$. The minimum of the three probabilities is used;

$$p_s = \min(p(\text{small} \,|\, \varrho), p(\text{small} \,|\, \theta), p(\text{small} \,|\, V)). \tag{10.20}$$

Starting with the correlation value, Bayes' rule gives

$$\begin{aligned}p(\text{small} \,|\, \varrho) &= \frac{f(\varrho \,|\, \text{small}) p(\text{small})}{f_\varrho(\varrho)} \\ p(\text{large} \,|\, \varrho) &= \frac{f(\varrho \,|\, \text{large}) p(\text{large})}{f_\varrho(\varrho)}\end{aligned} \tag{10.21}$$

The small error model should be selected if $p(\text{small} \,|\, \varrho) > p(\text{large} \,|\, \varrho)$, which is equivalent to

$$f(\varrho \,|\, \text{small}) p(\text{small}) > f(\varrho \,|\, \text{large}) p(\text{large}) \tag{10.22}$$

To be able to use the above formula, it is necessary to know $p(\text{small})$ and $p(\text{large})$. However, these two probabilities vary a lot during tracking. For instance, when the head is frontal, $p(\text{small})$ is almost one, whereas when the head is rotated far from the initialization pose, $p(\text{large})$ might be the bigger one. Assuming they are equally probable, the decision rule is simplified to

$$f(\varrho \,|\, \text{small}) > f(\varrho \,|\, \text{large}). \tag{10.23}$$

To estimate $f(\varrho \,|\, \text{small})$ and $f(\varrho \,|\, \text{large})$, the following experiment was conducted. The tracker was allowed to run, and the user's head was rotated to a critical pose where the tracked point deviated from its correct position. The correlation value was measured for a number of frames, each measurement being an example of $f(\varrho \,|\, \text{large})$. Next, the head was moved until the feature point just moved back to the correct position. The correlation value was then measured continuously during a rotation from this critical pose to a frontal pose, in order to get correlation values from all types of poses. These correlation values were used as examples of $f(\varrho \,|\, \text{small})$. The procedure was repeated for all points, and $f(\varrho \,|\, \text{large})$ and $f(\varrho \,|\, \text{small})$ were estimated using normalized histograms. The same number of measurements was used for each point. The result is shown in Figure 10.9. The diagram shows that, using the rule in (10.23), the decision

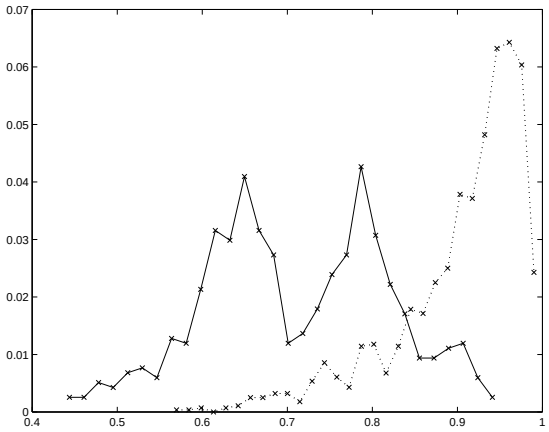



Figure 10.9: *Estimated* $f(\varrho \,|\, \text{small})$ *(dotted) and* $f(\varrho \,|\, \text{large})$ *(solid).*

boundary should be put at around $\varrho = 0.8$. Thus

$$p(\text{small} \,|\, \varrho) = \begin{cases} 1, & \varrho \geq 0.8 \\ 0, & \varrho < 0.8. \end{cases} \tag{10.24}$$

For the angle $\theta$ between the triangle normal and the viewing direction, a decision boundary of $\cos(\theta) = 0.2$ was found:

$$p(\text{small} \,|\, \theta) = \begin{cases} 1, & \cos(\theta) \geq 0.2 \\ 0, & \cos(\theta) < 0.2. \end{cases} \tag{10.25}$$

If the triangle with which the model position of the feature point is associated is facing away from the camera, $\cos(\theta)$ is smaller than zero and the above rule will make sure that the large error case is selected. However, due to self occlusion, the triangle might not be visible even if $\cos(\theta) > 0$. For this reason, a check is performed to ensure that the triangle is visible. In the implementation this is done by rendering the head model a second time, drawing every triangle with a different color. For each triangle containing a feature point, the pixel of the triangle centroid is read back. If the color that is read back differs from the one used to draw the feature point triangle, the feature point must be occluded ($V = 0$), else ($V = 1$). Hence,

$$p(\text{small} \,|\, V) = V. \tag{10.26}$$

$p_s$ can now be calculated using Equation (10.20).

### 10.3.8 Soft Decision

As mentioned above, one drawback with the hard decision method is that small changes in the decision parameters $\varrho$, $\theta$ and $V$ may cause large changes in the estimated covariance $\Sigma$ and therefore in the tracked sequence. It is possible that the tracker can get into a state where a small movement changes $\Sigma$ from $\Sigma_l$ to $\Sigma_s$ for a certain feature point, and that this will change the estimated pose of the head so that, in the next frame, $\Sigma$ will move back to $\Sigma_l$, creating a jerky oscillation of the estimated head pose.

To investigate whether this is the case, the tracking was set up with two covariance estimation rules that the user could easily toggle between. The first was a simpler version of the hard decision, with $p_s^1 = p(\text{small} \,|\, \varrho)$. The second was a $C^0$-continuous version of this,

$$p_s^2 = \begin{cases} 0, & \varrho < 0.75 \\ \frac{\varrho - 0.75}{0.85 - 0.75}, & 0.75 \leq \varrho < 0.85 \\ 1, & \varrho \geq 0.85. \end{cases} \tag{10.27}$$

The two functions can be seen in Figure 10.10. While changing between $p_s^1$ and $p_s^2$ during tracking, no discernable difference in the tracking was found. This led to the conclusion that the discontinuity of $p_s^1$ is not devastating to the tracking, and a hard decision can thus be used. The soft decision rule certainly has the potential of performing better than the hard one, but involves more degrees of freedom and is thus harder to design. Rather than inventing ad hoc functions mapping $(\varrho, \theta, V)$ to $p_s$ that could even deteriorate performance, the simpler hard decision rule is used.

In summary, for each feature point $i$, the covariance matrices $\Sigma_s^i$ and $\Sigma_l^i$ are calculated, and a confidence value $p_s^i \in \{0, 1\}$ is found. A covariance matrix $\Sigma^i$ is then selected as either $\Sigma_s^i$ (if $p_s^i = 1$) or $\Sigma_l^i$ (if $p_s^i = 0$). The measurement covariance matrix $R_k$ of the Kalman filter is then set to the block matrix $R_k = \text{diag}(\Sigma^1, \Sigma^2, \ldots, \Sigma^N)$.

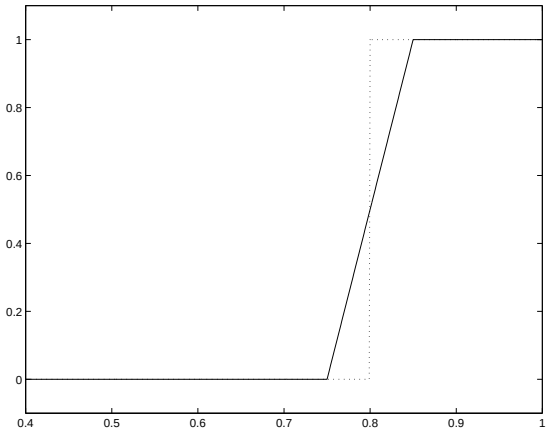



Figure 10.10: *Dotted: The hard decision rule $p_s^1$ as a function of $\varrho$. Solid: The $C^0$ continuous decision rule $p_s^2$.*

## 10.4 Texture Update

Since the initialization procedure is performed on a head-on snapshot of the head, all the feature points will be situated in the frontal face area. Thus, at large out-of-plane rotations, few or no feature points will be visible in the image and the system will inevitably lose track. Tracking points at the side of the head would solve this problem. However, since the texture is acquired from the first frame during the initialization procedure, parts of the head that are not visible in a head-on shot will not get an accurate texture. Therefore the system automatically extracts new texture when the head has rotated enough. More specifically, the texture for the entire side of the face will be acquired from the video when the scalar product between the camera direction and the triangle normal is greater than a certain constant value. Figure 10.11 illustrates this; the first image is the head-on shot used for initialization. The second image is

Figure 10.11: *From left: head-on shot used for initialization, first tracked frame, frame just before texture update, frame just after texture update. The small area in last image shows where the system looks for new feature points.*

the texture map acquired at the first frame. The third and the fourth image is the model just before and after the acquisition of the new texture on the side. The new feature points on the acquired texture are obtained the same way as

described in Section 10.1, with one difference. Since it is undesirable to select points that are too close to the head boundary, or to the discontinuous "seam" between the two textures, only a smaller area of the texture is searched for feature points. The small area in the last image in Figure 10.11 shows where the system looks for new feature points. The technique proposed in Chapter 9 is then used to incorporate the new points into the Kalman filter.

# Chapter 11

# Reinitialization

This chapter investigates how to reinitialize a model-based head tracker after tracking failure. The information gathered in the original initialization is used to obtain a quick and reliable reinitialization. The color information of the model's texture map is used to obtain a skin color model that is per definition adapted to the lighting, the characteristics of the camera and the specific skin tone of the user. The search for the largest skin colored blob is then further refined by template matching using a part of the face texture between the eyes. From this refined position the tracker is restarted. A method for automatic detection of tracking failure is also presented.

## 11.1 Background

With any tracking system, there is always a risk of tracking failures. In fact, the word tracking itself indicates that the system is trying to *follow* something rather than *finding* it. Of course, it would be possible for a system to start from scratch in each image, regardless of the result from the previous image. Such a system would not be able to "lose track", and would thus be more robust in that sense. However, it would also have to search the entire solution space each time, instead of just investigating a small part of it around the previous solution. That is slower than doing tracking, and therefore harder to justify in a real-time system. Thus tracking failures will always have to be expected. In many applications, a tracking failure can be tolerated if it is detected and followed by a swift and reliable reinitialization procedure. For instance, if the tracker output is used for controlling a human computer interface, then a tracking failure may not be severe: When the computer operator is turned away from the screen, he/she does not need to control the computer. However, when the user returns to the screen, the tracker should start working again. In a model-based coder scenario it is less acceptable to have tracking failures, but even here it is more important that the tracker works when the user is facing the camera than when she/he is facing away. Thus, combining a tracker that sometimes fails with a

good reinitialization procedure can result in a system that is almost as useful as a system that provides continuous tracking.

Prior work in reinitialization includes that of Crowley and Berard [19]. They use three visual processes for tracking and continuous reinitialization of a 2D-based head tracker; a skin-color blob model, a correlation tracker and a blink detector. Jebara and Pentland [35] detect tracking failure by normalizing the face texture back to frontal position and measuring the "Distance From Face Space" (DFFS). When the DFFS is larger than a threshold, a face detection process restarts the tracker from scratch, disregarding information gathered so far. The most similar approach to reinitialization is proposed by Matsumoto and Zelinsky [46]. They use the correlation values from the tracking to detect failure, and a coarse-to-fine template matching step of the entire face in order to find a good restarting position. Their face tracking system is using stereo camera input, in contrast to monocular camera input as used in this thesis.

The key point of this chapter is that robustness and speed can be gained by having a special reinitialization procedure, different from the original initialization procedure. The basic idea is that a lot of information that was gathered in the initialization stage can be reused to help the reinitialization. For example, the texture map of the face that was obtained during initialization can be used to build an accurate skin color model as well as providing templates for template matching.

The next section will define the term convergence zone, and discuss typical failures that occur. The section that follows will go through the proposed reinitialization scheme; failure detection, skin color processing, template matching and restarting. The chapter will be concluded by results and a discussion on possible improvements.

## 11.2 Tracking Failure

### 11.2.1 Convergence Zone

The convergence zone of a tracking system for a certain image in the sequence is here defined to be the set of all poses (translation and rotation) of the model that will converge to the true pose of the head in that image, if the tracker is allowed to run. The tracker works by starting inside the convergence zone in the first frame, and as the pose changes in each frame (caused by the user's movements), the tracker will converge to the true pose. As long as the model pose that is estimated in the previous frame is within the convergence zone of the current frame, the tracking will work. It should be noted that, in this definition,

the size of the convergence zone will be different for each pose. Moreover, it will generally be smaller in the beginning of the sequence when the $\alpha-$ and $\beta-$ values have not had time to converge, compared to later when these values are correctly estimated.

### 11.2.2 Typical Failures

There are three typical cases when a tracker fails. The first is if the user moves too quickly from one frame to the next. Then the pose of the model will be outside the convergence zone, and the tracking will fail. The second case is when the user's head is rotated far away from the original (frontal) position, such as in Figure 12.4c. Many of the easily trackable points (around the eye and nose) are then facing away from the camera, and errors in the three-dimensional modeling are also becoming more visible. Both of these factors make the convergence zone shrink substantially, and tracking can be lost even at slow pose velocities. The third case is when an obstacle such as an arm completely occludes the face — the convergence zone is then virtually zero. A scenario containing all three of the typical cases is when the user turns the head quickly 180 degrees towards someone standing behind the computer, shown in Figure 12.2d.

## 11.3 Reinitialization

### 11.3.1 Failure Detection

The tracking procedure described in Section 10.2 produces a correlation value, $\varrho$, for each point. A $\varrho$ close to 1 indicates an accurate match, whereas a low $\varrho$ means that the measurement is insecure. If the tracking has failed, it is unlikely that any point will produce a high $\varrho$. A tracking-error is declared if the best $\varrho$ is smaller than a threshold value for the entire duration of ten frames. The resulting detector is not water proof, but since the $\varrho$ measurements are a bi-product of the tracking, it is virtually cost-free. It also works reasonably well in practice.

### 11.3.2 Skin Color Processing

Once a tracking failure is declared, the reinitialization process starts with building a skin color model. This is a widely used method for finding faces and hands, introduced by Crowley and Berard [19] and by Olivier, Pentland and Berard [50].

For a pixel with color $\mathbf{x} = (Y, U, V)$, let $p(\text{skin} \mid \mathbf{x})$ be the probability that it is a skin pixel, and similarly $p(\text{not skin} \mid \mathbf{x})$ the probability that it is not. A pixel will be labeled as a skin pixel if

$$p(\text{skin} \mid \mathbf{x}) > p(\text{not skin} \mid \mathbf{x}). \tag{11.1}$$

Using Bayes rule,

$$\begin{aligned} p(\text{skin} \mid \mathbf{x}) &= \frac{f(\mathbf{x} \mid \text{skin}) p(\text{skin})}{f_{\mathbf{x}}(\mathbf{x})} \\ p(\text{not skin} \mid \mathbf{x}) &= \frac{f(\mathbf{x} \mid \text{not skin}) p(\text{not skin})}{f_{\mathbf{x}}(\mathbf{x})}, \end{aligned} \tag{11.2}$$

the inequality in (11.1) becomes

$$f(\mathbf{x} \mid \text{skin}) p(\text{skin}) > f(\mathbf{x} \mid \text{not skin}) p(\text{not skin}). \tag{11.3}$$

Since the lighting, camera and the skin tone of the person can be assumed to be the same as when this texture was acquired, a simple Gaussian in the YUV color space suffices to approximate the skin probability density function $f(\mathbf{x} \mid \text{skin})$. Statistics is gathered from points in the original texture (Figure 11.1a), using the image shown in Figure 11.1b to mask all but the forehead and the cheeks. In detail, the covariance matrix $\Sigma$ and the mean vector $\mu$ of the gaussian pdf

$$f(\mathbf{x} \mid \text{skin}) = \frac{1}{(2\pi)^{\frac{3}{2}} \sqrt{|\Sigma|}} \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu)) \tag{11.4}$$

is estimated. The probability density function for non-skin pixels is assumed to be uniform, i.e., all colors are assumed to be equally probable. The color coefficients delivered by the image grabbing hardware are in headroom range, i.e., $Y \in [16, 235]$ and $U, V \in [16, 240]$. This means that

$$f(\mathbf{x} \mid \text{not skin}) = \frac{1}{(234 - 16)(240 - 16)^2}. \tag{11.5}$$

Finally, if the head is assumed to occupy about 5% of the screen, $p(\text{skin}) = \frac{1}{20}$ and $p(\text{not skin}) = \frac{19}{20}$. Equation (11.3) can thus be rewritten as

$$f(\mathbf{x} \mid \text{skin}) > \frac{p(\text{not skin})}{p(\text{skin})} f(\mathbf{x} \mid \text{not skin}) \approx 2 \cdot 10^{-6}. \tag{11.6}$$

The input image (Figure 11.1c) is subsampled and $f(\mathbf{x} \mid \text{skin})$ is calculated and thresholded for each pixel. A connected components processing step is performed on the binary image, and the largest connected skin color blob is assumed to be the face. This is illustrated in Figure 11.1d.

### 11.3.3 Template Matching Refinement

Although the color blob obtained in the previous section is a robust way to locate the face, it is not accurate enough. For instance, it is not easy to know where the face ends and the neck starts. Furthermore, the dark features of the eyes sometimes divide the face in two blobs, one containing the forehead and one containing the rest of the face. The connected component processing step will then choose the lower part of the face as the largest connected blob, and

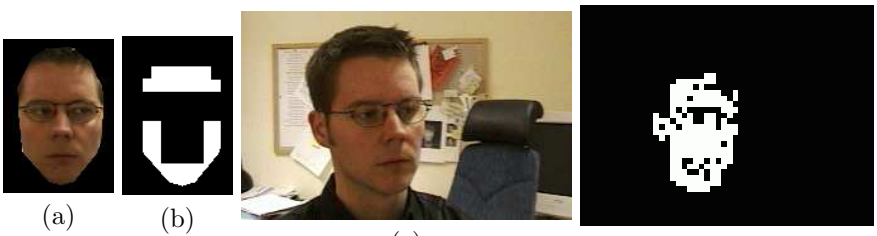



Figure 11.1: *(a) Original texture (b) mask (c) input image (d) color blob*


Figure 11.2: *(a) Between-eyes template, (b) correlation surface, (c) result, (d) model position*

this will result in a large error in the vertical direction. To refine this position, a $13 \times 13$ template (Figure 11.2a) from the area between the eyes is cropped out from a subsampled version of the original texture. The bounding box of the color blob is used as a search window for the template. Exhaustive search using normalized correlation is performed in the search window, and the location of the maximum of the correlation (shown in Figure 11.2b) is used. Figure 11.2c shows a result of the template matching. Since the texture between the eyes is situated in the middle of the face, it is visible even for relatively large out-of-plane rotations. Empirically, it also works for comparatively large changes in scale.

### 11.3.4 Restarting the Tracking

The $x-$ and $y-$ image coordinates for the "between-eyes" patch obtained above will not suffice to resolve the unknown six degrees of freedom that constitute the head pose. However, if the model is placed within the convergence zone, the tracker will resolve the remaining unknowns. Thus it is important that the model is positioned in a pose that is as plausible as possible. Assuming that the head is approximately as big on the screen as it was during the original initialization, the model is simply translated in the $x-$ and $y-$domain, using the same $z$ as in the original initialization. The rotation is zeroed so that the face is head on. Figure 11.3 shows an example of this. The left-most image shows the video input at the time of reinitialization. The tracker is restarted in the position shown in the second image. After five iterations (about 0.2 s) the tracker has converged to the correct pose.

Figure 11.3: *Tracker convergence at restart.*

## 11.4 Conclusions and Future Work

This chapter has shown how a model-based tracker can be reinitialized after tracking failure. Information from the original initialization is used to make the process fast and robust. Due to this information, it is possible to use rather simple techniques that would otherwise not work as well, such as template matching and skin color blob extraction. However, improvements are possible in a number of areas. The failure detection method that is used is simple and cost free but (1) sometimes creates false alarm if the head is rotated too much and (2) sometimes fails to detect a tracking failure if the translation is correct but the rotation is wrong. Type (2) errors can be avoided by adding a DFFS type criterion as proposed by Jebara and Pentland [35], whereas type (1) errors are harder to handle. In general, it would be good to detect failure with several methods rather than relying on a single one. One possibility is to use the skin color blob as an additional estimate of the head pose; if it differs significantly from the estimate of the Kalman filter, the tracker could be reinitialized.

As shown in Figure 11.3, the system can reinitialize even if the user is not head-on. To be able to reinitialize during larger out of plane rotations however, the system must be able to guess the rotation of the head before the tracker is restarted. This can be done either by starting the tracker from a small set of rotated poses or by creating a small number of templates by rotating the three-dimensional model. The templates can then be correlated with the video in the skin blob area and the best template will indicate the approximate pose of the head.

# Chapter 12

# System Evaluation

In this chapter, the tracker is evaluated. First, the performance of the system is described. Then, an evaluation on real data is presented. In particular, it is investigated how the estimation of the covariance matrix $R_k$ and the texture update improves the performance of the tracker. This is followed by a section that investigates the performance on a synthetic sequence, where the ground truth is known. The chapter will be concluded by a section on how to turn the tracking system into a model-based coder.

## 12.1 Performance

The tracking system performs in real time on a SGI O2 R12000 270 MHz workstation. The feature point finding algorithm (executed once at the start of the tracking process) takes about 100 ms, and the rest of the tracking runs at 25 Hz. Rotations from left to right up to $\pm 90^\circ$ are possible. In the up to down range, rotations of up to about $\pm 30^\circ$ can be tracked. For the in-plane rotation component, the system effortlessly tracks a $360^\circ$ rotation. Translation in the $x-$ and $y-$directions of about 7 pixels per frame are possible, which corresponds to a movement of 80% of the width of the screen in one second for the images in Figure 12.1[1].

## 12.2 Evaluation on Real Data

The tracker has been tested on several subjects, as seen in Figure 12.1. A typical tracking sequence can be seen in Figure 12.2. The system is initialized (a), the head is tracked (b and c), the track is lost (d) and regained in (e) through automatic reinitialization.

[1] Only the middle $228 \times 174$ sections of the $320 \times 240$ images are shown in Figure 12.1.

Figure 12.1: *The tracker has been evaluated on different subjects.*

(a) (b) (c) (d) (e)

Figure 12.2: *Typical tracking sequence. The tracking is initialized in (a), and continued in (b) and (c). The track is lost in (d) and in (e) the tracking is regained.*

### 12.2.1 Depth Convergence

As shown in Figure 6.5, the tracked trajectories of the feature points are forwarded to the Kalman filter, which uses this information to infer the depths of the points. In Figure 12.3 the depths are shown before and after convergence. Small circles indicate large depth. In the normal implementation of the tracker, the starting values for the depths $\alpha_i$ are the depths of the model positions obtained from the three-dimensional head model. In this example however, the same depth $\alpha$ has been given to all the points (left) in order to show that they converge to something reasonable (right). Note that the points near the centers of the eyes are the smallest, which is reasonable since they are the points furthest away. Experimentally, the system does not suffer from the possible near-planar shape of the face. Thus the conditions in terms of noise, flatness of the object, prior and motion excitation seems to be so that the system is inside the area of convergence depicted in Figure 8.6.

### 12.2.2 Improvements due to Estimation of $R_k$

The estimation of the covariance matrix $R_k$ (presented in Section 10.3) improves the robustness of the tracker. Figure 12.4 shows an example of a sequence where some or many of the feature points are occluded. A fixed $R_k$ will result in mismatched points (column a) and severe oscillations in pose (column b) and most often to a tracking failure. In contrast, with the proposed estimation of $R_k$, the

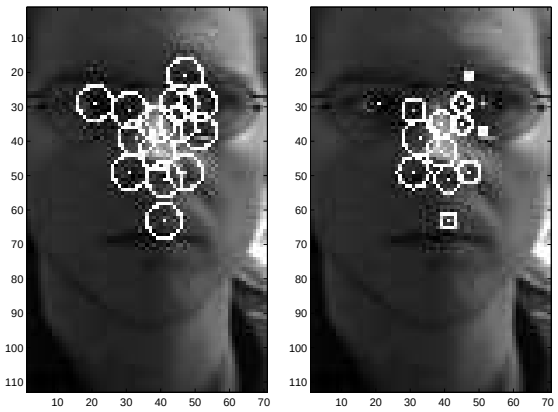



Figure 12.3: *Left: Initial depth estimates. Right: Converged depths estimates. Small circles indicate large depth. Please note the small circles near the centers of the eyes.*

tracking is almost completely unaffected by the occlusion (columns c and d). The sequence shown in Figure 12.4 is representative for how the system works under occlusion. Of course, the tracking can still fail if too many of the points are occluded during too long a period of time, especially if the head is moving behind the occludor.

Moreover, the use of $\theta$ and $V$ in the estimation of $R_k$ prevents invisible or hardly visible points from contributing to the tracking. Without the use of $\theta$ and $V$, points on the occlusion boundary and on the back side of the head will generate erroneous measurements. In this way, the estimated pose can slide away from the correct pose, and tracking fails. By using $\theta$ and $V$, this situation is avoided.

The same effect is also apparent when the head is rotated to the limit of what the tracker can handle, such as in Figure 12.6 c. In this case, about half of the feature points are occluded, and their erroneous measurements will result in a large pose error. This in turn will mean that the templates for the matching in the next frame will be even poorer, making the pose to oscillate and finally making the tracking fail. When $R_k$ is estimated using $\theta$ and $V$, the degradation is more graceful — instead of starting to oscillate, the pose will freeze, since all measurements are finally deemed unreliable. If the head rotates further, there will be a large error between the estimated and the true pose. However, if the head rotates back the same way, tracking can often be resumed.


Figure 12.4: *Occlusion robustness due to estimation of $R_k$. Left: fixed $R_k$, resulting in mismatched points (a) and lost track (b). Right: $R_k$ estimated with the proposed method. Only points that are estimated to be small error cases are shown in (c). The tracking is almost unaffected (d).*

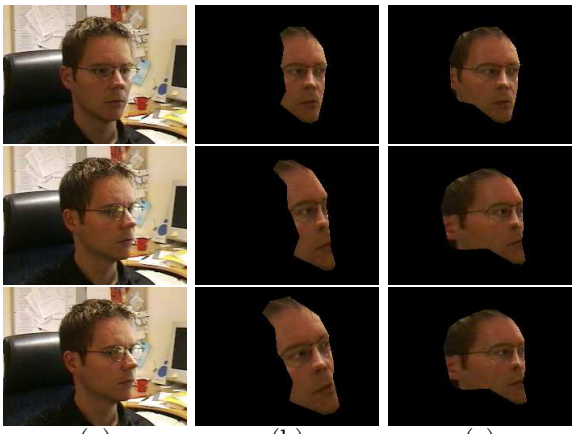
### 12.2.3 Improvements due to Texture Update

Using points on the side of the head extends the rotational range of the tracker. Figure 12.5 shows the difference in tracking performance for a sequence with large out-of-plane rotation (column a). In column b, the additional texture is not used, and the large number of hidden feature points and the poor matching of the visible ones make the tracker oscillate and ultimately lose track. Column c shows the same sequence tracked with the additional texture being used. The tracking is stable.

Even with the added texture, there is a limit to how far the head can be rotated before losing track. Rotations up to 90° are possible. Figure 12.6 shows an example sequence where the head is tracked during a rotational movement of about 90°, and back again.

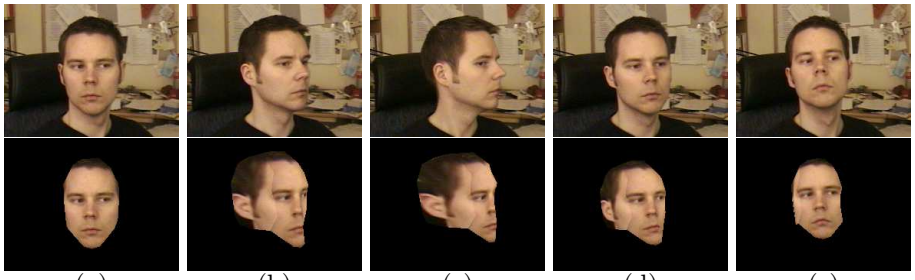
### 12.2.4 Performance of Reinitialization

Figure 12.2 shows a sequence with reinitialization. The reinitialization works when the head is rotated approximately ±20° from the frontal position, in any translatory position. The automatic detection of tracking failure is not waterproof. In order not to have the tracking interrupted by a reinitialization when the tracking works fine, the threshold is moved so that the false alarm



(a) (b) (c)

Figure 12.5: *Improved rotational range due to texture update. For sequences with large rotational motion such as (a), the tracker will normally oscillate strongly and lose track (b). By updating the texture and tracking points on the side of the head, the tracking is made stable (c).*



(a) (b) (c) (d) (e)

Figure 12.6: *Example of a sequence where a rotation of around* $90^\circ$ *is correctly tracked due to the texture update scheme.*

rate is rather small. This means that the system will sometimes not discover a real tracking failure. Still, the reinitialization procedure is useful since it most often will detect a tracking failure. In the few cases where it fails to do so, the user can trigger the reinitialization by pressing a button or even quickly moving the head. A better solution to this problem is of course to improve the failure detection unit, for instance through eigenspace analysis of the face such as in [35]. During the evaluation of the tracker on different persons, it was found that the reinitialization improves the system's ease of use considerably, and would be useful even if it were completely manually triggered.

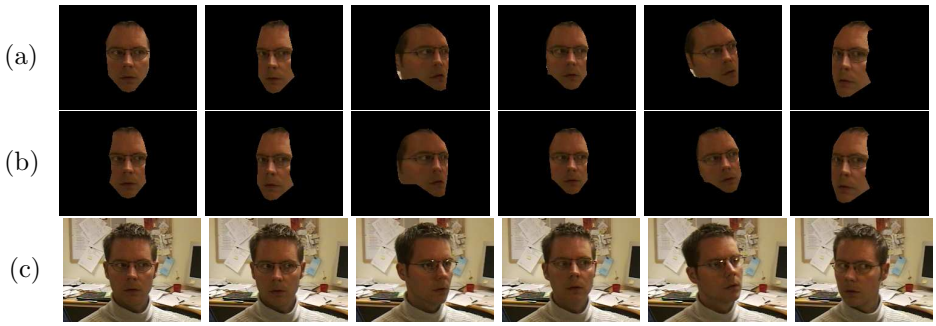



Figure 12.7: *(a) Rendered image sequence that was used for tracker performance evaluation. (b) The result of tracking the synthetic sequence. (c) Original image sequence that was used to obtain the motion parameters used in (a).*

## 12.3 Evaluation on Synthetic Data

To get an idea of the accuracy of the tracking, the following experiment has been conducted: First the tracker is run on a live image sequence (Figure 12.7 c) to provide ground truth motion parameters (the "first pass"). Then a synthetic sequence is rendered using these motion parameters (Figure 12.7 a). The tracker is now run a second time on the synthetic sequence ("second pass", Figure 12.7 b), and the estimated motion parameters are compared to the ground truth data. It should be noted that such an experiment does not measure how well the tracker can follow the motion in the original, live image sequence. The reason for this is that it is not possible to rule out that the original sequence is moving in a way that the tracker in the first pass does not follow, and that this complex motion is "filtered out" from the ground truth data. Tracking the synthetic sequence would then be an easier task than tracking the original live sequence. Still, the experiment gives an idea of how well the tracker can behave on a naturally moving head. The first two rows of Figure 12.8 show the rotation parameters (one graph for each of the four quaternions) and the last row shows the translation parameters. In each chart the ground truth is marked with a dashed curve and the tracked data is marked with a solid curve. The most striking systematic error is that there is a delay between the ground truth and the estimates from zero to about ten frames (0.4 s). Also, at around frame 150, the error between the estimated parameters and the ground truth widens temporarily, as best seen in the $Y$ translation and in quaternion 1, 2 and 4. Around frame 160 however, the estimates have recovered.

## 12.4 Coding

As reviewed in the first part of this thesis, Principal Component Analysis (PCA), or eigenspace analysis, has proved to be a powerful tool for analysis and representation of face- as well as lip images [59, 45, 72, 48, 14].

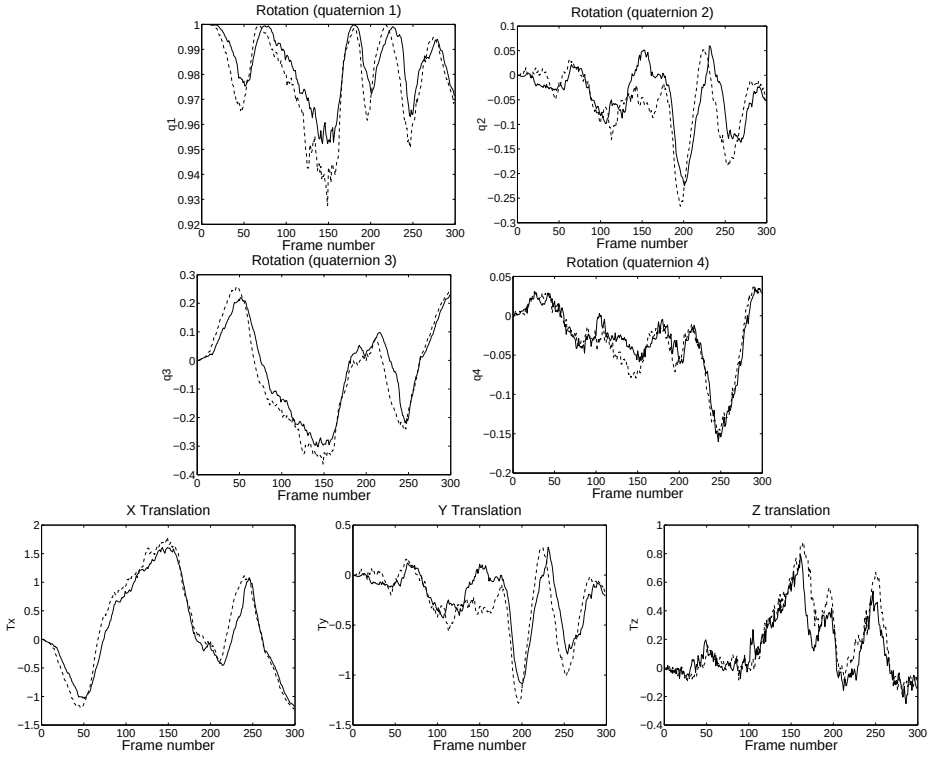



Figure 12.8: *Example of tracking of a synthesized sequence where the motion is known. In each graph, the dashed curve represents the ground truth motion, and the solid curve represents the motion estimated from the tracker.*

In the implementation of the head tracking system, a "zeroth-order model-based coder" is included: The face in the incoming video is reconstructed using the face model, head pose information, the texture of the first image plus the real-time texture around the mouth region. The idea is to show a low bit rate, model-based coder operating in real time (12 Hz).

Using the three-dimensional head pose information from the tracker, it is possible to warp back the face to a frontal position. Figure 12.9 shows an example of this: The original image (a) is tracked (b), and it's texture is then warped

a b c d

Figure 12.9: *The original image (a) is tracked (b), and the pose information is used to warp the image to a head-on shot (c). From this image, a* $36 \times 28$ *pixel area around the mouth is cut out (d).*

back to a frontal pose (c). From this pose-normalized image it is reasonable to do projection onto an eigenspace.

A $36 \times 28$ pixel area around the mouth is cut out from the warped-back incoming video (Figure 12.9d). Since the human vision system is more sensitive to differences in luminance than in chrominance, the mouth image is converted from RGB to YUV; the chrominance components U and V are subsampled by a factor of two, thus diminishing their impact on the eigenspace analysis. The mouth image is then encoded using an eigenspace constructed from mouth images obtained the same way. The ensemble mean (first image in Figure 12.10) is subtracted from the image and the residual is projected onto the eigenvectors

Figure 12.10: *The mean image (extreme left) followed by the first six eigenimages.*

(the first six are shown next to the mean image in Figure 12.10). Each coefficient is then quantized by dividing it by its standard deviation (the square-root of the corresponding eigenvalue) and encoded using a Lloyd-Max quantizer for a Gaussian source. The number of bits used to quantize a coefficient increase with the size of the corresponding eigenvalue. Twelve coefficients are used, and a total of 50 bits are used in the quantization.

The eigenspace is trained on images from the same person, but from a different sequence. The eigenvectors and eigenvalues must thus be transmitted to the decoder in order for it to be able to reconstruct the images. By quantizing the eigenvectors to 8 bits and compressing them losslessly (gzip), this data can be sent at around 17000 bytes. The first texture must also be sent (e.g. using JPEG) adding another 7000 bytes to the startup cost of a transmission. Each additional frame requires a mere 98 bits of information; 50 bits for the eigencoefficients and 48 bits of motion information (each degree of freedom quantized to 8 bits). A five-minute conversation at 12 Hz would thus result in a data rate of about 1.8 kbit/s (1.1 kbit/s if startup costs are ignored). Figure 12.11 shows examples of the mouth-chip coded this way. Figure 12.12 shows two frames

Figure 12.11: *Top row: original images. Bottom row: coded with 50 bits per mouth.*

from a longer sequence that has been tracked and coded using the model-based

Figure 12.12: *Two frames from a sequence with coded mouth regions. Left column: original images. Right column: tracked and coded result.*

coder.

# Chapter 13

# Conclusions

This thesis has concentrated on two topics in model-based coding; facial texture compression (Part 1) and head tracking (Part 2).

The first part treats the efficient coding of face images, or face textures. Chapter 2 argues that an efficient parameterization of the face space (the set of all face images) should be convex, and that this is not the case for eye matching normalization followed by PCA/KLT. By introducing a geometrical normalization step, the parameterization of the face space becomes convex and the measured and perceived image quality rises.

Chapter 3 introduces a block-based algorithm that substantially lowers complexity and at the same time increases image quality, at the expense of compression efficiency. Still, comparisons with JPEG show an increase in PSNR of more than 8 dB for the same bit rate.

In Chapter 4 it is investigated how much can be gained by distributing the bits unevenly over the face. It is shown that PSNR on the average rises 0.4 dB, and the perceived quality even more, since bits are concentrated on visually important regions such as the nose and the mouth.

In Chapter 5 the problem of placing of the feature points is formulated as an optimization problem. By starting with a set of reasonable feature points, it is possible to make improvements by optimization of the target function using simple coordinate search. The work of Cootes et al. on active appearance models involves a PCA based parameterization of the feature points and a better search method which makes it more powerful[1]: Whereas the optimization procedure in Chapter 5 can only refine the feature points from a good starting position, the algorithm by Cootes et al. converges if started with the mean shape in a

[1]The work in Chapter 5 was published in [64] shortly after the work of Cootes et al. [17] but was conducted independently. The work in Chapter 2 and Chapter 3 was published in [63] prior to the work of Cootes et al.

reasonable position.

In the second part, it is shown that it is possible to build a stable head tracker by tracking a large number of automatically selected feature points, constrained by dynamically estimated structure. Naturally, the algorithm used for estimating the structure is a vital component of the system. Therefore the method of choice, the Kalman filter approach of Azarbayejani and Pentland, needs to be examined carefully. In Chapter 7 it is presented and compared to standard methods based on multilinear constraints. In Chapter 8, the behavior of the algorithm is studied for the degenerate case of planar objects. The method is found to converge, albeit not as reliably as in the case of a general three-dimensional object. Experiments are also conducted with a three-view filter, that at least theoretically should be more stable than the original Azarbayejani-Pentland formulation. However, the experiments do not indicate any large gains in performance.

To be able to add points that are not visible in the original image, Chapter 9 presents an extension of the Azarbayejani-Pentland algorithm. By having two reference frames — one for the old points and one for the new — it is possible to avoid biases due to erroneous depth estimates. It should be noted that it is this extension that allows the tracker to benefit from texture updates. Points located on the updated texture areas can then be followed and incorporated in the pose estimation.

Chapter 10 describes the implementation of the system. A large section treats the estimation of the measurement covariance. Different methods for the covariance estimation were tried during the construction of the system. It is surprisingly easy to invent ad hoc solutions that may not increase system performance at all, or even decrease it. It seems to be especially dangerous to assign a covariance that is too small, with a divergence of the SfM solution as a result. A good rule of thumb is to let the standard deviation be larger than one pixel in all directions.

Chapter 11 shows that it can be advantageous to have a reinitialization procedure that is different from the original initialization procedure. By using information gathered in the original initialization procedure, the reinitialization procedure can be made very simple and quite reliable. Possible improvements could be to add more visual processes to make the detection of tracking failure more robust; the DFFS measure from [35] is one candidate, the distance between the estimated pose and the largest skin colored blob is another. Moreover, to allow reinitialization from larger out-of-plane rotations, a small number of templates could be produced by rotating the three-dimensional model. The template with the best correlation would then decide the starting pose of the head model before reinitialization is started.

In Chapter 12, the system is evaluated both on real and on synthetic data.

The evaluation confirms the hypothesis that a stable tracker can be constructed this way. The system evaluation also shows how estimation of the measurement noise covariance $R_k$ can improve the robustness of the tracker. Furthermore, it is shown how the extension of the SfM algorithm in Chapter 9 in combination with texture updating can improve the range of the tracker.

The last section of Chapter 12 treats a small model-based coder that builds mainly on the head tracking system of Part 2, but also touches on the PCA/KLT based algorithms treated in Part 1. The result is a simple version of a model-based coder that can transmit the image of a talking face at rates down to 1 kbit/s, and which works in real-time.

# Appendix A

# KLT, PCA and SVD

This appendix will explain the details of the different mathematical and statistical tools that are used in the first part of this thesis. Mainly this appendix will be about the Karhunen-Loève transform (KLT), but will also briefly cover Principal Component Analysis (PCA) and Singular Value Decomposition (SVD), since these techniques are closely related to the KLT.

## A.1 The Karhunen-Loève Transform

This section essentially follows the presentation by Fukunaga [25], but is included here for the convenience of the reader. The notation is somewhat changed to conform to the one used in this work. Hence a random variable will be written in boldface letters (e.g., $\mathbf{X}$), and a vector (column matrix) will be written with a bar (e.g., $\bar{x}$). Matrices will be written with capital or Greek letters (e.g., $A, \Gamma$). For example, $\bar{\mathbf{X}}$ is a vector valued random variable and $X$ is a matrix. The inner product between two vectors $\bar{a}$ and $\bar{b}$ is written $\bar{a}^T\bar{b}$ or $\langle \bar{a} \,|\, \bar{b} \rangle$.

We may regard an image $X[i,j]$ supported on $[n_x \times n_y]$ pixels as an $n = n_x n_y$ dimensional vector $\bar{x}$. This vector can in turn be thought of as the outcome of an $n$-dimensional random variable $\bar{\mathbf{X}}$. $\bar{\mathbf{X}}$ can be represented losslessly (without error) by a summation of $n$ linearly independent vectors, such as

$$\bar{\mathbf{X}} = \sum_{k=1}^{n} \mathbf{Y}_k \bar{\varphi}_k \tag{A.1}$$

or, in vector notation, $\bar{\mathbf{X}} = \Phi\bar{\mathbf{Y}}$ where $\Phi = [\bar{\varphi}_1 \dots \bar{\varphi}_n]$ and $\bar{\mathbf{Y}} = [\mathbf{Y}_1 \dots \mathbf{Y}_n]^T$. $\Phi$ is a deterministic matrix which can be assumed to be orthogonal (i.e., $\Phi^T = \Phi^{-1}$). It is thus possible to calculate the components of $\mathbf{Y}$ as $\mathbf{Y}_k = \bar{\varphi}_k^T\bar{\mathbf{X}}$, which means that the random vector $\bar{\mathbf{Y}}$ is just an orthonormal transformation of $\bar{\mathbf{X}}$.

There are several reasons for transforming an image before coding it. Two

of the most important effects to achieve are *energy compaction* and *decorrelation of pixels*. This section will show that both of these are characteristics of the Karhunen-Loève transform.

Instead of sending the individual pixel intensity values over the channel, a transform image coder works by transforming the image and sending the transform coefficients over the channel. Energy compaction means that, after the transform, the energy of the image will be concentrated to a small number of coefficients. Thus it will be possible to truncate the coefficient stream and send only $M$ ($< n$) coefficients $\mathbf{Y}_k$ and still be able to approximate the image $\bar{\mathbf{X}}$ well. The coefficients that are not sent can be replaced by preselected constants $b_k$, $\hat{\mathbf{Y}} = [\mathbf{Y}_1 \dots \mathbf{Y}_M \, b_{M+1} \dots b_n]$. (Here the $\mathbf{Y}_k$s are renumbered so that the expected energy is concentrated to the first $M$ coefficients.) $\bar{\mathbf{X}}$ can thus be approximated by $\hat{\mathbf{X}} = \Phi\hat{\mathbf{Y}}$. The error in the representation can be measured by the *mean-square error* (MSE) as follows,

$$\begin{aligned} \text{MSE} &= \frac{1}{n} E\{\|\bar{\mathbf{X}} - \hat{\mathbf{X}}\|^2\} = \frac{1}{n} E\{\|\bar{\mathbf{Y}} - \hat{\mathbf{Y}}\|^2\} = \\ &= \frac{1}{n} \sum_{k=M+1}^{n} E\{|\mathbf{Y}_k - b_k|^2\}. \end{aligned} \tag{A.2}$$

Each choice of $\Phi$ and constant terms $b_k$ yields a value of MSE. To find the constant terms that minimize MSE for a given $\Phi$, the differential of Equation A.2 with respect to $b_k$ is set to zero and solved for $b_k$, which results in $b_k = E\{\mathbf{Y}_k\}$. In other words, the $\mathbf{Y}_k$s that are not transmitted should be replaced by their expected values. The mean-square error can be written as

$$\begin{aligned} \text{MSE} &= \frac{1}{n} \sum_{k=M+1}^{n} E\left[(\mathbf{Y}_k - E\{\mathbf{Y}_k\})^2\right] \\ &= \frac{1}{n} \sum_{k=M+1}^{n} \bar{\varphi}_k^T E\left[(\bar{\mathbf{X}} - E\{\bar{\mathbf{X}}\})(\bar{\mathbf{X}} - E\{\bar{\mathbf{X}}\})^T\right] \bar{\varphi}_k \\ &= \frac{1}{n} \sum_{k=M+1}^{n} \bar{\varphi}_k^T C_{\bar{\mathbf{X}}} \bar{\varphi}_k, \end{aligned} \tag{A.3}$$

where $C_{\bar{\mathbf{X}}}$ is, by definition, the covariance matrix of $\bar{\mathbf{X}}$.

THEOREM 2 *The $\bar{\varphi}_k$s that minimizes the mean-square error* MSE *in Equation A.3 are the eigenvectors of $C_{\bar{\mathbf{X}}}$, that is, the ones which satisfy*

$$C_{\bar{\mathbf{X}}} \bar{\varphi}_k = \lambda_k \bar{\varphi}_k. \tag{A.4}$$

*The mean-square error will be equal to*

$$\text{MSE}_{opt} = \frac{1}{n} \sum_{k=M+1}^{n} \lambda_k. \tag{A.5}$$

The proof is omitted here, but can be found in [25]. Two things should be noted about the Karhunen-Loève transform. Firstly, the $M$ eigenvectors of $C_{\bar{\mathbf{X}}}$ that constitute the basis functions, minimize the mean-square error MSE over all choices of $M$ orthogonal basis vectors. Therefore, as promised in the beginning of the section, the Karhunen-Loève transform compacts the energy better than (or as well as) any other orthogonal transform. Secondly, the coefficients $\mathbf{Y}_k$ are mutually uncorrelated, since their covariance matrix is diagonal:

$$C_{\bar{\mathbf{Y}}} = \Phi^T C_{\bar{\mathbf{X}}} \Phi = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \tag{A.6}$$

## A.2 Principal Component Analysis

Following Jolliffe [37], the first step of principal component analysis (PCA) of a random vector $\bar{\mathbf{X}}$ is to find a linear function $\bar{\rho}_1^T \bar{\mathbf{X}}$ of $\bar{\mathbf{X}}$ which has maximum variance. The second step is to find a linear function $\bar{\rho}_2^T \bar{\mathbf{X}}$ uncorrelated with $\bar{\rho}_1^T \bar{\mathbf{X}}$ which has maximum variance, and so on, so that at the $k$th stage a linear function $\bar{\rho}_k^T \bar{\mathbf{X}}$ is found which has maximum variance subject to being uncorrelated with $\bar{\rho}_1^T \bar{\mathbf{X}}, \bar{\rho}_2^T \bar{\mathbf{X}}, \ldots, \bar{\rho}_{k-1}^T \bar{\mathbf{X}}$. The $k$th variable, $\bar{\rho}_k^T \bar{\mathbf{X}}$ is called the $k$th *principal component*, or PC for short. As shown by Jolliffe [37], the vector $\bar{\rho}_k$ associated with the $k$th PC is the $k$th eigenvector of the covariance matrix $\mathbf{C}_{\bar{\mathbf{X}}}$ of $\bar{\mathbf{X}}$. This means that the vectors $\bar{\rho}_k$ obtained in the PCA are the same as the basis vectors $\bar{\varphi}_k$ of the Karhunen-Loève transform. This thesis will refer to the Karhunen-Loève transform as the procedure of projecting a vector $\bar{\mathbf{X}}$ onto its KLT basis vectors $\bar{\varphi}_k$, whereas the calculation of the vectors $\bar{\varphi}_k$ will be called PCA.

## A.3 Calculation of Karhunen-Loève Basis Vectors, PCA

As seen in the preceding section, the Karhunen-Loève basis vectors $\bar{\varphi}_k$ are calculated from the covariance matrix $C_{\bar{\mathbf{X}}}$ of the random vector $\bar{\mathbf{X}}$. One way to estimate the covariance matrix is to assume some statistical model and then derive the covariance matrix analytically. In most cases, however, it is not easy to find a model that applies to the problem in any reasonable way. It is then neither possible to derive the covariance matrix nor the basis vectors analytically. However, given sufficient amounts of training data, it is possible to estimate the covariance matrix.

Let $\bar{\mathbf{X}}$ represent a random vector that generates an image with $n$ pixels, according to $\bar{\mathbf{X}} = [\mathbf{X}_1 \mathbf{X}_2 \ldots \mathbf{X}_n]^T$. Assume there are $N$ outcomes $\{\bar{x}^1, \bar{x}^2, \ldots, \bar{x}^N\}$ of this random vector that constitute a *training set*, denoted $\{\bar{x}^k\}_1^N$. Each element

$\bar{x}^k$ in this training set is a vector $\bar{x}^k = \left[x_1^k x_2^k \dots x_n^k\right]^T$. It is then possible to estimate the covariance between two pixels $i$ and $j$ by the measure

$$\hat{c}_{ij} = \frac{1}{N-1}\left[\sum_{k=1}^{N} x_i^k x_j^k - \frac{1}{N}\left(\sum_{k=1}^{N} x_i^k\right)\left(\sum_{k=1}^{N} x_j^k\right)\right]. \tag{A.7}$$

The covariance matrix can thus be estimated using

$$C_{\bar{\mathbf{X}}} \approx \hat{C}_{\bar{\mathbf{X}}} = (\hat{c}_{ij})\,. \tag{A.8}$$

If the random vector is zero-mean, the calculations can be simplified. (If not, a new random vector of zero mean can be constructed through $\bar{\mathbf{Y}} = \bar{\mathbf{X}} - E\{\bar{\mathbf{X}}\}$, where $E\{\bar{\mathbf{X}}\}$ is approximated by $\frac{1}{N}\sum_{k=1}^{N}\bar{x}^k$.) The covariance matrix can then be formed as

$$\hat{C}_{\bar{\mathbf{X}}} = \frac{1}{N-1}\begin{bmatrix} \sum_{k=1}^{N} x_1^k x_1^k & \sum_{k=1}^{N} x_1^k x_2^k & \dots & \sum_{k=1}^{N} x_1^k x_n^k \\ \sum_{k=1}^{N} x_2^k x_1^k & \sum_{k=1}^{N} x_2^k x_2^k & \dots & \sum_{k=1}^{N} x_1^k x_n^k \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{k=1}^{N} x_n^k x_1^k & \sum_{k=1}^{N} x_n^k x_2^k & \dots & \sum_{k=1}^{N} x_n^k x_n^k \end{bmatrix} \tag{A.9}$$

or, in vector notation:

$$\hat{C}_{\bar{\mathbf{X}}} = \frac{1}{N-1} AA^T, \tag{A.10}$$

where $A$ is a matrix with the training set images as columns,

$$A = \begin{bmatrix} | & | & & | \\ \bar{x}^1 & \bar{x}^2 & \cdots & \bar{x}^N \\ | & | & & | \end{bmatrix}. \tag{A.11}$$

Note that $\hat{C}_{\bar{\mathbf{X}}}$ has the same eigenvectors as $AA^T$. Moreover, if $l_k$ is an eigenvalue of $AA^T$, $\lambda_k = \frac{1}{N-1} l_k$ will be an eigenvalue to $\hat{C}_{\bar{\mathbf{X}}}$. It is thus possible to calculate the transform basis vectors $\hat{\varphi}_k$ and the corresponding eigenvalues $\lambda_k$ from the matrix $AA^T$. The hat over $\hat{\varphi}_k$ is there to remind us that the basis vectors obtained are only approximations of the true $\bar{\varphi}_k$, since they are calculated from $\hat{C}_{\bar{\mathbf{X}}}$ and not from $C_{\bar{\mathbf{X}}}$.

Note also that $\hat{C}_{\bar{\mathbf{X}}}$ (and thus also $AA^T$) is an $n \times n$ matrix which makes it computationally expensive to calculate the eigenvectors. However, if $N$, the number of images in the training set, is smaller than $n$, it is computationally advantageous instead to consider the eigenvectors $\bar{\xi}_k$ of $A^T A$:

$$A^T A \bar{\xi}_k = l_k \bar{\xi}_k \tag{A.12}$$

The above equation is multiplied by $A$,

$$(AA^T)(A\bar{\xi}_k) = l_k (A\bar{\xi}_k) \tag{A.13}$$

and it is clear that $(A\tilde{\varphi}_k)$ is an eigenvector to $AA^T$ (and hence also to $\hat{C}_{\bar{\mathbf{x}}}$). Thus

$$\hat{\varphi}_k = A\bar{\xi}_k, \tag{A.14}$$

is the $k$th principal component (or Karhunen-Loève basis vector) and

$$\lambda_k = \frac{1}{N-1} l_k \tag{A.15}$$

is its corresponding eigenvalue. In other words, the eigenvectors $\hat{\varphi}_k$ can be computed from both $AA^T$ and $A^TA$, so it is always possible to choose the one that is smallest. Moreover, by rewriting equation A.14,

$$\hat{\varphi}_k = \left[\bar{x}^1 \bar{x}^2 \dots \bar{x}^n\right] \begin{bmatrix} \xi_{k1} \\ \xi_{k2} \\ \vdots \\ \xi_{kN} \end{bmatrix} \tag{A.16}$$

it becomes apparent that the basis vectors $\hat{\varphi}_k$ are simply linear combinations of the vectors in the training set. Note also that the $(i,j)$th element of the matrix $A^TA$ is the inner product between the $i$th and the $j$th image of the training set,

$$(A^TA)_{ij} = \left\langle \bar{x}^i \,|\, \bar{x}^j \right\rangle \tag{A.17}$$

## A.4 Singular Value Decomposition

In Section A.1, the Karhunen-Loève Transform was shown to find, given knowledge of the statistics, the optimal basis vectors with which to describe the data. The basis vectors obtained are optimal in the sense that they minimize the mean-square error. The Singular Value Decomposition, on the other hand, is not founded on statistics but is a deterministic way of factorizing a given matrix. However, as will be seen in this section, the two approaches can be used to give exactly the same result.

Before going into details on the relations between the SVD and the KLT, it is important to point out that the SVD, due to the existence of a robust numerical implementation, is used as a tool in numerical analysis to solve a large range of problems such as determining the rank of a matrix, least-squares fit, etc. Thus the possibility of using the SVD to calculate the KLT basis vectors is far from the only application.

THEOREM 3 *Any $n \times N$-matrix $A$ can be factorized*

$$A = U\Sigma V \tag{A.18}$$

*(**Singular Value Decomposition**) where $U$ and $V$ are orthogonal matrices of sizes $n \times n$ and $N \times N$ respectively, and $\Sigma$ is an $n \times N$-matrix of the form*

$$\Sigma = \begin{bmatrix} S_r & 0 \\ 0 & 0 \end{bmatrix} \tag{A.19}$$

*where* $S_r = diag(\sigma_1, \sigma_2, \ldots, \sigma_r)$ *and* $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0$. *The scalars* $\sigma_1, \sigma_2, \ldots, \sigma_r$ *are uniquely determined by* $A$ *and are called the* ***singular values*** *of the matrix. The number* $r$ *is the rank of the matrix.*

The proof is omitted here, but can be found in [62]. It should be noted that if all $N$ columns of $A$ are linearly independent, the first $N$ columns of $U$ constitute an orthonormal base spanning the same subspace as the columns of $A$. Thus, by constructing a matrix $A$ where each column is an image vector from the training set according to

$$A = \left[\bar{x}^1 \bar{x}^2 \ldots \bar{x}^N\right], \tag{A.20}$$

it would be possible to obtain a set of basis vectors $\psi_k$ by performing SVD on the matrix

$$A = U\Sigma V \tag{A.21}$$

and letting the basis vectors be the $N$ first columns of $U$,

$$\psi_1 = U_1, \psi_2 = U_2, \ldots, \psi_N = U_N. \tag{A.22}$$

As will be shown below, the basis vectors $\psi_k$ achieved this way will be exactly the same as the Karhunen-Loève basis vectors $\hat{\varphi}_k$ obtained in the last subsection.

THEOREM 4 *The* $m$ *first columns* $U_1, U_2, \ldots, U_m$ *of the matrix* $U$ *obtained by the singular value decomposition* $A = U\Sigma V$ *of* $A$ *are equal to the eigenvectors* $\hat{\varphi}_k$ *of the matrix* $AA^T$.

PROOF 4 From $A = U\Sigma V$, follows

$$\begin{aligned} AA^T &= U\Sigma V(U\Sigma V)^T \\ &= U\Sigma V V^T \Sigma^T U^T \\ &= U\Sigma^2 U^T, \end{aligned} \tag{A.23}$$

since $V$ is unitary and $\Sigma$ is diagonal. Multiplying equation A.23 from the right by $U$ gives

$$\begin{aligned} AA^T U &= U\Sigma^2 U^T U \\ &= U\Sigma^2. \end{aligned} \tag{A.24}$$

This gives

$$\begin{aligned} AA^T U_1 &= U_1 \sigma_1^2 \\ AA^T U_2 &= U_2 \sigma_2^2 \\ &\vdots \\ AA^T U_N &= U_N \sigma_N^2, \end{aligned} \tag{A.25}$$

and thus the $N$ first columns of $U$ are eigenvectors to $AA^T$. □

As shown above, the SVD can be used for finding the Karhunen-Loève Transform basis vectors $\hat{\varphi}_k$. This should not be confused with the technique proposed by Andrews and Patterson [6], where an image is compressed by performing SVD on a *single image*. In this case, the basis vectors will be different for each image, and will have to be transmitted over the channel.

# Appendix B

# Optical Flow

Consider a point **P** that is depicted by a moving pin-hole camera. Between two consecutive frames, the camera has rotated $(\Omega_x, \Omega_y, \Omega_z)$ around and translated $(T_x, T_y, T_z)$ along its three coordinate axes. This is equivalent to keeping the camera still and moving the point, which is shown in Figure B.1, where the point moves from $P_1 = (X_1, Y_1, Z_1)$ to $P_2 = (X_2, Y_2, Z_2)$. The images in Figure B.1 are

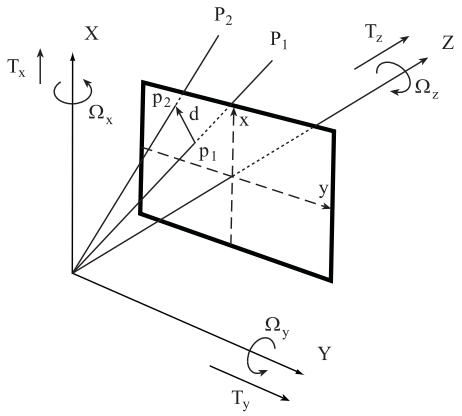



Figure B.1: *The point P moves from $P_1$ to $P_2$ in the camera coordinate sytem due to rotational $(\Omega_x, \Omega_y, \Omega_z)$ and translational $(T_x, T_y, T_z)$ motion of the camera.*

denoted $I(x, y, t_1)$ and $I(x, y, t_2)$ (or $I_1$ and $I_2$ for short), and the coordinates of the points in the image plane are represented by $p_1 = (x_1, y_1)$, and $p_2 = (x_2, y_2)$. The *displacement vector* $(\alpha, \beta)$ (denoted $d$ in Figure B.1) of the point $(x_1, y_1)$ can then be defined as

$$\begin{aligned} \alpha &= x_2 - x_1 \\ \beta &= y_2 - y_1. \end{aligned} \tag{B.1}$$

The displacement vector $(\alpha, \beta)$ thus tells where to find a corresponding point from image $I_1$ in image $I_2$. If the displacement is defined for each pixel position

$(x, y)$ in image $I_1$, the result is a *displacement field*, $(\alpha(x, y), \beta(x, y))$, where both $\alpha$ and $\beta$ depends on $x$ and $y$. A pixel $(x_1, y_1)$ in image $I_1$ can then be found in $\big(x_1 + \alpha(x_1, y_1), \quad y_1 + \beta(x_1, y_1)\big)$ in image $I_2$.

Closely related to the displacement field is the *velocity field* $(u, v)$, which is defined as the velocity of a pixel in the image plane. Thus

$$\begin{aligned} u &= \frac{dx}{dt} \\ v &= \frac{dy}{dt}, \end{aligned} \tag{B.2}$$

where $(dx, dy)$ is the displacement vector generated by camera motion during $dt$. In the same manner as the displacement field, $u$ and $v$ are both functions of $x$ and $y$; $u = u(x, y)$, $v = v(x, y)$. If the time $\Delta t = t_2 - t_1$ between the two frames $I_1$ and $I_2$ is known, the velocities $u$ and $v$ can be approximated using

$$\begin{aligned} u &\approx \frac{x_2 - x_1}{\Delta t} = \frac{\alpha}{\Delta t} \\ v &\approx \frac{y_2 - y_1}{\Delta t} = \frac{\beta}{\Delta t}. \end{aligned} \tag{B.3}$$

Choosing the time scale such that $\Delta t = 1$ yields $(u, v) \approx (\alpha, \beta)$. This blurs the border between the velocity field and the displacement field. Following the notation used by Adiv [2], this thesis will use the term *optical flow* field when referring to both the velocity field and the displacement field. The notation $(u, v)$ will be used for both types of optical flow fields.

If the object is fixed and only the camera moves, (or equivalently, if the camera is fixed but the object moves rigidly), the motion can be described by a *rigid body transformation*,

$$P_2 = RP_1 + T, \tag{B.4}$$

where $R$ is a $3 \times 3$ rotation matrix and $T = (T_x, T_y, T_z)$ is a translation vector. Furthermore, the projection from a three dimensional point to the image can be calculated using

$$\begin{aligned} x &= f\frac{X}{Z} \\ y &= f\frac{Y}{Z}, \end{aligned} \tag{B.5}$$

where $f$ is the distance from the image plane to the center of projection as shown in Figure B.1. By assuming that $f = 1$, that the rotation between frames is small, and that $T_z/Z \ll 1$, Adiv [2] showed that the displacement field can be approximated by

$$\begin{aligned} u(x, y) &\approx -\Omega_x xy + \Omega_y(1 + x^2) - \Omega_z y + (T_x - T_z x)/Z \\ v(x, y) &\approx -\Omega_x(1 + y^2) + \Omega_y xy + \Omega_z x + (T_y - T_z y)/Z, \end{aligned} \tag{B.6}$$

where $\Omega_x$, $\Omega_y$ and $\Omega_z$ are the differential Euler angles around the x-, y- and z-axis respectively, as shown in Figure B.1. Equation B.6 is also valid for the velocity field. Setting $\Delta U = (\Omega_x, \Omega_y, \Omega_z, T_x, T_y, T_z)$, Equation (B.6) can be written in matrix notation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} c^u \\ c^v \end{bmatrix} \Delta U, \tag{B.7}$$

or

$$\begin{bmatrix} u \\ v \end{bmatrix} = C\Delta U, \tag{B.8}$$

where

$$C = \begin{bmatrix} c^u \\ c^v \end{bmatrix} = \begin{bmatrix} -(1+y^2) & xy & x & 0 & 1/Z & -y/Z \\ -xy & (1+x^2) & -y & 1/Z & 0 & -x/Z \end{bmatrix}. \tag{B.9}$$

## B.1 The Optical Flow Constraint

To estimate the optical flow, a model is needed that explains how the optical flow relates to the image intensity. A common approach is to assume that the projection of an object point onto a pixel preserves its brightness from one frame to another. This is not generally true, since many materials reflect different amounts of light in different angles, but it is a reasonable approximation. The assumption can be formulated

$$I(x+u, y+v, t+1) = I(x, y, t). \tag{B.10}$$

Taylor expanding the left-hand side of Equation (B.10) around $(x, y, t)$ and removing higher order terms results in

$$\begin{aligned} I(x,y,t) + I_x u + I_y v + I_t &= I(x,y,t) \\ I_x u + I_y v + I_t &= 0. \end{aligned} \tag{B.11}$$

The above equation is called the *optical flow constraint equation*, and is valid for small motions between the frames. The notation can be further simplified to

$$\begin{bmatrix} I_x & I_y \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = -I_t \tag{B.12}$$

$$\nabla I \begin{bmatrix} u \\ v \end{bmatrix} = -I_t. \tag{B.13}$$

# Appendix C

# Extended Kalman Filtering

## C.1 Kalman Filtering

This section will treat Kalman filtering [38] and its non-linear extension. Following Gelb et al. [26], consider a discrete, dynamic system

$$\bar{\mathbf{x}}_k = \Phi_{k-1}\bar{\mathbf{x}}_{k-1} + \bar{\mathbf{w}}_{k-1}, \tag{C.1}$$

where $\bar{\mathbf{x}}_k$, the *state vector*, is a vector valued random variable that we are interested in estimating. $\Phi_{k-1}$ is a matrix describing how the state vector changes from state $\bar{\mathbf{x}}_{k-1}$ to $\bar{\mathbf{x}}_k$, and $\bar{\mathbf{w}}_k$ is a zero mean, white gaussian sequence with covariance matrix $Q_k$.

The model further assumes that the state vector $\bar{\mathbf{x}}_k$ cannot be measured directly, instead linear combinations $\bar{\mathbf{z}}_k$ of the state are measured according to

$$\bar{\mathbf{z}}_k = H_k\bar{\mathbf{x}}_k + \bar{\mathbf{v}}_k. \tag{C.2}$$

Here $\bar{\mathbf{z}}_k$ is a vector of measurements at time $k$, and $\bar{\mathbf{v}}_k$ is a zero mean gaussian sequence with covariance matrix $R_k$. Note that the matrix $H_k$ that can be of less than full rank or even rectangular. This means that the dimensionality of $\bar{\mathbf{z}}$ can be less than that of $\bar{\mathbf{x}}$.

The state estimate is updated using

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k(\bar{\mathbf{z}}_k - H_k\hat{\mathbf{x}}_k^-), \tag{C.3}$$

where the $\hat{\mathbf{x}}_k^-$ denotes the prediction of the state *before* the measurement $\bar{\mathbf{z}}_k$, and $\hat{\mathbf{x}}_k^+$ is the estimate *after* the measurement.

The estimated from the previous time step, $\hat{\mathbf{x}}_{k-1}^+$, can be improved before the measurement using the deterministic part of Equation (C.1):

$$\hat{\mathbf{x}}_k^- = \Phi_{k-1}\hat{\mathbf{x}}_{k-1}^+. \tag{C.4}$$

### C.1.1 Calculating $K_k$

The equations C.3 and C.4 can now be used interchangeably to obtain an estimate of the state $\bar{\mathbf{x}}_k$. In order to do so, the matrix $K_k$ must be calculated. The derivation of $K_k$ below follows Gelb et al. [26].

Let $P_k^+$ denote the covariance matrix of the error of $\hat{\mathbf{x}}_k^+$,

$$P_k^+ = E\left[(\hat{\mathbf{x}}_k^+ - \bar{\mathbf{x}}_k)(\hat{\mathbf{x}}_k^+ - \bar{\mathbf{x}}_k)^T\right]. \tag{C.5}$$

Simplifying yields

$$P_k^+ = (I - K_k H_k)P_k^-(I - K_k H_k)^T + K_k R_k K_k^T, \tag{C.6}$$

where $R_k = E\left[\bar{\mathbf{v}}_k \bar{\mathbf{v}}_k^T\right]$, i.e., the covariance matrix of $\bar{\mathbf{v}}_k$. Differentiating $P_k^+$ with respect to $K_k$ and setting the result to zero yields

$$-2(I - K_k H_k)P_k^- H_k^T + 2K_k R_k = 0. \tag{C.7}$$

Solving for $K_k$, we get

$$K_k = P_k^- H_k^T \left[H_k P_k^- H_k^T + R_k\right]^{-1} \tag{C.8}$$

which is referred to as the *Kalman gain matrix*.

An update equation for $P_k^-$ is also needed. Using the definition $P_k^- = E\left[(\hat{\mathbf{x}}_k^- - \bar{\mathbf{x}})(\hat{\mathbf{x}}_k^- - \bar{\mathbf{x}})^T\right]$ and Equation (C.4), yields, after some manipulations,

$$P_k^- = \Phi_{k-1} P_{k-1}^+ \Phi_{k-1}^T + Q_{k-1}. \tag{C.9}$$

### C.1.2 Summary of Update Equations

To give a better overview of the Kalman Equations, the summary below is presented.

We want to estimate an quantity $\bar{\mathbf{x}}_k$ that changes dynamically according to the linear system

$$\bar{\mathbf{x}}_k = \Phi_{k-1}\bar{\mathbf{x}}_{k-1} + \bar{\mathbf{w}}_{k-1}, \qquad \bar{\mathbf{w}}_k \sim N(\bar{\mathbf{0}}, Q_k), \tag{C.10}$$

and can be measured by

$$\bar{\mathbf{z}}_k = H_k\bar{\mathbf{x}}_k + \bar{\mathbf{v}}_k, \qquad \bar{\mathbf{v}}_k \sim N(\bar{\mathbf{0}}, R_k). \tag{C.11}$$

The initial conditions that must be known are

$$\hat{\mathbf{x}}_0^+ = E\left[\bar{\mathbf{x}}_0\right], \qquad P_0^+ = E\left[(\bar{\mathbf{x}}_0 - \hat{\mathbf{x}}_0^+)(\bar{\mathbf{x}}_0 - \hat{\mathbf{x}}_0^+)^T\right]. \tag{C.12}$$

The estimate of the state and the error covariance matrix can now be extrapolated to the next time step $k$,

$$\hat{\mathbf{x}}_k^- = \Phi_{k-1}\hat{\mathbf{x}}_{k-1}^+ \tag{C.13}$$

$$P_k^- = \Phi_{k-1} P_{k-1}^+ \Phi_{k-1}^T + Q_{k-1}. \tag{C.14}$$

The Kalman gain $K_k$ and the error covariance $P_k^+$ are updated,

$$K_k = P_k^- H_k^T \left[ H_k P_k^- H_k^T + R_k \right]^{-1} \tag{C.15}$$

$$P_k^+ = (I - K_k H_k) P_k^- \tag{C.16}$$

where Equation (C.16) is a shorter form of Equation (C.6) obtained by substituting with Equation (C.15). Finally the measurement $\bar{\mathbf{z}}_k$ can be used to update the state vector,

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_k^- + K_k(\bar{\mathbf{z}}_k - H_k \hat{\mathbf{x}}_k^-). \tag{C.17}$$

The algorithm then continues by iterating Equation (C.13) – (C.17).

## C.2 Extended Kalman Filtering

Not all problems are possible to express with the linear equations in Equation (C.10) and (C.11). A more general formulation that also models non-linear cases is

$$\begin{cases} \bar{\mathbf{x}}_{k+1} = f_k(\bar{\mathbf{x}}_k) + \bar{\mathbf{w}}_k, & \bar{\mathbf{w}}_k \sim N(\bar{\mathbf{0}}, Q_k), \\ \bar{\mathbf{z}}_k = h_k(\bar{\mathbf{x}}_k) + \bar{\mathbf{v}}_k, & \bar{\mathbf{v}}_k \sim N(\bar{\mathbf{0}}, R_k). \end{cases} \tag{C.18}$$

Here $f_k(\cdot)$ and $h_k(\cdot)$ are non-linear functions of the state $\bar{\mathbf{x}}_k$. We have the same initial conditions as in the linear case

$$\hat{\mathbf{x}}_0^+ = E\left[\bar{\mathbf{x}}_0\right], \qquad P_0^+ = E\left[(\bar{\mathbf{x}}_0 - \hat{\mathbf{x}}_0^+)(\bar{\mathbf{x}}_0 - \hat{\mathbf{x}}_0^+)^T\right]. \tag{C.19}$$

The extrapolation of the state can be performed using the non-linear equation

$$\hat{\mathbf{x}}_k^- = f_{k-1}(\hat{\mathbf{x}}_{k-1}^+), \tag{C.20}$$

but the propagation of the error covariance matrix must be approximated using the truncated Taylor series

$$P_k^- = F_{k-1}(\hat{\mathbf{x}}_{k-1}^+) P_{k-1}^+ F_{k-1}(\hat{\mathbf{x}}_{k-1}^+)^T + Q_{k-1}, \tag{C.21}$$

where

$$F_{k-1}(\hat{\mathbf{x}}_{k-1}^+) = \left. \frac{\partial f_{k-1}(\bar{\mathbf{x}})}{\partial \bar{\mathbf{x}}} \right|_{\bar{\mathbf{x}} = \hat{\mathbf{x}}_{k-1}^+} \tag{C.22}$$

is the linear term of the Taylor expansion of $f_k(\bar{\mathbf{x}}_k)$. The update of the Kalman gain $K_k$ and the error covariance $P_k^+$ are also done using the Taylor series

$$K_k = P_k^- H_k(\hat{\mathbf{x}}_k^-)^T \left[ H_k(\hat{\mathbf{x}}_k^-) P_k^- H_k(\hat{\mathbf{x}}_k^-)^T + R_k \right]^{-1} \tag{C.23}$$

$$P_k^+ = (I - K_k H_k(\hat{\mathbf{x}}_k^-)) P_k^-, \tag{C.24}$$

where

$$H_k(\hat{\mathbf{x}}_k^-) = \left.\frac{\partial h_k(\bar{\mathbf{x}})}{\partial \bar{\mathbf{x}}}\right|_{\bar{\mathbf{x}}=\hat{\mathbf{x}}_k^-} \tag{C.25}$$

is the linear term of the Taylor expansion of $h_k(\bar{\mathbf{x}}_k)$. Finally, the non-linear measurement function $h_k(\cdot)$ is used together with the measurement $\bar{\mathbf{z}}_k$ to update the state vector,

$$\hat{\mathbf{x}}_k^+ = \hat{\mathbf{x}}_{k-1}^- + K_k(\bar{\mathbf{z}}_k - h_k(\hat{\mathbf{x}}_k^-)). \tag{C.26}$$

### C.2.1 Optimality

The linear Kalman filter is the optimal solution to the linear problem of Equations (C.10) and (C.11). The extended Kalman filter, since it is based on approximations, does not claim optimality. In fact, there is no guarantee that Extended Kalman filtering will produce an estimate that is close to the truly optimal solution of the non-linear Equations (C.18). For example, the gaussian characteristic of the noise is not preserved when filtered through non-linear filters. Fortunately, the method has been shown to work well in many practical applications, as reported by Gelb et al. [26].

## C.3 Properties of $R_k$

Scrutinizing equations (C.14) through (C.16), it is evident that $K_k$, $P_k^-$ and $P_k^+$ are estimated only from parameters that are known *a priori* — there is no feedback from the measurement $\bar{\mathbf{z}}_k$ or the state estimate $\hat{\mathbf{x}}_k$ to these matrices. As long as $\Phi_k$, $H_k$, $Q_k$ and $R_k$ are known, $K_k$, $P_k^-$ and $P_k^+$ can be calculated in advance and be stored in memory. This is a good thing when computer processing power must be kept low. Since $P_k$ is the covariance of the error, it is also possible to know the error of the estimate in advance. For instance, it is possible to calculate that the error in the first element of the estimate $\hat{\mathbf{x}}$ at time $k = 1000$ will have a variance of $P_{1000}(0,0) = 0.0001$, even before the first measurement has taken place. This highlights the importance of $\Phi_k$, $H_k$, $Q_k$ and $R_k$ being correct, since there is no mechanism for correcting bad guesses of these matrices[1] based on the measurements $\bar{\mathbf{z}}_k$. Especially dangerous is if, for instance, a measurement error covariance $R_k$ is used that is "smaller" than in reality, since the filter then will trust the measurements more than it should, with a possible divergence of the state estimation error as a result.

### C.3.1 Subspace Locking

In an application where $R_k$ is estimated automatically such as in Section 10.3, it is important to make sure that the smallest value that can come out of such an

[1] Correcting matrices $\Phi_k$, $H_k$, $Q_k$ and $R_k$ falls into the topic of Adaptive Kalman Filtering, described for instance by Gelb et al. [26].

estimation is large enough. The following example illustrates what can happen if too small a variance is used. Consider the simple scalar system

$$\begin{aligned} \mathbf{x}_k &= \mathbf{x}_{k-1} + \mathbf{w}_{k-1} \\ \mathbf{z}_k &= \mathbf{x}_k + \mathbf{v}_k. \end{aligned} \tag{C.27}$$

which is equal to (C.10) and (C.11) for $\Phi_k = 1$ and $H_k = 1$. If the variance $R_k = \mathrm{Var}\left[\mathbf{v}_k\right]$ is set to zero for a certain $k$, the state variable $\mathbf{x}_k$ will be set to the incoming measurement $\mathbf{z}_k$ *irrespectively* of all previous measurements: Assuming the variance $P_k^-$ to be 1, the matrix $K_k$ will equal (using Equation (C.15))

$$\begin{aligned} K_k &= P_k^- H_k^T \left[H_k P_k^- H_k^T + R_k\right]^{-1} = \\ &= 1\left[1+0\right]^{-1} = \\ &= 1. \end{aligned} \tag{C.28}$$

A $K_k$ of 1 means that only the measurement will matter for the update of the state $\hat{\mathbf{x}}_k^-$. If, for instance, previous measurements have generated the prediction $\hat{\mathbf{x}}_k^- = 3$ and the measurement is $\mathbf{z}_k = 27$, the new state estimate $\hat{\mathbf{x}}_k^+$ will be

$$\begin{aligned} \hat{\mathbf{x}}_k^+ &= \hat{\mathbf{x}}_k^- + K_k(\bar{\mathbf{z}}_k - H_k\hat{\mathbf{x}}_k^-) = \\ &= 3 + 1(27-3) \\ &= 27. \end{aligned} \tag{C.29}$$

If the measurement $\mathbf{z}_k = 27$ is bad, the state estimate has been ruined in a single step. Worse still, if normal values of $R_k$ are used for future measurements, the erroneous value will stay for some time. How long it will stay depends on the variance of the state noise $Q_k = \mathrm{Var}\left[\mathbf{w}_k\right]$. Indeed, if $Q_k = 0$, the erroneous value will stay indefinitely: Using Equation (C.16) yields

$$\begin{aligned} P_k^+ &= (I - K_k H_k)P_k^- = \\ &= (1-1)P_k^- = \\ &= 0, \end{aligned} \tag{C.30}$$

which inserted into Equation (C.14) gives

$$\begin{aligned} P_k^- &= \Phi_{k-1}P_{k-1}^+\Phi_{k-1}^T + Q_{k-1} = \\ &= 1 \cdot 0 \cdot 1 + 0 = \\ &= 0. \end{aligned} \tag{C.31}$$

Thus both $P_k^+$ and $P_k^-$ will always be zero in the future, which in turn means that

$$\begin{aligned} K_k &= P_k^- H_k^T \left[H_k P_k^- H_k^T + R_k\right]^{-1} = \\ &= 0 \cdot H_k^T \left[H_k P_k^- H_k^T + R_k\right]^{-1} = \\ &= 0, \end{aligned} \tag{C.32}$$

and hence

$$\begin{aligned}\hat{\mathbf{x}}_k^+ &= \hat{\mathbf{x}}_k^- + K_k(\bar{\mathbf{z}}_k - H_k\hat{\mathbf{x}}_k^-) = \\ &= 27 + 0(\bar{\mathbf{z}}_k - 3) \\ &= 27.\end{aligned} \tag{C.33}$$

### C.3.2 Multi-Dimensional Case

In the multi-dimensional case, a singular covariance matrix $R_k$ plays a similar role to that of a zero variance. In this case, the estimated state vector $\hat{\mathbf{x}}_k^+$ will be restricted to a subspace of the solution space. This is illustrated in the following two-dimensional example system,

$$\begin{aligned}\bar{\mathbf{x}}_k &= \bar{\mathbf{x}}_{k-1} + \bar{\mathbf{w}}_{k-1}, & \bar{\mathbf{w}}_k &\sim N(\bar{\mathbf{0}}, I), \\ \bar{\mathbf{z}}_k &= \bar{\mathbf{x}}_k + \bar{\mathbf{v}}_k, & \bar{\mathbf{v}}_k &\sim N(\bar{\mathbf{0}}, R_k),\end{aligned} \tag{C.34}$$

which is equivalent to Equations (C.10) – (C.11) for $\Phi_k = H_k = Q_k = I$. $R_k$ is set to the singular matrix

$$R_k = \begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix} \tag{C.35}$$

which gives a correlation coefficient of 1 between $z_1$ and $z_2$. This means that the error in both directions is believed to be of exactly the same size. If the next measurement is $\bar{\mathbf{z}}_k = \begin{pmatrix}2 & 3\end{pmatrix}^T$ for instance, the above $R_k$ means that the correct solution must lie on the line through $\begin{pmatrix}2 & 3\end{pmatrix}^T$ with slope 1: $x_2 = x_1 + 1$. Assuming $P_k^- = I$, $K_k$ can be calculated as

$$\begin{aligned}K_k &= P_k^- H_k^T \left[H_k P_k^- H_k^T + R_k\right]^{-1} = \\ &= II\left[III + R_k\right]^{-1} = \\ &= \begin{bmatrix}2 & 1\\ 1 & 2\end{bmatrix}^{-1} = \\ &= \frac{1}{3}\begin{bmatrix}2 & -1\\ -1 & 2\end{bmatrix}^{-1}.\end{aligned} \tag{C.36}$$

If the previous state prediction $\hat{\mathbf{x}}_k^- = \begin{pmatrix}8 & 3\end{pmatrix}^T$, the new estimate will become

$$\begin{aligned}\hat{\mathbf{x}}_k^+ &= \hat{\mathbf{x}}_k^- + K_k(\bar{\mathbf{z}}_k - H_k\hat{\mathbf{x}}_k^-) = \\ &= \begin{pmatrix}8\\3\end{pmatrix} + \frac{1}{3}\begin{bmatrix}2 & -1\\ -1 & 2\end{bmatrix}\left\{\begin{pmatrix}2\\3\end{pmatrix} - \begin{pmatrix}8\\3\end{pmatrix}\right\} = \\ &= \begin{pmatrix}8\\3\end{pmatrix} + \frac{1}{3}\begin{pmatrix}-12\\6\end{pmatrix} = \\ &= \begin{pmatrix}4\\5\end{pmatrix},\end{aligned} \tag{C.37}$$

which is on the line $x_2 = x_1 + 1$. The situation is illustrated in Figure C.1.

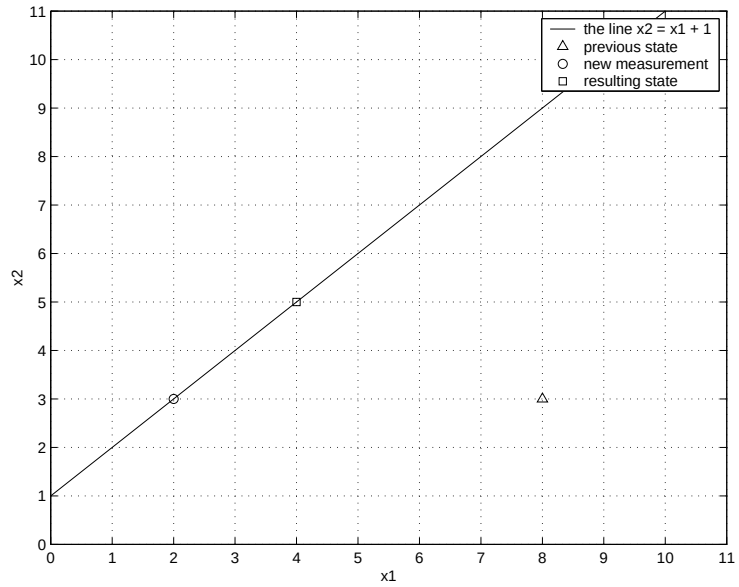



Figure C.1: *The singular $R_k$ forces the state to lie on the line $x_2 = x_1 + 1$. Due to the measurement (circle), the state estimate jumps from the old location (triangle) to a point on the line (square).*

In summary, a singular $R_k$ will constrain the state vector $\hat{\mathbf{x}}$ to a subspace of the solution space. It will be locked there for a time that depends on $Q_k$. Similar problems occur when $R_k$ is close to singular. The situation can be avoided by choosing $R_k$ wisely. For extended Kalman filters, this is even more important, since a displaced state vector will mean that the linearizations performed in the next step will be wrong, which could make the state estimation error diverge. A good rule of thumb is that the standard deviation in any direction of $R_k$ never should be smaller than one pixel.

# Appendix D

# Variance Calculations

In Section 10.3.1, the covariance matrix of the noise is assumed to be known up to a constant $\xi$. Given a measurement of the summed squared error $\sqrt{\text{SSD}}$, a Maximum Likelihood estimate of $\xi$ is obtained. This chapter will go through in detail how this is done, both for the four-dimensional case and for the two-dimensional case.

## D.1 Four-Dimensional Case

Presume

$$S = \Delta\mathbf{X}^T A \Delta\mathbf{X}, \tag{D.1}$$

where $A$ is a symmetric positive definite 4×4 matrix and $\Delta\mathbf{X} = (X_1, X_2, X_3, X_4)$. Moreover, assume that $\Delta\mathbf{X} \sim N(0, \xi A^{-1})$ where $\xi$ is a positive scalar. The cumulative distribution function (cdf) of $S$ will then be

$$\begin{aligned} F_S(s) =& Pr\{S < s\} = Pr\{\Delta\mathbf{X}^t A \Delta\mathbf{X} < s\} = \\ =& \int\limits_{\mathbf{x}^T A\mathbf{x} < s} \frac{1}{(2\pi)^2 |\xi A^{-1}|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathbf{x}^T (\xi A^{-1})^{-1}\mathbf{x}} d\mathbf{x} = \\ =& \int\limits_{\mathbf{x}^T A\mathbf{x} < s} \frac{1}{(2\pi)^2 \xi^2 |A|^{-\frac{1}{2}}} e^{-\frac{1}{2\xi}\mathbf{x}^T A\mathbf{x}} d\mathbf{x}. \end{aligned} \tag{D.2}$$

Since $A$ is positive definite and symmetrical, Cholesky factorization $A = Q^T Q$ can be used. Using the variable substitution $\mathbf{y} = Q\mathbf{x}$ with the Jacobian $|\frac{\partial \mathbf{x}}{\partial \mathbf{y}}| = |Q^{-1}| = |A|^{-\frac{1}{2}}$, this is transformed to

$$\int\limits_{\mathbf{y}^T\mathbf{y} < s} \frac{1}{(2\pi)^2 \xi^2} e^{-\frac{1}{2\xi}\mathbf{y}^T\mathbf{y}} d\mathbf{y}. \tag{D.3}$$

Transforming to polar coordinates $(y_1, y_2) = (r_1 \cos\varphi_1, r_1 \sin\varphi_1)$ and $(y_3, y_4) = (r_2 \cos\varphi_2, r_2 \sin\varphi_2)$ with the Jacobian $|\frac{\partial \mathbf{y}}{\partial \mathbf{r}\varphi}| = r_1 r_2$ yields

$$\int_0^{2\pi}\int_0^{2\pi}\int_V \frac{r_1 r_2}{(2\pi)^2\xi^2} e^{-\frac{1}{2\xi}r_1^2+r_2^2} dr_1 dr_2 d\varphi_1 d\varphi_2, \tag{D.4}$$

where $V$ is the area $\{r|r_1^2 + r_2^2 < s, r_1 > 0, r_2 > 0\}$. Integrating over $\varphi_1$ and $\varphi_2$ and parameterizing $V$ gives

$$\begin{aligned}
&\int_0^{\sqrt{s}}\int_0^{\sqrt{s-r_1^2}} \frac{r_1 r_2}{\xi^2} e^{-\frac{1}{2\xi}r_1^2+r_2^2} dr_1 dr_2 = \int_0^{\sqrt{s}} \left[-\frac{r_1}{\xi} e^{-\frac{1}{2\xi}r_1^2+r_2^2}\right]_0^{\sqrt{s-r_1^2}} dr_1 = \\
&= \int_0^{\sqrt{s}} -\frac{r_1}{\xi} e^{-\frac{s}{2\xi}} + \frac{r_1}{\xi} e^{-\frac{1}{2}r_1^2} dr_1 = \left[-\frac{r_1^2}{2\xi} e^{-\frac{s}{2\xi}} - e^{-\frac{1}{2}r_1^2}\right]_0^{\sqrt{s}} = \\
&= -\frac{s}{2\xi} e^{-\frac{s}{2\xi}} - e^{-\frac{s}{2}} + 0 + 1 = 1 - (1 + \frac{s}{2\xi}) e^{-\frac{s}{2\xi}}.
\end{aligned} \tag{D.5}$$

Thus the cdf of $S$ is $F_S(s) = 1 - (1 + \frac{s}{2\xi})e^{-\frac{s}{2\xi}}$, the pdf is $\frac{\partial}{\partial s} F_S(s)$, i.e.,

$$\begin{aligned}
&\frac{\partial}{\partial s}(1 - (1 + \frac{s}{2\xi})e^{-\frac{s}{2\xi}}) = -(1 + \frac{s}{2\xi})(-\frac{1}{2\xi})e^{-\frac{1}{2\xi}s} - (\frac{1}{2\xi})e^{-\frac{s}{2\xi}} = \\
&= -\frac{1}{2\xi} e^{-\frac{s}{2\xi}} \left[1 - (1 + \frac{s}{2\xi})\right] = \frac{s}{4\xi^2} e^{-\frac{s}{2\xi}}.
\end{aligned} \tag{D.6}$$

### D.1.1 Maximum Likelihood

Using the pdf of $S$ and a measurement of $s$, the parameter $\xi$ of the distribution can be estimated using Maximum Likelihood,

$$\xi^* = \underset{\xi}{\operatorname{argmax}}\, L(\xi). \tag{D.7}$$

Since only one measurement is available, the (log-) likelihood function will be

$$L(\xi) = \ln f_S(s) = \ln(\frac{s}{4\xi^2} e^{-\frac{1}{2\xi}s}) = \ln(\frac{s}{4\xi^2}) - \frac{s}{2\xi}. \tag{D.8}$$

Differentiating $L(\xi)$ and setting to zero

$$\begin{aligned}
L'(\xi^*) =& \frac{1}{(\frac{s}{4\xi^{*2}})}(\frac{s}{4}(-2)\xi^{*-3}) - \frac{1}{2}(-1)\xi^{*-2}s = \\
=& -\frac{2}{\xi^*} + \frac{s}{2\xi^{*2}} = \\
=& \frac{1}{\xi^*}(\frac{s}{2\xi^*} - 2) = 0
\end{aligned} \tag{D.9}$$

gives

$$\frac{s}{2\xi^*} - 2 = 0 \tag{D.10}$$

and thus

$$\xi^* = \frac{1}{4}s. \tag{D.11}$$

The solution $\xi^* = \frac{s}{4}$ is a maximum since

$$L''(\xi) = \frac{1}{\xi^2}(2 - \frac{s}{\xi}) \tag{D.12}$$

so

$$L''(\xi^*) = L''(\frac{s}{4}) = \frac{16}{s^2}(2-4) = -\frac{32}{s^2} < 0 \tag{D.13}$$

for any $s$.

### D.1.2 Bias

To investigate whether the estimate $\xi^* = \frac{1}{4}s$ from Equation (D.11) is biased, the expected value of $S$ is calculated.

$$E[S] = \int_0^\infty s\frac{s}{4\xi^2}e^{-\frac{s}{2\xi}}ds. \tag{D.14}$$

By partial integration twice this can be transformed to

$$E[S] = \int_0^\infty 2e^{-\frac{s}{2\xi}}ds = \left[2(-2\xi)e^{-\frac{s}{2\xi}}\right]_0^\infty = 4\xi. \tag{D.15}$$

The expected value of Equation (D.11) equals

$$E[\xi^*] = E\left[\frac{1}{4}S\right] = \frac{1}{4}4\xi = \xi, \tag{D.16}$$

and the estimate is thus unbiased.

## D.2 Two-Dimensional Case

The two dimensional case is similar, but less complex. Presume

$$S = \Delta\mathbf{X}^T A \Delta\mathbf{X}, \tag{D.17}$$

where $A$ is a symmetric positive definite $2 \times 2$ matrix and $\Delta\mathbf{X} = (X_1, X_2)$. Moreover, assume that $\Delta\mathbf{X} \sim N(0, \xi A^{-1})$ where $\xi$ is a positive scalar. The

cumulative distribution function (cdf) of $S$ will then be

$$\begin{aligned} F_S(s) =& Pr\{S < s\} = Pr\{\Delta\mathbf{X}^t A \Delta\mathbf{X} < s\} = \\ =& \int\limits_{\mathbf{x}^T A\mathbf{x} < s} \frac{1}{(2\pi)|\xi A^{-1}|^{\frac{1}{2}}} e^{-\frac{1}{2}\mathbf{x}^T(\xi A^{-1})^{-1}\mathbf{x}} d\mathbf{x} = \\ =& \int\limits_{\mathbf{x}^T A\mathbf{x} < s} \frac{1}{(2\pi)\xi|A|^{-\frac{1}{2}}} e^{-\frac{1}{2\xi}\mathbf{x}^T A\mathbf{x}} d\mathbf{x}. \end{aligned} \tag{D.18}$$

Since $A$ is positive definite and symmetrical, Cholesky factorization $A = Q^T Q$ can be used. Using the variable substitution $\mathbf{y} = Q\mathbf{x}$ with the Jacobian $|\frac{\partial \mathbf{x}}{\partial \mathbf{y}}| = |Q^{-1}| = |A|^{-\frac{1}{2}}$, this is transformed to

$$\int\limits_{\mathbf{y}^T\mathbf{y} < s} \frac{1}{(2\pi)\xi} e^{-\frac{1}{2\xi}\mathbf{y}^T\mathbf{y}} d\mathbf{y}. \tag{D.19}$$

Transforming to polar coordinates $(y_1, y_2) = (r\cos\varphi, r\sin\varphi)$ with the Jacobian $|\frac{\partial y_1 y_2}{\partial r\varphi}| = r$ yields

$$\int_0^{2\pi}\int_0^{\sqrt{s}} \frac{r}{(2\pi)\xi} e^{-\frac{1}{2\xi}r^2} dr d\varphi, = \int_0^{\sqrt{s}} \frac{r}{\xi} e^{-\frac{1}{2\xi}r^2} = \left[-e^{-\frac{1}{2\xi}r^2}\right]_0^{\sqrt{s}} = \tag{D.20}$$

$$= \begin{cases} 1 - e^{-\frac{s}{2\xi}}, & s \geq 0 \\ 0, & s < 0 \end{cases}. \tag{D.21}$$

Differentiating with respect to $s$ gives $f_S(s) = \frac{1}{2\xi} e^{-\frac{s}{2\xi}}$ , which is an exponential distribution with mean $2\xi$.

### D.2.1 Maximum Likelihood

Using the pdf of $S$ and a measurement of $s$, the parameter $\xi$ of the distribution can be estimated using Maximum Likelihood,

$$\xi^* = \underset{\xi}{\operatorname{argmax}}\, L(\xi). \tag{D.22}$$

with the (log-) likelihood function

$$L(\xi) = \ln f_S(s) = \ln \frac{1}{2\xi} e^{-\frac{s}{2\xi}} = \ln(\frac{1}{2\xi}) - \frac{s}{2\xi}. \tag{D.23}$$

Differentiating $L(\xi)$ and setting to zero

$$\begin{aligned} L'(\xi^*) =& \frac{1}{(\frac{1}{2\xi^*})}(-\frac{1}{2\xi^{*2}}) + \frac{s}{2\xi^{*2}} \\ =& -\frac{1}{2\xi^{*2}}(2\xi^* - s) = 0 \end{aligned} \tag{D.24}$$

gives

$$\begin{aligned} 2\xi^* - s &= 0 \\ \xi^* &= \frac{s}{2}. \end{aligned} \tag{D.25}$$

This solution is a maximum since

$$L''(\xi) = \frac{1}{\xi^2}(1 - \frac{s}{\xi}) \tag{D.26}$$

so

$$L''(\xi^*) = L''(\frac{s}{2}) = \frac{4}{s^2}(1-2) = -\frac{4}{s^2} < 0 \tag{D.27}$$

for any $s$.

### D.2.2 Bias

The expected value of Equation (D.11) equals

$$E\left[\xi^*\right] = E\left[\frac{1}{2}S\right] = \frac{1}{2}2\xi = \xi, \tag{D.28}$$

and the estimate is thus unbiased.

# Bibliography


[1] Andrea Dell'Acqua, Augusto Sarti, Stefano Tubaro, "Effective Analysis of Image Sequences for 3D Camera Motion Estimation," *Proceedings of the EuroImage ICAV3D 2001 Conference*, Greece, pp. 307-310, May 2001.

[2] Gilad Adiv, "Determining Three-Dimensional Motion and Structure from Optical Flow Generated by Several Moving Objects," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 7, no. 4, pp. 384-401, July 1985.

[3] J. Ahlberg and H. Li, "Representing and compressing MPEG-4 facial animation parameters using facial action basis functions," *Technical Report*, LiTH-ISY-R-2010, ISSN 1400-3902, Linköping University, 1998.

[4] J. Ahlberg, "Facial Feature Tracking using an Active Appearance-driven Model," *Proceedings of the EuroImage ICAV3D 2001 Conference*, Greece, pp. 116-119, May 2001.

[5] K. Aizawa, H. Harashima, and T. Saito, "Model-based image coding system — construction of a 3-D model of a person's face," *Proc. Int. Picture Coding Symp.*, Stockholm, Sweden, 1987, paper 3.11.

[6] Harry C. Andrews and Claude L. Patterson, "Singular value decomposition (SVD) image coding," *IEEE Transactions on Communications*, pp. 425–432, April 1976.

[7] Kalle Åström, *Invariancy Methods for Point, Curves and Surfaces in Computational Vision*, Ph. D. Thesis, Departement of Mathematics, Lund University, Sweden 1996.

[8] A. Azarbayejani and A. Pentland, "Recursive Estimation of Motion, Structure, and Focal Length," *IEEE Pattern Analysis and Machine Intelligence*, vol. 17, no. 6, pp. 562-575, June 1995.

[9] A. Azarbayejani, T. Starner, B. Horowitz, Alex P. Pentland "Visually Controlled Graphics," *IEEE Pattern Analysis and Machine Intelligence*, vol. 15, no. 6, June 1993.

[10] J. L. Barron, D. J. Fleet and S. S. Beauchemin, "Performance of Optical Flow Techniques," *International Journal of Computer Vision*, vol. 12 no. 1, pp. 43-77, 1994.

[11] S. Basu, I. Essa and A. Pentland, "Motion regularization for model-based head tracking," *Proceedings of ICPR*, pp. 611-616, 1996.

[12] M. Bichsel and A. Pentland, "Human face recognition and the face image set's topology," *CVGIP; Image Understanding*, vol. 59, no. 2, March, pp. 254–261, 1994.

[13] M. J. Black and Y. Yacoob, "Recognizing facial expressions in image sequences using local parameterized models of image motion," *International Journal of Computer Vision*, 25(1), pp. 23-48, 1997.

[14] C. Bregler and Y. Konig, "Eigenlips for Robust Speech Recognition," *Proc. of IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Adelaide, Australia, 1994.

[15] Ted J. Broida, S. Chandrashekhar and Rama Chellappa. "Recursive estimation of 3-d motion from a monocular image sequence," *IEEE Trans. Aerop. Electron. Syst.,* 26(4):639-656, July 1990.l

[16] M. L. Cascia, S. Sclaroff and V. Athitsos "Fast, Reliable Head Tracking under Varying Illumination: An Approach Based on Registration of Texture-Mapped 3D Models," *IEEE Transactions on Pattern Analyis and Machine Intelligence*, vol. 22, no. 4, pp. 322-336, April 2000.

[17] T.F.Cootes, G.J. Edwards and C.J.Taylor. "Active Appearance Models," in *Proc. European Conference on Computer Vision 1998* (H.Burkhardt & B. Neumann Ed.s). vol. 2, pp. 484-498, Springer, 1998.

[18] N.P. Costen, I.G. Craw, G.J. Robertsona and S. Akamatsu "Automatic face recognition: What representation?," *Computer Vision — Proceedings of ECCV'96*, vol. I, no. 1064, in *Lecture Notes on Computer Science*, Springer-Verlag, pp. 504–513, 1996.

[19] J. L. Crowley and F. Berard, "Multi-Modal Tracking of Faces for Video Communications," In *Proceedings of IEEE Conference in Computer Vision and Pattern Recognition, CVPR '97*, San Juan, June 1997.

[20] Douglas DeCarlo and Dimitris Metaxas, "The Integration of Optical Flow and Deformable Models with Applications to Human Face Shape and Motion Estimation," *Proceedings of CVPR '96*, pp. 231-238, 1996.

[21] O. D. Faugeras, "What can be seen in three dimensions with an uncalibrated stereo rig," In G. Sandini, editor, *Computer Vision – Proc. 2. ECCV*, pp. 563–578. Springer Verlag, May 1992.

[22] O. Faugeras, *Three-Dimensional Computer Vision: A Geometric Viewpoint*, MIT Press, 1993

[23] R. Forchheimer and O. Fahlander, "Low bit-rate coding through animation," *Proc. Int. Picture Coding Symp.*, Davis, CA, pp. 113–114, 1983.

[24] R. Forchheimer, O. Fahlander and T. Kronander, "A semantic approach to the transmission of face images," *Proc. Int. Picture Coding Symp.*, Cesson-Sevigne, France, paper 10.6, 1984.

[25] K. Fukunaga, *Introduction to statistical pattern recognition*, Second Edition, San Diego: Academic Press, 1990.

[26] A. Gelb (editor) *Applied Optimal Estimation* Cambridge, MA: The MIT Press, 1974.

[27] G. D. Hager and P. N. Buelhumeur, "Real-Time Tracking of Image Regions with Changes in Geometry and Illumination," *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 403-410, 1996.

[28] R. I. Hartley, "Ambigous Configurations for 3-View Projective Reconstruction," *Sixth European Conference on Computer Vision*, Dublin, Ireland, pp. 922-935, June/July 2000.

[29] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, ISBN 0-521-62304-9, Cambridge University Press, 2000.

[30] M. Harville, A. Rahimi, T. Darell, G. Gordon, J. Woodfill, *The Proceedings of the Seventh IEEE International Conference on Computer Vision, 1999*, vol. 1, pp. 206-213, 1999.

[31] Anders Heyden, *Geometry and Algebra of Multiple Projective Transformations*, Ph.D. Thesis, Department of Mathematics, Lund University, Sweden 1995.

[32] Anders Heyden and Kalle Åström, "Computer Vision," lecture notes for the VISIT-course, Centre for Mathematical Sciences, Lund University, 1999.

[33] ISO/IEC Standard 14496-2, *Information Technology — Coding of audio-visual objects — Part 2: Video.* International Standard, First edition 1998-10.

[34] T. Jebara, A. Azarbayejani and A. Pentland, "3D Structure from 2D motion," *IEEE Signal Processing Magazine,*, pp. 66-84, May 1999.

[35] T. Jebara and A. Pentland, "Parametrized Structure from Motion for 3D Adaptive Feedback Tracking of Faces," *IEEE Conference on Computer Vision and Pattern Recognition* (CVPR'97), 1997.

[36] Tony Jebara, Kenneth Russel and Alex Pentland "Mixtures of eigenfaces for real-time structure from texture," *Proceedings of ICCV'98*, Bombay, India, January 4–7, 1998.

[37] I. T. Jolliffe, *Principal component analysis.* Springer-Verlag New York Inc., 1986.

[38] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems," *Journal of Basic Engineering* (ASME), vol. 82D, pp. 35-45, March 1960.

[39] M. Kirby and L. Sirovich, "Application of the Karhunen-Loève procedure for the characterization of human faces," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 1, January 1990.

[40] H. Li, *Low Bitrate Image Sequence Coding*, Ph.D. Thesis No 318, Linkoping University, Sweden 1993

[41] H. Li, A. Lundmark and R. Forchheimer, "Image sequence coding at very low bit rates: A review," *IEEE Trans. Image Proc.*, vol. 3, pp. 589–609, Sept 1994.

[42] H. Li and R. Forchheimer, "Two-view facial movement estimation," *IEEE Trans. Circuits and Systems for Video Technology*, vol. 4, no. 3, June 1994.

[43] H. Li, P. Roivainen, and R. Forchheimer, "3-D motion estimation in model-based facial image coding," *IEEE Trans. on PAMI*, vol. 15, no. 6, pp. 545-555, June 1993.

[44] H.C. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, 293:133–135, 1981.

[45] K. Mase and A. Pentland, "LIP READING: Automatic Visual Recognition of Spoken Words," Proc. Image Understanding and Machine Vision, Optical Society of America, June 1989.

[46] Y. Matsumoto, A. Zelinsky, "Real-time Face Tracking System for Human-Robot Interaction," *Proceedings of 1999 IEEE International Conference on Systems, Man, and Cybernetics* (SMC'99), pp. II-830-II-835, Tokyo, Japan, October 12-15, 1999.

[47] B. Moghaddam and A. Pentland, "An automatic system for model-based coding of faces," *IEEE Data Compression Conference*, Snowbird, Utah, March 1995.

[48] B. Moghaddam and A. Pentland, "Probabilistic Visual Learning for Object Representation," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, no. 7, pp. 696 - 710, July 1997.

[49] D. Nistér, *Automatic Dense Reconstruction from Uncalibrated Video Sequences*, Ph.D. Thesis, Royal Institute of Technology, Sweden, March 2001.

[50] N. Oliver, A. Pentland and F. Berard "LAFTER: A Real-time Lips and Face Tracker with Facial Expression Recognition," *Proceedings of CVPR'97* San Juan, Puerto Rico, June 1997.

[51] D. Pearson, "Developments in model-based video coding," *Proc. IEEE*, vol. 83, no. 6, pp. 829–906, 1995.

[52] A. Pentland, B. Moghaddam, T. Starner "View-based and modular eigenspaces for face recognition," *IEEE Conference on Computer Vision & Pattern Recognition*, 1994.

[53] P. Roivainen, "Motion estimation in model-based coding of human faces," Ph.D. Thesis LIU-TEK-LIC-1990:25, ISY, Linköping Univ., Sweden, 1990.

[54] M. Rydfalk, "CANDIDE, a parameterized face," *Technical Report*, LiTH-ISY-I-0866, Linköping University, 1987.

[55] A. Said and W.A. Pearlman, "A new fast and efficient image codec based on set partitioning in hierarchical trees," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 6, no. 3, pp. 243–250, June 1996.

[56] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* vol. 27, pp. 379—423, 623—656, July, October, 1948.

[57] J. Shapiro. "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Transactions on Signal Processing*, vol. 41, no. 12, pp. 3445–3462, December 1993.

[58] A. Shashua and M. Werman, "Trilinearity of Three Perspective Views and its Associated Tensor," in International Conf. on Computer Vision, Cambridge, MA, pp. 920-925, 1995.

[59] L. Sirovich and M. Kirby, "Low-dimensional procedure for the characterization of human faces" *J. Opt. Soc. Am.*, vol. 4, no. 3 pp. 519–524, March 1987.

[60] S. Spanne, *Föreläsningar i MATRISTEORI*, Department of Mathematics, Lund Institute of Technology, Printed in Lund, 1993.

[61] G. P. Stein and A. Shashua, "On Degeneracy of Linear Reconstruction from Three Views: Linear Line Complex and Applications," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 21(3):244-251, 1999.

[62] G. Strang, *Linear algebra and its applications,* New York: Academic Press, 1980.

[63] J. Ström, F. Davoine, J. Ahlberg, H. Li and R. Forchheimer. "Very Low Bit Rate Facial Texture Coding," *International Workshop on Synthetic - Natural Hybrid Coding and Three Dimensional Imaging*, Rhodes, Greece, September 1997.

[64] J. Ström, "Facial Texture Compression for Model-Based Coding," Licentiate Thesis, LIU-TEK-LIC-1998-51, ISY, Linköping University, Sweden, August 1998.

[65] J. Ström and P. C. Cosman, "Medical Image Compression with Lossless Regions of Interest," *Signal Processing*, vol. 59, no. 2, June 1997.

[66] J. Ström, T. Jebara, S. Basu, and A. Pentland, "Real Time Tracking and Modeling of Faces: An EKF-based Analysis by Synthesis Approach," *Proceedings of the Modelling People Workshop* at ICCV'99, August 1999.

[67] J. Ström, T. Jebara and A. Pentland, "Model-Based Real-Time Face Tracking with Adaptive Texture Update," Technical Report, LiTH-ISY-R-2342, Linköping University, Sweden, March 30, 2001.

[68] J. Ström, "Reinitialization of a Model-Based Face Tracker," *Proceedings of the EuroImage ICAV3D 2001 Conference*, May 2001.

[69] J. Ström, "Structure from Motion of Planar Surfaces, (Analysis of an EKF Based Approach)," *Proceedings of Symposium on Image analysis, Swedish Society for Automated Image Analysis*, pp. 13-16, Norrköping, Sweden, March 2001.

[70] P. Torr, W. Fitzgibbon and A. Zisserman, "Maintaining multiple motion model hypetheses over many views to recover matching and structure," in

*Proc. of Sixth Int. Conf. on Computer Vision*, Bombay, India, pp. 485-491, Jan 1998.

[71] B. Triggs, "Autocalibration from planar scenes," in *Proceedings of the 5th European Conference on Computer Vision* (ECCV'98), Freiburg, Germany, pp. 89–105, June 1998.

[72] M. Turk and A. Pentland, "Eigenfaces for recognition," *J. of Cognitive Neuroscience*, vol. 3, no. 1, pp. 71–86, 1991.

[73] G. K. Wallace, "The JPEG still picture compression standard," Commun. ACM, vol. 34, no. 4, pp. 30–40, Apr. 1991.

[74] W.J. Welsh, "Model-based coding of moving images at very low bit rates," *Proc. Int. Picture Coding Symp.*, Stockholm, Sweden, paper 3.9, 1987.

[75] W. Wunderlich, "Rechnerische Rekonstruktion eines Ebenen Objekts aus Zwei Photographien," *Mitteilungen der Geodätischen Institute der Technischen Universität Graz*, Folge 40 (Festschrift K. Rimmer zum 70. Geburtstag), pp. 365–377.

[76] Ye Zhang and Chandra Kambhamettu, "Robust 3D Head Tracking Under Partial Occlusion," *Fourth International Conference on Face and Gesture Recognition*, pp. 176-182, Grenoble, France, March 2000.