



UNIVERSITEIT•STELLENBOSCH•UNIVERSITY
jou kennisvennoot • your knowledge partner

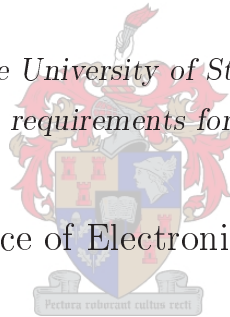
Deterministic Tracking using Active Contours

by

Emmerentia Jacobs

*Thesis presented at the University of Stellenbosch in partial
fulfilment of the requirements for the degree of*

Master of Science of Electronic Engineering



Department of Electric and Electronic Engineering
University of Stellenbosch
Private Bag X1, 7602 Matieland, South Africa

Study leaders: Prof B. Herbst and Prof J.A. du Preez

April 2005

Copyright © 2005 University of Stellenbosch
All rights reserved.

Declaration

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and that I have not previously in its entirety or in part submitted it at any university for a degree.

Signature:

E. Jacobs

Date:

Abstract

Deterministic Tracking using Active Contours

E. Jacobs

Department of Electric and Electronic Engineering

University of Stellenbosch

Private Bag X1, 7602 Matieland, South Africa

Thesis: MScEng

April 2005

This thesis relates to the problem of deterministically tracking an object that moves in a video sequence. A variety image processing and contour fitting algorithms and techniques are discussed. The tracking algorithm fits B-spline templates as is set out in the work done by Blake and Isard. The method builds on the concept that only certain deformations of the moving object's bounding curve is allowed. This is governed by projection onto a subspace that allow only the wanted deformations. The deformations are restricted to be mostly linear, but small, nonlinear deformations can also be accomodated.

Uittreksel

Deterministic Tracking using Active Contours

E. Jacobs

Departement Elektries en Elektroniese Ingenieurswese

Universiteit van Stellenbosch

Privaatsak X1, 7602 Matieland, Suid Afrika

Tesis: MScIng

April 2005

Die tesis ondersoek die probleem om 'n bewegende voorwerp in 'n videobeeld heeltemal deterministies te volg. Verskeie beeldverwerkings- en kontoerpasings tegnieke en algoritmes word hier bespreek. Die volgings-algoritme maak van die passing van 'B-spline templates' gebruik, soos weergegee in die werk deur Isard en Blake. Die metode werk vanaf die uitgangspunt dat die bewegende voorwerp se buitelyn net op sekere maniere kan vervorm. Vervorming word beheer deur projeksies na 'n subruimte wat slegs gewenste transformasies toelaat. Toelaatbare vervormings is grootliks lineêr, maar klein nie-linieêre vervormings kan ook deur die stelsel hanteer word.

Acknowledgements

Many thanks to the following people who have contributed to making this work possible:

- All the people in the DSP lab for their input and help, in particular Pieter Rautenbach, Stephen Ballot, Emli-Mari Nel and Francois Cilliers.
- Prof. Johan du Preez of the University of Stellenbosch as my co-study leader.
- Prof. Ben Herbst of the University of Stellenbosch as my study leader.

Contents

Declaration	ii
Abstract	iii
Uittreksel	iv
Acknowledgements	v
Contents	vi
List of Figures	ix
Nomenclature	xv
1 Introduction	1
1.1 Objective	2
1.2 Background	2
1.3 Literature Overview	4
1.4 Contribution	5
1.5 Overview of Chapters	5
2 Snakes	7
2.1 The Parametric Snake Model	7
2.2 The Canny Edge Detector	10
2.3 Corners	12
2.4 Handling Concavities	15
2.5 An Example	18
2.6 Summary	19

<i>Contents</i>	vii
3 Spline Curves	20
3.1 B-Splines	21
3.2 Norms and Moments	25
3.3 Summary	28
4 Curve Fitting	29
4.1 Shape Space	29
4.2 The Space of Euclidean Similarities	31
4.3 Planar Affine Shape Space	32
4.4 Norms and Moments in the Planar Affine Shape Space	34
4.5 Keyframes	35
4.6 The Fitting Algorithm	38
4.7 Reparameterisation	45
4.8 Summary	46
5 Tracking Planar Affine Objects	48
5.1 Obtaining the Necessary Curves	48
5.2 The General Tracking Algorithm	49
5.3 Tracking Results	50
5.4 An Application	65
5.5 Summary	65
6 Tracking with Keyframes	67
6.1 How to Find Keyframes with PCA	67
6.2 Tracking with Simulated Data	68
6.3 Tracking with Real Data	71
6.4 An Application	73
6.5 Summary	73
7 Kalman Filter: Comparative Results	74
7.1 The Kalman Filter	74
7.2 Comparative Results	81
7.3 Summary	82
8 Conclusion	84

<i>Contents</i>	viii
A Image Processing Filters	87
A.1 The Sobel Filter	87
A.2 The Gaussian Lowpass Filter	88
B Principle Component Analysis	90
C Dynamic Programming	92
D Optical Flow	94
E Results	96
Bibliography	97

List of Figures

2.1	The Greedy Algorithm: locating a local minimum point.	9
2.2	The eight possible edge directions surrounding a pixel.	11
2.3	Step 5 of the Canny edge detector.	12
	(a) Image gradient for a small part of the ring's edge.	12
	(b) Traversing the edge direction, looking for an edge.	12
	(c) The next pixel visited is adjacent to the just found edge pixel in a direction normal to the edge direction.	12
2.4	The influence of finding the corners during preprocessing.	13
	(a) Snake convergence without implementing the corners found.	13
	(b) The corners that were found.	13
	(c) The snake result with the found corners now implemented.	13
2.5	The influence of external forces on the convergence of the tradi- tional snake.	15
	(a) Convergence of the traditional snake.	15
	(b) The external energy.	15
	(c) Close-up of external energy within the concavity.	15
2.6	The influence of external forces on the convergence of the gradient vector flow snake.	16
	(a) Convergence of the GVF snake.	16
	(b) The GVF vector field.	16
	(c) Close-up of the GVF field within the concavity.	16
2.7	Using snakes to segment faces from the background.	18
	(a) Image before applying the snake.	18
	(b) The result of segmentation with the snake.	18

<i>List of Figures</i>	x
3.1 The first four B-splines.	21
3.2 The difference between interpolating and approximating splines.	23
(a) Interpolating 3 rd order spline.	23
(b) Approximating 3 rd order spline.	23
3.3 The influence of multiple knots.	24
(a) Approximating 3 rd order spline fitted through the control points.	24
(b) Approximating 3 rd order spline with multiple knots at some control points.	24
3.4 The influence of parameterisation on the norm or average distance between curves.	26
(a) The original control points.	26
(b) Distortion of some control points. Average displacement: norm = 1.05	26
(c) A change in the parameterisation. Average displacement: norm = 3.34	26
4.1 The influence of shape space when fitting curves.	30
(a) The template.	30
(b) The new feature points.	30
(c) Curve fitting without reduced variability.	30
(d) Curve fitting in shape space.	30
4.2 The four deformations from the space of Euclidean similarities.	32
(a) Horizontal translation.	32
(b) Vertical translation.	32
(c) Scaling.	32
(d) Rotation.	32
4.3 The six planar affine deformations.	33
(a) Horizontal translation.	33
(b) Vertical translation.	33
(c) Rotation.	33
(d) Horizontal scaling.	33
(e) Vertical scaling.	33
(f) Diagonal scaling.	33

4.4	The keyframes used to describe the two dimensional deformation of a cube rotating in three dimensions.	35
(a)	The template, \mathbf{Q}_0	35
(b)	Keyframe 1, \mathbf{Q}'_1	35
(c)	Keyframe 2, \mathbf{Q}'_2	35
4.5	Curve fitting with regularisation.	40
(a)	The noisy feature data \mathbf{Q}_f	40
(b)	The mean shape $\bar{\mathbf{X}}$	40
(c)	The fitted curve with $\alpha = 0.5$	40
(d)	The fitted curve with $\alpha = 3$	40
4.6	Recursive algorithm for curve fitting.	44
4.7	The feature curve without reparameterisation.	45
4.8	The curve fitting result for the features found in Figure 4.7.	45
4.9	The curve fitting result after the reparameterisation of the feature curve.	46
5.1	Normal line feature finding.	51
(a)	Finding the features with normal lines that are too long.	51
(b)	Finding the features with the object too far away.	51
(c)	Features found with the length of the normal line increased.	51
(d)	Interpolation features for control points that are still unmatched.	51
5.2	Random selection of result frames to indicate the performance of the curve fitting algorithm.	52
5.3	Improved curve fitting by reparameterising curves.	53
5.4	The result of the Canny edge detector applied to frame 1.	53
5.5	Two different ways of handling missing endpoints.	54
(a)	Solving endpoints by sharing them.	54
(b)	Solving endpoints by synthesizing features.	54
5.6	Tracking with an open curve.	55
5.7	Random selection of result frames to indicate the performance of the $\hat{\mathbf{X}} = W^+(\mathbf{Q}_f - \mathbf{Q}_0)$ solution, when tracking with an open curve.	55

5.8	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with an open curve, without regularisation.	55
5.9	Improved curve fitting by reparameterising curves.	56
5.10	A typical image with a busy background and its edge detection result.	56
5.11	Initial tracking with a closed curve against a cluttered background.	56
5.12	Indication of the influence of α on the new regulator.	58
	(a) $\alpha = 0.02$	58
	(b) $\alpha = 0.05$	58
	(c) $\alpha = 0.1$	58
5.13	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking against a cluttered background and limiting scaling and rotation.	61
5.14	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with an open curve against a cluttered background; and limiting scaling and rotation.	61
5.15	Poorer tracking results with increased deformation between frames.	61
5.16	More and more features are located on the background.	62
5.17	With optical flow more features are found on the moving object.	63
	(a) Finding features without optical flow.	63
	(b) Finding features with optical flow.	63
5.18	An illustration of the influence of applying momentum.	63
	(a) Locating features with the initial estimate lying on background edges.	63
	(b) Locating features after shifting the initial estimate.	63
5.19	Random selection of result frames to indicate the performance of the curve fitting algorithm when the deformation between frames becomes quite big. Tracking is done with an open curve, implementing optical flow as well as scale and rotation limiting.	64
5.20	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking the movement of a hand drawing a picture. The Euclidean regulator was used with $\alpha = 0.01$.	65

5.21	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking a hand that moves against a realistic background. The Euclidean regulator was used with $\alpha = 0.1$.	66
6.1	The singular values of the simulated keyframes as found by PCA.	68
6.2	The keyframes corresponding to the eight biggest singular values as derived from the simulated data.	69
6.3	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with seven keyframes and an Euclidean regulator ($\alpha = 0.001$). Deformations are rotation and translation.	69
6.4	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking simulated data, against a background, with five keyframes and an Euclidean regulator ($\alpha = 0.1$). Typical deformations are rotation and translation.	70
6.5	Some images of the nonlinear deformation of the hand.	71
6.6	The singular values of the keyframes as found by PCA.	71
6.7	The six keyframes with the biggest singular values as derived from real data.	71
6.8	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking real data with six keyframes and an Euclidean regulator ($\alpha = 0.005$). Typical deformations are rotation and translation.	72
6.9	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking a hand against a background. Six keyframes and an Euclidean regulator ($\alpha = 0.01$) is used. Typical deformations are rotation and translation.	72
6.10	Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking the movement of lips during speech, using keyframes.	73
7.1	The algorithm for the general, linear Kalman filter.	75
7.2	Second order Kalman filter.	79

7.3	Finding and ignoring the outliers.	80
7.4	Random selection of result frames to indicate the performance of the Kalman filter when tracking simulated data against a background.	81
7.5	Edges found in background and on moving object (hand).	82
7.6	Random selection of result frames to compare the performance of the Kalman filter (green) with that of the deterministic filter (red) when applied to a planar affine object that undergoes large deformations.	82
7.7	Random selection of result frames to compare the performance of the Kalman filter (green) with that of the deterministic filter (red) when applied to a real tracking scenario.	83
A.1	The Sobel convolution masks.	87
	(a) G_x	87
	(b) G_y	87
A.2	Application of the two directional edge detectors.	88
	(a) Image of Lenna.	88
	(b) Thresholded G_x	88
	(c) Thresholded G_y	88
A.3	The Gaussian Lowpass Filter.	88
	(a) Plot of the one dimensional Gaussian function.	88
	(b) The Gaussian function in two dimensions.	88
A.4	Applying the Gaussian lowpass filter.	89
	(a) Lenna.	89
	(b) Lenna smoothed.	89
B.1	Illustration of the workings of PCA.	91
C.1	Optimally matching the sample curve points to the reference curve points using dynamic programming.	92
C.2	Allowed travel directions through the grid.	93

Nomenclature

Constants:

k = 0,04 (Harrison's cornerness threshold)

κ = 2 (Search line length factor)

Variables:

α, β, γ Snake variables that control the different energies

μ Trade-off variable for gradient vector flow

α Regularisation variable

Vectors and Tensors:

$\mathbf{c}(s)$ Snake vector

$I(x, y)$ Gray-level image

G_x, G_y Image gradient in the x - and y -direction respectively

$\mathbf{r}(s)$ Spline curve

\mathbf{Q} Vector of control point coordinates, also called weights

$\mathbf{B}(s)$ Vector of B -splines

\mathcal{U} Metric matrix

S_Q Vector space

S_X Shape space

\mathbf{X} Shape space vector

W Shape matrix

$\mathbf{r}_0(s)$ The template curve

\mathbf{Q}_0	The template curve in vector notation
W^+	Pseudo inverse of W
$\mathbf{r}_f(s)$	The feature curve
\mathbf{Q}_f	Vector notation of the feature curve
$\hat{\mathbf{X}}$	The best solution to the fitting problem
$\bar{\mathbf{r}}(s)$	Mean shape or initial estimate
$\bar{\mathbf{X}}$	Shape space vector for the mean shape or initial estimate
\bar{S}	Regularising weight matrix
S_s	Invariant subspace, $S_s \subset S_X$
W_s	The invariant subspace's shape matrix
$d(\mathbf{r}, \mathbf{r}_f)$	The distance measure
$\tilde{\mathbf{X}}$	The Kalman filter's state prediction
K	The Kalman gain
A	The state transition model
P	The state covariance
\mathbf{w}	Process noise

Chapter 1

Introduction

Computer vision is the development of computational methods to process and understand digital images. It typically requires a combination of low level image processing to enhance the image quality (noise removal, edge detection etc.) and higher level contour fitting, pattern recognition and image understanding to recognize features present in the image.

An important subsection of computer vision is the tracking of objects. Tracking refers to the process of analyzing and following shape deformations as the object propagates over time in a video sequence. It can also be seen as the ability to locate a certain object in an image, allowing certain deformations of the moving object to occur. Tracking is widely used in applications such as video surveillance (Collins *et al.* (2000); Cohen & Medioni (1999)), speech reading (Barnard *et al.* (2002); Wojdel (2003); Kaucic *et al.* (1996)), some medical applications (Wang *et al.* (2004a); Abolmaesumi *et al.* (2000)), animation (Essa & Basu (1996); Agarwala *et al.* (2004)), sports (Pera & Kovacic (2000)) etc.

There exist two types of tracking algorithms: deterministic and probabilistic. Generally, deterministic tracking is done through feature finding and contour fitting, which will also be the case here. Contours provide valuable information about an object, such as its shape, size, orientation and position.

The deterministic methods sometimes form the basis for more sophisticated probabilistic tracking algorithms that use probability functions, combined with state transition models and measurements to estimate the next state of the fil-

ter. Popular examples of probabilistic filters are the Kalman (Maybeck (1979)) and particle filter (Wan & Van Der Merwe (2001)).

The tracking algorithm implemented in this thesis is derived to track objects whose boundaries undergo linear deformations such as rotation, scaling and translation. By extending these algorithms small, nonrigid shape deformations can also be accommodated.

1.1 Objective

The objective is to build a deterministic tracker through the repeated fitting of B-spline curves. These fitted curves have to be robust against unwanted shape deformations.

The curve fitting method implemented is described by Blake & Isard (1998) in their book on active contours. The aim of this thesis is to implement, explain and elaborate on the work done by Blake and Isard in the context of a deterministic tracking problem.

1.2 Background

Much has been written about the tracking of moving objects in a video sequence. Some methods are very specialised and only apply to certain applications, others are more general but their performance usually suffers as a result.

Probably the simplest tracking situation is when a rigid, uniform object is moving against a background that contains no information. For example, a black piece of paper moving against a uniform, white background. By simply searching for the black pixels in the image, one can determine the exact shape and location of it. There are obviously very few environments that exhibit these ideal conditions. Normally the background is noisy and the object deforms in some way.

Another simple solution to determine the position of an object is to use correlation to match a template of the object to the image. This method, unfortunately, only gives an approximate location and does not shed any light on

the object's size, orientation and other deformations. It is also very dependent on the consistency of the object's shape and texture.

Snakes, introduced by Kass *et al.* (1987), is an energy minimising curve that is attracted to image features, such as boundaries, where its energy functional is minimised. The curve's curvature and elasticity also plays a role in the minimisation. No model or shape is imposed on the fitted curve, and therefore, the snake can get confused between background features and object features. In this thesis the snake is implemented to learn the initial shape and location of the moving object.

Blockmatching is introduced by Chan *et al.* (1990) to determine the correspondence among local image patterns in a image sequence. The region surrounding a control point of the previous fitted curve is investigated to find the area that has the highest measure of similarity. The control point is then moved there. Again no model or shape constraint is enforced. This method is computationally very expensive.

Another way to deterministically track a moving object would be to find localised image features and then to fit a curve through them. In literature today there exist numerous curve fitting methods that apply to images. They can roughly be organised into two categories:

- Methods that simply fit a curve through the control points. These include interpolation techniques, spline curves and subdivision. Note that these methods can be interpolating or approximating.
- Methods that fit a curve according to some basic shape.

This thesis will be based on the latter, focusing mainly on the fitting of B-spline curves as described by Blake & Isard (1998). The curve is fitted according to a template and its variability can be governed. This is done by representing the fitted curve in a subspace (called shape space) that allows only certain deformations to occur. A mean shape is also defined towards which the fitted curve is biased.

Other curve fitting methods that include a model or shape constraint are Yuille *et al.* (1992) who, for example, build a model for the human eye consisting of a parameterised circle bounded by two parabolas. This template is

deformed to fit the image by optimising a cost function based on morphological features. A completely new solution, however, is required for every application.

Similarly Parl *et al.* (1996) models the three dimensional surface of a moving object with a family of parameterised deformable expressions. Limited deformation from the initial shape is allowed through the introduction of mass, damping and a deformation strain energy.

Another way to reduce shape variability is to use Fourier descriptors (Staib & Duncan (1992)) to represent the object's shape. Smoothing is automatically obtained by truncating the higher order harmonics. Fourier descriptors, however, cannot easily represent open curves.

Along the same line Wang *et al.* (2004b) describe a method of fitting B-spline curves to a 'cloud' of features according to some target shape by trying to minimize a special error function called the squared distance minimisation function. The fitting is iterated until the error becomes smaller than a certain limit.

1.3 Literature Overview

The work presented here can be grouped into two areas of interest: First is the segmentation of the moving object. This is done to get a starting or reference point for the tracker in terms of shape (called the template), position, orientation and scale. Segmentation is achieved through the application of the snake algorithm by Kass *et al.* (1987). The snake is implemented according to the greedy algorithm found in Trucco & Verri (1998) and explained by Bebis (2003). The method, however, had to be optimised and gradient vector flow (Xu & Prince (1998)), amongst others, is included.

Second is the repeated fitting of the B-spline curve through the control points according to a certain shape (obtained from the snake algorithm). Blake & Isard (1998) proposes a method that restricts certain deformations of the given shape via the projection of the curve to a subspace that allows only the required deformations. Biasing towards a mean shape is also included in their work. The reparameterisation method added to the curve fitting algorithm implements dynamic programming (Deller *et al.* (2000), Cormen *et al.* (1992) and Nel (2003)).

The above curve fitting method is based on spline notation, norms and moments as described in Blake & Isard (1998) or by Herbst & Fornberg (2003).

During the testing of the tracking algorithm it became obvious that some idea of the direction travelled in is necessary for the tracker to keep up with the moving object. As a result optical flow was included (Trucco & Verri (1998) and Horn & Schunck (1981)).

Finally, the performance of the deterministic tracker is compared to that of the Kalman filter. The Kalman filter is a probabilistic method that also tracks the deformation of B-spline fitted curves, represented in shape space. The implementation of the filter is again based on the work by Blake & Isard (1998); and also Blake *et al.* (1995) and Ramamoorthy *et al.* (2003).

1.4 Contribution

This work shows that the tracking of a moving object in the image plane can be done deterministically with great success. Especially if the deformation of the moving object's bounding curve is linear and the movement between consecutive frames is small. Nonlinear deformations can also be tracked, but requires the careful selection of parameters.

Deterministic tracking has the benefit of being much less complex than the stochastic filters. It is, however, not as robust as the stochastic trackers when applied to more noisy environments.

1.5 Overview of Chapters

The organisation of the thesis is as follows:

Chapter 2 looks at the theory behind the snakes algorithm. Ways of optimising the snake's performance are discussed. These include edge detection, finding the corners in the image and forcing the snake to move into concave areas.

In Chapter 3 the notation for B-splines is given. The derivation of the B-spline moments and norms are also covered.

The most important part of the theory is contained in Chapter 4. The focus here is on vector subspaces and the curve fitting algorithm itself.

Tracking results for rigid and nonrigid deformations are given in Chapters 5 and 6, respectively. These two chapters also include methods for handling difficulties that arised during testing.

In Chapter 7 the Kalman filter is implemented and some comparative results of the two filters are given.

The thesis concludes in Chapter 8 with an overview of the method followed and a discussion of the results.

Note that a compact disc with some of the results, in .mpg format, is included in Appendix E.

Chapter 2

Snakes

The first step in tracking any object is to find the moving object's bounding curve. Low-level feature detection processes can be used to find boundaries, but because they cannot find entire geometric structures, they are effective only up to a point. Therefore, the snake model is used for segmentation, as described by Kass *et al.* (1987)

In literature today there are two types of snakes: parametric and geometric. The focus will be on parametric snakes.

A parametric snake is a curve, $\mathbf{c}(s) = [x(s), y(s)]$, where s is a parameter that increases as the curve is traversed. The user places the snake on the image, close to the object it has to track. An algorithm is then applied to the snake to make it move towards the edges in the image. This algorithm, called the Greedy algorithm (to be discussed later) adjusts the snake's shape and position in order to minimise an energy functional. The energy functional acquires a minimum when the snake has settled on the edges of the object.

2.1 The Parametric Snake Model

From Bebis (2003), parametric snakes move under control of internal and external forces in order to minimize the energy functional

$$E_{total} = \int_0^1 [E_{int}(\mathbf{c}(s)) + E_{ext}(\mathbf{c}(s))] ds.$$

E_{int} is a combination of all the internal forces and represent snake characteristics such as smoothness and elasticity. All the outside influences on the snake

is derived from the image itself and forms the external energy. Taking this into consideration the energy functional can now be written as

$$E_{total} = \int_0^1 [\alpha E_{elastic}(\mathbf{c}(s)) + \beta E_{curve}(\mathbf{c}(s)) + \gamma E_{image}(\mathbf{c}(s))] ds, \quad (2.1.1)$$

where α , β and γ are the weights for the energy terms. Note that in some instances it is desirable to let the weights vary over the length of the curve. They will then also be functions of s .

$E_{elastic}$ is the elastic energy term that discourages the snake from stretching. The greater the distance between neighbouring control points, the greater is the elastic energy. To minimise this energy, the snake tries to bring all its control points closer together. This is done through the minimisation of the absolute value of the first derivative:

$$E_{elastic} = \left| \frac{d\mathbf{c}}{ds} \right|^2.$$

In the discrete case the snake is represented by N control points $\mathbf{c}(s_i)$, $i = 1, \dots, N$. The first derivative is now approximated by a normalised finite difference

$$\begin{aligned} E_{elastic} &= |\mathbf{c}(s_i) - \mathbf{c}(s_{i-1})|^2 \quad \text{or} \\ &= (x_i - x_{i-1})^2 + (y_i - y_{i-1})^2. \end{aligned}$$

Because this energy term causes the snake to shrink, a better form of $E_{elastic}$ would be

$$E_{elastic} = (\bar{d} - |\mathbf{c}(s_i) - \mathbf{c}(s_{i-1})|)^2,$$

where \bar{d} is the average distance between the points. This new $E_{elastic}$ tries to keep the points at equal distances.

The smoothness term is given by E_{curve} . Smoothness is enforced by avoiding oscillations of the snake. This is done by penalizing high contour curvatures, or rather, by minimizing the absolute value of the second derivative:

$$E_{curve} = \left| \frac{d^2\mathbf{c}}{ds^2} \right|^2.$$

For discrete values the curvature can be approximated by the following normalised finite difference:

$$\begin{aligned} E_{curve} &= |\mathbf{c}(s_{i-1}) - 2\mathbf{c}(s_i) + \mathbf{c}(s_{i+1})|^2 \quad \text{or} \\ &= (x_{i-1} - 2x_i + x_{i+1})^2 + (y_{i-1} - 2y_i + y_{i+1})^2. \end{aligned}$$

E_{image} is the only external energy term and is derived from the image itself. This energy term takes on smaller values at the features of interest, such as boundaries, attracting the snake. There are various ways to calculate this energy but the absolute value of the gradient of the gray level image,

$$E_{image} = -|\nabla I(x, y)|^2,$$

is mostly used, where $I(x, y)$ represents the image intensity at the position (x, y) . The gradient will typically have large values at the edges, but the objective is to minimise the energy functional at boundaries, attracting the snake. Therefore, the negative of the gradient is taken as E_{image} .

2.1.1 The Greedy Algorithm

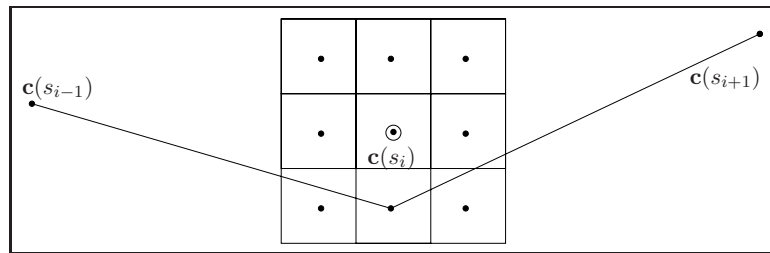


Figure 2.1: The Greedy Algorithm: locating a local minimum point.

A popular way of implementing the snake is through the greedy algorithm. This algorithm makes local optimal choices, hoping that the result would be a global minimum. The input to the algorithm is the image and the initial control points, $\mathbf{c}(s_i)$, $i = 1, \dots, N$. Normally the user would provide these starting points for the snake through a graphical user interface. The algorithm consists of the following steps:

1. The $M \times M$ neighbourhood surrounding every control point, $\mathbf{c}(s_i)$, is investigated, looking for the location that minimizes the energy functional, as shown in Figure 2.1. Move $\mathbf{c}(s_i)$ there.
2. Update the value of \bar{d} , by determining the average distance between the new control points.

Recursion of this process will cause the snake to shrink, stretch and slide. The snake is optimally fitted when only a small fraction of points move in an iteration.

In this tracking application the snake algorithm was used only in the first frame of the videostream, to find the boundary of the object to be tracked. This boundary is called the template.

2.2 The Canny Edge Detector

As stated earlier, there are numerous ways to determine E_{image} . The most common is to use an edge detector to find the boundaries in the image. This will give the required local minimum at the features of interest. It is, however, very important to find these boundaries as accurately as possible. One reason being that the boundary segmented by the snake will be used as the template in the tracking algorithm. (The template is at the basis of the tracking algorithm.) Another reason is that the tracking algorithm makes extensive use of edge detection while processing frames.

The Canny edge detector was developed by John F. Canny in 1986, (Canny, 1986). Canny's intentions were to enhance the performance of the many edge detectors already available. He followed a list of criteria to improve the then-current methods:

- Low error rate – edges in images should not be missed, and there should not be any responses to non-edges.
- The edge points should be well localized – the distance between the actual and found edge should be a minimum.
- There can be only one response to an edge.

The Canny filter is a multi-stage algorithm, consisting of the following steps:

Step 1 Filter out any noise in the image by smoothing it with a Gaussian lowpass filter. This will keep erroneous edges from being detected. For more information on the Gaussian lowpass filter, see Appendix A.2.

Step 2 Find the edge strength at every pixel by taking the gradient of the image. The Sobel filter is used to highlight regions with high spatial derivatives

$$G = \nabla I(x, y) = \begin{bmatrix} G_x \\ G_y \end{bmatrix}.^1 \quad (2.2.1)$$

The filter output is two images which contains the gradient in the x- and y directions, G_x and G_y respectively. The edge strength is then approximated by $|G| = |G_x| + |G_y|$. See Appendix A.1 for more detail on the Sobel filter.

Step 3 Determine the edge direction with $\theta = \arctan \frac{G_y}{G_x}$.

Step 4 Relate the edge direction to a direction that can be traced in an image. The possible directions surrounding a pixel are shown in Figure 2.2

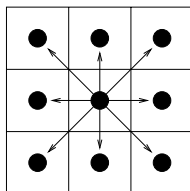


Figure 2.2: The eight possible edge directions surrounding a pixel.

Step 5 Every pixel has an edge direction. The algorithm starts by traversing the edge direction of the first pixel in the image, suppressing any pixel that is not considered to be an edge. It compares the magnitude of the current pixel's edge strength, Q , to both adjacent pixels' values, P and R , as in Figure 2.3(b). If it is less than any of the two adjacent values, it will be made a non-edge. Supposing Q is an edge, the next pixel visited will be adjacent to Q in a direction perpendicular to the edge direction. From Figure 2.3(c) it is clear that it will be either T or S . This will give a thin line in the output image.

¹Some sources use I_x and I_y instead, but $G_x = I_x$ and $G_y = I_y$.

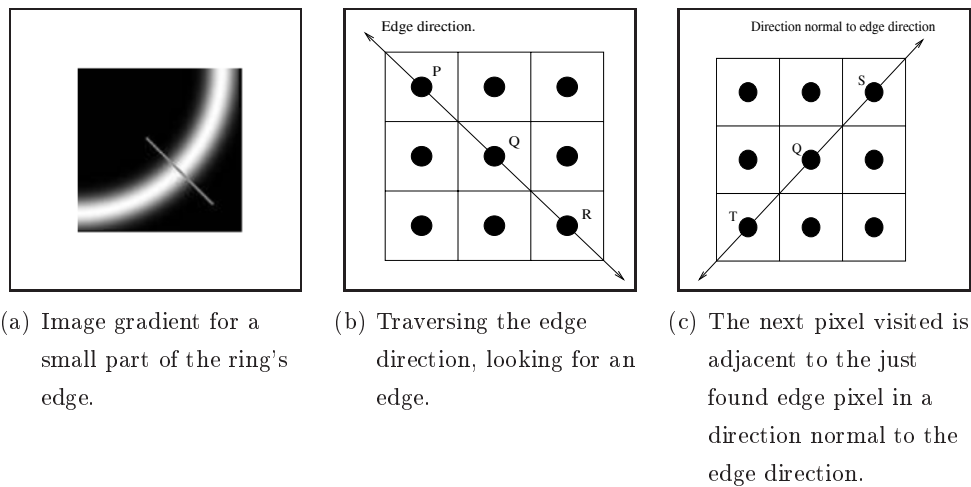


Figure 2.3: Step 5 of the Canny edge detector.

Step 6 To prevent an edge from looking like a dashed line, hysteresis is used with a high and a low threshold, T_1 and T_2 respectively ($T_1 > T_2$). Every pixel with a value higher than T_1 is presumed to be an edge pixel. Any pixels connected to this pixel with a value greater than T_2 are also selected as edge pixels.

The Canny edge detector is a very popular edge detector. The performance is in a way, however, user bound: the better the selection of the thresholds, the better the performance. Also, for different applications there can be different threshold values.

2.3 Corners

2.3.1 Snakes and Corners

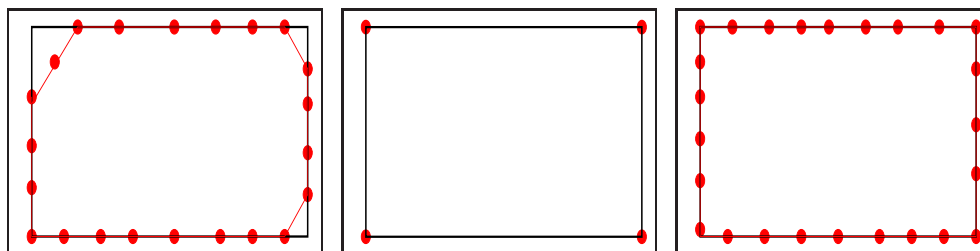
Snakes have a tendency to not always fit nicely around a corner as it sometimes cuts them. This can be attributed to the influence of the snake's internal energies.

In this application the snake is initialized by placing the contour points around the outside of the object to be segmented, making it desirable to build in a little shrinking to let the snake move inwards, towards the object. (This

is done by making \bar{d} a bit smaller than the average.) If, for example, the snake is then to be fitted around a rectangle, and the shrinking forces are big enough, $E_{elastic}$ can make the snake pull inwards, away from the corners. See Figure 2.4.

The smoothness term, E_{curve} , penalizes high contour curvatures. It, therefore, directly influences the snake's accuracy when it comes to corners.

A further problem is the fact that the snake is discrete. If a control point is not located directly at the corner, but there are control points on opposite sides of it, the corner will also be cut. This can be seen in Figure 2.4(a).



(a) Snake convergence without found corners
 (b) The corners that were found.
 (c) The snake result with the found corners now implemented.

Figure 2.4: The influence of finding the corners during preprocessing.

2.3.2 Finding the Corners

To help the snake with handling corners a part of the Kanade-Lucas-Tomasi (KLT) tracking algorithm is implemented to find the corners in the image. This method is described by Wagener & Herbst (2002). For every pixel in the image a window of size W is considered. The gradient of this windowed image is determined in the x- and y directions, by (2.2.1):

$$G = \begin{bmatrix} G_x \\ G_y \end{bmatrix}.$$

Now consider the product

$$\begin{aligned} GG^T &= \begin{bmatrix} G_x \\ G_y \end{bmatrix} [G_x G_y] \\ &= \begin{bmatrix} G_x^2 & G_x G_y \\ G_x G_y & G_y^2 \end{bmatrix}. \end{aligned}$$

Integrating the matrix just derived, we get

$$Z = \iint_W \begin{bmatrix} G_x^2 & G_x G_y \\ G_x G_y & G_y^2 \end{bmatrix} \omega(x, y) d\mathbf{x},$$

with $\mathbf{x} = (x, y)$ and ω a weighting function that attenuates pixel values at the ends of the window. Note that in practice the integrations will be replaced with summations. By analyzing the eigenvalues of Z , the existence of a corner can be determined. If Z has two large eigenvalues, the area, W , contains a strong feature. This feature can be a corner, salt-and-pepper texture or any other prominent texture pattern.

When are the eigenvalues big enough? The Harris ‘cornerness’ function,

$$R = \det Z - k(\text{trace} Z)^2,$$

is a criterion for deciding when a corner was detected. The local maxima, of all the R -values computed, will give a clear indication of the existence of a corner in the corresponding area W . The value of k is taken as 0.04 as suggested by Harris.

To implement the corners found into the greedy algorithm, the E_{curve} weight, β , becomes a function of the parameterization: $\beta_i, i = 1, \dots, N$. All the β values are initially set to one. If a control point encounters a corner, its corresponding β value would be set to zero, making E_{curve} a minimum at that point.

2.4 Handling Concavities

Snakes have poor convergence at concave areas. In Figure 2.5(a) it can be seen that the snake tends not to progress into concavities. One reason is that although the external forces, E_{image} , correctly point towards the boundary,

inside the concavity these forces point in directly opposite directions, as can be seen in Figure 2.5(b). Thus, they pull the snake towards the two side edges, but do not make the snake move down, towards the bottom of the concave area.

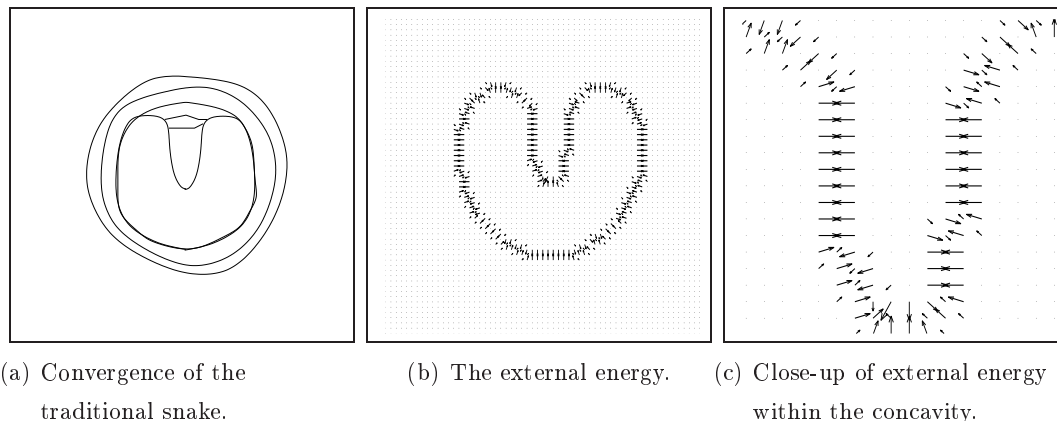


Figure 2.5: The influence of external forces on the convergence of the traditional snake.

Another problem is that the snake has a limited capture range: an $M \times M$ area around every control point. Because the magnitude of the external forces dies out quite rapidly as one moves away from the object boundary, the snake often does not see the edges. See Figure 2.5(c).

A possible solution can be to use the Euclidean Distance Transform (Cohen & Cohen (1993)), where every pixel gets assigned the distance to the nearest edge. This method greatly improves the snake's capture range, as it can now 'see' an edge that is quite a long way off. It, however, does not improve behaviour in the concave areas. The distances to the sides of the concavity are much smaller than that to the turning point. So, again the snake gets pulled apart to the closest edges, but is not forced to move into the concave region.

Several other solutions have been proposed such as pressure forces (Tek & Kimia (1995)), multiresolution methods (Leroy *et al.* (1996)) and using solenoidal external fields (Prince & Xu (1996)). They all solve one or both problems, but create new difficulties.

2.4.1 Gradient Vector Flow

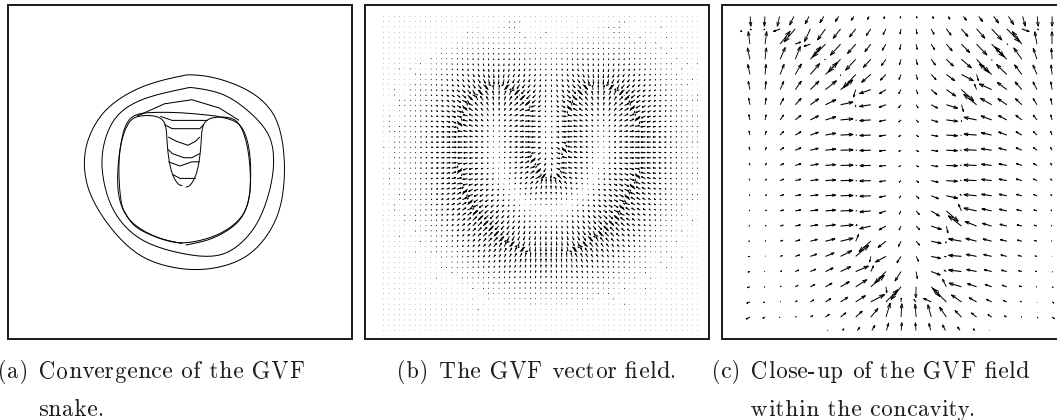


Figure 2.6: The influence of external forces on the convergence of the gradient vector flow snake.

To solve the above mentioned problems Xu & Prince (1998) introduced a concept called gradient vector flow, GVF. The gradient vector flow field is defined as the vector field, $\mathbf{v} = [u(x, y), v(x, y)]$, that minimizes the energy functional

$$\xi = \iint \mu (u_x^2 + u_y^2 + v_x^2 + v_y^2) + |\nabla I|^2 |\mathbf{v} - \nabla I|^2 dx dy,$$

where μ is a biasing constant and the integration is taken over the whole image. From the equation above it follows that when $|\nabla I|$ is small, the energy is dominated by the sum of the partial derivatives, yielding a slowly varying field. When $|\nabla I|$ is large, the second term dominates, and is minimized by setting $\mathbf{v} = |\nabla I|$. The effect is that \mathbf{v} is kept nearly equal to the gradient of the edge map when it is large, but varies slowly in homogeneous areas. The parameter μ governs the tradeoff between the first and second terms of the integrand. Its value is determined by the amount of noise in the system: the more noise, the larger μ should be.

Using the calculus of variations (Hall & Porsching (1990)), it can be shown that the GVF can be found by solving the following Euler equations:

$$\mu \nabla^2 u - (u - G_x) (G_x^2 + G_y^2) = 0 \quad (2.4.1)$$

$$\mu \nabla^2 v - (v - G_x) (G_x^2 + G_y^2) = 0 \quad (2.4.2)$$

where ∇^2 is the Laplace operator. Note that in homogeneous regions the second term of both equations is zero because the gradient of $I(x, y)$ is zero. Within these regions u and v are determined by Laplace's equation. The resulting GVF field is interpolated from the region's boundary, suggesting a kind of competition among the boundary vectors. The vectors, therefore, point into the concavities as can be seen in Figure 2.6(c).

Equations (2.4.1) and (2.4.2) can be solved by treating u and v as functions of time and solving:

$$\begin{aligned} u_t(x, y, t) &= \mu \nabla^2 u(x, y, t) - (u(x, y, t) - G_x(x, y)) \cdot (G_x(x, y)^2 + G_y(x, y)^2) \\ v_t(x, y, t) &= \mu \nabla^2 v(x, y, t) - (v(x, y, t) - G_x(x, y)) \cdot (G_x(x, y)^2 + G_y(x, y)^2) \end{aligned}$$

The steady-state solution ($t \rightarrow \infty$) of these linear parabolic equations is the desired solution of (2.4.1) and (2.4.2). The variables u and v is initialised by determining the image gradient in the x- and y directions, respectively. Note that this is the normal gradient - the Sobel filter was not implemented here.

After the computation of $\mathbf{v}(x, y)$, the external energy, E_{image} , in (2.1.1) is replaced by $\mathbf{v}(x, y)$, or rather E_{gvf} to obtain

$$E_{total} = \int_0^1 \alpha E_{cont}(c(s)) + \beta(s) E_{curve}(c(s)) + \gamma E_{gvf}(c(s)) ds.$$

The rest is solved as usual: discretisation and iteration.

The result of implementing gradient vector flow is shown in Figure 2.6. The snake now moves into concavities, because the vectors' direction is slightly into the concave areas. The range of the external forces has also increased.

2.5 An Example

In this application snakes was implemented only to segment the object in the first frame of the video sequence, thereafter a repeated curve fitting algorithm

was used to track the objects' movement. This does not mean that snakes cannot be deployed elsewhere too.

In order to do facial recognition, one seldom gets images that contain only face information. The majority of databases have some kind of background in the image that has to be handled in some way. This can be done by either including the background in the model or by discarding it, that is, separating the faces from the background. Snakes can be used to find the boundaries of the faces, making the separation from the background easy. Figure 2.7 shows an example of where a snake was used to segment the facial information from the background information.

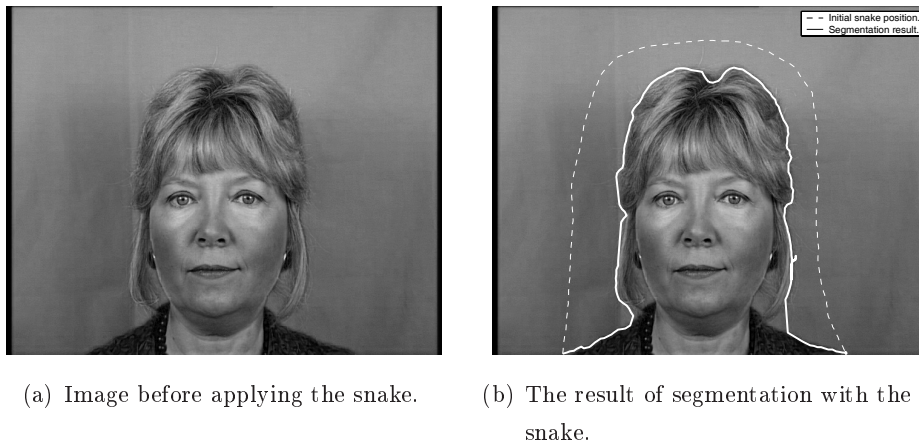


Figure 2.7: Using snakes to segment faces from the background.

2.6 Summary

Snakes are used to segment the boundaries of objects. The movement of the snake is determined by the evaluation of an energy functional that has to be minimised.

The performance of the snake is enhanced by:

- implementing the Canny edge detector, as it enables the accurate detection of the image edges;
- finding the corners so that the boundary is not unnecessarily distorted;
- replacement of E_{image} with E_{gvf} to ensure that the snake propagates into concavities.

Chapter 3

Spline Curves

This chapter is a short summary on B-splines as explained by Blake & Isard (1998) and Herbst & Fornberg (2003). B-splines is particularly important as the whole tracking algorithm represents its curves in a B-splines basis.

A parametric spline is a curve that consists of a set of control points, $\mathbf{r}(s) = (x(s), y(s))$, where s is a parameter that increases as the the curve is traversed. The spline functions, $x(s)$ and $y(s)$, have a special smoothness quality. They form smooth piecewise polynomial curves that are built from concatenated polynomial segments (spans), each of which can be of some polynomial order d . These spans are smoothly joined at the control points (also called breakpoints), that is, they have $d-2$ continuous derivatives at the breakpoints. The numbers of spans used by a spline is given by N_B .

Parametric spline curves are capable of representing boundary curves of images very efficiently, as different polynomials may be used in different parts of the curve. Simple shapes can be represented by only a few spans, whereas more complex shapes may need higher order polynomials or more spans. Splines can be divided into two main categories:

- Interpolating splines which pass through the control points.
- Approximating splines which pass near the control points

Interpolating splines are used for piecewise polynomial interpolation. Images contain noise, therefore, the control points will also contain noise. The approximating spline gets rid of this noise by smoothing the control points.

3.1 B-Splines

3.1.1 Interpolating B-Splines

To interpolate a given periodic ¹ function, $f(x)$, at equidistant points $x = i$, $i = 0, \dots, N_B - 1$, a function, $B(x)$, is used to construct the interpolation polynomial

$$p(x) = \sum_{i=0}^{N_B-1} w_i B(x - i).$$

The coefficients or weights, w_i , are solved to satisfy the interpolation condition: $p(i) = f(i)$; and $B(x)$ belongs to a useful set of interpolation functions, called *B-splines*. Therefore, a spline is a weighted sum of $N_B - 1$ basis functions, $B(x)$. Note that the above spline is formed by linear combinations of shifted versions of $B(x)$.

Every basis function consists of d polynomials, each defined over a span of the x -axis. They are recursively constructed, with the first being:

$$B_1(x) = \begin{cases} 1, & x \in [-\frac{1}{2}, \frac{1}{2}) \\ 0, & x \notin [-\frac{1}{2}, \frac{1}{2}) \end{cases}.$$

Higher order splines are then defined as:

$$\begin{aligned} B_d(x) &= \int_{-\frac{1}{2}}^{\frac{1}{2}} B_{d-1}(x-t) dt && \text{or} \\ &= B_{d-1} \star B_1(x), \end{aligned}$$

where \star indicates a convolution. The first four B-splines can be seen in Figure 3.1.

3.1.2 B-Spline Parametric Curves

So far, the interpolation points have been uniformly spaced. For nonuniform spaced points a parametric representation of the periodic curve or function is used.

¹A-periodic functions is discussed later.

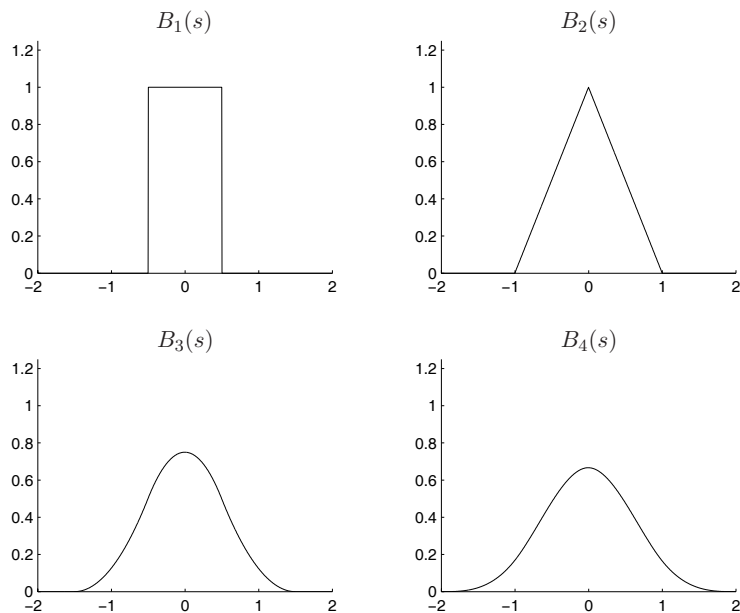


Figure 3.1: The first four B-splines.

To interpolate the following set of control points, $\begin{bmatrix} x_i \\ y_i \end{bmatrix}$, where $i = 0, \dots, N_B - 1$, the vector

$$\mathbf{c}_i = \begin{bmatrix} x_i \\ y_i \end{bmatrix}, \quad i = 0, \dots, N_B - 1 \quad (3.1.1)$$

is formed. This gives rise to the following vector interpolation polynomial:

$$\mathbf{p}(s) = \sum_{i=0}^{N_B-1} \mathbf{w}_i B_d(s - i)$$

with $0 \leq s \leq L$. The weights, \mathbf{w}_i , is obtained by satisfying the interpolation condition, $\mathbf{p}(i) = \mathbf{c}_i$, $0, \dots, N_B - 1$. This equation is a parametric curve drawn in the x-y plane, forced to go through the points \mathbf{p} . Note that for periodic functions $N_B = L$, so that the number of control points equals the number of spans.

3.1.3 Approximating B-Splines

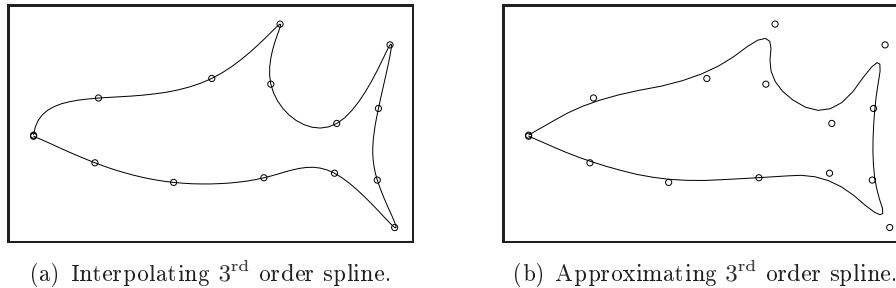


Figure 3.2: The difference between interpolating and approximating splines.

It was already established in the start of the chapter that it is not always desirable to insist on the interpolation condition, because the original values may be contaminated by noise, among other reasons. The data points from equation (3.1.1) are now implemented as control points that determine the shape of the approximating spline, $\mathbf{r}(s)$.

$$\mathbf{r}(s) = \sum_{i=0}^{N_B-1} \mathbf{c}_i B_d(s-i) \quad (3.1.2)$$

Note that the interpolation condition does not hold in this case. Again the result is a parametric curve in the x-y plane, but the curve is a lot smoother, as it is not forced to go through the control points. This is illustrated in Figure 3.2.

The formulation derived so far has been for periodic functions. Non-periodic functions is handled by forcing the spline to go through the end points and stop there. This is done by forming multiple knots at the endpoints. The multiple knots reduce the continuity of the derivatives at that point. If the multiplicity of the knot is m , then the point has $d - m - 1$ continuous derivatives. (Note that $m = 1$ normally.) For a quadratic spline ($d = 3$), for instance, a double knot will let the spline be continuous at that point, but for a triple knot the function becomes discontinuous. The number of spline functions necessary for nonperiodic functions is $N_B = L + d - 1$.

Multiple knots can also be used to force the spline to go through specific control points, i.e interpolate them. For example, let

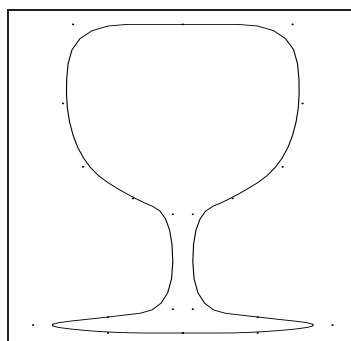
$$\mathbf{c} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ 0 & 2 & 0 & 2 & 0 & 2 \end{bmatrix}.$$

To force a quadratic spline to go through $(3,0)$, the point gets doubled:

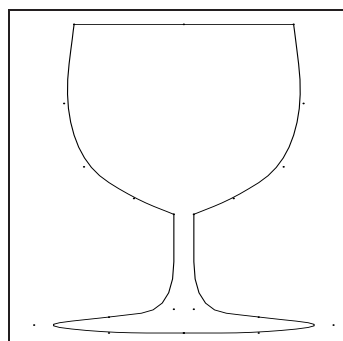
$$\mathbf{c} = \begin{bmatrix} 1 & 2 & 3 & 3 & 4 & 5 & 6 \\ 0 & 2 & 0 & 0 & 2 & 0 & 2 \end{bmatrix}.$$

Figure 3.3 depicts how multiple knots force the spline to pass through them.

One of the uses of splines in this application is to smooth the noise in the snake segmented template. The corners found during the snake preprocessing can be kept by forming multiple knots when the spline is fitted.



(a) Approximating 3rd order spline fitted through the control points.



(b) Approximating 3rd order spline with multiple knots at some control points.

Figure 3.3: The influence of multiple knots.

3.2 Norms and Moments

To conform to the notation used in the remainder of the thesis, (3.1.2) can be rewritten as,

$$\mathbf{r}(s) = \sum_{i=0}^{N_B-1} B_i(s)\mathbf{q}_i \quad 0 \leq s \leq L$$

where $\mathbf{q}_i = (q_i^x, q_i^y)^T$ replaces \mathbf{c}_i to indicate the control points and $B_d(s - i)$ is substituted by $B_i(s)$. This representation is more general: instead of using shifted versions of the same d^{th} order basis function, basis functions of different orders can now be used to describe a span.

3.2.1 Splines in Vector Notation

The vectors containing the control point coordinates are formed by first listing all the x -coordinates, then all the y -coordinates:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{Q}^x \\ \mathbf{Q}^y \end{bmatrix} \quad \text{where} \quad \mathbf{Q}^x = \begin{bmatrix} q_0^x \\ \dots \\ q_{N_B}^x \end{bmatrix}$$

and similarly for \mathbf{Q}^y . The coordinate functions can be written as

$$x(s) = \mathbf{B}(s)^T \mathbf{Q}^x,$$

where $\mathbf{B}(s)$ is a vector of B-spline basis functions. The same can be done for $y(s)$, so that

$$\mathbf{r}(s) = U(s)\mathbf{Q} \quad 0 \leq s \leq L, \quad (3.2.1)$$

where

$$U(s) = \begin{bmatrix} \mathbf{B}(s)^T & \mathbf{0} \\ \mathbf{0} & \mathbf{B}(s)^T \end{bmatrix}. \quad (3.2.2)$$

3.2.2 Norm of Curves

A norm $\|\cdot\|$ for B-spline curves, induced by the Euclidean distance measure in the image plane, is defined as

$$\|\mathbf{Q}\|^2 = \frac{1}{L} \int_0^L |\mathbf{r}(s)|^2 ds$$

or equivalently, from (3.2.1) and (3.2.2),

$$\|\mathbf{Q}\|^2 = \mathbf{Q}^T \mathcal{U} \mathbf{Q},$$

where the metric matrix \mathcal{U} is defined as

$$\mathcal{U} \equiv \frac{1}{L} \int_0^L U(s)^T U(s) ds. \quad (3.2.3)$$

The curve norm is particularly meaningful when distance is used as a means of comparison between two curves

$$\|\mathbf{Q}_1 - \mathbf{Q}_2\|. \quad (3.2.4)$$

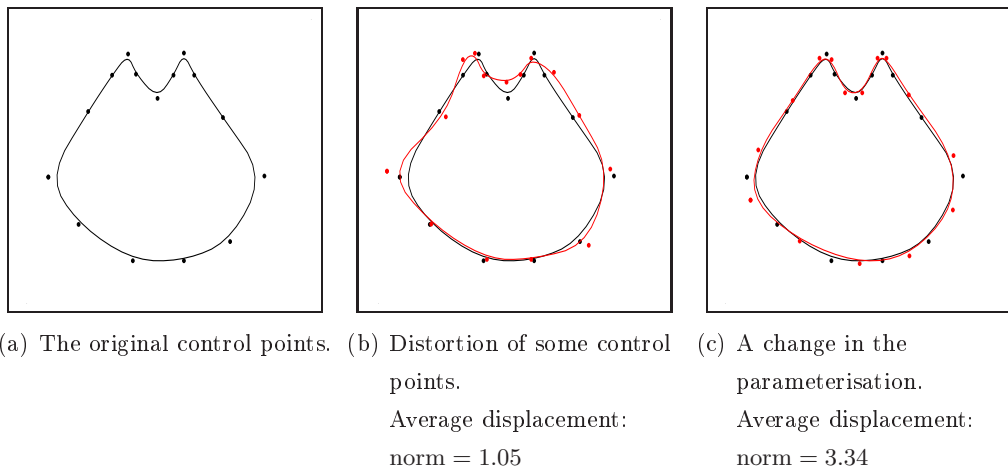


Figure 3.4: The influence of parameterisation on the norm or average distance between curves.

Note that the curve norm as mentioned above is not invariant to the parameterisation. For example, the curve

$$\mathbf{r}^*(s) = \mathbf{r}(1 - s) \quad 0 \leq s \leq 1$$

will have exactly the same shape as $\mathbf{r}(s)$, with the only difference between them the reversal of the parameterisation. Ideally the shape difference must be invariant to the various parameterisations so that $\mathbf{r}^* - \mathbf{r}$ would have a norm of zero. However, this is not the case:

$$\|\mathbf{r}^*(s) - \mathbf{r}(s)\|^2 = \frac{1}{L} \int |\mathbf{r}(1 - s) - \mathbf{r}(s)|^2 ds \neq 0,$$

as can be seen in the example in Figure 3.4. In this figure the black indicates the original control points with its fitted spline, and the red the influence of distortion and change in parameterisation.

A solution to the parameterisation problem could be to search over all possible reparameterisations to obtain the one that will minimise the norm difference

$$\min_g \|\mathbf{r}(s) - \mathbf{r}^*(g(s))\|,$$

where g is the reparameterisation function. This method is computationally very costly. A more economical approach is to use a distance measure $d(\mathbf{r}, \mathbf{r}^*)$ that is not affected by minor reparameterisations of the curve \mathbf{r} . More on this in the next chapter.

3.2.3 Moments

Generally moments (Blake & Isard (1998)) have two roles in active contours. The first is to move the spline template sufficiently close to the tracked object in order to lock onto it. The second role is to determine the position and orientation of the tracked object.

The zeroth moment gives the area, the first moment yields the centroid and the second one represents the inertia.

Area: The simplest available parameterisation-invariant measure for a closed curve is the area

$$\int |\mathbf{r}(s), \mathbf{r}'(s)| ds$$

where $\mathbf{r}'(s)$ is the derivative of $\mathbf{r}(s)$ and $|\mathbf{a}, \mathbf{b}|$ denotes the determinant of the matrix whose columns are \mathbf{a}, \mathbf{b} . This is expressible as a quadratic form in \mathbf{Q} :

$$A(\mathbf{Q}) = \mathbf{Q}^T \mathcal{A} \mathbf{Q}, \quad (3.2.5)$$

where

$$\mathcal{A} \equiv \begin{bmatrix} \mathcal{B}' & 0 \\ 0 & -\mathcal{B}' \end{bmatrix} \quad \text{and} \quad \mathcal{B}' = \frac{1}{L} \int_0^L \mathbf{B}(s) \mathbf{B}'^T(s) ds.$$

Centroid: The conventional definition for the centroid of a curve is

$$\bar{\mathbf{r}} = \frac{1}{L} \int_0^L \mathbf{r}(s) ds.$$

This definition has the drawback that it is not invariant to reparameterisation of the curve, because s is not generally the true arc length. The length of

an infinitesimal segment of curve is not ds but $|\mathbf{r}'(s)| ds$, so that an invariant centroid would be:

$$\bar{\mathbf{r}} = \frac{\int_0^L \mathbf{r}(s) |\mathbf{r}'(s)| ds}{\int_0^L |\mathbf{r}'(s)| ds}$$

The centroid of an area enclosed by a closed B-spline curve, which is invariant to reparameterisations, can be given in terms of the spline vector, \mathbf{Q} :

$$\bar{\mathbf{r}} = \frac{1}{A(\mathbf{Q})} \int_0^L |\mathbf{r}(s), \mathbf{r}'(s)| \mathbf{r}(s) ds. \quad (3.2.6)$$

Inertia: The second moment is given by

$$\mathcal{I} = \frac{1}{A(\mathbf{Q})} \int_0^L |\mathbf{r}(s), \mathbf{r}'(s)| \mathbf{r}(s) \mathbf{r}(s)^T ds, \quad (3.2.7)$$

which is also invariant to reparameterisation.

3.3 Summary

There are two kinds of splines: Interpolating and approximating.

B-splines are used to smooth the boundary found by the snake, hoping to get rid of noise. Therefore, an approximating spline was fitted through the snake's control points.

To force the approximating spline to go through a certain control point, a multiple knot is formed.

The norm difference is a good estimate of the Euclidean distance between curves, but it depends on the parameterisation.

Chapter 4

Curve Fitting

So far, the smoothed outline of the object to be tracked, called the template, has been found as accurately as possible. The next step is to track this object as it is propagating through the image. The general idea is to search the area surrounding the last known location of the object, trying to find its new position. To maintain the basic template shape throughout the propagation is a very difficult task, as the description of the template is in a vector space of B-splines. The dimension of this vector space, \mathcal{S}_Q , is $N_Q = 2N_B$, where N_B is the number of control points. Every one of these N_B points can move horizontally or vertically, meaning that the template has N_Q degrees of freedom. That is, it can deform in N_Q ways; and basically any N_B points in the neighbourhood can be fitted by the spline and wrongly be identified as the moved object, see Figure 4.1.

To solve this problem the allowable deformations of the template is restricted to belong to a so-called shape space, \mathcal{S}_X . The work done by Blake & Isard (1998) is referenced here to explain how the shape space limits the variability of the object.

4.1 Shape Space

The shape space is constructed from an underlying linear vector space and has dimension N_X , much smaller than N_Q . It restricts the movement of the control points to a lower dimension, thereby allowing only specific deformations of the

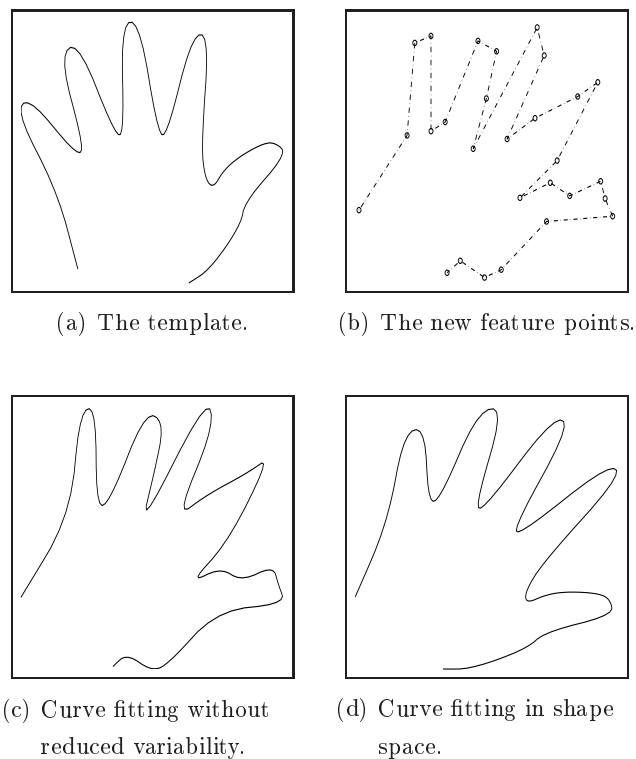


Figure 4.1: The influence of shape space when fitting curves.

template.

The linearity requirement is to simplify the curve-fitting and tracking algorithms. It works well for rigid objects and simpler nonrigid ones.

4.1.1 Definition

Shape space is a linear mapping of a shape space vector, $\mathbf{X} \in \mathbb{R}^{N_X}$, to a spline vector, $\mathbf{Q} \in \mathbb{R}^{N_Q}$,

$$\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0, \quad (4.1.1)$$

with W a $N_Q \times N_X$ shape matrix, describing the allowable variations. \mathbf{Q}_0 is the template curve against which shape variations are measured. If one restricts \mathbf{X} , it should be clear that the deformations of template is also restricted. Note that \mathbf{Q}_0 is the vector representation of the template curve, $\mathbf{r}_0(s)$, found by the snake algorithm, with its centroid shifted to the origin.

4.2 The Space of Euclidean Similarities

The space of Euclidean similarities has four degrees of freedom, of which two are for translation, one is for rotation and the last is for isotropic scaling. This space allows transformations of the kind as when, for example, a camera with a zoom lense looks directly down at a flat object that moves on a table top. The typical deformations are depicted in Figure 4.2.

The Euclidean similarities of a template curve, represented by \mathbf{Q}_0 , form a four dimensional shape space. Its shape matrix is

$$W = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{bmatrix},$$

with $\mathbf{1}$ a vector of ones, and $\mathbf{0}$ a vector of zeros, the same length as \mathbf{Q}_0^x . The first two columns allows horizontal and vertical translation respectively, while the third and the fourth columns govern the rotation and the scaling.

From (4.1.1) the shape space vector, \mathbf{X} , is mapped by the shape matrix to form a vector curve. Therefore,

$$\mathbf{Q} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_0^x \\ \mathbf{Q}_0^y \end{bmatrix}.$$

Some examples of the shape vector is:

1. $\mathbf{X} = (0, 0, 0, 0)^T$, which represents the template curve.
2. $\mathbf{X} = (0, 5, 0, 0)^T$, which represents the template curve moved five unit upwards.
3. $\mathbf{X} = (0, 0, 1, 0)^T$, which represents the template curve doubled in size.
4. $\mathbf{X} = (0, 0, \cos \theta - 1, \sin \theta)^T$, which represents the template curve rotated through an angle θ .

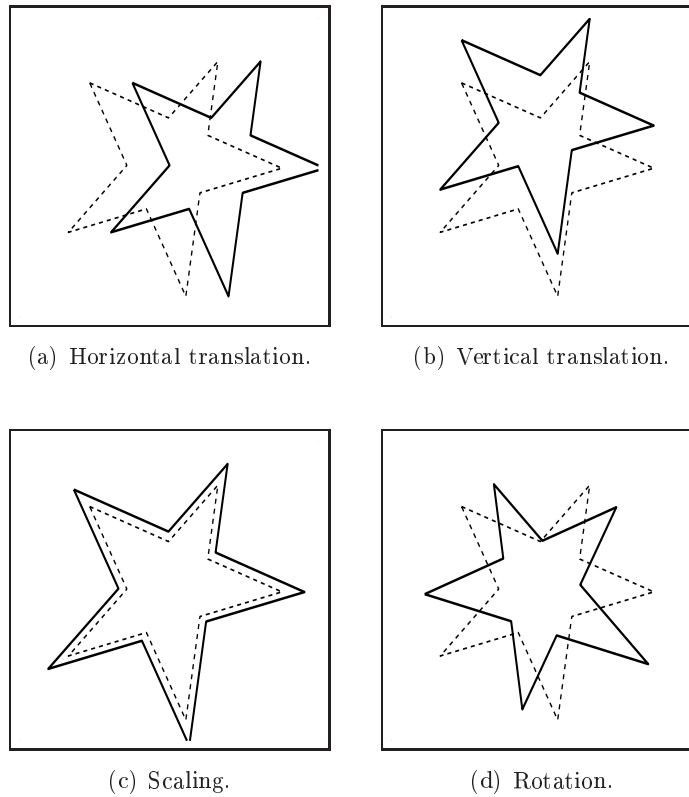


Figure 4.2: The four deformations from the space of Euclidean similarities.

4.3 Planar Affine Shape Space

The planar affine shape space arises when one has a strict two dimensional view of a planar (flat) object that has complete freedom to move in three dimensions, as in Figure 4.3. Its motion has six degrees of freedom, three for translation and three for rotation. The three translational freedoms represent the object's movement in the x , y and z -directions. When viewed in two dimensions, the movement in the z -direction will resemble the scaling of the object, keeping the aspect ratio constant. The rotations are around the x , y and z -axii, where rotation around the x -axis represents vertical scaling and rotation around the y -axis will look like horizontal scaling. So, just six affine degrees of freedom are required to describe all the possible deformations of a planar object's bounding curve, as it moves in three dimensions.

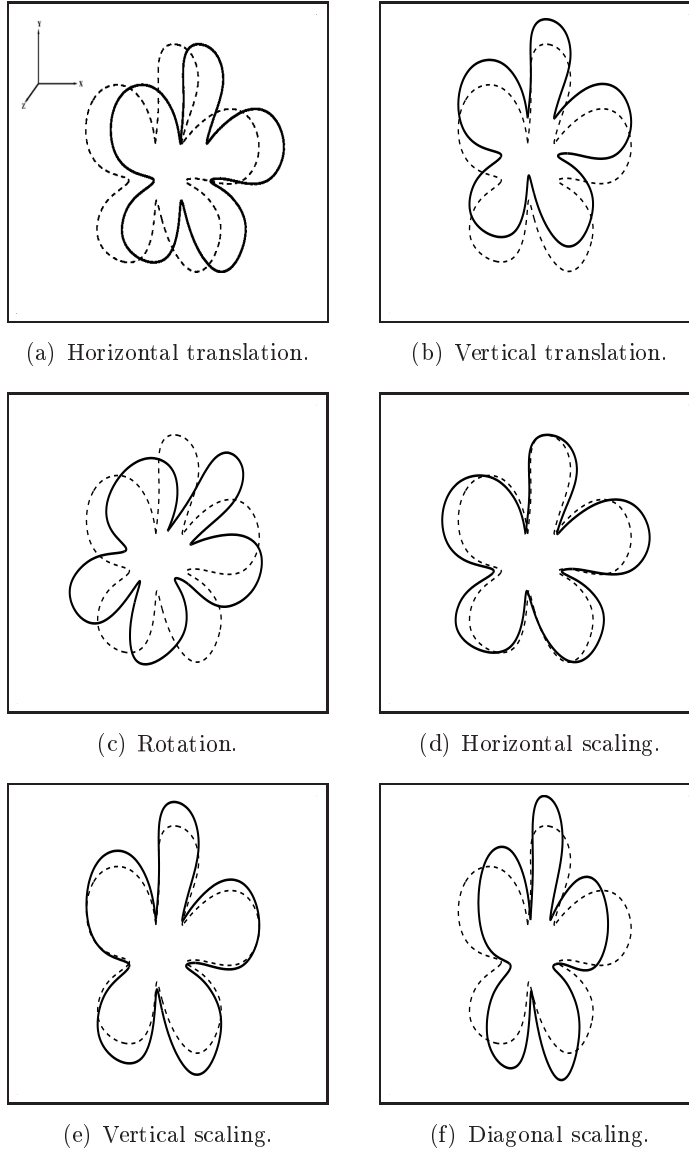


Figure 4.3: The six planar affine deformations.

The planar affine group can be viewed as the class of all linear transformations that can be applied to a template curve, $\mathbf{r}_0(s)$

$$\mathbf{r}(s) = \mathbf{u} + M\mathbf{r}_0(s),$$

and still preserve the parallelism between lines. M is a 2×2 matrix that allows rotation and scaling, and $\mathbf{u} = (u_1, u_2)^T$ is a two dimensional translation vector.

The shape matrix, W , for a planar affine shape space looks as follows:

$$W = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{0} \end{bmatrix}. \quad (4.3.1)$$

From (4.1.1), W maps the shape space vector, \mathbf{X} , with dimension six, to a spline vector of dimension N_Q . The first two columns of W represents horizontal and vertical translation respectively. By linear combinations of the last four columns of W , the remaining four affine motions can be described. The elements of \mathbf{X} act as weights on W to deform \mathbf{Q}_0 ,

$$\mathbf{X} = [u_1, u_2, M_{11} - 1, M_{22} - 1, M_{21}, M_{12}]^T. \quad (4.3.2)$$

From (4.1.1):

$$\mathbf{Q} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & \mathbf{0} & \mathbf{0} & \mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{0} \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \\ M_{11} - 1 \\ M_{22} - 1 \\ M_{21} \\ M_{12} \end{bmatrix} + \begin{bmatrix} \mathbf{Q}_0^x \\ \mathbf{Q}_0^y \end{bmatrix}.$$

Some examples of the allowed deformations are:

1. $\mathbf{X} = (0, 0, 0, 0, 0, 0)^T$: No deformation - the fitted curve is the template.
2. $\mathbf{X} = (1, 5, 0, 0, 0, 0)^T$: The template is translated 1 unit to the right and 5 units up.
3. $\mathbf{X} = (0, 0, 1, 0, 0, 0)^T$: The template has doubled in width.
4. $\mathbf{X} = (0, 0, 1, 1, 0, 0)^T$: The template has doubled in size.
5. $\mathbf{X} = (0, 0, \cos \theta - 1, \cos \theta - 1, -\sin \theta, \sin \theta)^T$: The template rotated through an angle θ .

The planar affine shape space can also be applied to nonplanar objects. The only requirement is that their silhouette view's bounding curve undergoes only planar affine transformations. That is, the bounding curve's shape is not allowed to change, but it can be scaled, translated and rotated.

4.4 Norms and Moments in the Planar Affine Shape Space

4.4.1 Finding the Norm

A formulation of the norm has to be found in terms of the shape space parameter \mathbf{X} . The definition has to be consistent so that

$$\|\mathbf{Q}_1 - \mathbf{Q}_2\| = \|\mathbf{X}_1 - \mathbf{X}_2\|. \quad (4.4.1)$$

From (4.1.1) and the above condition, the norm is defined as

$$\|\mathbf{X}\|^2 = \mathbf{X}^T \mathcal{H} \mathbf{X} \quad \text{with} \quad \mathcal{H} = W^T \mathcal{U} W.$$

The shape space norm has a geometric interpretation, where

$$\|\mathbf{X}\| = \|\mathbf{Q} - \mathbf{Q}_0\| \quad (4.4.2)$$

is the average displacement of the curve represented by the shape vector from the template curve, \mathbf{Q}_0 . A natural mapping from vector space, \mathcal{S}_Q , to shape space, \mathcal{S}_X , can now be defined using a pseudo inverse for W (given that it has full rank)

$$\mathbf{X} = W^+(\mathbf{Q} - \mathbf{Q}_0) \quad \text{where} \quad W^+ = \mathcal{H}^{-1} W^T \mathcal{U}. \quad (4.4.3)$$

4.4.2 Determining the Moments

The computation of moments can be greatly simplified in the affine space, because of the affine transformation matrix, M . The inertia can be computed with

$$\mathcal{I} = M \mathcal{I}_0 M^T, \quad (4.4.4)$$

where \mathcal{I}_0 is the inertia matrix of the template as computed in (3.2.7); and M is described in (4.3.2).

The area is given by

$$A = (\det M) A_0, \quad (4.4.5)$$

where the equation for the area of the template, A_0 , is given in (3.2.5). To determine the factor with which the template are scaled, solve

$$F_{scale} = \sqrt{A/A_0}. \quad (4.4.6)$$

4.5 Keyframes

Affine spaces are sufficient for modelling the movement of rigid bodies. In most cases, however, motion is nonrigid. To handle these nonrigid movements, a shape space has to be defined that will allow all the necessary deformations, but will still reduce the variability (Blake & Isard (1998)). The most effective strategy is to learn the shape space from a training set of sample motion.

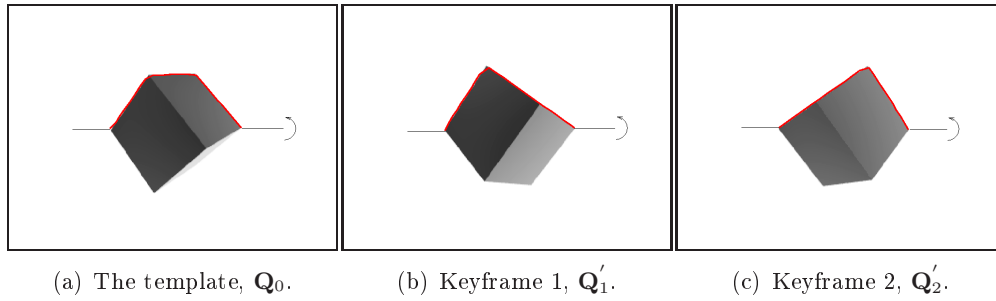


Figure 4.4: The keyframes used to describe the two dimensional deformation of a cube rotating in three dimensions.

In Figure 4.4 the three frames represent some of the two dimensional deformations of a cube rotating around its maximum axis. From the possible deformations of the cube, a few bounding curves, as shown in Figure 4.4(b) and 4.4(c), called keyframes, can be selected to build the shape space. This shape space will allow all possible two dimensional deformations of the cube's outline as it is rotating around the indicated axis.

Let the curve in Figure 4.4(a) act as the template, \mathbf{Q}_0 . To construct a shape space that will be a linear combination of all three frames, subtract the template from the remaining two frames:

$$\begin{bmatrix} \mathbf{Q}_i^x \\ \mathbf{Q}_i^y \end{bmatrix} \equiv \mathbf{Q}_i = \mathbf{Q}'_i - \mathbf{Q}_0.$$

These keyframes are used to construct the shape matrix

$$W = \begin{bmatrix} \mathbf{Q}_1^x & \mathbf{Q}_2^x \\ \mathbf{Q}_1^y & \mathbf{Q}_2^y \end{bmatrix}.$$

For more freedom than mere linear combinations of themselves, the three frames can be used to build a shape space that will allow transformations of, for example, the Euclidean similarity set. The shape matrix will then look as follows:

$$W = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y & \mathbf{Q}_1^x & -\mathbf{Q}_1^y & \mathbf{Q}_2^x & -\mathbf{Q}_2^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x & \mathbf{Q}_1^y & \mathbf{Q}_1^x & \mathbf{Q}_2^y & \mathbf{Q}_2^x \end{bmatrix}. \quad (4.5.1)$$

Different object deformations are represented by the shape vector:

1. $\mathbf{X} = (5, 0, 0, 0, 1, 0, 0, 0)^T$ represents the cube in the same position as in Figure 4.4(b), moved 5 units to the right.
2. $\mathbf{X} = (0, 0, \cos \theta - 1, \sin \theta, 0, 0, \frac{1}{2} \cos \theta, \frac{1}{2} \sin \theta)^T$ is the cube halfway between Figure 4.4(a) and 4.4(c), with the image rotated through an angle θ .

Equation (4.5.1) can be extended to include an arbitrary number of keyframes, with a general space of rigid transformations:

$$W = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^1 & \mathbf{Q}_0^2 & \dots & \mathbf{Q}_0^{N_r} & \mathbf{Q}_1^1 & \mathbf{Q}_1^2 & \dots & \mathbf{Q}_1^{N_r} & \dots & \mathbf{Q}_{N_k}^1 & \mathbf{Q}_{N_k}^2 & \dots & \mathbf{Q}_{N_k}^{N_r} \\ \mathbf{0} & \mathbf{1} & & & & & & & & & & & & & & \end{bmatrix}, \quad (4.5.2)$$

where N_k denotes the number of keyframes used and N_r represents the size of the set of transformations. For example, N_r would be four for transformations of the planar affine set and two for transformations of the Euclidean kind. (The translation is not included.)

With N_k keyframes and N_r degrees of transformational freedom, there are a total of $N_k + N_r$ degrees of freedom in the system. The linear representation as a shape space with a W matrix, has dimension $N_r \times (N_k + 1)$. This leads to a wastage of $N_k(N_r - 1)$ degrees of freedom. With more keyframes, the wastage becomes more severe, and a smaller space is often constructed using techniques such as the principle components analysis. More detail on principle components analysis (PCA) can be found in Chapter 6 and Appendix B.

4.6 The Fitting Algorithm

Suppose the image feature (the tracked object at the new location) is expressed in the form of a spline curve $\mathbf{r}_f(s) = U(s)\mathbf{Q}_f$. The fitting problem is then to find a deformation of template, \mathbf{Q}_0 , so that the resulting curve, \mathbf{Q} , will be as close as possible to the measured feature curve, \mathbf{Q}_f . The fitting problem can be defined as the solution for \mathbf{X} that will minimize the distance between the feature and the curve to be fitted. Using the norm defined in (4.4.1),

$$\min_{\mathbf{X}} \|\mathbf{Q} - \mathbf{Q}_f\|.$$

From (4.1.1), \mathbf{Q} can be substituted with $W\mathbf{X} + \mathbf{Q}_0$. The fitting problem is now to minimize,

$$\min_{\mathbf{X}} \|W\mathbf{X} + \mathbf{Q}_0 - \mathbf{Q}_f\|^2. \quad (4.6.1)$$

Let $\mathbf{Q}' = \mathbf{Q}_f - \mathbf{Q}_0$:

$$\begin{aligned} \|W\mathbf{X} - \mathbf{Q}'\|^2 &= (W\mathbf{X} - \mathbf{Q}')^T U (W\mathbf{X} - \mathbf{Q}') \\ &= \mathbf{X}^T W^T U W \mathbf{X} - \mathbf{X}^T W^T U \mathbf{Q}' - \mathbf{Q}'^T U W \mathbf{X} + \text{const.} \end{aligned}$$

Completing the square yields:

$$\|W\mathbf{X} - \mathbf{Q}'\|^2 = (\mathbf{X} - \hat{\mathbf{X}})^T W^T U W (\mathbf{X} - \hat{\mathbf{X}}) + \text{const}$$

with

$$\begin{aligned} \hat{\mathbf{X}} &= (W^T U W)^{-1} W^T U \mathbf{Q}' \\ &= \mathcal{H}^{-1} W^T U \mathbf{Q}'. \end{aligned}$$

Substituting W^+ from (4.4.3) gives:

$$\hat{\mathbf{X}} = W^+(\mathbf{Q}_f - \mathbf{Q}_0). \quad (4.6.2)$$

The equation in (4.6.1) is thus minimized by setting $\mathbf{X} = \hat{\mathbf{X}}$.

4.6.1 Regularisation

So far, noise has been limited by the approximating spline, and in a way by projection onto shape space. Further tolerance to noise can be achieved by biasing the fitted spline curve, \mathbf{Q} , towards a mean shape or initial estimate, $\bar{\mathbf{r}}(s)$. This initial estimate is created by remembering that the template, \mathbf{Q}_0 , has its centroid at the origin. Any transformation, $\bar{\mathbf{X}}$, that moves the template from the origin, closer to the feature curve, can be called the initial shape estimate, with its corresponding spline curve $\bar{\mathbf{r}}(s)$. The degree of biasing is determined by a regularisation constant α .

In the vector space, the biased fitted curve is the solution of $\mathbf{r}(s)$ that minimizes

$$\min_{\mathbf{r}(s)} \alpha \|\mathbf{r} - \bar{\mathbf{r}}\|^2 + \|\mathbf{r} - \mathbf{r}_f\|^2.$$

The problem can also be expressed in shape space as

$$\min_{\mathbf{X}} \alpha \|\mathbf{X} - \bar{\mathbf{X}}\|^2 + \|\mathbf{Q} - \mathbf{Q}_f\|^2.$$

In order to restrict certain deformations of the fitted curve a more general regulariser, called $\bar{\mathcal{S}}$, is defined.

Suppose it is desirable for $\bar{\mathbf{r}}$ to only influence the shape of the fitted spline, but, for example, not its location or its orientation. This set of disallowed transformations belongs to the subspace of Euclidean similarities. The new regulator, $\bar{\mathcal{S}}$, will give the desired invariance over this subspace S_s , where $S_s \subset S_X$. Therefore, the implementation of $\bar{\mathcal{S}}$ ensures that the mean curve cannot impose any deformations belonging to the subspace S_s on the feature curve. This is achieved by a projection operator E^d where

$$\bar{\mathcal{S}} = \alpha E^{dT} \mathcal{H} E^d.$$

The projection operator is expressed in terms of the shape matrix of the subspace S_s :

$$E^d = I - E^s \quad \text{where} \quad E^s = W^+ W_s W_s^+ W.$$

Example: If the subspace S_s belongs to the set of Euclidean similarities, the shape matrix, W_s , will look as follows:

$$W_s = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{Q}_0^x & -\mathbf{Q}_0^y \\ \mathbf{0} & \mathbf{1} & \mathbf{Q}_0^y & \mathbf{Q}_0^x \end{bmatrix}.$$

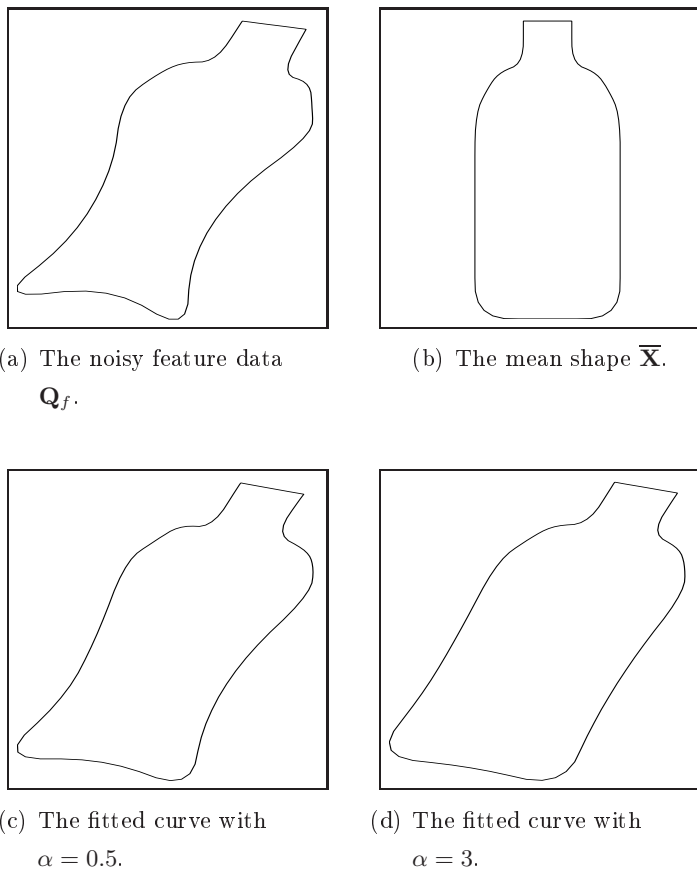


Figure 4.5: Curve fitting with regularisation.

Figure 4.5 shows an implementation of the above regulator. A curve has to be fitted to the noisy feature data shown in Figure 4.5(a). For a small value of α the regulator does not have much influence on the fitted curve's shape, but for larger values the shape of the fitted curve becomes more representative of the mean shape, as can be seen in Figures 4.5(c) and 4.5(d).

Implementing \bar{S} gives rise to the following fitting problem:

$$\min_{\mathbf{X}} (\mathbf{X} - \bar{\mathbf{X}})^T \bar{S} (\mathbf{X} - \bar{\mathbf{X}}) + \|\mathbf{Q} - \mathbf{Q}_f\|^2. \quad (4.6.3)$$

The minimisation of the above equation can be obtained by first projecting the newly found feature onto shape space as shown in (4.6.2):

$$\mathbf{X}_f = W^+(\mathbf{Q}_f - \mathbf{Q}_0).$$

The next step is to substitute this projection, (4.4.3) and (4.1.1) into (4.6.3):

$$(\mathbf{X} - \overline{\mathbf{X}})^T \overline{\mathcal{S}} (\mathbf{X} - \overline{\mathbf{X}}) + (\mathbf{X} - \mathbf{X}_f^T) \mathcal{H} (\mathbf{X} - \mathbf{X}_f).$$

Remember that $\|W\mathbf{X} - W\mathbf{X}_f\| = \|\mathbf{X} - \mathbf{X}_f\|$. By completing the square the above can be simplified to

$$(\mathbf{X} - \hat{\mathbf{X}})^T (\overline{\mathcal{S}} + \mathcal{H}) (\mathbf{X} - \hat{\mathbf{X}}) + \text{const.}$$

The regularized fitting expression is minimized by setting $\mathbf{X} = \hat{\mathbf{X}}$, where

$$\hat{\mathbf{X}} = (\overline{\mathcal{S}} + \mathcal{H})^{-1} (\overline{\mathcal{S}} \overline{\mathbf{X}} + \mathcal{H} \mathbf{X}_f). \quad (4.6.4)$$

4.6.2 Normal Displacement

Two curves with similar shapes' norms will differ considerably if the parameterisation of the two curves does not match. Curve fitting based on the norm will, therefore, suffer from sensitivity to parameterisation. A solution is to redefine the fitting problem to take the parameterisation into account when measuring the difference between the fitted curve, \mathbf{r} , and the feature curve, \mathbf{r}_f .

A reparameterisation function $g(s)$ is defined on the fitted curve so that the parameterisation of \mathbf{r} will match that of \mathbf{r}_f . The curve displacement measure, $d(\mathbf{r}, \mathbf{r}_f)$, is defined to be a minimum over all possible reparameterisations,

$$d^2 = \min_g \frac{1}{L} \int D^2 ds \quad \text{where} \quad D^2(s) = (\mathbf{r}(g(s)) - \mathbf{r}_f(s))^2.$$

A local minimum of d is achieved when

$$\frac{\partial D^2}{\partial g} = 0 \quad \text{for all } s.$$

That is, when

$$[\mathbf{r}_f(s) - \mathbf{r}(g(s))] \cdot \mathbf{r}'(g(s)) = 0.$$

Therefore, the optimal parameterisation is achieved when the vector $\mathbf{r}_f(s) - \mathbf{r}(g(s))$ is perpendicular to the tangent vector $\mathbf{r}'(g(s))$. That is, it is in the direction of the vector normal to the curve \mathbf{r} , denoted by $\mathbf{n}(g(s))$. The distance between corresponding points is now

$$D(s) = \|\mathbf{r}_f(s) - \mathbf{r}(g(s))\| = [\mathbf{r}_f(s) - \mathbf{r}(g(s))] \cdot \mathbf{n}(g(s)).$$

Assuming that the effect of reparameterisation is small, it follows that

$$\mathbf{n}(g(s)) \approx \mathbf{n}(s) \quad (4.6.5)$$

and finally

$$D(s) \approx [\mathbf{r}_f(s) - \mathbf{r}(s)] \cdot \mathbf{n}(s).$$

The above shows that the normal distance is a suitably invariant measure to describe the fitting problem, provided that the displacement between the curves is small.

A new norm with respect to the estimated curve $\bar{\mathbf{r}}(s)$, with normals $\bar{\mathbf{n}}(s)$, is now defined as

$$\|\mathbf{r}\|_{\bar{\mathbf{n}}}^2 \equiv \frac{1}{L} \int [\mathbf{r}(s) \cdot \bar{\mathbf{n}}(s)]^2 ds.$$

This norm difference is a good approximation of the invariant distance measure, $d(\mathbf{r}, \mathbf{r}_f)$, if both \mathbf{r} and \mathbf{r}_f are sufficiently close to the estimate \mathbf{r}

$$\|\mathbf{r} - \mathbf{r}_f\|_{\bar{\mathbf{n}}}^2 \approx d(\mathbf{r}, \mathbf{r}_f).$$

In the discrete case the norm difference can be approximated as a summation

$$\|\mathbf{r} - \mathbf{r}_f\|_{\bar{\mathbf{n}}}^2 \approx \frac{1}{L} \sum_{i=1}^L [(\mathbf{r}_f(s_i) - \mathbf{r}(s_i)) \cdot \bar{\mathbf{n}}(s_i)]^2.$$

In shape space, $\mathbf{r}(s_i)$ can be expressed in terms of the shape space vector, \mathbf{X}

$$\mathbf{r}(s_i) = U(s_i)(W\mathbf{X} + \mathbf{Q}_0).$$

Subsequently, the norm difference in terms of the shape space vector becomes

$$\|\mathbf{r} - \mathbf{r}_f\|_{\bar{\mathbf{n}}}^2 \approx \frac{1}{L} \sum_{i=1}^L (\nu_i - \mathbf{h}(s_i)^T [\mathbf{X} - \bar{\mathbf{X}}])^2,$$

with

$$\nu_i = (\mathbf{r}_f(s_i) - \bar{\mathbf{r}}(s_i)) \cdot \bar{\mathbf{n}}(s)$$

as the normal displacement measured relative to the mean shape, and

$$\mathbf{h}(s_i)^T = \bar{\mathbf{n}}(s_i)^T U(s_i)W.$$

4.6.3 The Algorithm

Blake & Isard (1998) proposes a recursive algorithm to find the best fitted B-spline curve, $\hat{\mathbf{X}}$. The algorithm is given in Figure 4.6. As the data curve is traversed, the estimated shape, $\hat{\mathbf{X}}$ is updated. S_i is an information matrix which measures the strength of the each intermediate estimate, $\hat{\mathbf{X}}_i$, taking into account the first i data points. \mathbf{Z} is the information weighted sum that accumulates the influence of the mean shape and the individual measurements. The measurement error is given by σ with its value $\sigma = \sqrt{L}$, where L is the number of samples.

The Curve Fitting Problem

Given an initial shape estimate $\bar{\mathbf{r}}(s)$, with shape space vector $\bar{\mathbf{X}}$ and normals $\bar{\mathbf{n}}(s)$, as well as a regularisation matrix \bar{S} , find the \mathbf{X} that will minimize T , where:

$$T = (\mathbf{X} - \bar{\mathbf{X}})^T \bar{S} (\mathbf{X} - \bar{\mathbf{X}}) + \frac{1}{\sigma} \sum_{i=1}^L (\nu_i - \mathbf{h}(s_i)^T [\mathbf{X} - \bar{\mathbf{X}}])^2.$$

The Algorithm

1. Choose samples s_i , $i = 1, \dots, L$.
2. For each i , apply some image processing filter along a suitable line, such as, the curve normal that passes through $\bar{\mathbf{r}}(s_i)$, to establish the position of the feature, $\mathbf{r}_f(s_i)$.

3. Initialise:

$$\mathbf{Z}_0 = 0, \quad S_0 = 0.$$

4. Iterate for $i = 1, \dots, L$:

$$\begin{aligned} \nu_i &= (\mathbf{r}_f(s_i) - \bar{\mathbf{r}}(s_i)) \cdot \bar{\mathbf{n}}(s_i); \\ \mathbf{h}(s_i)^T &= \bar{\mathbf{n}}(s_i)^T U(s_i) W; \\ S_i &= S_{i-1} + \frac{1}{\sigma^2} \mathbf{h}(s_i) \mathbf{h}(s_i)^T; \\ \mathbf{Z}_i &= \mathbf{Z}_{i-1} + \frac{1}{\sigma^2} \mathbf{h}(s_i) \nu_i. \end{aligned}$$

5. The aggregated observation vector is $\mathbf{Z} = \mathbf{Z}_L$, with associated statistical information $S = S_L$.

6. Finally, the best fitted curve is given in shape space by:

$$\hat{\mathbf{X}} = \bar{\mathbf{X}} + (\bar{S} + S)^{-1} \mathbf{Z}.$$

Figure 4.6: Recursive algorithm for curve fitting.

4.7 Reparameterisation

In the previous section it became clear that the parameterisation is important when measuring curve differences with the norm. Therefore, curve fitting based on the norm will suffer when the parameterisation of the feature and fitted curve differs. By taking reparameterisation into account, the distance between curves can be determined by

$$d(\mathbf{r}, \mathbf{r}_f) \approx \frac{1}{L} \int [(\mathbf{r}(s) - \mathbf{r}_f(s)) \cdot \bar{\mathbf{n}}(s)]^2 ds.$$

This is a suitably invariant measure for curve fitting, *given that the distance between the two curves is small*. This assumption of close proximity allowed the approximation in (4.6.5),

$$\mathbf{n}(g(s)) \approx \mathbf{n}(s),$$

i.e. there is little difference in the normal components of the two curves even if there is a difference in their parameterisations.

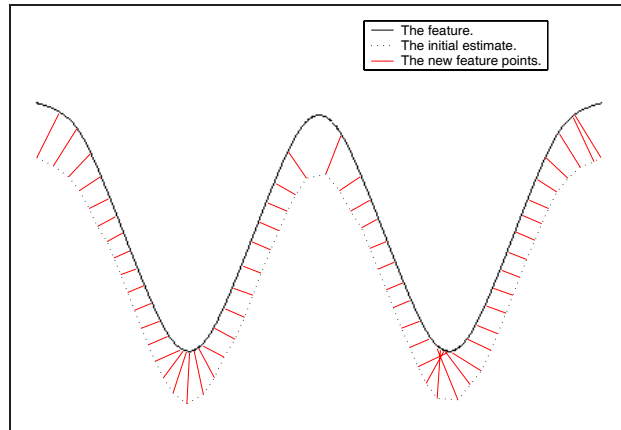


Figure 4.7: The feature curve without reparameterisation.

When the deformation increases, the above assumption no longer holds. In Figure 4.7 it is clear that the differences in parameterisation is significant. The number of features points found at the peaks and troughs differ considerably from that of the initial estimate. The order of the feature points are also not consistent at the second trough. The resulting fitted curve is shown in Figure

4.8. At the peaks one can clearly see that the scarceness of feature points influenced the fitting result and the second trough is also not very well matched. This curve fitting results will improve if the differences in parameterisation is lessened.

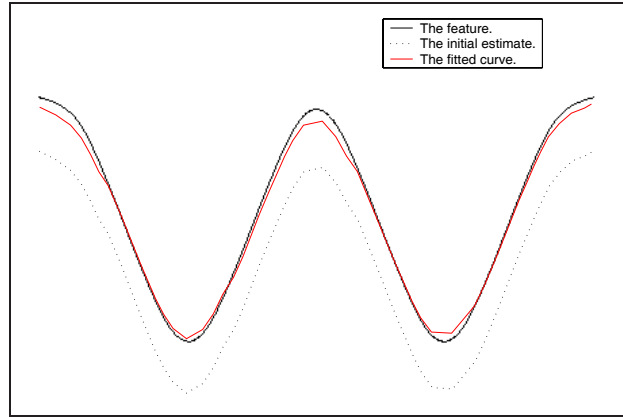


Figure 4.8: The curve fitting result for the features found in Figure 4.7.

Remember, the parameterisation of the fitted curve will be the same as that of the template by equation (4.1.1): $\mathbf{Q} = W\mathbf{X} + \mathbf{Q}_0$. It is also the same as that of the initial estimate (seeing that the initial estimate is a transformation of the template). Therefore, it makes sense to try and match the parameterisation of the feature curve to that of the initial estimate.

This can be done by implementing dynamic programming as follows:

1. An approximating spline is fitted through the feature curve. This spline consists of thousands of points; think of it a number of snooker balls lying in a pipe the shape of the feature curve.
2. The two curves' centroids are determined by (3.2.6), and subtracted so that the curves are situated about the origin.
3. Next the curves are rotated to have the same orientation. After this normalisation the feature curve and the initial estimate should lie on top of each other.

4. Dynamic programming is used to find the points of the approximating spline that match the initial estimate best. That is, for every control point of the estimate, a point on the approximating spline is found, such that the Euclidean distance between them is a minimum. More detail on dynamic programming can be found in Appendix C.
5. The curves are now restored to their original position and orientation to form the new feature curve.

The new feature curve's parameterisation will be closer to that of the fitted curve and the curve fitting will improve, as can be seen in Figure 4.9.

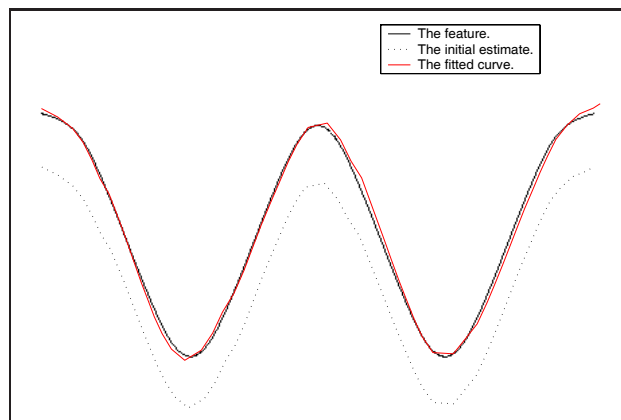


Figure 4.9: The curve fitting result after the reparameterisation of the feature curve.

4.8 Summary

Any given two dimensional curve consisting of N_B points has $2 \times N_B$ degrees of freedom.

Shape space is introduced to reduce this variability.

The space of Euclidean similarities has four degrees of freedom, allowing deformations that keep the aspect ratio constant.

The planar affine shape space allows only deformations of the initial curve that will preserve the parallelism between lines. It has six degrees of freedom.

Small, nonrigid deformations are handled by keyframes.

The fitting problem tries to find the transformation \mathbf{X} , that will minimise the norm $\|\mathbf{Q}-\mathbf{Q}_f\|$. In the simplest case the solution $\hat{\mathbf{X}}$ is given by $W^+(\mathbf{Q}_f - \mathbf{Q}_0)$. To reduce noise the fitted curve can be biased towards a mean shape $\mathbf{r}(s)$, with the degree of biasing determined by α . If it is desirable for this mean shape to only influence the result in some way and not another, a more general regulariser, \bar{S} , is defined. Usage of \bar{S} ensures that the mean curve cannot impose, for example, any Euclidean deformations on the feature curve.

Differences in parameterisation have a significant impact on the norm difference between two curves. Therefore, the distance measure is used instead. The distance measure uses only the normal component when calculating the distance between curves.

When the distance between the feature and fitted curve becomes larger, the assumption $\mathbf{n}(g(s)) \approx \mathbf{n}(s)$, no longer holds. Dynamic programming is then used to reparameterise the feature curve so that its parameterisation is closer to that of the fitted curve.

Chapter 5

Tracking Planar Affine Objects

Previously much has been said about the feature curve, the template and the initial estimate. This chapter will explain how these two curves are found in this application. After that the general tracking algorithm is given. Some considerations and results are also discussed with regard to the tracking of a planar affine object. In this chapter the focus will be on the tracking of objects whose bounding curves undergo only planar affine transformations. Its general shape, therefore, remains constant. The next chapter will look at tracking objects that deforms nonlinearly (i.e. they experience shape deformations).

5.1 Obtaining the Necessary Curves

5.1.1 Creating the template

In this application the snake was applied to the first frame of the video sequence to find the outline, or boundary, of the object to be tracked. Translating the centroid of this boundary to the origin yields the template.

5.1.2 Finding the Feature Curve

There are numerous ways to find trackable features in an image, such as edge detection, corner finding etc. The fitting algorithm requires that a corresponding feature point be found for every control point of the initial estimate. Thus, lines normal to the initial estimate, $\bar{\mathbf{F}}(s)$, are determined at all its control

points. These normal lines are traversed until a feature is encountered (a feature being an edge point as obtained by the Canny filter) or the maximum length of the normal line has been reached. This set of found feature points is called the feature curve, \mathbf{Q}_f . So, basically one looks from the estimate in a direction normal to it, for an edge.

5.1.3 Determining the Initial Estimate

Features are found on lines normal to the initial estimate. To find the features as accurately as possible, the estimate has to be situated very close to the moving object. Since the only deformations that the rigid moving object can suffer are that of the planar affine kind, it makes sense that the initial estimate should have the same form as an allowed deformation of the template. Clearly, the template itself cannot be the initial estimate as its centroid is at the origin. Therefore, the initial estimate, or mean curve, should be formed by a planar affine transformation of the template. This will ascertain that the initial estimate is close to the object tracked.

In this application the initial estimate was taken as the previous frame's curve fitting result. This is the best possible estimate as no other information of the moving object is known, except the last curve fitting result.

Let \mathbf{Q}_{pre} be the previous fitted curve. The mean shape vector, $\bar{\mathbf{X}}$, can then be calculated as

$$\bar{\mathbf{X}} = W^+(\mathbf{Q}_{pre} - \mathbf{Q}_0).$$

5.2 The General Tracking Algorithm

The tracker consists of the following steps:

1. Find the template in the first frame.
2. Determine the initial estimate (if it is the first iteration the initial estimate will be the curve found by the snake).
3. Find the lines normal to the control points of the initial estimate.
4. Search for the features along the normal lines.

5. Fit the curve through the features.

Steps 2 to 5 are iterated for the entire video sequence.

5.3 Tracking Results

Four different planar affine situations have been investigated:

1. Tracking with a closed curve against a clean (uniform) background.
2. Tracking with an open curve against a clean background.
3. Tracking with a closed curve against a busy background.
4. Tracking with an open curve against a busy background.

In every one of the above mentioned cases, a flat, peanut-like object, made from cardboard, as is shown in Figure 5.2, is used as the moving object to illustrate the tracker's performance. The tracking results and a few considerations are discussed in the following sections.

5.3.1 Tracking with a Closed Curve against a Clean Background

The simplest test for the performance of the curve fitting algorithm is to track a black planar affine object as it is propagating through the video sequence against a white background. The noise is, therefore, a minimum. The frame rate is taken as 29 frames per second to ensure that the deformation between consecutive frames is quite small.

Normal Lines and Feature Points: One of the considerations is the length of the normal lines, which is limited to six pixels. This is sufficient, as the object's deformation between consecutive frames is assumed to be small at first. A longer normal line may locate a feature point at the opposite or 'wrong' side of the object as is illustrated in Figure 5.1(a). This error will have quite an influence on the position, orientation and scaling of the fitted curve. The short normal line may not have detected any feature point at all,

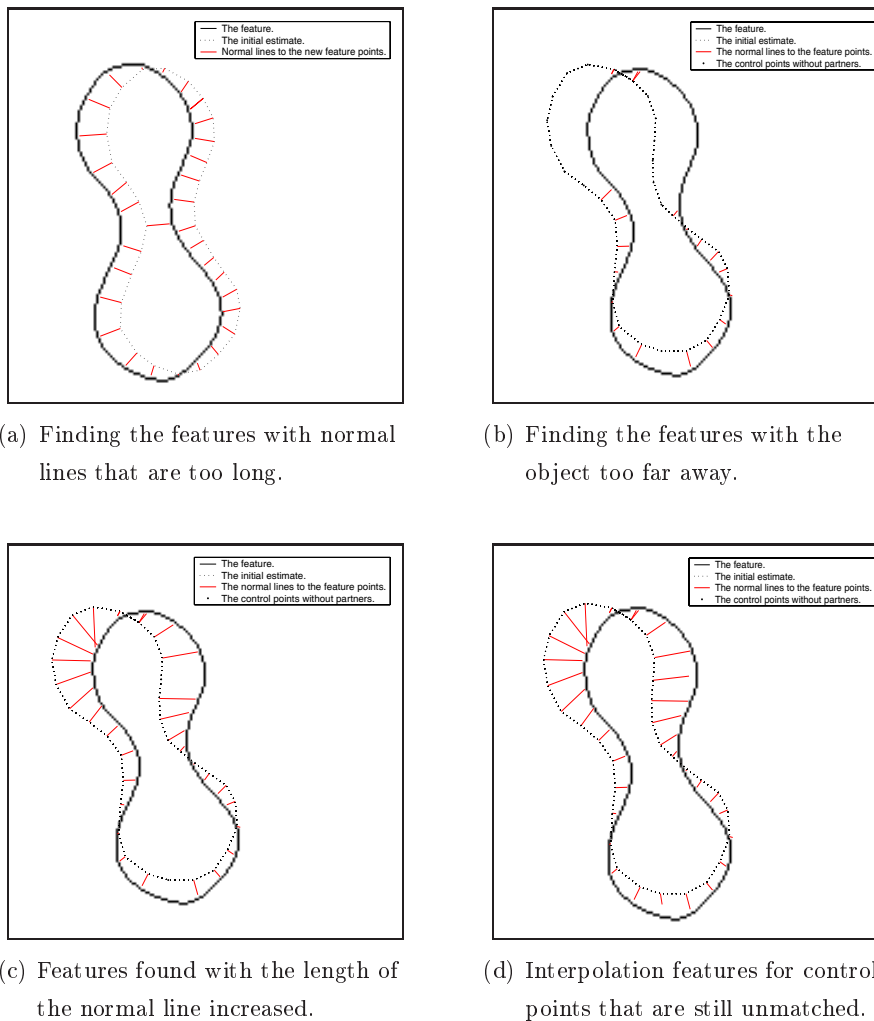


Figure 5.1: Normal line feature finding.

but there are ways to handle missing feature points that will minimise the error.

Another drawback of a short normal line is that the object to be tracked may have moved out of reach and will go undetected. This happens when the deformation is too big between frames or the previous curve fitting result was poor. As a result a number of consecutive control points will not have found a corresponding feature. This is illustrated in Figure 5.1(b). To solve this problem, the number of consecutive pairless control points are counted and if they exceed a certain limit, the length of the normal line is increased

and the searching process is repeated. The resulting features found are given in Figure 5.1(c).

If some control points still have not found a corresponding feature, they are matched with the result of a straight line interpolation between the two last found side features. (This usually happens when the edge detector is not performing well or the object to be tracked is still out of reach.) See Figure 5.1(d).

Curve Fitting Results: Three different tracking modes were investigated to track the peanut-like object given in Figure 5.2:

1. The simplest of all the curve fitting solutions: $\hat{\mathbf{X}} = W^+(\mathbf{Q}_0 - \mathbf{Q}_f)$;
2. The curve fitting algorithm of Figure 4.6, but without any regulator ($\bar{\mathbf{S}} = \mathbf{0}$);
3. The curve fitting algorithm with a regulator that does not allow the mean shape to have any Euclidean influences on the fitted curve.

The differences in the tracking results between the three methods are minor. Because of the small deformation between frames even the simplest solution, $\hat{\mathbf{X}} = W^+(\mathbf{Q}_0 - \mathbf{Q}_f)$, performed well. The curve fitting results given in Figure 5.2 are obtained by implementing the algorithm of Figure 4.6 without making use of a regulator. The results of the other two methods are not included, because there is so little difference, if any, between them.

When tracking in the planar affine shape space, there is no need really to include a regulator. The regulator allows the mean shape to influence the *shape* of the fitted curve. Here, the planar affine shape space already governs the shape of the fitted curve very strictly, as only certain deformations of the template curve are allowed. Therefore, the regulator will only try to regulate something that is already being regulated!

As soon as the step size between frames is increased the performance of the curve fitting algorithm starts to decrease. By reparameterising the feature curve the fitting is improved. An example of this improvement can be seen in Figure 5.3.

The green represents the fitted curve with no reparameterisation and the red is the fitted curve with reparameterisation. No regulator is implemented.



Figure 5.2: Random selection of result frames to indicate the performance of the curve fitting algorithm.



Figure 5.3: Improved curve fitting by reparameterising curves.

Note: The figure in 5.4 is the result of the Canny edge detector applied to frame 1 from Figure 5.2. Looking at this image one is tempted to ask: why not use only the edges to track the object? If one knows where the edges are (the only black pixels on a white background), one knows the location and the deformation of the object. This is an easy solution to the tracking problem in a noiseless environment, but as soon as the background becomes cluttered, this method fails.

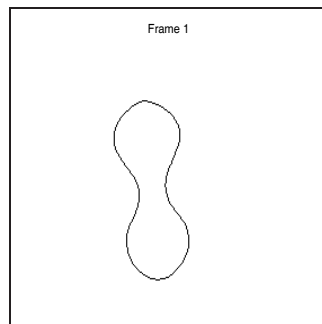


Figure 5.4: The result of the Canny edge detector applied to frame 1.

5.3.2 Tracking with an Open Curve against a Clean Background

Tracking with an open curve is studied separately from tracking with a closed curve as there are some differences in how these two are handled.

Endpoints: In the previous section a lot of attention was paid to the normal lines, and the handling of control points that did not find corresponding feature points. That all stays exactly the same when working with open curves, except

that one now has to make provision for endpoints without partners. The endpoints normally do not find a corresponding feature when the deformation is too big or the edge detector does not perform well. There are two ways to handle the situation.

One can let all the endpoints without partners share the feature of the last matched endpoint. (See Figure 5.5(a).) This method has the drawback that it lets the fitted curve shrink a little, especially if there is a number of unmatched endpoints.

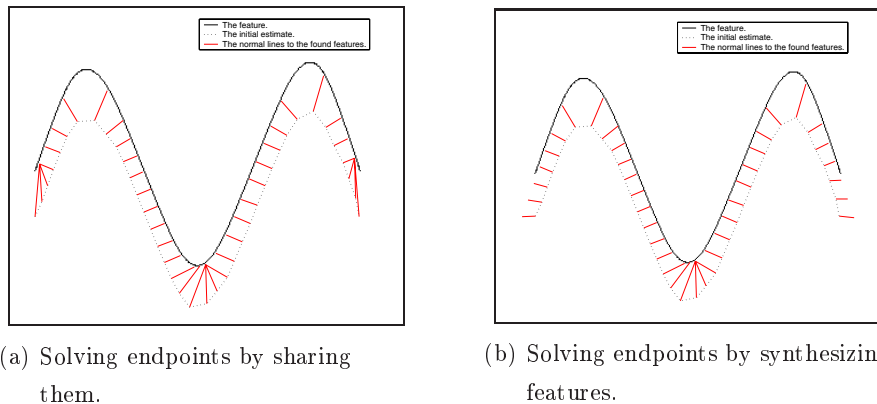


Figure 5.5: Two different ways of handling missing endpoints.

Another method is to synthesize features:

- Get the straight line equation through n of the adjacent matched feature points, where n is the number of missing end points.
- On this straight line create n features that have the same spacing as the corresponding unmatched control points.

This method performs much better as it does not force the fitted curve to shrink, and the straight line gives a good approximation of the slope of the feature curve at its end point. Figure 5.5(b) demonstrates an implementation of this method.

Curve Fitting Results: When working with open curves, one instinctively feels that there is a good chance that the object might slide out from under

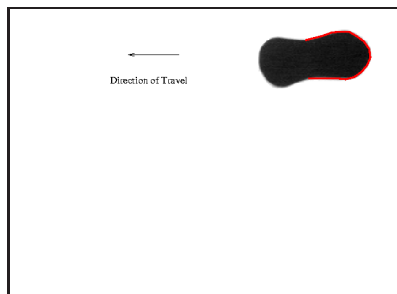


Figure 5.6: Tracking with an open curve.

the fitted curve. Especially if the direction of movement is away from the fitted curve's end as in Figure 5.6. This is the case when the moving object is tracked with $\hat{\mathbf{X}} = W^+(\mathbf{Q}_0 - \mathbf{Q}_f)$ as can be seen in Figure 5.7.

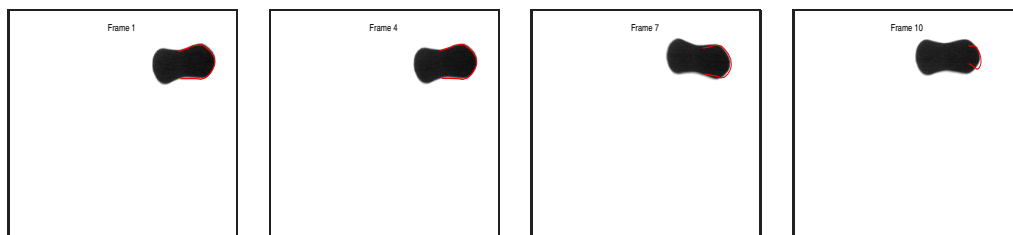


Figure 5.7: Random selection of result frames to indicate the performance of the $\hat{\mathbf{X}} = W^+(\mathbf{Q}_f - \mathbf{Q}_0)$ solution, when tracking with an open curve.

However, implementation of the tracking algorithm given in Figure 4.6 prevents this reluctance to move. This is because the tracking algorithm takes the reparameterisation into account when measuring the norm difference. Figure 5.8 shows the tracking results without using a regulator. Again there was little difference between tracking with and without a regulator.



Figure 5.8: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with an open curve, without regularisation.

With greater deformation the fitting curve does not follow the moving object so well anymore. Better results are achieved when the feature curve is reparameterised, as can be seen in Figure 5.9. The green and red represents the fitted curve before and after reparameterisation, respectively. No regulator is implemented.



Figure 5.9: Improved curve fitting by reparameterising curves.

5.3.3 Tracking with a Closed Curve against a Busy Background

As shown in the previous sections, the tracker performs very well against a clean background. The next two sections will look at the tracker's robustness against background noise.

In Figure 5.10 one can see the results of the Canny edge detector applied to the gray scale image. It is obvious that one cannot track the object by just locating all the black pixels in the image anymore. It is now also clear that



Figure 5.10: A typical image with a busy background and its edge detection result.

shorter normal lines should be used, as the chance for erroneously detecting a feature is increased. Erroneous edges detected at a distance from the actual feature, have a significant influence on the fitted curve's scale, orientation and position.



Figure 5.11: Initial tracking with a closed curve against a cluttered background.

Some Preliminary Tracking Results: The results in Figure 5.11 are obtained by tracking the moving object with an Euclidean regulator, and making use of reparameterisation. Initially the object is tracked very well, but as soon as the environment becomes too cluttered, it starts to fail. The normal lines at the back end of the object have locked onto the surfer's shoulder. Subsequently the fitted curve becomes more stretched with every frame investigated. This mistake is not corrected because the planar affine shape space allows scale deformations.

Using the Regulator to Limit the Scaling: Up to now an Euclidean regulator has been used to govern deformations. This regulator does not allow

the mean shape to have any influence on the fitted curve's position, scale and orientation.

In order to restrict the excessive scaling of the fitted curve, the regulator can be changed to let the mean curve have an influence on the size of the fitted curve. To achieve this, the invariant shape space's shape matrix, W_s , cannot contain either \mathbf{Q}_0^x or \mathbf{Q}_0^y as column elements, as their inclusion would enable the scaling of the fitted curve. As a result the orientation of the fitted curve will also be influenced by that of the mean shape. The new shape matrix, therefore, is as follows

$$W_s = \begin{bmatrix} \mathbf{1} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} \end{bmatrix}.$$

Results indicating the effect of the new regulator can be seen in Figure 5.12. With $\alpha = 0.1$ hardly any changes in the orientation of the fitted curve are allowed. The overall size of the fitted curve is, however, strictly regulated. The other extreme is when $\alpha = 0.02$. Here the fitted curve rotate with the moving object, but the regulator is not nearly restrictive enough with the scaling. When $\alpha = 0.05$ one can clearly see that there is no optimal value for α that will restrict the scaling and at the same time allow the desirable rotation. A solution has to be found that will separately manipulate the rotation and the scaling.

Limiting the Scaling between Frames: Another solution is to manipulate the fitted curve's size through the shape vector, \mathbf{X} , as the elements of \mathbf{X} directly influences the fitted curve's size and orientation. First a local and global limit for the amount of scaling allowed, has to be set. The local limit is the amount of scaling allowed from the previous fitted curve to the new fitted curve. The upper threshold for the local limit is T_{local} and the lower threshold is chosen as $2 - T_{local}$. To determine the current fitted curve's scaling factor, (4.4.6) is rewritten as

$$F_{scale} = \sqrt{A/A_{pre}},$$

where A is the area of the current fitted curve, and A_{pre} is the area of the previous fitted curve. If F_{scale} is more than the upper local threshold, the



Figure 5.12: Indication of the influence of α on the new regulator.

fitted curve needs to be rescaled by a factor

$$F_{resc} = T_{local}/F_{scale} - 1.$$

If F_{scale} is smaller than the lower limit, the rescaling factor will be

$$F_{resc} = (2 - T_{local})/F_{scale} - 1.$$

To implement this adjustment, the current fitted curve is taken as the new template, $\mathbf{Q}_{0_{cur}}$, and the corresponding shape matrix, W_{cur} , is determined. The shape vector that denotes the resized curve is

$$\mathbf{X}_{cur} = [0, 0, F_{resc}, F_{resc}, 0, 0]^T.$$

Therefore,

$$\mathbf{Q}_{cur} = W_{cur}\mathbf{X}_{cur} + \mathbf{Q}_{0_{cur}}$$

is the rescaled fitted curve, that lies within the set local boundaries.

The global limit, T_{global} , represents the amount of scaling allowed from the template to the current fitted curve. It is, therefore, the overall scaling allowed in the system. To determine and correct the global scaling, the exact same procedure as given above is followed, with A_{pre} replaced by A_0 , the area of the template. The thresholds will also take on different values.

Typically one would choose the local limit to be quite small, depending on the expected deformation between frames. The global limit is a bit bigger as there will be no scaling allowed pass this threshold.

Limiting the Rotation between Frames: A possible result of erroneous feature detection can be that the fitted curve is wrongly rotated through quite a big angle. By limiting the amount of rotation allowed between consecutive frames, this error can be improved.

In order to determine the rotation between two successive frames, one needs to:

1. Find the inertia of the template, I_o , given by (3.2.7).
2. Calculate the inertia of the first (or previous) fitted curve with (4.4.4) using I_o . Do the same for the second (or current) fitted curve.
3. From these inertia values, determine the amount of rotation between the template and the first fitted curve. That is, solve the angle, θ_{pre} , through which the largest eigenvalues of the two curves' inertias have rotated. The rotation between the template and the second curve, θ_{cur} , is found in the same manner.
4. Finally, the rotation between the two successive frames is given by the difference of their individual rotations with respect to the template:

$$\theta_{diff} = \theta_{cur} - \theta_{pre}.$$

To control the rotation a threshold, T_θ , is set. If the angle, θ_{diff} , is greater than the threshold, the rotation is restored by rotating through an angle θ_{rest} ,

$$\theta_{rest} = \theta_{diff} - T_\theta.$$

To implement the rotation, the current fitted curve is again taken as the template, \mathbf{Q}_{cur} , with its corresponding shape matrix, W_{cur} . The shape vector that represents the corrected rotation is as follows

$$\mathbf{X}_{cur} = [0, 0, \cos(\theta_{rest}) - 1, \cos(\theta_{rest}) - 1, -\sin(\theta_{rest}), \sin(\theta_{rest})]^T.$$

Finally, the corrected fitted curve can be found by

$$\mathbf{Q}_{cur} = W_{cur}\mathbf{X}_{cur} + \mathbf{Q}_{0_{cur}}.$$

Tracking Results The results shown in Figure 5.13 are obtained by limiting the amount of scaling and rotation allowed. Previously the tracker could not hold on to the moving object as it passed the surfer's shoulder. Now it manages quite well. What happens is that the one end of the fitted curve gets stuck to the background, but the other end keeps *seeing* the moving object. Because no stretching is allowed, the fitted curve is continuously being resized, or made smaller. Through this constant rescaling the fitted curve manages to loosen the background's hold on it.

The tracker even performs well while the object is moving around the scater's legs and does not loose track of it. This region is quite dark and subsequently the edges are difficult to find. The saving factor is that the object is tracked with a closed curve. That gives more stability as the 'feature capture range' of the tracker is much larger than would have been the case with an open curve.

5.3.4 Tracking with an Open Curve against a Cluttered Background

In this section an object is followed as it moves against a busy background. An open curve is used to track the object and the movement is away from the fitted curve (the same as in section 5.3.2).

Some Preliminary Results: The performance of the tracking algorithm, using a regulator, reparameterisation and limiting the deformations are shown in Figure 5.14. The tracker follows the moving object very well. Even some



Figure 5.13: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking against a cluttered background and limiting scaling and rotation.



Figure 5.14: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with an open curve against a cluttered background; and limiting scaling and rotation.



Figure 5.15: Poorer tracking results with increased deformation between frames.

horizontal lines that had to be crossed, proved to be no problem. This result was obtained with rather small deformations between subsequent frames.

With increased deformation, the tracker's performance starts to drop. Figure 5.15 shows some results with a larger step size between frames. As the moving object reaches the letter 'Y', the tracker starts struggling to keep up. Fewer features are found on the moving object and more features are located on the background, see Figure 5.16

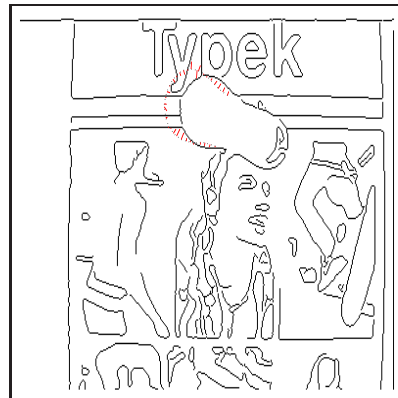


Figure 5.16: More and more features are located on the background.

Optical Flow: The above problems can be solved if the tracker can form some idea of the direction of movement. For example, if the tracker knows that the object is moving to the right, more attention can be paid to features on the right-hand side of the last fitted curve.

Optical flow is a method to determine the direction of travel in an image. It is based on an assumption that the illumination and reflection are constant throughout the video sequence. The graylevel changes from frame to frame

are, therefore, only due to the motion of the underlying objects. By calculating the optical flow, a motion vector can be formed at each pixel. More detail on optical flow can be found in Appendix D.

In this application the optical flow is calculated for a small area around the location of the initial estimate. This reduces the chance of background movement influencing the accuracy. It also optimises execution times. A general direction of travel is found by taking the average of all the motion vectors in the selected area.

To implement the information obtained from optical flow, one has to find all the normal lines that have the same gradient as that of the direction of travel. These normal lines are traversed only in the direction of travel and not in the opposite direction. This is to ensure that the fitted curve does not lock onto background information that is opposite to the direction of travel. Figure 5.17 illustrates this concept.

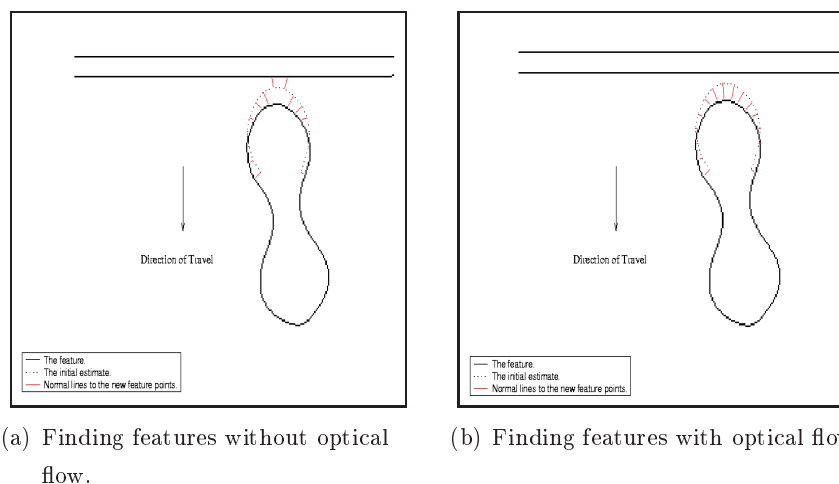


Figure 5.17: With optical flow more features are found on the moving object.

A sort of momentum is also built into the system using information obtained from optical flow. If, for example, the movement is found to be downwards, the initial estimate is shifted a few pixels (one or two) down, before searching for features. This helps to prevent the tracker from getting stuck to a background edge that coincides with the initial estimate. The momentum of

the tracker moves the initial estimate a little away from the background edge, before it starts searching for features. An example is shown in Figure 5.18.

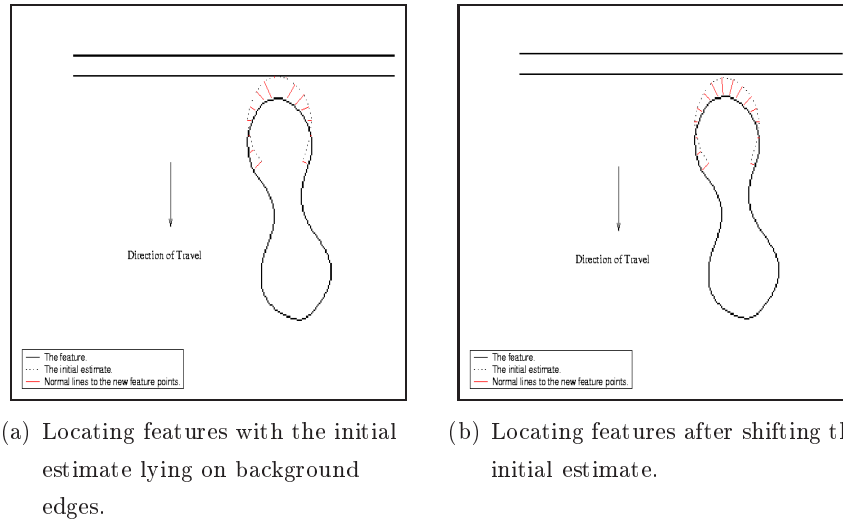


Figure 5.18: An illustration of the influence of applying momentum.

Tracking Results: Implementing optical flow in the tracking algorithm enabled the tracking of objects when deformation becomes quite large. The results can be seen in Figure 5.19. The tracker was not held back by the background, as had been the case previously. From these figures one can also note that optical flow is not always very accurate in determining the direction of travel. This is because optical flow is based on the assumption that illumination stays constant, which is not the case in this video stream. To improve tracking results, one can play around with the length of the normal lines, and with the rotation and scaling limits.



Figure 5.19: Random selection of result frames to indicate the performance of the curve fitting algorithm when the deformation between frames becomes quite big. Tracking is done with an open curve, implementing optical flow as well as scale and rotation limiting.

5.4 An Application

Tracking hand motion is widely used in security applications and sign learning. In Figure 5.20 the tracker follows the movement of a hand drawing a picture. Figure 5.21 shows the tracking result when applied to a hand moving against a realistic background. These are very simplified examples of the real life situation/application, but the tracking results are quite good. Note that the hand, which is not a planar object, can in some instances be treated as such if the distance from the camera is great enough (the perspective will have little influence).

5.5 Summary

There are some differences between tracking with an open or a closed curve. The closed curve proved to be more stable during tracking. Deformation between frames also plays an important role in the tracker's performance.

The above difficulties are handled by the choice of the length of the normal line, reparameterisation, limiting the amount of scaling and rotation allowed, and by determining the general direction of travel through optical flow.



Figure 5.20: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking the movement of a hand drawing a picture. The Euclidean regulator was used with $\alpha = 0.01$.



Figure 5.21: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking a hand that moves against a realistic background. The Euclidean regulator was used with $\alpha = 0.1$.

Chapter 6

Tracking with Keyframes

The previous chapter investigated the performance of the tracking algorithm when following a planar object as it moves through the video sequence. Movement, however, is rarely as rigid. To accommodate nonrigid movement keyframes are introduced. Results and some considerations regarding tracking with keyframes are presented here.

6.1 How to Find Keyframes with PCA

One can think of keyframes as a few outlines, taken from the nonrigid moving object, that can reconstruct, through linear combinations, every possible deformation of the moving object's bounding curve. But how does one go about to get one or two curves that, when linearly combined, will be able to match all the different deformations of the tracked object's boundary?

One starts by finding a video sequence or a bunch of images that encapsulate all the possible deformations of the object boundary. A snake is then applied to each frame to locate the edges of the moving object. These results are normalised so that they all have the same orientation and scaling, and are positioned around the origin. If there are, for example, a hundred frames, there are a hundred possible keyframes. Obviously this dataset contains redundancies. If the set of keyframes were to be used just as it is, allowing planar affine transformations, there are $100 + 6$ degrees of freedom. PCA is applied to this one hundred frames to reduce the dimensionality.

There are one hundred two dimensional datacurves (x and y coordinates). The aim is to decrease the number of curves, by forming new ones that encapsulates more information. First one needs to reduce the two dimensional data to one dimensional data and then reduce the number of one dimensional datacurves by doing a singular value decomposition of the dataset. The size of the singular values will determine the number of eigenvectors necessary to build the reduced subspace. In Figure 6.1, for example, the tenth singular value is already small enough to be discarded. A good representation of the possible deformation can be obtained by keeping only the first nine eigenvectors/principle components. These are restored to their two dimensional equivalents. Thus, the number of curves necessary to represent all the possible deformations of the moving object's bounding curve is drastically reduced.

6.2 Tracking with Simulated Data

The performance of tracking with keyframes were first tested by making use of simulated data. The same rotating cube of section 4.5 was used.

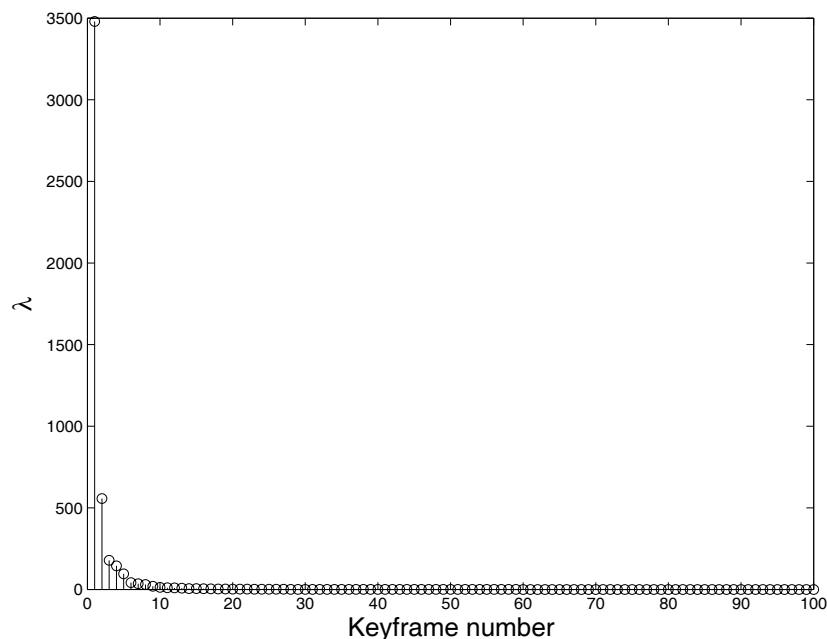


Figure 6.1: The singular values of the simulated keyframes as found by PCA.

After applying PCA to the different curves, the singular values, given in Figure 6.1, are found. There are about nine significant singular values. They contain the most information about the cube's deformation. Some of the keyframes corresponding to the bigger singular values are shown in Figure 6.2. Note that the shape of these keyframes differ considerably from the cube's bounding curve. This is mainly because of the differences in parameterisation between the initial curves (these are too big to be solved by reparameterisation). Another reason is that these curves are formed through a combination of all the original curves' control points.

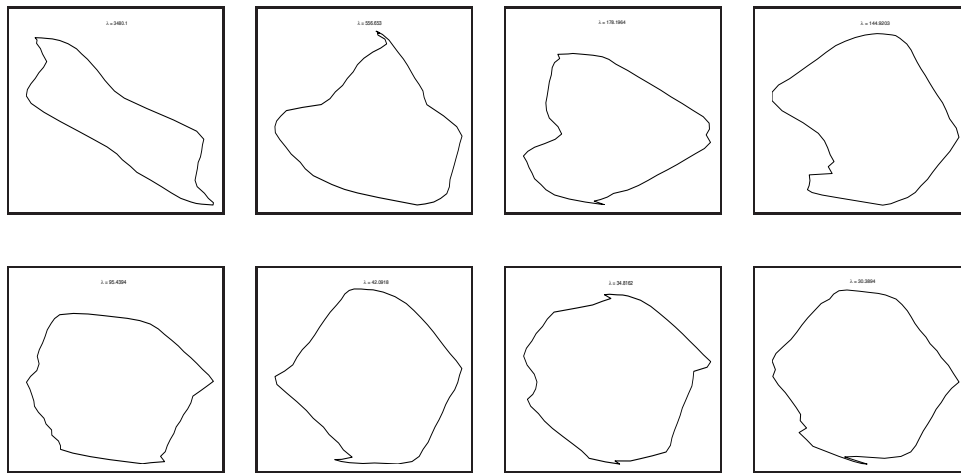


Figure 6.2: The keyframes corresponding to the eight biggest singular values as derived from the simulated data.

To test the tracker's ability, a video stream were created where the rotating cube underwent translation and rotation around an axis normal to the page. Figure 6.3 depicts some tracking results using seven keyframes to build the shape space. A regulator ($\alpha = 0.001$) governs the shape of the fitted curve, disallowing the mean curve to have any Euclidean influence on the fitted curve.

Some oscillations can be seen along the edges of the cube. These oscillations decrease if the number of frames used are decreased. Fewer frames, however, limits the tracker's ability to reproduce the required boundary throughout the simulation. (Corners are cut etc.)



Figure 6.3: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking with seven keyframes and an Euclidean regulator ($\alpha = 0.001$). Deformations are rotation and translation.

The regulator's value of $\alpha = 0.001$ is also not very strict, but a larger value will create a reluctance to deviate from the shape of the previous fitted curve. (Recall that the previous fitted curve are the new frame's mean shape.)

When background is added to the above video stream, the performance drops. The number of keyframes necessary to describe the cube's boundary allows too much freedom. Therefore, the tracker does not penalise erroneous feature points, but rather changes the shape to include them. These feature points become stuck to the background, causing the fitted curve to grow in size. To limit the scaling is not very effective when working with keyframes as the allowed shape deformation has a significant influence on the fitted curve's area. The only tool still available is optical flow, which gives the system an idea of the movement in the image.

Figure 6.4 shows the tracking results for the rotating cube as it propagates against a noisy background. Five frames are used to build the boundary's possible deformations. This is less than before. In the busy environment one does not want to allow the tracker too much freedom to deform. A stricter Euclidean regulator was also used with $\alpha = 0.005$.

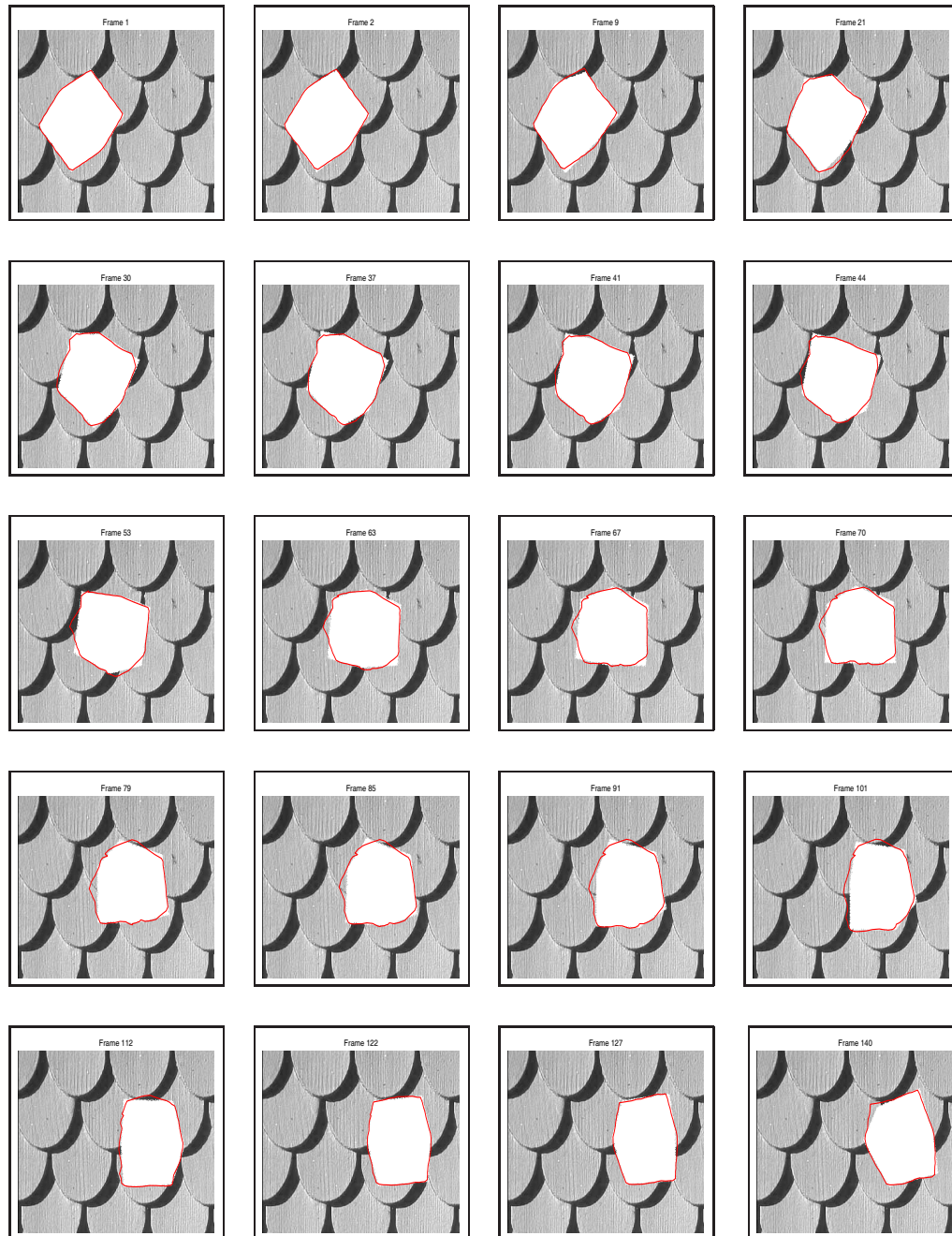


Figure 6.4: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking simulated data, against a background, with five keyframes and an Euclidean regulator ($\alpha = 0.1$). Typical deformations are rotation and translation.

6.3 Tracking with Real Data

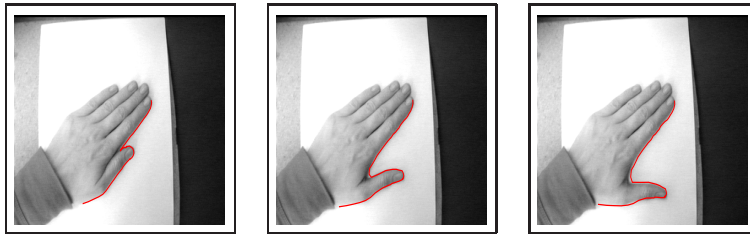


Figure 6.5: Some images of the nonlinear deformation of the hand.

There is not much difference between tracking with real and simulated data, except that the edges are better resolved in the simulated case. The next set of results are obtained from tracking a moving hand. The nonlinearity is created by moving the thumb to and from the rest of the fingers as shown in Figure 6.5.

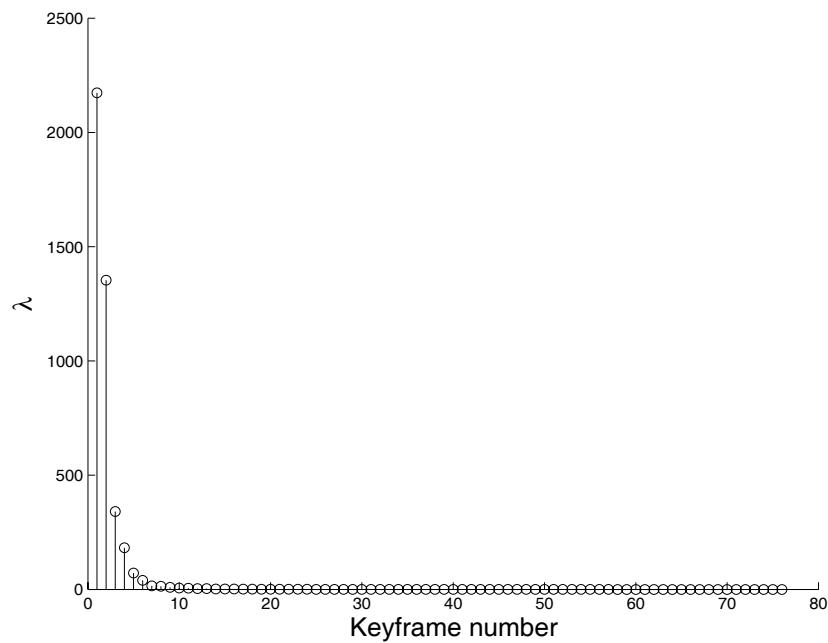


Figure 6.6: The singular values of the keyframes as found by PCA.

The singular values of the keyframes obtained are shown in Figure 6.6. There are more or less six significant keyframes. They are plotted in Figure 6.7. Again the keyframes does not resemble the initial curves very much.

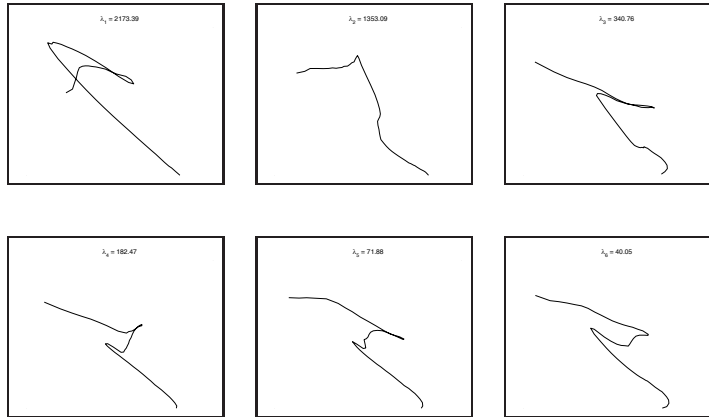


Figure 6.7: The six keyframes with the biggest singular values as derived from real data.

The first experiment involves the tracking of the moving hand against a plain white background. The six keyframes with the largest singular values are used to build the shape space. An Euclidean regulator is implemented with $\alpha = 0.005$. Figure 6.8 has a random selection of result frames.

After adding some background information to the system, the tracker's performance declines. Optical flow is used to steer the fitted curve in the direction of travel. The tracker manages to follow the object throughout its propagation as can be seen in Figure 6.9, but not always as accurately as would be desired. Again an Euclidean regulator is used with $\alpha = 0.1$ and the shape space is build with five keyframes. The result is obtained, however, only after careful consideration of all the key variables and the reduction in deformation between frames.



Figure 6.8: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking real data with six keyframes and an Euclidean regulator ($\alpha = 0.005$). Typical deformations are rotation and translation.

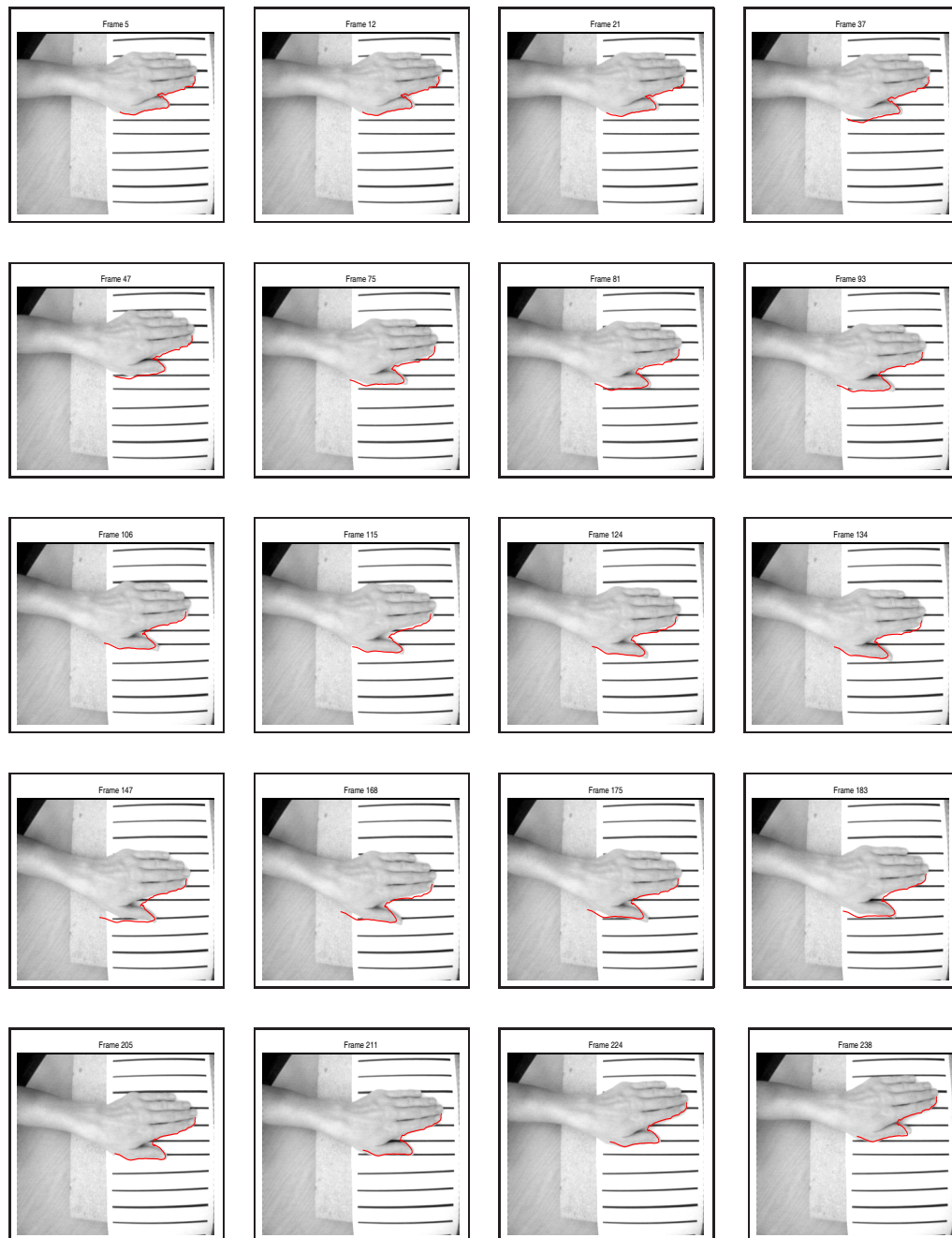


Figure 6.9: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking a hand against a background. Six keyframes and an Euclidean regulator ($\alpha = 0.01$) is used. Typical deformations are rotation and translation.

6.4 An Application

Being able to not only track an object, but to also follow its shape deformation, comes in very handy in applications such as lipreading. Keyframes enable the tracker to follow the shape changes of the bounding curve, giving important information about the mouth's deformations when speaking. These shape changes can be fused with the recorded sound to build a model for lipreading. Figure 6.10 shows some results when tracking the lips' movement with keyframes.

6.5 Summary

Keyframes are used to track objects whose bounding curve deforms nonlinearly. The performance of the tracker is dependent on the quality and the number of keyframes used, as well as the level of regularisation. Background noise does play a role in the tracker's ability to accurately follow the moving object's deformations. Again optical flow is used to help overcome some of the tracking difficulties.



Figure 6.10: Random selection of result frames to indicate the performance of the curve fitting algorithm, when tracking the movement of lips during speech, using keyframes.

Chapter 7

Kalman Filter: Comparative Results

This chapter gives a quick summary of the linear Kalman filter as it is used for the tracking of curves in shape space (Blake & Isard (1998)). For a more detailed study on the Kalman filter in general, the reader is referred to Maybeck (1979) and Haykin (2002). Also included here is a comparison of the tracking results achieved with the deterministic tracker and the Kalman filter.

7.1 The Kalman Filter

The objective of the Kalman filter is to obtain the best possible estimate of the state of a process, given all the available information, such as, measurements and the state transition model. The state transition model maps the previous system state to the current state.

Basically, the Kalman filter consists of two steps. They are the prediction of the next system state and the assimilation of the measurements. The system state, \mathbf{x} , is modelled as the output of a linear, auto regressive system excited by white noise

$$\mathbf{x}_k = A\mathbf{x}_{k-1} + \mathbf{w}_{k-1}, \quad (7.1.1)$$

where A is the state transition matrix and \mathbf{w} is a zero mean, white process noise with covariance matrix, B . The state is not observed directly, but through

noisy measurements, \mathbf{y} ,

$$\mathbf{y}_{k-1} = C\mathbf{x}_{k-1} + \mathbf{v}_{k-1},$$

where \mathbf{v} represents a white, zero mean measurement noise that has a covariance matrix, R . The measurement matrix, C , maps the measurements to the system state \mathbf{X} .

The estimate of the next state of the system is determined by combining the prediction's likelihood with the measurements matrix to form the Kalman gain, K . This gain is then combined with the measurements and prediction to determine the new estimate and its covariance matrix. The complete, general, linear Kalman filter is given in Figure 7.1.

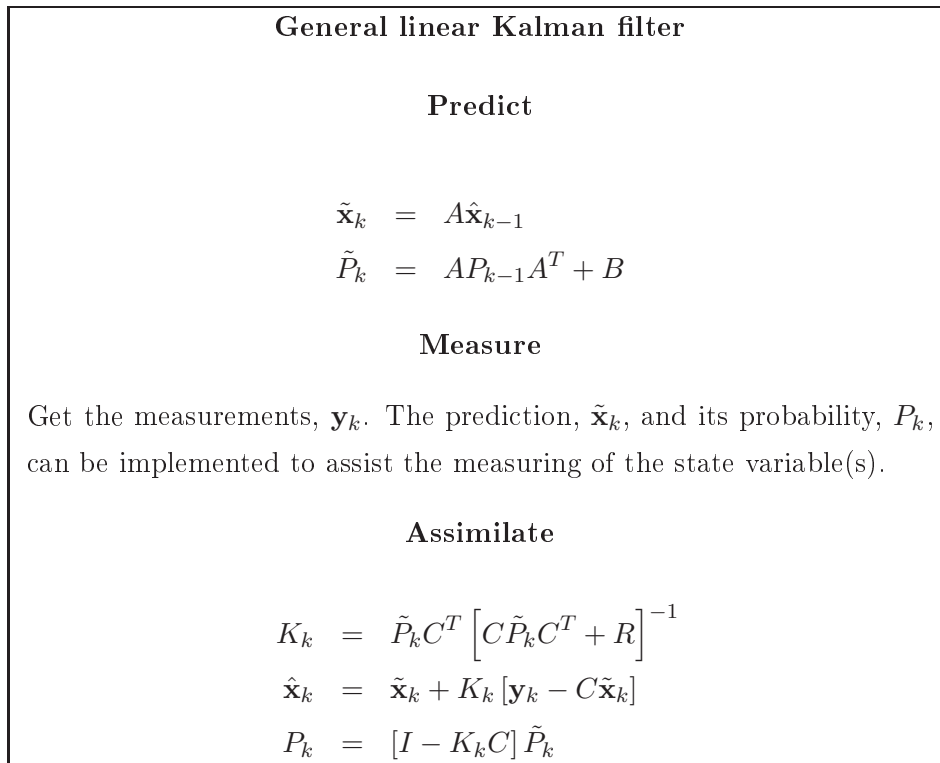


Figure 7.1: The algorithm for the general, linear Kalman filter.

7.1.1 First Order System

Applying (7.1.1) to the shape space curve fitting problem of Chapter 4, yields a first order process that looks as follows

$$\hat{\mathbf{X}}(t_k) - \bar{\mathbf{X}} = A(\hat{\mathbf{X}}(t_{k-1}) - \bar{\mathbf{X}}) + B\mathbf{w}_k, \quad (7.1.2)$$

where A is the $N_X \times N_X$ transition matrix and \mathbf{w}_k is again the noise with covariance B . Recall from (4.6.2) that $\hat{\mathbf{X}}$ is the shape space vector that denotes the estimate. This estimate minimises the norm difference between the fitted and the feature curve. In the probabilistic framework, $\hat{\mathbf{X}}$ is the mean of a Gaussian distribution with covariance, $P(t_k)$. The covariance is updated in every iteration to provide a prior probability distribution for all possible shapes at time t_k given the shape at time t_{k-1} ,

$$p(\mathbf{X}(t_k)|\mathbf{X}(t_{k-1})).$$

The above implements the Markov assumption which states that

$$p(\mathbf{X}(t_k)|\mathbf{X}(t_{k-1})) \approx p(\mathbf{X}(k)|\mathbf{X}(t_0) \dots \mathbf{X}(t_{k-1})).$$

The state transition model is used to update the covariance

$$P(t_k) = AP(t_{k-1})A^T + BB^T.$$

The first order systems does not allow for substantial changes in velocity components over time, even if it changes slowly, therefore, a second order system is derived.

7.1.2 Second Order System

The second order system is a uniform velocity model, i.e. if the state were to represent a position \mathbf{p} , the position at time t_k can be computed by

$$\begin{aligned} \mathbf{p}(t_k) &= \mathbf{p}(t_{k-1}) + \Delta\mathbf{p}(t_{k-1}) \\ &= \mathbf{p}(t_{k-1}) + (\mathbf{p}(t_{k-1}) - \mathbf{p}(t_{k-2})) \\ &= 2\mathbf{p}(t_{k-1}) - \mathbf{p}(t_{k-2}). \end{aligned}$$

Therefore, the second order system, with constant velocity, is obtained by extending the first order system to

$$\mathbf{X}(t_k) - \bar{\mathbf{X}} = A_2(\mathbf{X}(t_{k-2}) - \bar{\mathbf{X}}) + A_1(\mathbf{X}(t_{k-1}) - \bar{\mathbf{X}}) + B_0 \mathbf{w}_k, \quad (7.1.3)$$

where A_1 , A_2 and B are $N_X \times N_X$ matrices and

$$A_1 = 2 \text{ and } A_2 = -I.$$

The above can be expressed more compactly by defining a new state vector

$$\mathcal{X}(t_k) = \begin{bmatrix} \mathbf{X}(t_{k-1}) \\ \mathbf{X}(t_k) \end{bmatrix}.$$

Equation (7.1.3) can now be rewritten as

$$\mathcal{X}(t_k) - \bar{\mathcal{X}} = A(\mathcal{X}(t_{k-1}) - \bar{\mathcal{X}}) + B \mathbf{w}_k,$$

where

$$A = \begin{bmatrix} 0 & I \\ A_2 & A_1 \end{bmatrix}, \quad \bar{\mathcal{X}} = \begin{bmatrix} \bar{\mathbf{X}} \\ \bar{\mathbf{X}} \end{bmatrix} \text{ and } B = \begin{bmatrix} 0 \\ B_0 \end{bmatrix}.$$

The second order state vector has mean and covariance

$$\hat{\mathcal{X}}(t_k) = \mathcal{E}[\mathcal{X}(t_k)] \text{ and } \mathcal{P}(t_k) = \mathcal{E}[\mathcal{X}(t_k) - \bar{\mathcal{X}}],$$

where $\mathcal{P}(t_k)$ is a $2N_X \times 2N_X$ matrix that can be written as

$$\mathcal{P}(t_k) = \begin{bmatrix} P''(t_k) & P'(t_k)^T \\ P'(t_k) & P(t_k) \end{bmatrix}.$$

These covariance matrices are determined by

$$\begin{aligned} P'' &= \mathcal{E} \left[\left(\mathbf{X}(t_{k-1}) - \hat{\mathbf{X}}(t_{k-1}) \right) \left(\mathbf{X}(t_{k-1}) - \hat{\mathbf{X}}(t_{k-1}) \right)^T \right]; \\ P' &= \mathcal{E} \left[\left(\mathbf{X}(t_k) - \hat{\mathbf{X}}(t_k) \right) \left(\mathbf{X}(t_{k-1}) - \hat{\mathbf{X}}(t_{k-1}) \right)^T \right]; \\ P &= \mathcal{E} \left[\left(\mathbf{X}(t_k) - \hat{\mathbf{X}}(t_k) \right) \left(\mathbf{X}(t_k) - \hat{\mathbf{X}}(t_k) \right)^T \right]. \end{aligned}$$

The propagation of the covariance matrices now becomes

$$\mathcal{P}(t_k) = A\mathcal{P}(t_{k-1})A^T + BB^T.$$

In Figure 7.2 the complete second order Kalman filter algorithm is given. The system is initialised by choosing appropriate starting values for the state and the covariance matrices. By applying the snake algorithm to the moving object in the first image of the video sequence, the initial state can be determined by

$$\hat{\mathbf{X}}(t_0) = W^+(\mathbf{Q}_s - \mathbf{Q}_0),$$

where \mathbf{Q}_s is the vector representation of the snake curve.

To find the covariance matrices is a bit harder. One way is to initialise them as identity matrices, multiplied by a large constant. This indicates the uncertainty in the system. More accurate initial values for the covariance matrices can then be learned by running the algorithm for a test sequence that are similar to the real sequence, but contains less noise.

Algorithm for one time step**Predict**

$$\begin{aligned}\tilde{P}''(t_k) &= P(t_{k-1}) \\ \tilde{P}'(t_k) &= A_2 P'^T(t_{k-1}) + A_1 P(t_{k-1}) \\ \tilde{P}(t_k) &= A_2 P''(t_{k-1}) A_2^T + A_1 P'(t_{k-1}) A_2^T \\ &\quad + A_2 P'^T(t_{k-1}) A_1^T + A_1 P(t_{k-1}) A_1^T + BB^T\end{aligned}$$

$$\begin{aligned}\tilde{\mathbf{X}}'(t_k) &= \hat{\mathbf{X}}(t_{k-1}) \\ \tilde{\mathbf{X}}(t_k) &= A_2 \hat{\mathbf{X}}'(t_{k-1}) + A_1 \hat{\mathbf{X}}(t_{k-1}) + (I - A_2 - A_1) \bar{\mathbf{X}}.\end{aligned}$$

Measure

The algorithm of Figure 4.6 is used to obtain the measurement, with $\tilde{\mathbf{X}}(t_k)$ as the initial estimate. Note, that the measurement is not given in terms of the fitted curve, but the aggregated observation vector \mathbf{Z} and the information matrix S is used instead.

Assimilate

$$\begin{aligned}K'(t_k) &= \tilde{P}'(t_k) \left[S \tilde{P}(t_k) + I \right]^{-1} \\ K(t_k) &= \tilde{P}(t_k) \left[S \tilde{P}(t_k) + I \right]^{-1} \\ P''(t_k) &= \tilde{P}''(t_k) - K'(t_k) S \tilde{P}'(t_k) \\ P'(t_k) &= \tilde{P}'(t_k) - K(t_k) S \tilde{P}'(t_k) \\ P(t_k) &= \tilde{P}(t_k) - K(t_k) S \tilde{P}(t_k) \\ \hat{\mathbf{X}}'(t_k) &= \tilde{\mathbf{X}}'(t_k) + K'(t_k) \mathbf{Z} \\ \hat{\mathbf{X}}(t_k) &= \tilde{\mathbf{X}}(t_k) + K(t_k) \mathbf{Z}\end{aligned}$$

Figure 7.2: Second order Kalman filter.

7.1.3 Validation

The Kalman filter finds feature points slightly different than the deterministic tracker does. The normal lines can now be infinitely long as a validation gate width is used to detect outliers. The validation gate width is an algorithm that identify features that are detected too far away from the estimate and throw them out. The validation gate width is implemented by replacing step 4 of Figure 4.6 with Figure 7.3.

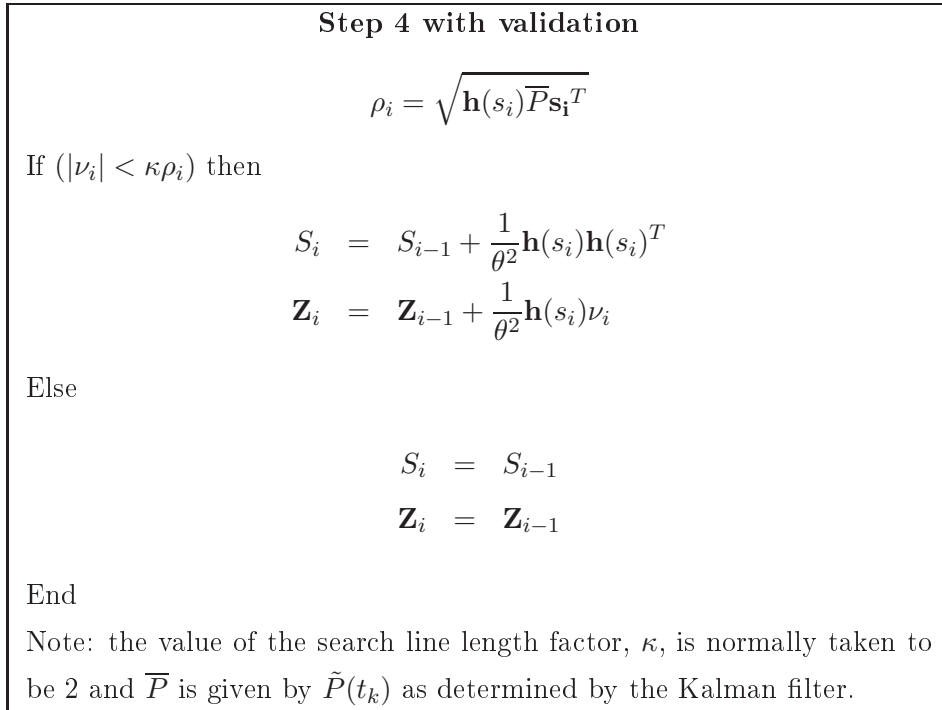


Figure 7.3: Finding and ignoring the outliers.

Previously, with the deterministic tracker, only features that were located close to the initial estimate are kept. Recall the assumption that the deformation between frames are small. Therefore, it is unlikely that a feature detected a long distance away belongs to the moving object. As a result the length of the normal lines is restricted. The Kalman filter, however, has a Gaussian distribution that can determine the likelihood that a certain feature belongs to the moving object and can classify it accordingly.

7.1.4 Implementation

An application of the Kalman filter to simulated data can be seen in Figure 7.4. The filter tracks the outline of a black cube as it follows a sinusoidal trajectory whilst rotating around an axis that is normal to the page. The filter performs well and follows the moving object very closely for the duration of the simulated movie. The covariance matrices are trained by first applying the filter to an image that contained no background.

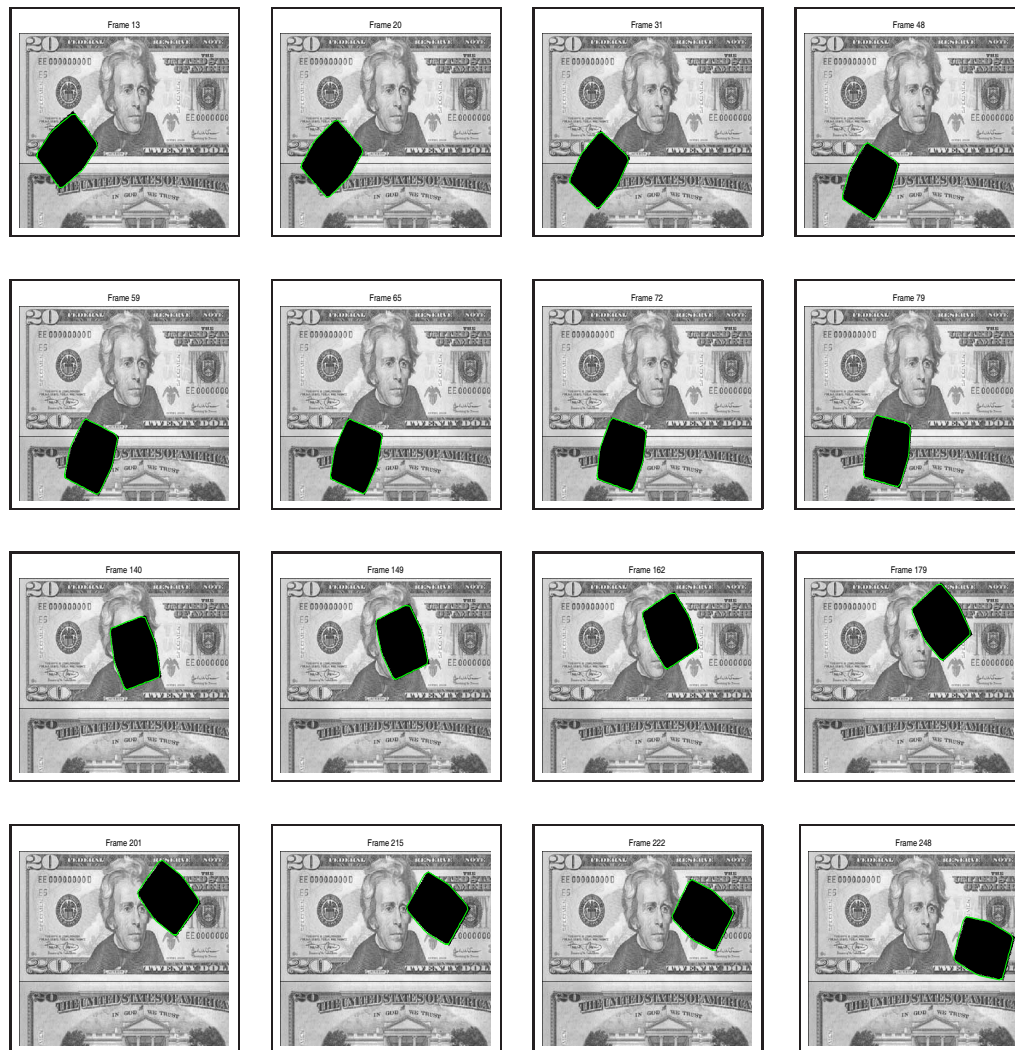


Figure 7.4: Random selection of result frames to indicate the performance of the Kalman filter when tracking simulated data against a background.

7.2 Comparative Results

The deterministic tracker performs well in all the test cases presented in Chapters 5 and 6. It tracks the moving objects accurately and, therefore, very little is gained by applying the Kalman filter to the same sequences, as both trackers will track the moving object with the minimum error.

The situations where the deterministic tracker does not always fare too well, are when:

- the deformation between successive frames are too big; and
- there are too many edges found in the background and on the moving object. See, for example, Figure 7.5.



Figure 7.5: Edges found in background and on moving object (hand).

Figure 7.6 shows the comparative results when the two filters are tracking a black cardboard object as it is moving against a cluttered background. The same video is also used in Figure 5.19, but now the step size between consecutive frames are increased considerably to increase deformation. The red indicates the deterministic tracking result and the green is the Kalman filter's result. Optical flow is implemented to boost the performance of the deterministic tracker, but it still latches onto background edges, hence the lag. It does, however, manages to recover as can be seen in the final frame. The Kalman filter tracks the object perfectly.



Figure 7.6: Random selection of result frames to compare the performance of the Kalman filter (green) with that of the deterministic filter (red) when applied to a planar affine object that undergoes large deformations.

If there are many features on the moving object, the deterministic tracker is not always as accurate. In Figure 7.7 the Kalman filter clearly outperforms the deterministic tracker. Note that the deterministic tracker manages to track the hand in a similar video (Figure 5.21), where the lighting effects do not create such strong edges on the hand.

Due to time constraints the Kalman filter is not applied to keyframes. It is, however, strongly suggested for future work as it would be interesting to compare the tracking results of the linearisation of nonrigid deformations with that of, for example, the Unscented Kalman filter.



Figure 7.7: Random selection of result frames to compare the performance of the Kalman filter (green) with that of the deterministic filter (red) when applied to a real tracking scenario.

7.3 Summary

A second order linear Kalman filter is derived to track B-spline curves represented in shape space.

From the comparative results it is clear that the Kalman filter's performance is more robust than that of the deterministic tracker when applied to noisy videos and videos with large deformations.

Chapter 8

Conclusion

The tracking procedure is initialised through the snake algorithm. It segments the moving object from its background. These control points form the template for the tracking algorithm.

To optimise the snake's performance, the edges are found through the Canny edge detector. Another concept introduced was the Harris cornerness which is used to find the corners in the image. Gradient vector flow is implemented to force the snake to propagate into concavities.

The snake performs very well, with the only drawback being that it cannot distinguish between background and object features. It tends to latch on to the closest boundary that sufficiently minimises its energy functional.

An approximating B-spline is fitted through the control points to remove noise from the snake segmented boundary. It is also used throughout the tracking sequence for the smoothing of the fitted curve.

Features are located in a direction normal to the control points of the initial estimate (the previous curve fitting result). A curve is fitted through these feature points. In the vector space this curve can deform significantly from its original shape. To reduce the fitted curve's variability, it is projected to a subspace, called the shape space, which allows only certain deformations of the fitted curve. This shape space can be specified to allow only linear transformations such as the planar affine shape space, or it can be extended to allow small, nonlinear deformations by implementing keyframes. A regulator is also included to bias the fitted curve towards a mean shape.

Differences in the parameterisation of the fitted and feature curve also need to be dealt with. Dynamic programming is used to reparameterise the feature curve.

Stumbling blocks such as excessive scaling of the fitted curve are overcome by setting limits for the amount of scaling and rotation allowed between successive frames. Optical flow is added to the tracking algorithm to keep the fitted curve from getting stuck to background features.

A Kalman filter that tracks B-spline curves is also implemented to give an idea of the performance of the deterministic tracker. It is a second order Kalman filter that allows uniform velocity in the system. This is not always optimal, but it is all that time allowed.

The deterministic tracker performs well. Submitted to favourable and not so favourable conditions it still manages to track the moving object for the duration of the video sequence, provided that the deformation between successive frames is small. To put things in perspective, it must be kept in mind that the tracker does not implement any image processing such as background subtraction. It is a deterministic tracker, not making any predictions about the moving object's expected deformations.

One of the limitations of the system is the acquisition of the image features for both the snake and the curve fitting algorithm. They mostly depend on edges found in the image. These edges do not necessarily correspond to the boundaries of the moving object. Other problems are the detection of spurious edges and the failure to detect parts of the boundary in some instances. By including region-based segmentation (methods that assume that the pixel values within one region satisfy a specific homogeneity criterion, Mirmehdi & Petrou (2000)) and combining it with edge based segmentation, a better image segmentation can be achieved. Wavelets can also be used to locate image features, Graps (1995).

Another limitation is the way in which the bounding curves, necessary to build the keyframes, are found. The snake algorithm finds the object boundaries well, but the parametric differences between them are too big for the reparameterisation method to solve. Therefore, some of the resulting PCA keyframes shown in Figure 6.9 does not nearly resemble the object they were taken from. As a result the tracker does not follow the shape changes too

closely when tested against a background, but it also does not lose the moving object.

Stochastic trackers such as the Kalman filter and the Sequential Monte Carlo method or particle filter (Wan & Van Der Merwe (2001)) outperform deterministic trackers. The curve fitting algorithm implemented here will generally be included in stochastic trackers to provide and update the measurements. This is done here by implementing a second order linear Kalman filter to track the B-spline fitted curves via their representation in shape space. The Kalman filter proved to be more robust than the deterministic filter, especially in situations with big deformations between consecutive frames or where there are a lot of noise present.

The Kalman filter was not extended to include keyframes. This is, however, recommended as it would be quite interesting to compare its performance to that of, for example, an Unscented Kalman filter or any other filter that can handle nonrigid deformations.

Appendix A

Image Processing Filters

A.1 The Sobel Filter

The Sobel filter is used to determine the gradient of a grayscale image. It consists of a pair of convolution masks. The one will compute the image gradient in the y -direction, G_y , and the other the gradient in the x -direction, G_x . The masks look as follows:

-1	-2	-1
0	0	0
1	2	1

(a) G_x

-1	0	1
-2	0	2
-1	0	1

(b) G_y

Figure A.1: The Sobel convolution masks.

What makes the Sobel filter so popular is that it combines smoothing with finding the gradient. For example:

1	2	1
---	---	---

 gives smoothing by a weighted combination of the current pixel and its two horizontal neighbours. This helps to prevent false edges from being detected. The

-1	0	1
----	---	---

 part gives the derivative by assigning the difference between the two horizontal pixels to the current pixel.

To use the Sobel filter as an edge detector, the edge strength has to be determined at every pixel. This is done by

$$|G| = |G_x| + |G_y|.$$

The edges are found by thresholding the edge strength. The result of applying the two directional filters to Figure A.2(a) and thresholding them can be seen in Figures A.2(b) and A.2(c).



Figure A.2: Application of the two directional edge detectors.

A.2 The Gaussian Lowpass Filter

The Gaussian function is given by:

$$f(x) = e^{-\frac{x^2}{2\sigma^2}},$$

with σ as the standard deviation. When filtering, σ denotes the cut off characteristics of the filter.

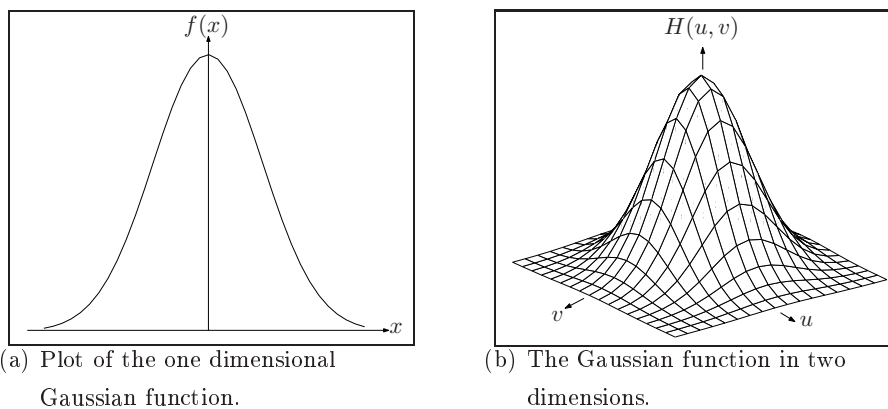


Figure A.3: The Gaussian Lowpass Filter.

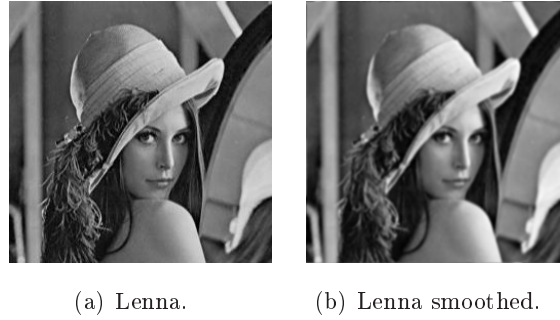


Figure A.4: Applying the Gaussian lowpass filter.

From the image in A.3(a) it is clear that $f(x)$ represents a lowpass filter. To make the filter applicable to an image, it has to be extended to two dimensions:

$$F(u, v) = e^{-\frac{D^2(u, v)}{2\sigma^2}},$$

where, if the size of the image to be filtered is $M \times N$, $D(u, v)$ is given by:

$$D(u, v) = \left[\left(u - \frac{M}{2} \right)^2 + \left(v - \frac{N}{2} \right)^2 \right]^{\frac{1}{2}}.$$

When applying the filter, it must be remembered that the image is in the spatial domain and the filter in the image domain. Therefore, either the image has to be transformed to the frequency domain and multiplied with the filter; or the filter has to be transformed to the spatial domain and be convolved with the image.

Figure A.4(b) shows the result of applying the Gaussian lowpass filter to Figure A.4(a). Notice how the sharp edges in the original image have been smoothed by the Gaussian filter.

Appendix B

Principle Component Analysis

Principle Component Analysis (PCA) is also known as the Karhunen-Loeve Transform (KLT) or the Hotelling Transform, (Shlens (2003)). In a multi-dimensional space, PCA allows data compaction by reducing the data's dimensionality. PCA projects the data onto a new subspace that will lead to a more efficient representation of the data. This projection result will encapsulate the maximum amount of variation of the dataset to ensure that the mean square error between the data and the projection is minimised.

Let \mathbf{x}_n be the a feature vector with $n = 1, \dots, N$. The PCA steps are:

1. Calculate the covariance¹ matrix of the dataset

$$\Sigma = \frac{1}{N-1} \sum_{n=1}^N (\mathbf{x}_n - \mu)(\mathbf{x}_n - \mu)^T,$$

where

$$\mu = \frac{1}{N} \sum_{n=1}^N \mathbf{x}_n.$$

2. Do an eigenanalysis of the covariance matrix $\Sigma = U\Lambda U^T$.
3. Select the M largest eigenvalues $\lambda_1, \dots, \lambda_M$.
4. Use the associated eigenvectors to form the transformation matrix, $W = [\mathbf{u}_1 \dots \mathbf{u}_M]$.

¹The correlation matrix can also be used.

5. Do the projection

$$\mathbf{y} = W\mathbf{x}.$$

The eigenvector with the largest eigenvalue represents the direction in which the data has the biggest variance, i.e. it contains a lot of information. By keeping the largest eigenvectors and eliminating the smaller ones, the mean square error is minimised, as most of the information is still contained in the new and smaller subspace. Remember that the eigenvectors are orthonormal, therefore the projected data will be uncorrelated.

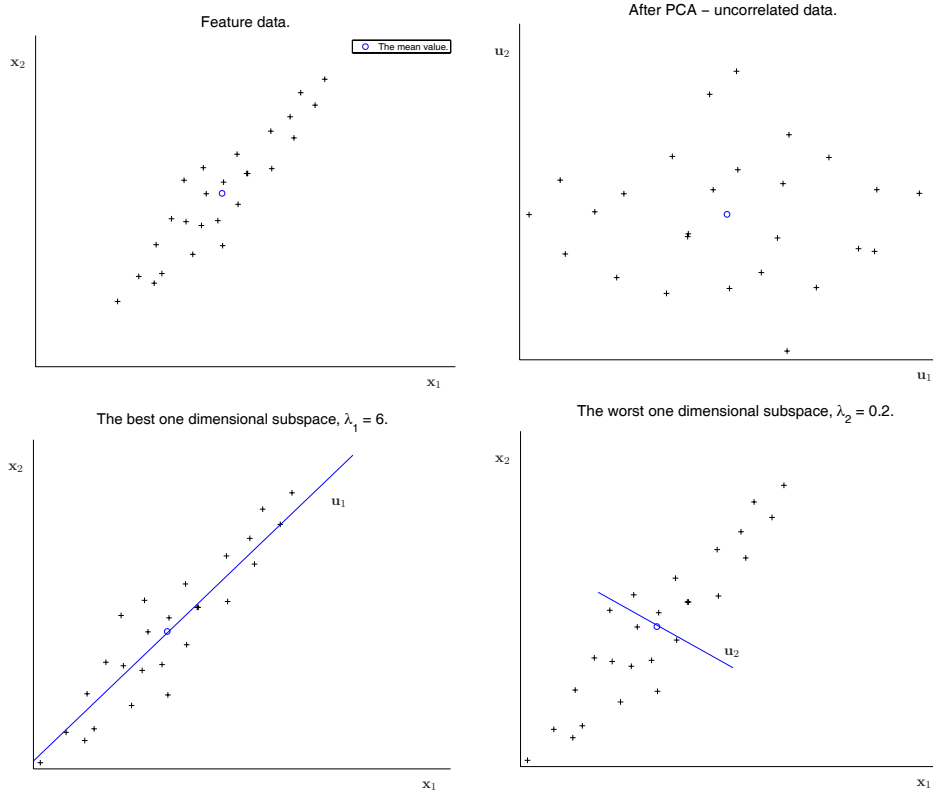


Figure B.1: Illustration of the workings of PCA.

Appendix C

Dynamic Programming

A definition for dynamic programming can be found in Deller *et al.* (2000):

Dynamic programming is a broad mathematical concept for analyzing processes involving optimal decisions.

As it is such a broad concept, dynamic programming will not be discussed here in full. The reader is referred to Cormen *et al.* (1992) for detailed information on the algorithm. The focus here will be on the optimal matching of a

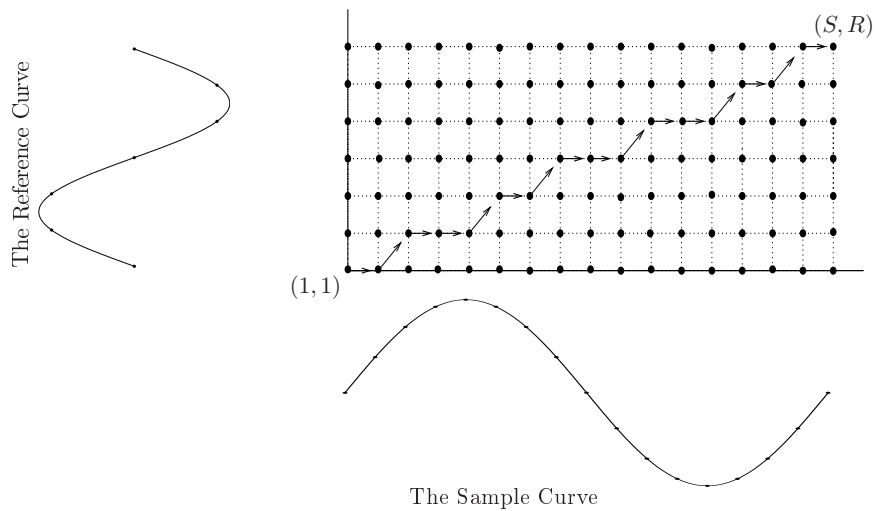


Figure C.1: Optimally matching the sample curve points to the reference curve points using dynamic programming.

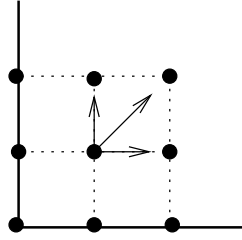


Figure C.2: Allowed travel directions through the grid.

sample curve to a reference curve. Or rather, to find the points of the sample function that will optimally match those of the reference function.

Consider the grid in Figure C.1. The horizontal axis of the grid represents the sample curve, \mathbf{X}_S , of length S , and the vertical axis the reference curve, \mathbf{X}_R , with length R . Each node of the grid describes the level of correspondence between a point of the reference curve and one of the sample curve. For example, node (i,j) expresses the level of correspondence between point \mathbf{X}_{S_i} of the sample curve and point \mathbf{X}_{R_j} of the reference curve. This level of correspondence is determined by a cost function. In this application the cost function at a node was defined as the Euclidean distance between the two points, $D(\mathbf{X}_{S_i}, \mathbf{X}_{R_j})$. Therefore, if the Euclidean distance between them is small they will have a high level of correspondence, i.e. they are a good match.

To match the whole curve, dynamic programming forms a path through the grid, starting with the Euclidean distance at node $(1,1)$. It can then move from the left to the right, or from the bottom to the top or skew (see Figure C.2 for allowed movements) until it reaches node (S,R) . The decision whether to travel upwards, sideways or skew next is made by searching for the adjacent node that has the smallest cost function associated with it. The total cost function at a node is given by

$$C(i, j) = \min \begin{cases} C(i-1, j) + D(\mathbf{X}_{S_i}, \mathbf{X}_{R_j}) \\ C(i-1, j-1) + D(\mathbf{X}_{S_i}, \mathbf{X}_{R_j}) \\ C(i, j-1) + D(\mathbf{X}_{S_i}, \mathbf{X}_{R_j}) \end{cases} .$$

For each node the previous node that lead to the minimum local cost is stored. At node (S,R) this will yield an optimal path through the grid that links all the sample points to their corresponding reference points.

The sample curve is described by considerably more points than the reference curve. Subsequently, one will find that a number of sample curve points match a reference curve point. To improve parameterisation, only the sample point with the smallest Euclidean distance to the reference curve point will be retained to form the reparameterised curve.

Appendix D

Optical Flow

Optical flow is the method used to determine the motion vector for a small region in a frame of a video sequence. By investigating the change in image brightness, a good idea can be formed of how the underlying objects has traveled from the one frame to the next. Optical flow, as described here, is summarised from Trucco & Verri (1998). Three assumptions are made when determining the optical flow:

1. The illumination and reflection are assumed to be constant throughout, allowing the changes in the graylevel image to be due only to the movement of objects.
2. The image brightness constancy yield a good approximation of the normal component of the motion field.
3. The motion field is well approximated by a constant vector field within any small patch of the image plain.

Let the grayscale image at time t be given by $I(x, y, t)$. The motion field of a small $N \times N$ region, Q , is described by a constant vector field, \mathbf{v} . For each point \mathbf{p}_i within Q one can write

$$(\nabla I)^T \mathbf{v} + I_t = 0$$

where the spatial and temporal derivatives of the image brightness are computed at $\mathbf{p}_1 \mathbf{p}_2 \dots \mathbf{p}_{N^2}$.

Within this small region the optical flow can be determined as the constant vector, $\bar{\mathbf{v}}$, that minimises

$$\Psi(\mathbf{v}) = \sum_{\mathbf{p}_i \in Q} ((\nabla E)^T \mathbf{v} + E_t)^2.$$

The above is a least squares problem. Its solution can be found by solving the linear system

$$A^T A \mathbf{v} = A^T \mathbf{b},$$

where A is a $N^2 \times 2$ matrix. The i^{th} row of A is the spatial image gradient at point \mathbf{p}_i (as for example, determined by the Sobel filter):

$$A = \begin{bmatrix} \nabla I(\mathbf{p}_1) \\ \nabla I(\mathbf{p}_2) \\ \vdots \\ \nabla I(\mathbf{p}_{N^2}) \end{bmatrix}.$$

The vector \mathbf{b} contains the partial temporal derivatives of the image brightness as evaluated at $\mathbf{p}_1 \mathbf{p}_2 \dots \mathbf{p}_{N^2}$:

$$\mathbf{b} = -[I_t(\mathbf{p}_1) I_t(\mathbf{p}_2) \dots I_t(\mathbf{p}_{N^2})]^T.$$

The least squares solution for the optical flow at the center of Q can be calculated by

$$\bar{\mathbf{v}} = (A^T A)^{-1} A^T \mathbf{b}.$$

The size of Q is generally taken as 5×5 .

Appendix E

Results

The results of the following figures can be found on the compact disc included here:

Figure 5.2; Figure 5.8; Figure 5.13; Figure 5.14; Figure 5.19; Figure 5.20; Figure 5.21; Figure 6.3; Figure 6.4; Figure 6.8; Figure 6.9; Figure 6.10; Figure 7.4; Figure 7.6; and Figure 7.7.

Bibliography

- Abolmaesumi, P., Sirouspour, M. and Salcudean, S. (2000). Real-Time Extraction of Cartroid Artery Contours from Ultrasound Images. *IEEE Symposium on Computer-Based Medical Systems*, pp. 181–188.
- Agarwala, A., Hertzmann, A., Salesin, D. and Seitz, S. (2004). Keyframe-Based Tracking for Rotoscoping and Animation. *Assosiation for Computing Machinery Transactions on Graphics*, vol. 23, no. 3, pp. 584–591.
- Barnard, M., Holden, E. and Owens, R. (2002). Lip Tracking Using Pattern Matching Snakes. In: *Asian Conference on Computer Vision*, pp. 273–278.
- Bebis, G. (2003). Deformable/Active Contours (or Snakes). Class Notes.
Available at: <http://www.cs.unr.edu/~bebis/CS791E/Notes/DeformableContours.pdf>
- Blake, A. and Isard, M. (1998 March). *Active Contours*. 1st edn. Springer-Verlag, London.
- Blake, A., Isard, M. and Reynard, D. (1995). Learning to Track the Visual Motion of Contours. *Artificial Intelligence*, vol. 78, pp. 179–212.
- Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 8.
- Chan, M., Yu, Y. and Constantinides, A. (1990 August). Variable Size Block Matching Motion Compensation with Applications to Video Coding. *IEE Proceedings on Communications, Speech and Vision.*, vol. 137, pp. 205–212.
- Cohen, I. and Medioni, G. (1999). Detecting and Tracking Moving Objects for Video Surveillance. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 2, pp. 319–325.

- Cohen, L. and Cohen, I. (1993 November). Finite-Element Methods for Active Contour Models and Balloons for 2-D and 3-D Images. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1131–1147.
- Collins, R., Lipton, A., Kanade, T., Fujiyoshi, H., Duggins, D., Tsin, Y., Tolliver, D., Enomoto, N. and Hasegawa, O. (2000). A System for Video Surveillance and Monitoring. Tech. Rep. CMU-RI-TR-00-12, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA.
- Cormen, T., Leiserson, C. and Rivest, R. (1992). *An Introduction to Algorithms*. 8th edn. McGraw-Hill.
- Deller, J., Hansen, J. and Proakis, J. (2000). *Discrete-Time Processing of Speech Signals*. 1st edn. IEEE Press, New York.
- Essa, I. and Basu, S. (1996). Modeling, Tracking and Interactive Animation of Facial Expressions and Head Movements using Input from Video. In: *Proceedings of Computer Animation*.
- Graps, A. (1995). An Introduction to Wavelets. *IEEE Computational Science and Engineering*, vol. 2, no. 2, pp. 50–61.
- Green, B. (2002). Canny Edge Detection Tutorial. Tutorial.
Available at: http://www.pages.drexel.edu/~weg22/can_tut.html
- Hall, C. and Porsching, T. (1990 March). *Numerical Analysis of Partial Differential Equations*. Prentice Hall.
- Haykin, S. (2002). *Adaptive Filter Theory*. 4th edn. Prentice Hall, New Jersey.
- Herbst, B. and Fornberg, B. (2003). Modeling in Applied Mathematics. Unpublished.
- Horn, B. and Schunck (1981 August). Determining Optical Flow. *Artificial Intelligence*, vol. 16, no. 1–3, pp. 185–203.
- Kass, M., Witkin, A. and Terzopoulos, D. (1987). Snakes: Active Contour Models. In: *Proceedings of the First International Conference on Computer Vision*, pp. 258–269. London.
- Kaucic, R., Dalton, B. and Blake, A. (1996). Real-Time Lip Tracking for Audio-Visual Speech Recognition Applications. In: *Second European Conference on Computer Vision*, pp. 376–387.

- Leroy, B., Herlin, I. and Cohen, L. (1996). Multi-resolution Algorithms for Active Contour Models. In: *Twelfth International Conference on Analysis and Optimization of Systems*, pp. 58–65.
- Maybeck, P. (1979). *Stochastic Models, Estimation, and Control*, vol. 141 of *Mathematics in Science and Engineering*. 1st edn. Academic Press.
- Mirmehdi, M. and Petrou, M. (2000 February). Segmentation of Colour Textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 2, no. 2, pp. 142–159.
- Nel, E. (2003). Estimating the Pen Trajectories of Static Signatures Using Hidden Markov Models. Unpublished.
- Parl, J., Mataxas, D., Young, A. and Axel, L. (1996). Deformable Models with Parameter Functions for Cardiac Motion Analysis from Tagged MRI Data. *IEEE Transactions on Medical Imaging*, vol. 15, pp. 278–289.
- Pera, J. and Kovacic, S. (2000). A System for Tracking Players in Sports Games by Computer Vision. *Electrotechnical Review - Journal for Electrical Engineering and Computer Science*, vol. 67, no. 5, pp. 281–288.
- Prince, J. and Xu, C. (1996). A New External Force Model for Snakes. In: *Image and Multidimensional Signal Processing Workshop*, pp. 30–31.
- Ramamoorthy, A., Vaswani, N., Chaudhury, S. and Banerjee, S. (2003). Recognition of Dynamic Hand Gestures. *Pattern Recognition*, vol. 36, pp. 2069–2081.
- Shlens, J. (2003 March). A Tutorial on Principal Component Analysis.
Available at: <http://www.sn1.salk.edu/~shlens/pub/notes/pca.pdf>
- Staib, L. and Duncan, J. (1992). Boundary Finding with Parametrically Deformable Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 11, pp. 1061–1075.
- Tek, H. and Kimia, B. (1995 June). Image Segmentation by Reaction-Diffusion Bubbles. In: *Proceedings of the Fifth International Conference on Computer Vision*, pp. 156–162.
- Tomasi, C. and Kanade, T. (1991 April). Detection and tracking of point features. Report CMU-CS-91-132, Carnegie Mellon University.
- Trucco and Verri, A. (1998 March). *Introductory Techniques for 3-D Computer Vision*. 1st edn. Prentice Hall.

- Wagener, D. and Herbst, B. (2002). Face Tracking: An Implementation of the Kanade-Lucas-Tomasi Tracking Algorithm. Unpublished.
- Wan, E. and Van Der Merwe, R. (2001 September). *Kalman Filtering and Neural Networks*, chap. 7. Wiley.
- Wang, J., Ji, L., Lin, X. and Ma, H. (2004a). Tracking Deforming Aortas in Two-Photon Autofluorescence Images and its Application on Quantitative Evaluation of Aorta-Related Drugs. *Computerized Medical Imaging and Graphics*, vol. 28.
- Wang, W., Pottman, H. and Liu, Y. (2004b). Fitting B-Spline Curves to Point Clouds by Squared Distance Minimisation. Technical report, Hong Kong University.
- Wojdel, J. (2003). *Automatic Lipreading in the Dutch Language*. Ph.D. thesis, Delft University of Thechnology.
- Xu, C. and Prince, J. (1998 March). Snakes, Shapes and Gradient Vector Flow. *IEEE Transactions on Image Processing*, vol. 7, no. 3, pp. 359–369.
- Yuille, A., Cohen, D. and Hallinan, P. (1992). Feature Extraction from Faces Using Deformable Templates. *International Journal of Computer Vision*, vol. 8, no. 2, pp. 99–112.