Stochastic visual tracking with active appearance models

by

McElory Roberto Hoffmann

Dissertation approved for the degree of Doctor of Philosophy in Engineering at Stellenbosch University

Department of Mathematical Sciences (Division Applied Mathematics)
University of Stellenbosch
Private Bag X1, Matieland, 7602, South Africa

Promoters:

Prof. B. Herbst Dr. K. Hunter Prof. L. van Zijl Prof. J. du Preez

December 2009

Declaration

By submitting this dissertation electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the owner of the copyright thereof (unless to the extent explicitly otherwise stated) and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Signature: ....
M.R. Hoffmann

Date: 17 November 2009 ....

Abstract

Stochastic visual tracking with active appearance models

M.R. Hoffmann

Department of Mathematical Sciences (Division Applied Mathematics)
University of Stellenbosch
Private Bag X1, Matieland, 7602, South Africa

Dissertation: PhD (Applied Mathematics)

December 2009

In many applications, an accurate, robust and fast tracker is needed, for example in surveillance, gesture recognition, tracking lips for lip-reading and creating an augmented reality by embedding a tracked object in a virtual environment. In this dissertation we investigate the viability of a tracker that combines the accuracy of active appearance models with the robustness of the particle filter (a stochastic process)—we call this combination the PFAAM. In order to obtain a fast system, we suggest local optimisation as well as using active appearance models fitted with non-linear approaches.

Active appearance models use both contour (shape) and greyscale information to build a deformable template of an object. They are typically accurate, but not necessarily robust, when tracking contours. A particle filter is a generalisation of the Kalman filter. In a tutorial style, we show how the particle filter is derived as a numerical approximation for the general state estimation problem.

The algorithms are tested for accuracy, robustness and speed on a PC, in an embedded environment and by tracking in 3D. The algorithms run real-time on a PC and near real-time in our embedded environment. In both cases, good accuracy and robustness is achieved, even if the tracked object moves fast against a cluttered background, and for uncomplicated occlusions.

Uittreksel

Stochastiese volg van voorwerpe met aktiewe voorkomsmodelle

("Stochastic visual tracking with active appearance models")

M.R. Hoffmann

Departement Wiskunde Wetenskappe (Afdeling Toegepaste Wiskunde)
Universiteit van Stellenbosch
Privaatsak X1, Matieland, 7602, Suid Afrika

Proefskrif: PhD (Toegepaste Wiskunde)

Desember 2009

'n Akkurate, robuuste en vinnige visuele-opspoorder word in vele toepassings benodig. Voorbeelde van toepassings is bewaking, gebaarherkenning, die volg van lippe vir liplees en die skep van 'n vergrote realiteit deur 'n voorwerp wat gevolg word, in 'n virtuele omgewing in te bed. In hierdie proefskrif ondersoek ons die lewensvatbaarheid van 'n visuele-opspoorder deur die akkuraatheid van aktiewe voorkomsmodelle met die robuustheid van die partikelfilter ('n stochastiese proses) te kombineer—ons noem hierdie kombinasie die PFAAM. Ten einde 'n vinnige visuele-opspoorder te verkry, stel ons lokale optimering, sowel as die gebruik van aktiewe voorkomsmodelle wat met nie-lineêre tegnieke gepas is, voor.

Aktiewe voorkomsmodelle gebruik kontoer (vorm) inligting tesame met grysskaalinligting om 'n vervormbare meester van 'n voorwerp te bou. Wanneer aktiewe voorkomsmodelle kontoere volg, is dit normaalweg akkuraat, maar nie noodwendig robuust nie. 'n Partikelfilter is 'n veralgemening van die Kalmanfilter. Ons wys in tutoriaalstyl hoe die partikelfilter as ’n numeriese benadering tot die toestand-beramingsprobleem afgelei kan word.

Die algoritmes word vir akkuraatheid, robuustheid en spoed op ’n persoonlike rekenaar, ’n ingebedde omgewing en deur volging in 3D, getoets. Die algoritmes loop intyds op ’n persoonlike rekenaar en is naby intyds op ons ingebedde omgewing. In beide gevalle, word goeie akkuraatheid en robuustheid verkry, selfs as die voorwerp wat gevolg word, vinnig, teen ’n besige agtergrond beweeg of eenvoudige okklusies ondergaan.

Acknowledgements

I would like to gratefully and sincerely thank Prof. Ben Herbst and Dr. Karin Hunter for their supervision, understanding, patience, and friendship from the beginning of my studies until now. Without them, this dissertation would not have been possible. Thanks Karin for always walking the extra mile with me.

Many parts of this work has been completed at WSI/GRIS, University of Tübingen. It is therefore a pleasure to thank Prof. Wolfgang Straßer for being such a gracious host and for the many discussions we had. I owe my gratitude to Dr. Sven Fleck with whom I worked to complete many parts of this work.

I am pleased to thank Prof. Lynette van Zijl for many discussions and proof-reading parts of this dissertation.

I am heartily thankful to Albert Swart for many conversations and his co-operation in work on the smart camera. Many thanks to Dr. Willie Brink for his collaboration on the 3D-tracker.

Many thanks to Dr. Jason Saragih for his implementations of active appearance models.

Thanks to Jacques, Leendert, Markos, François and Wynand for allowing me to use them in the images in this dissertation.

I am thankful to Dr. Danie Els for answering all my LaTeX questions.

I offer my regards and blessings to all my friends who supported me in the completion of my dissertation. Thanks to Heinrich for many discussions and editorial help in this dissertation, Lourens for many late hours discussions, and Jock for all the conversations we had. Thanks also to Johann, Henning, Nelia, Aldi, Anne-Marie, Cobus, Retha, Magdaleen, Lize-Marie, Tinus, Almatine, Francis, Hanri, Hannes, Shaun, Jaco Jacobs, Jaco Fourie, Linda, and the rest that slipped my mind.

I would like to thank George for his encouragement and support throughout my studies. This is also a great opportunity to express my thanks to my brothers (Len and Keith) and my sisters (Lizette, Allison, and Susan) for their support.

Lastly, I owe my deepest gratitude to my parents for their support, encouragement, and the fact that they believe in me.

The financial assistance of the South African National Research Foundation towards this research is gratefully acknowledged. I would like to thank the University of Tübingen's International office and the Landesstiftung Baden Wurttemberg for their financial assistance that made the visit to WSI/GRIS possible. Many thanks to the Division for Research Development, University of Stellenbosch for the Research Capacity Building Fund. Also my gratitude towards Matrix Vision for the successful cooperation. Opinions expressed and conclusions arrived at, are those of the author and are not necessarily to be attributed to any of the institutions.

Deo Gratias.

Dedications

Vir Mamma en Pappa

Brief contents

Contents

Nomenclature

Acronyms

AAM	Active appearance model
ASM	Active shape model
AC	Active contour
PFAAM	AAM-based particle filter tracker
AC-AAM	Active contour-active appearance model tracker
PF	Particle filter
svd	Singular value decomposition
pdf	Probability density function
PCA	Principal component analysis

Linear algebra

x	Scalar value x
$\boldsymbol{x}$	Column vector $\boldsymbol{x}$
A	Matrix A
A^{T}	Transpose of the matrix A
λ_i	The ith eigenvalue
$\lVert\boldsymbol{x}\rVert_p$	The pth norm of the vector $\boldsymbol{x}$
I	Identity matrix

$\mathcal{I}$	An image

Probability and state estimation

$p(x)$	Probability density function over x
$\mathbb{E}[f]$	Expected value of f
$\mathrm{Cov}[f]$	Covariance of f
$\mathcal{N}(\boldsymbol{m}, \mathrm{C})$	Normal distribution with mean $\boldsymbol{m}$ and covariance C
$\mathrm{Unif}(a, b)$	Uniform distribution over the interval $[a, b]$
$\boldsymbol{x}_t$	Vector $\boldsymbol{x}$ at time step t
$\widehat{\boldsymbol{x}}_t$	Estimate of $\boldsymbol{x}$ at time step t
$\boldsymbol{x}_t^{(i)}$	Value of $\boldsymbol{x}$ at time step t for the ith particle
$w_t^{(i)}$	Weight for particle i at time t
$\widehat{\mathrm{N}}_{\mathrm{eff}}$	Effective sampling size of a PF

Active appearance models

$\boldsymbol{p}_p$	AAM global pose vector
$\boldsymbol{p}_s$	AAM shape vector
$\boldsymbol{p}_g$	AAM texture vector
$\boldsymbol{p}_t$	AAM global texture normalisation
$\boldsymbol{p}_a$	Generic AAM parameters
$\mathcal{W}(\boldsymbol{j}; \boldsymbol{p})$	Warping function. The parameters $\boldsymbol{p}$ are needed to construct the warp and $\boldsymbol{j}$ is the index in the resulting image.
$\bar{\boldsymbol{s}}$	Mean shape
Φ_s	AAM shape deformation matrix
Φ_g	AAM texture deformation matrix
$\mathcal{S}(\boldsymbol{p}_s)$	AAM shape generating function
$\bar{\boldsymbol{g}}$	Mean texture
$\mathcal{G}(\boldsymbol{p}_g)$	AAM texture generating function

Chapter

1

Introduction

There are two mistakes one can make along the road to truth... not going all the way, and not starting.

— Gautama Siddharta

Tracking objects in a video sequence is one of the fundamental problems in computer vision, and can be one of the toughest. Tracking a single, rigid object against a smooth background is relatively easy. Tracking multiple, deformable objects through occlusions against cluttered backgrounds, is difficult. Yet it is an essential part of many applications.

We need computers to be able to track in order to automate applications, such as following a suspect in surveillance (Hu et al., 2004), gesture recognition (Wu and Huang, 1999), tracking lips for lip-reading (Matthews et al., 2002), and creating an augmented reality by embedding a tracked object in a virtual environment (Fischer et al., 2007). Most applications need accurate, robust and fast tracking before they become useful. For example, in the case of lip-reading, we need real-time translation, and since the recognition algorithms will take the tracker's output as input, the tracker needs to be accurate, robust and fast. Our aim in this dissertation is to provide and investigate such a tracker.

Ideally one would prefer to track arbitrary objects. This has proved to be too difficult and the emphasis has shifted to tracking specific objects where the basic shape and deformations are known, or can be learnt, see (Blake and Isard, 1998). This is the approach that is followed in this dissertation.

An object is identified through its shape and/or texture. When an object is described by a shape only, it results in active contours (Kass et al., 1987; Blake and Isard, 1998). The use of only texture information leads to kernel-based (blob) tracker, for example (Comaniciu et al., 2003). The combination of shape and texture information results in active appearance models (Cootes et al., 1998; Edwards et al., 1998).

1.1 Overture

In order to track objects, we need two things: a way to describe an object, *i.e.*, an object representation, and a way to follow the object representation from one frame to the next.

The three ways to represent an object are points, kernels (blobs), and silhouettes (the shape and contours of a deformable object), each with its own advantages. We focus on a silhouette representation. In particular, we focus on active appearance models which are template based, built from a combination of contour (shape) and greyscale information, allowing accurate, but not necessarily robust, tracking of contours (Stegmann, 2001).

Having chosen an active appearance model representation of the object to be tracked, we need a way to follow this representation from one frame to the next. Here the approaches are either deterministic or stochastic. Deterministic approaches initialise a search for the object in the current frame with the previous frame's object parameters. No knowledge of the noise or the dynamics of the moving object is taken into account. However, stochastic approaches incorporate our knowledge of both the noise and the dynamics into the prediction of the parameters of the object representation.

With our choice of active appearance models as an object representation, a deterministic approach is shown to break under certain conditions. We therefore choose a stochastic approach—particle filtering.

Particle filters have been shown to provide robust stochastic tracking (Isard and Blake, 1998a; Pérez et al., 2002). A particle filter is a generalisation of the Kalman filter (Kalman, 1960)—a method of updating the state of a system given some noisy measurements, some knowledge of the dynamics, how the measurements are made, and the statistics of the noise. For example, the estimate of a boat's position at sea can be updated using a (noisy) measurement of the sun's position and knowledge of our current velocity. This updated measurement combines all information (measurement and dynamics) and is therefore more accurate than if the information were used separately.

We combine active appearance models and particle filters to obtain accuracy and robustness, bearing the speed constraints in mind. This yields *stochastic tracking with active appearance* *models*. In particular, we assume that the parameters describing an active appearance model are distributed according some probability density function (pdf). This pdf is approximated by a set of samples. Using the particle filter, the pdf is propagated over time. This enables us to track an object.

We have chosen implementation in an embedded environment and 3D-tracking as final test beds for the algorithms.

Implementing computer vision algorithms in an embedded environment, such as smart cameras, has recently become more prevalent. This enables distributed computing, relieving the bandwidth problem, and avoiding privacy issues (Fleck and Straßer, 2008). Since this environment has limited processing power per node, it is an excellent test for the delicate speed versus accuracy trade-off. Our algorithms run in near real-time on a standard smart camera.

The 3D-tracking algorithms are an extension of their 2D counterparts. Our 2D-tracker is built on an active appearance model allowing a straightforward extension to 3D. We also get an estimate of the accuracy of the 2D-trackers from the epipolar constraints in the 3D-tracker. Satisfactory results are obtained.

1.2 Dissertation objectives

This work's principal objective is to provide a detailed investigation into the viability (in terms of speed, robustness and accuracy) of stochastic tracking with active appearance models. The secondary objectives are to provide a fully fleshed derivation of all the algorithms in a tutorial style, and to comment on implementation issues.

1.3 Contributions of this work

This dissertation makes the following contributions:

- A combination of the particle filter and active appearance models has also been proposed by Hamlaoui and Davoine (2005); our approach is similar, but with some differences. For example, we use different models, a fixed number of particles and local optimisation to increase robustness of the tracker (Fleck et al., 2007). A detailed description of the differences is provided in Chapter 4.
- We compare the combined active appearance models and particle filter tracker with an active contours-active appearance models combination (Hoffmann et al., 2007b) and investigate

in particular how these different methods deal with occlusions (Hoffmann et al., 2007a). These results are presented in Chapter 4.

- ➤ Based on Saragih and Göcke (2006), we investigate the advantages of non-linear iterative active appearance model fitting methods in a particle filter and present an implementation of the tracker on a smart camera (Hoffmann et al., 2008b) and an extension to a 3D-face tracker in Chapter 5.
- ➤ A software library, written in C++, allowing tracking of objects with active appearance models and particle filters.

1.4 Outline of the dissertation

We review particle filters in Chapter 2, followed by a discussion of active appearance models in Chapter 3. Combinations of these two methods and other tracking methods are presented in Chapter 4, with experimental results where appropriate. Chapter 5 presents the implementation and further experimentation.

We have written each of Chapters 2 and 3 so that they can be read independently as tutorials from the rest of the dissertation. This also means that the reader can comfortably skip either chapter if he is familiar with their topics.

Note that we have not included a literature review as a separate chapter so that Chapters 2 and 3 remain independent. We provide references within each body of work.

CHAPTER

2

The particle filter

"Have you pen and ink, Master Doctor?"
"A scholar is never without them, your Majesty," answered Doctor Cornelius.

— C.S. LEWIS, THE CHRONICLES OF NARNIA

SUPPOSE one wants to model a dynamic process that is contaminated by noise, for example tracking an object through an image sequence. The process is usually described by a state vector at time t denoted by $\boldsymbol{x}_t \in \mathbb{R}^{n_x}$. Furthermore, suppose the state vector $\boldsymbol{x}_t$ is not observed directly, but is known through some noisy measurements $\boldsymbol{z}_t \in \mathbb{R}^{n_z}$ and knowledge of the dynamic evolution of the system. Using all the available information, *i.e.*, all measurements and knowledge of the dynamic process, the aim is to find the best possible estimate for the state $\boldsymbol{x}_t$.

In particular, we assume that the states evolve according to

$$\boldsymbol{x}_t = \boldsymbol{f}_{t-1}\left(\boldsymbol{x}_{t-1}, \boldsymbol{v}_{t-1}\right) \tag{2.1}$$

where $\boldsymbol{f}_{t-1}$ is a known, possibly non-linear function and $\boldsymbol{v}_{t-1}$ is the process noise. The state is related to the measurements via the measurement equation

$$\boldsymbol{z}_t = \boldsymbol{g}_t\left(\boldsymbol{x}_t, \boldsymbol{w}_t\right) \tag{2.2}$$

where $\boldsymbol{g}_t$ is again a known, possibly non-linear function and $\boldsymbol{w}_t$ is the measurement noise. The state equation (2.1) describes the transitional probability, $p\left(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}\right)$, whereas the likelihood $p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)$

is depicted by the measurement equation (2.2). If no assumptions are made about $\boldsymbol{f}_{t-1}$, $\boldsymbol{g}_t$, and the statistics of the noise, explicit formulas for $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ and $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ are not in general available.

A special case is the linear Gaussian dynamic system when equations (2.1) and (2.2) reduce to

$$\boldsymbol{x}_t = \mathrm{F}_{t-1}\boldsymbol{x}_{t-1} + \boldsymbol{v}_{t-1} \tag{2.3}$$

$$\boldsymbol{z}_t = \mathrm{G}_t\boldsymbol{x}_t + \boldsymbol{w}_t\ , \tag{2.4}$$

with $\boldsymbol{v}_{t-1}$ and $\boldsymbol{w}_t$ Gaussian distributed random variables. In this case, $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ and $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ are known explicitly. The exact solution to equations (2.3) and (2.4) is given by the Kalman filter (Kalman, 1960). Here F_{t-1} is called the state transition matrix and G_t the measurement matrix.

The stochastic filtering problem given by (2.1) and (2.2) can also be described as a Bayesian network, illustrated in Figure 2.1. A Bayesian network (Pearl, 1988; Lauritzen, 1996) is a directed graphical model which represents the relationships between a set of variables as a directed acyclic graph. Here we clearly see that the current state $\boldsymbol{x}_t$ depends on the previous state $\boldsymbol{x}_{t-1}$ and the measurement at time t, $\boldsymbol{z}_t$, depends on the state $\boldsymbol{x}_t$, but is conditionally independent of the measurements at other time steps. This is consistent with equations (2.1) and (2.2).

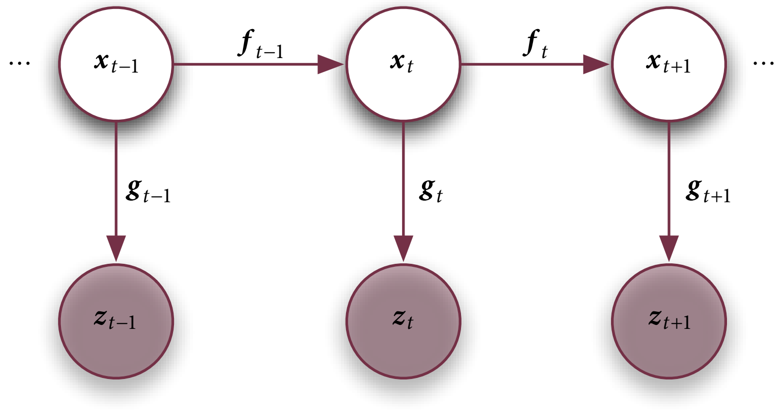


Figure 2.1: *Graphical model of stochastic filtering.*

The aim of stochastic filtering is thus to find the pdf $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t})$ that is a complete solution for the problem. However, this is often intractable since typically $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t})$ is a density function and may not be available in closed form. Nevertheless, it is instructive to understand the exact conceptual solution. Here we use the conceptual solution (discussed in Section 2.1.1) as a starting point in the development of the ideas underlying a Monte Carlo approximation to the problem:

the particle filter (presented in Section 2.1.3). En route we present the Kalman filter as the exact solution to the special case (2.3), (2.4) in Section 2.1.2 and review Monte Carlo methods in Section 2.1.3.1. Resulting algorithms are presented in Section 2.2. Finally we discuss a simple example in Section 2.3.

The concepts presented here have been extensively investigated in the literature. The goal of this chapter is to provide a concise summary of the theory of particle filters, together with a small example to aid in the understanding of the topic. We also provide references for further investigation.

2.1 Recursive Bayes filter

In this section we discuss the conceptual solution to the stochastic filtering problem described by Equations (2.1) and (2.2) (Ho and Lee, 1964; Maybeck, 1979), the Kalman filter as the exact solution of a special case and the particle filter as a numerical approximation.

2.1.1 The conceptual solution

We use the notation $\boldsymbol{x}_{0:t}$ to denote the set of states, up to and including the state at time t, *i.e.*, $\boldsymbol{x}_{0:t} \triangleq \{\boldsymbol{x}_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_t\}$.

Implicit in the model described above, are the assumptions that the states follow a first order Markov process, that is $p\left(\boldsymbol{x}_t | \boldsymbol{x}_{0:t-1}, \boldsymbol{z}_{0:t-1}\right) = p\left(\boldsymbol{x}_t | \boldsymbol{x}_{t-1}\right)$, and that the measurements are conditionally independent of each other given the state sequence.

The goal in solving the stochastic filtering problem (Bar-Shalom et al., 2001) in a Bayesian framework, is finding the posterior pdf of the states given the measurements, $p\left(\boldsymbol{x}_t | \boldsymbol{z}_{0:t}\right)$. This posterior pdf contains all the information about the latent variables and can thus be used to find estimates of the state. The recursive Bayesian filter provides a formal way to propagate the posterior pdfs over time if an initial condition is given.

In order to see how the recursive Bayesian filter operates, let us consider the posterior pdf

$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t})$ at time t. We have that

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}) &= \frac{p(\boldsymbol{z}_{0:t},\boldsymbol{x}_t)}{p(\boldsymbol{z}_{0:t})} \\
&= \frac{p(\boldsymbol{x}_t,\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1})\,p(\boldsymbol{z}_{0:t-1})}{p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1})\,p(\boldsymbol{z}_{0:t-1})} \\
&= \frac{p(\boldsymbol{z}_t|\boldsymbol{x}_t,\boldsymbol{z}_{0:t-1})\,p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})}{p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1})} \\
\text{(conditional independence)} &= \frac{p(\boldsymbol{z}_t|\boldsymbol{x}_t)\,p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})}{p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1})}\,. \qquad (2.5)
\end{aligned}$$

The recursive formula for the posterior pdf (2.5) consists of the prior $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})$, the likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ and the model evidence $p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1})$. Using the state transition pdf $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$, the posterior at time $t-1$, $p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})$, and marginalising[1] over $\boldsymbol{x}_{t-1}$ the prior is written as

$$\begin{aligned}
p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}) &= \int p(\boldsymbol{x}_t,\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_{t-1} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1},\boldsymbol{z}_{0:t-1})\,p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_{t-1} \\
&= \int p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\;p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_{t-1}\,. \qquad (2.6)
\end{aligned}$$

For the calculation of (2.6), the initial estimate and dynamics are needed.

The likelihood $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ is the probability of the measurement given the current state, *i.e.*, how likely the measurement $\boldsymbol{z}_t$ is. The evidence is given by

$$\begin{aligned}
p(\boldsymbol{z}_t|\boldsymbol{z}_{0:t-1}) &= \int p(\boldsymbol{z}_t,\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_t \\
&= \int p(\boldsymbol{z}_t|\boldsymbol{x}_t,\boldsymbol{z}_{0:t-1})\,p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_t \\
&= \int p(\boldsymbol{z}_t|\boldsymbol{x}_t)\;p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1})\,d\boldsymbol{x}_t. \qquad (2.7)
\end{aligned}$$

In (2.7), the measurement $\boldsymbol{z}_t$ and the dynamic update (2.6) are needed to successfully compute the evidence.

All the components of $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t})$ can thus be calculated since, crucially we assume $p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$ and $p(\boldsymbol{z}_t|\boldsymbol{x}_t)$ are known. The recursive Bayesian filter is seldom implemented because the marginalisations (2.6) and (2.7) are intractable. For the Kalman filter, these marginalisations can be done analytically.

Using the posterior pdf at time t, it is possible to calculate several estimates for the state. One

[1]Also known as the Chapman-Kolmogorov equation.

such estimate is the conditional mean

$$\overline{\boldsymbol{x}}_{j|k} \triangleq \mathbb{E}[\boldsymbol{x}_j|\boldsymbol{z}_{0:k}] = \int \boldsymbol{x}_j \cdot p\left(\boldsymbol{x}_j|\boldsymbol{z}_{0:k}\right)\, d\boldsymbol{x}_j\,, \tag{2.8}$$

with its conditional variance

$$\mathrm{P}_{j|k} \triangleq \mathbb{E}\left[\left(\boldsymbol{x}_j - \overline{\boldsymbol{x}}_{j|k}\right)\left(\boldsymbol{x}_j - \overline{\boldsymbol{x}}_{j|k}\right)^{\mathrm{T}} |\boldsymbol{z}_{0:k}\right]. \tag{2.9}$$

In all the cases we will consider, $j \leqslant k$.

2.1.2 The Kalman filter

Thus far we have presented the conceptual solution to the recursive Bayesian filter. In the special case of a linear Gaussian system, the recursive Bayesian filter reduces to the Kalman filter (Kalman, 1960; Zarchan and Musoff, 2005). Here we present the Kalman filter, as viewed from the recursive Bayes point (Ho and Lee, 1964; Maybeck, 1979).

For the linear Gaussian system, we assume that the process and measurement models are given by (2.3) and (2.4) respectively, listed again for convenience:

$$\begin{aligned}\boldsymbol{x}_t &= \mathrm{F}_{t-1}\boldsymbol{x}_{t-1} + \boldsymbol{v}_{t-1}\\ \boldsymbol{z}_t &= \mathrm{G}_t\boldsymbol{x}_t + \boldsymbol{w}_t\,.\end{aligned}$$

We denote a Gaussian distribution with mean $\boldsymbol{m}$ and covariance C as $\mathcal{N}\left(\boldsymbol{m}, \mathrm{C}\right)$. Using this notation, we assume that $\boldsymbol{v}_t \sim \mathcal{N}\left(\mathbf{0}, \mathrm{Q}_t\right)$ and $\boldsymbol{w}_t \sim \mathcal{N}\left(\mathbf{0}, \mathrm{R}_t\right)$ and that $\boldsymbol{v}_t$ and $\boldsymbol{v}_{t'}$ are independent for $t \neq t'$. Similarly, $\boldsymbol{w}_t$ and $\boldsymbol{w}_{t'}$ are assumed to be independent for $t \neq t'$. We also assume that the noise $\boldsymbol{v}_t$ and $\boldsymbol{w}_t$ are independent. Consequently, the dynamics and measurement equations can be written as

$$p\left(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}\right) = \mathcal{N}\left(\mathrm{F}_{t-1}\boldsymbol{x}_{t-1}, \mathrm{Q}_{t-1}\right) \tag{2.10}$$

and

$$p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right) = \mathcal{N}\left(\mathrm{G}_t\boldsymbol{x}_t, \mathrm{R}_t\right) \tag{2.11}$$

respectively.

It is well-known that the product of two Gaussian pdfs is again a Gaussian pdf. Likewise, the marginal and conditional of a Gaussian distribution, is Gaussian. Thus to derive the Kalman filter, we use the conceptual solution (2.5) and simply read off the values from the formulas given in Appendix A.

It will be useful to write

$$p\left(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1}\right) = \mathcal{N}\left(\widehat{\boldsymbol{x}}_{t-1}, \mathrm{P}_{t-1}\right). \tag{2.12}$$

A solution for (2.5) requires the specification of (2.6) and (2.7). Using our assumptions of the Kalman filter, (2.6) is given by

$$\begin{aligned} p\left(\boldsymbol{x}_{t}|\boldsymbol{z}_{0:t-1}\right) &= \mathcal{N}\left(\mathrm{F}_{t-1}\widehat{\boldsymbol{x}}_{t-1}, \mathrm{Q}_{t-1} + \mathrm{F}_{t-1}\mathrm{P}_{t-1}\mathrm{F}_{t-1}^{\mathrm{T}}\right) && (2.13)\\ &\triangleq \mathcal{N}\left(\boldsymbol{x}_t^-, \mathrm{P}_t^-\right) && (2.14)\end{aligned}$$

where we use the marginalisation property of a Gaussian. Here we have defined $\boldsymbol{x}_t^-$ as the predicted estimate that is obtained without seeing any measurement; similarly P_t^- is the predicted covariance. For the evidence (2.7), we have (again using the marginalisation property of a Gaussian)

$$\begin{aligned} p\left(\boldsymbol{z}_{t}|\boldsymbol{z}_{0:t-1}\right) &= \mathcal{N}\left(\mathrm{G}_t\boldsymbol{x}_t^-, \mathrm{R}_t + \mathrm{G}_t\mathrm{P}_t^-\mathrm{G}_t^{\mathrm{T}}\right) && (2.15)\\ &\triangleq \mathcal{N}\left(\boldsymbol{z}_t^-, \mathrm{S}_t\right). && (2.16)\end{aligned}$$

Finally

$$p\left(\boldsymbol{x}_{t}|\boldsymbol{z}_{0:t}\right) = \mathcal{N}\left(\widehat{\boldsymbol{x}}_{t}, \mathrm{P}_{t}\right) \tag{2.17}$$

where

$$\widehat{\boldsymbol{x}}_t = \mathrm{P}_t\left(\mathrm{P}_t^-\right)^{-1}\boldsymbol{x}_t^- + \mathrm{P}_t\mathrm{G}_t^{\mathrm{T}}\mathrm{R}_t^{-1}\boldsymbol{z}_t \tag{2.18}$$

and

$$\mathrm{P}_t = \left[\left(\mathrm{P}_t^-\right)^{-1} + \mathrm{G}_t^{\mathrm{T}}\mathrm{R}_t\mathrm{G}_t\right]^{-1}. \tag{2.19}$$

At this point, we have all the information for the solution of (2.5). However, if we define the Kalman gain

$$\mathrm{K}_t \triangleq \mathrm{P}_t^-\mathrm{G}_t^{\mathrm{T}}\mathrm{S}_t^{-1}, \tag{2.20}$$

then (2.18) and (2.19) simplify (after considerable algebraic manipulation) to

$$\widehat{\boldsymbol{x}}_t = \boldsymbol{x}_t^- + \mathrm{K}_t \left(\boldsymbol{z}_t - \boldsymbol{z}^- \right) \tag{2.21}$$

and

$$\mathrm{P}_t = \mathrm{P}_t^- - \mathrm{K}_t \mathrm{S}_t \mathrm{K}_t^\mathrm{T} \tag{2.22}$$

respectively.

The Kalman filter operates in two steps (this can also be said for the recursive Bayes filter). During the first step we propagate the pdf $p\left(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1}\right)$ to $p\left(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}\right)$ using (2.14) and (2.16). Then a new measurement becomes available. By using (2.20), (2.21), and (2.22) we propagate the pdf $p\left(\boldsymbol{x}_t|\boldsymbol{z}_{0:t-1}\right)$ to $p\left(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}\right)$.

2.1.3 The particle filter

As discussed in Section 2.1.1, exact inference in the Bayesian filter is not in general possible due to intractable integrals. In general, $p\left(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}\right)$ could be multivariate, multi-modal or unavailable in closed form. In these cases one has to resort to Monte Carlo techniques to approximate the integrals. Hence we proceed to provide an overview of Monte Carlo (MC) methods. They form the cornerstone of the numerical approximations of the recursive Bayesian filter. Thereafter we apply the MC techniques to the recursive Bayesian filter resulting in sequential importance sampling (SIS) (Doucet and de Freitas, 2001; Isard and Blake, 1998a; Ristic et al., 2004), also known as the particle filter.

2.1.3.1 Monte Carlo methods

Loosely following the notation of Bishop (2006), we provide an overview of Monte Carlo (MC) methods.

In the MC framework, we wish to estimate the expected value of a function $f\left(\boldsymbol{x}\right)$ with respect to the distribution function $p\left(\boldsymbol{x}\right)$,

$$\mathbb{E}\left[f\right] = \int f\left(\boldsymbol{x}\right) p\left(\boldsymbol{x}\right) d\boldsymbol{x}\,. \tag{2.23}$$

We assume that N independent samples $\boldsymbol{x}^{(i)}$, with $i = 1, \ldots \mathrm{N}$, drawn from $p\left(\boldsymbol{x}\right)$ are available.

Then the expectation in (2.23) is approximated by

$$\hat{f} = \frac{1}{N} \sum_{i=1}^{N} f\left(\boldsymbol{x}^{(i)}\right), \tag{2.24}$$

that is by the sample mean of the function f.

Two important measures of the MC estimate are the expected value and variance of the estimator. The expected value is given by

$$\mathbb{E}\left[\hat{f}\right] = \mathbb{E}\left[f\right]. \tag{2.25}$$

This tells us that the MC estimator is an unbiased estimate for our initial problem. For the derivation of the variance, we use a well-known identity that $\text{var}\left[f\right] = \mathbb{E}\left[f^2\right] - \left(\mathbb{E}\left[f\right]\right)^2$ and (2.25):

$$\begin{aligned} \text{var}\left[\hat{f}\right] &= \mathbb{E}\left[\hat{f}^2\right] - \left(\mathbb{E}\left[\hat{f}\right]\right)^2 \\ &= \mathbb{E}\left[\hat{f}^2\right] - \left(\mathbb{E}\left[f\right]\right)^2. \end{aligned} \tag{2.26}$$

Now note that

$$\begin{aligned} \mathbb{E}\left[f\left(\boldsymbol{x}^{(k)}\right) f\left(\boldsymbol{x}^{(m)}\right)\right] &= \begin{cases} \text{var}\left[f\right] + \left(\mathbb{E}\left[f\right]\right)^2 & \text{if } k = m \\ \left(\mathbb{E}\left[f\right]\right)^2 & \text{otherwise} \end{cases} \\ &= \left(\mathbb{E}\left[f\right]\right)^2 + \delta_{mk}\text{var}\left[f\right]. \end{aligned} \tag{2.27}$$

Substituting (2.24) into (2.26) and then using (2.27), we obtain

$$\begin{aligned} \text{var}\left[\hat{f}\right] &= \mathbb{E}\left[\frac{1}{N}\sum_{k=1}^{L} f\left(\boldsymbol{x}^{(k)}\right) \frac{1}{N}\sum_{m=1}^{N} f\left(\boldsymbol{x}^{(m)}\right)\right] - \left(\mathbb{E}\left[f\right]\right)^2 \\ &= \frac{1}{N^2}\sum_{k=1}^{N}\sum_{m=1}^{N}\left[\left(\mathbb{E}\left[f\right]\right)^2 + \delta_{mk}\text{var}\left[f\right]\right] - \left(\mathbb{E}\left[f\right]\right)^2 \\ &= \frac{1}{N}\text{var}\left[f\right]. \end{aligned} \tag{2.28}$$

From (2.28), we see that the variance of the estimator gets smaller as we increase the number of samples.

The MC techniques suffer from several problems. Amongst others, it may be difficult or impossible to sample from $p\left(\boldsymbol{x}\right)$; in this case one can use importance sampling. The idea behind importance sampling is to use a proposal density function $q(\boldsymbol{x})$ that is easy to sample from, instead of $p(\boldsymbol{x})$. The support of the proposal pdf should include $p(\boldsymbol{x})$, *i.e.*,

$$p(\boldsymbol{x}) > 0 \implies q(\boldsymbol{x}) > 0\,. \tag{2.29}$$

Now we sample N independent samples from $q(\boldsymbol{x})$. We can write the expectation in (2.23) as

$$\begin{aligned}\mathbb{E}[f] &= \int f(\boldsymbol{x})\,p(\boldsymbol{x})\,d\boldsymbol{x}\\ &= \int f(\boldsymbol{x})\,\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})\,d\boldsymbol{x}\\ &\approx \frac{1}{\mathrm{N}}\sum_{l=1}^{\mathrm{N}}\frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}f\left(\boldsymbol{x}^{(i)}\right).\end{aligned} \tag{2.30}$$

The importance sampling estimate in (2.30) is similar to the MC estimate (2.24). The only difference is the additional factor, $\frac{p(x^{(i)})}{q(x^{(i)})}$ that corrects the bias since we are not sampling from $p(\boldsymbol{x})$; we define this as the importance weights

$$w^{(i)} \triangleq \frac{p(\boldsymbol{x}^{(i)})}{q(\boldsymbol{x}^{(i)})}. \tag{2.31}$$

Suppose further that $p(\boldsymbol{x})$ can only be evaluated up to a normalising constant, such that

$$p(\boldsymbol{x}) = \frac{\widetilde{p}(\boldsymbol{x})}{\mathrm{Z}_p}\,,$$

where $\widetilde{p}(\boldsymbol{x})$ can be easily evaluated and Z_p is the normalising constant. Similarly, we assume that $q(\boldsymbol{x})$ can be evaluated up to a normalising constant Z_q where $\widetilde{q}(\boldsymbol{x})$ can be easily evaluated,

$$q(\boldsymbol{x}) = \frac{\widetilde{q}(\boldsymbol{x})}{\mathrm{Z}_q}\,.$$

Then we calculate the MC estimate as

$$\begin{aligned}\mathbb{E}[f] &= \int f(\boldsymbol{x})\,p(\boldsymbol{x})\,d\boldsymbol{x}\\ &= \frac{\mathrm{Z}_q}{\mathrm{Z}_p}\int f(\boldsymbol{x})\,\frac{\widetilde{p}(\boldsymbol{x})}{\widetilde{q}(\boldsymbol{x})}q(\boldsymbol{x})\,d\boldsymbol{x}\\ &\approx \frac{\mathrm{Z}_q}{\mathrm{Z}_p}\frac{1}{\mathrm{N}}\sum_{i=1}^{\mathrm{N}}\widetilde{w}^{(i)}f\left(\boldsymbol{x}^{(i)}\right)\end{aligned} \tag{2.32}$$

where $\widetilde{w}^{(i)} = \frac{\widetilde{p}(x^{(i)})}{\widetilde{q}(x^{(i)})}$.

We proceed by calculating the MC estimate for the normalising factor as

$$\begin{aligned}\frac{Z_p}{Z_q} &= \frac{1}{Z_q}\int \widetilde{p}(\boldsymbol{x})\,d\boldsymbol{x} \\ &= \int \frac{\widetilde{p}(\boldsymbol{x})}{\widetilde{q}(\boldsymbol{x})} q(\boldsymbol{x})\,d\boldsymbol{x} \\ &\approx \frac{1}{N}\sum_{i=1}^{N} \widetilde{w}^{(i)}. \end{aligned} \tag{2.33}$$

Now we define the weights as

$$w^{(i)} = \frac{\widetilde{w}^{(i)}}{\sum_{m=1}^{N} \widetilde{w}^{(m)}}, \tag{2.34}$$

and therefore

$$\mathbb{E}[f] = \sum_{i=1}^{N} w^{(i)} f\left(\boldsymbol{x}^{(i)}\right).$$

This result should be emphasised. Equation (2.34) tells us that if $p(\boldsymbol{x})$ and $q(\boldsymbol{x})$ can only be evaluated up to a normalising constant, we can find an approximation for this constant by normalising the importance weights. We will use this fact to simplify the equations when we derive the particle filter.

2.1.3.2 Sequential importance sampling (SIS)

At this point we have introduced all the numerical techniques that are used to approximate the recursive Bayesian filter. The fundamental idea of particle filtering is to approximate the pdf $p(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t})$ and eventually the marginal of it by a weighted sample set S_t. Thus, suppose N samples or particles $\boldsymbol{x}_{0:t}^{(i)}$ from the pdf $p(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t})$ are available, with a weight $w_t^{(i)}$ associated with each sample $\boldsymbol{x}_t^{(i)}$ normalised such that $\sum_{i=1}^{N} w_t^{(i)} = 1$. Using (2.23) and (2.30) we have the following discrete representation

$$p(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}) \approx \sum_{i=1}^{N} w_t^{(i)} \delta\left(\boldsymbol{x}_{0:t} - \boldsymbol{x}_{0:t}^{(i)}\right). \tag{2.35}$$

To derive the recursive formulation, using a notation similar to (Isard and Blake, 1998a; Doucet and de Freitas, 2001; Ristic et al., 2004), we begin by finding a recursive approximation for pdf $p\left(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}\right)$. We assume that N samples $\boldsymbol{x}_{0:t-1}^{(i)}$ with associated weights $w_{t-1}^{(i)}$ are available approximating the posterior $p\left(\boldsymbol{x}_{0:t-1}|\boldsymbol{z}_{0:t-1}\right)$. We have that

$$\begin{aligned}
p\left(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}\right) &= \frac{p\left(\boldsymbol{x}_{0:t},\boldsymbol{z}_{0:t}\right)}{p\left(\boldsymbol{z}_{0:t}\right)} \\
&= \frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{0:t},\boldsymbol{z}_{0:t-1}\right)p\left(\boldsymbol{z}_{0:t-1},\boldsymbol{x}_{0:t}\right)}{p\left(\boldsymbol{z}_{0:t}\right)} \\
&= \frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}\right)p\left(\boldsymbol{x}_{t}|\boldsymbol{x}_{0:t-1},\boldsymbol{z}_{0:t-1}\right)p\left(\boldsymbol{x}_{0:t-1},\boldsymbol{z}_{0:t-1}\right)}{p\left(\boldsymbol{z}_{0:t}\right)} \\
&= \frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}\right)p\left(\boldsymbol{x}_{t}|\boldsymbol{x}_{t-1}\right)p\left(\boldsymbol{x}_{0:t-1}|\boldsymbol{z}_{0:t-1}\right)p\left(\boldsymbol{z}_{0:t-1}\right)}{p\left(\boldsymbol{z}_{t}|\boldsymbol{z}_{0:t-1}\right)p\left(\boldsymbol{z}_{0:t-1}\right)} \\
&= \frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}\right)p\left(\boldsymbol{x}_{t}|\boldsymbol{x}_{t-1}\right)}{p\left(\boldsymbol{z}_{t}|\boldsymbol{z}_{0:t-1}\right)}p\left(\boldsymbol{x}_{0:t-1}|\boldsymbol{z}_{0:t-1}\right) \\
&\propto p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}\right)p\left(\boldsymbol{x}_{t}|\boldsymbol{x}_{t-1}\right)p\left(\boldsymbol{x}_{0:t-1}|\boldsymbol{z}_{0:t-1}\right). \qquad (2.36)
\end{aligned}$$

Since we use importance sampling, the importance weights become (see (2.31))

$$w_t^{(i)} \propto \frac{p\left(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{z}_{0:t}\right)}{q\left(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{z}_{0:t}\right)}. \qquad (2.37)$$

Note that they are unnormalised. By normalising them, we get an MC approximation for the normalising factor.

We choose the proposal density to factorise as

$$q\left(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}\right) = q\left(\boldsymbol{x}_{t}|\boldsymbol{x}_{0:t-1},\boldsymbol{z}_{0:t}\right)q\left(\boldsymbol{x}_{0:t-1}|\boldsymbol{z}_{0:t-1}\right). \qquad (2.38)$$

Upon substituting (2.36) and (2.38) in (2.37), we obtain

$$\begin{aligned}
w_t^{(i)} &\propto \frac{p\left(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{z}_{0:t}\right)}{q\left(\boldsymbol{x}_{0:t}^{(i)}|\boldsymbol{z}_{0:t}\right)} \\
&\propto \frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}^{(i)}\right)p\left(\boldsymbol{x}_{t}^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)p\left(\boldsymbol{x}_{0:t-1}^{(i)}|\boldsymbol{z}_{0:t-1}\right)}{q\left(\boldsymbol{x}_{t}^{(i)}|\boldsymbol{x}_{0:t-1}^{(i)},\boldsymbol{z}_{0:t}\right)q\left(\boldsymbol{x}_{0:t-1}^{(i)}|\boldsymbol{z}_{0:t-1}\right)} \\
&= w_{t-1}^{(i)}\frac{p\left(\boldsymbol{z}_{t}|\boldsymbol{x}_{t}^{(i)}\right)p\left(\boldsymbol{x}_{t}^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)}{q\left(\boldsymbol{x}_{t}^{(i)}|\boldsymbol{x}_{0:t-1}^{(i)},\boldsymbol{z}_{0:t}\right)}. \qquad (2.39)
\end{aligned}$$

We make one further assumption (conditional independence assumption) that

$$q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{0:t-1}^{(i)}, \boldsymbol{z}_{0:t}\right) = q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{z}_t\right),$$

and calculate the weights as

$$w_t^{(i)} \propto w_{t-1}^{(i)} \frac{p\left(\boldsymbol{z}_t|\boldsymbol{x}_t^{(i)}\right) p\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)}{q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{z}_t\right)} . \tag{2.40}$$

This allows us to write down the discrete representation of the marginal of $p\left(\boldsymbol{x}_{0:t}|\boldsymbol{z}_{0:t}\right)$ as

$$p\left(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}\right) \approx \sum_{i=1}^{\mathrm{N}} w_t^{(i)} \delta\left(\boldsymbol{x}_t - \boldsymbol{x}_t^{(i)}\right). \tag{2.41}$$

The resulting algorithm is summarised in Algorithm 1.

Algorithm 1: *Sequential importance sampling (SIS)*

```
  Input: Samples $\boldsymbol{x}_{t-1}^{(i)}$ with weights $w_{t-1}^{(i)}$, $i = 1, \ldots, \mathrm{N}$. Measurement $\boldsymbol{z}_t$.
1 for $i = 1 : \mathrm{N}$ do
2     Sample $\boldsymbol{x}_t^{(i)}$ from $q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{z}_t\right)$ ;
3     Evaluate the importance weights using (2.40) ;
4 end
5 Normalise weights using (2.34) ;
```

A common simplification is to choose the proposal density q as the transitional density, *i.e.*, $q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{z}_t\right) = p\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)$. Then the importance weights simplify to

$$w_t^{(i)} \propto w_{t-1}^{(i)} p\left(\boldsymbol{z}_t|\boldsymbol{x}_t^{(i)}\right) . \tag{2.42}$$

This is known as the basic PF. It has the advantage that the particles are easily obtained. However, no knowledge of the observations is incorporated in obtaining the samples and therefore this PF is not particularly efficient.

2.2 Algorithmic issues

When the sequential importance sampling (SIS) is implemented, often all but one of the weights become zero. The result is that the algorithm performs badly in practice.

Doucet et al. (2000) proved that when the proposal density $q(\boldsymbol{x})$ is written as in (2.38), the variance of the weights increases over time, resulting in unavoidable degeneracy. A measure of the degeneracy phenomenon is the effective sampling size

$$\widehat{\mathrm{N}}_{\mathrm{eff}} = \frac{1}{\sum_{i=1}^{\mathrm{N}} \left(w_t^{(i)}\right)^2}. \tag{2.43}$$

Note that $\widehat{\mathrm{N}}_{\mathrm{eff}} = 1$ when all but one weight is zero; $\widehat{\mathrm{N}}_{\mathrm{eff}} = \mathrm{N}$ if the weights are uniform.

A remedy to the problem is resampling: Whenever the effective sampling size falls below a certain threshold $\mathrm{N}_{\mathrm{thr}}$, a new set of particles is sampled from the current set, each sample proportional to its weight, *i.e.*, a new sample $\boldsymbol{x}_t^{(i)*}$ is chosen such that

$$\mathrm{P}\left\{\boldsymbol{x}_t^{(i)*} = \boldsymbol{x}_t^{(j)}\right\} = w_t^{(j)}. \tag{2.44}$$

Resampling is illustrated in Figure 2.2. A cumulative sum of the weights $w_t^{(i)}$ is calculated. Then a variable $u^{(i)}$ is drawn uniformly from $\mathrm{Unif}\,[0, 1]$. A particle is selected by mapping $u^{(i)}$ to the corresponding index j. Consequently, the weights after resampling are $\frac{1}{\mathrm{N}}$.

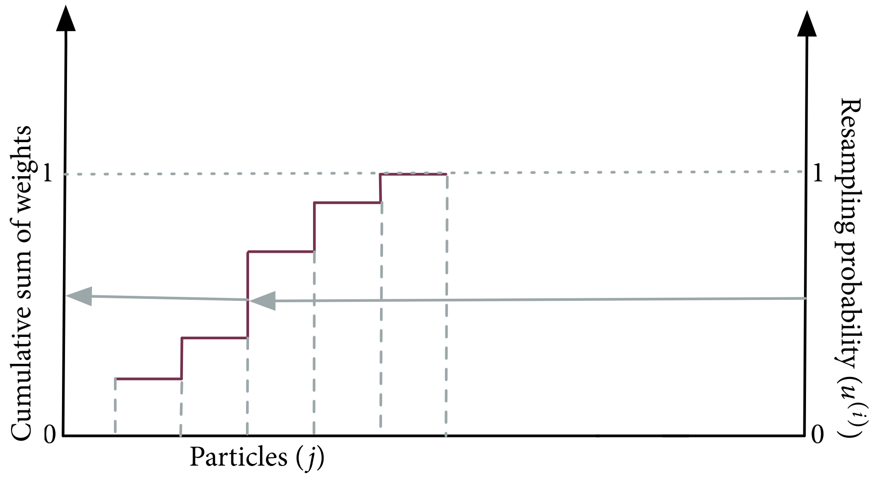


Figure 2.2: *Resampling of particles.*

Several implementations of resampling exist. One can implement resampling directly obeying (2.44); other alternatives include systematic resampling (Kitagawa, 1996) and residue sampling (Liu and Chen, 1998). Embedding resampling in SIS yields the generic particle filter and this algorithm is listed in Algorithm 2.

Algorithm 2: *Generic particle filter*

```
   Input: Samples $\boldsymbol{x}_t^{(i)}$ with weights $w_t^{(i)}$ obtained from SIS($\boldsymbol{x}_{t-1}^{(i)}$, $w_{t-1}^{(i)}$, $\boldsymbol{z}_t$).
1  Calculate $\widehat{\mathrm{N}}_{\mathrm{eff}}$ using (2.43);
2  if $\widehat{\mathrm{N}}_{\mathrm{eff}} < \mathrm{N}_{\mathrm{thr}}$ then
3      Resample such that (2.44) holds. Any technique can be used.;
4  end
```

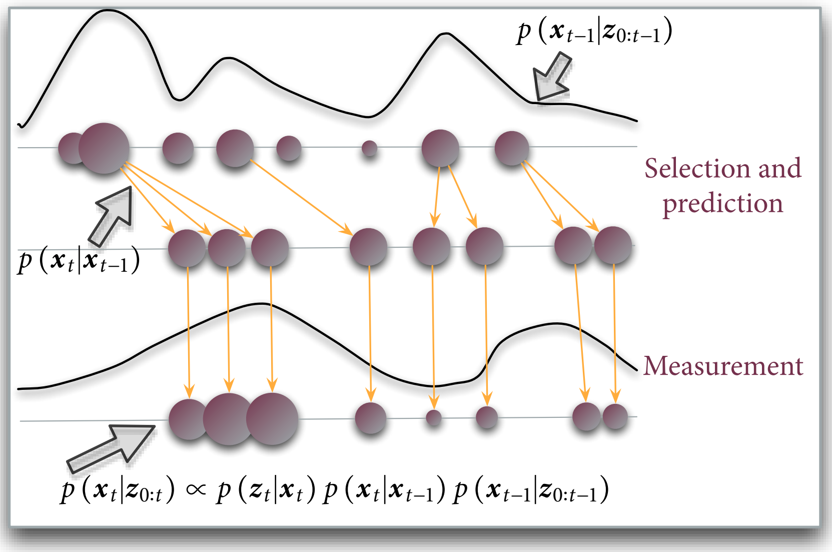
As an example of resampling as part of the PF, consider the pdf $p\left(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1}\right)$ represented by 8 samples as illustrated in Figure 2.3. Samples from this pdf are circular points in this graphical illustration. The particle filter algorithm propagates this posterior pdf at time $t-1$ to the posterior at time t using the sample-set representation. It is assumed that the proposal density is given by $p\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)$. The steps are outlined below.

- **Selection and prediction**: First, a cumulative histogram of all the samples' weights is computed. Then, according to each sample's weight $w_{t-1}^{(i)}$, its number of successors is determined according to its relative probability in this cumulative histogram. In the figure, the weight of a particle is indicated by the size of a circular point. Thus samples with larger weights (bigger circular points), will be chosen several times. The successors are then fed to the process model (2.1), yielding the new samples $\boldsymbol{x}_t^{(i)}$. This lead to particles with uniform weights (the circular points have the same size). Thus the particles are resampled at each iteration—this variation of the PF is known as sampling importance resampling (SIR). Since the particles are resampled at each step, they may suffer from loss of diversity.

- **Measurement update**: Here, the new sample $\boldsymbol{x}_t^{(i)}$ is weighted with the likelihood of the new measurement $\boldsymbol{z}_t$, *i.e.*, $w_t^{(i)} = p\left(\boldsymbol{z}_t|\boldsymbol{x}_t^{(i)}\right)$.

2.3 An example

Consider a boat travelling in a one-dimensional ocean as illustrated in Figure 2.4 (Sundvall, 2009). Given some noisy depth measurements and knowledge about the sea floor, the position must be



$p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})$
Selection and prediction
$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$
Measurement
$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}) \propto p(\boldsymbol{z}_t|\boldsymbol{x}_t)\, p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\, p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})$

Figure 2.3: *Particle propagation from time step $t-1$ to t. The horizontal axis denotes for example the position of an object being tracked.*

estimated. The dynamics of the boat are given by

$$x_t = \mathrm{A}x_{t-1} + v_{t-1}$$

where $v_{t-1} \sim \mathrm{Unif}(-\mathrm{F}_0, \mathrm{F}_0)$, $\mathrm{F}_0 \in \mathbb{R}$. The noisy measurement is described by

$$z_t = d(x_t) + w_t$$

where $w_t \sim \mathcal{N}(0, \sigma)$ and $d(x) = \sin(x) + ax + bx^2$. The proposal density is chosen as $p(x_t|x_{t-1})$. Consequently, we should be able to sample from $p(x_t|x_{t-1})$, but a closed form formula is not needed. The measurement noise w_t is Gaussian, and therefore the pdf $p(z_t|x_t)$ is also Gaussian. Thus the likelihood is given by

$$p(z_t|x_t) = \mathcal{N}(d(x_t), \sigma).$$

Note that a Kalman filter cannot find a solution for this problem, because the process noise is not Gaussian and the measurement function is non-linear.

In Figure 2.4, an estimate of the posterior pdf at some time t is shown. Clearly the pdf is multi-modal.

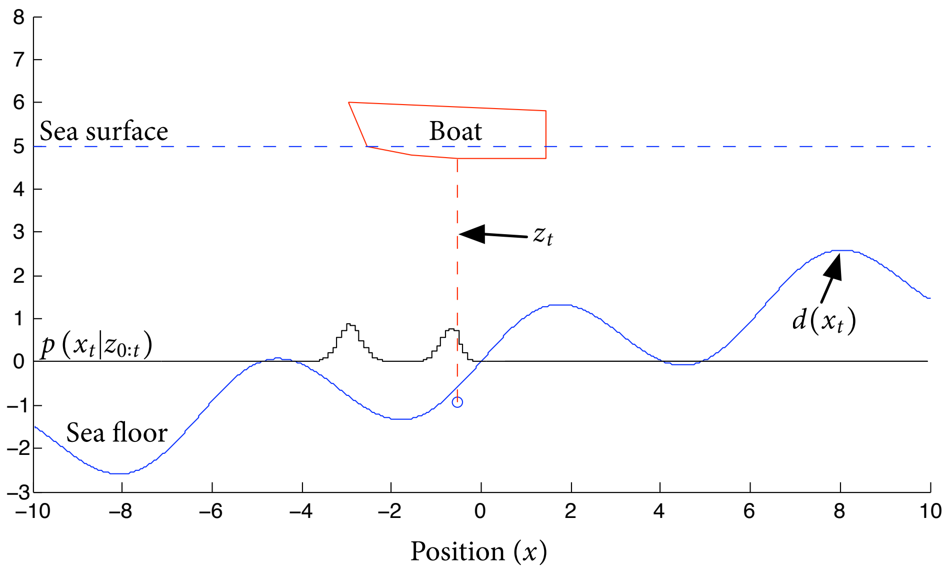


Figure 2.4: *A boat travelling in a one-dimensional ocean. A particle estimates the position of the boat. Image courtesy of Paul Sundvall.*

2.4 Summary

We discussed the estimation of a process state if only noisy measurements are observed. The recursive Bayesian filter provides a solution to this problem. However, this is seldom implemented due to intractable integrals. In the special case of a linear Gaussian system, the Kalman filter is the exact solution to the problem, *i.e.*, the recursive Bayesian filter reduces to the Kalman filter. Otherwise one has to use numerical approximations; when Monte Carlo methods are used the result is the particle filter.

Chapter

3

Active appearance models

Selfs ywer is sonder kennis nie goed nie; en hy wat haastig is met die voete, trap mis.

— Spreuke 19:2

Active appearance models (AAMs) (Cootes et al., 1998; Edwards et al., 1998) are deformable template models using shape and texture (greyscale or colour information) to segment objects of interest from an image. They are a generalisation of active shape models (ASMs) (Cootes et al., 1995) that only use shape information.

AAMs are template based because they use samples from the object in question during the training process. Furthermore, they are deformable since the objects can undergo shape and texture deformations, learnt from the training examples. For example, if the object is a person's face, the shape and texture deform as the person smiles.

AAMs were introduced in the context of modelling facial features (Edwards et al., 1998). Later, they were applied to numerous other problems, including segmentation of medical images (Stegmann et al., 2001), lip-reading (Matthews et al., 2002), facial expression recognition and synthesis (Abboud et al., 2004), and object tracking (Stegmann, 2001; Hansen et al., 2002b).

This chapter describes AAMs. We start by discussing the modelling of shape in Section 3.1. Here we also show the similarities to active contours (Blake and Isard, 1998). In Section 3.2 we discuss the modelling of texture. Having modelled shape and texture, one is able to define an independent AAM—the shape and texture models are used separately. However, the shape and texture information can be combined yielding a more compact model presentation, known as a dependent AAM and this is discussed in Section 3.3. Section 3.4 discusses model fitting when only a shape model is assumed. Thereafter, in Section 3.5 we describe finding the parameters of AAMs.

3.1 Modelling of shape

In an image, a shape describes the geometrical information of an object after normalisation with respect to location, scale, and rotational effects. The geometry of an object is described by landmark points. Thus, a shape can be viewed as a set of landmark points after normalisation. Landmark points are corresponding points that lie on the edge, boundary, corners, or distinct features inside an object. It is important to emphasise that landmark points are corresponding points that match between images of the same object. Figure 3.1 shows a face annotated with landmark points.

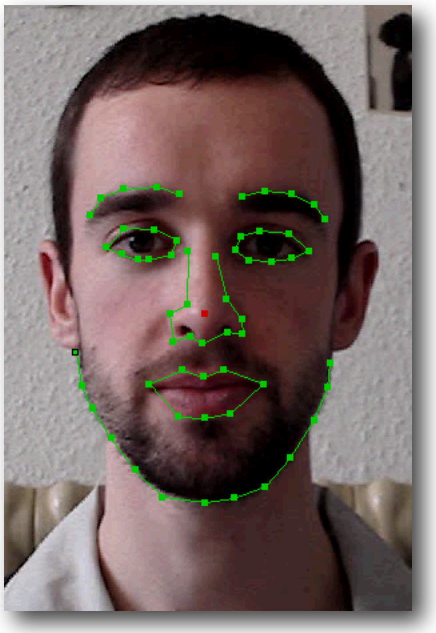


Figure 3.1: *An example of an image annotated with some landmark points. The landmark points are connected with straight lines.*

The goal of shape modelling is a compact representation of the shape given some landmark points in a set of training images. Collecting the landmark points is done manually or by an automatic annotation process. For automated collection procedures, see for example (Walker et al., 2000; Saragih and Göcke, 2006).

3.1.1 From landmark points to shape points

The spatial coordinates of the landmark points in the $\mathrm{N_T}$ training images are stacked into vectors $\boldsymbol{x}_i = [x_{i1}, \cdots, x_{in}, y_{i1}, \cdots, y_{in}]^{\mathrm{T}}$, $i = 1, \ldots, \mathrm{N_T}$ where n is the number of landmark points. To conform to our definition of a shape, these recorded features must be normalised with respect to scale, rotation and location. This normalisation is done by setting up a common coordinate frame. The features are then all aligned in this common frame by a procedure known as general Procrustes analysis (Goodall, 1991). This procedure yields parameters s_c and θ that describe the scale and rotational effects respectively, and t_x and t_y that are the translation parameters. We define the pose therefore as

$$\boldsymbol{p}_p \triangleq [s_c, \theta, t_x, t_y]^{\mathrm{T}}. \tag{3.1}$$

Using the pose $\boldsymbol{p}_p$ and the shapes, landmark points can be obtained by translation, scale and rotation of a shape. Thus, let $[x, y]^{\mathrm{T}}$ be a single shape point. Then the coordinates of the corresponding landmark point $\boldsymbol{x}$ is

$$\boldsymbol{x} = \begin{bmatrix} s\cos(\theta) & -s\sin(\theta) \\ s\sin(\theta) & s\cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}. \tag{3.2}$$

The normalised spatial coordinates are called the shape vectors and we denote them by $\boldsymbol{s}_i$.

3.1.2 From shapes to a model

A mean shape $\bar{\boldsymbol{s}}$ is calculated by

$$\bar{\boldsymbol{s}} = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} \boldsymbol{s}_i, \tag{3.3}$$

where $\boldsymbol{s}_i$ is a shape vector. The covariance is given by

$$\Sigma_s = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} (\boldsymbol{s}_i - \bar{\boldsymbol{s}})(\boldsymbol{s}_i - \bar{\boldsymbol{s}})^{\mathrm{T}}. \tag{3.4}$$

Principal component analysis (PCA) is performed on the shape covariance matrix Σ_s and the eigenvectors corresponding to the largest eigenvalues of Σ_s are recorded in Φ_s. Thus m modes are chosen such that

$$\sum_{i=1}^{m} \lambda_i \geqslant \frac{c}{100} \sum_{i=1}^{2n} \lambda_i \tag{3.5}$$

where λ_i is the ith eigenvalue of Σ_s and c is the percentage of variation we want to keep. This yields a generative model $\mathcal{S}(\boldsymbol{p}_s)$ where linear combinations of the columns of Φ_s, with coefficients $\boldsymbol{p}_s$, are taken according to

$$\mathcal{S}(\boldsymbol{p}_s) = \bar{\boldsymbol{s}} + \Phi_s \boldsymbol{p}_s. \tag{3.6}$$

Note that the mean shape $\bar{\boldsymbol{s}}$ is distorted by deformations spanned by the column space of Φ_s. Figure 3.2 illustrates the mean shape deformations obtained by varying the coefficients of the first and second principal components while the other coefficients are left unchanged.

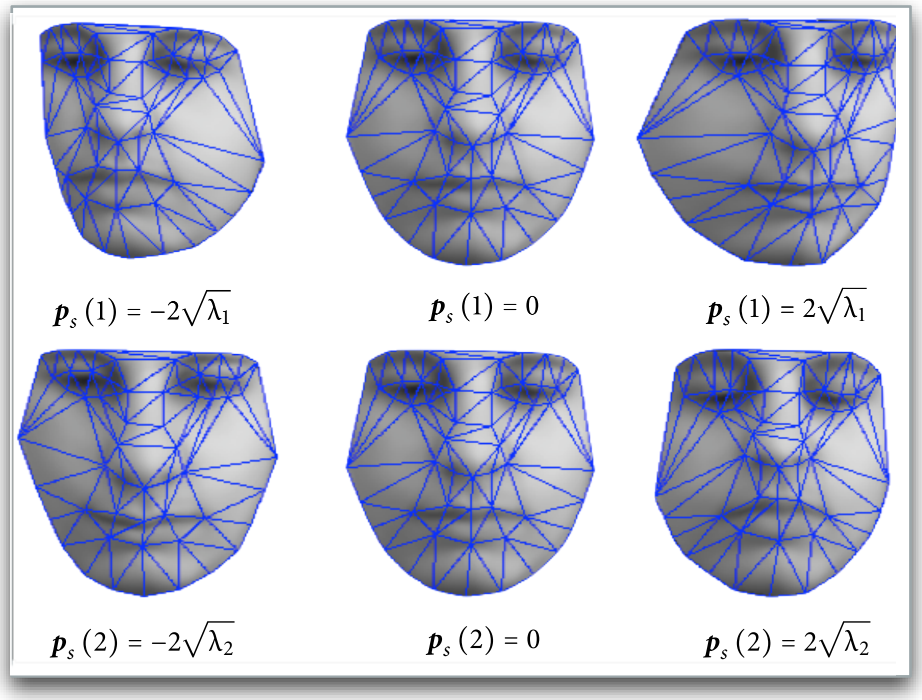

$\boldsymbol{p}_s(1) = -2\sqrt{\lambda_1}$
$\boldsymbol{p}_s(1) = 0$
$\boldsymbol{p}_s(1) = 2\sqrt{\lambda_1}$
$\boldsymbol{p}_s(2) = -2\sqrt{\lambda_2}$
$\boldsymbol{p}_s(2) = 0$
$\boldsymbol{p}_s(2) = 2\sqrt{\lambda_2}$

Figure 3.2: *Shape deformations obtained by varying the coefficients of the first and second principal components while the other coefficients are left unchanged. The shapes are overlaid with the mean greyscale information.*

3.1.3 Spline curves and shape space

The modelling of shape described above, is similar to the shape space of spline curves as discussed by Blake and Isard (1998).

Given a set of coordinates of control points $(x_1, y_1), \ldots, (x_n, y_n)$, a B-spline (see for example (de Boor, 1978)), is the curve $\boldsymbol{r}(t) = (x(t), y(t))$ formed by a parametrisation with parameter t on the real line,

$$\boldsymbol{r}(t) = \begin{bmatrix} \mathbf{B}(t)^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(t)^{\mathrm{T}} \end{bmatrix} \begin{bmatrix} \mathbf{Q}^x \\ \mathbf{Q}^y \end{bmatrix} \tag{3.7}$$

where $\mathbf{B}(t)$ is the $n \times 1$ vector of B-spline basis functions, and $\mathbf{Q}^x$, $\mathbf{Q}^y$ are the vectors of control points consisting of the x and y coordinates respectively. We refer to a curve as defined by (3.7) as a contour.

A set of landmark points may contain many sets of control points for different B-splines. For example, in Figure 3.1, page 22, the subset of landmark points outlining the mouth of the person forms a set of control points. Similarly, the subset of points outlining the nose forms another set of control points. Now for the rest of the discussion in this section, we restrict the set of landmark points so that it contains only one set of control points. A shape parameter vector is mapped to a spline vector $\mathbf{Q} = \left[\mathbf{Q}^x,\ \mathbf{Q}^y\right]^{\mathrm{T}}$ by

$$\mathbf{Q} = \mathcal{S}(\boldsymbol{p}_s) = \bar{\boldsymbol{s}} + \Phi_s \boldsymbol{p}_s, \tag{3.8}$$

where $\bar{\boldsymbol{s}}$ and Φ_s are the same as previously defined. However, when mapped to landmark points, the shapes contain only a single set of control points. The space spanned by these spline vectors is the shape space. By restricting $\boldsymbol{p}_s$, $\mathbf{Q}$ is essentially a deformation of the template $\bar{\boldsymbol{s}}$, and the type of deformation allowed is determined by Φ_s.

Typically, the control points $\mathbf{Q}$ are not normalised with respect to scale, rotation, and location. Therefore, (3.8) yields landmark points.

3.2 Modelling of texture

Now that we have modelled the shape of an object, we construct a model for the texture. The texture of an object refers to the greyscale or colour pixels across it, but other texture representations such as wavelet coefficients (Larsen et al., 2007) can also be used. In general, the texture values are normalised to compensate for lighting variations.

3.2.1 Sampling the texture

Correspondences between points in different shapes are defined by corresponding landmark points. On the contrary, texture points are obtained within shapes that can have different deformations, scales and rotations. Therefore, care must be taken to sample these texture points consistently across different shapes.

The process of collecting texture points starts by calculating a warp from a reference shape, typically the mean shape $\bar{\boldsymbol{s}}$, to a shape in an image that contains the texture. This is illustrated in Figure 3.3. We denote this warp from the reference shape to the synthesised shape as $\mathcal{W}(\boldsymbol{j};\boldsymbol{p})$ where $\boldsymbol{p} = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_s^{\mathrm{T}}\right]^{\mathrm{T}}$. Using the warp, we can form a new image $\mathcal{I}'$ by going through the points in $\bar{\boldsymbol{s}}$, calculating their location in the synthesised shape and use the corresponding greyscale values to form $\mathcal{I}'$. In other words, we obtain

$$\mathcal{I}'(\boldsymbol{j}) = \mathcal{I}(\mathcal{W}(\boldsymbol{j};\boldsymbol{p})). \tag{3.9}$$

Here $\boldsymbol{j}$ is a two-dimensional index of the allowable pixel coordinates in the mean shape; we shall sometimes abuse the notation $\boldsymbol{j} \in \bar{\boldsymbol{s}}$ to indicate this. We use the notation $\mathcal{I}(\cdot)$ to index the pixels of an image.

Several warping methods exist (Mardia, 1998), but within the AAM framework a piece-wise affine warp is normally implemented. First, a Delaunay triangulation of the reference shape $\bar{\boldsymbol{s}}$ is calculated. Given three vertices $\boldsymbol{j}_1$, $\boldsymbol{j}_2$ and $\boldsymbol{j}_3$ of a triangle in $\mathcal{I}$, we can write any point $\boldsymbol{j}$ within that triangle as a linear combination of these vertices,

$$\begin{aligned} \boldsymbol{j} &= \boldsymbol{j}_1 + \eta_2(\boldsymbol{j}_2 - \boldsymbol{j}_1) + \eta_3(\boldsymbol{j}_3 - \boldsymbol{j}_1) \\ &= \eta_1\boldsymbol{j}_1 + \eta_2\boldsymbol{j}_2 + \eta_3\boldsymbol{j}_3, \end{aligned} \tag{3.10}$$

where $\eta_1 + \eta_2 + \eta_3 = 1$. The warp from $\bar{\boldsymbol{s}}$ to $\boldsymbol{s}$ is then calculated as

$$\begin{aligned} \boldsymbol{j}' &= \mathcal{W}(\boldsymbol{j};\boldsymbol{p}) \\ &= \eta_1\boldsymbol{j}_1' + \eta_2\boldsymbol{j}_2' + \eta_3\boldsymbol{j}_3' \end{aligned} \tag{3.11}$$

with $\boldsymbol{j}_1'$, $\boldsymbol{j}_2'$ and $\boldsymbol{j}_3'$ the vertices of the corresponding triangles in $\mathcal{I}'$. All that remains is to solve for the warping parameters η_i, $i = 1,\dots,3$ for a known point $\boldsymbol{j} = [x,y]^{\mathrm{T}}$; this can be easily carried out using (3.10) and the three vertices of the triangle.

Now we form a vector representation of $\mathcal{I}'$ and denote it by $\boldsymbol{g}_{\text{image}}$. This is the sampled texture.

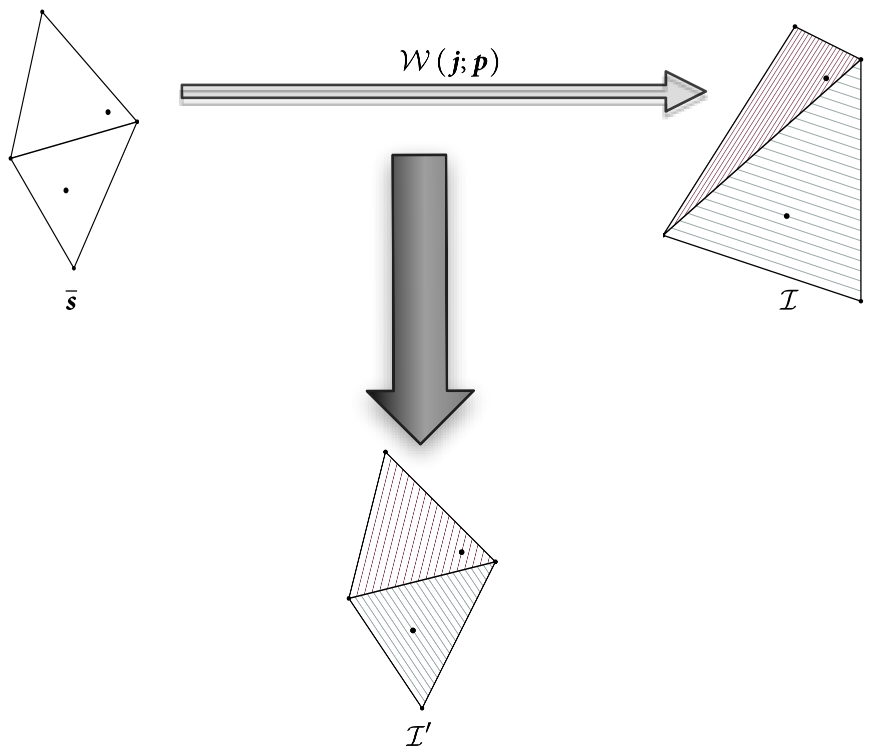


Figure 3.3: *Graphical illustration of the texture sampling procedure.*

3.2.2 Normalising the texture

After the pixels have been sampled into $\boldsymbol{g}_{\text{image}}$, a photometric normalisation is carried out to compensate for global changes in illumination,

$$\boldsymbol{g}_{\text{norm}} = \frac{\boldsymbol{g}_{\text{image}} - \beta\mathbf{1}}{\alpha}. \tag{3.12}$$

Here $\mathbf{1}$ is a vector of ones, $\alpha = \boldsymbol{g}_{\text{image}} \cdot \overline{\boldsymbol{g}}$ and $\beta = \frac{\boldsymbol{g}_{\text{image}} \cdot \mathbf{1}}{m}$, $\overline{\boldsymbol{g}}$ is the mean texture (to be defined shortly) and m is the number of texture points. We define the global texture normalisation $\boldsymbol{p}_t \triangleq \left[\alpha, \beta\right]^{\text{T}}$.

3.2.3 Constructing the model

Let the normalised texture of the object in question be described by a vector, $\boldsymbol{g}_i = [g_{i1}, g_{i2}, \ldots, g_{im}]^{\text{T}}$, $i = 1, \ldots, \text{N}_{\text{T}}$ with m the number of texture points. The mean texture of N_{T} normalised texture vectors is given by

$$\bar{\boldsymbol{g}} = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} \boldsymbol{g}_i. \tag{3.13}$$

Furthermore, the covariance matrix of the $\mathrm{N_T}$ texture vectors is

$$\Sigma_g = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} \left(\boldsymbol{g}_i - \bar{\boldsymbol{g}}\right)\left(\boldsymbol{g}_i - \bar{\boldsymbol{g}}\right)^{\mathrm{T}}. \tag{3.14}$$

PCA is performed on the covariance matrix Σ_g to obtain a basis Φ_g. New texture instances can be synthesised by

$$\mathcal{G}\left(\boldsymbol{p}_g\right) = \bar{\boldsymbol{g}} + \Phi_g \boldsymbol{p}_g. \tag{3.15}$$

Since Φ_g is the result of PCA, the dimension can be reduced by including only the eigenvectors with the largest eigenvalues, similar to (3.5).

Now that a shape and texture model exist, one can use them to describe an object in an image. Hence we use the shape deformation parameters, $\boldsymbol{p}_s$, the texture deformation parameters, $\boldsymbol{p}_g$, the global pose, $\boldsymbol{p}_p$, and the global texture normalisation, $\boldsymbol{p}_t$, to describe an object. This is referred to as an independent AAM. However, we can perform one more combination, resulting in the dependent AAM, discussed in the next section.

3.3 Modelling of the combination of shape and texture

A dependent AAM is constructed by forming a model parameter $\boldsymbol{p}_c$ obtained by combining the PCA deformation parameters into

$$\boldsymbol{b} = \begin{bmatrix} \Psi_s \boldsymbol{p}_s \\ \boldsymbol{p}_g \end{bmatrix}$$

with Ψ_s a weighting matrix between pixel intensities and pixel distances. The weighting matrix Ψ_s is given by

$$\Psi_s = \frac{\lambda_g}{\lambda_s} \mathrm{I}$$
$$\lambda_g = \sum_i \lambda_{gi}$$
$$\lambda_s = \sum_i \lambda_{si}$$

where λ_{si} and λ_{gi} are the eigenvalues of the shape and texture covariance matrices respectively. A third PCA is performed on the combined model parameters to obtain

$$\boldsymbol{b} = \Phi_c \boldsymbol{p}_c,$$

where Φ_c consists of eigenvectors. Writing $\Phi_c = \begin{bmatrix} \Phi_{c,s} \\ \Phi_{c,g} \end{bmatrix}$, it is now possible to generate new shape and texture instances by

$$\mathcal{S}(\boldsymbol{p}_c) = \bar{\boldsymbol{s}} + \Phi_s \Psi_s^{-1} \Phi_{c,s} \boldsymbol{p}_c \tag{3.16}$$
$$\mathcal{G}(\boldsymbol{p}_c) = \bar{\boldsymbol{g}} + \Phi_g \Phi_{c,g} \boldsymbol{p}_c \,. \tag{3.17}$$

Note from (3.16) and (3.17) that changing $\boldsymbol{p}_c$ varies both the shape $\mathcal{S}(\boldsymbol{p}_s)$ and the texture $\mathcal{G}(\boldsymbol{p}_c)$ of an object.

For notational convenience elsewhere, we define the AAM parameter $\boldsymbol{p}_a$ as

$$\boldsymbol{p}_a \triangleq \begin{cases} \boldsymbol{p}_c & \text{if a dependent AAM is used} \\ \left[\boldsymbol{p}_s^{\mathrm{T}}, \boldsymbol{p}_g^{\mathrm{T}}\right]^{\mathrm{T}} & \text{if an independent AAM is used.} \end{cases} \tag{3.18}$$

3.4 Fitting ASMs

This section discusses parameter estimation when only shape information is used to model a deformable object.

The simplest strategy for finding the model parameters is edge detection. An initialisation for the shape is assumed. A search along the normal line of each point on the initial shape is performed, looking for the strongest edge. The shape point is then moved to the point detected on the edge of the object. See Figure 3.4 for a graphical illustration.

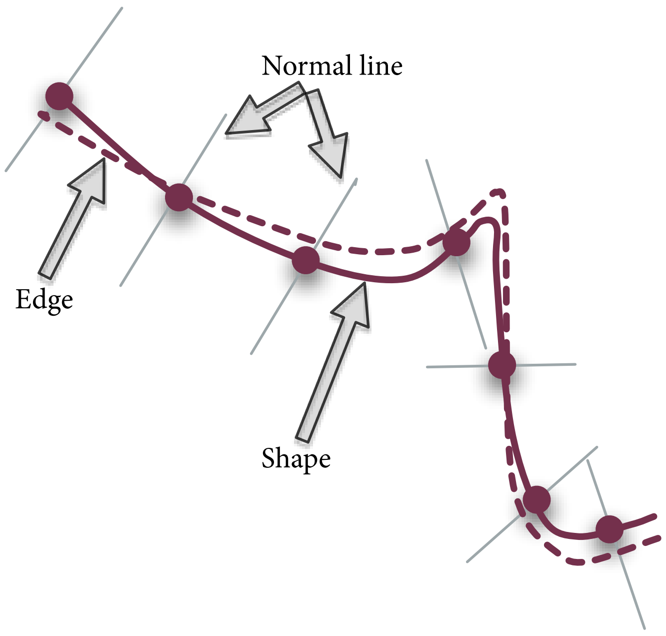


Figure 3.4: *Fitting an ASM. Searches are done along the normal lines to build a statistical profile of the greyscale values.*

The edge detection approach for finding the model parameters is not robust. First, an edge might not exist along the normal lines. Second, the best landmark point is not necessarily the point on the strongest edge, but may be a less prominent edge. To overcome these problems, a statistical model of grey level information is used. For each point $\boldsymbol{j}$ in the shape $\boldsymbol{s}_i$, $i = 1, \ldots, \mathrm{N_T}$, k pixels are uniformly sampled along the normal line. The gradient of these pixels is taken to compensate for global effects, such as lighting variations. Let the vector of gradients be $\boldsymbol{g}'_{ji}$. Thereafter, the pixels are normalised yielding

$$\boldsymbol{g}_{ji} = \frac{\boldsymbol{g}'_{ji}}{\left\| \boldsymbol{g}'_{ji} \right\|_1} . \tag{3.19}$$

It is assumed that the greyscale values of the pixels (more precisely the gradient of the greyscale values) along the normal lines are normally distributed. Sufficient statistics for the distribution of the $\boldsymbol{j}$th shape point are therefore given by the mean

$$\bar{\boldsymbol{g}}_j = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} \boldsymbol{g}_{ji} , \tag{3.20}$$

and the covariance

$$\mathrm{S}_j = \frac{1}{\mathrm{N_T}} \sum_{i=1}^{\mathrm{N_T}} \left(\boldsymbol{g}_{ji} - \overline{\boldsymbol{g}}_j\right) \left(\boldsymbol{g}_{ji} - \overline{\boldsymbol{g}}_j\right)^{\mathrm{T}}. \tag{3.21}$$

Given a new sampled shape with its pixel information, the quality of the fit is the probability under the distribution $\mathcal{N}\left(\overline{\boldsymbol{g}}_j, \mathrm{S}_j\right)$. To implement ASM fitting, an initial shape is assumed. Then K, (K $\gg k$) pixels are sampled along the normal line for each point $\boldsymbol{j}$ in the initial shape. Subsequently, using a sliding window of length k, k pixels are chosen and the quality of each fit is recorded. The shape point is then moved to the position where the quality of the fit is the best.

3.5 Fitting AAMs

During the fitting phase of AAMs, the model parameters $\boldsymbol{p} = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_a^{\mathrm{T}}\right]^{\mathrm{T}}$ are estimated that best represent an object in a new image $\mathcal{I}$ not contained in the original training set. The idea of AAM fitting is to vary $\boldsymbol{p}_a$ (optimise over $\boldsymbol{p}_a$) so that the shape and texture generated by (3.16) and (3.17) or (3.6) and (3.15) fit the object in the image as well as possible. Note that one can distinguish between dependent AAM fitting where (3.16) and (3.17) are used to generate the shape and texture respectively, or independent AAM fitting where (3.6) and (3.15) are used instead. However, most of the time the techniques are the same. For this reason, unless otherwise indicated, we shall discuss from now on, independent model fitting. Thus, for an independent model, the objective function that is minimised,

$$\begin{aligned} \mathrm{E} &= \sum_{j \in \overline{s}} \left[\mathcal{G}\left(\boldsymbol{j}; \boldsymbol{p}_g\right) - \mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j}; \boldsymbol{p}\right)\right)\right]^2 && (3.22) \\ &= \|\boldsymbol{g}_{\text{model}} - \boldsymbol{g}_{\text{image}}\|^2 && (3.23) \\ &= \|\delta \boldsymbol{g}\|^2 && (3.24) \end{aligned}$$

is the difference between the texture values generated by $\boldsymbol{p}_g$ and (3.15), denoted as $\boldsymbol{g}_{\text{model}}$, and the texture values in the image, $\boldsymbol{g}_{\text{image}}$. Note that the image texture values $\boldsymbol{g}_{\text{image}}$ for a specific value of $\boldsymbol{p}_s$ are the values sampled from the shape generated by (3.6) and then translated, scaled and rotated using the pose $\boldsymbol{p}_p$. We also explicitly indicate that $\mathcal{G}$ is indexed with $\boldsymbol{j}$. Indexing $\mathcal{G}$ is possible, since the texture, although synthesised, is an image. In summary, the optimisation over $\boldsymbol{p}_a$ and $\boldsymbol{p}_p$ minimises (3.24), *i.e.*, produces the best fit of texture values.

Two approaches exist to fit the AAM, namely discriminative and generative fitting. Discriminative fitting learns a fixed function (or a set of functions) that produce the optimal parameter updates given some observed features. Generative fitting, on the other hand, minimises the error between the synthesised texture and the warped images.

3.5.1 Discriminative fitting

Given an initial estimate $\widetilde{\boldsymbol{p}}_0$, discriminative fitting finds the optimal update $\delta\boldsymbol{p}$, such that $\boldsymbol{p} \leftarrow \widetilde{\boldsymbol{p}}_0 + \delta\boldsymbol{p}$. This is done iteratively so that the updates are given by

$$\boldsymbol{p} \leftarrow \widetilde{\boldsymbol{p}}_0 + \sum_{i=1}^{n_u} \delta\boldsymbol{p}_i \tag{3.25}$$

where n_u is the number of iterations.

The corrections $\delta\boldsymbol{p}_i$ are calculated through an update function $\mathcal{U}_i$ as follows,

$$\delta\boldsymbol{p}_i = \mathcal{U}_i\left(\mathcal{F}\left(\mathcal{I};\widetilde{\boldsymbol{p}}_i\right)\right). \tag{3.26}$$

Here $\mathcal{F}$ is a function that maps an image $\mathcal{I}$ with model parameters $\widetilde{\boldsymbol{p}}_i$, onto a set of features. Discriminative approaches learn a set of functions $\mathcal{U}_i$, $i = 1, \ldots, n_u$, given a training set formed by perturbed model parameters

$$\left\{\mathcal{F}\left(\mathcal{I};\boldsymbol{p}^* - \delta\boldsymbol{p}\right), \delta\boldsymbol{p}\right\} \tag{3.27}$$

where $\boldsymbol{p}^*$ is the optimal parameter setting.

The original AAM formulation (Cootes et al., 1998) assumes that there is a linear relation between the residual texture

$$\mathcal{F}\left(\mathcal{I};\boldsymbol{p}\right) = \mathcal{G}\left(\boldsymbol{j};\boldsymbol{p}_g\right) - \mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{p}\right)\right) \tag{3.28}$$

and the optimal parameter updates. Furthermore, it is assumed that the parameter updates are close to linear around the optimal parameter settings of an image. Using these assumptions, we have that

$$\begin{aligned}\delta\boldsymbol{p}_i &= \mathcal{U}_i\left[\mathcal{G}\left(\boldsymbol{p}_g\right) - \mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{p}_s\right)\right)\right] \\ &= \mathcal{U}_i\left(\delta\boldsymbol{g}\right).\end{aligned} \tag{3.29}$$

Given a training set of the form (3.27), estimates exist for $\delta\boldsymbol{p}_i$ and $\delta\boldsymbol{g}$. If it is assumed that $\mathcal{U}_i$ is a linear function, it can then be found using multivariate linear regression. This yields

$$\delta \boldsymbol{p}_i = \mathrm{R}_i \delta \boldsymbol{g} \tag{3.30}$$

where R_i is a matrix found using linear regression.

Alternatively, one can solve for $\mathcal{U}_i$ using support vector regression (Saragih and Göcke, 2006). Support vector regression, (for a tutorial see (Smola and Schölkopf, 2004)) uses the "kernel-trick" to build linear regressors in a high-dimensional feature space. The result is a non-linear regressor in input space. Applying this to AAMs, the update functions in (3.29) are obtained from support vector regression training.

Saragih and Goecke (2007) also proposed techniques based on solving $\mathcal{U}_i$ through boosting. All these techniques based on learning a non-linear update function, are referred to as iterative non-linear discriminant fitting methods.

Note that it is not necessary to learn an update function for $\boldsymbol{p}_g$, since its effect is captured by the function $\mathcal{W}$ if we assume the latter operates on the greyscale values of the image, *e.g.* (3.28) is used. The resulting update functions $\mathcal{U}_i$, $i = 1, \ldots, n_u$ are faster to learn; $\boldsymbol{p}_g$ can be solved through equation (3.15) after sampling $\boldsymbol{g}$ in the shape obtained by (3.6).

For further detail on discriminative fitting, the reader is referred to (Cootes et al., 1998; Stegmann, 2000; Saragih and Göcke, 2006).

3.5.2 Generative fitting

AAM fitting methods using the generative approach try to solve (3.22) directly, in contrast to the previous paragraph's approach that sought an optimal update by learning some update functions, $\mathcal{U}_i$, $i = 1, \cdots, n_u$.

To derive the generative update, also known as the fixed Jacobian approach (Cootes et al., 2001), the AAM residual is defined as

$$\boldsymbol{r}(\boldsymbol{p}) = \boldsymbol{g}_{\text{model}} - \boldsymbol{g}_{\text{image}}. \tag{3.31}$$

Subsequently, a first order Taylor approximation of (3.31) around $\boldsymbol{p}$ yields

$$\boldsymbol{r}(\boldsymbol{p} + \delta \boldsymbol{p}) = \boldsymbol{r}(\boldsymbol{p}) + \frac{\partial \boldsymbol{r}}{\partial \boldsymbol{p}} \delta \boldsymbol{p} \,. \tag{3.32}$$

Now the L_2-norm of (3.32) is minimised by setting $\boldsymbol{r}(\boldsymbol{p}+\delta\boldsymbol{p}) = 0$ and solving for $\delta\boldsymbol{p}$. We obtain

$$\delta\boldsymbol{p} = -\mathrm{R}\boldsymbol{r}(\boldsymbol{p}) \tag{3.33}$$

$$\mathrm{R} = \left(\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{p}}^{\mathrm{T}}\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{p}}\right)^{-1}\frac{\partial\boldsymbol{r}}{\partial\boldsymbol{p}}^{\mathrm{T}}. \tag{3.34}$$

The matrix R is expensive to calculate. Therefore the assumption is made that it is constant and can be pre-computed. This also leads to the name for the method, namely the fixed Jacobian approach. This method gives generally smaller fitting errors than the linear regression method (Cootes et al., 2001).

Matthews and Baker (2004) showed that the assumption of a fixed Jacobian is wrong and one cannot expect in general a small fitting error. However, if the assumption is not made, the AAM fitting becomes computationally too expensive. For this reason, they proposed the inverse compositional approach—another generative fitting technique. The key difference between the fixed Jacobian and the inverse compositional approach is that the warp $\mathcal{W}(\boldsymbol{j};\boldsymbol{p})$ is updated instead of the parameter vector. This modification has been proven to be equivalent when solving the fitting problem (Baker and Matthews, 2004). Here $\boldsymbol{p}$ is a concatenation of the shape and pose vectors. The warp is updated as

$$\mathcal{W}(\boldsymbol{j};\boldsymbol{p}) \leftarrow \mathcal{W}(\boldsymbol{j};\boldsymbol{p}) \circ \mathcal{W}(\boldsymbol{j};\delta\boldsymbol{p}), \tag{3.35}$$

where $\mathcal{W}(\boldsymbol{j};\boldsymbol{p})$ is a warp function such that (3.9) holds, $\circ$ is the compositional operator [1].

To simplify our discussion of the inverse compositional algorithm, we look at a simpler form of the problem where the appearance variation is ignored, *i.e.*, we minimise

$$\sum_{\boldsymbol{j}\in\bar{\boldsymbol{s}}}\left[\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\boldsymbol{j};\boldsymbol{p}))\right]^2. \tag{3.36}$$

Note that in (3.36), it is explicitly shown that $\bar{\boldsymbol{g}}$ is an image that can be indexed with $\boldsymbol{j}$. The problem in (3.36) is also known as Lukas-Kanade image alignment (Lucas and Kanade, 1981). Minimising (3.36) is a non-linear optimisation problem. In the Lukas-Kanade algorithm, however, it is assumed that an initial value for the parameters $\boldsymbol{p}$ is known and an update $\delta\boldsymbol{p}$ is then sought. In other words, the problem changes to the minimisation of

$$\sum_{\boldsymbol{j}\in\bar{\boldsymbol{s}}}\left[\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\boldsymbol{j};\boldsymbol{p}+\delta\boldsymbol{p}))\right]^2 \tag{3.37}$$

[1] $f(x) \circ g(x) \triangleq f(g(x))$ with f and g functions on x

and solving for $\delta \boldsymbol{p}$. The first-order Taylor expansion of (3.37) around $\boldsymbol{p}$ yields

$$\sum_{\boldsymbol{j} \in \bar{\boldsymbol{s}}} \left[\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{p} + \delta \boldsymbol{p})) - \nabla \mathcal{I} \frac{\partial \mathcal{W}}{\partial \boldsymbol{p}} \delta \boldsymbol{p} \right] \tag{3.38}$$

where $\nabla \mathcal{I}$ is the image gradient. An analytic solution for (3.38) exists and is computed iteratively. The solution is computationally expensive though, since $\nabla \mathcal{I}$ and $\frac{\partial \mathcal{W}}{\partial \boldsymbol{p}}$ depend on $\boldsymbol{p}$ and should be recalculated on each iteration.

The inefficient computation inspired the next idea of updating the warp instead of the parameters; the algorithm is known as forward compositional image alignment. Now the problem is the minimisation of

$$\sum_{\boldsymbol{j} \in \bar{\boldsymbol{s}}} [\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\mathcal{W}(\boldsymbol{j}; \delta \boldsymbol{p}); \boldsymbol{p}))]^2 \tag{3.39}$$

and then updating the warp using (3.35). Similar to what we did before, a first-order Taylor approximation of (3.39) is calculated, given by

$$\begin{aligned} &\sum_{\boldsymbol{j} \in \bar{\boldsymbol{s}}} \left[\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{0})); \boldsymbol{p}) - \nabla \mathcal{I}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{p};)) \frac{\partial \mathcal{W}}{\partial \boldsymbol{p}} \delta \boldsymbol{p} \right] \\ = &\sum_{\boldsymbol{j} \in \bar{\boldsymbol{s}}} \left[\bar{\boldsymbol{g}}(\boldsymbol{j}) - \mathcal{I}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{p})) - \nabla \mathcal{I}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{p};)) \frac{\partial \mathcal{W}}{\partial \boldsymbol{p}} \delta \boldsymbol{p} \right]. \end{aligned} \tag{3.40}$$

We assumed that $\mathcal{W}(\boldsymbol{j}; \boldsymbol{0})$ is the identity warp. Note that the gradient is calculated at $\mathcal{W}(\boldsymbol{j}; \boldsymbol{p})$. Furthermore, the Jacobian is evaluated at $(\boldsymbol{j}; \boldsymbol{0})$ and can therefore be pre-computed.

The final refinement, known as inverse compositional image alignment, reverses the role of the image $\mathcal{I}$ and the mean template $\bar{\boldsymbol{g}}$. The alignment problem is then expressed as the minimisation of

$$\sum_{\boldsymbol{j} \in \bar{\boldsymbol{s}}} [\mathcal{I}(\mathcal{W}(\boldsymbol{j}; \boldsymbol{p})) - \bar{\boldsymbol{g}}(\mathcal{W}(\boldsymbol{j}; \delta \boldsymbol{p}))] \tag{3.41}$$

and updating the warp as

$$\mathcal{W}(\boldsymbol{j}; \boldsymbol{p}) \leftarrow \mathcal{W}(\boldsymbol{j}; \boldsymbol{p}) \circ \mathcal{W}(\boldsymbol{j}; \delta \boldsymbol{p})^{-1}. \tag{3.42}$$

A first-order Taylor approximation of (3.41) gives

$$\begin{aligned}
&\sum_{j\in\bar{s}}\left[\mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{p}\right)\right)-\bar{\boldsymbol{g}}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{0}\right)\right)-\nabla\bar{\boldsymbol{g}}\frac{\partial\mathcal{W}}{\partial\boldsymbol{p}}\delta\boldsymbol{p}\right]\\
=&\sum_{j\in\bar{s}}\left[\mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{p}\right)\right)-\bar{\boldsymbol{g}}\left(\boldsymbol{j}\right)-\nabla\bar{\boldsymbol{g}}\frac{\partial\mathcal{W}}{\partial\boldsymbol{p}}\delta\boldsymbol{p}\right]. \qquad (3.43)
\end{aligned}$$

Since $\bar{\boldsymbol{g}}$ is constant, it can be pre-computed and with the forward compositional algorithm, $\frac{\partial\mathcal{W}}{\partial\boldsymbol{p}}$ can also be pre-computed. A slight complication is the inverse warp needed to update $\mathcal{W}$. However, it can be efficiently computed.

The inverse compositional algorithm is applied with the AAM formulation resulting in a fast fitting approach. In other words, (3.22) is minimised directly. For a complete derivation, the reader is referred to (Matthews and Baker, 2004).

In both fitting approaches, an initial estimate is assumed to be available. When this initial value is far from the optimal solution, the fitting techniques often fail. Consequently, AAM fitting techniques are sensitive to initialisations.

3.6 Summary

AAMs provide a general framework to track or segment different types of objects. Furthermore, no parameters need to be specified by an expert to use them. On the downside, AAMs require objects to have distinct features/outlines and there is a training phase involved. Also, a good initialisation is required for the fitting algorithm.

We have not presented any AAM fitting examples in this chapter. However, in the next chapter, when tracking is discussed, we shall demonstrate how AAMs can be used to track an object.

Chapter

4 Tracking

Glücklich ist, wer vergisst, was nicht mehr zu ändern ist.

— Johann Strauss

The aim of this dissertation is the tracking of objects. To be more precise, we are interested in tracking the outline of an object, *i.e.*, a contour. Here we confine ourselves to tracking objects that can be described by a model, for example constructing a model of the facial features of a person allows us to accurately track faces.

4.1 A taxonomy of trackers

Tracking algorithms have been widely studied. In this section, an overview of trackers is presented. In addition, the work that is discussed in this chapter, is placed within the context of the existing tracking algorithms. A taxonomy of trackers, taken from (Yilmaz et al., 2006), and illustrated in Figure 4.1, is used to accomplish these objectives. This overview is not complete; for a comprehensive survey the reader is referred to (Yilmaz et al., 2006).

Object tracking algorithms are built on the representation of an object. This representation is employed to classify trackers.

Point trackers track a single point or a set of points. At each consecutive frame, the set of interest points are found using some interest point detector, such as the Harris point detector (Harris and Stephens, 1988). These points are then matched with the interest points in the previous frame.

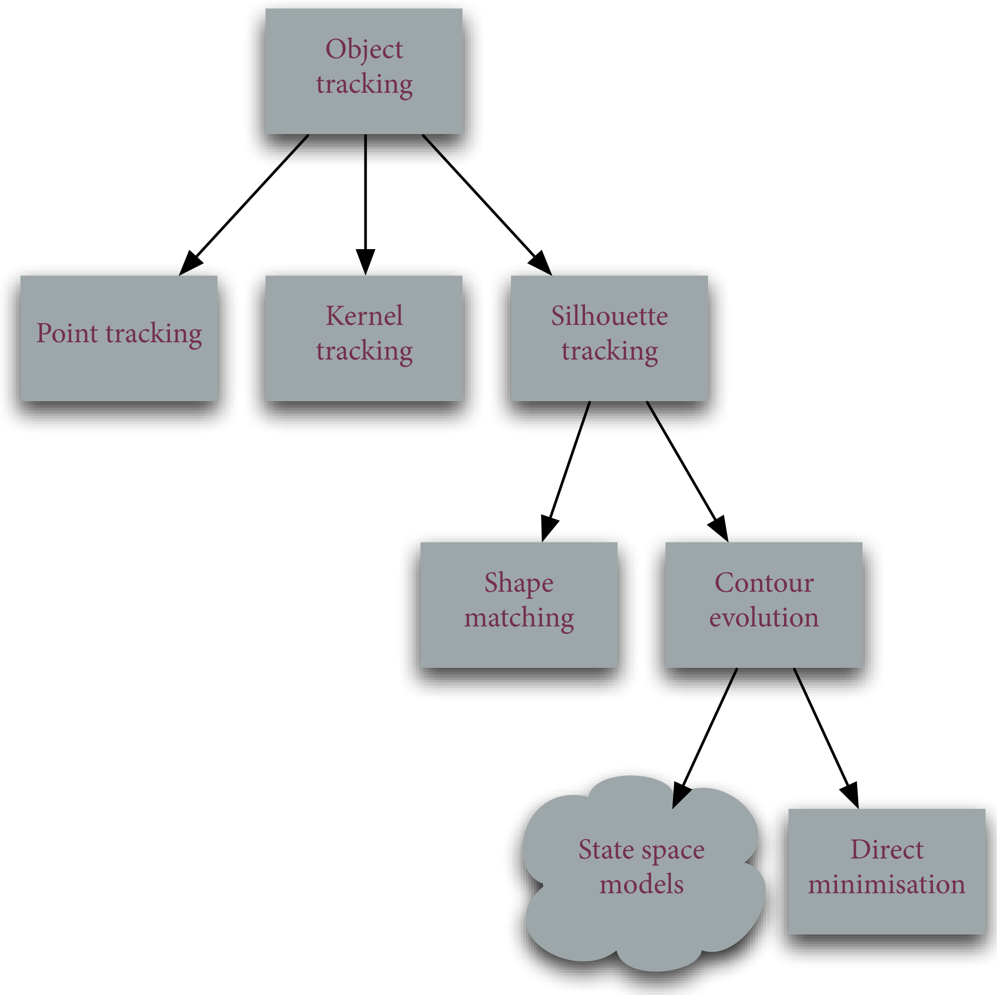


Figure 4.1: *A taxonomy of trackers. The region in the cloud indicates the work in this chapter.*

Examples of such trackers are (Sethi and Jain, 1987) and (Veenman et al., 2001). One can also use temporal filtering to move the points from one frame to the next (Broida and Chellappa, 1986).

Kernel trackers use a kernel based on an object's shape and appearance. The most popular kernel is a rectangle or elliptical region of interest around the object in question, together with a histogram of the pixel values within this region. These trackers are often referred to as blob trackers. A kernel tracker minimises the differences between the histograms in one frame and the next, by translating, scaling and rotating the region of interest. An example of such a tracker is the mean-shift tracker (Comaniciu and Meer, 1999). Often a PF is used to predict the translation, scale and rotation parameters (Nummiaro et al., 2002). Similarly, optical flow can be used to calculate the translation of the region of interest (Shi and Tomasi, 1994).

Silhouette trackers track the complete outline of an object. Typically, this is done by fitting a contour to the outline of an object. An alternative is to use a shape template and match this template to the underlying image on each consecutive frame (Huttenlocher et al., 1993).

When an object is described by a contour, this contour must evolve from one frame to the next, *i.e.*, the contour must be fitted in each frame. Evolution of contours is done by state space models, such as (Isard and Blake, 1998a) where a PF is used to predict and find the parameters of the control-points of a B-spline fitted around an object. Alternatively, a contour can be evolved by direct minimisation. Typically, some energy functional defined over the object's boundary is minimised. Examples include the snakes of Kass et al. (1987) and the work of Yilmaz et al. (2004).

Point trackers are suitable to track small rigid objects. Kernel trackers track rigid objects or even more complex objects if one is only interested in the centre of gravity with knowledge of scale and rotation. Silhouette trackers are appropriate to track non-rigid objects with complex boundaries.

In this chapter, the shape model of an AAM is used to describe an object. Thus, referring to the taxonomy in Figure 4.1, we perform silhouette tracking. The shape of an AAM forms contour representations of the underlying object. Therefore, we confine ourselves to tracking algorithms based on contour evolution. We use a PF to evolve the shape from one frame to the next. A PF leads to a state space approach. In summary, given the taxonomy of trackers, we use state space models to evolve contours resulting in a silhouette tracker.

4.2 Deterministic and stochastic tracking

The trackers described in the previous section can also be divided into those with temporal filtering and those without. This leads to an alternative taxonomy: deterministic and stochastic tracking. In deterministic tracking, some function, defined over the object's outline or greyscale values, is optimised in order to obtain the location and description of the object in the image domain. Stochastic tracking techniques usually employ probabilistic temporal filtering techniques to predict, and so restrict, the regions where the functions defined by deterministic tracking are then optimised.

Deterministic approaches in general follow a bottom-up approach where only certain parts of the image are used to directly solve the inverse problem of estimating the features of an object from the image. If a solution is found, it is accurate since low-level information is being used. However, these approaches offer limited robustness, since ambiguities that appear over time cannot be handled well. For example, in a small region of an image, it is difficult to decide if an edge is part of the object or the background.

Stochastic tracking uses a top-down approach where all the information in the image is used to render from feature space to image domain and measuring the quality of the hypothesis by comparing it with the actual image. This can either be done directly in the image domain or in feature space, *e.g.* on histograms, and is typically done with a feedback loop where the measurements are used to update the initial value of the features.

Top-down analysis offers better robustness than bottom-up approaches. The initial values of the features are important and if the values are not good estimates, the subsequent updates through the feedback loop might fail. A remedy, also used in this dissertation, is a combination of top-down and bottom-up processing, in the spirit of analysis by synthesis (Yuille and Kersten, 2006) (see Figure 4.2).

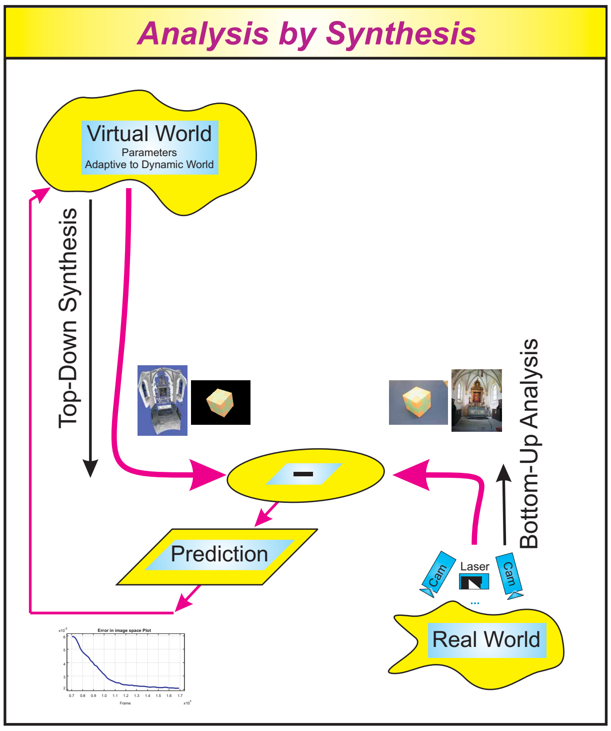


Figure 4.2: *Analysis by synthesis.*

A popular deterministic approach to track contours, is snakes (Kass et al., 1987). A curve is defined over image features such as boundaries. The energy of this curve is then minimised by the snake algorithm. This energy is defined in terms of the length, curvature, and elasticity of the curve.

Other deterministic contour tracking approaches proceed by fitting a curve through image feature points. These fitting techniques include standard interpolation, spline curves and subdivision; a better method is to fit a curve according to some basic shape (Blake et al., 1995; Blake and Isard, 1998).

AAMs provide yet another deterministic tracking approach (Stegmann, 2001; Birkbeck and Jagersand, 2004). They allow the simultaneous tracking of object contours and internal object features. AAMs offer specialisation to a particular tracking problem as they follow a model-based approach. AAMs follow the top-down idea of rendering the model from a feature set and then evaluating this model in the image domain instead of solving the inverse problem, *i.e.*, extracting parameters from the image directly. This deterministic approach is presented in Section 4.4.

An example of a stochastic tracker is the work of Isard and Blake (1998a): they used the PF to track contours described by B-splines. This method combines measurements sampled from a B-spline's control points with temporal filtering. Thereafter, Isard and Blake (1998b) combined colour information with the contour information in a PF. A PF that combines more then one feature in its state is known as a multi-cue PF.

An AAM typically includes an iterative optimisation (*e.g.* Newton/Gauss-Newton) to improve the quality of the AAM parameters by searching along the gradient direction for an improved feature vector. This offers high precision, since a local optimum will always be found successfully. However, it is not necessarily the global optimum, making the robustness of the AAM an issue. Thus, although the AAM runs top-down, the feedback loop of optimising the parameter vector works bottom-up, which leads to the problem of getting trapped in local optima. Moreover, the top-down processing requires good initialisation. A combination of PFs and AAMs can lead to the best of both worlds—the robustness of PFs combined with the precision of AAMs.

We present the AAM and PF combination in Section 4.5.

4.3 Evaluation of tracking algorithms

Throughout the rest of this chapter and the next, we present and evaluate tracking algorithms. At this point, it is therefore necessary to introduce the evaluation protocol.

Tracking algorithms can be evaluated qualitatively or quantitatively. When an algorithm is evaluated qualitatively, the question is the ability of the algorithm to track an object in a video sequence, as judged by a human. Furthermore, if the tracker loses the target, for example, due to occlusions, the question is asked whether the tracker can recover from this by finding the object again. This evaluation protocol is acceptable, since the main goal of a tracker is to track an object.

We apply this evaluation strategy in this chapter.

When a tracking algorithm is evaluated quantitatively, some metric is used that describes how well the tracker performs. Most quantitative performance metrics require ground-truth. In the case of tracking algorithms, the gathering of ground-truth is a cumbersome procedure. For example, if a 10s video clip is recorded at a frame rate of 15Hz, it would require 150 frames to be annotated manually. Moreover, this manual annotation, especially if contours are tracked, might not be accurate.

In the next chapter, where we evaluate the algorithms quantitatively, we obtain the ground-truth by AAM fitting methods. Thus, in each frame an AAM is manually fitted to obtain the location and outline of the object being tracked. Therefore, we assume that the contours found by the AAM is our reference of accuracy. Tracking algorithms are tested on how well they perform to a manually fitted AAM. This approach has some disadvantages. An AAM might not have found the exact location of an object accurately (due to the nature of the AAM fitting algorithms). However, since we gather the ground-truth manually, great care is taken to ensure that the AAMs are fitted as well as possible and are not stuck in local optima. Secondly, in the presence of occlusions, the AAM will not be able to produce ground-truth. We shall, nonetheless, apply this approach to obtain ground-truth, since we aim to mimic AAMs' accuracy.

Sometimes, we use the measure for evaluating an AAM's fit to evaluate our tracking algorithms. This measure is the quadratic norm, (3.22)–(3.24), page 31, repeated here for convenience,

$$\begin{aligned} \mathrm{E} &= \sum_{j \in \bar{s}} \left[\mathcal{G}\left(\boldsymbol{j}; \boldsymbol{p}_g\right) - \mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j}; \boldsymbol{p}\right)\right) \right]^2 \\ &= \|\boldsymbol{g}_{\text{model}} - \boldsymbol{g}_{\text{image}}\|_2 \\ &= \|\delta \boldsymbol{g}\|_2 \,. \end{aligned}$$

The norm (3.24) above is not robust if outliers exists, *e.g.* occlusions. For this reason, the Lorentzian norm (Stegmann, 2000) is used instead, defined by

$$\mathrm{E}^2_{\text{Lor}} = \log\left(1 + \frac{\mathrm{E}}{2\sigma_s^2}\right) \tag{4.1}$$

where σ_s^2 is the scale parameter that discards outliers. This measure does not incorporate ground-truth. Notwithstanding, it gives an indication of how well the AAM fits the underlying object from one frame to the next.

Methods exist to measure performance without ground-truth (Erdem et al., 2004). These methods make the assumption that the statistics of the object being tracked do not change throughout the video sequence. Since these methods are still actively investigated, we shall not use them.

Apart from a suitable performance measure, the sequences used for evaluation of the tracking algorithms, must also be selected. Robustness can be compromised due to cluttered backgrounds, fast movements, occlusions, and object deformations (Yilmaz et al., 2006). Therefore, as an example of tracking in the presence of occlusions and object deformations, the facial outlines of a person are tracked. Selected frames are illustrated in Figure 4.3. To test for fast movements, we track a moving hand, as shown in Figure 4.4. Moreover, all these videos are taken against a cluttered background.

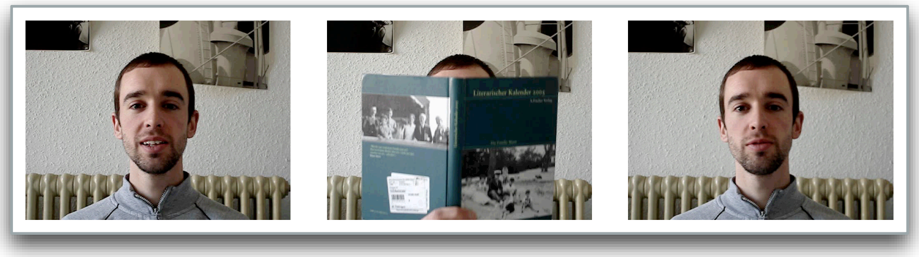


Figure 4.3: *Selected frames from a video sequence used to test for occlusions and object deformations.*

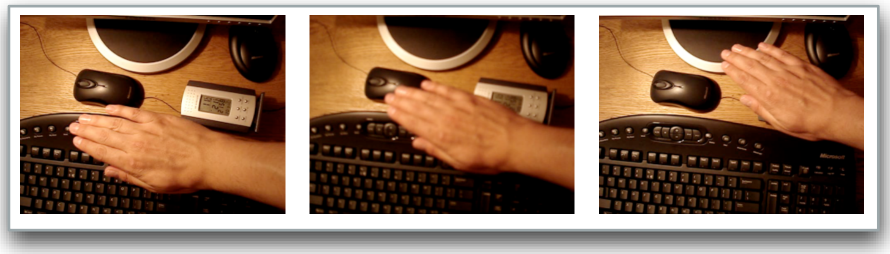


Figure 4.4: *Selected frames from a hand moving against a cluttered background.*

4.4 Deterministic tracking using AAMs

AAMs can be used to perform tracking without temporal filtering. However, before dwelling upon this topic, it is instructive to formalise AAM tracking. Let the state of a tracker at time t be given by

$$\boldsymbol{x}_t = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_a^{\mathrm{T}}\right]^{\mathrm{T}} , \tag{4.2}$$

where $\boldsymbol{p}_p$ and $\boldsymbol{p}_a$ are the AAM's pose and model parameters respectively as described by (3.1) and (3.18). The aim of any AAM tracking method is to find the optimal values of $\boldsymbol{x}_t$ given a new image (an image that is not in the training set) at time t. The parameters are optimal if the AAM fitting error (3.24) is as small as possible.

During deterministic tracking using AAMs, the model parameters $\boldsymbol{p}_a$ and pose $\boldsymbol{p}_p$ from the current frame are simply used to initialise the AAM search in the next frame. Thus, during deterministic tracking, the state of the tracker, $\boldsymbol{x}_t$, is updated as

$$\boldsymbol{x}_t = \widehat{\boldsymbol{x}}_{t-1} + \delta\boldsymbol{x} . \tag{4.3}$$

Here $\widehat{\boldsymbol{x}}_{t-1}$ contains the optimal values of the AAM at time $t-1$. When discriminative fitting is assumed, using (3.25) and (3.26),

$$\delta\boldsymbol{x} = \sum_{i=1}^{n_u} \mathcal{U}_i \left(\mathcal{F}\left(\mathcal{I}; \widetilde{\boldsymbol{p}}_i\right)\right) , \tag{4.4}$$

where $\widetilde{\boldsymbol{p}}_0 = \widehat{\boldsymbol{x}}_{t-1}$. When generative fitting is used, $\delta\boldsymbol{x}$ is calculated by minimising the differences in texture values, *i.e.*, (3.22) is minimised.

Results and examples

Video sequences for the results presented in this dissertation, are available at Hoffmann (2009).

Figure 4.5 shows selected frames when the facial features of a person are tracked. Note that the tracker fails under occlusions.

Figure 4.6 illustrates selected frames as a hand moves against a cluttered background. The tracker is not able to track robustly when the object moves fast.

The failure of the AAM algorithm in the examples presented is inherent to the AAM fitting algorithm. As discussed before, a good initialisation is required for an AAM to segment an object correctly. However, in cases of fast movements, the AAM parameters in the current frame will not necessarily be a good initialisation for the next frame. Moreover, in cases of occlusion, no optimum exists if the differences in texture values $||\boldsymbol{g}_{\text{model}} - \boldsymbol{g}_{\text{image}}||_2$ is used as the optimisation criterion. For these reasons, temporal filtering in the form of the particle filter is used to increase

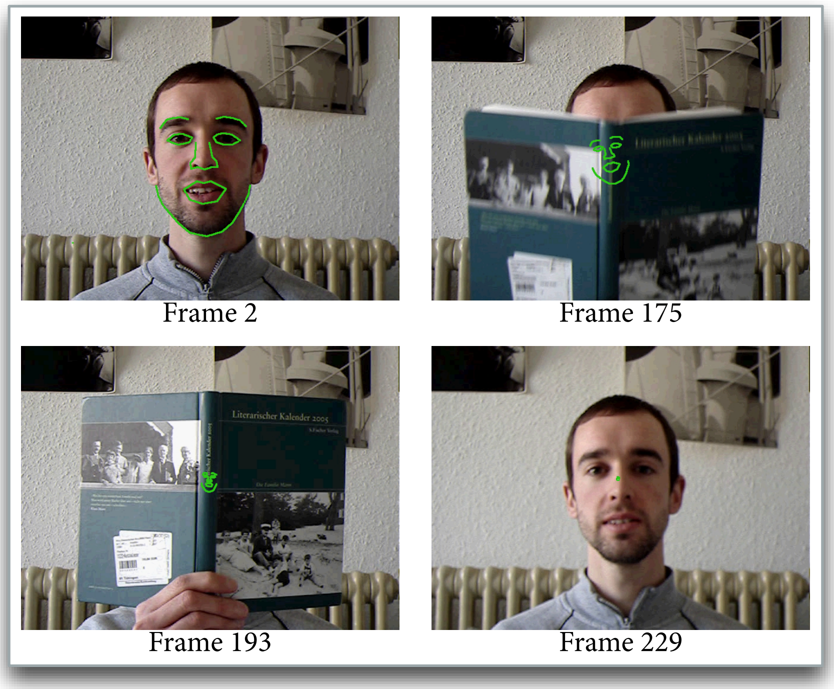


Figure 4.5: *Tracking the facial features of a person.*

the robustness of the AAM tracker—this is discussed in the next section.

4.5 Stochastic tracking using AAMs

We showed in the previous section that a deterministic AAM tracker fails when an object is being tracked in the presence of occlusions or when the object moves fast. In other tracking paradigms, such as active contours (Isard and Blake, 1998a) and blob tracking (Nummiaro et al., 2002; Pérez et al., 2002), PFs have been successfully applied to overcome tracking failures. We shall proceed to develop a similar approach.

The use of a PF in conjunction with AAMs was first proposed by Hamlaoui and Davoine (2005) in the context of facial feature tracking. They built their work on Zhou et al. (2004) to obtain an AAM and PF combination. Zhou et al. (2004) used appearance models (not AAMs) together with a PF to perform tracking. AAMs and PFs were also used by Bagnato et al. (2007) to track faces of infants.

The basic part of our work shares some similarities with Hamlaoui and Davoine (2005). The main difference is that we use multiple dynamic models. Furthermore, we use a different robust

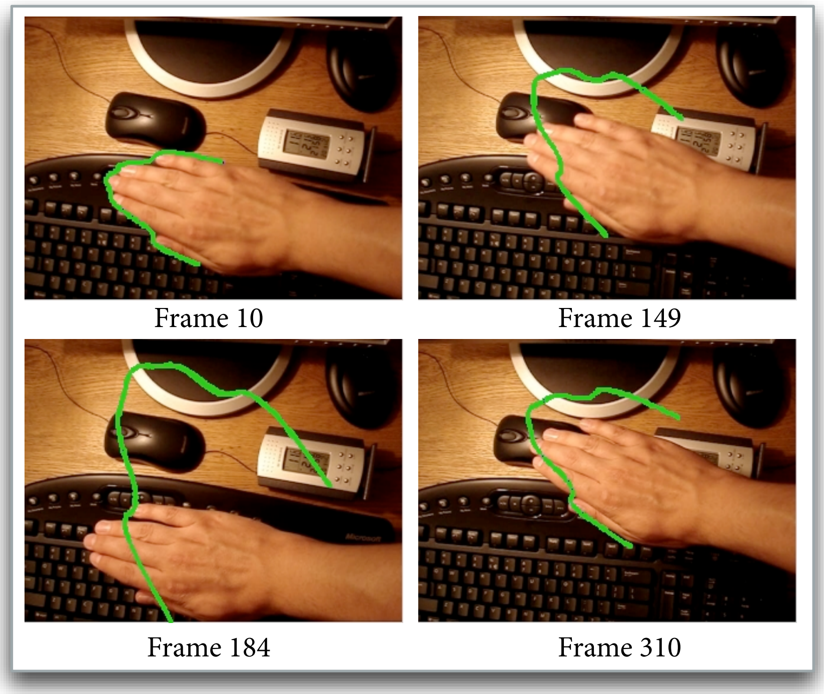


Figure 4.6: *Selection of result frames to indicate the performance of the deterministic AAM tracker. In frames 149, 184, and 310 the tracker loses its target due to fast movements.*

norm (Lorentzian norm) based on the extensions done by Stegmann (2000). We detect occlusions and then switch to a different *motion* model. On the contrary, Hamlaoui and Davoine modify their *measurement* model in the PF to deal with occlusions. We find the parameters needed for occlusion detection by training. We propose a local optimisation step in which the standard AAM optimisation routines are used to enhance the AAM particle filter tracker—the PFAAM. We also extend the work to include a non-linear AAM formulation. In Section 4.6 we shall provide a comparison between our work and Hamlaoui and Davoine (2005).

In Section 4.5.1, we detail the combination of PFs and AAMs. In Section 4.5.2, we define occlusion detection for our problem. Thereafter, in Section 4.5.3, we discuss tracking when a smaller state space is used. This is typically the situation when the iterative non-linear or inverse compositional fitting methods are applied. The local optimisation procedure—the standard AAM optimisation routines are used to enhance the AAM particle filter—is discussed in Section 4.5.4. Section 4.5.5 details the case when AAMs are used in conjunction with active contours.

4.5.1 PFs and AAMs: The PFAAM

Particle filters, as described in Chapter 2, can be applied to a general class of problems. For any specific problem, the state vector $\boldsymbol{x}_t$, the process model, and the measurement model need to be specified.

The process model can be described as the process function (2.1), repeated here for convenience,

$$\boldsymbol{x}_t = \boldsymbol{f}_{t-1}\left(\boldsymbol{x}_{t-1}, \boldsymbol{v}_{t-1}\right).$$

An equivalent specification for the process model is the transitional pdf $p\left(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}\right)$. Similarly, the measurement model is given by the measurement equation (2.2),

$$\boldsymbol{z}_t = \boldsymbol{g}_t\left(\boldsymbol{x}_t, \boldsymbol{w}_t\right)$$

or the likelihood $p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)$.

The process function $\boldsymbol{f}_{t-1}$ and the measurement function $\boldsymbol{g}_t$ are dependent on the tracking problem, and the chosen features. For example, in PFs working on colour distributions, the process function predicts the region of interest according to the position information contained within the PF's state. The measurement function can be any similarity measure that is accessible through measurements on the underlying image. The similarity measure can be calculated in the image space or some feature space, *e.g.* the Bhattacharrya distance in colour histogram space.

We now introduce the PF using AAMs.

The state vector

If an object is tracked with an AAM and PF, we assume that the AAM pose $\boldsymbol{p}_p$ and model parameters $\boldsymbol{p}_c$ at time t are distributed according some pdf $p\left(\left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_c^{\mathrm{T}}\right]^{\mathrm{T}}, \boldsymbol{z}_{0:t}\right)$, where the time dependence of $\boldsymbol{p}_p$ and $\boldsymbol{p}_c$ is not indicated in order to keep the notation uncluttered. Thus we are uncertain of the exact values of $\boldsymbol{p}_p$ and $\boldsymbol{p}_c$. The state is therefore a combination of the AAM pose and model parameters at time t and is given by

$$\boldsymbol{x}_t = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_c^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{4.5}$$

Note that the pdf over $\boldsymbol{x}_t$ is not available in closed form and is therefore represented by a set of particles. The PF propagates this pdf from time t to $t+1$.

In (4.5), we assume a dependent AAM. Using the parameters contained in the state $\boldsymbol{x}_t$, it is possible to synthesise a shape using (3.16) and a texture using (3.17). Furthermore, we can sample a texture from the generated shape after it has been scaled, rotated and translated using the pose parameters $\boldsymbol{p}_p$.

The value of $\boldsymbol{x}_0$ is obtained by fitting an AAM at time 0. The AAM is initialised manually. Then we assume that $p(\boldsymbol{x}_0)$ is distributed normally with mean $\boldsymbol{x}_0$ and variance Σ_0.

The process model

The process model is given by a first-order Markov process

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \boldsymbol{\omega}_{t-1} + \mathrm{S}_{t-1}\boldsymbol{v}_{t-1} \tag{4.6}$$

where $\boldsymbol{x}_{t-1}$ is the state at time $t-1$, $\boldsymbol{\omega}_{t-1}$ is the deterministic update or feedback (that will be discussed shortly), S_{t-1} is the process noise covariance and $\boldsymbol{v}_{t-1}$ is a vector of normally distributed white noise, with zero mean and unit covariance.

The deterministic update $\boldsymbol{\omega}_{t-1}$ is estimated as the optimal parameter update (3.26)

$$\boldsymbol{\omega}_{t-1} \triangleq \left[\delta\boldsymbol{p}_p^{\mathrm{T}}, \delta\boldsymbol{p}_c^{\mathrm{T}}\right]^{\mathrm{T}} \tag{4.7}$$

$$= \sum_{j=1}^{n_u} \mathcal{U}_j\left(\mathcal{F}\left(\mathcal{I}; \widetilde{\boldsymbol{p}}_j\right)\right) \tag{4.8}$$

when an AAM fitting is performed in the previous frame with the previous state as initialisation, *i.e.*, $\widetilde{\boldsymbol{p}}_0 = \boldsymbol{x}_{t-1}$. Note if a linear model is used, $\mathcal{U}_i$ is the linear regression matrix. A non-linear model is obtained if $\mathcal{U}_i$ is estimated with some non-linear predictor. In (4.8) $\mathcal{F}$ is a function of the image $\mathcal{I}$ at time $t-1$ and the previous state $\boldsymbol{x}_{t-1}$.

When implementing and estimating the model (4.6), we use the following heuristic. We assume that the change in the state from one frame to the next is small, *i.e.* we assume that state at time t will be near the previous state. Let us denote the estimate of $\boldsymbol{x}_{t-1}$ by $\widehat{\boldsymbol{x}}_{t-1}$. We rewrite the model (4.6) as

$$\boldsymbol{x}_t = \widehat{\boldsymbol{x}}_{t-1} + \boldsymbol{\omega}_{t-1} + \mathrm{S}_{t-1}\boldsymbol{v}_{t-1}. \tag{4.9}$$

We call this the AAM process model. We found that in practise, (4.9) performed better in terms of tracking accuracy.

However, this model (4.9) is sensitive to occlusions. Although (4.9) gives an accurate prediction, the update $\boldsymbol{\omega}_{t-1}$ makes it vulnerable to occlusions because standard AAM fitting methods calculate this update and the procedure is initialised with the estimate $\widehat{\boldsymbol{x}}_{t-1}$. During occlusions, the standard AAM fitting methods might not find a solution.

Furthermore, if a PF is used to propagate the model (which is in the fact the case) and suppose that the process model is the proposal density, $\widehat{\boldsymbol{x}}_{t-1}$ has the same value across all particles. During fast movements or occlusions, a more diverse model (a model with particles exploring more of the state space) is preferred. For these reasons, a second process model given by a second-order auto-regressive process

$$\boldsymbol{x}_t = \overline{\boldsymbol{x}} + \mathrm{A}_2\left(\boldsymbol{x}_{t-2} - \overline{\boldsymbol{x}}\right) + \mathrm{A}_1\left(\boldsymbol{x}_{t-1} - \overline{\boldsymbol{x}}\right) + \mathrm{B}_0\boldsymbol{v}_t \tag{4.10}$$

is used in parallel with (4.9). The parameters A_2, A_1, B_0, and $\overline{\boldsymbol{x}}$ are learnt off-line. For detail on the learning procedure, see (Blake and Isard, 1998, Chapter 11, page 244).

A fixed number of particles is propagated through each model. We show in the next chapter how we propagate particles through the two process models.

The measurement model

The purpose of the measurement model is to classify how well the current measurement $\boldsymbol{z}_t$ fits the underlying image, given a prediction for the state. In other words, we seek $p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right)$.

We proceed by specifying the measurement function for the AAM as

$$\begin{aligned} \boldsymbol{z}_t &= \overline{\boldsymbol{g}} + \Phi_g\Phi_{c,g}\boldsymbol{p}_c + \boldsymbol{w}_t && (4.11)\\ &= \mathcal{G}\left(\boldsymbol{p}_c\right) + \boldsymbol{w}_t && (4.12)\end{aligned}$$

where we have used (3.17) and assumed that our measurement is corrupted with normally distributed white noise $\boldsymbol{w}_t$ with zero mean and covariance $\sigma_w^2\mathrm{I}$. In (4.12) $\boldsymbol{z}_t$ only depends on $\boldsymbol{p}_c$ and not $\boldsymbol{p}_p$. However, when the actual measurement is made, $\boldsymbol{p}_p$ will be incorporated.

Since we assume that the measurement noise $\boldsymbol{w}_t$ is normally distributed and (4.12) is a linear function, $\boldsymbol{z}_t$ will be normally distributed. Sufficient statistics are then given by the conditional mean

$$\mathbb{E}\left[\boldsymbol{z}_t|\boldsymbol{x}_t\right] = \mathcal{G}\left(\boldsymbol{p}_c\right), \tag{4.13}$$

and the conditional covariance

$$\mathrm{Cov}\left[\boldsymbol{z}_t|\boldsymbol{x}_t\right] = \sigma_w^2 \mathrm{I}\,. \tag{4.14}$$

Thus

$$\boldsymbol{z}_t|\boldsymbol{x}_t \sim \mathcal{N}\left(\mathcal{G}\left(\boldsymbol{p}_c\right), \sigma_w^2 \mathrm{I}\right). \tag{4.15}$$

Now we make the actual measurement, $\widetilde{\boldsymbol{z}}_t$. First we generate a shape using (3.16) and $\boldsymbol{p}_c$. Subsequently, we transform the shape to the correct coordinates in the image plane with the pose parameters $\boldsymbol{p}_p$. Now we warp it to the mean shape $\bar{\boldsymbol{s}}$ and sample the texture. More compactly

$$\widetilde{\boldsymbol{z}}_t = \mathcal{I}\left(\mathcal{W}\left(\boldsymbol{j};\boldsymbol{p}\right)\right) \tag{4.16}$$

using the notation defined in (3.9). Here $\boldsymbol{p} = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_c^{\mathrm{T}}\right]^{\mathrm{T}}$, similar as defined previously in Chapter 3.

Substituting the real measurement (4.16) into (4.15), we have that

$$\begin{aligned}\widetilde{\boldsymbol{z}}_t|\boldsymbol{x}_t &\sim \exp\left(-\frac{1}{2\sigma_w^2}\left(\widetilde{\boldsymbol{z}}_t - \mathcal{G}\left(\boldsymbol{p}_c\right)\right)^{\mathrm{T}}\left(\widetilde{\boldsymbol{z}}_t - \mathcal{G}\left(\boldsymbol{p}_c\right)\right)\right)\\ &= \exp\left(-\frac{\mathrm{E}^2}{2\sigma_w^2}\right).\end{aligned} \tag{4.17}$$

Notice that E^2 is the residual error as defined in Equation (3.24), page 31.

The residual error E^2 is calculated using all the pixels in the textures $\widetilde{\boldsymbol{z}}_t$ and $\mathcal{G}\left(\boldsymbol{p}_c\right)$. If occlusion occurs, the occluded pixels increase the value of the residual error, since they correspond to a large difference between the generated texture $\mathcal{G}\left(\boldsymbol{p}_c\right)$ and the measured texture $\widetilde{\boldsymbol{z}}_t$. We can regard these occluded pixels as outliers. The residual error is therefore not robust when outliers occur. The likelihood of the outlier pixels will be zero. This results in a PF where many particles have a weight of zero and the effective sampling size ((2.43), page 17) is thus smaller.

We solve this problem heuristically by replacing the residual error E^2 with the robust Lorenzian norm

$$\mathrm{E}_{\mathrm{Lor}}^2 = \log\left(1 + \frac{\mathrm{E}^2}{2\sigma_s^2}\right) \tag{4.18}$$

where σ_s^2 is the scale parameter that discards outliers. To investigate the effect of this robust norm, let $\sigma_s^2 = \sigma_w^2$ and replace $\frac{E^2}{2\sigma_w^2}$ in (4.17) with E_{Lor}^2 yielding

$$z_t | \boldsymbol{x}_t \sim \frac{1}{1 + \frac{E^2}{2\sigma_w^2}} \, . \tag{4.19}$$

This distribution is compared with a Gaussian in Figure 4.7. It clearly has a heavier tail resulting in a large likelihood for particles with a larger error. For small values of E, we have that $\log\left(1 + \frac{E^2}{2\sigma_w^2}\right) \approx \frac{E^2}{2\sigma_w^2}$, our original distribution.

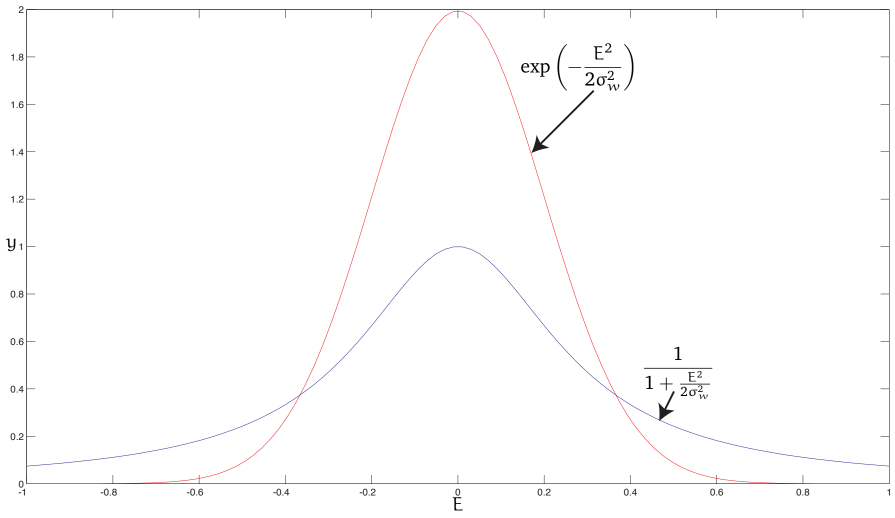


Figure 4.7: *Effect of Lorenzian norm on likelihood*

Other robust norms may be used; for a discussion of different robust norms and AAM fitting, see (Theobald et al., 2006).

The reader might have noticed that an estimate of the current state is involved when performing the actual measurement. In particular, in (4.16), the warping function $\mathcal{W}$ depends on $\boldsymbol{p}$. Here $\boldsymbol{p}$ is initialised with an estimate of the current state. We therefore violate the assumption that the measurement is independent of the state. This is a common problem with particle filter-based trackers in computer vision. For example, the colour-based histogram tracker uses the $x - y$ position of the current state to sample pixels for the histogram measurement. One way to overcome these problems would be a separate object detector to obtain an independent measurement.

The proposal density

PFs sample from a proposal density $q\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}, \boldsymbol{z}_t\right)$ instead of the posterior density. We choose the transitional density $p\left(\boldsymbol{x}_t^{(i)}|\boldsymbol{x}_{t-1}^{(i)}\right)$ as the proposal density (as described in Section 2.1.3.2 on page 14).

We assume a dependent AAM in this section. However, it is straightforward to modify the equations when an independent model is assumed. We conclude that the above discussion is valid for dependent and independent AAMs.

Results and examples

Figure 4.8 illustrates the output when the facial features are used to track a person through an occlusion. The AAM was trained with 6 images of the person being tracked. After losing its target, the deterministic tracker is not able to relocate (see Figure 4.5). However, the stochastic approach is able to track the sequence successfully. We used N = 60 particles.

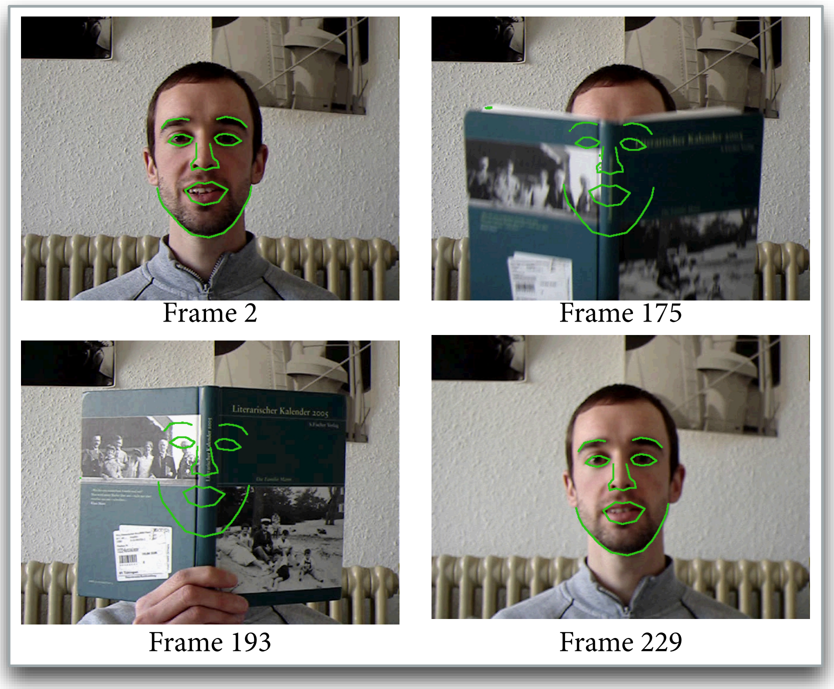


Figure 4.8: *Tracking the facial features of a person using AAMs and PF.*

In Figure 4.9 a hand that moves fast against a cluttered background is tracked. Compare these results to Figure 4.6—clearly the PFAAM tracker can handle fast movements better than its deterministic counterpart.

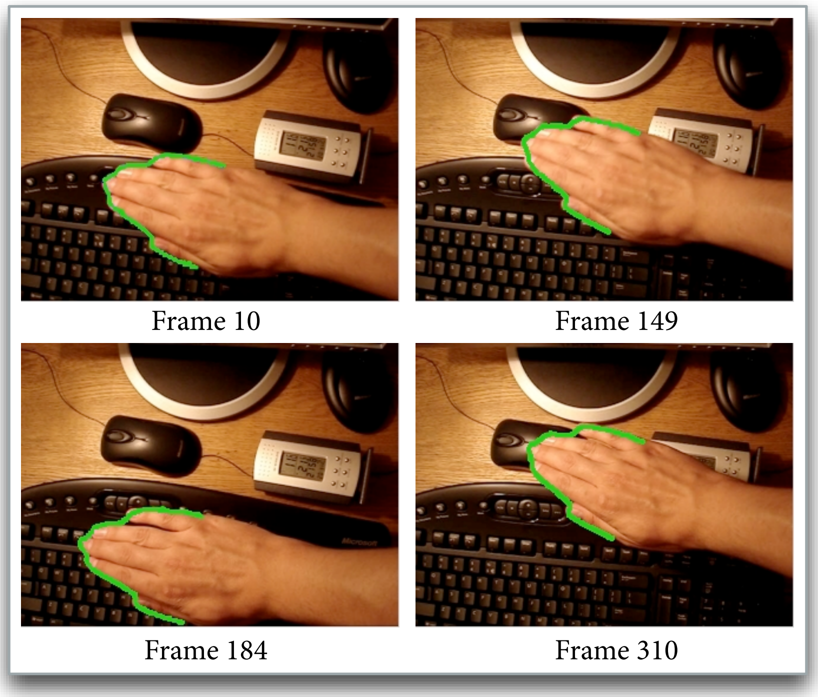


Figure 4.9: *Random selection of result frames to indicate the performance of the AAM-based particle filter tracker.*

4.5.2 Detection of occlusion

If occlusion is present, the AAM update process model (4.9) does not apply. Instead, only the second-order auto-regressive process (4.10) is used. Therefore, occlusion must be detected.

In Section 4.5.1, a robust error function has been used in the measurement model. Within the context of the measurement model, this robust error function compensates for outlier pixels in the measured texture. We proceed with a similar strategy using robust error functions. Now the aim is to detect occlusion and if present, switch to a different process model based.

The occlusion detection is based on the residual error (3.24), page 31. For a given particle, using its state $\boldsymbol{x}_t = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_c^{\mathrm{T}}\right]^{\mathrm{T}}$, a shape is calculated using (3.6). Subsequently, a texture $\boldsymbol{g}^*$ is sampled within this shape. The orthogonal projection $\boldsymbol{g}^{\perp}$ of $\boldsymbol{g}^*$ onto the column space of Φ_s is computed. Now

$$\rho_c = \sum_{j=1}^{m} \mathrm{H}\left[\|\boldsymbol{g}^{\perp}(j) - \boldsymbol{g}^{*}(j)\|_2 - 2\left(\sigma_x^j\right)^2\right], \tag{4.20}$$

where H is the Heaviside function, σ_x^j is the scale, and m is the length of the texture (as explained in Section 3.2, page 25). The Heaviside function is defined as

$$\mathrm{H}[n] = \begin{cases} 0, & n < 0 \\ 1, & n \geq 0 \end{cases},$$

where $n \in \mathbb{R}$. In (4.20), we regard a pixel as occluded if the difference between the projected and the sampled texture is greater than $2\left(\sigma_x^j\right)^2$. In our implementation, we obtain the value of $\|\boldsymbol{g}^{\perp}(j) - \boldsymbol{g}^{*}(j)\|_2$ in several unoccluded images and calculate the standard deviation of this error to get an estimate of σ_x^j.

If

$$\rho_c > 0.8m, \tag{4.21}$$

then the texture for the particle is regarded as occluded. Moreover, occlusion is declared if the number of occluded particles exceeds a certain, pre-defined threshold.

For more information on robust error functions and occlusion detection, the interested reader is referred to (Theobald et al., 2006).

4.5.3 AAM and PFs with a smaller state space

It is possible that the state vector merely consists of the AAM's pose $\boldsymbol{p}_p$ and shape parameters $\boldsymbol{p}_s$ at time t, *i.e.*,

$$\boldsymbol{x}_t = \left[\boldsymbol{p}_p^{\mathrm{T}}, \boldsymbol{p}_s^{\mathrm{T}}\right]^{\mathrm{T}}. \tag{4.22}$$

Note that we do not include the texture parameters $\boldsymbol{p}_g$ in the state vector and that we are assuming an independent AAM. This choice is particularly appropriate for iterative non-linear fitting methods, since fewer non-linear functions are learnt and a speed improvement is obtained.

When the state vector (4.22) is used, it is not possible to calculate the likelihood (4.17) as the texture parameters $\boldsymbol{p}_g$ are not available to make a prediction. In these cases, we suggest a likelihood based on the back-projection error of a texture sampled inside a shape synthesised by the parameters contained in a particle. More precisely, the measurement likelihood is

$$p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right) \propto \exp\left(-\frac{\mathrm{E}^2}{\sigma_w^2}\right). \tag{4.23}$$

It is estimated as follows:

- Calculate a shape using (3.6).
- Sample a texture $\boldsymbol{g}^*$ inside this shape.
- Calculate the projection error E^2 when $\boldsymbol{g}^*$ is projected on the column space of Φ_g.

An alternative, proposed by Hamlaoui and Davoine (2005), is to sample the texture in the first frame, denoted by $\boldsymbol{g}_{\text{fly}}$. Thereafter, the texture is updated with a sampled texture, $\boldsymbol{g}_{\text{sampled}}$, acquired with the PFs estimate for the current state,

$$\boldsymbol{g}_{\text{fly}} = \alpha\boldsymbol{g}_{\text{fly}} + (1-\alpha)\,\boldsymbol{g}_{\text{sampled}} \tag{4.24}$$

where α is a weighting (forgetting) factor. Then the likelihood is

$$p\left(\boldsymbol{z}_t|\boldsymbol{x}_t\right) \propto \exp\left(-\frac{||\boldsymbol{g}^* - \boldsymbol{g}_{\text{fly}}||_2^2}{\sigma_w^2}\right). \tag{4.25}$$

Results and examples

The tracking results, when compared qualitatively, are similar to the results discussed in Section 4.5.1, *i.e.*, the tracker is able to track the sequences successfully. In the next chapter, we show quantitative results when the non-linear fitting method is compared to other trackers.

4.5.4 Local optimisation

We now present how the top-down PF approach can be combined with a bottom-up local optimisation for increased robustness and precision within tracking. This works by embedding the local optimisation loop as an inner loop within the PFAAM main loop described in Section 4.5.1. The intuition behind this is illustrated in Figure 4.10: a set of particles is used to approximate a pdf. Using the PF, these particles are propagated over time. However, a PF will not in general find the optimum (the state corresponding to the global peak within the pdf), as no correction is made after a prediction. This can also be observed in the PF algorithm on page 16. Only the weights are corrected using the measurement. The only correction made is throwing away particles with a low likelihood. Our proposed algorithm works as follows (on a per sample basis): each sample is first predicted according to the motion model and then measured as in the top-down PF and AAM combination described above. Then, starting from this new position in state space, the local Gauss-Newton optimisation of the standard AAMs is started, working on the same camera image and using the same measurement function (4.17). Thus, each sample moves to an adjacent local optimum. If the representation of the pdf $p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t})$ by samples is good enough, the global optimum should be among them. The particles must be reweighed after the local optimisation step because their position and thus their likelihood in the state space changed.

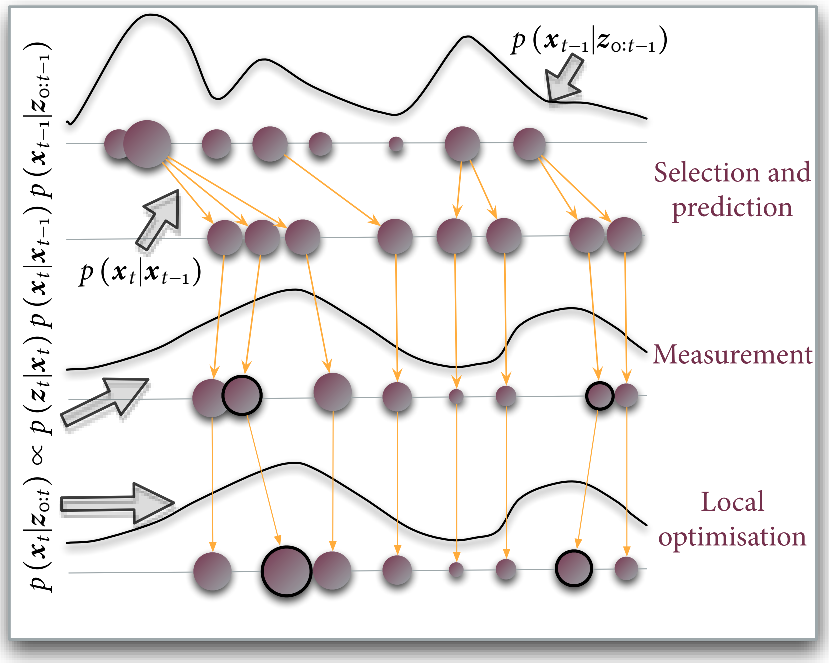

$p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})$
Selection and prediction
$p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$
Measurement
Local optimisation
$p(\boldsymbol{x}_t|\boldsymbol{z}_{0:t}) \propto p(\boldsymbol{z}_t|\boldsymbol{x}_t)\, p(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})\, p(\boldsymbol{x}_{t-1}|\boldsymbol{z}_{0:t-1})$

Figure 4.10: *The PFAAM algorithm. This shows how the local optimisation of AAMs is embedded consistently within the particle filter loop.*

Algorithm 3 details the local optimisation step that is carried out after the measurement step of the standard particle filter.

Local optimisation can be applied to all the particles representing a pdf. We propose to restrict the optimisation to the most promising particles. The most promising particles are those with the largest associated weights. The idea is to use the given computational resources where we expect the best optima exist. When implementing this, the weights of the particles must be stored in

Algorithm 3: *Local optimisation of particles with highest weights*

```
1 I = Indices of particles with largest weights;
2 foreach Index $i \in$ I do
3     $\widehat{\boldsymbol{x}}_t^{(i)}$ = AAM Optimise( $\boldsymbol{x}_t^{(i)}$);
4     if $p\left(\boldsymbol{z}_t|\widehat{\boldsymbol{x}}_t^{(i)}\right) < p\left(\boldsymbol{z}_t|\boldsymbol{x}_t^{(i)}\right)$ then
5         $\boldsymbol{x}_t^{(i)} = \widehat{\boldsymbol{x}}_t^{(i)}$;
6         Recalculate $w_t^{(i)}$;
7     end
8 end
9 Normalise weights;
```

order of increasing values. Alternatively—this is the approach we use— a sorting algorithm is applied on the PF's weights to obtain the most promising particles.

Local optimisation in general provides better fitting of the object to be tracked to the underlying image, but under occlusion, the particles will degenerate severely. Consequently the update of the particles with the largest weights does not take place under occlusion.

Results and examples

The implemented tracker is used to track a hand performing fast movements. We refer to the PFAAM algorithm without local optimisation as the PFAAM_light. A selection of the tracking results is shown in Figure 4.11. We use the Lorentzian norm (4.18) as a measure of the AAM fit to the underlying image. The norm for the model output by tracker is indicated for each frame. All the trackers are able to track the hand correctly when it moves slowly (frames 150 and 309), but in cases of fast movements, the deterministic tracker fails.

To further demonstrate the influence of local optimisation, the Lorentzian norm versus the frame number is plotted in Figure 4.12. The reader is reminded that a smaller Lorentzian norm implies a better fit of the estimate to the underlying image. Together with the results illustrated in Figure 4.11 we notice that the deterministic AAM tracker provides the best results in cases of slow movements (frames 0 – 210). On the contrary, the PFAAM without local minimisation, is not precise, but robust. The PFAAM offers the best of both worlds: a small Lorentzian norm, but also robustness. Furthermore, we see that more particles lead to a better fit, but the PFAAM offers better precision with fewer particles.

Figure 4.13 illustrates the results when tracking in the presence of occlusion. The AAM is trained with 9 images of the person being tracked. We used $N = 150$ particles in the particle filter. In the figure, the first image for a particular tracker is obtained before occlusion; the image

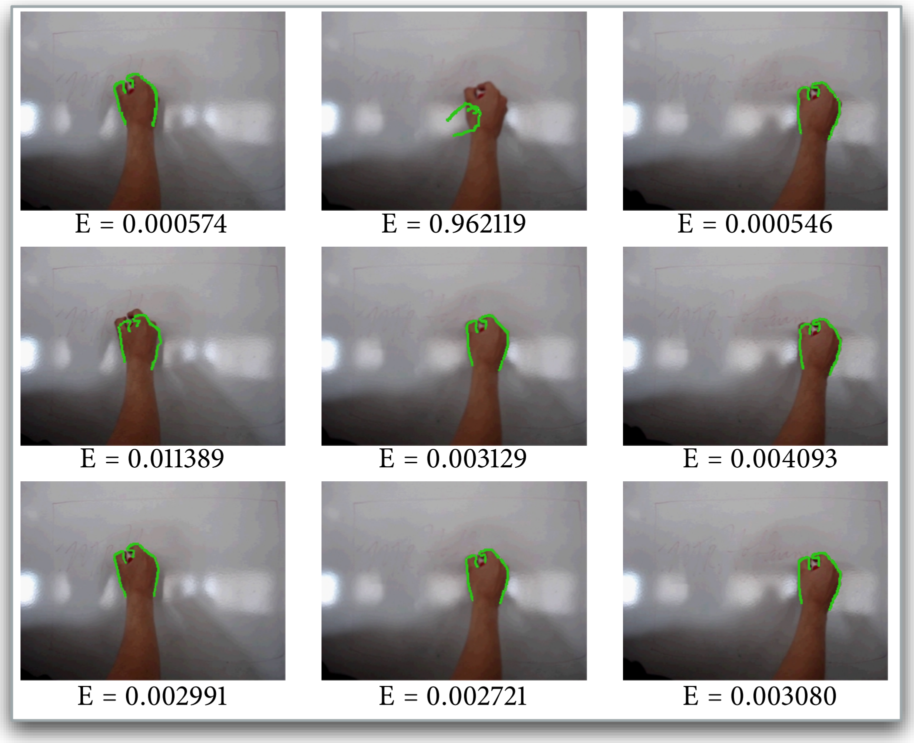
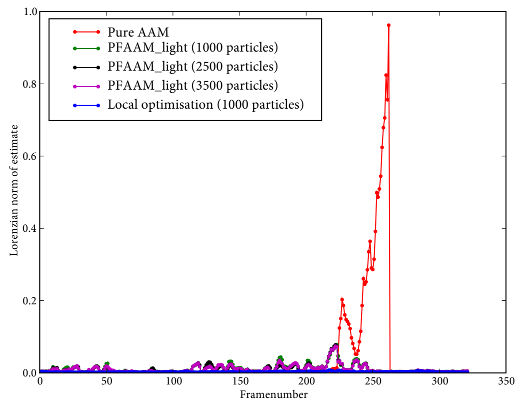


Figure 4.11: *Tracking a fast hand movement, frames 150 (left column), 262 (middle column), and 309 (right column). Top row: Deterministic tracking, Middle row: Tracking in the PFAAM_light, Bottom row: PFAAM tracking with local optimisation.*

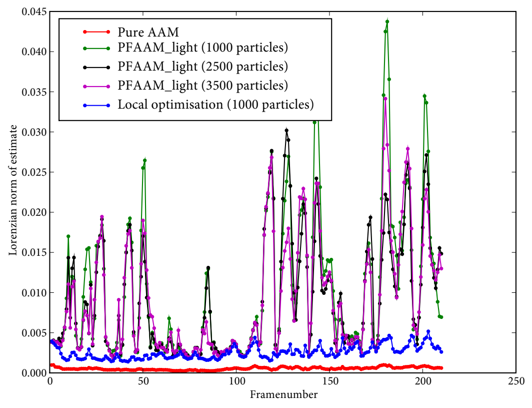
thereafter is a few frames after occlusion. The deterministic tracker is not able to deal with occlusions and once it loses its target, it cannot acquire it again. The PFAAM_light tracker can deal with occlusions, but it takes longer before it finds the target after occlusion, whereas the PFAAM tracker offers the same robustness as the PFAAM_light tracker, but after occlusion, it re-acquires the target fairly easily.

4.5.5 AAMs and active contours

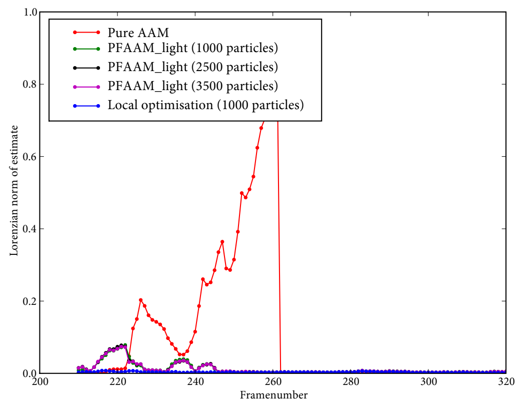
The combined active contour and active appearance model (AC-AAM) proposed by Sung and Kim (2006) finds a shape to initialise the AAM algorithm using active contours. This method thus provides an alternative to initialise an AAM. Given a good initialisation to the AAM, this leads to a tracker with improved tracking robustness. We discuss active contours, followed by the combination of active contours and active appearance models.



(a) Lorentzian norm for complete sequence



(b) Lorentzian norm for frames 0 – 210



(c) Lorentzian norm for frames 210 until the end of the sequence

Figure 4.12: *The effects of local optimisation*

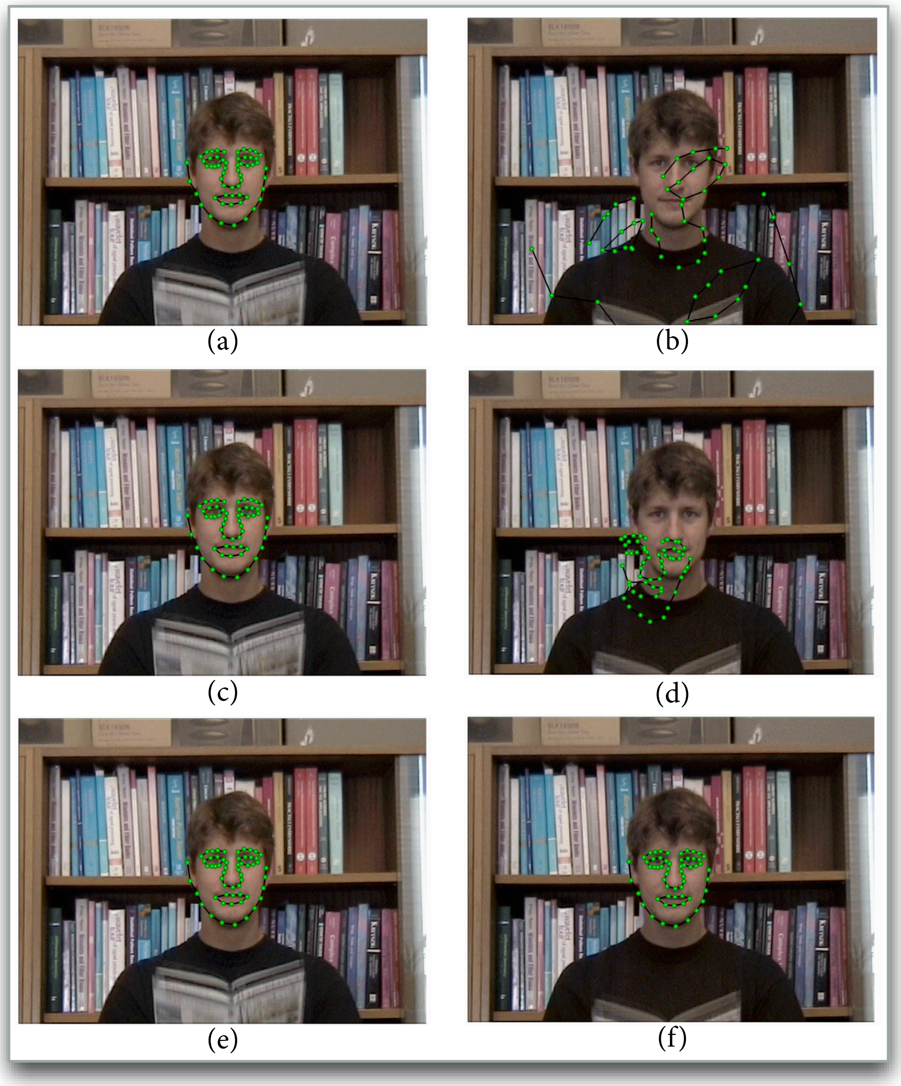


Figure 4.13: *Tracking facial features under occlusion. (a) – (b): Deterministic tracking, (c) – (d): Tracking in the PFAAM_light, (e) – (f): PFAAM tracking with local optimisation. Left column is before occlusion; Right column is after occlusion.*

4.5.5.1 Active contours

Following Blake and Isard (1998), we summarise the contour based particle filter. Adapting a particle filter for a particular implementation, requires the specification of the state vector, process model and measurement model.

The state vector

The state vector at each time step t is given by the shape space vector. Thus $\boldsymbol{x}_t = \boldsymbol{p}_s$. This allow us to generate a vector of B-spline control points $\mathbf{Q}$ for each particle using (3.8).

The process model

States evolve according to a simple random walk given by

$$\boldsymbol{x}_t = \boldsymbol{x}_{t-1} + \mathrm{S}_t \boldsymbol{v}_{t-1} \tag{4.26}$$

where S_t is the process noise covariance and $\boldsymbol{v}_{t-1}$ is a vector of normal distributed random variables. This process model assumes small changes between frames. One can use more sophisticated process models and the reader is referred to (Blake and Isard, 1998) for a detailed discussion.

The measurement model

The binary edge map for the current frame is given to the algorithm to estimate the weight associated with each particle. An example of such an edge map, with a contour, its control points and normal lines superimposed on the edge map, are illustrated in Figure 4.14. To calculate the weight for the ith particle, the following procedure is followed:

- Using (3.8) and the state vector, calculate the vector of control points $\mathbf{Q}^{(i)}$.
- Calculate the normal lines for each control point.
- For each control point, search along the normal line until an edge is found. Let the distance from the control points to the edge be d_j, $j = 1, \ldots, n$ where n is the number of control points. If an edge is not found, set the distance equal to the length of the normal line. Calculate the total length $d_t^{(i)} = \sum_{j=1}^{n} d_j$.
- The weight for the ith particle is then given by

$$w_t^{(i)} = \exp\left(-\frac{\left(d_t^{(i)}\right)^2}{\sigma_w^2} \right). \tag{4.27}$$

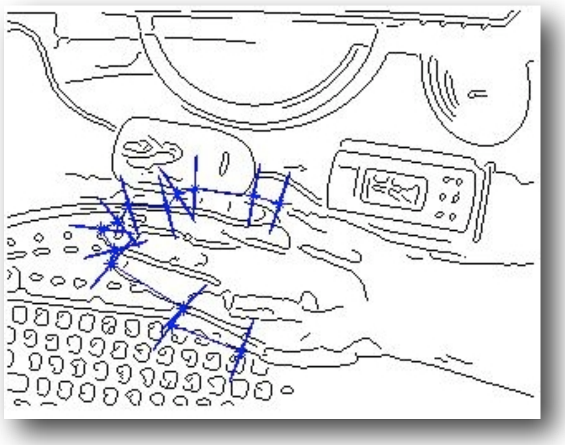


Figure 4.14: *An example output from the Canny edge detector with a contour and its normal lines drawn on it.*

The weights are normalised afterwards to sum to unity. From equation (4.27), we see that a small value of $d_t^{(i)}$ will result in a larger value of $w_t^{(i)}$ and *vice versa*. Furthermore, the variance σ_w^2 determines how much preference we give to particles with a lower distance, $d_t^{(i)}$.

Every time an edge is not found for a particular particle, it is recorded and denoted by $n_d^{(i)}$. If $\sum_{i=1}^{N} n_d^{(i)}$ is larger than a certain pre-set threshold, occlusion is declared. This method for the detection of occlusion provides adequate results; for more sophisticated techniques see *e.g.* (MacCormick, 2002).

4.5.5.2 Initialisation of AAMs with ACs

The AC-AAM tracker consists of two parts. At time step t, the first part performs standard AC tracking with the estimate from the tracker denoted by $\widehat{\boldsymbol{x}}_t$. In the second part, (3.8) is used together with $\widehat{\boldsymbol{x}}_t$ to generate a shape estimate

$$\widehat{\mathbf{Q}}_t = \bar{\boldsymbol{s}} + \Phi_s \widehat{\boldsymbol{x}}_t \, . \tag{4.28}$$

Note that $\widehat{\mathbf{Q}}_t$ and the AAM shape representation $\mathcal{S}(\widehat{\boldsymbol{x}}_t)$ (not normalised with respect to the pose) are equivalent. This shape $\widehat{\mathbf{Q}}_t$ is then used to initialise standard AAM optimisation and the result is output as the best fitted AAM. The result of the AAM is then used to initialise the AC tracker again.

In the presence of occlusion the AAM optimisation fails. Therefore, when occlusion is detected by the AC part, the tracker will output no estimate for the resulting AAM, *i.e.*, the AAM is "switched off".

This technique uses the particle filter indirectly. The particle filter is an integral part of the active contour tracker, but the AAM part of the AC-AAM tracker does not utilise the particle filter.

Results and examples

In Figure 4.15 the results obtained with the AC-AAM tracker are shown, while the corresponding results obtained with the AAM-based particle filter are illustrated in Figure 4.16. Both trackers are able to track the movement of the object accurately.

When the AC-AAM approach is used, a simple dynamic model for the active contour tracker suffices. This can be seen in Figure 4.15 where the output of the active contour tracker is not as accurate, but the resulting AAM fit is. This illustrates the principal idea of the AC-AAM approach: the robustness of the active contour tracker is used to initialise the AAM which in turn, adjusts well to the underlying image. The AC-AAM tracker detects occlusion as can be seen in Figure 4.15(c) and (d), and the AAM optimisation is disabled for these frames.

The AAM-based particle filter successfully tracks the head and shoulders in the presence of occlusion as illustrated in Figure 4.16(c) and (d). Note that this tracker, contrary to the AC-AAM tracker, provides an estimate for all frames (including those with occlusion) and the estimate is consistent with our intuition.

4.6 Discussion

This chapter examines visual tracking of objects, with emphasis on tracking with AAMs and PFs.

We showed that a deterministic tracker AAM tracker fails when the tracked object moves fast or is occluded. This is explained by the inability of an AAM to find optimal parameters if it is initialised poorly.

We therefore introduced several variations of stochastic tracking with AAMs. They are the

PFAAM. This tracker represents the general combination of tracking with an AAM and a PF. It is capable of tracking an object in cluttered backgrounds that moves fast or is occluded.

PFAAM where the state space is smaller. This is similar to the PFAAM, but provides the option for tracking with non-linear fitted AAMs or AAMs fitted with the inverse compositional algorithm. Using a non-linear fitted AAM, we shall show in the next chapter, improves the accuracy of the tracker. Also, the measurement in this variation is calculated differently.

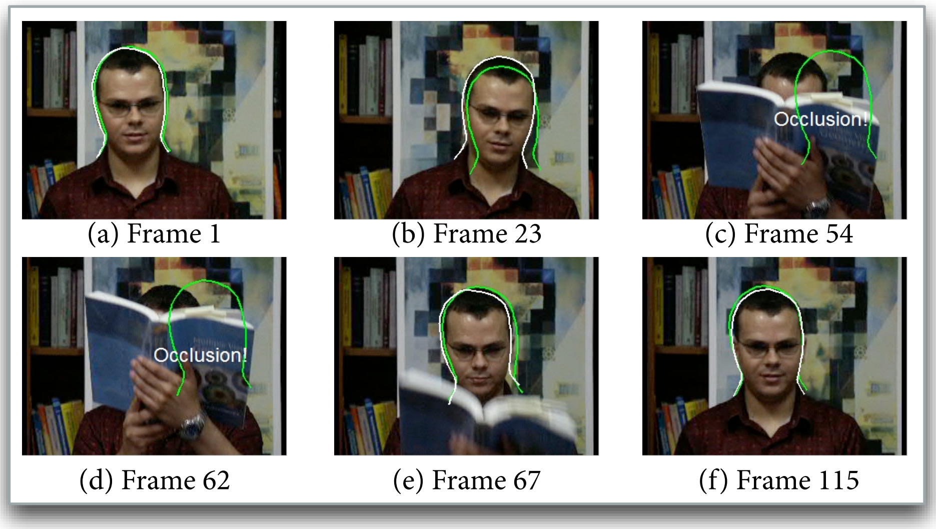


Figure 4.15: *Selection of result frames to indicate the performance of the AC-AAM tracker. The green and white shapes are the output of active contours and AC-AAM respectively. Note that the AC-AAM provides no AAM when the object undergoes occlusion in (c) and (d).*

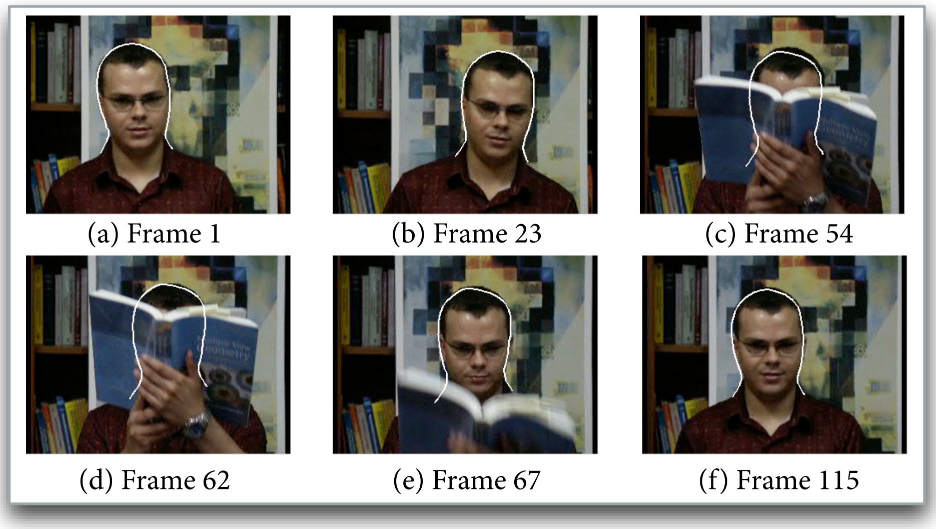


Figure 4.16: *Selection of result frames to indicate the performance of the AAM-based particle filter tracker. Note that the tracker still outputs an estimate in (c) and (d) when the object undergoes occlusion.*

PFAAM with local optimisation. By using local optimisation in tandem with the standard PFAAM, the accuracy is increased and fewer particles are needed (improving running time).

We recommend therefore to use a PFAAM with local optimisation where the state space is smaller.

As mentioned in Section 4.5, similar ideas were also proposed by Hamlaoui and Davoine (2005). Table 4.1 details the differences between this work and theirs. Hamlaoui and Davoine tested their algorithms on facial sequences, while we have employed a more general approach. There is, however, no reason to believe that their tracker will not handle general video sequences. Our process model consists of two models used in a combination. When occlusion occurs, we change our process model. Hamlaoui and Davoine use a single process model and handle occlusions through a robust norm. Occlusion handling through detection allows one to switch to a process model that is designed to handle occlusions. The parameters for our occlusion detection are trained off-line.

We have a running time of 10 – 20 frames per second. Hamlaoui and Davoine reported 2 frames per second. Since they use a variable number of particles, this frame rate is also not a constant. In critical applications this might be a drawback. We use local optimisation as part of the PFAAM to increase accuracy and robustness. This has a omissible effect on the running time.

We described and implemented the algorithms for both dependent and independent AAMs, while Hamlaoui and Davoine focused on dependent AAMs.

A quantitative analysis is not possible. The source code is not available and even if one had implemented the described algorithms, these results would not necessarily correspond to those published by Hamlaoui and Davoine since it is impossible to follow the exact optimisations implemented by the original authors.

Comparing this dissertation's algorithms quantitatively to other silhouette trackers is problematic, since no universal ground-truth exists and since standardised implementations of all the tracking algorithms are not available. We therefore compare the algorithms qualitatively. The qualitative comparison is summarised in Table 4.2; taken from (Yilmaz et al., 2006) with two rows added to reflect Hamlaoui and Davoine (2005) and our work. Note that the PFAAM can, in principle, be extended to a multi-object tracker, since a PF forms the base for the algorithm. It is worth noting that the tracker of MacCormick (2002) is based on active contours. Thus the PFAAM has the added benefit of simultaneously tracking shape and texture.

We also showed in this chapter how ACs and AAMs are combined, resulting in yet another AAM-based tracker. By forming the combination, the accuracy of the AC-tracker is improved.

Stochastic visual tracking with AAMs	**Hamlaoui and Davoine (2005)**
General object tracking	Facial tracking (can be extended to general tracking)
Error norms based on Lorentzian norm	Error norms based on Huber norm
Dynamics: two models in combination	Dynamics: single model
Local optimisation	No local optimisation
Occlusion detection based on trained model. Thereafter the dynamical models are switched.	Occlusions handled through robust norm. The threshold for the robust norm is fixed (not trained).
Speed: about 10 – 20 frames per second	Speed: average of 2 frames per second (variable number of particles means frame rate also varies
Can handle dependent and independent AAMs	Dependent AAMs
Implementation: PC and smart camera	Implementation: PC

Table 4.1: *Comparison between this work and Hamlaoui and Davoine (2005)*

Description	**#**	**Occ.**	**Trn.**	**Features**	**Technique**
Terzopoulos and Szeliski (1993)	S	×	$\surd$	Gradient mag.	Kalman filtering
Isard and Blake (1998a)	S	×	$\surd$	Gradient mag.	Particle filtering
MacCormick (2002)	M	F	$\surd$	Gradient mag.	Particle filtering
Chen et al. (2001)	S	×	$\surd$	Gradient mag.	JPDAF
Hamlaoui and Davoine (2005)	S	F	$\surd$	Texture statistics	Particle filtering
PFAAM	S	F	$\surd$	Texture statistics	Particle filtering

Table 4.2: *Comparison of PFAAM to other silhouette trackers. The table is taken from (Yilmaz et al., 2006) (as well as the description in brackets that follow) with the last row added to reflect our work. (Occ. denotes occlusion handling and Trn. denotes training. #: number of objects, S: single, M: multiple, P: particle, F: full. Symbols $\surd$ and × denote whether the tracker can or cannot handle occlusions, and require or does not require training.)*

Moreover, the robustness of the deterministic AAM tracker is improved. Table 4.3 explains the differences between the PFAAM and the AC-AAM.

AC-AAM	**PFAAM**
Temporal filtering by using PF through AC	Temporal filtering by using PF directly
Use AAM to increase accuracy	Use AAM to increase accuracy
Handle occlusions, but no AAM estimate	Handle occlusions with AAM estimate

Table 4.3: *Comparison between AC-AAM and PF-AAM.*

We recommend that if an AC-tracker is available, one can use it in tandem with an AAM for increased accuracy. Otherwise, one should implement the PFAAM directly.

Chapter

5 Implementation and experimentation

Ideas, like stalactites and stalagmites, form in the dark inner cave of consciousness. They form in drips and drops... We must learn to wait for an idea to hatch. All too often we try to push, pull, outline, and control our ideas instead of letting them grow organically. The creative process is a process of surrender, waiting, not control.

— Julia Cameron

The previous chapter presented tracking algorithms based on AAMs and PFs. At the same time, it presented experimental results where appropriate. This chapter builds on the previous chapter by presenting additional empirical results. It also discusses the software implementation and gives an overview of the hardware configurations. A particular emphasis is placed on the smart camera implementation that demonstrates how the tracking algorithms can run on different hardware platforms. Finally, we show how the 2D-AAM tracker can be extended to a 3D-tracker.

This chapter is organised as follows: Section 5.1 overviews the software implementation. The hardware configurations are discussed in Section 5.2. Section 5.3 presents the tracking results. An implementation of the algorithms for a smart camera is presented in Section 5.4. The 3D-tracker is detailed in Section 5.5.

5.1 Software overview

The software is mainly implemented in C++. An initial version of the software is implemented using the open-source AAM-API (Stegmann et al., 2003). The AAM-API is only available for the

Microsoft Windows platform. Since our goal is a cross-platform implementation, we implemented a second version of the software. In this implementation, computer vision tasks are done with the open-source and cross-platform computer vision libraries, `vxl` (vxl-maintainers, 2009). The AAM implementation is provided by DeMoLib (Saragih, 2009). Thus, in the implementations, the basic AAM algorithms are provided by the AAM-API or DeMoLib. We implemented the PF algorithms and algorithms for the combination of AAMs and the PF. We use the facade design pattern (Gamma et al., 1994) in our interface class with DeMoLib. This enables us to change the basic AAM library easily, since only the facade class needs to reflect another AAM library. The rest of the code is kept unchanged. The resulting system runs on at least Apple OS X 10.5.7, Linux 2.6.27, and Microsoft Windows XP, because `vxl` itself is cross-platform and forms the base of the implementation.

A version has also been developed for the Matrix Vision mvBlueCOUGAR-P smart camera (Matrix Vision, 2008a). Here we used the same implementation as discussed in the previous paragraph. However, we implemented extra routines enabling us to interface with the smart camera. The code is compiled for the smart camera using a GCC cross-compiler for PowerPC.

When optimising the code for speed, bottlenecks in the implementation are identified with the dynamic tracing framework DTrace (Cantrill et al., 2004). The main cause for a deterioration in speed is allocation of memory. This problem is rectified by pre-allocation of data structures. A significant performance gain is obtained with automatic vectorisation of loops as implemented by the Intel C/C++ or the GNU GCC (Naishlos, 2004) compiler.

The active contour implementation is in MATLAB. This implementation builds on the active contour Kalman filter of Jacobs (2005). We provided the particle filter implementation.

The software will not be released publicly in the near future due to commercial possibilities. However, all the code are available in a Subversion repository within the Stellenbosch University Network.[1] Details of the implementation are provided in Appendix C.

5.2 Hardware overview

Two different hardware systems were used to conduct experiments. The first system, used for most of the experiments, is based on a standard personal computer. The second system is a smart camera, *i.e.*, all the processing is performed directly on the camera. A smart camera allows the algorithms to run on the raw, artefact free video data, transmitting only the resulting parameters. The benefits are that the video feed does not leave the camera, reducing the bandwidth and

[1] Access detail can be obtained be contacting the author.

freeing the centralised servers in computer vision applications to concentrate on higher level processing. A comprehensive overview of current smart camera based vision systems can be found in (Dietrich et al., 2007) and (Rinner and Wolf, 2008).

5.2.1 The desktop testing environment

The hardware consists of a 2.66GHz Intel Core 2 Duo processor with 4GB of DDR3 RAM. The operating system is Mac OS X 10.5.7 and the C++-code are compiled with GNU GCC 4.4. Automatic vectorisation of loops is enabled.

5.2.2 The smart camera system

We embed the PFAAM algorithm on the mvBlueCOUGAR-P smart camera series available from Matrix Vision (Matrix Vision, 2008a). These industrial cameras are powered by a 400MHz PowerPC processor with 64MB of SDRAM and run an embedded version of the Linux operating system. Interfacing with the device is done via Gigabit Ethernet. Both CCD and CMOS models are available with resolutions up to 1600×1200 pixels. Their power consumption is below 7W. Results as reported are obtained using the VGA colour version (-P120aC) with $\frac{1}{3}$" CCD progressive scan sensor. An illustration of the hardware can be seen in Figure 5.1.

5.3 Experimentation

This section has multiple goals: it demonstrates the accuracy of the tracking algorithms and gives estimated running times, given different configurations. A number of parameters are user-definable. For example, when training an AAM, the number of perturbed model parameters is specified by the user. An exhaustive report is impossible. We select those parameters that are the most critical for object tracking with an AAM and PF combination. We focus on the following aspects of the PFAAM tracker:

- the number of particles in the PF,
- the process models of the PF,
- the effect of the measurement covariance in the PF,
- local optimisation, and
- the type of AAM used in the PF.

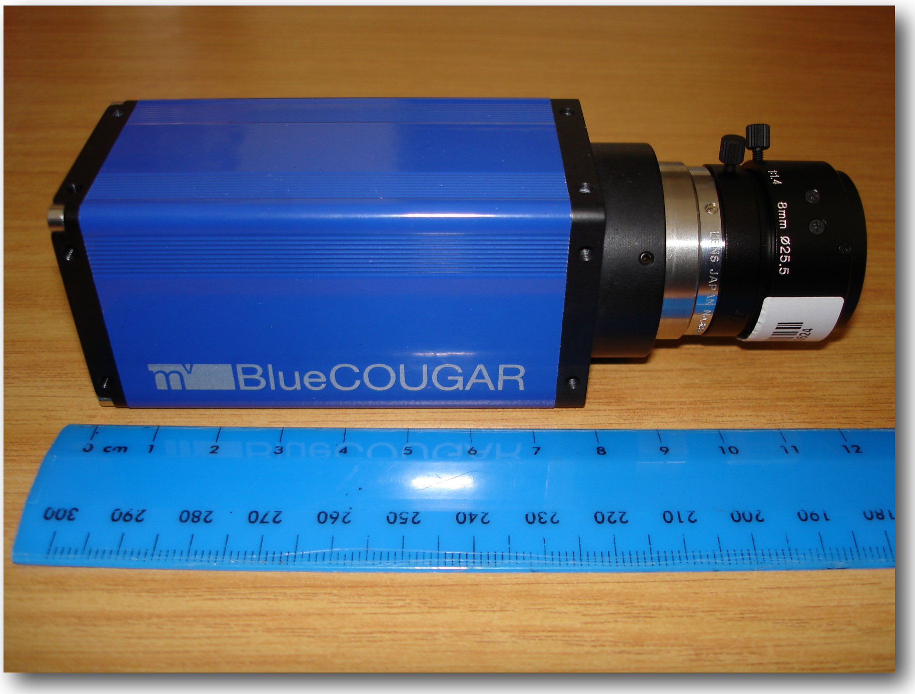


Figure 5.1: *The smart camera system.*

These configurations will be demonstrated on the same video sequence. We carefully collected ground-truth for this sequence. Selected frames from the sequence are given in Figure 5.2. When a deterministic AAM tracker is used to track these sequences, it fails due to erratic movements. The results are shown in Figure 5.3. Clearly we need the AAM and PF combination to perform the tracking.

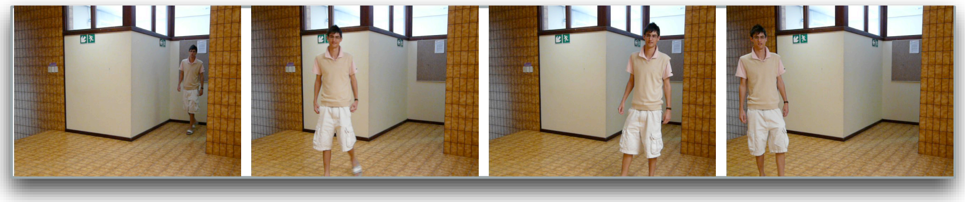


Figure 5.2: *Selected frames from our benchmark video sequence.*

Before presenting the results, it is instructive to discuss the evaluation protocol. The evaluation protocol is built on the evaluation of tracking algorithms of Chapter 4, page 41. However, more detail is necessary to understand the results of this chapter.

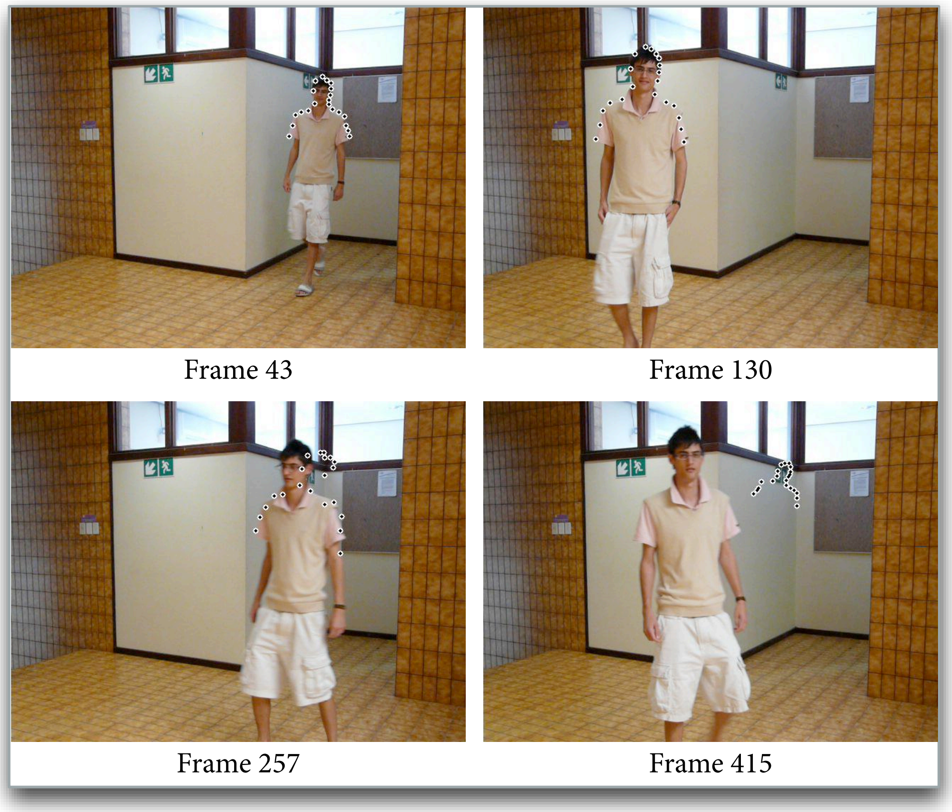


Figure 5.3: *Selected frames when a deterministic AAM tracker is used to track the benchmark video sequence.*

5.3.1 Evaluation protocol

Throughout this chapter, we use quantitative performance measures. We commence by collecting the ground-truth in the video sequences. For each frame, it consists of the shape found by the AAM fitting algorithm. Let us denote it by $\widetilde{\boldsymbol{s}}_t$. Using $\widetilde{\boldsymbol{s}}_t$, given by (3.1) and (3.6), we calculate the pose $\widetilde{\boldsymbol{p}}_p$. Now, let the estimate of a tracking algorithm be $\widehat{\boldsymbol{s}}_t$. Similarly for $\widehat{\boldsymbol{s}}_t$, we calculate the pose parameters $\widehat{\boldsymbol{p}}_p$, where we have dropped the t-dependence of $\widetilde{\boldsymbol{p}}_p$ and $\widehat{\boldsymbol{p}}_p$ to simplify the notation. The relative error between between the ground-truth and the tracked frame is then given by

$$e_t^{\mathrm{GT}} = \frac{\left\|\widetilde{\boldsymbol{p}}_p - \widehat{\boldsymbol{p}}_p\right\|_2}{\left\|\widetilde{\boldsymbol{p}}_p\right\|_2} \,. \tag{5.1}$$

The relative error is based on the pose, since it contains the $x - y$ position of the object, as well as information of the scale and rotation. Therefore, it gives an indication of geometrical interpreted accuracy.

The performance of the tracker is measured as the mean of e_t^{GT}, $t = 1, \ldots \mathrm{T}$, where T is the length of the video sequence.

5.3.2 The number of particles in the PF

The number of particles is varied between 1 and 500. A non-linear iterative AAM is used and the process model is a trained auto-regressive process. Figure 5.4 illustrates the accuracy versus the number of particles. The error drops as the number of particles increases, but having more than 100 particles, the error is more or less constant. This suggests that the pdf is well approximated and that the limit of the model is reached. The remaining error is attributed to the AAM fitting.

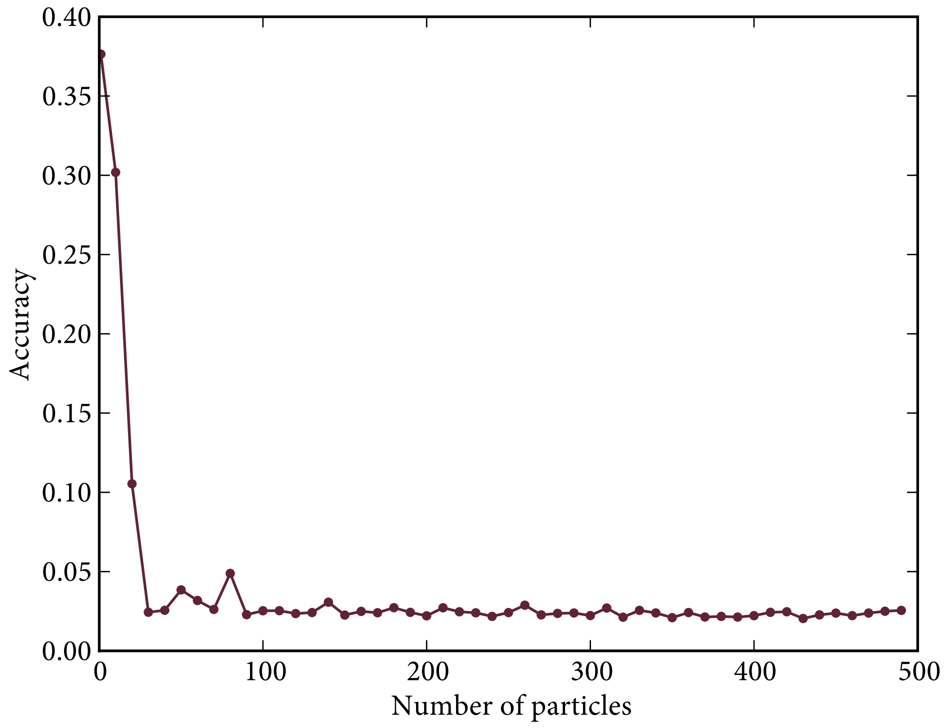


Figure 5.4: *The accuracy of the tracker compared versus the number of particles.*

The running time of the particle filter is linear in the number of particles. Assume that k operations are performed for a particle. Then, for N particles, Nk operations are performed, because the particles do not interact with each other. Furthermore, an additional c operations are done for normalisation of the particles. This normalisation is also linear in the number of particles (it is only a sum over all particles). Consequently, the overall running time is $Nk + Nc$ which yield a linear running time.

5.3.3 The process models of the PF

As discussed in Chapter 4, the PF applies two process models. One process model is a second-order auto-regressive process, given by (4.10) and repeated below for convenience

$$\boldsymbol{x}_t = \overline{\boldsymbol{x}} + A_2 \left(\boldsymbol{x}_{t-2} - \overline{\boldsymbol{x}}\right) + A_1 \left(\boldsymbol{x}_{t-1} - \overline{\boldsymbol{x}}\right) + B_0 \boldsymbol{v}_t \ .$$

The second process model relies on an AAM to calculate an update. It is given by (4.9), again listed below for convenience,

$$\boldsymbol{x}_t = \widehat{\boldsymbol{x}}_{t-1} + \boldsymbol{\omega}_{t-1} + S_{t-1}\boldsymbol{v}_{t-1} \ .$$

In our PF implementation, any number of particles can be propagated through both models. A parameter α_p specifies the number of particles on which the AAM update process are applied. For example, if $\alpha_p = 0.1$, then (4.9) is used to predict the new values of $0.1 \times N$ particles. This is done by adding $0.1 \times N$ particles to the PF and propagating them through the process. Particles are added to the PF to ensure that all the information after prediction by the AR process is taken into account when calculating the weights. At each iteration, resampling is done, but only N particles are selected. If $\alpha_p = 1.0$, however, we use only the auto-regressive process; if $\alpha_p = 0.0$ all the particles are propagated through the AAM update process.

Figure 5.5 illustrates the accuracy for different values of α_p. As expected, the AAM update process is more accurate. However, particles from a process other than the AAM update are needed, because AAM updates cannot be predicted in the presence of occlusions.

Figure 5.6 shows the running time if α_p is varied. An increase in running time is observed for $0 < \alpha_p < 1$, since more particles are effectively in the PF. A possible remedy is to specify a smaller number of initial particles when $0 < \alpha_p < 1$.

5.3.4 The effect of the measurement covariance in the PF

In Section 4.5.1 on page 47, we introduced the measurement covariance as part of the measurement model. Now we shall vary the values of σ_w. A non-linear iterative AAM is used with 100 particles

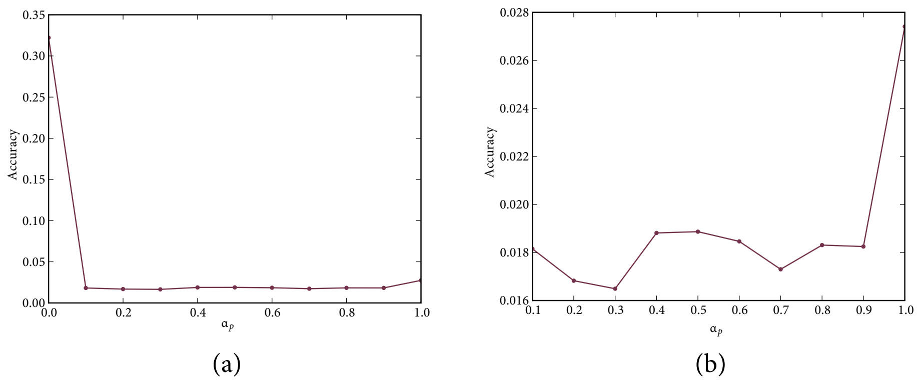


Figure 5.5: *Accuracy of tracker for different process models. In (a), the accuracy is plotted for* $0 < \alpha_p < 1$; *in (b)* $\alpha_p = 0$ *is removed, allowing us to focus on the effect of varying* α_p *when* $\alpha_p > 0$.

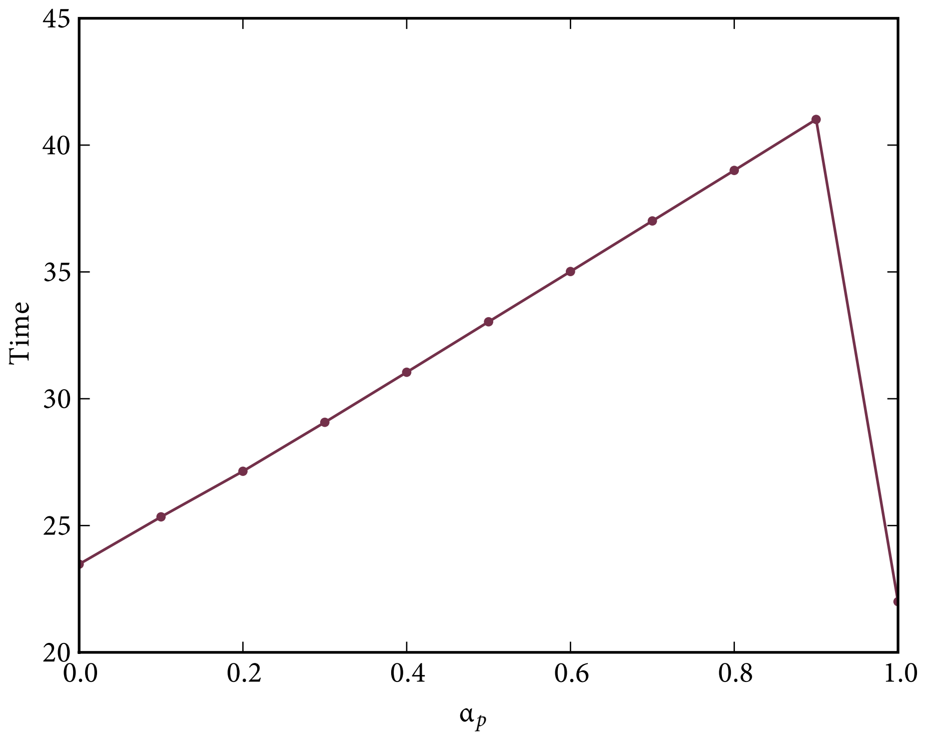


Figure 5.6: *Running time versus* α_p. *If* $\alpha_p = 0$, *the process model is an AAM update process. If* $\alpha_p = 1$, *the process model is a second-order AR process.*

in the PF. A combination of an auto-regressive process and AAM update model ($\alpha_p = 0.1$) forms the process model.

In Figure 5.7, the accuracy is shown as a function of σ_w. Here we can clearly see that small values of σ_w lead to better accuracy. One can immediately draw the conclusion that smaller values of σ_w are preferable. This is, however, not true. In Figure 5.8, the effective sampling size (Equation (2.43) on page 17) is plotted against the measurement covariance. In this graph, one observes that smaller values of σ_w lead to a smaller effective sampling size. Having a small effective sampling size, the particles in the PF are less diverse and therefore the target might be lost during fast movements or occlusions.

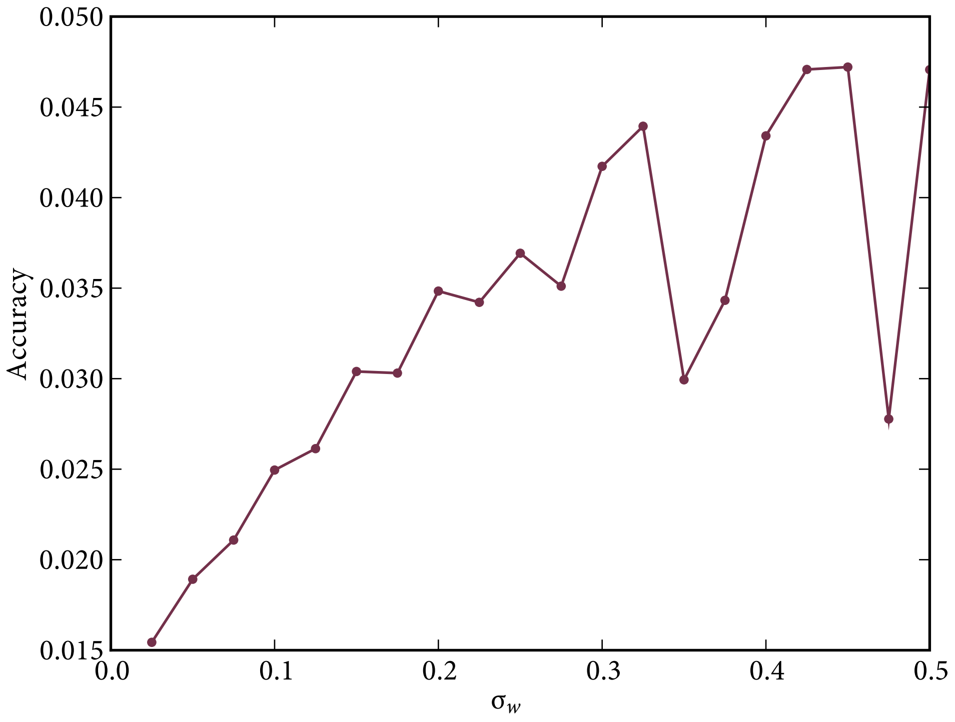


Figure 5.7: *Accuracy versus the measurement covariance, σ_w.*

We observe thus a trade-off between the values of σ_w and the effective sampling size. A value of σ_w should therefore be chosen such that the effective sampling size is at least half the number of particles. If accuracy is of more importance, a smaller value of σ_w suffices.

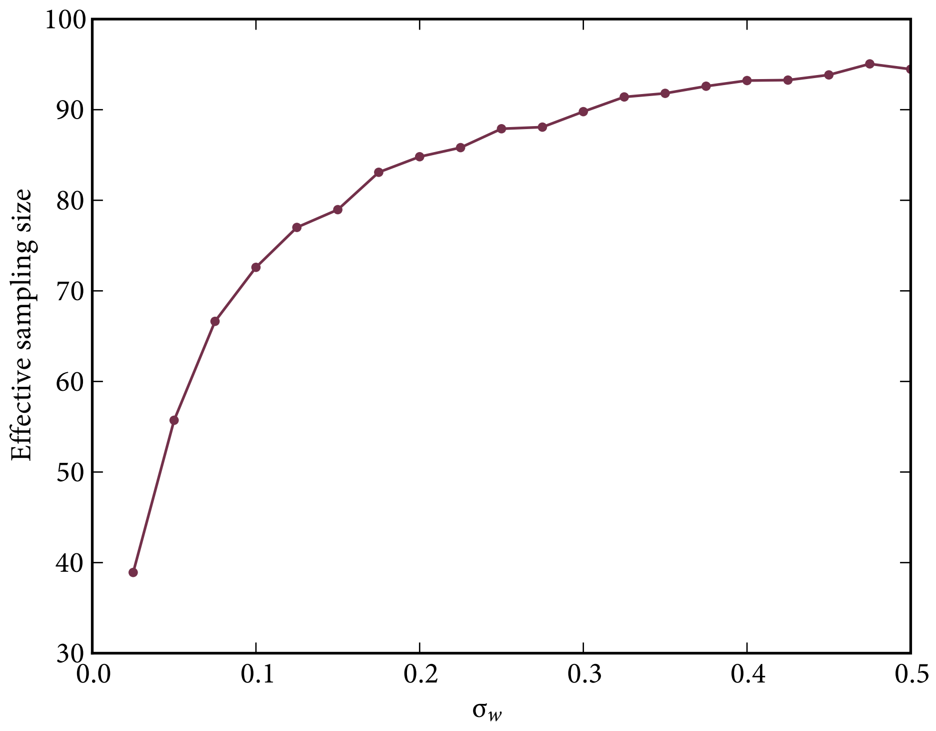


Figure 5.8: *Effective sampling size versus the measurement covariance,* σ_w.

5.3.5 Local optimisation

Previously, in Section 4.5.4 on page 55, we presented detailed analysis of local optimisation based on the Lorentzian norm. Table 5.1 describes the running time and accuracy (based on ground-truth) if local optimisation is applied. We see a reduction in the ground-truth error if local optimisation is used. Notice the time penalty that one pays. However, this is justifiable, since one can use fewer particles to achieve the same accuracy as a particle filter with more particles and no local optimisation.

Description	**Accuracy**	**Running time (in ms)**
No local optimisation (100 particles)	0.012	25.02
Local optimisation (100 particles)	0.0052	44.09
Local optimisation (40 particles)	0.016	21.00

Table 5.1: *Accuracy and running time for different cases of local optimisation*

5.3.6 The type of AAM used in the PF

Table 5.2 reports the accuracy and running times for different AAMs (see Section 3.5, page 31). The PF consists of 100 particles. No local optimisation is used. The results confirm the literature about AAM fitting, but generalises to tracking algorithms employing a PF. Cootes et al. (2001) argued that the fixed Jacobian method performs better in practise than linear regression. Saragih and Goecke (2007) show empirically that the non-linear iterative fitting algorithms are at least as accurate as the fixed Jacobian.

Description	Accuracy	Running time (in ms)
Linear regression	>0.5	23.29
Fixed Jacobian	0.144	20.25
Non-linear iterative fitting	0.017	25.13

Table 5.2: *Accuracy and running time for different AAMs.*

5.4 An implementation on a smart camera

We implemented the tracker for the smart camera in C++. Acquisition of image sequences, as well as control of the camera's settings, was done with the mvIMPACT SDK from Matrix Vision (Matrix Vision, 2008b).

An AAM was trained offline for a person's face. This was done on a training set comprising 6000 perturbations for each of the 32 images of size 320×240. Approximately 380 pixels were sampled inside the shape during an observation. The particle filter uses 50 samples.

In Figure 5.9 (left column) the results are shown of tracking the facial features of a person. The tracker runs at approximately 5 frames per second on the smart camera. On a standard laptop computer (Intel Core 2 Duo 2.16Ghz, 2GB of RAM), we achieve a framerate of 50Hz. We currently do not use a background model to perform pre-processing of the frames. If we have a background model, this would reduce the regions of interest and increase the frame rate.

In Figure 5.9 (right column) the reconstructions are illustrated using the AAM parameters. We sampled about 6000 texture values inside the shape on the tracker. Using these texture values, the appearance parameters are calculated through (3.15) and then transmitted to the server. The dimension of the parameter vector is typically 30. The transmission of only these parameters reduces the bandwidth requirement. However, the reconstructed images are still good enough to interpret.

5.5 3D-head tracking with AAM-based particle filter

Until now, we focused only on 2D-tracking algorithms. This section extends the ideas of the 2D-tracking algorithms, resulting in a 3D-tracker. Classical tracking algorithms track a cloud of points from one frame to the next (Gennery, 1992). The problem is then to find these corresponding points in a stereo pair of images.

We follow a different approach. The outline of an object is tracked in 2D. This outline consists of a set of points. If an AAM is used to describe the outline, corresponding points are trivially found.

Using the correspondences, the 2D-points are mapped to 3D-space. The result is object contours in 3D. In 3D-space we can map these points to a generic object model. By doing so, we can manipulate the 3D-model using detail obtained from the 2D-trackers. If one extends this idea of mapping the 3D-points onto a 3D-model, animation can for example be performed from movements captured by the 2D-trackers. We also get an estimate of the 3D-pose.

The 3D-tracker gives us an indication of the accuracy of the 2D-tracker if we observed how well the corresponding points obey the epipolar constraints.

Using AAMs to track an object in 3D has been investigated before; see for example (Sung and Kim, 2004; Mittrapiyanuruk et al., 2005; Faggian et al., 2006). In these approaches, the AAM is extended to include 3D-information within its shape formulation.

In this section, we assume that the reader is familiar with projective geometry and 3D-reconstruction. Appendix B provides a revision of these topics.

5.5.1 The stereo 2D-trackers

Finding the corresponding points in the pair of 2D images is a problem in typical 3D-reconstruction using stereo geometry. However, this problem is easily solved when an AAM tracker is used. Given a synchronised pair of images, an AAM is fitted in these images. Then shapes $\boldsymbol{s}$ and $\boldsymbol{s}'$ can be obtained from these pair of images. If the same AAM is used, the points in $\boldsymbol{s}$ correspond to the points in $\boldsymbol{s}'$ (see Section 3.1 on page 22). Figure 5.10 shows the points being tracked in the synchronised setup.

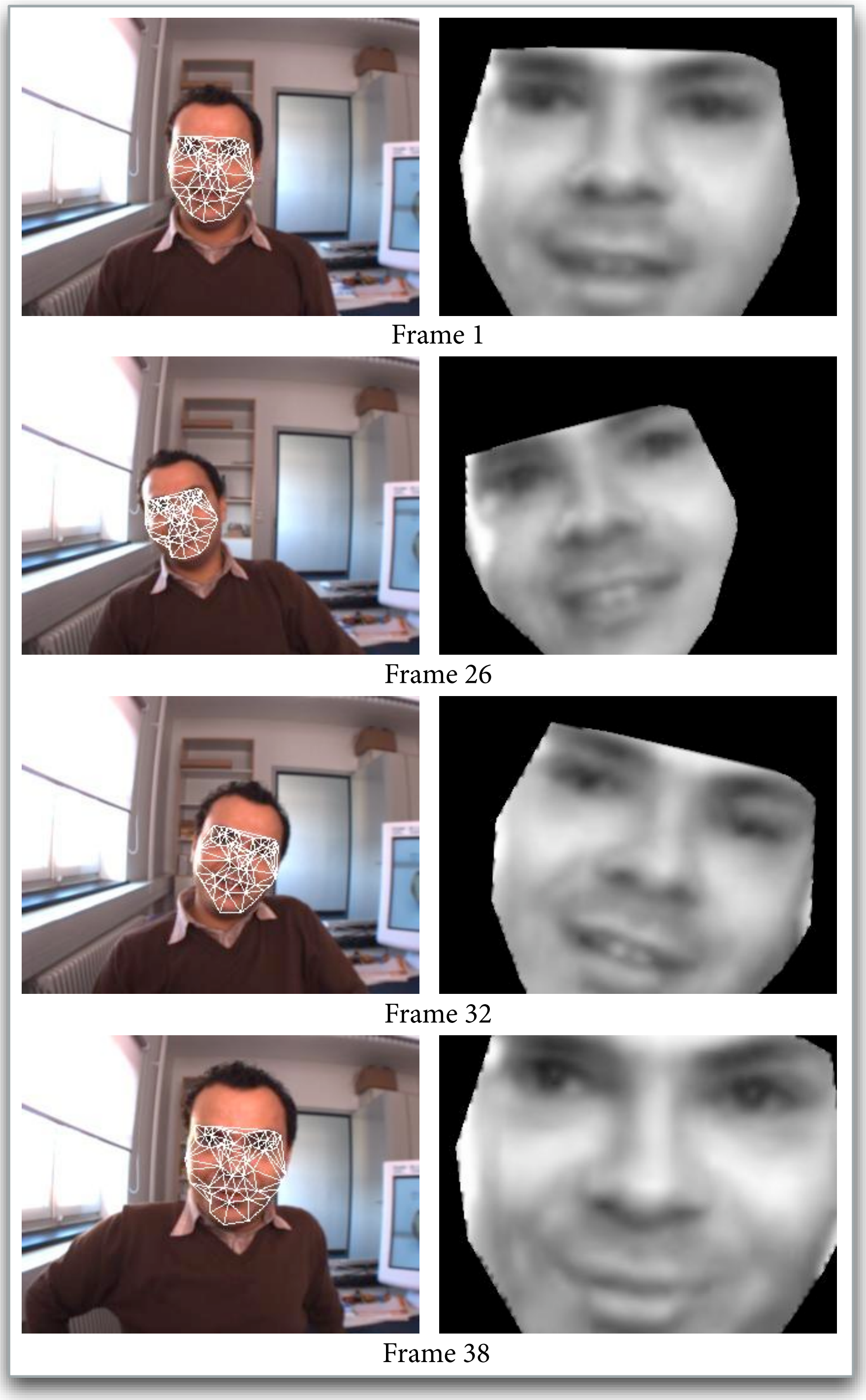


Figure 5.9: *Tracking facial features. Left column is the result of the tracker; Right column is the reconstruction using the AAM's texture parameters.*

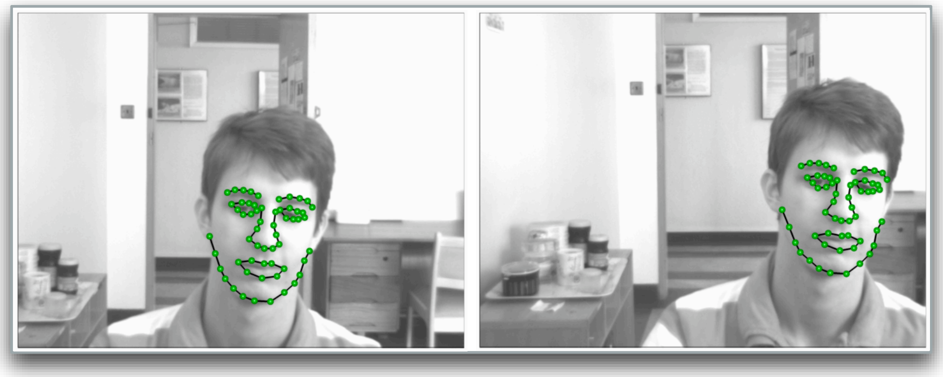


Figure 5.10: *Corresponding points tracked with PFAAM.*

5.5.2 Estimating the 3D head pose

The corresponding points obtained with the PFAAM tracker, are mapped to 3D space using stereo geometry. Thus, for shape $\boldsymbol{s}$, we obtain points $\boldsymbol{x}_i$, $i = 1, \ldots, \frac{N}{2}$, where N is the length of the shape vector $\boldsymbol{s}$. Similarly, we obtain the points $\boldsymbol{x}'_i$, $i = 1, \ldots, \frac{N}{2}$ from $\boldsymbol{s}'$. Notice that $\boldsymbol{x}_i$ and $\boldsymbol{x}'_i$ form a corresponding point set. Using triangulation, these points are mapped into 3D-space. Figure 5.11 shows the result of such a triangulation.

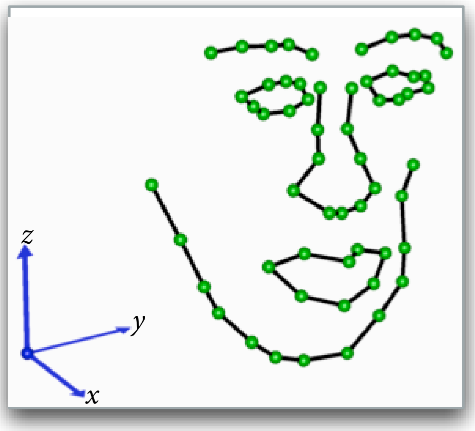


Figure 5.11: *3D points obtained from triangulation.*

Next we mapped these 3D-points onto a generic 3D-face model by manually defining similar features between the reconstructed points and the 3D-model. Figure 5.12 illustrates this generic face model.

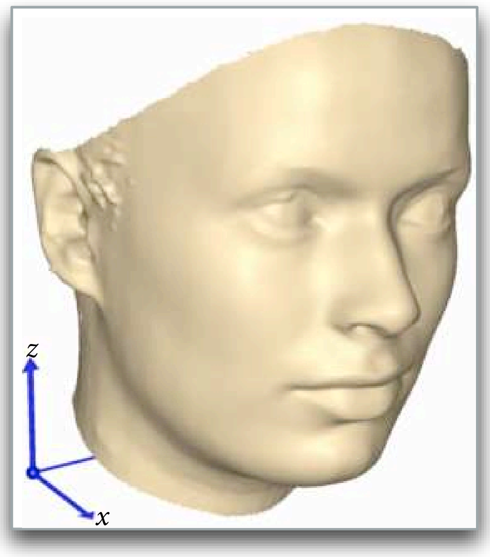


Figure 5.12: *Generic face model in 3D-space.*

5.5.3 Results

The experiments were conducted on a pair of Firefly cameras from Point Grey Research Inc (Point Grey Research Inc., 2009). The cameras are synchronised externally. Frames are captured at a resolution of 640 × 480. Camera calibration is performed offline.

In Figure 5.13 we compare the tracked points and those rectified using Sampson correction. Notice that these lines are nearly straight; an indication that the epipolar constraints are satisfied in the sense that their rectified y-coordinates in the two images are almost equal. This suggests that the PFAAM tracker is able to accurately track images. In Figure 5.14, selected frames are shown with the estimated 3D-pose for each.

5.6 Summary

This chapter presented two implementations of the tracking algorithms that use a combination of the AAM and PF. One implementation is a cross-platform implementation that runs on Mac OS X, Linux and Windows. The second implementation runs on the mvBlueCOUGAR-P smart camera of Matrix-Vision.

Quantitative results were presented for different configurations. In particular, we focused on the number of particles in the PF, the process models of the PF, the effect of the measurement covariance in the PF, local optimisation, and the type of AAM used in the PF. Many times a

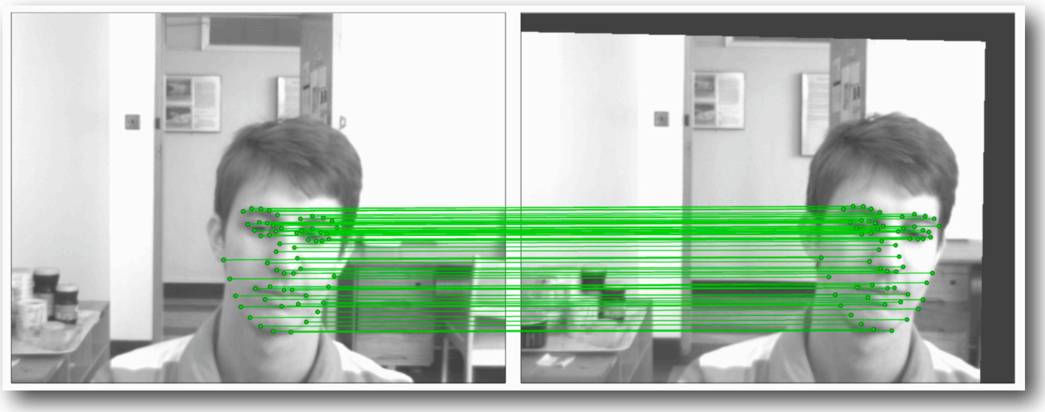


Figure 5.13: *PFAAM tracks accurately in 2D, since the epipolar constraints are satisfied.*

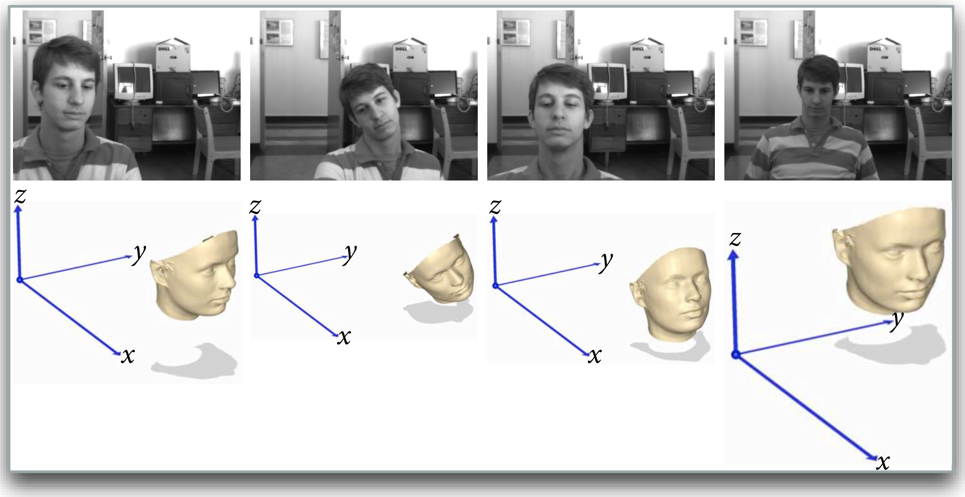


Figure 5.14: *Selected frames from the right camera sequence (top), and the estimated 3D-pose for each (bottom).*

trade-off existed between accuracy and speed or accuracy and robustness.

Using the smart camera implementation, we demonstrated the advantage of transmitting only the AAM parameters in tracking applications where bandwidth is critical. Finally, we presented a 3D-head tracker that use as its building blocks a tracker consisting of an AAM and PF.

Chapter

6 Concluding remarks

And for us this is the end of all the stories, and we can most truly say that they all lived happily ever after. But for them it was only the beginning of the real story. All their life in this world and all their adventures in Narnia had only been the cover and the title page: now at last they were beginning Chapter One of the Great Story which no one on earth has read: which goes on forever: in which every chapter is better than the one before.

— C.S. Lewis, The Chronicles of Narnia

6.1 Reprise

This dissertation's main aim is tracking. It investigated, in particular, the viability of stochastic tracking with active appearance models. In order to be called viable, the tracker needs to be

➤ robust,

➤ accurate, and

➤ fast.

Concerning robustness, a deterministic active appearance model tracker was shown to fail, without recovery if the tracked object moved fast or was occluded. We therefore introduced a combination of active appearance models and particle filters. This tracker tracked correctly where an object moved fast in cluttered backgrounds or in cases of uncomplicated occlusions. We showed that with enough samples in a particle filter, the probability density functions are well approximated. Then the tracker is as accurate as the underlying active appearance model. This implies an accurate tracker. We implemented the algorithms on a standard desktop computer and achieved a real-time running time. Near real-time is achieved on a smart camera. Furthermore, if local optimisation is applied, fewer particles are needed resulting in a faster tracker. On the other hand, non-linear fitted active appearance models boosted the accuracy of the tracker. Thus stochastic tracking with active appearance models is indeed viable.

En route, we presented an extension of the 2D-tracker to 3D. It also confirmed the accuracy of the tracker.

6.2 Limitations

In Figure 6.1 a person is tracked, and as he moves, he passes another person. The dynamic model consists of a combination of a trained auto-regressive process and an AAM update process model. In the image sequence shown below, the PFAAM tracker is able to recover after it has lost its target due to the occlusion that took place. However, when a second occlusion occurs, illustrated in Figure 6.2, the tracker do not recover.

Thus, the question is raised: what went wrong the second time? In the first sequence, the person reappears more or less in the position where the tracker lost him. This is not the case in the second image sequence. This problem is intensified, since when the target is lost, a different target (the wrong person) appears.

The auto-regressive process was trained on a sequence where if a person stood still in a sequence and then moved again, the movement would be in the opposite direction to the original movement (similar to the sequence in Figure 6.1). Also the particle filter uses only the previous state when predicting the current state, and as the particle filter resamples, diversity of the particles is lost. Although one could train the auto-regressive process differently, this would not solve the problem completely, since the main reason for the failure of the tracker is that too little of the particle filter's history is used when tracking through occlusions.

Thus a limitation of the PFAAM is that it cannot handle occlusions if the tracked object is occluded by an object of the same type. To rectify this, one could derive and implement a particle filter with more history. Alternatively, one could use an object recognition system together with the tracker. Thus, when the particle filter loses its target, it is reinitialised with the objects found by the recognition system.

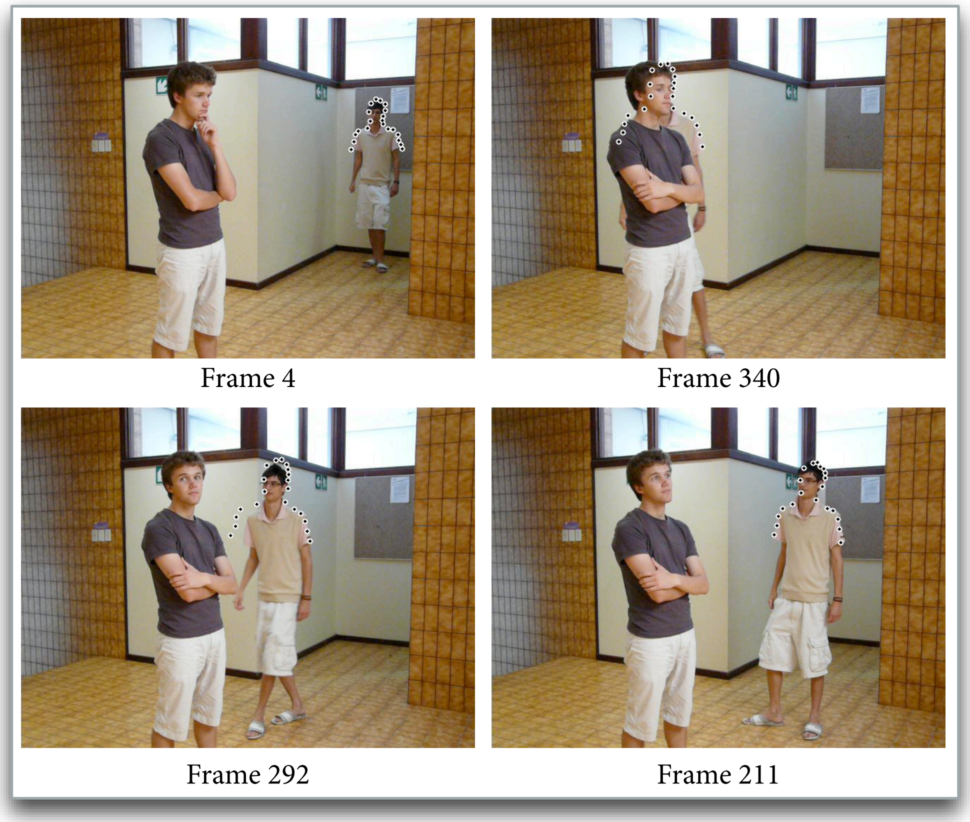


Figure 6.1: *Tracking a person passing another. Occlusion occurs and the tracker recovers.*

6.3 Recommendations and future directions

To track deformable objects, an active appearance model and particle filter combination is a viable alternative. If an active contour implementation is available, one can use it in tandem with an active appearance model to obtain an accurate and robust tracker.

Future work would include incorporating a background model to further improve the running time of the tracker. It would also be interesting to attempt to quantitatively compare the active appearance model and particle filter combination with other silhouette trackers. As discussed in this work, two of the obstacles are to obtain ground-truth and standardised implementations of the the tracking algorithms. There is however already work done in this direction, and more work will surely follow.

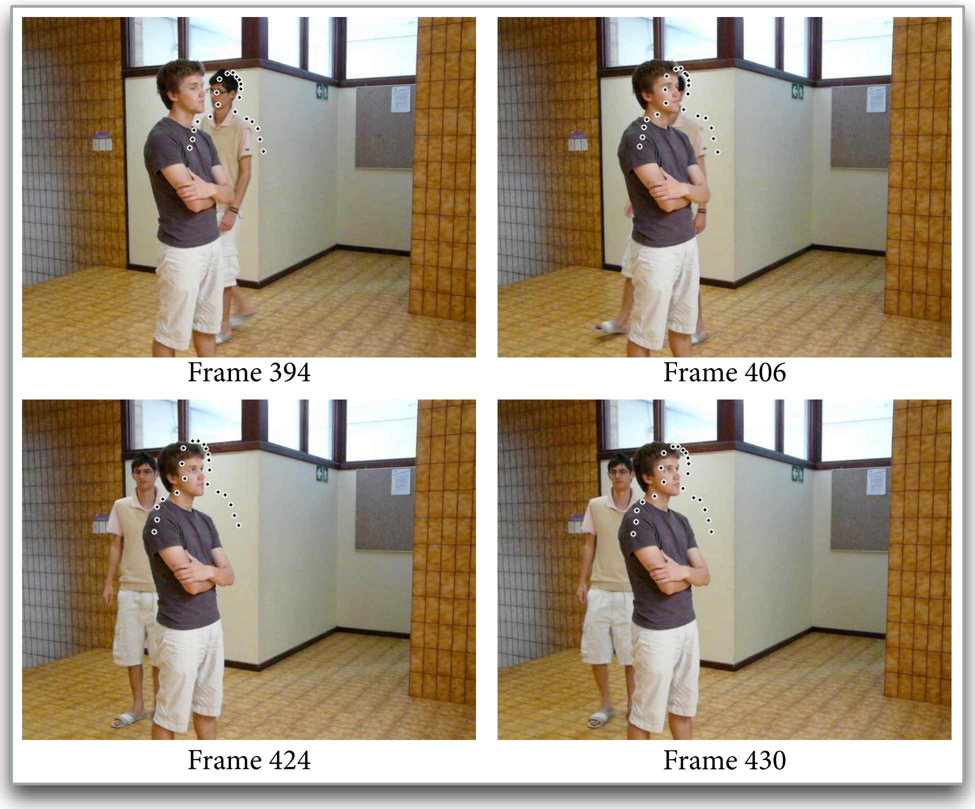


Figure 6.2: *Tracking the same person as in Figure 6.1. Occlusion occurs, but this time the tracker does not recover.*

Appendices

Appendix

A

Rules for Gaussian distributions

Mathematics, rightly viewed, possess not only truth, but supreme beauty—a beauty cold and austere, like that of sculpture.

— Bertrand Russel

Given a marginal pdf

$$p\left(\boldsymbol{x}\right) = \mathcal{N}\left(\boldsymbol{\mu}, \Lambda^{-1}\right) \tag{A.1}$$

and a conditional pdf

$$p\left(\boldsymbol{y}|\boldsymbol{x}\right) = \mathcal{N}\left(\mathrm{A}\boldsymbol{x} + \boldsymbol{b}, \mathrm{L}^{-1}\right) \tag{A.2}$$

where $\mathcal{N}\left(\boldsymbol{m}, \mathrm{C}\right)$ is a Gaussian pdf with mean $\boldsymbol{m}$ and covariance C. Then

$$p\left(\boldsymbol{y}\right) = \mathcal{N}\left(\mathrm{A}\boldsymbol{\mu} + \boldsymbol{b}, \mathrm{L}^{-1} + \mathrm{A}\Lambda^{-1}\mathrm{A}^{\mathrm{T}}\right) \tag{A.3}$$

and

$$p\left(\boldsymbol{x}|\boldsymbol{y}\right) = \mathcal{N}\left(\Sigma\left\{\mathrm{A}^{\mathrm{T}}\mathrm{L}\left(\boldsymbol{y} - \boldsymbol{b}\right) + \Lambda\boldsymbol{\mu}\right\}, \Sigma\right) \tag{A.4}$$

where

$$\Sigma = \left(\Lambda + A^{T} L A\right)^{-1}. \tag{A.5}$$

A proof can be found in (Bishop, 2006).

Appendix

B Epipolar geometry

Attempt the end, and never stand to doubt;
Nothing's so hard but search will find it out.

— Robert Herrick

In a stereo reconstruction set-up, two images are given with corresponding points defined in them. Knowledge of the cameras that produced these images is also assumed. With all this information, one can obtain the 3D coordinates of the original corresponding points. In this chapter, an overview of the solution to the 3D reconstruction problem is presented. We commence by discussing in Section B.1 the 2D projective geometry that forms the basis for the camera model we assume. Thereafter, in Section B.2, the camera model is detailed. Once the camera model is known, Section B.3 uses this model to define epipolar geometry—the geometry of two cameras—and finally put all the information together to obtain the 3D coordinates of the original points by means of triangulation. For a thorough exposition of these topics, the reader is referred to Hartley and Zisserman (2003).

B.1 2D projective geometry

Projective geometry unifies the concept of the intersection of two lines. In Euclidian geometry, the intersection of parallel lines is not well-defined. However, in projective geometry the intersection of all lines, including parallel lines, is well-defined.

Consider a line described by

$$ax + by + c = 0. \tag{B.1}$$

If we let $\boldsymbol{x} = \left[x, y, 1\right]^{\mathrm{T}}$ and $\boldsymbol{l} = \left[a, b, c\right]^{\mathrm{T}}$, the the line in (B.1) is given by

$$\boldsymbol{x}^{\mathrm{T}}\boldsymbol{l} = 0. \tag{B.2}$$

It is also true that the line $k\boldsymbol{l}$, $k \neq 0$ gives rise to the same line (B.1). Similarly, the line $k\boldsymbol{x}^{\mathrm{T}}\boldsymbol{l} = 0$, where $k \neq 0$ is also the same line. For these reasons, we regard $\boldsymbol{l} = \left[a, b, c,\right]^{\mathrm{T}}$ and $k\boldsymbol{l} = \left[ka, kb, kc,\right]^{\mathrm{T}}$, and $\boldsymbol{x} = \left[x, y, 1,\right]^{\mathrm{T}}$ and $k\boldsymbol{x} = \left[kx, ky, k,\right]^{\mathrm{T}}$ as equivalent. In all these cases, $k \neq 0$. Vectors in these equivalent classes are referred to as homogenous vectors.

The vectors $\boldsymbol{x}$ and $\boldsymbol{l}$ are elements of the projective space $\mathbb{P}^2$. The study of $\mathbb{P}^2$ is projective geometry. We can map a vector from $\mathbb{R}^2$ to $\mathbb{P}^2$ by concatenating a 1, *i.e.*,

$$\begin{bmatrix} x \\ y \end{bmatrix} \in \mathbb{P}^2 \rightarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \in \mathbb{R}^2. \tag{B.3}$$

A vector in $\mathbb{P}^2$ is mapped to $\mathbb{R}^2$ by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \in \mathbb{R}^2 \rightarrow \begin{bmatrix} \frac{x}{z} \\ \frac{y}{z} \end{bmatrix} \in \mathbb{P}^2. \tag{B.4}$$

Given two lines $\boldsymbol{l}_1$ and $\boldsymbol{l}_2$, the intersection of these lines is

$$\boldsymbol{x} = \boldsymbol{l}_1 \times \boldsymbol{l}_2 \tag{B.5}$$

where $\times$ is the Cartesian cross product. Moreover, the line through $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ is

$$\boldsymbol{l} = \boldsymbol{x}_1 \times \boldsymbol{x}_2. \tag{B.6}$$

A proof of (B.5) and (B.6) can be found in (Hartley and Zisserman, 2003).

As an example, consider the lines

$$\begin{aligned} \boldsymbol{l}_1 &: \; x - 1 = 0 \\ \boldsymbol{l}_2 &: \; x - 2 = 0. \end{aligned}$$

Then the intersection is

$$\boldsymbol{x} = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ -2 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}.$$

From this example, we see that two parallel lines intersect in a well-defined point. This point is known as the point at infinity.

B.2 The camera model

A camera is modelled as a transformation from $\mathbb{P}^3$ to $\mathbb{P}^2$. Suppose that $\mathbf{X}$ is the object in world coordinates. This point is projected to $\boldsymbol{x}$ in pixel coordinates on the CCD. This projection is given by

$$\boldsymbol{x} = \mathrm{P}\mathbf{X} \tag{B.7}$$

where P is a 3×4 homogenous matrix with rank 3. This is illustrated in Figure B.1. The matrix P is the camera projection matrix.

In order to find P, camera calibration is performed. During camera calibration, an image of a calibration object is taken. The geometry of the calibration object is assumed known. Points on the calibration object are identified in the image taken. Then the camera matrix is found by solving for P in

$$\boldsymbol{x}_i = \mathrm{P}\mathbf{X}_i \tag{B.8}$$

where $\mathbf{X}_i$ is a known point on the calibration object and $\boldsymbol{x}_i$ is the corresponding point in the image plane. The index i runs through the corresponding points identified on the calibration object and the image taken of it. Figure B.2 illustrates a simple Lego calibration object.

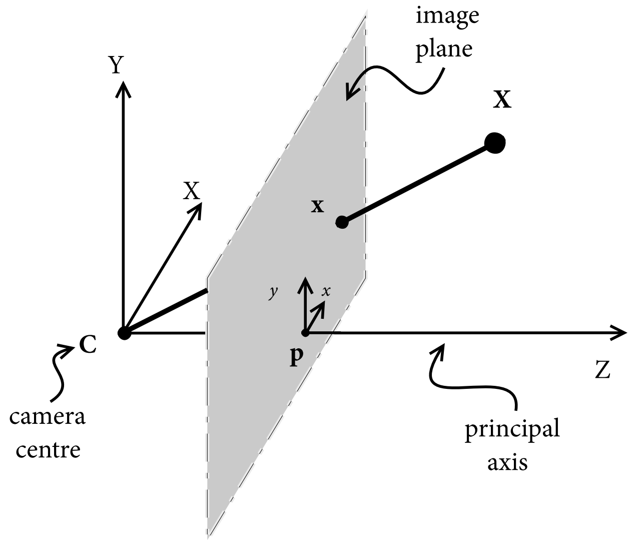


Figure B.1: *The camera model.*

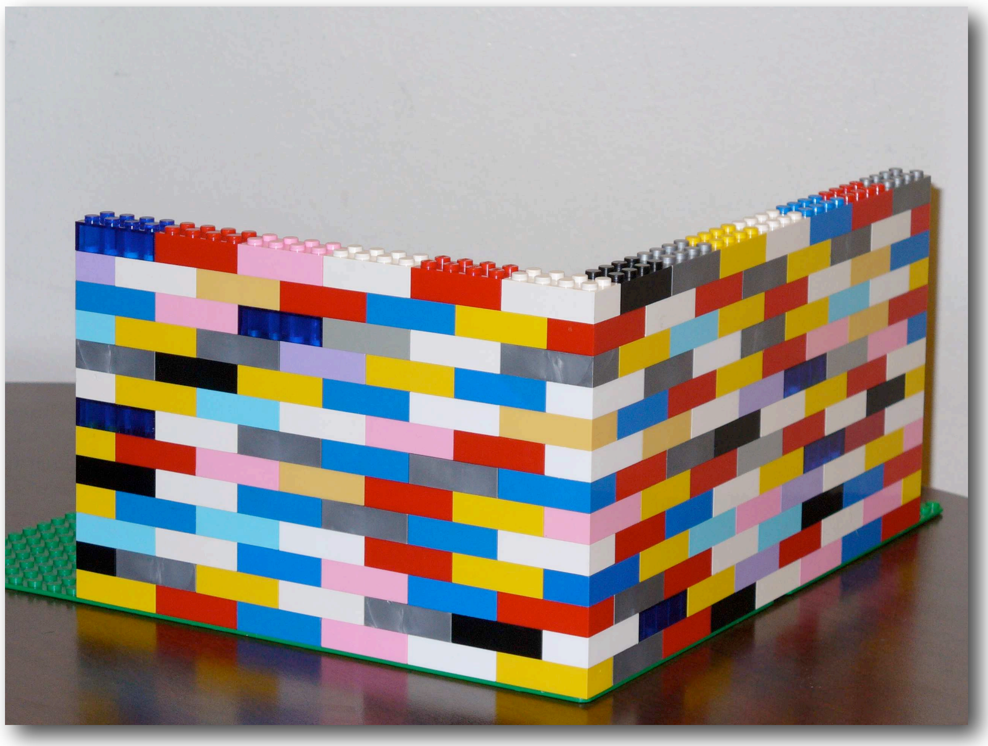


Figure B.2: *An example calibration object built from Lego blocks.*

B.3 Epipolar geometry and triangulation

So far this discussion has only considered one camera. For two cameras, referred to as a stereo pair, we use epipolar geometry.

Figure B.3 details epipolar geometry. In this figure, the point **X** is projected onto $\boldsymbol{x}$ in the image plane by the camera matrix P, while $\boldsymbol{x}'$ is the image of **X** under the camera matrix P′. The camera centres, **X**, and the images $\boldsymbol{x}$ and $\boldsymbol{x}'$ lie in the same plane. This is the epipolar plane. The epipoles are formed by intersections of the line that joins the camera centres. An epipolar plane intersects the image plane at an epipolar line.

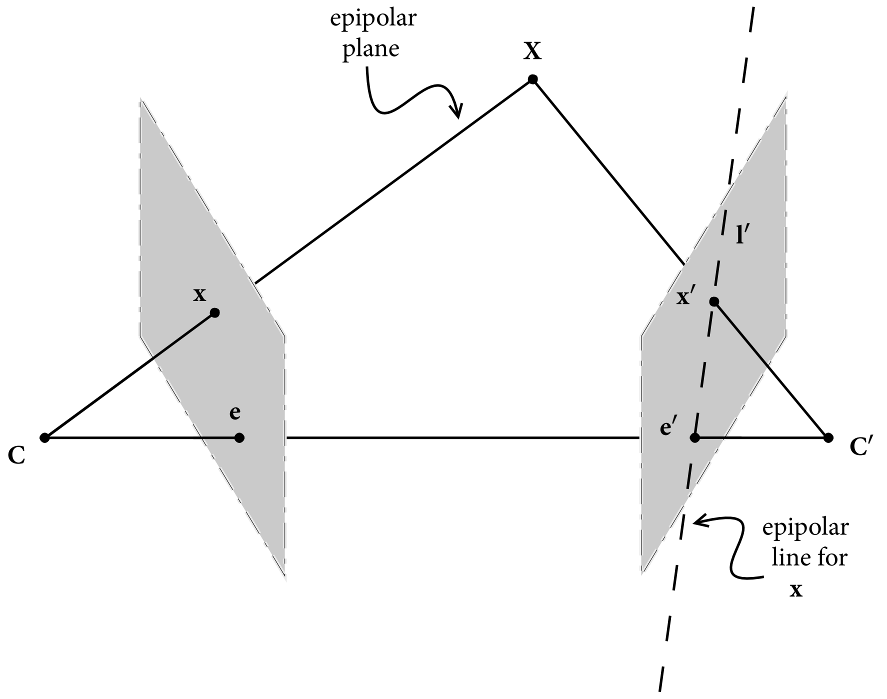


Figure B.3: *Graphical depiction of epipolar geometry.*

Epipolar geometry is also described algebraically by the fundamental matrix F which is a 3 × 3 homogeneous matrix that satisfies

$$\boldsymbol{x}'\mathrm{F}\boldsymbol{x} = 0 \tag{B.9}$$

for all corresponding points $\boldsymbol{x}$ and $\boldsymbol{x}'$.

If two images $\boldsymbol{x}$ and $\boldsymbol{x}'$ are given, the point $\mathbf{X}$ is found using triangulation. Thus two camera matrices P and P′ as well as a pair of corresponding points $\boldsymbol{x}$ and $\boldsymbol{x}'$ are given. The aim is to calculate the 3D feature $\mathbf{X}$ where

$$\boldsymbol{x} = \mathrm{P}\mathbf{X}, \tag{B.10}$$

$$\boldsymbol{x}' = \mathrm{P}'\mathbf{X}. \tag{B.11}$$

One cannot solve for $\mathbf{X}$ directly in (B.10) and (B.11) because homogenous coordinates are used. However, $\boldsymbol{x} \times \mathrm{P}\mathbf{X} = 0$ and from (B.10) we get

$$\begin{aligned} y\boldsymbol{p}_3^{\mathrm{T}}\mathbf{X} - z\boldsymbol{p}_2\mathbf{X} &= \mathbf{0} \\ z\boldsymbol{p}_1^{\mathrm{T}}\mathbf{X} - z\boldsymbol{p}_3\mathbf{X} &= \mathbf{0} \\ x\boldsymbol{p}_2^{\mathrm{T}}\mathbf{X} - z\boldsymbol{p}_1\mathbf{X} &= \mathbf{0}, \end{aligned}$$

where p_i^{T}, $i = 1, \ldots, 3$ are the rows of P. Similar equations is written for $\boldsymbol{x}' \times \mathrm{P}\mathbf{X} = 0$. Then a system

$$\mathrm{A}\mathbf{X} = \mathbf{0} \tag{B.12}$$

is formed by taken the first two equations of the two camera systems. The 3D feature $\mathbf{X}$ is then solved using (B.12).

B.4 Summary

This chapter briefly described epipolar geometry, *i.e.*, the geometry of two cameras. It introduced projective geometry and showed the camera model based on this geometry. Having a camera model, the geometry for two cameras were defined and used to reconstruct a 3D feature given two images of the point. For more a more detailed discussion, refer to Hartley and Zisserman (2003).

Appendix

C Notes on the software implementation

Ad aspra per aspera et per ludum.

— W.A. Fowler

The algorithms developed and described in this dissertation, were implemented as a software library. We refer to this software library as Vestigo. The software was predominantly implemented in C++. In order to run the software, at least Apple OS X 10.5.7, Linux 2.6.27, or Microsoft Windows XP is required. A version has also been developed for the Matrix Vision mvBlueCOUGAR-P smart camera (Matrix Vision, 2008a).

This chapter gives an overview of the implemented software. Section C.1 explains the filesystem layout of Vestigo. Section C.2 provides installation instructions for the software. Finally, Section C.3 describes the main classes implemented.

C.1 Organisation of the library

The top level of Vestigo consists of the folders below. The content of the folder is shown next to it:

`bin/`	Executables
`config/`	Temporary files when building the library
`lib/`	Static linked libraries
`src/`	Source code

`python/`	Utility scripts implemented in Python

The `src` folder is divided as:

`exe/`	Driver programs; C++ classes in this folder contain `main` methods
`st3p/`	Third party libraries
`stmvc/`	Classes related to the smart camera implementation
`sttrk/`	The source code of the tracker

We use three third party libraries, apart from `vxl` and all its dependencies. These libraries are located in `st3p`. Therefore, the folder `st3p` is organised as:

`configfile/`	Third party library to read and write configuration files
`demolib/`	Source code of DeMoLib (AAM library)
`libsvm/`	Source code of `libsvm`; used by DeMoLib

C.2 Installation instructions

Suppose that the Subversion repository url to Vestigo's root folder and DeMoLib's root folder are `svn://localhost/Vestigo` and `svn://localhost/DeMoLib` respectively. The exact url, together with authorisation credentials, can be obtained by contacting the author. We assume that `vxl` is installed. We use the notation $ to indicate interaction at a Bash command shell. The steps below are then followed to install Vestigo.

1. Check out Vestigo from the repository.
```
$ svn co svn://localhost/Vestigo/trunk Vestigo
```

2. Change to the `st3p` folder and check out DeMoLib and `libsvm` from the repository.
```
$ cd Vestigo/src/st3p
$ svn co svn://localhost/DeMoLib/trunk/src/lib demolib
$ svn co svn://localhost/DeMoLib/trunk/trink/src/libsvm libsvm
```

3. In the file `src/CMakeLists.txt`, change the value of `VXL_DIR` to reflect the installation directory of `vxl` on the system.

4. Create the `makefiles` using `cmake`.
   ```
   $ cd config
   $ cmake -i ../src/
   ```

5. Compile the source code using `make`.
   ```
   $ cd Vestigo/config
   $ make
   ```

C.3 Overview of main classes

The C++ classes of the tracker are located in the folder `src/sttrk`. Here follows a list of the main classes with descriptions:

`sttrk_generic_pf`	This class is a pure virtual, templatised class implementing a PF. It forms the base of all PF implementations. When this class is extended, methods providing a process function and a measurement function, must be implemented.
`sttrk_aam`	This class provides an AAM implementation. The underlying AAM implementation is DeMoLib. The class uses the facade design pattern (Gamma et al., 1994) to reduce dependencies on DeMoLib.
`sttrk_aam_pf`	The class implements a combination of an AAM and PF. It inherits from `sttrk_generic_pf`.
`sttrk_aam_tracker`	An entry point towards the tracking library is provided. This class can be seen as the top level of the AAM tracker.
`sttrk_aam_state`	The state of the AAM-based PF is encapsulated in this class. It is thus the type of the particles in `sttrk_aam_pf`.
`sttrk_arp`	This class predicts the values of the next state using a second-order auto-regressive process.
`sttrk_sym_aam`	This pure virtual class forms the base of all measurement functions.
`sttrk_sym_aam_factory`	Using the factory design pattern (Gamma et al., 1994), this class creates classes of type `sttrk_sym_aam`.

`sttrk_aam_orig`	This class provides the measurement function for the original AAM formulation.
`sttrk_aam_di`	The measurement function for the iterative non-linear AAM is implemented by this class.
`sttrk_aam_ic`	It implements the measurement function for the inverse-compositional project-out AAM.
`sttrk_weight_index`	This helper class associate a weight with a particular particle. The class is only used when sorting particles to find the most promising particles for local optimisation.
`sttrk_options`	The values of all the parameters in the system are encapsulated in this class.
`sttrk_options_reader`	A configuration file is read by the class. The information in the file is used to initialise the values of a `sttrk_options` object.
`sttrk_math_util`	Miscellaneous mathematical functions are implemented as static functions in this class.

Authored and co-authored publications

Fleck, S., Hoffmann, M., Hunter, K., and Schilling, A. (2007). PFAAM—an active appearance model based particle filter for both robust and precise tracking. In *Proceedings of the Fourth Canadian Conference on Computer and Robot Vision (CRV 2007)*, pages 339–346.

Hoffmann, M., Herbst, B., and Hunter, K. (2007a). Occlusions and active appearance models. In *Second International Conference on Computer Vision Theory and Applications (VISAPP 2007)*, volume IU/MTSV, pages 441–446.

Hoffmann, M., Herbst, B., and Hunter, K. (2007b). On visual object tracking using active appearance models. *SAIEE Africa Research Journal*, 98(2):52–58.

Hoffmann, M., Hunter, K., and Herbst, B. (2008a). The hitchhiker's guide to the particle filter. In *Proceedings of Nineteenth Annual Symposium of the Pattern Recognition Association of South Africa (Prasa 2008)*, pages 33–38.

Hoffmann, M., Swart, A., Hunter, K., Herbst, B., Fleck, S., and Straßer, W. (2008b). Model-based robust and precise tracking embedded in smart cameras—the pfaam. In *Second ACM/IEEE International Conference on Distributed Smart Cameras (ICDSC 2008)*, pages 1–8.

Bibliography

Abboud, B., Davoine, F., and Dang, M. (2004). Facial expression recognition and synthesis based on an appearance model. *Signal Processing: Image Communication*, 19(8):723–740.

Bagnato, L., Sorci, M., Antonini, G., Baruffa, G., Maier, A., Leathwood, P., and Thiran, J. (2007). Robust infants face tracking using active appearance models: a mixed-state CONDENSATION approach. *Lecture Notes in Computer Science*, 4841:13.

Baker, S. and Matthews, I. (2004). Lucas-Kanade 20 years on: A unifying framework. *International Journal of Computer Vision*, 56(3):221–255.

Bar-Shalom, Y., Li, X. R., and Kirubarajan, T. (2001). *Estimation with Application to Tracking and Navigation*. John Wiley & Sons, Inc.

Birkbeck, N. and Jagersand, M. (2004). Visual tracking using active appearance models. In *Proceedings of the First Canadian Conference on Computer and Robot Vision*, pages 2–9, Washington, DC, USA. IEEE Computer Society.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Blake, A. and Isard, M. (1998). *Active contours*. Springer-Verlag, London.

Blake, A., Isard, M., and Reynard, D. (1995). Learning to track the visual motion of contours. *Artificial Intelligence*, 78(1-2):179–212.

Broida, T. and Chellappa, R. (1986). Estimation of object motion parameters from noisy images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(1):90–99.

Cantrill, B., Shapiro, M., and Leventhal, A. (2004). Dynamic instrumentation of production systems. In *USENIX Annual Technical Conference*, pages 15–28.

Chen, Y., Rui, Y., and Huang, T. (2001). JPDAF based HMM for real-time contour tracking. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 543–550. IEEE Computer Society; 1999.

Comaniciu, D. and Meer, P. (1999). Mean shift analysis and applications. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 1197–1203.

Comaniciu, D., Ramesh, V., and Meer, P. (2003). Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–577.

Cootes, T., Edwards, G., Taylor, C., et al. (2001). Active appearance models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):681–685.

Cootes, T., Taylor, C., Cooper, D., and Graham, J. (1995). Active Shape Models-Their Training and Application. *Computer Vision and Image Understanding*, 61(1):38–59.

Cootes, T. F., Edwards, G. J., and Taylor, C. J. (1998). Active appearance models. In *European Conference on Computer Vision*, volume 2, pages 484–498.

de Boor, C. (1978). *A Practical Guide to Splines*. Springer, New York.

Dietrich, D., Garn, H., Kebschull, U., C.Grimm, and Ben-Ezra, M., editors (2007). *EURASIP Journal on Embedded Systems, Volume 2007*. Hindawi Publishing Corporation.

Doucet, A. and de Freitas, N. (2001). *Sequential Monte Carlo Methods in Practice*. Springer.

Doucet, A., Godsill, S., and Andrieu, C. (2000). On sequential Monte Carlo sampling methods for Bayesian filtering. *Statistics and Computing*, 10(3):197–208.

Edwards, G. J., Taylor, C. J., and Cootes, T. F. (1998). Interpreting face images using active appearance models. In *Proceedings of the Third. International Conference on Face & Gesture Recognition*, pages 300–305, Washington, DC, USA. IEEE Computer Society.

Erdem, C., Sankur, B., and Tekalp, A. (2004). Performance measures for video object segmentation and tracking. *IEEE Transactions on Image Processing*, 13(7):937–951.

Faggian, N., Paplinski, A., and Sherrah, J. (2006). Active appearance models for automatic fitting of 3D morphable models. In *IEEE International Conference on Video and Signal Based Surveillance*, pages 90–90, Sydney, Australia.

Fischer, J., Eichler, M., Bartz, D., and Straßer, W. (2007). A hybrid tracking method for surgical augmented reality. *Computers & Graphics*, 31(1):39–52.

Fleck, S. and Straßer, W. (2008). Smart camera based monitoring system and its application to assisted living. *Proceedings of the IEEE*, 96(10):1698–1714.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Gennery, D. B. (1992). Visual tracking of known three-dimensional objects. *International Journal of Computer Vision*, 7(3):243–270.

Goodall, C. (1991). Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical Society. Series B (Methodological)*, 53(2):285–339.

Hamlaoui, S. and Davoine, F. (2005). Facial action tracking using particle filters and active appearance models. In *Proceedings of the 2005 joint conference on Smart Objects and Ambient Intelligence*, pages 165–169, Grenoble, France.

Hansen, D. W., Nielsen, M., Hansen, J. P., Johansen, A. S., and Stegmann, M. B. (2002b). Tracking eyes using shape and appearance. In *IAPR Workshop on Machine Vision Applications - MVA*, pages 201–204.

Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of The Fourth Alvey Vision Conference*, volume 15, pages 147–151, Manchester, UK.

Hartley, R. and Zisserman, A. (2003). *Multiple view geometry in computer vision*. Cambridge Univ Pr.

Ho, Y. and Lee, R. (1964). A Bayesian approach to problems in stochastic estimation and control. *IEEE Transactions on Automatic Control*, 9(4):333–339.

Hoffmann, M. (2009). Phd supported material. Retrieved 17 November 2009 from `http://dip.sun.ac.za/~mcelory/phd`.

Hu, W., Tan, T., Wang, L., and Maybank, S. (2004). A survey on visual surveillance of object motion and behaviors. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 34(3):334–352.

Huttenlocher, D., Noh, J., and Rucklidge, W. (1993). Tracking non-rigid objects in complex scenes. In *Proceedings of the Fourth International Conference on Computer Vision*, pages 93–101.

Isard, M. and Blake, A. (1998a). Condensation—conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28.

Isard, M. and Blake, A. (1998b). Icondensation: Unifying low-level and high-level tracking in a stochastic framework. In *Proceedings of the Fifth European Conference on Computer Vision*, volume 1, pages 893–908, London, UK. Springer-Verlag.

Jacobs, E. (2005). Deterministic tracking using active contours. Master's thesis, University of Stellenbosch.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 82.

Kass, M., Witkin, A., and Terzopoulos, D. (1987). Snakes: active contour models. In *Proceedings of the First International Conference on Computer Vision*, pages 258–269, London, England.

Kitagawa, G. (1996). Monte Carlo filter and smoother for non-Gaussian nonlinear state space models. *Journal of Computational and Graphical Statistics*, 5:1–25.

Larsen, R., Stegmann, M. B., Darkner, S., Forchhammer, S., Cootes, T. F., and Ersbøll, B. K. (2007). Texture enhanced appearance models. *Computer Vision and Image Understanding*, 106:20–30.

Lauritzen, S. (1996). *Graphical Models*. Oxford University Press.

Liu, J. and Chen, R. (1998). Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93:1032–1044.

Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. In *Proceedings of the 1981 DARPA Image Understanding Workshop*, pages 121–130.

MacCormick, J. (2002). *Stochastic Algorithms for Visual Tracking: Probabilistic Modelling and Stochastic Algorithms for Visual Localisation and Tracking (Distinguished Dissertations)*. Springer London Ltd.

Mardia, C. A. G. K. V. (1998). A review of image-warping methods. *Journal of Applied Statistics*, 25(2):155–171.

Matrix Vision (2008a). Matrix vision product page. Retrieved 25 January 2008 from `http://www.matrix-vision.com/products/hardware/mvbluecougar-p.php?lang=en`.

Matrix Vision (2008b). mvIMPACT Product Page. Retrieved 25 January 2008 from `http://www.matrix-vision.com/products/software/mvimpact.php`.

Matthews, I. and Baker, S. (2004). Active appearance models revisited. *International Journal of Computer Vision*, 60(2):135–164.

Matthews, I., Cootes, T. F., Bangham, J. A., Cox, S., and Harvey, R. (2002). Extraction of visual features for lipreading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(2):198–213.

Maybeck, P. (1979). *Stochastic models estimation and control, Vol. 1, Mathematics in science and engineering*. Academic Press.

Mittrapiyanuruk, P., DeSouza, G. N., and Kak, A. C. (2005). Accurate 3D tracking of rigid objects with occlusion using active appearance models. In *IEEE Workshop on Motion and Video Computing*, volume 2, pages 90–95.

Naishlos, D. (2004). Autovectorization in GCC. In *Proceedings of the 2004 GCC Developers Summit*, pages 105–118.

Nummiaro, K., Koller-Meier, E., and Van Gool, L. (2002). A color-based particle filter. In *First International Workshop on Generative-Model-Based Vision, in conjuction with ECCV'02*, pages 53–60.

Pearl, J. (1988). *Probabilistic Reasoning In Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers, Inc.

Pérez, P., Hue, C., Vermaak, J., and Gangnet, M. (2002). Color-based probabilistic tracking. In *Proceedings of the Seventh European Conference on Computer Vision*, volume 1, pages 661–675, London, UK. Springer-Verlag.

Point Grey Research Inc. (2009). Firefly product page. Retrieved 23 June 2009 from `http://www.ptgrey.com/products/fireflymv/index.asp`.

Rinner, B. and Wolf, W., editors (2008). *Proceedings of the IEEE, Special Issue on Distributed Smart Cameras*. IEEE.

Ristic, B., Arulampalam, S., and Gordon, N. (2004). *Beyond the Kalman filter—particle filters for tracking applications*. Artech House, 1st edition.

Saragih, J. (2009). Demolib. Retrieved 18 June 2009 from `http://users.rsise.anu.edu.au/~u3302419/`.

Saragih, J. and Göcke, R. (2006). Iterative error bound minimisation for AAM alignment. In *Proceedings of the Eighteenth International Conference on Pattern Recognition ICPR 2006*, volume 2, pages 1192–1195, Hong Kong. IEEE.

Saragih, J. and Göcke, R. (2006). Learning active appearance models from image sequences. In *Proceedings of the HCSNet workshop on Use of Vision in Human-Computer Interaction*, volume 56, pages 51–60. Australian Computer Society, Inc. Darlinghurst, Australia, Australia.

Saragih, J. and Goecke, R. (2007). A nonlinear discriminative approach to aam fitting. In *IEEE Eleventh International Conference on Computer Vision*, pages 1–8.

Sethi, I. and Jain, R. (1987). Finding trajectories of feature points in a monocular image sequence. *IEEE Transactions on pattern analysis and machine intelligence*, 9(1):56–73.

Shi, J. and Tomasi, C. (1994). Good features to track. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition.*, pages 593–600.

Smola, A. and Schölkopf, B. (2004). A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222.

Stegmann, M. B. (2000). Active appearance models: theory, extensions and cases. Master's thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby.

Stegmann, M. B. (2001). Object tracking using active appearance models. In Olsen, S. I., editor, *Proceedings of the Tenth Danish Conference on Pattern Recognition and Image Analysis*, volume 1, pages 54–60, Copenhagen, Denmark. DIKU.

Stegmann, M. B., Ersbøll, B. K., and Larsen, R. (2003). FAME—a flexible appearance modelling environment. *IEEE Transactions on Medical Imaging*, 22(10):1319–1331.

Stegmann, M. B., Nilsson, J. C., and Grønning, B. A. (2001). Automated segmentation of cardiac magnetic resonance images. In *Proceedings of International Society of Magnetic Resonance In Medicine*, volume 2, page 827, Glasgow, Scotland, UK.

Sundvall, P. (2009). Particle filter example. Retrieved 5 July 2009 from `http://www2.paulsundvall.net/optfilt/particle_filter.html`.

Sung, J. and Kim, D. (2004). Extension of AAM with 3D shape model for facial shape tracking. In *International Conference on Image Processing*, volume 5, pages 3363–3366.

Sung, J. and Kim, D. (2006). A background robust Active Appearance Model using active contour technique. *The Journal of the Pattern Recognition Society*, 40(1):108–120.

Terzopoulos, D. and Szeliski, R. (1993). *Tracking with Kalman snakes*, pages 3–20. MIT Press, Cambridge, MA, USA.

Theobald, B.-J., Matthews, I., and Baker, S. (2006). Evaluating error functions for robust active appearance models. In *Proceedings of the International Conference on Automatic Face and Gesture Recognition*, pages 149 – 154.

Veenman, C., Reinders, M., and Backer, E. (2001). Resolving motion correspondence for densely moving points. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(1):54–72.

vxl-maintainers (2009). The vxl home page. Retrieved 10 November 2009 from `http://vxl.sourceforge.net/`.

Walker, K., Cootes, T., and Taylor, C. (2000). Determining correspondences for statistical models of facial appearance. In *Proceedings of the IEEE International Conference on Automatic Face and Gesture Recognition*, pages 271–276.

Wu, Y. and Huang, T. S. (1999). Vision-based gesture recognition: a review. In *Proceedings of the International Gesture Workshop on Gesture-Based Communication in Human-Computer Interaction*, pages 103–115, London, UK. Springer-Verlag.

Yilmaz, A., Javed, O., and Shah, M. (2006). Object tracking: a survey. *ACM Computer Survey*, 38(4):13.

Yilmaz, A., Li, X., and Shah, M. (2004). Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(11):1531–1536.

Yuille, A. and Kersten, D. (2006). Vision as Bayesian inference: analysis by synthesis? *Trends in Cognitive Sciences*, 10(7):301–308.

Zarchan, P. and Musoff, H. (2005). *Fundamentals of Kalman Filtering: A Practical Approach.* Aiaa.

Zhou, S., Chellappa, R., and Moghaddam, B. (2004). Visual tracking and recognition using appearance-adaptive models in particle filters. *IEEE Transactions on Image Processing*, 13(11):1491–1506.