

A TIME VARYING APPEARANCE MODEL

A THESIS SUBMITTED TO THE UNIVERSITY OF MANCHESTER
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN THE FACULTY OF SCIENCE AND ENGINEERING

2005

By
Franck Bettinger
Department of Computer Science

Contents

Accompanying CD-ROM	13
Abstract	15
Declaration	16
Copyright	17
Acknowledgements	18
Publications	19
Dedication	20
1 Introduction	21
1.1 Motivation	21
1.2 Aims and objectives	23
1.3 Overview of the framework	24
1.4 Thesis organisation	25
2 Related work on behaviour modelling	26
2.1 Introduction	26
2.2 Learning and analysis of behaviours	26
2.3 Non-generative methods	27
2.4 Generative methods	31
2.4.1 Sample based methods	31
2.4.2 Appearance model based methods	33
2.4.3 Talking heads	36
2.4.4 Other generative methods	39

2.5	Discussion	43
3	Statistical appearance models	46
3.1	Introduction	46
3.2	Statistical shape models	47
3.3	Statistical models of appearance	49
3.4	Active appearance model	51
3.5	The tracking module	56
3.6	Conclusion	57
4	Description of the data	58
4.1	Introduction	58
4.2	Annotation of the data	58
4.3	Description of the data collected	59
4.3.1	Video sequence V1: shaking	61
4.3.2	Video sequence V2: change of expressions	65
4.3.3	Video sequence V3: dialog	67
4.4	Conclusion	69
5	Finding and modelling pathlets	70
5.1	Introduction	70
5.2	Extracting the pathlets	71
5.2.1	Finding nodes in the trajectory	71
5.2.2	The mean shift algorithm	76
5.2.3	The pathlets	78
5.3	The pathlet model	80
5.3.1	The spatial model	80
5.3.2	A spatiotemporal model	82
5.3.3	The spatiotemporal model with linear residuals	83
5.4	Grouping the pathlets	84
5.4.1	The grouping criterion	84
5.4.2	Grouping algorithm based on dynamic time warping and normalised cuts	85
5.4.2.1	The pathlet similarity measure	85
5.4.2.2	The normalised cut algorithm	85
5.4.3	The greedy algorithm	86

5.5	The results	87
5.5.1	Results of clustering based on dynamic time warping and normalised cut	87
5.5.2	Results of clustering based on the greedy algorithm	91
5.6	Conclusion	100
6	Variable length Markov model	102
6.1	Introduction	102
6.2	Encoding transition probabilities	103
6.2.1	The storage of transitions	103
6.2.2	Towards a more effective storage of transitions	103
6.3	Training a VLMM	106
6.3.1	Definitions	106
6.3.2	The learning algorithm	106
6.3.2.1	General idea	106
6.3.2.2	The pruning of nodes	107
6.3.2.3	The algorithm	108
6.3.3	The estimation of observed probabilities	108
6.3.3.1	Laplace's law of succession	109
6.3.3.2	The maximum likelihood estimate	110
6.3.3.3	Lidstone's law of succession	111
6.3.3.4	The natural law of succession	111
6.3.4	Comparison of the probability distributions	112
6.3.4.1	The Kullback-Leibler divergence	112
6.3.4.2	The Matusita distance	113
6.4	Prediction using VLMM	113
6.5	VLMM results	115
6.5.1	Comparison of the Lidstone probability estimation with the maximum likelihood probability estimation	115
6.5.1.1	Comparison of trees built using the Matusita distance	116
6.5.1.2	Comparison of trees built using the Kullback-Leibler divergence	117
6.5.2	Comparison of trees built using the Matusita measure with trees built using the Kullback-Leibler divergence for large texts	119

6.5.3	Quantitative assessment of the prediction	120
6.5.3.1	Performance given a large training set	121
6.5.3.2	Performance given a small training set	122
6.6	Conclusion	123
7	Generation of new behaviour movies	125
7.1	Introduction	125
7.2	The VLMM tree for pathlet states: an explanation of the behaviour	126
7.3	Generating a new sequence of pathlet models	128
7.4	Generating a new trajectory	131
7.4.1	Generation without the residual model	131
7.4.2	Generation with a linear residual model	133
7.5	Synthesising the new video sequence	134
7.6	Conclusion	136
8	Qualitative and Quantitative Results	137
8.1	Introduction	137
8.2	A model based on an autoregressive process	137
8.3	A measure of quality of generated videos	139
8.3.1	The comparison measure	139
8.3.2	Results with the comparison measure	144
8.4	The psychophysical experiment	146
8.4.1	The aim of the experiment	146
8.4.2	Description of the experiment	146
8.4.3	Statistics	152
8.4.4	Results of the psychophysical experiment	152
8.5	Conclusion	155
9	Conclusions	157
9.1	Summary of contributions	157
9.2	Future work and extensions	158
9.3	Final conclusions	159
	Bibliography	160
	A Dynamic time warping	170

List of Tables

5.1	Comparison of the different algorithms of the node extraction process.	73
5.2	Pathlet groups for trajectory T1 (groups 1 to 15) based on dynamic time warping and normalised cut clustering.	88
5.3	Pathlet groups for trajectory T1 (groups 16 to 25) based on dynamic time warping and normalised cut clustering.	89
5.4	Pathlet groups for trajectory T2 (groups 1 to 15) based on dynamic time warping and normalised cut clustering.	92
5.5	Pathlet groups for trajectory T2 (groups 16 to 26) based on dynamic time warping and normalised cut clustering.	93
5.6	Pathlet groups for trajectory T3 (groups 1 to 9) based on dynamic time warping and normalised cut clustering.	94
5.7	Pathlet groups for trajectory T1 (groups 1 to 12) based on the greedy algorithm.	96
5.8	Pathlet groups for trajectory T2 (groups 1 to 15) based on the greedy algorithm.	97
5.9	Pathlet groups for trajectory T2 (groups 16 to 23) based on the greedy algorithm.	98
5.10	Pathlet groups for trajectory T3 (groups 1 to 7) based on the greedy algorithm.	99
6.1	Comparison of the prediction of a VLMM learnt from a long text.	122
6.2	Comparison of the prediction of a VLMM learnt from a short text using the Matusita distance.	124
6.3	Comparison of the prediction of a VLMM learnt from a short text using the Kullback-Leibler divergence.	124

8.1	Comparison of the autoregressive process (ARP) with our model (WOR and WR).	145
8.2	Summary of the content of the questions in the experiment (questions 1 to 34).	150
8.3	Summary of the content of the questions in the experiment (questions 35 to 52).	151
8.4	Psychophysical experiment answers' summary.	154
8.5	Psychophysical experiment answers' summary for video V2.	155
8.6	Psychophysical experiment answers' summary for video V3.	155

List of Figures

1.1	Goal: interaction with an avatar.	23
1.2	Overview of the components of the model	24
2.1	Classification of the related work in behaviour modelling	27
2.2	Non-generative methods in behaviour modelling	28
2.3	Representation of a gesture by a sequence of states	30
2.4	Sample based methods in behaviour modelling	32
2.5	Appearance model based methods in behaviour modelling	34
2.6	Appearance model based methods for talking heads.	38
2.7	Other generative methods in behaviour modelling	40
2.8	Graphical representation of HMM, linked HMM, coupled HMM and cyclic HMM	44
3.1	Overview of the components of the model. This chapter explains how the face tracking module works.	46
3.2	Example of hand labelled face.	47
3.3	Example of the first mode of variation of the shape of a face.	49
3.4	Decomposition of a face into a shape and a shape-free texture.	49
3.5	Example of the first mode of variation of the face using a statistical model of appearance.	50
3.6	An example of difference image.	51
3.7	Active appearance model search algorithm.	53
3.8	Example of application of the active appearance model search al- gorithm.	54
3.9	Example of result of the AAM search algorithm.	55
4.1	Final mean square error on a video sequence of a face talking.	60
4.2	Comparison between a synthesised face and the corresponding orig- inal face.	61

4.3	Frames extracted from the video V1.	61
4.4	Hand labelling of a frame from video V1.	62
4.5	First mode of variation of the appearance model extracted from video V1.	62
4.6	Projection of the first 3 appearance parameters extracted from video V1.	63
4.7	Projection of the first 2 appearance parameters extracted from video V1.	64
4.8	Frames extracted from the video V1 after tracking.	64
4.9	Frames extracted from the video V2.	65
4.10	Hand labelling of a frame from video V2.	66
4.11	First mode of variation of the appearance model extracted from video V2.	66
4.12	Frames extracted from the video V2 after tracking.	67
4.13	Frames extracted from the video V3.	67
4.14	Hand labelling of a frame from video V3.	68
4.15	First mode of variation of the appearance model extracted from video V3.	69
4.16	Frames extracted from the video V3 after tracking.	69
5.1	Overview of the components of the model	70
5.2	Overview of the pathlet groups extraction process.	72
5.3	Algorithm A2	74
5.4	Selection of nodes using algorithm A2.	75
5.5	The mean shift algorithm.	77
5.6	Small distance for the selection of neighbours.	78
5.7	Large distance for the selection of neighbours.	79
5.8	Selection of the pathlets given the nodes	80
5.9	Resampling using cubic splines.	81
5.10	Linear model of residuals.	84
5.11	The greedy algorithm.	87
5.12	Generation of pathlets with negative timings.	91
6.1	Overview of the components of the model	102
6.2	Example of storage of transitions in a VLMM model.	105

6.3	VLMM tree learnt using the maximum likelihood estimation of probability and the Matusita distance.	116
6.4	VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.05$ and the Matusita distance.	117
6.5	VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.1$ and the Matusita distance.	118
6.6	VLMM tree learnt using the Laplace estimation of probability and the Matusita distance.	118
6.7	VLMM tree learnt using the maximum likelihood estimation of probability and the Kullback-Leibler divergence.	119
6.8	VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.5$ and the Kullback-Leibler divergence.	120
6.9	VLMM tree learnt using the Laplace estimation of probability and the Kullback-Leibler divergence.	121
6.10	VLMM tree learnt using the maximum likelihood estimation of probability and the KL divergence.	122
6.11	VLMM tree learnt using the maximum likelihood estimation of probability and the Matusita distance.	123
7.1	Overview of the components of the model	125
7.2	VLMM tree of pathlet states.	127
7.3	Representation of the sequence B	128
7.4	Representation of the sequence A	129
7.5	VLMM tree of pathlet states for the trajectory T2.	130
7.6	Trajectories generated with the normalised cut algorithm and the dynamic time warping algorithm.	132
7.7	Trajectories generated with the greedy algorithm.	133
7.8	Constraining the generation of pathlets.	134
7.9	Trajectories generated with the normalised cut algorithm, the dynamic time warping algorithm and a linear residual model.	135
7.10	Trajectories generated with the greedy algorithm and a linear residual model.	135
8.1	Frames taken every 4 seconds from the video sequence generated with an autoregressive process.	138
8.2	Frames taken every 4 seconds from the original long video sequence.	138

8.3	Frames taken every 4 seconds from the video sequence generated with our model and the linear residual model.	138
8.4	Comparison of generated trajectories.	140
8.5	Comparison between the original trajectory and a trajectory generated by the autoregressive process.	142
8.6	Comparison between the original trajectory and a trajectory generated by our model.	143
8.7	Screen capture of the description of the experiment.	147
8.8	Screen capture of a question during the experiment.	148
8.9	Time taken to answer the questions in the experiment.	153
A.1	The dynamic time warping matching algorithm.	170
A.2	Grid representation of the optimal warp.	171

Accompanying CD-ROM

The CD-ROM inside the back cover of this thesis contains several video sequences referenced in the text. The video sequences are referenced according to their filenames on the CD-ROM:

- `tracked/V1_graph.m1v`, video V1 along with a plot of the tracked parameters,
- `tracked/aam_orig.m1v`, video comparing an original video sequence with the resynthesised frames from the tracked parameters,
- `examples/V1/V1_orig.m1v`, the original video sequence V1,
- `examples/V1/V1_track.m1v`, video V1 resynthesised from tracked parameters,
- `examples/V1/V1_arp.m1v`, generated from video V1 using the autoregressive process,
- `examples/V1/V1_wor.m1v`, generated from video V1 using our model without residuals (greedy algorithm),
- `examples/V1/V1_wr.m1v`, generated from video V1 using our model and a linear residual model (greedy algorithm),
- `examples/V1/V1_ncwor.m1v`, generated from video V1 using our model without residuals (normalised cuts algorithm) ,
- `examples/V1/V1_ncwr.m1v`, generated from video V1 using our model and a linear residual model (normalised cuts algorithm),
- `examples/V2/V2_orig.m1v`, the original video sequence V2,

- `examples/V2/V2_track.m1v`, video V2 resynthesised from tracked parameters,
- `examples/V2/V2_arp.m1v`, generated from video V2 using the autoregressive process,
- `examples/V2/V2_wor.m1v`, generated from video V2 using our model without residuals (greedy algorithm),
- `examples/V2/V2_wr.m1v`, generated from video V2 using our model and a linear residual model (greedy algorithm),
- `examples/V3/V3_orig.m1v`, the original video sequence V3,
- `examples/V3/V3_track.m1v`, video V3 resynthesised from tracked parameters,
- `examples/V3/V3_arp.m1v`, generated from video V3 using the autoregressive process,
- `examples/V3/V3_wor.m1v`, generated from video V3 using our model without residuals (greedy algorithm),
- `examples/V3/V3_wr.m1v`, generated from video V3 using our model and a linear residual model (greedy algorithm);
- `exp/m1v/q01.m1v` to `exp/m1v/q52.m1v` are the videos used for the psychophysical experiment (in mpeg format);
- `exp/gif/q01.gif` to `exp/gif/q52.gif` are the videos used for the psychophysical experiment (in animated gif format).

Those video sequences can be viewed by browsing the file `index.htm`.

Abstract

Statistical appearance models are used to model objects from images using their shape and texture. Such models have been applied successfully in a large number of applications. Nevertheless, the appearance model does not model video sequences of animated deformable objects.

The aim of this thesis is to add a temporal dimension to the appearance model in order to properly represent movement in video sequences. We apply this extended model to the study of facial behaviour.

The method uses a statistical framework learnt from a training video sequence. The series of parameters extracted from the sequence is modelled by a set of pathlets in parameter space. A higher level model learns how to organise the pathlets into meaningful sequences representing facial expressions or typical movements of the head.

A measure of quality of generated video sequences is derived. This measure shows that our model outperforms an alternative based on autoregressive processes. A forced choice psychophysical experiment confirms this conclusion.

Declaration

No portion of the work referred to in this thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institution of learning.

Copyright

Copyright in text of this thesis rests with the Author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the Author and lodged in the John Rylands University Library of Manchester. Details may be obtained from the Librarian. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the Author.

The ownership of any intellectual property rights which may be described in this thesis is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

Further information on the conditions under which disclosures and exploitation may take place is available from the head of Department of Computer Science.

Acknowledgements

I would like to thank my supervisor, Dr Tim Cootes for his useful advice, his encouragement, guidance and support over the past four years.

I would like to thank all the members of the Imaging Science and Biomedical Engineering group for providing such a good work environment.

I would like to thank my parents for supporting me.

Finally, I would like to thank all the people who volunteered for the psychophysical experiment set up to assess this thesis' framework. I would particularly like to thank Lilian, Nicolas, Fabrice, Juana, Gilles, Vivek, Jun, Alexandre, Arnaud, Bruno, Laurent, Fernand, Marie-Jeanne, Paul, José, Kostas, Domitille, Xavier, Sylvie, David, Kolawole, Roy, John, Mike, Patrick and Panachit for their help and their patience.

Publications

Some of the work described in this thesis has also appeared in:

- F. Bettinger, T. F. Cootes, and C. J. Taylor. Modelling facial behaviours. In Paul L. Rosin and David Marshall, editors, *British Machine Vision Conference*, pages 797–806, Cardiff, UK, September 2002.
- F. Bettinger and T. F. Cootes. A model of facial behaviour. In *Proceedings of the 6th International Conference on Automatic Face and Gesture Recognition*, pages 123–128, Seoul, Korea, May 2004.

Dedication

I dedicate this thesis to my family.

Chapter 1

Introduction

1.1 Motivation

Humans are social animals, and communication underpins our society. The most natural way of interacting with anyone is face-to-face. It has thus long been a goal of research into human computer interaction to be able to mimic this face-to-face communication.

Raudsepp [78] has pointed out that in a dialog, 7% of the message conveyed is verbal, 38% is vocal and 55% lies in the body language. The verbal part of a conversation is the content of the conversation itself. The vocal part is the paralanguage, that is the tone of voice, intonation, pauses and sighs. The body language consists of posture, distance maintained with the speaker, eye contact, gestures and facial expression. The importance of eye contact in a conversation suggests that a natural human-computer interface should display a picture of a face on the screen.

Gestures have been extensively studied by the machine vision community in order to enable computers to understand humans using their natural way of expressing themselves. Results of these studies can provide an elegant alternative to input devices used with computers, especially for interaction in 3D environments.

More recently, facial expressions have been studied. Classification or assessment of facial expressions from a video sequence can be used by computers to understand how the user feels and to react in an adapted manner to his emotional state. Indeed, even if it is often difficult for a human to detect an expression on the face of a person, the face reflects most of our emotions. Many of the studies in facial expression classification concentrate on extracting the six basic emotions

from images or sequences of images: happiness, sadness, surprise, fear, anger and disgust. A trained human is able to distinguish these emotions using facial clues with an error average of 13% [75].

Building a human-computer interface based on visual clues requires several stages:

- 1) a tracking system that is able to locate the face of the user each time it is required. The tracking problem is challenging because of the variability of the expressions one can show. In order to be able to bring useful information to the next stages, it has to be both accurate and robust. Facial hair, glasses, occlusions and sensor noise are problems that make this task difficult.
- 2) the analysis of the user's face to extract information such as facial expression or the user's gaze direction. We can then combine this information to deduce the state of the user.
- 3) a decision of how to react to the user.
- 4) the synthesis of a virtual face which seems to react in a manner which appears natural.

Unfortunately, such an ideal human-computer interface is still in its infancy. In order to achieve such a complex task, a good model of facial behaviour is needed. This thesis describes one approach to creating such a model.

Such a model could be used to animate a crowd in games, for instance. Each face can be synthesised using a model of facial behaviour. Typical motions could be achieved this way by learning the model on typical videos from the target game. For instance, learning the model from a spectator of a tennis game, would produce shaking head avatars.

Although we apply our model to facial behaviour, it can model the behaviour of any deformable object, such as fire for instance. Synthesised videos of fire could be used in games, movies or 3D environments.

Finally, synthesis is not the only purpose of a behaviour model. We would also like a model we can interpret. Interpretation could be useful for a variety of tasks such as diagnosis (if we model the appearance of a heart valve or another organ), or theft detection (if we model the appearance of people in car parks for instance).

1.2 Aims and objectives

The aim of this work is to create a generic model that is able to synthesise both the appearance and behaviour of deformable objects such as human faces. The model will be an extension of the appearance model designed by Lanitis, Cootes and Taylor [57]. The appearance model is a probabilistic model used to describe an object in a still image by learning its shape and appearance from a database of hand labelled images of similar objects. It is described in more detail in chapter 3 of this report.

In the long term, we would like to apply the model to the study of interaction between people during conversations in order to synthesise one person as a response to the behaviour of a real person (see figure 1.1). For instance, we would expect a virtual person to close his mouth when the real person is speaking or the virtual person may smile when the real person is smiling.

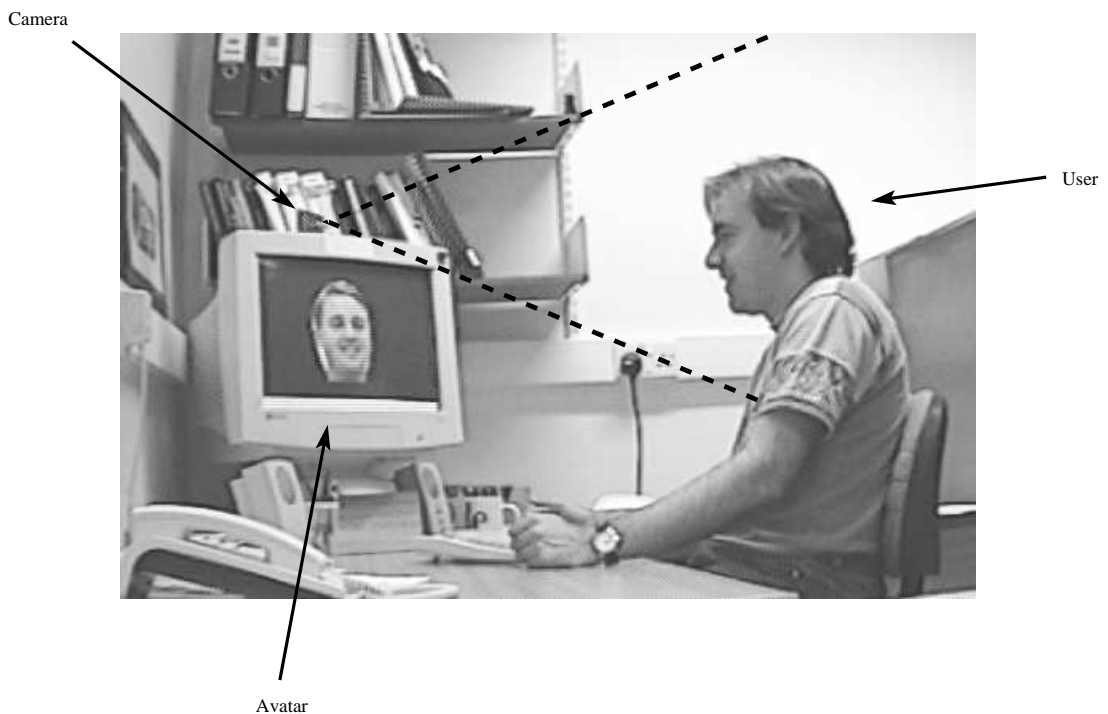


Figure 1.1: Goal: interaction with an avatar. In an ideal system, a camera films the user and an avatar is displayed reacting to the user's expressions. Our system does not model the interaction.

For the purpose of this thesis, we bound ourselves to the generation of facial behaviour of one person. The facial behaviour is learnt from a video sequence of

the person. The resulting model can be used to generate new video sequences of the same person exhibiting a similar behaviour.

We wish to avoid using frames from the original video sequence when generating behaviours, since people are good at noticing loops and frames reappearing several times. We also want the model to be explicit. We want to be able to understand the behaviour of a face in the video sequence by looking at representations of the different components of the model.

1.3 Overview of the framework

Figure 1.2 shows an overview of the model of facial behaviour we have developed. First, the face has to be tracked in the video sequence (1). The appearance model parameters have then to be deduced from the tracked face (1 \rightarrow 2). The trajectory formed by those appearance parameter vectors is then broken into pathlets (2 \rightarrow 3) which are grouped and modelled. The trajectory is now a sequence of pathlet models (3). The sequence of pathlet models is learnt (3 \rightarrow 4) by a variable length Markov model (4).

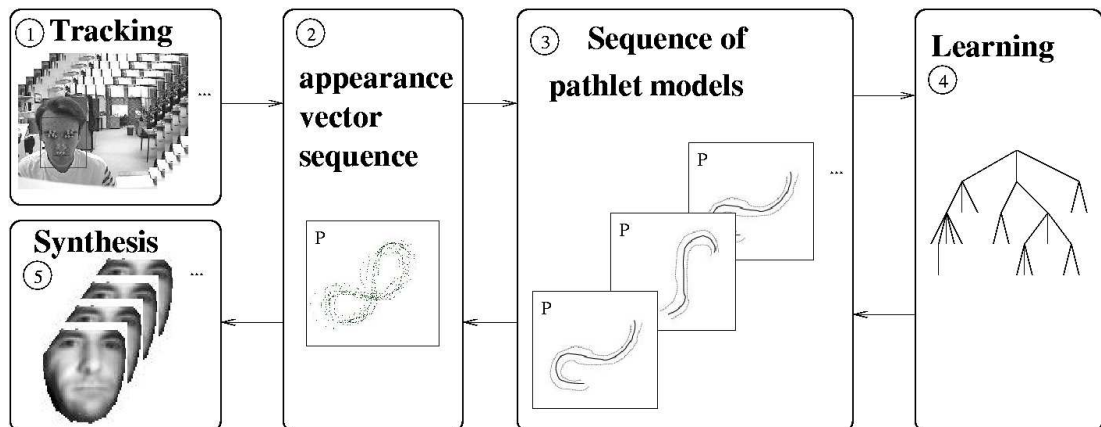


Figure 1.2: Overview of the components of the model. P is the appearance parameter space. Arrows from left to right represent the learning phase and arrows from right to left represent the generation phase. Each face from the frames of the original video sequence corresponds to a point in the space P . The trajectory of the points in P is modelled by a sequence of pathlet models. The temporal relationship of the pathlet models is learnt by a variable length Markov model.

In order to generate new trajectories, a new sequence of pathlet models has

to be sampled from the variable length Markov model ($4 \rightarrow 3$). A new pathlet has to be sampled from each pathlet model in the sequence ($3 \rightarrow 2$) to give a sequence of pathlets, that is a trajectory (2). Each point in that new trajectory in the appearance parameter space can then be synthesised ($2 \rightarrow 5$) to give a video sequence of faces (5).

1.4 Thesis organisation

The remainder of the thesis is organised as follows:

- **chapter 2** reviews the related work on behaviour modelling.
- **chapter 3** describes the active appearance model and the tracking module.
- **chapter 4** describes the data acquired to assess the model.
- **chapter 5** describes how to transform a trajectory into a sequence of pathlet models and provides a visual comparison of the two methods used to find those sequences.
- **chapter 6** describes the variable length Markov model and gives results on the quality of its prediction capabilities.
- **chapter 7** describes how to synthesise new video sequences of behaviour with our model.
- **chapter 8** describes an alternative model and a measure of comparison between behaviour models; it provides some quantitative results as well as qualitative results based on a psychophysical experiment.
- **chapter 9** concludes.

Chapter 2

Related work on behaviour modelling

2.1 Introduction

The main aim in machine vision is to interpret images. By extension, it also aims at interpreting image sequences. In this chapter, we review how people have tried to model behaviour and interactions between different actors in a video sequence.

2.2 Learning and analysis of behaviours

Behaviour from image sequences has been studied by several research groups. We can separate those studies into two main classes: the generative methods and the non-generative methods. The non-generative methods are used for recognition while the generative model can be used for both recognition and synthesis. The generative methods can be further separated into sample based methods, methods using appearance models and other generative methods. The sample based methods use frames from the original sequence to synthesise new sequences (the sequences can be video sequences or 3D skeleton sequences or any kind of sequences described in the following sections). A graphical tree representation of this classification is shown on figure 2.1. Our model can be classified as being in the appearance model based methods as denoted by the bold lines on the figure 2.1.

The following sections describe the work related to ours for each class of method used.

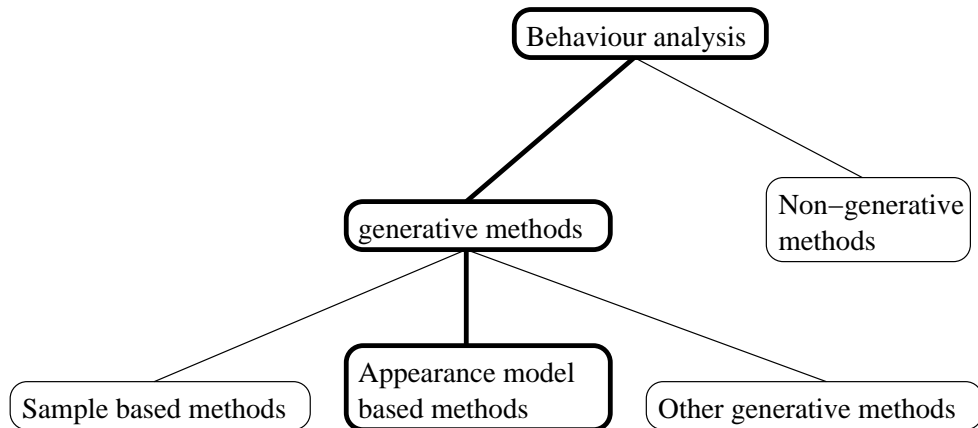


Figure 2.1: Classification of the related work in behaviour modelling. The classification of our model is represented by bold lines.

2.3 Non-generative methods

A classification of the non-generative methods can be found in figure 2.2. The methods are usually used for recognition of behaviour. However, a new sequence that represents this behaviour cannot be generated from the models directly.

Fleet *et al.* [32] use optical flow to learn motion in a sequence of images. A principle component analysis is used on data obtained from an optical flow algorithm to find the first order components of the velocity fields. It has been tried on the modelling of lips motion of a talking head. The first seven components found explained 91.4% of the variance. They used this result to build a simple user interface controlled by the head and the mouth. The mouth gives orders such as “track”, “release” or “print” and the head is used like a joystick in order to scroll when in tracking mode.

Black and Jepson [7] use the particle filtering methods to recognise signs written on a white board in order to control a computer. A training set of manually aligned signs is used to compute a model for each sign. Each model represents the mean trajectory of signs that encode the same action. A set of states is defined and represent the model used and the scale used to fit to the template stored for the model. When a new sign is presented to the system, probabilities of matching signs in the model database are computed by a chosen probability measure. A current state is chosen using the computed probabilities. The next state is then predicted from the current state by diffusion and sampling,

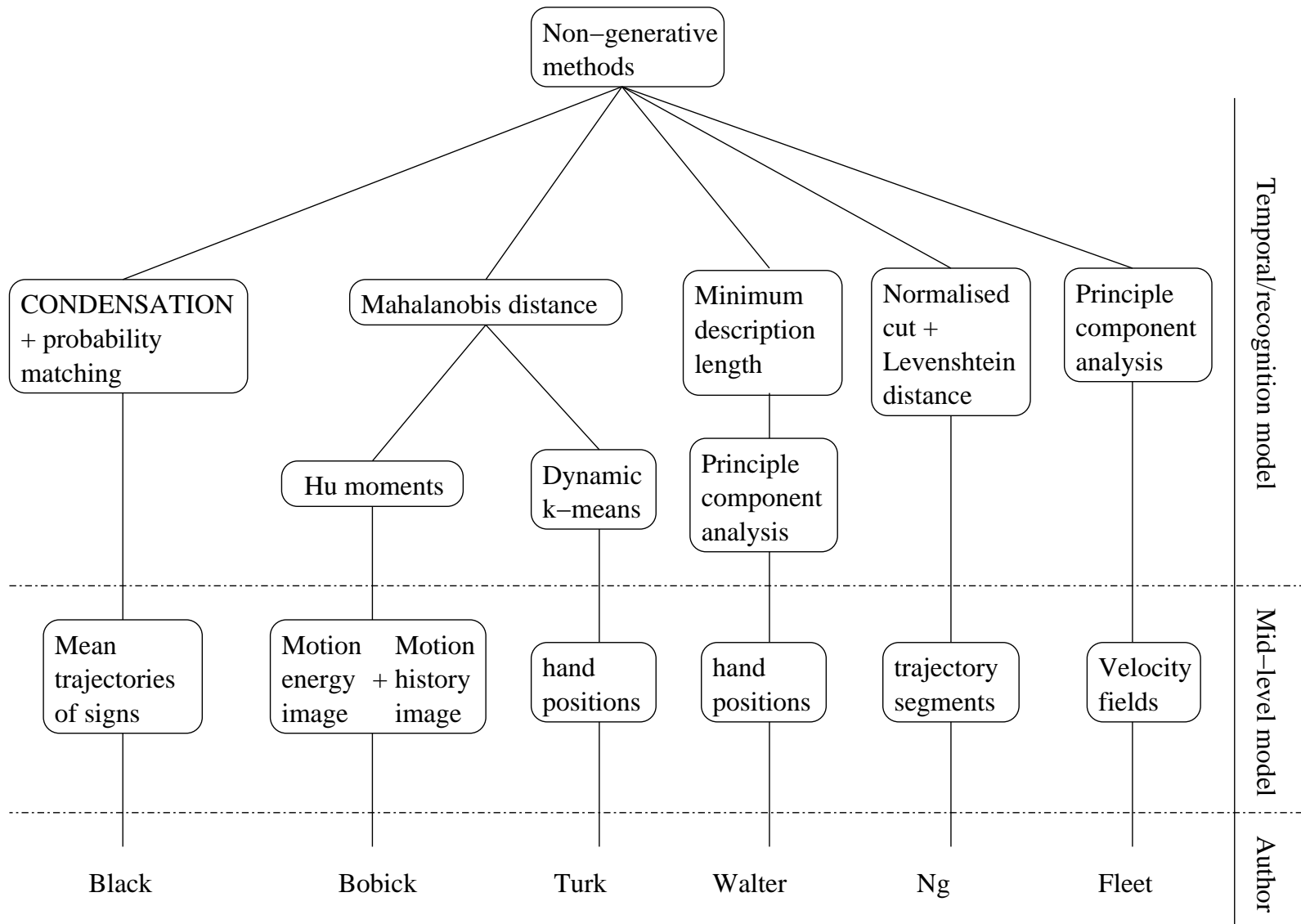


Figure 2.2: Non-generative methods in behaviour modelling

as it is done in particle filtering algorithms. If likelihood of this state is too small, the previous step is tried again up to a fixed number of tries. If the likelihood is still too small, a random state is chosen. A set of new weighted states are generated using this method by choosing different current weighted states. The gestures are then recognised using these weights.

Bobick and Davis [10] use a combination of motion energy image and motion history image to store templates of gestures in a database. The motion energy image describes the distribution of motion by summing the square of thresholded differences between successive pairs of images [9]. This gives blob-like images describing the areas where motion has been observed. The motion history image is an image template where the pixels represent the recency of the movement at the corresponding position. This image represents how a gesture is performed. The motion energy image and the motion history image form a vector that describes a gesture. The gestures are then discriminated using Hu moments on the motion energy image. These moments and other geometric features form the parameters of the motion energy image [9]. This is specific to their application and can be done using other methods for different applications. The recognition is performed by comparing a gesture with the stored parameters using Mahalanobis distance. If this distance is less than a threshold, the similarity between motion history images is used to make the decision. If several gestures are selected, the one closest to the motion history image stored in the database is chosen. This approach has been extended in [26] by using a hierarchical motion history image instead of a fixed size one in order to improve recognition properties.

Hong, Turk and Huang [42] recognise gestures of a user interacting with a computer by using finite state machines in order to model those gestures. The user's head and hands are first tracked by the algorithm proposed by Yang *et al.* [99]. The spacial data are represented by states that are computed using dynamic k -means. The Mahalanobis distance is used for the clustering. During the clustering, each time the improvement is small, the state with the largest variance is split into two states if the variance is larger than a previously chosen threshold. Therefore, the number of clusters increases until each cluster has a variance smaller than the threshold. The sequences of clusters during training gestures are then entered manually. The range of time we can stay in a state is then associated with each state. For instance figure 2.3 represents a sequence like $\{0,1,1,1,1,2,2,2,2,2,2,2,2,1,1,0\}$ and the user can stay between 2 and 3 cycles in

the state 1 during the recognition phase. The recognition of the gesture is done by simply following the links in the finite state machine. If the hand is close to the centroid of a state during a particular number of cycles, then the algorithm moves to the next state. If two gestures are recognised at the same time, the gesture which was closest to the centroids of the clusters is chosen.

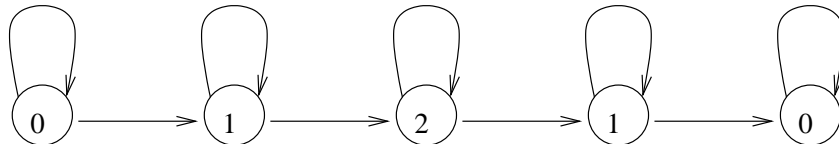


Figure 2.3: Representation of a gesture by a sequence of states in [42]. Each state represents a cluster amongst a set of cluster obtained from a trajectory of a gesture. The sequence is entered manually and the time spent in each state is associated with the state. The arrows represent the possible transitions of states between two cycles of the recognition phase.

In [95], Walter *et al.* model gestures by groups of trajectory segments. The trajectory segments are extracted by detecting discontinuities in the gesture trajectory. After normalising the trajectory segments, their dimensions are reduced using a principle component analysis. Clusters are then extracted from the component space using an iterative algorithm based on minimum description length. The clusters form atomic gesture components. There is a parallel between groups of trajectory segments and the actions or visual units we want to extract from the video sequence. However our segmentation and grouping algorithms are both different.

In [71], Ng and Gong use another algorithm to group trajectory segments. They use the Levenshtein distance to compare two trajectory segments. This distance is based on the dynamic time warping algorithm and a reinterpolation of the trajectory segments. An affinity matrix is constructed by comparing the segments two by two. Their unsupervised version of the normalised cut algorithm is then used on this affinity matrix to cluster the trajectory segments. An optimal threshold of the normalised cut algorithm is found maximising the intra-cluster affinity while minimising the inter-cluster affinity.

2.4 Generative methods

2.4.1 Sample based methods

Sample based methods reuse frames from an original video sequence to generate new sequences. A tree representation of the literature on this class of generative methods can be found on figure 2.4.

Schödl *et al.* [86] introduce the concept of video textures. Their aim is to generate an infinitely long video sequence based on frames from an existing video sequence. Loops in the video sequence are created by jumping from one frame to another in the original video sequence. The chosen frames are selected to exhibit similar appearance and dynamics while avoiding dead ends in the generated video sequence.

Graf and Cosatto [37] use a database approach to create a talking head. Using a pre-recorded video sequence, features such as the position of the head or the size of the mouth are computed. For each frame, those features are recorded in a database along with the corresponding phoneme and the appearance of the mouth. The latter is modelled by a local linear embedding [83], which is a compression technique and provides a reconstruction of the image of the mouth close to the original image. Their aim is to create new video sequences constrained by a sequence of phonemes describing the text to be pronounced by the talking head. The new sequence takes images from an existing video sequence and overlays images of the mouth found in the database. A Viterbi search is used to find the closest appearance that satisfies the constraints and produces smooth articulation.

Kovar *et al.* [55] creates realistic motions of 3D skeletons from motion capture data. The original motion capture data is segmented and the segments are stored in a database. A measure of similarity is computed between each pair of frames. Local minima of this similarity measure that have a value less than a selected threshold are computed. The corresponding pairs of frames are selected to construct a motion graph that represents how we can move from one capture sequence stored in the database to another. Transition sequences between stored captured sequences are also synthesised by a linear blending technique, stored in the database and added to the motion graph. The motion graph is then pruned to avoid dead ends and to ensure that arbitrary long streams can be generated while using most of the sequences stored in the database. Sequences of 3D skeletons

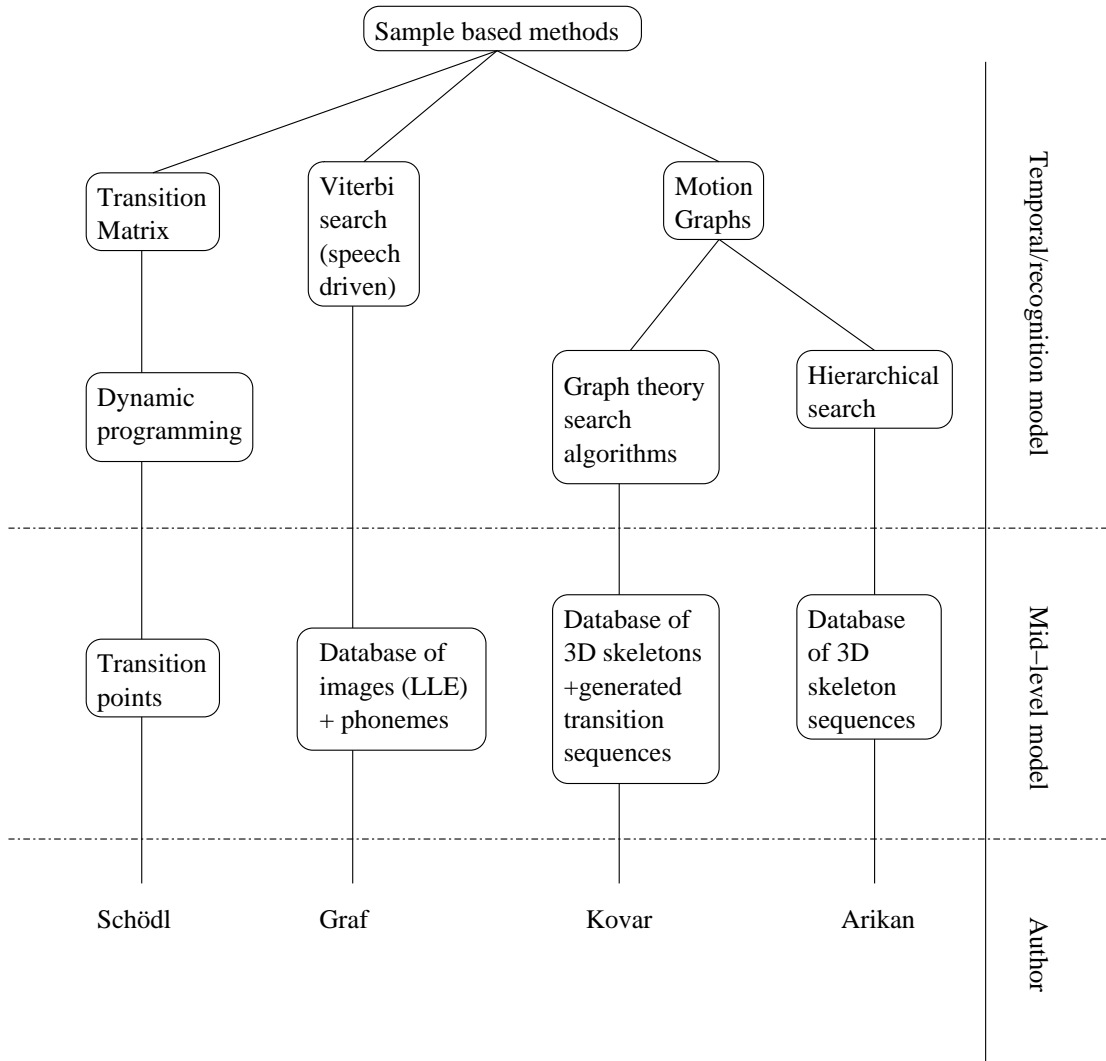


Figure 2.4: Sample based methods in behaviour modelling

can then be generated by moving along the motion graph. The motion graph can also be used to find the best motion between two frames given by the user. This is done by standard graph theory algorithms that minimise a metric between two nodes on the motion graph. Metrics such as the length of the generated sequence are used. This method can be used by animators to generate realistic body motions by only specifying how the body should look for a few frames in the sequence.

Arikan *et al.* [2] use a hierarchical motion graph to model trajectories of 3D skeletons. As in [55], a database of capture motion data is used. The hierarchy of motion graphs is constructed so that each level represents a broader view of the motion graph of the next level. Therefore, higher levels of the hierarchy of motion graphs contain fewer nodes, each node being a representation of a group of nodes in the next level. User constraints for generating new sequences (such as contact constraints between the feet and the floor) can be applied by restricting a search to valid path examples and mutating paths in a way that preserves the constraints. The search uses high level graphs to find crude solutions while refining the solution using low level graphs.

2.4.2 Appearance model based methods

Figure 2.5 summarises some of the techniques using appearance models. Such models represent the shape and texture of an object (see section 3.4), and have been found to be very effective for synthesising faces and facial behaviour.

Magee and Boyle [62] built a behaviour classifier called “history space classifier”. The history space is constructed by a succession of a spacial and a temporal operator. The spacial operator selects the nearest prototype using a winner take all approach. The temporal operator used is simply a multiplication by a weight which is the reciprocal of the time spent since the prototype has last been selected [60]. A more complex temporal operator could be used such as a set of leaky neurons as has been done by Sumpter and Bulpitt [92]. A modified version of the expectation maximisation introduced by Cootes and Taylor [23] is then used to model the point distribution function. In order to decrease the computational load of the algorithm, a principle component analysis is applied before the expectation maximisation algorithm so that the size of the data set is reduced. The resulting point distribution function is then used to predict the next state in the sequence.

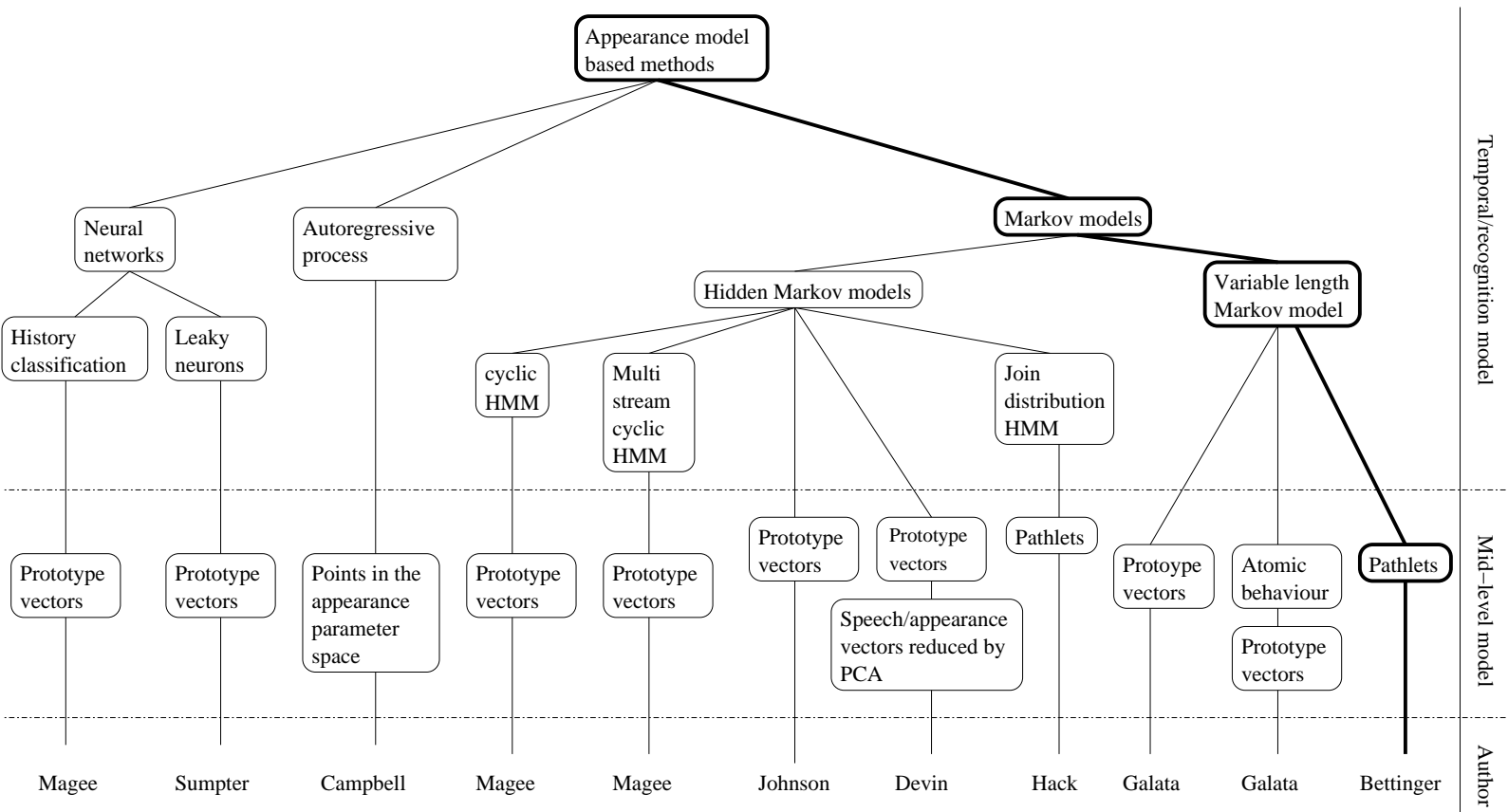


Figure 2.5: Appearance model based methods in behaviour modelling. The branch corresponding to our model is enhanced using bold lines.

Magee and Boyle [61] are using cyclic hidden Markov models in order in conjunction with Black and Jepson's variant of the particle filtering algorithm. Their aim is to recognise cows having difficulty walking. Figure 2.8(d) shows how hidden nodes are connected in a cyclic hidden Markov model. Those models are good at describing the cyclic motion that can be observed from a cow's leg. Two cyclic hidden Markov models are used to model the motion of cows walking. One learns a typical behaviour of a healthy cow and the other one learns a lame behaviour. The cyclic hidden Markov models are integrated in the re-sampling particle filtering framework so that the model that described a test behaviour more accurately dominates over time. The decision is then based on the number of samples produced for each model. In [61], Magee *et al.* also use a multi-stream cyclic hidden Markov model to model the two motions with only one stream of hidden states.

In [51], shapes are approximated by splines. The parameters controlling the splines as well as their speed are first clustered into prototype vectors using a competitive learning neural network. A compressed sequence derived from the prototype vector sequence is learnt using a Markov chain. A cubic Hermite interpolation is used along with the learnt Markov chain to recover the temporal structure of the sequence before compression and to extrapolate a behaviour. Furthermore, for generation purposes, a single hypothesis propagation and a maximum likelihood framework are described. During the generation, states of the Markov chain are chosen according to the state of the shape of a tracked person. This can allow generation of a shape of a virtual partner driven by a tracked real person. In [28], Devin and Hogg added sound and appearance to the framework in order to demonstrate that producing a talking head is possible.

Galata, Johnson and Hogg [33] also split trajectories into prototypes in order to model movement. The structure of the movements is then learnt using a variable length Markov model. As this approach seems to be one of the state of the art in behaviour modelling, we studied its components, in particular the variable length Markov model (see chapter 6). A prototype clustering is performed using a k -means like algorithm. The sequence of prototypes is then learnt using the variable length Markov model. The advantage of this approach is that it can model joint behaviour of people [52]. This is an essential characteristic for building a realistic human-computer interface. In [34], sequences of prototypes delimited by key-frames prototypes form atomic behaviours. The variable length

Markov model can learn those atomic behaviours instead of directly learning the prototypes, thus producing a higher level model.

Hack *et al.* [40] model trajectories by a sequence of pathlets. The dimensions of the pathlets observed in a training sequence are reduced using a principle component analysis. Using a Gaussian hidden Markov model, pairs of consecutive pathlets are clustered. New sequences of pathlets can then be created by sampling pairs of pathlets from the model, constrained by the choice of the first pathlet being the second pathlet of the previous sampled pathlet pair. This assures continuity in the sampling of pathlets through the generated sequence.

Campbell *et al.* [17] introduce another way of generating video sequences of faces based on an existing video clip without direct reuse of the original frames. They encode frames from the original sequence using an appearance model and a trajectory is obtained in the parameter space. This trajectory is then learnt using a second order autoregressive process. An autoregressive process predicts the position of a point \mathbf{y}_k in the appearance parameter space, given the two previous points \mathbf{y}_{k-1} and \mathbf{y}_{k-2} where \mathbf{k} represents the frame number, using the equation:

$$\mathbf{y}_k - \bar{\mathbf{y}} = \mathbf{A}_2 (\mathbf{y}_{k-2} - \bar{\mathbf{y}}) + \mathbf{A}_1 (\mathbf{y}_{k-1} - \bar{\mathbf{y}}) + \mathbf{B}_0 \mathbf{w}_k \quad (2.1)$$

where $\bar{\mathbf{y}}$ is the limit of the mean value of \mathbf{y}_k as \mathbf{k} tends to infinity, \mathbf{w}_k contains white noise ($\mathbf{w}_k \sim \mathcal{N}(0, 1)$), \mathbf{A}_2 , \mathbf{A}_1 and \mathbf{B}_0 are parameter matrices. $\bar{\mathbf{y}}$, \mathbf{A}_2 , \mathbf{A}_1 and \mathbf{B}_0 are learnt from the original data set. The learning method used in this work is due to Reynard *et al.* and is described in [79]. Given two initial points in the parameter space, a new trajectory can be generated by applying equation 2.1 repetitively. This new trajectory can be used to synthesise a video sequence of a face. They use this model in [74] and [38] to create an expression space that can be easily displayed and to help animators to generate new video sequences of expressions from an intuitive interface.

2.4.3 Talking heads

An extensive number of articles try to solve the problem of generating realistic talking heads. A talking head uses external inputs to produce the video sequence of a face. Such external inputs can be a phoneme sequence or an audio track that corresponds to the text that have to be pronounced by the head [73, 67, 35, 44, 72, 31, 3, 43, 16]. Here we concentrate only on the methods using appearance

models.

Theobald *et al.* [93] use an appearance model to synthesise videos of a talking head based on a sequence of phonemes. The sound track is first segmented into phonemes. A set of codebook vectors is created from some training videos of someone speaking with a neutral expression. Each codebook vector encodes a phoneme and the corresponding appearance parameters. In order to create the talking head, a sequence of phonemes is given as an input to the system. This sequence can either be given manually or generated from a given audio track. A measure of similarity of phonemes and their context is used to find the closest codebook vectors to the current phoneme. The appearance parameters are then extracted from the first few codebook vectors and a weighted average used for synthesis. The resulting appearance parameters from each phoneme are concatenated together. The result is then smoothed by interpolating the sequence of appearance parameters using a smoothing spline. The parameters are synthesised back to a video sequence to produce a photorealistic head talking with a neutral expression.

Cosker *et al.* [25] present a talking head based on hierarchical non-linear speech-appearance models. Instead of using a standard appearance model, a hierarchical facial model is used. Several parts of the faces are modelled separately using an appearance model for each. For instance, the mouth has its own model and should capture all the possible mouth variations, even the fine details that are harder to capture with a appearance model of the whole face. This approach is similar to [94]. The appearance vectors corresponding to the parts are concatenated. The distribution of the vectors is then approximated by a mixture of Gaussians using the EM algorithm [27]. The 40ms voice data corresponding to each frame is encoded in the same way as [28] using a Mel-Cepstral analysis. It is concatenated to the appearance vector in the corresponding cluster. Each cluster is then modelled by a principle component analysis. For the synthesis, a sound sample of 40ms is presented to the system. The cluster that is closest to that sound is used for synthesis. The Mahalanobis distance is used to compute distances between the sound sample and the parameters of sound corresponding to the mean of each cluster. A linear relationship is then assumed within the cluster to map the speech parameters to the appearance parameters. After smoothing the trajectory of the appearance parameters, the face is finally synthesised by combining the reconstruction of the different appearance models used

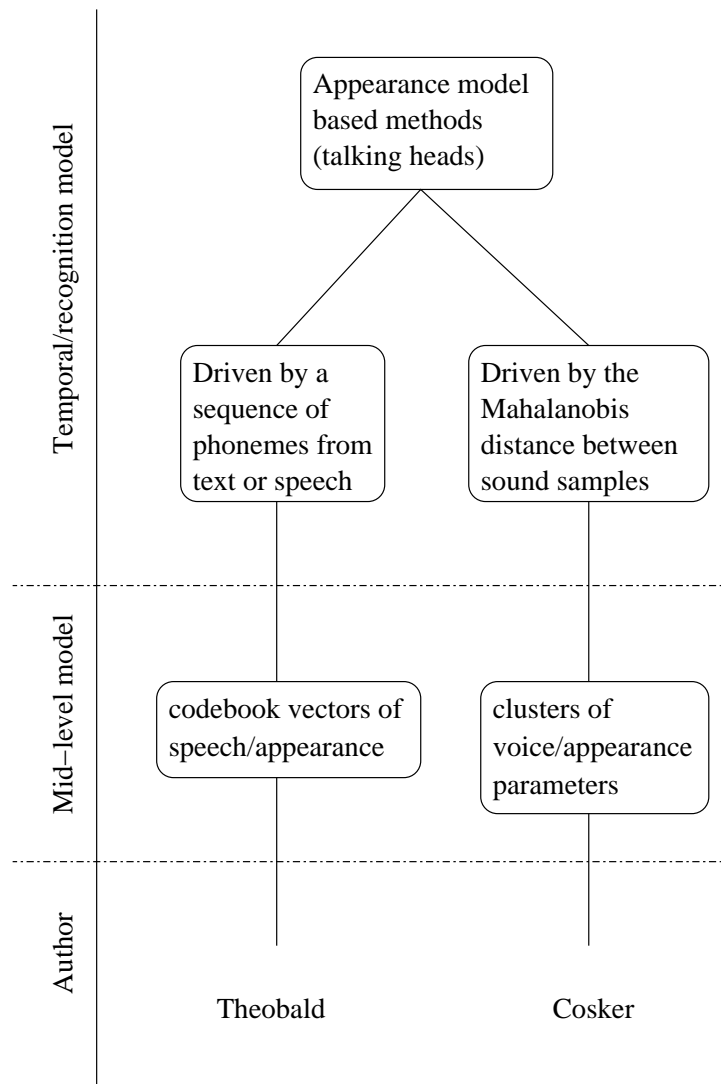


Figure 2.6: Appearance model based methods for talking heads.

(face, mouth and eyes).

The two works can be classified as appearance model based methods in figure 2.6.

2.4.4 Other generative methods

Other methods that use neither appearance model nor samples from original video sequences are described in this section. A tree representation of this part of the literature can be found in figure 2.7.

Saisan *et al.* [84] use autoregressive models to recognise dynamic textures such as waterfalls, smoke, waves or fire. After subsampling and reduction of the frames of the video sequences, a principle component analysis or an independent component analysis is applied to the set of frames to reduce their dimensionality. The dynamics of the process are then learnt from the reduced data in such a way that it produces a canonical representation of the data. Several distances between canonical representations of autoregressive models are tested with both the independent component analysis and the principle component analysis representation. Best results were obtained with the principle component analysis using Martin's distance [66]: a recognition rate of 89.5% has been achieved on a database of 200 dynamic textures.

Jebara and Pentland [48] propose an Action-Reaction Learning (ARL) in order to synthesise human behaviour. After using colour classification [47], blobs corresponding to the positions and orientations of the head and hands of a user are extracted using an expectation maximisation algorithm. A Kalman filter is used to improve the tracking of the blobs. A time series approach is used to map past to future. A window is chosen to represent the 128 previous frames (about 6.5 seconds of video). This gives a series of vectors of dimension 3840, that is 15 parameters for the blobs by 2 individuals by 128 frames. The dimensionality of these vectors are then reduced using a principle component analysis to form a 40 parameter vector. The most recent vector and the most recent velocity are then concatenated to this vector to form the input vector of the system. This input vector encodes the past action of each frame. The corresponding output vector is given by the 30 dimensional vector representing the current frame. The conditional expectation maximisation algorithm [49] is then used to find the probability of the location and direction of the blobs given the compressed past vector. After having observed the interaction between two users, the system is the able

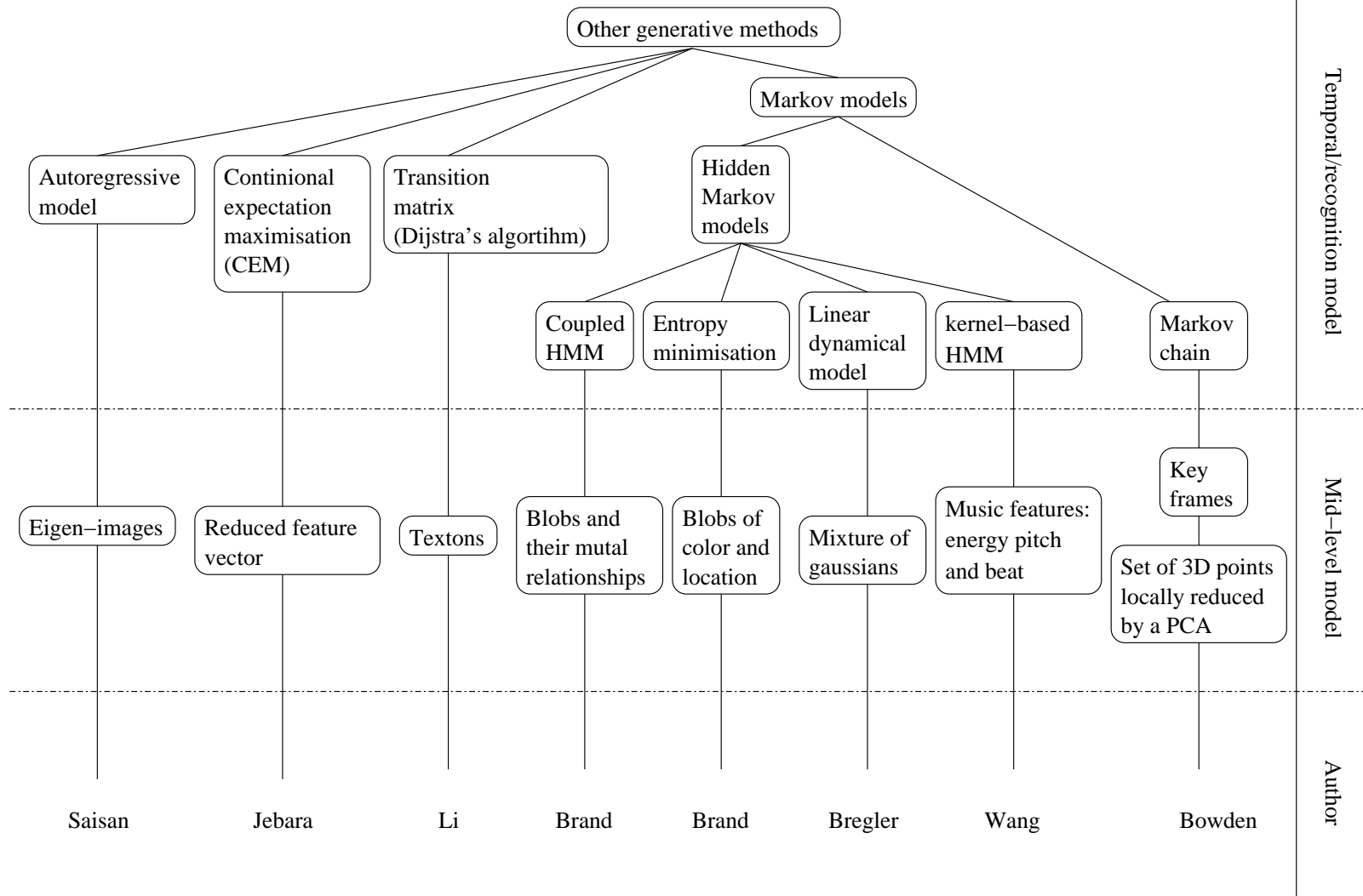


Figure 2.7: Other generative methods in behaviour modelling.

to display blobs that interact in a meaningful sense with a single user.

Bowden [11] models a sequence of captured 3D body movements and learns the associated motions. The set of 3D points is first modelled by a principle component analysis and then clustered in the component space. Each cluster is then modelled separately by a principle component analysis, in contrast with the global principle component analysis usually used in point distribution models. Each principle component models a part of the trajectory of the 3D points. The motion of the points is then modelled by a Markov chain that gives the probability of moving from one centre of the clusters to another one. New motions can be synthesised by generating a sequence of cluster centres to be used as key frames. A smooth trajectory of 3D points is obtained by interpolating between the key frames.

Li *et al.* [58] use textons to model behaviour of 3D skeletons. The dimension of the 3D capture data is first reduced by a singular value decomposition. They automatically segment a stream of 3D capture data to find what they call a texton. It represents an action or a part of an action performed by the actor in the 3D capture data sequence. A texton is modelled by an autoregressive model of hidden state variables that are projected back onto the space of 3D skeletons (this is similar to [84]). A maximum likelihood framework is used to segment the stream. Considering the transition points and the labels of the segments as hidden variables, an expectation maximisation algorithm [27] is used to solve the maximum likelihood problem. Once each texton is modelled, a transition matrix is created by counting the frequencies of the textons. Generating textons is only valid locally and the generation quickly deviates from a realistic behaviour. In order to overcome the problem, Li *et al.* constrains the generation of textons by specifying the starting and ending two frames during the generation. This is done by solving a system of linear equations, but only copes with given realistic poses. Otherwise the generated texton becomes unrealistic (this is similar to the over-constrained problem that we describe in section 7.4.2). A new sequence can then be generated by sampling new textons accordingly to the transition matrix and constraining any new texton to begin with the last generated frame. Finally, new sequences can also be generated by providing the starting and ending two frames. The optimal sequence of textons used is computed from the transition matrix using Dijkstra's algorithm [24].

Bregler [15] uses a hierarchical framework to recognise human dynamics in

video sequences of people running. The framework can be decomposed into four steps: the raw sequence, a model of movements by mixture of Gaussians, a model of linear dynamics and a model of complex movements. The basic movements are modelled with a mixture of Gaussians using the expectation maximisation algorithm on a probabilistic estimation of the motion based on the gradient of the raw sequence. The Gaussians are then grouped into blobs. Each blob is tracked by a Kalman filter. The Kalman filter is chosen to model a movement with constant velocity because we do not know the specific motion of each blob. A cyclic hidden Markov model is then used to model the high level complexity of motion. The hidden Markov model is trained with an iterative procedure that requires an initial guess of the model parameters. This guess is obtained by dynamic programming and is fine-tuned by the procedure.

Brand and Kettner [13, 54] introduce a new framework to learn hidden Markov models. Instead of the conventional Baum-Welch algorithm based on expectation maximisation, their algorithm is based on entropy minimisation. In practice, the M-step of the expectation maximisation algorithm is modified to minimise the entropy of the data, the entropy of the model distribution and the cross-entropy between the expected statistic of the data and the model distribution instead of maximising the likelihood. This method gives deterministic annealing within the expectation maximisation framework and turns it into a quasi-global optimiser. It has been tested on the learning of the activity of an office. Contrary to the conventional way of learning hidden Markov model, this algorithm gives a learnt model with meaningful hidden states such as: enter/exit activity, whiteboard writing, use of the phone or use of the computer. The transition matrix obtained is sparse and thus shows that the structure of the scene has been discovered. It is also shown that it is more successful in detecting abnormal behaviour.

Brand *et al.* [14] claim that classical hidden Markov models are not good at correctly modelling interactions. Indeed, the Markov assumptions used to construct the models are false for most interaction data. They do not encode time properly. They investigate a new model called coupled hidden Markov model (see figure 2.8(c)) where a state does not depend only on the previous state but on several previous states that are intuitively modelling the state of each actor of the interaction. They use the entropy minimisation algorithm previously described to train these coupled hidden Markov models [12]. They compare the

classical hidden Markov model (see figure 2.8(a)), a linked hidden Markov models introduced by Saul and Jordan [85], which models balance between actors of an interaction (see figure 2.8(b)), and the coupled hidden Markov models (see figure 2.8(c)). The data set used is a set of Thai Chi gestures. It is shown that coupled hidden Markov models outperform the two other models. Linked hidden Markov models were generally better than classical hidden Markov models except for particular moves where the dynamics of the gesture cannot be capture correctly.

Wang *et al.* [98] synthesise dynamic sequences of a virtual conductor using a music track as an input to the system. The joint input and output distribution sequence is modelled by a kernel-based hidden Markov model (KHMM). The joint probability of the input and output given the hidden state is estimated by a mixture of Gaussians. The parameters of this mixture are computed using the EM algorithm [27]. Features such as energy, pitch and beat are used as an input, and positions and velocities of 15 markers on the conductor body are used as an output. The training data set is acquired from a real conductor performing conducting gestures on several types of music. Once the model has been trained, the probability of the output given the input can be computed using a maximum likelihood framework. The resulting output sequence exhibit the global dynamics of the music while preserving the fine details of the conducting gestures.

2.5 Discussion

Figure 2.5 shows how our method [6, 5] compares to the literature. By focusing on the last three branches corresponding to the variable length Markov model branch, a progression can be seen. Galata *et al.* first used a variable length Markov model directly on the clustered prototypes of their system. However, in order to model longer-term behaviours, they had to introduce an atomic behaviour level to their model (which consists of a sequence of a few prototypes). Our approach is to model atomic behaviours without the drawback of using prototypes. Indeed, a prototype raises problems when we want to synthesise new behaviours. Each time we generate a prototype vector, the synthesis corresponds exactly to the prototype. So the same frame is generated many times in the output video sequence. This also highlights the need of a large number of prototypes to model a trajectory in the appearance parameter space.

Our alternative is inspired by the work of Walter *et al.* . They represent

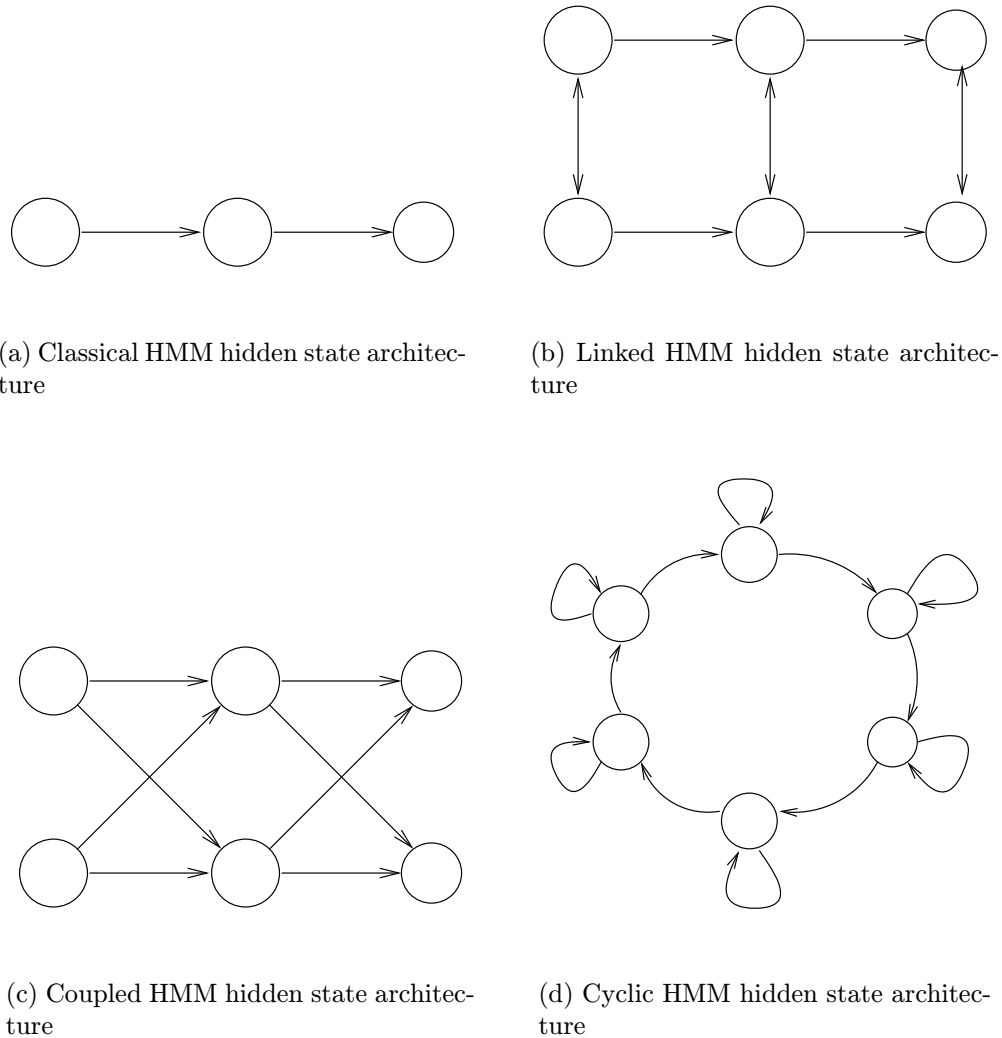


Figure 2.8: Graphical representation of HMM, HMM, coupled HMM and cyclic HMM. Figure 2.8(a) describes the representation of hidden Markov models. The states are linked with the conditional probabilities of going to a state given we were in a previous state. A state represents both actors of an interaction. The hidden Markov chain is presented unrolled in time. Figure 2.8(b) represents the linked hidden Markov chains where states are also linked with the joint probability of both actors being in the linked states. Figure 2.8(c) represents the coupled hidden Markov model where the states depend on the previous states of both actors in the interaction. Finally, figure 2.8(d) models cyclic motions. The hidden states are temporally linked, each one of them representing a part of a cyclic motion.

trajectories of hands by atomic trajectories and model those using a principle component analysis. Our pathlet group model is the transposition of this model to the appearance parameter space without, however, using the same algorithms to build the model.

Chapter 3

Statistical appearance models

3.1 Introduction

Here we give a brief introduction to statistical appearance models. It describes how one can build a model of the appearance of an object given a set of manually labelled images of this object. Our eventual aim is to extend such a model to be able to model objects in video sequences.

The second part of the chapter describes how an active appearance model is used to track the face through a long video sequence, in the face tracking module. This is a description of the first module shown in the overview of the system (see figure 3.1).

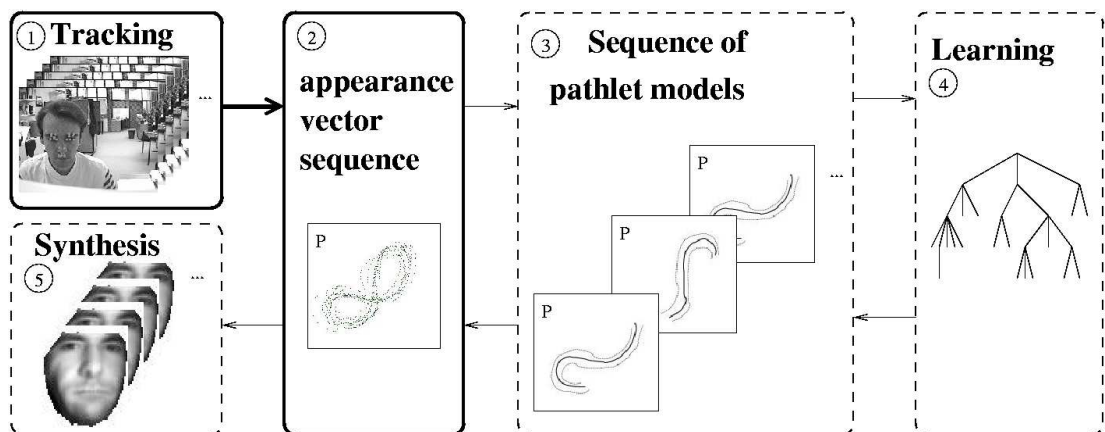


Figure 3.1: Overview of the components of the model. This chapter explains how the face tracking module works.

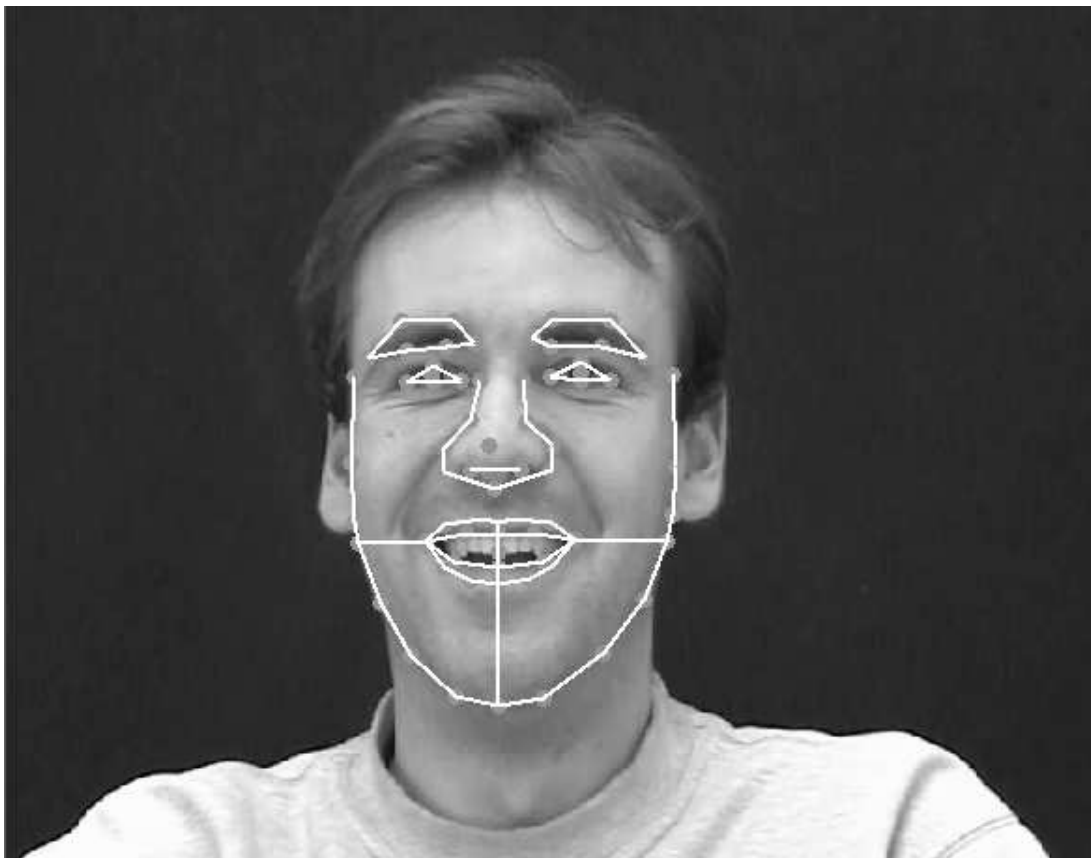


Figure 3.2: Example of hand labelled face. The points are manually put on features such as corners of the eyes and mouth, and along boundaries.

3.2 Statistical shape models

Statistical shape models represent the structure of an object using a set of landmark points. They are trained using manually annotated images. The manual labelling is a way of including human knowledge into a learning mechanism. Figure 3.2 shows an example of hand labelled image. The shape is described by a vector \mathbf{x} that contains the coordinates of each point of the shape.

The first step of the statistical shape model is to align the shapes found in the training set. This is done by an approach called Procrustes analysis [29]. This algorithm is iterative and reduces the sum of the distances between each shape to the mean shape. On its completion, all the shapes have the same centre of gravity, scale and orientation.

The variation in shape is then estimated by applying a principle component analysis (PCA) to the vectors representing the aligned shapes. The mean of these

s vectors is computed:

$$\bar{\mathbf{x}} = \frac{1}{s} \sum_{i=1}^s \mathbf{x}_i \quad (3.1)$$

as well as the covariance of the data:

$$\mathbf{S} = \frac{1}{s-1} \sum_{i=1}^s (\mathbf{x}_i - \bar{\mathbf{x}})(\mathbf{x}_i - \bar{\mathbf{x}})^T \quad (3.2)$$

and finally the eigenvectors Φ_i of \mathbf{S} are computed with their associated eigenvalues λ_i .

In order to decrease the dimensionality of the data, the eigenvectors corresponding to the t largest eigenvalues are chosen, as they explain most of the variation of the dataset. A threshold f_v is previously chosen (usually 95% or 98%). We then compute t by taking the minimum integer where the equation:

$$\sum_{i=1}^t \lambda_i \geq f_v \sum_{i=1}^s \lambda_i \quad (3.3)$$

holds.

If we define $\Phi = (\Phi_1, \dots, \Phi_t)$, each vector \mathbf{x} in the training set can be approximate by:

$$\mathbf{x} \approx \bar{\mathbf{x}} + \Phi \mathbf{b} \quad (3.4)$$

where \mathbf{b} is defined by:

$$\mathbf{b} = \Phi^T (\mathbf{x} - \bar{\mathbf{x}}) \quad (3.5)$$

\mathbf{b} describes the shape \mathbf{x} . The approximation of the shape \mathbf{x} can be reconstructed with only \mathbf{b} , given that we know the model (that is, $\bar{\mathbf{x}}$ and Φ). Constraining the model to small variations allows it to generate only shapes that are similar to the training shapes. This can be done either by restricting the elements b_i of \mathbf{b} to vary between bounds (for instance $\pm 3\sqrt{\lambda_i}$) or by constraining \mathbf{b} to be in a hyper-ellipsoid:

$$\sum_{i=1}^t \frac{b_i^2}{\lambda_i} \leq M_t \quad (3.6)$$

where M_t is a threshold chosen using the χ^2 distribution.

Figure 3.3a shows the first mode of variation of the model built on images of annotated faces.

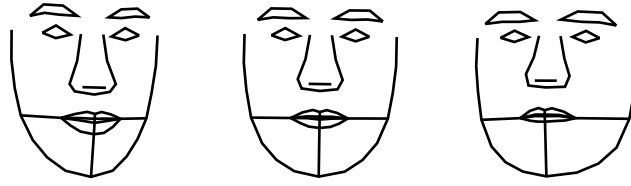


Figure 3.3: Example of the first mode of variation of the shape of a face. The parameter of \mathbf{b} corresponding to the largest eigenvalue, varies from $-3\sqrt{\lambda_1}$ to $3\sqrt{\lambda_1}$.

3.3 Statistical models of appearance

The aim of the statistical model of appearance is to add information about the texture of an object to the information about the shape of the object. The approach is the same as the statistical shape models. The model is learnt from a set of annotated images of the object.

First, a statistical shape model is built from the training set. New shapes \mathbf{x} can be generated by the equation:

$$\mathbf{x} = \bar{\mathbf{x}} + \Phi_s \mathbf{b}_s \quad (3.7)$$

where \mathbf{b}_s is a vector of shape parameters.

Then, given the mean shape we warp the training images into a shape-free patch of texture. Figure 3.4 shows how a face is decompose into a shape and a shape-free texture.

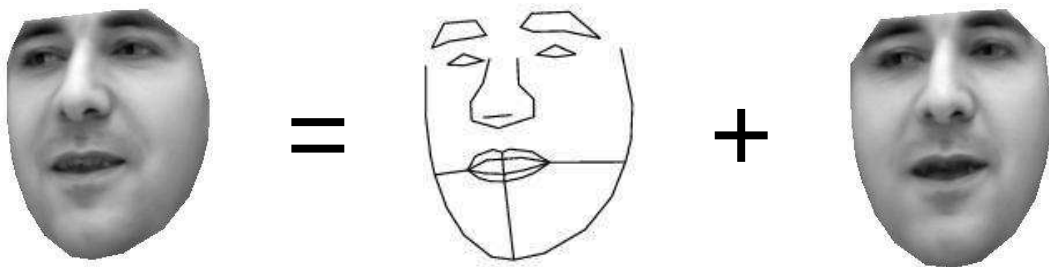


Figure 3.4: Decomposition of a face into a shape and a shape-free texture.

The set of shape-free textures is then normalised in order to reduce the effect of lighting. The pixel values are changed by a linear function so that the shape-free texture of a image is as close as possible to the shape-free texture of the

normalised mean.

A principle component analysis is then applied to the shape-free texture in order to model this patch by a linear equation:

$$\mathbf{g} = \bar{\mathbf{g}} + \Phi_g \mathbf{b}_g \quad (3.8)$$

where \mathbf{b}_g represents the parameters, as it was already the case with the statistical shape model and \mathbf{g} is a vector containing the pixel values of the shape-free texture patch.

\mathbf{b}_s and \mathbf{b}_g can then describe an object and its texture. Nevertheless, the two vectors can still be correlated. In order to reduce the dimensionality again, we compute the vector:

$$\mathbf{b} = \begin{pmatrix} \mathbf{W}_s \mathbf{b}_s \\ \mathbf{b}_g \end{pmatrix} \quad (3.9)$$

for each training image, where \mathbf{W}_s is a diagonal matrix of weights that allows to compare \mathbf{b}_s and \mathbf{b}_g . A third principle component analysis is applied to these data to obtain the model:

$$\mathbf{b} = \mathbf{Q}\mathbf{c} \quad (3.10)$$

where \mathbf{c} describes the new set of parameters that controls the model. For every \mathbf{c} , an object can be synthesised. This is an important characteristic of the model. It will allow us to synthesise behaviour once the proper parameter stream is found.

The figure 3.5 shows the first mode of variation of such a model.



Figure 3.5: Example of the first mode of variation of the face using a statistical model of appearance. The parameter of \mathbf{c} corresponding to the largest eigenvalue varies from $-3\sqrt{\lambda_1}$ to $3\sqrt{\lambda_1}$.

3.4 Active appearance model

The aim of active appearance search is to find the parameters of the statistical appearance model that best fits it to a previously unseen image. The active appearance model search is based on the idea that the reconstruction of the image should be close to the original image. We aim to minimise the difference between the target image and the synthesis of the model.

This minimisation is done by iteratively improving a current estimate of the parameters of the model. For each step, the algorithm uses the difference between the target image and the synthesis of the model using the current estimate of the parameters. This difference is represented in a residual vector:

$$\mathbf{r}(\mathbf{p}) = \mathbf{g}_s - \mathbf{g}_m \quad (3.11)$$

where \mathbf{g}_s denotes the intensities in the image warped from the current shape given by the model parameters \mathbf{p} to the mean shape and \mathbf{g}_m denotes the intensities in the synthesis of the texture of the model. The parameters \mathbf{p} are a concatenation of the appearance parameters \mathbf{c} , and the scale and the position of the model in the image. Figure 3.6 shows an example of the image used to compute $|\mathbf{r}(\mathbf{p})|$.



Figure 3.6: An example of difference image. The image on the left is the difference image of the target image and the model reconstruction of given parameters.

We seek to minimise the square of the norm of $\mathbf{r}(\mathbf{p})$:

$$\mathbf{E}(\mathbf{p}) = \mathbf{r}(\mathbf{p})^T \mathbf{r}(\mathbf{p}) \quad (3.12)$$

with respect to \mathbf{p} .

We model the residuals by assuming a local linear relationship between the parameters and the residuals:

$$\mathbf{r}(\mathbf{p} + \delta\mathbf{p}) = \mathbf{r}(\mathbf{p}) + \frac{\partial\mathbf{r}}{\partial\mathbf{p}}\delta\mathbf{p} \quad (3.13)$$

For each step, we select $\delta\mathbf{p}$ which minimises $\mathbf{x} \mapsto \mathbf{E}(\mathbf{p} + \mathbf{x})$. We require $\frac{d\mathbf{E}(\mathbf{p}+\mathbf{x})}{d\mathbf{x}} = 0$ for $\mathbf{x} = \delta\mathbf{p}$.

We have:

$$\mathbf{E}(\mathbf{p} + \mathbf{x}) = \left(\mathbf{r}(\mathbf{p}) + \frac{\partial\mathbf{r}}{\partial\mathbf{p}}\mathbf{x} \right)^{\mathbf{T}} \left(\mathbf{r}(\mathbf{p}) + \frac{\partial\mathbf{r}}{\partial\mathbf{p}}\mathbf{x} \right) \quad (3.14)$$

So:

$$\mathbf{E}(\mathbf{p} + \mathbf{x}) = \mathbf{E}(\mathbf{p}) + 2\mathbf{r}(\mathbf{p})^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right) \mathbf{x} + \mathbf{x}^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right) \mathbf{x} \quad (3.15)$$

This gives us the derivative of $\mathbf{E}(\mathbf{p} + \mathbf{x})$ with respect to \mathbf{x} :

$$\frac{d\mathbf{E}(\mathbf{p} + \mathbf{x})}{d\mathbf{x}} = 2\mathbf{r}(\mathbf{p})^{\mathbf{T}} \frac{\partial\mathbf{r}}{\partial\mathbf{p}} + 2\mathbf{x}^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{\mathbf{T}} \frac{\partial\mathbf{r}}{\partial\mathbf{p}} \quad (3.16)$$

By equating $\frac{d\mathbf{E}(\mathbf{p}+\mathbf{x})}{d\mathbf{x}}$ to zero, we obtain:

$$\mathbf{x}^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{\mathbf{T}} \frac{\partial\mathbf{r}}{\partial\mathbf{p}} = -\mathbf{r}(\mathbf{p})^{\mathbf{T}} \left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right) \quad (3.17)$$

That is:

$$\left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{\mathbf{T}} \frac{\partial\mathbf{r}}{\partial\mathbf{p}} \mathbf{x} = -\left(\frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{\mathbf{T}} \mathbf{r}(\mathbf{p}) \quad (3.18)$$

So $\delta\mathbf{p}$ can be computed using the formula:

$$\delta\mathbf{p} = -\mathbf{R}(\mathbf{p})\mathbf{r}(\mathbf{p}) \quad (3.19)$$

where:

$$\mathbf{R}(\mathbf{p}) = \left(\frac{\partial\mathbf{r}^{\mathbf{T}}}{\partial\mathbf{p}} \frac{\partial\mathbf{r}}{\partial\mathbf{p}} \right)^{-1} \frac{\partial\mathbf{r}^{\mathbf{T}}}{\partial\mathbf{p}} \quad (3.20)$$

Computing $\mathbf{R}(\mathbf{p})$ at each iteration is computationally expensive. We assume that this matrix can be considered approximatively constant, since it is computed in a normalised reference frame. So we compute it once from our training data. The equation becomes:

$$\delta \mathbf{p} = -\mathbf{R}\mathbf{r}(\mathbf{p}) \quad (3.21)$$

Given this model of image differences, we generate displacements $\delta \mathbf{p}$ around the optimal value for training images, one parameter at a time, in order to find the corresponding $\mathbf{r}(\mathbf{p} + \delta \mathbf{p})$ by synthesis and image difference. This gives us a set of pairs $(\delta \mathbf{p}, \mathbf{r}(\mathbf{p} + \delta \mathbf{p}))$. Those can be used to approximate the matrix $\frac{\partial \mathbf{r}}{\partial \mathbf{p}}$ by averaging and combining with a normalised Gaussian kernel w to smooth the result:

$$\frac{\partial r_i}{\partial p_j} = \sum_k w(\delta \mathbf{p}_{jk}) (r_i(\mathbf{p} + \delta \mathbf{p}_{jk}) - r_i(\mathbf{p})) \quad (3.22)$$

where $\delta \mathbf{p}_{jk}$ is the k^{th} displacement of the parameter j . The matrix \mathbf{R} can then be computed using equation 3.20.

Once the relationship between differences of parameters and differences of intensities in the images is computed, we can derive a search algorithm in order to locate the face. The search algorithm is summarised on figure 3.7.

Figures 3.8 and 3.9 show an example of the application of this algorithm.

Compute the difference $\delta \mathbf{g}_0 = \mathbf{g}_s - \mathbf{g}_m$
 Evaluate the current error $E_0 = \|\delta \mathbf{g}_0\|^2$.
 Compute $\delta \mathbf{p} = -\mathbf{R}\delta \mathbf{g}_0$.
 Set $k=1$.
 Let $\mathbf{p}_1 = \mathbf{p}_0 - k\delta \mathbf{p}$.
 Calculate the new error $\delta \mathbf{g}_1$.
 If $\|\delta \mathbf{g}_1\|^2 < E_0$:
 accept the parameter \mathbf{p}_1 .
 if converged, stop.
 otherwise go to the first step with \mathbf{p}_1 as the new estimate.
 Otherwise try k at 1.5, 0.5, 0.25, etc.

From Cootes *et al.* [22]

Figure 3.7: Active appearance model search algorithm.

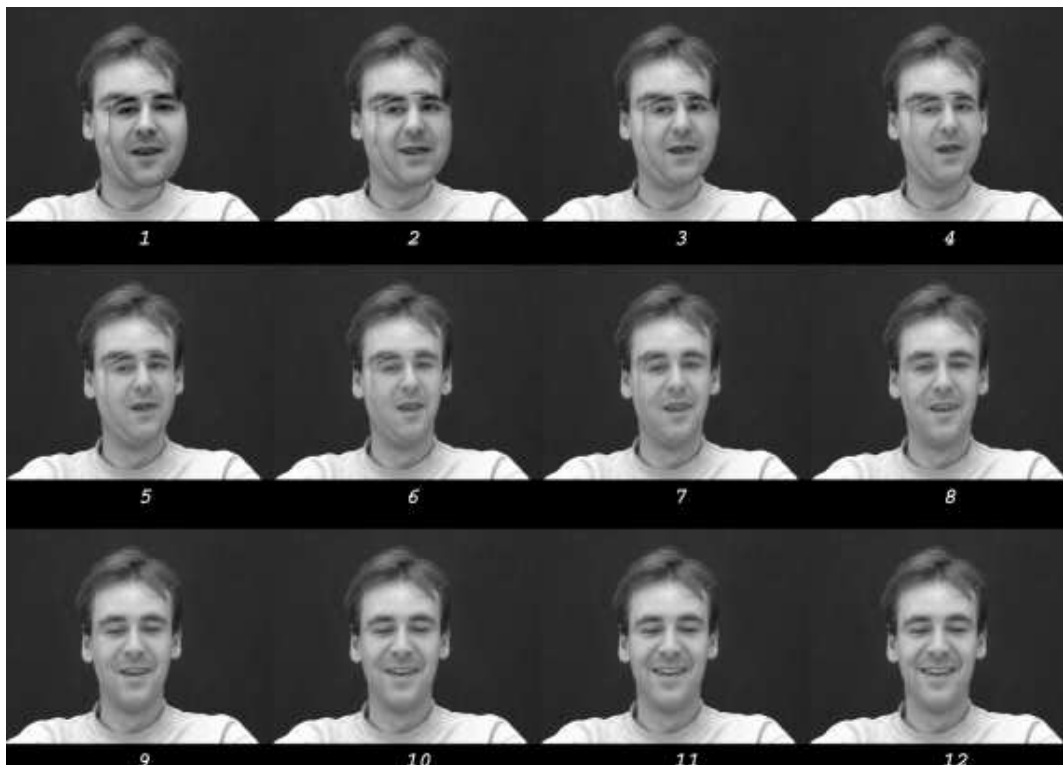


Figure 3.8: Example of application of the active appearance model search algorithm. The figures represent the synthesis of the parameters found after each step of the algorithm.

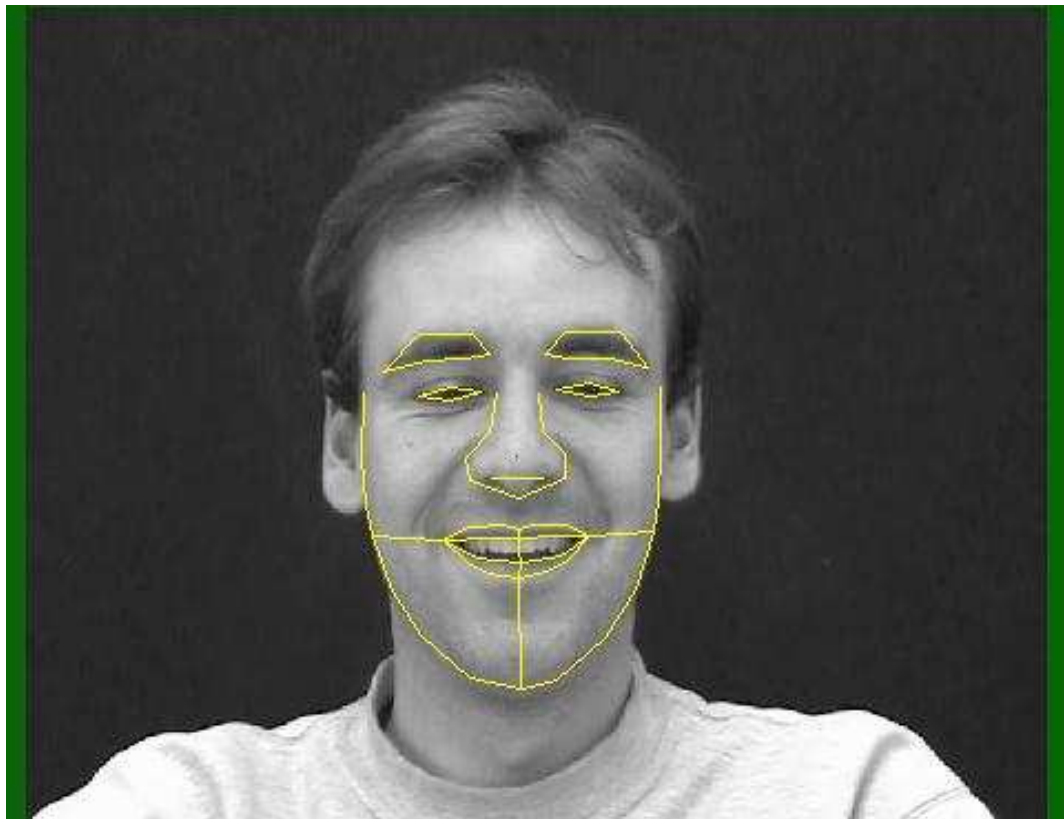


Figure 3.9: Example of result of the AAM search algorithm. The figure represent the synthesis of the shape found overlaid on the original image given to the algorithm. The different appearance parameters found at each steps are synthesised on figure 3.8.

3.5 The tracking module

Different approaches have been used in the literature to track deformable objects through a video sequence using the active appearance model or derived models. Such approaches include the work of:

- Edwards *et al.* [30], for tracking while recognising the identity of a face.
- Baumberg and Hogg [4], for tracking with a spline based active appearance model.
- Magee and Boyle [59], for tracking using multiple point distribution models.
- Stegmann [91], for tracking in real-time using the active appearance model.
- Matthews and Baker [68], for speeding up computation in the active appearance model search algorithm by using a compositional approach.
- Matthews *et al.* [69], for tracking using two active appearance models: one that evolves through time, increasing specificity while the other one keeps generality properties and allows convergence of the search algorithm.
- Jurie *et al.* [53], for tracking using appearance of an object under planar assumptions.

The active appearance model is a form of gradient descent, so converges to the nearest local minima. When tracking a video sequence it is usually sufficient to initialise the model with the result of the search in the previous frame. For relatively slow motions this is close enough to convergence, leading to accurate tracking. However, occasionally the movement between frames is sufficiently large that the active appearance model fails to follow the face, and falls into a non-optimal local minima.

To avoid this problem, we initialise the model at multiple starting points. Those starting points correspond to nodes on a grid placed on the image. The nodes are separated by 8 pixels, this distance being within the usual range of convergence [21]. The active appearance model search algorithm is performed for each initialisation, the result giving the smallest magnitude of the residuals is chosen. This has been found to lead to reliable tracking.

3.6 Conclusion

Models of appearance have been presented in this chapter. An associated algorithm, the active appearance model allows to search for instances of faces in an image.

The active appearance model search algorithm is a local optimiser, and is able to match well if initialised sufficiently close to the true optima. In sequences of faces, the difference between frames are usually sufficiently small that using the match to the previous frame as initialisation for the search in the next for satisfying tracking. However, large movements cause the search to fail. It can be recovered by using a global search.

The appearance model is generative. We can synthesise an image of a face given its appearance parameter vector.

The next chapter describes some of the video sequences of faces successfully tracked with the active appearance model. The video sequences constitute the training data our model should mimic.

Chapter 4

Description of the data

4.1 Introduction

This chapter explains how we can create an active appearance model that can track a face in a long video sequence. This approach is semi-automatic since we iteratively refine the appearance model by adding hand-labelled images in its training set. We then describe the video sequences acquired to assess our model of behaviour.

4.2 Annotation of the data

One important point when we want to use the active appearance model to track a face is to make sure that the model is general enough. Every frame in the video sequence has to be modelled, so the appearance model used should cope with all the expression changes in the video sequence. In order to make sure the model is general enough, we construct it from frames present in the video sequence that we aim to mimic.

We mark up the first frame manually, so that the tracker knows where the face is at the beginning of the sequence, thus avoiding an initialisation procedure. Then we mark up frames that look like extreme expressions such as a large smile or a surprised expression with the mouth open. These manually marked up frames are used to construct a first appearance model of the face. This model is used to track the face through the video sequence.

For each frame tracked, we can compute the sum of squares error of the fitting using formula 3.12 . The value of this error gives a quality of fit. During

the tracking, we monitor this value. If it rises above a given value, we assume that the search has failed.

The tracking typically failed on frames that contain expressions not present in the set of marked up frames used for training the appearance model. That suggested that the model was not generic enough for the video sequence. In order to make it more general, we marked up the first frame on which the tracker failed and added it to the training set of the appearance model. We can then track the sequence again with the new appearance model and repeat the procedure until the face is tracked through the whole sequence.

Figure 4.1 shows the final error that we obtained on a video sequence of a face talking. The large mean square errors after frame 85588 correspond to cases where a global search is needed. In practice, only 17 global searches have been applied for the face tracking on this video sequence, out of 88044 frames.

Figure 4.2 shows a visual comparison between an original frame and the reconstruction of this frame after the tracking. The reconstruction is still faithful although the error is one of the biggest error seen on figure 4.1. This shows that the sequence has been successfully tracked. The file `tracked/aam_orig.m1v` of the accompanying CD-ROM shows the reconstruction along with the original frames for a part of the original video sequence of figure 4.2.

Although a fully automated tracking system would be desirable, in this thesis we concentrate on modelling behaviour. We are happy to accept some manual intervention in the initial tracking of the training set.

4.3 Description of the data collected

In order to be able to test and assess our model of behaviour, we acquired different videos of individuals performing different tasks: shaking, changing expressions or talking to someone else.

We have selected those videos because they show increasing degree of structure in the way the action is performed. Shaking is a very structured behaviour. Changing expressions on the face at random is less structured. Finally, talking and listening to someone else without any constraints is the most general case.

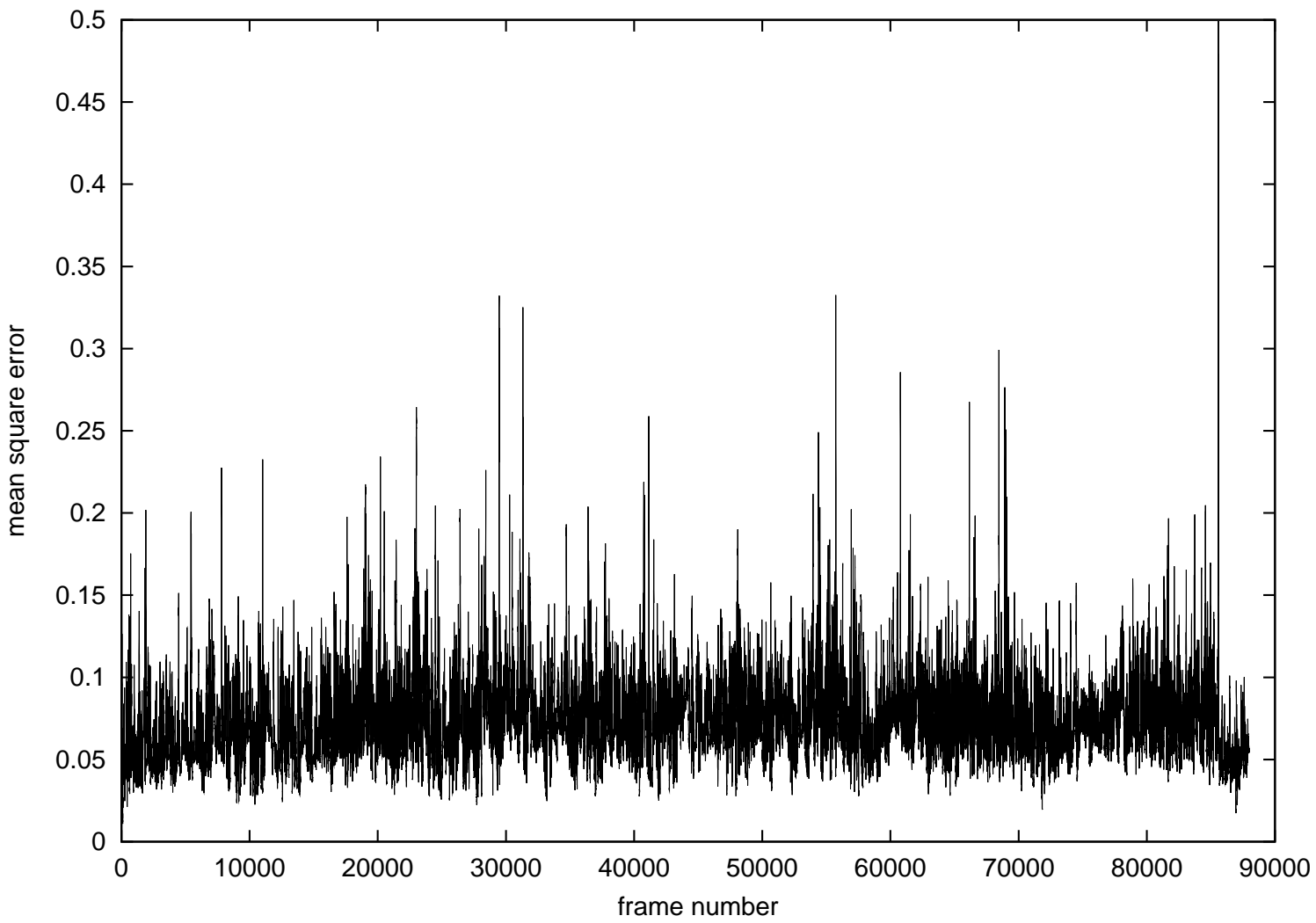


Figure 4.1: Final mean square error on a video sequence of a face talking.



Figure 4.2: Comparison between the synthesised version of the tracked parameters (left) and the face in the original frame of the video (right). This frame is the frame number 29480 of the video used to create the graph on figure 4.1. The corresponding mean square error on grey level values is 0.33 pixel.



Figure 4.3: Frames extracted from the video V1.

4.3.1 Video sequence V1: shaking

The first video sequence we used to assess our behaviour model is a video of a person gesturing “no” with their head.

This video has been shot with a Canon Powershot A80 digital camera. It contains 317 frames. Figure 4.3 shows some frames extracted from the video sequence. The file `examples/V1/V1_orig.m1v` of the accompanying CD-ROM shows the full sequence.

135 frames of this video have been marked up by hand using 15 points as depicted on figure 4.4. The hand labelled points as placed on the corners and on the centre of the mouth, on the eyes, on the nostrils and at the border of the face. The first mode of variation of the resulting appearance model can be seen on figure 4.5. The sequence has been successfully tracked using this model. The relative coordinates of the points as well as the pose, the scale and the position of the face, have been reduced to 7 parameters for each frame. The first 3 of those parameters represent the parameters controlling the expression of the face.



Figure 4.4: Hand labelling of a frame from video V1.



Figure 4.5: First mode of variation of the appearance model extracted from video V1.

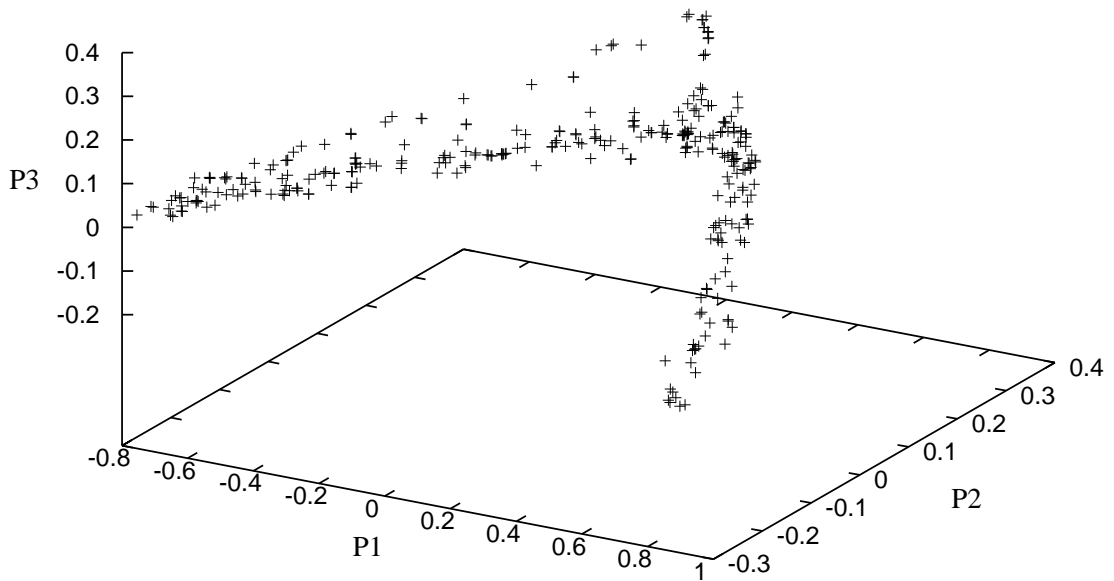


Figure 4.6: Projection of the first 3 appearance parameters extracted from video V1.

In this simple case, the parameters can be displayed on a graph. Figure 4.6 shows the points corresponding to the first 3 parameters extracted for each frame in the video V1. Figure 4.7 shows the projection of this graph onto the 2 first parameters, the two largest modes of variation of the face.

Figure 4.8 shows the synthesised version of the appearance parameters extracted from some frames of the video sequence V1.

The file `examples/V1/V1_track.m1v` on the accompanying CD-ROM shows the sequence of synthesised version of the tracked face for all frames. This video sequence can be visually compared to the original video sequence to verify that the face has been tracked correctly.

The file `tracked/V1_graph.m1v` on the accompanying CD-ROM shows the generated version of the video sequence V1 after tracking, along with the graph from figure 4.6 plotted in real time. For each frame, the point corresponding to the synthesised parameter is added.

Since the graph of appearance parameters can be easily plotted for this video, we choose this sequence to illustrate our algorithms in the following sections of the thesis.

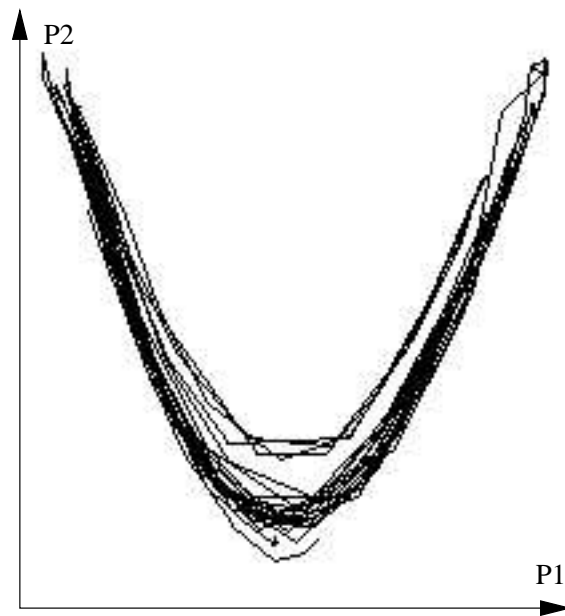


Figure 4.7: Projection of the first 2 appearance parameters extracted from video V1.

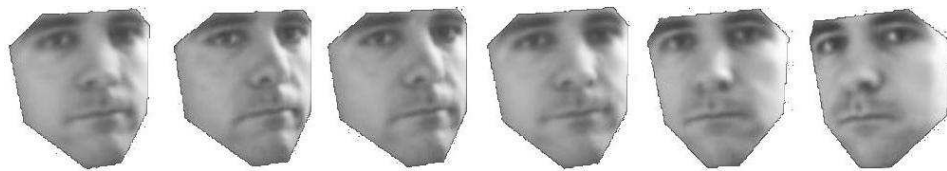


Figure 4.8: Frames extracted from the video V1 after tracking.



Figure 4.9: Frames extracted from the video V2.

4.3.2 Video sequence V2: change of expressions

The second video sequence we used is of a person changing expressions over time. The four expressions the face can switch to were:

- the neutral expression
- a smile
- a surprised expression
- a sad expression

The expressions changed several times through the video sequence. The sequence of expressions used was random, although some happened more than others. This video is therefore less structured than the video V1.

The video has been shot with the same digital camera as video V1. It contains 3943 frames. Figure 4.9 shows some frames extracted from the sequence (see also `examples/V2/V2_orig.m1v` on the accompanying CD-ROM).

Only 6 frames of this video have been marked up by hand using 43 points as depicted on figure 4.10. Extreme expressions have been marked up in the sequence and the appearance model was powerful enough to track effectively. The first mode of variation of the appearance model build from video V2 is shown on figure 4.11. The face has been successfully tracked through the video sequence. The relative coordinates of the points as well as the pose, the scale and the position of the face have been reduced to 7 parameters for each frame.

Figure 4.12 shows the synthesised version of the appearance parameters extracted from some frames of the video sequence V2.

The file `examples/V2/V2_track.m1v` on the accompanying CD-ROM shows the sequence of synthesised version of the tracked face for all frames from video V2.

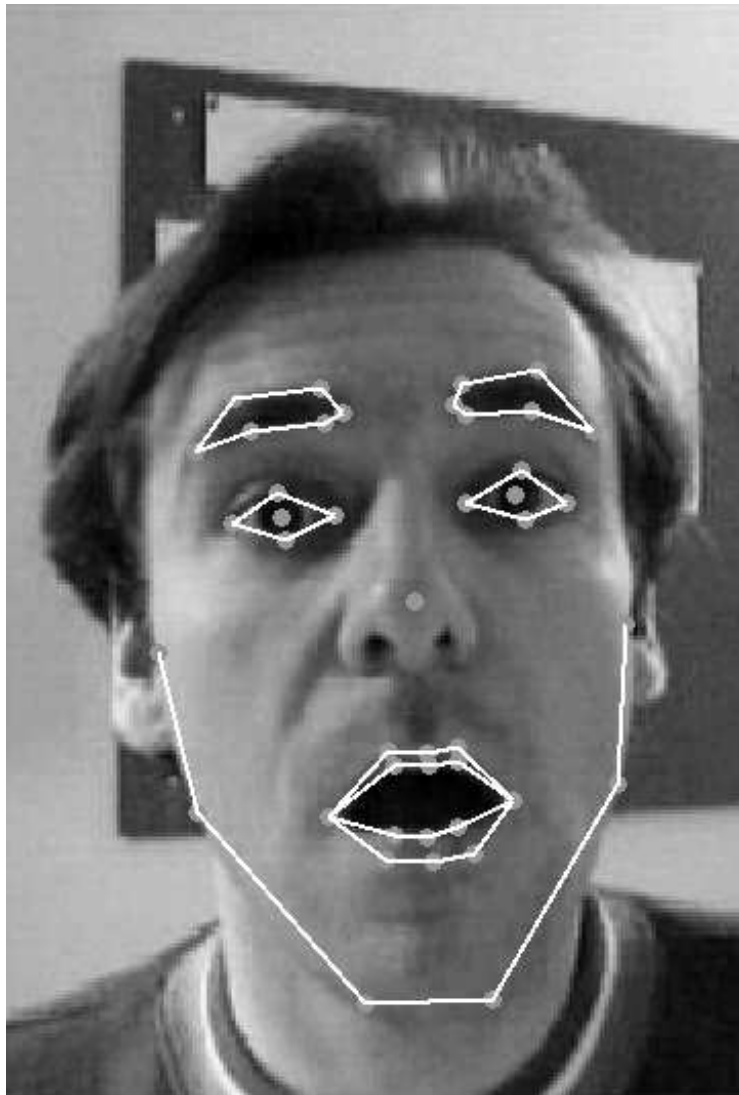


Figure 4.10: Hand labelling of a frame from video V2.



Figure 4.11: First mode of variation of the appearance model extracted from video V2.



Figure 4.12: Frames extracted from the video V2 after tracking.



Figure 4.13: Frames extracted from the video V3.

4.3.3 Video sequence V3: dialog

The last video sequence we used to assess our behaviour model is of a person in a dialogue. Since the person was speaking with someone else, the video includes talking, listening, laughing, and so on. The video covers a range of expressions. No scripting was used so that the conversation was spontaneous and natural.

This video was acquired with a JVC GR-DV2000 digital video camera. It contains 88044 frames. Figure 4.13 shows some frames extracted from that video sequence. The file `examples/V3/V3_orig.m1v` of the accompanying CD-ROM shows the full sequence.

We used 20000 frames from the video sequence V3 to train our behaviour model. Since the video was taken at 25 frames per second, those 20000 frames correspond to 13 minutes and 20 seconds. This length of training sequence is sufficient to train the behaviour model properly since it contains many repeated expressions. Most videos used by other works in this area are much shorter (a few seconds).

168 frames of video V3 have been marked up by hand using 96 points as depicted on figure 4.14. The face was successfully tracked through most of the video sequence. The active appearance model failed to model correctly the face on a small part of the video sequence. This is because the face was not entirely on the frame; the person on the video moved out of the visual field of the camera.

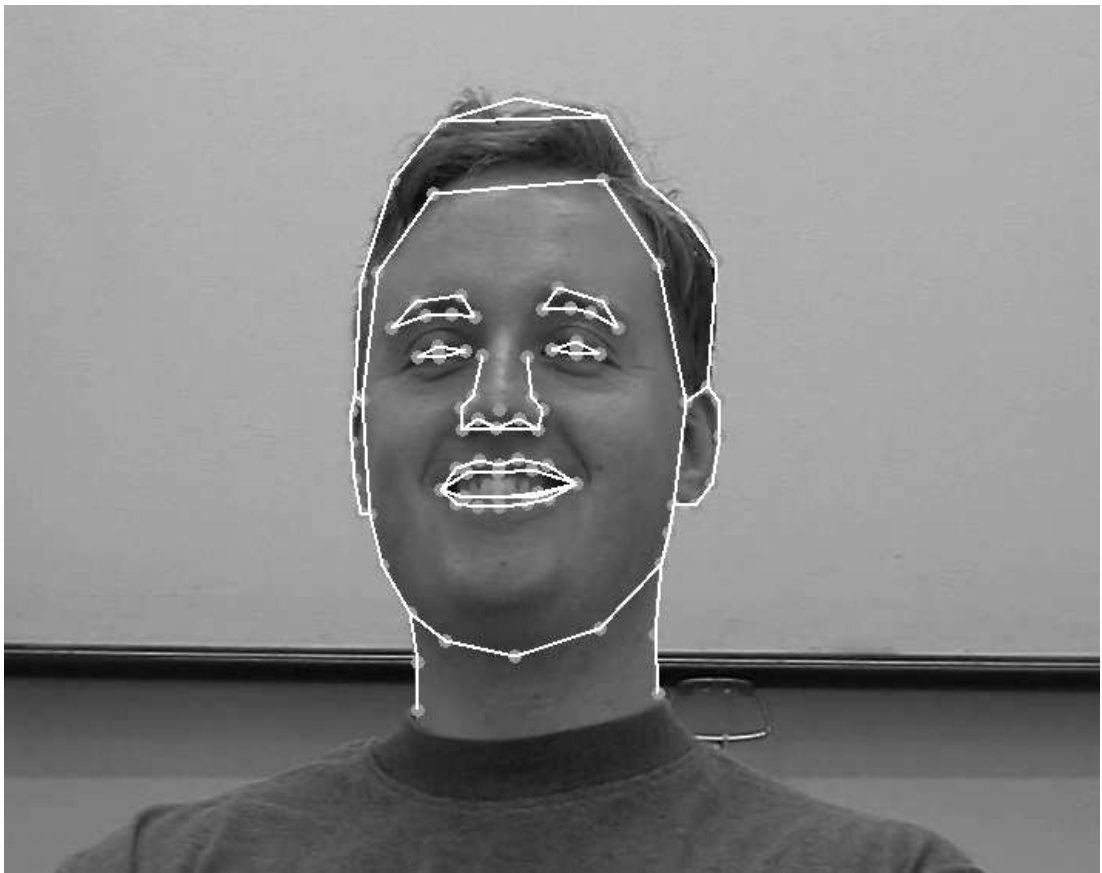


Figure 4.14: Hand labelling of a frame from video V3.



Figure 4.15: First mode of variation of the appearance model extracted from video V3.



Figure 4.16: Frames extracted from the video V3 after tracking.

The first mode of variation of the appearance model extracted from video V3 is shown on figure 4.15. The relative coordinates of those points as well as the pose, the scale and the position of the face have been reduced to 14 parameters for each frame.

A synthesised version of those parameters can be seen by watching the video file `examples/V3/V3_track.m1v` of the accompanying CD-ROM. Frames extracted from this file are shown on figure 4.16.

4.4 Conclusion

In this chapter, we have shown how we can use the active appearance model to track a face in a video sequence.

We have applied that tracking to 3 different video sequences. All the video sequences have been tracked entirely to generate sequences of appearance parameter vectors.

The following chapters will describe how we train models to generate trajectories through parameter space similar to these sequences. This allows us to synthesise convincing faces that mimic the behaviour of the face in the original video sequence.

Chapter 5

Finding and modelling pathlets

5.1 Introduction

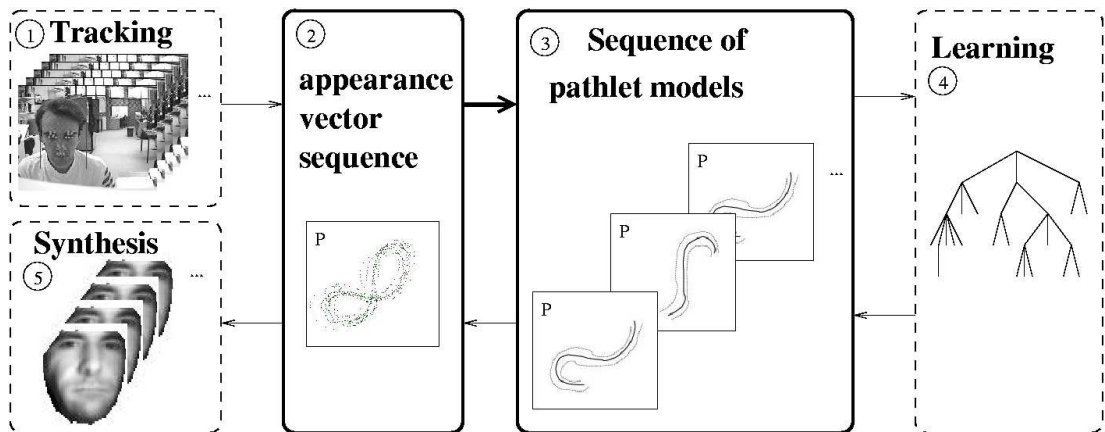


Figure 5.1: Overview of the components of the model. This chapter explains how the trajectory in the appearance parameter space P is transformed into a sequence of pathlet models. The trajectory is split into pathlets and the pathlets are grouped to form pathlet models.

This chapter describes how we break trajectories into sequences of pathlets, which are then grouped and used to train a set of statistical pathlet models. This is step 3 of our framework (see figure 5.1). The first section of this chapter describes how we extract pathlets from the full trajectory. In the second section, we describe how we model the groups of pathlets. In the next section, we present our algorithm for grouping a small set of extracted pathlets, as well as an alternative for a larger number of extracted pathlets. Finally we describe how to deduce the

sequence of pathlet models given the algorithms in the previous sections.

5.2 Extracting the pathlets

5.2.1 Finding nodes in the trajectory

We extract pathlets from the trajectory in the appearance parameter space by selecting points on that trajectory that split the trajectory into pathlets. We call those points “nodes”. Figure 5.2 illustrates this process. Two consecutive nodes are the beginning and the end of a pathlet.

Since each point on the trajectory represents a frame from the original video sequence, the nodes represent particular frames.

We have tried several algorithms to select those nodes, including:

- Selecting a node every N timeframes (algorithm A1).
- Selecting a node every N points or if the current point we are considering to be a node candidate is close to another node previously selected (algorithm A2).
- Selecting a node if the direction of the speed changes (algorithm A3).
- Selecting a node if it is close to a maximum number of other nodes (algorithm A4).

There are no good a priori reasons to choose one over the other.

Table 5.1 shows the nodes found on some trajectories. Its columns represent different trajectories. The first and second columns represent two hand drawn trajectories while the last column represents the trajectory given by the appearance parameters from video V1. The first row shows the points the trajectories are composed of. The remaining rows show the results of algorithms A1, A2, A3 and A4. For each algorithm and each trajectory, the trajectory points have been drawn in grey while the selected nodes have been highlighted with bigger black dots.

Algorithm A1 is the simplest approach: selecting pathlets of fixed length. For instance, [40] or [74] use fixed length segments of trajectory for their behaviour model. As you can see in Table 5.1, this strategy leads to unstructured pattern of nodes selected from the original trajectory. Although it might be valid for

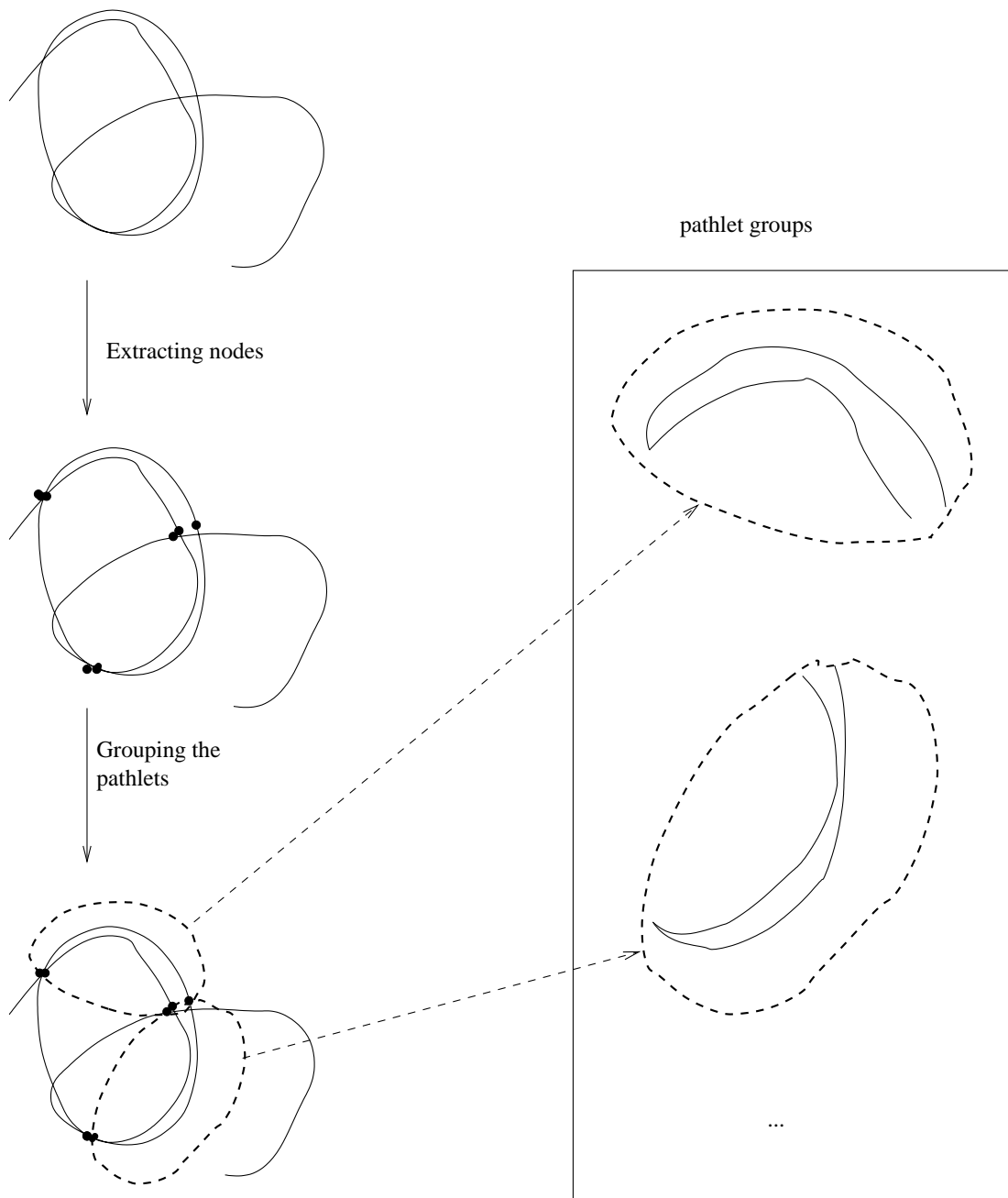


Figure 5.2: Overview of the pathlet groups extraction process. First nodes are extracted from the trajectory which split the trajectory into pathlets. Similar pathlets are then grouped together.

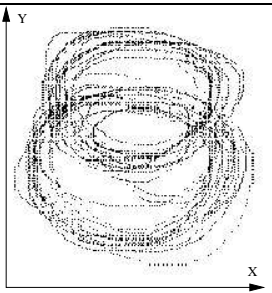
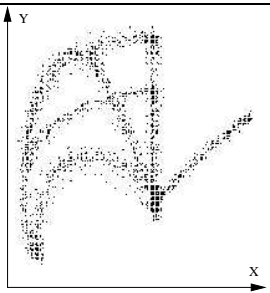
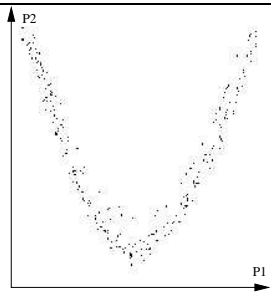
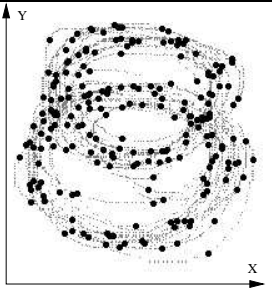
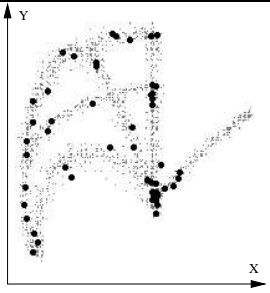
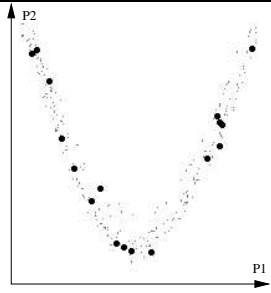
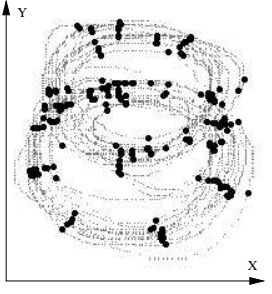
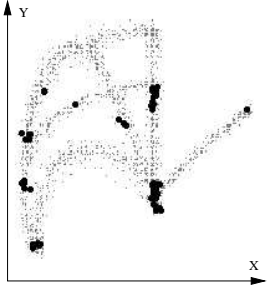
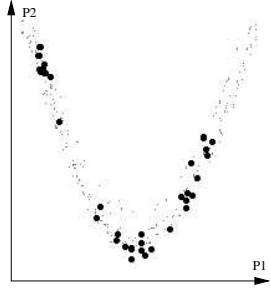
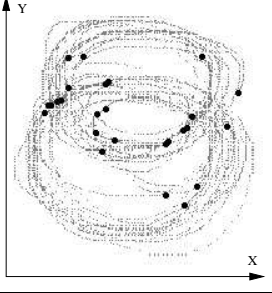
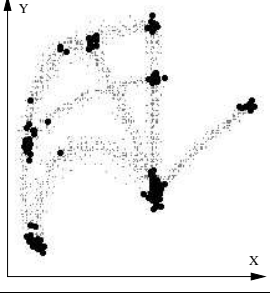
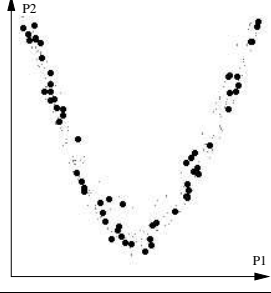
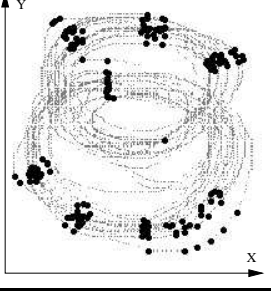
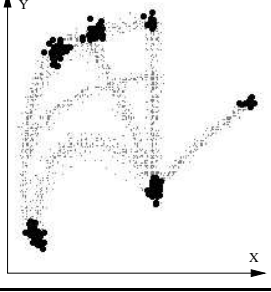
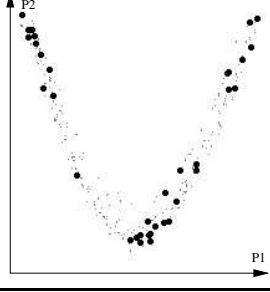
	Trajectory T1	Trajectory T2	Trajectory T3
Original trajectory			
Algorithm A1			
Algorithm A2			
Algorithm A3			
Algorithm A4			

Table 5.1: Comparison of the different algorithms of the node extraction process. Trajectory T2 has been drawn by hand so that it looks similar to the data used in [96] for direct visual comparison of the node selection results. Trajectory T1 is hand drawn while trajectory T3 is generated from the video sequence V1 (see section 4.3.1).

some behaviour models, it leads to sets of pathlets that are unlikely to be easily clustered.

Ideally we would like to split the trajectory into pathlets which can be effectively grouped and modelled. This suggests that the nodes (the ends of the pathlets) should form tight clusters where possible. Algorithm A1 does not achieve this.

A simple modification is to select every N^{th} point unless one of the next N points is close to an existing node, in which case it gets selected. Figure 5.4 illustrates this algorithm. In detail, algorithm A2 is given in figure 5.3.

```

Set  $i = 0$  (the first point).
Add point 0 to the node list.
Repeat:
  Consider points  $i + 1 \dots i + N$ .
  Find closest,  $j$ , to an existing node.
  Consider points  $j + 1 \dots i + N$ .
  Find the furthest,  $k$ , to existing nodes.
  If point  $j$  is within a threshold  $d$  of existing nodes and point  $k$  is not:
    Add point  $j$ .
    Set  $i = j$ .
  Otherwise:
    Add point  $i + N$  to the node list.
    Set  $i = i + N$ .
Until the end of the trajectory.

```

Figure 5.3: Algorithm A2

In Table 5.1 we can see that algorithm A2 gives a more structured segmentation than algorithm A1. Nodes are grouped together, but we can still see some remaining unstructured sets of nodes. Those unstructured sets of nodes often split the trajectory into small pathlets of one or two points long that are unlike the other pathlets. Such pathlets are outliers and are hard to group with others, since they are usually located further apart in the appearance parameter space.

In [95], Walter *et al.* propose a temporal segmentation of a gesture trajectory in two steps:

- detection of rest positions, where the velocity drops below a threshold
- detection of discontinuities in orientation to recover strokes

Following the same approach, algorithm A3 segments the trajectory by thresholding the scalar product of two consecutive unitary speed vectors. In order to

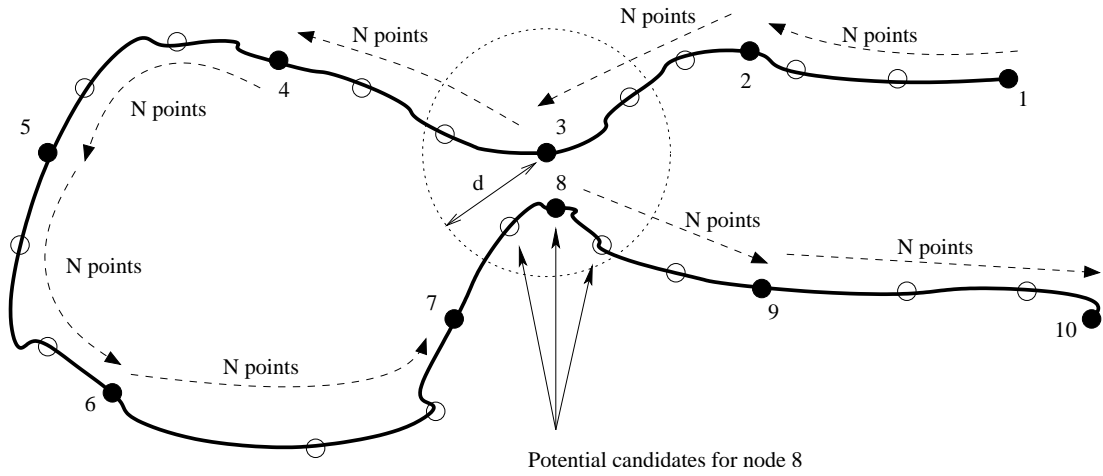


Figure 5.4: Selection of nodes using algorithm A2. The circles and black dots represent the points from the original trajectory. The nodes selected by the algorithm are represented by the black dots. The algorithm selects nodes every N points unless it passes close to a already selected node with a distance less than d . In that case, the next selected node is the point on the trajectory that is closest to the already selected node. This figure has been drawn with $N = 3$.

avoid selecting consecutive points as nodes, we only select the lowest scalar product for each set of consecutive points with low scalar products, in a similar way as for low distances in algorithm A2. The resulting selected nodes on trajectories T1, T2 and T3 are shown in Table 5.1. Trajectory T2 is a hand draw reproduction of the data shown in [95], representing hand positions during gestures. Algorithm A3 performs well on this trajectory but is sometimes disturbed by the noise in the data. Algorithm A3 does not perform well with the other two trajectories, due to noise, the small sampling rate of trajectory T3 and the behaviour of trajectory T1. Since no sudden change of velocity appears in trajectory T1, only changes of directions due to noise are selected.

The attempts at clustering the trajectory highlight the following:

- the nodes should be chosen so that they are close to other chosen nodes to ease the grouping of pathlets.
- the smaller the number of clusters of nodes, the smaller the number of pathlet groups, and the easier it will be to learn their temporal relationship.

Those remarks lead us to consider algorithm A4. We select points that are close to other trajectory points in the appearance parameter space. In order to

find those points, algorithm A4 is based on the mean shift algorithm.

5.2.2 The mean shift algorithm

The mean shift algorithm is a nonparametric estimator of density gradient. It has led to many applications such as tracking [18, 56], filtering and image segmentation [19] or recognition of freehand sketches [100].

The goal of the algorithm is to find a local maximum of density amongst a set of n points $\{\mathbf{x}_i\}_{i=1\dots n}$ in a d -dimensional Euclidian space. In our case, the space is the appearance parameter space P .

The discrete density of the set of points $\{\mathbf{x}_i\}_{i=1\dots n}$ is approximated by a continuous density. A kernel $K(\mathbf{x})$ is placed at each point \mathbf{x} to get a multivariate kernel density estimate for the whole set of points.

Given a window radius h , the density computed at a point \mathbf{x} is given by:

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (5.1)$$

In practice, we are using the Epanechnikov kernel given by the formula:

$$K(\mathbf{x}) = \begin{cases} \frac{1}{2}v_d^{-1}(d+2)(1 - \mathbf{x}^T\mathbf{x}) & \text{if } \mathbf{x}^T\mathbf{x} < 1 \\ 0 & \text{otherwise} \end{cases} \quad (5.2)$$

where v_d is the volume of the unit d -dimensional sphere [90].

Since K is differentiable, we can compute the gradient of \hat{f} :

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{nh^d} \sum_{i=1}^n \nabla K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \quad (5.3)$$

For the Epanechnikov kernel it becomes:

$$\nabla \hat{f}(\mathbf{x}) = \frac{1}{nh^d} \frac{d+2}{v_d h^2} \sum_{\mathbf{x}_i \in S_h(\mathbf{x})} (\mathbf{x}_i - \mathbf{x}) \quad (5.4)$$

where $S_h(\mathbf{x})$ is a hypersphere of radius h centred on \mathbf{x} .

We can see, from equation 5.4 that an extremum of density is reached at the point \mathbf{y} given by:

$$\mathbf{y} = \frac{1}{n_{\mathbf{x}}} \sum_{\mathbf{x}_i \in S_h(\mathbf{x})} \mathbf{x}_i \quad (5.5)$$

where $n_{\mathbf{x}}$ is the number of points present in the hypersphere $S_h(\mathbf{x})$.

The vector $M_h(\mathbf{x}) = \mathbf{y} - \mathbf{x}$ is called the mean-shift vector. If we translate \mathbf{x} by $M_h(\mathbf{x})$, we reach an extremum in the kernel density function \hat{f} .

The idea of the mean shift algorithm is to iteratively find an estimate of the local extremum of density within the current window and then translate the centre of the window to the found extremum.

Comaniciu and Meer have proved the convergence of this algorithm as well as the monotonic increase in the density estimate through the iterations given some hypotheses that are verified in the case of an Epanechnikov kernel [20].

The mean shift algorithm involves iterating the following steps:

- computation of the mean shift vector $M_h(\mathbf{x})$
- translation of the window $S_h(\mathbf{x})$ by $M_h(\mathbf{x})$

until convergence.

Figure 5.5 summarises the algorithm.

Set \mathbf{y}_0 to the start position.
 Iterate:

$$\mathbf{y}_{k+1} = \frac{1}{n_{\mathbf{y}_k}} \sum_{\mathbf{x}_i \in S_h(\mathbf{y}_k)} \mathbf{x}_i$$
 Until the hyperspheres $S_h(\mathbf{y}_k)$ and $S_h(\mathbf{y}_{k+1})$ contain the same points.
 Adapted from Comaniciu and Meer [20]

Figure 5.5: The mean shift algorithm.

In the mean shift algorithm, the radius h of the hypersphere acts as a smoothing parameter. It represents the scale at which we consider the extrema to be local. A small value will end up in selecting only the start position as being a maxima of density in its close neighbourhood. A large value will skip some local maxima of small areas to focus on local maxima of larger areas. Since different data have different scales, h has to be adapted manually for each dataset.

We can use the mean shift algorithm to find local maxima of density in the set of points defined by the trajectory. We initialise the mean shift algorithm at each point in turn, the result being the closest local maxima of density. This provides us with a set of local maxima in the trajectory points distribution. In practice, we are only using every q points as it does not significantly affect the result to much while increasing the speed of the search by a factor of q .

We compute the set of nodes by adding all nodes that are close to the local maxima. We use the same procedure as algorithm A2 to find the closest node with a distance less than d to a local maxima: we select the node amongst the consecutive nodes that lies in the hypersphere of radius d and centred at the closest local maxima.

The parameter d has to be adapted manually for each dataset. A too small value of d results in too few points being selected in the neighbourhood of a maxima of density. Often, only one point is selected in that case, as shown by figure 5.6. If the value is too large, we take the risk of selecting too few good nodes by choosing each nodes between too many candidates at a time. This is shown on figure 5.7.

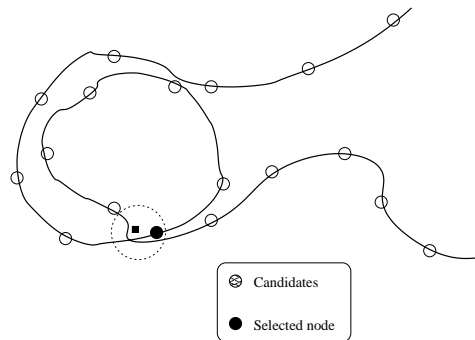


Figure 5.6: Small distance for the selection of neighbours. The maxima found by the mean shift algorithm is located by a black square. Only one point can be selected as a node due to the small size of the hypersphere.

5.2.3 The pathlets

Now that we have selected nodes on the original trajectory, we can find the associated pathlets. Each pathlet is the sequence of points between two consecutive nodes on the trajectory. Figure 5.8 illustrates how to find the pathlets.

The next step in the framework is to cluster the pathlets into meaningful groups which can be modelled. The next section describes how we model the pathlet groups.

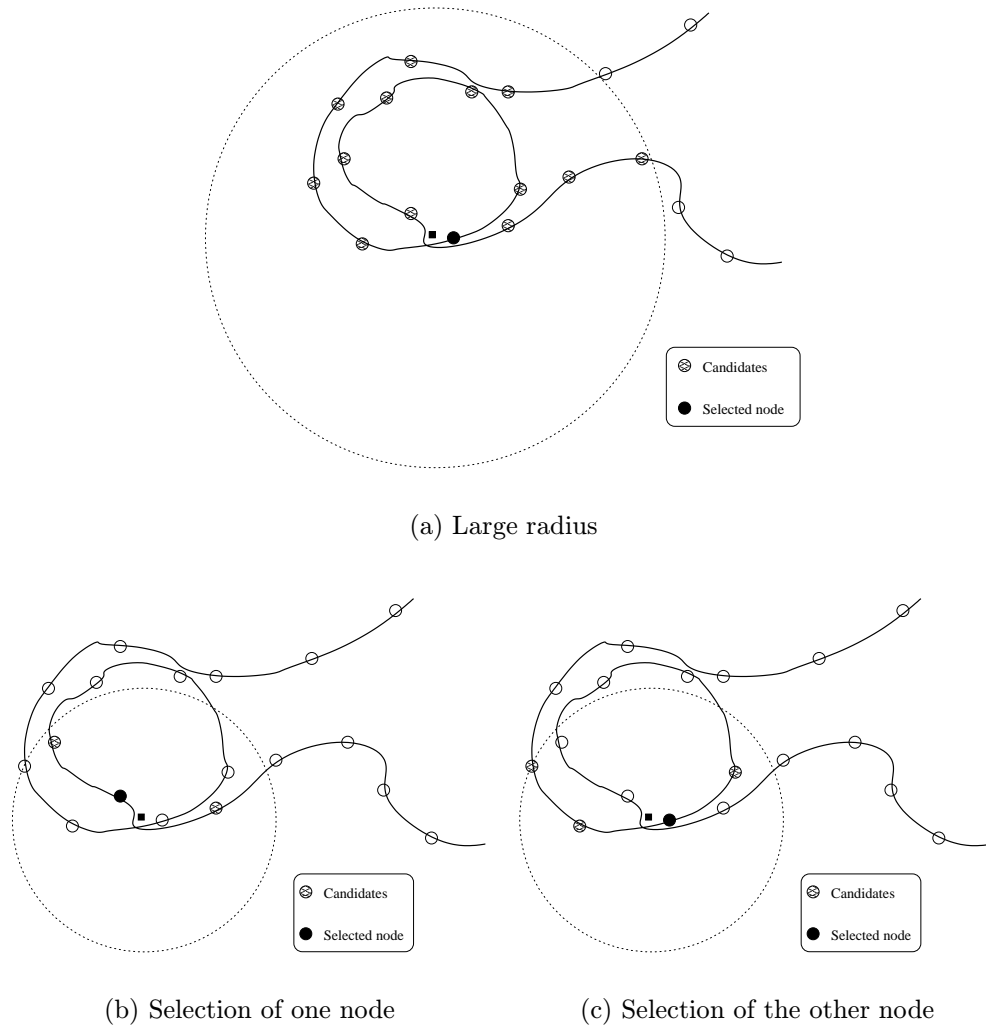


Figure 5.7: Large distance for the selection of neighbours. The maxima found by the mean shift algorithm is located by a black square. Only one point can be selected as a node on figure 5.7(a) due to the large size of the hypersphere covering a large part of the trajectory. A smaller distance d for the selection would have led to two nodes selected as shown on figures 5.7(b) and 5.7(c), since we are entering and leaving the hypersphere twice while following the trajectory.

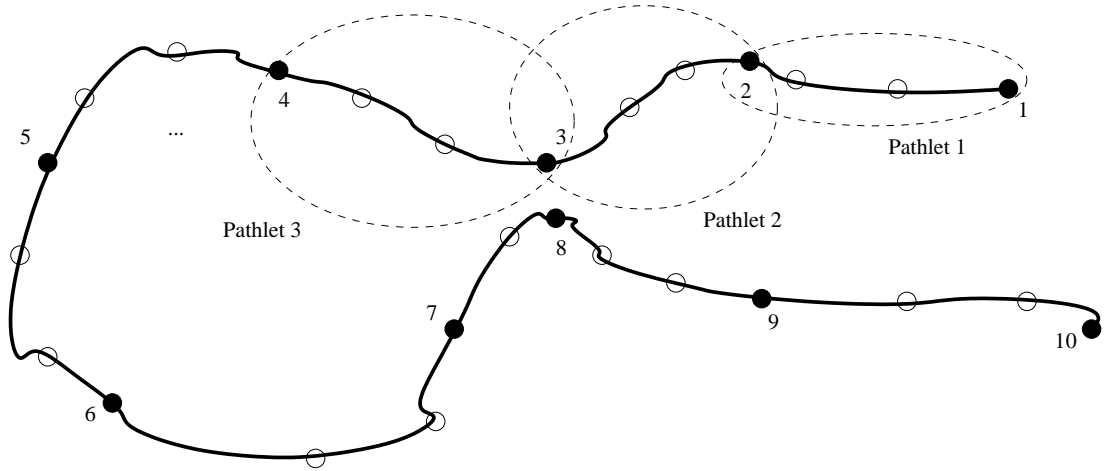


Figure 5.8: Selection of the pathlets given the nodes. The points of the original trajectory are represented by circles and the nodes are represented by filled circles. The first pathlet is the sequence of points from the original trajectory from node 1 to node 2, the second is the sequence from node 2 to node 3, and so on.

5.3 The pathlet model

5.3.1 The spatial model

We model each pathlet within a group by using a linear statistical model, assuming a Gaussian distribution. Each pathlet is described by a vector, which is a simple concatenation of the pathlet points. A pathlet group is a set of vectors, on which we can apply a principal component analysis. Each pathlet \mathbf{s} is approximated by:

$$\mathbf{s} = \bar{\mathbf{s}} + \mathbf{Q}_s \mathbf{b}_s \quad (5.6)$$

where $\bar{\mathbf{s}}$ is a vector representing the mean pathlet of the group, \mathbf{Q}_s is a matrix computed by the principal component analysis describing how the data varies, and \mathbf{b}_s is the vector of parameters for that particular pathlet. The probability density function of the set of parameters \mathbf{b}_s is modelled by a Gaussian, by computing the mean and variance of \mathbf{b}_s for all pathlet \mathbf{s} in the group. Work by Makris and Ellis [64] has shown that for pedestrian trajectories, the distribution of paths around a mean path is approximatively Gaussian.

In order to be able to perform the principal component analysis on a group of pathlets, it is required that all the pathlets are encoded with the same number of

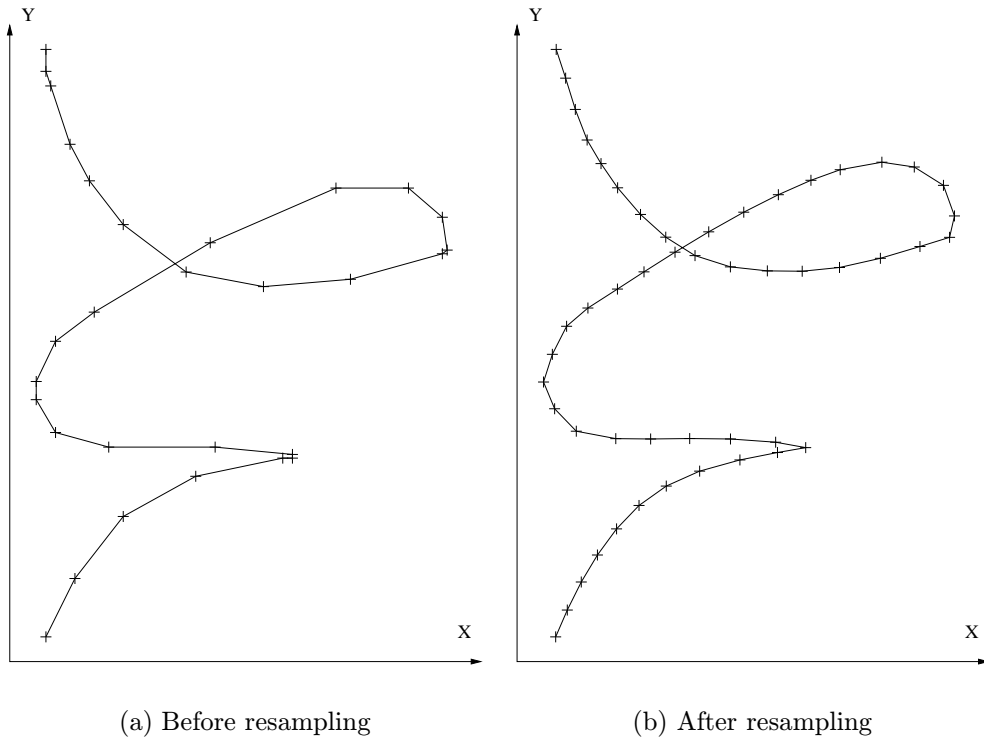


Figure 5.9: Resampling using cubic splines. The trajectory on figure 5.9(b) represents the trajectory on figure 5.9(a) resampled to 50 points using cubic splines. The data on figure 5.9(a) were created by manual drawing.

points. Therefore, we interpolate all the pathlets by cubic splines and homogeneously resample them to a given number of points [65]. Resampling to 50 points has been found to give good results in our experiments. Figure 5.9 shows the result of resampling a hand drawn trajectory using cubic splines.

Unfortunately the resampling process discards timing information. In the original pathlets, we know that the time spent between two successive points is constant¹. In the resampled pathlets, we do not know the time between two successive points. However, we need this information if we want to synthesise new pathlet from the pathlet model. The next section describes how we can extend the current model with timing information.

¹The time spent between two successive points is $\frac{1}{25}$ second if the original training video sequence is taken at 25 frames per second.

5.3.2 A spatiotemporal model

In order to be able to keep track of the timing information, we need to extract the times for each point in a resampled pathlet. The pathlets are resampled so that the length of the resampled pathlet is constant between successive points. This process does not take the original timing into account. As we do not know the exact time for each point of the resampled pathlet, we linearly interpolate the time with the length of the curve for each segment of $1/25$ seconds (the time between points on the original pathlet).

Let δ be the vector containing the time to reach each point of a resampled pathlet s . The new model of pathlet is the concatenation of the points of the resampled pathlet as well as the times to reach those points from the beginning of the pathlet. A pathlet is now represented by the vector $(\mathbf{s}^T, \delta^T)^T$.

Thus, the spatiotemporal model of a pathlet group becomes a linear combination of pathlets extended with timings. Each pathlet from a group is modelled by the equation:

$$\begin{pmatrix} \mathbf{s} \\ \delta \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{s}} \\ \bar{\delta} \end{pmatrix} + \mathbf{Q}_{\mathbf{s},\delta} \mathbf{b}_{\mathbf{s},\delta} \quad (5.7)$$

where $(\bar{\mathbf{s}}^T, \bar{\delta}^T)^T$ is a vector representing the mean pathlet of the group along with these timings, $\mathbf{Q}_{\mathbf{s},\delta}$ is a matrix computed by the principal component analysis and describing how the data varies and $\mathbf{b}_{\mathbf{s},\delta}$ is the vector of parameters for that particular pathlet. The probability density function of the set of parameters $\mathbf{b}_{\mathbf{s},\delta}$ is modelled by a Gaussian, by computing the mean and variance of $\mathbf{b}_{\mathbf{s},\delta}$ for all pathlets $(\mathbf{s}^T, \delta^T)^T$ in the group.

Generation of pathlets is therefore a simple sampling from the distribution of $\mathbf{b}_{\mathbf{s},\delta}$. Applying equation 5.6 gives a pathlet \mathbf{s} which is then resampled using cubic splines, and timing information δ . The points are resampled to correspond to times multiples of $\frac{1}{25}$ second. We linearly interpolate the time between two generated points and the length of the cubic spline curve between the points.

In some rare occasions, two consecutive generated points might have timings that are going backwards in time. This can occur because we use a (symmetric) Gaussian to represent strictly one sided (i.e. $\delta \geq 0$) data. Sampling from such a distribution can occasionally generate negative values. We could of course have used a one sided probability distribution function such as the gamma distribution, but we could not easily account correlations with the spatial variables. Instead

when sampling, we simply discard the generated pathlet and sample a new value for $\mathbf{b}_{\mathbf{s},\delta}$ from its probability distribution, to generate a new pathlet. This has also the advantage of eliminating some outlier pathlets that might have been generated without the timing model.

5.3.3 The spatiotemporal model with linear residuals

When generating several pathlets in sequence, one needs the output to be smooth. However, by concatenating generated pathlets, the ends of the successive pathlets might not match. In order to avoid this problem, we extend the spatiotemporal to:

$$\begin{pmatrix} \mathbf{s} \\ \delta \end{pmatrix} = \begin{pmatrix} \bar{\mathbf{s}} \\ \bar{\delta} \end{pmatrix} + \mathbf{Q}_{\mathbf{s},\delta} \mathbf{b}_{\mathbf{s},\delta} + \mathbf{R}_{\mathbf{s},\delta} \quad (5.8)$$

where $\mathbf{R}_{\mathbf{s},\delta}$ represents the residuals².

In this work, we assume a linear spatial model for the residuals. It can be computed given the last point of the previous pathlet.

We first generate a pathlet $(\mathbf{s}'^T, \delta'^T)^T$ in the same way we do using the spatiotemporal model. We want the first point of this pathlet to match the previous pathlet. Let's call \mathbf{e} the vector that translates the first point of $(\mathbf{s}'^T, \delta'^T)^T$ into the last point of the previous pathlet. In order to obtain $(\mathbf{s}^T, \delta^T)^T$, we keep the timings³ and translate the points of \mathbf{s}' so that:

$$s_{ik} = s'_{ik} + \frac{\sum_{j=i}^{N-1} \delta_j}{\sum_{j=1}^{N-1} \delta_j} e_k \quad (5.9)$$

where e_k is the k^{th} coordinate of \mathbf{e} , s_{ik} is the k^{th} coordinate of the i^{th} point in the pathlet \mathbf{s} , s'_{ik} is the k^{th} coordinate of the i^{th} point in the pathlet \mathbf{s}' and δ_j is the j^{th} coordinate of δ . Figure 5.10 is an illustration of the application of equation 5.9.

This generating process gives a new pathlet that begins at the end of a previously generated pathlet, thus generating a more continuous overall trajectory. However, this method is only valid if \mathbf{e} is small. A large translation of points can generate an unrealistic pathlet. In practise, this problem of generation happens if an unlikely sequence of pathlets is generated by the model of temporal

²The residuals represent the difference between a pathlet and its approximation by the linear model.

³ $\delta' = \delta$

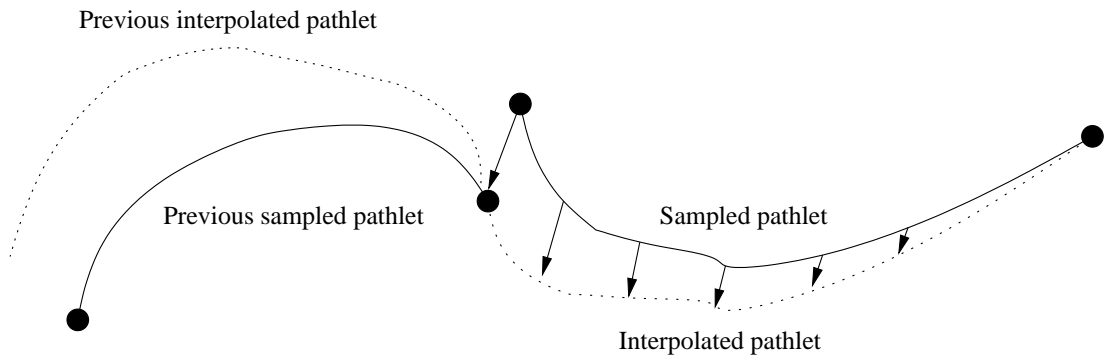


Figure 5.10: Linear model of residuals. The first point of the current pathlet is moved to match the last point of the previous pathlet.

relationship between the pathlet models, described in chapter 6.

5.4 Grouping the pathlets

The previous section describes a method of modelling a group of pathlets. This section describes how to cluster pathlets into groups of similar pathlets.

5.4.1 The grouping criterion

To group similar pathlets, one must define what “similar” means. A common approach for clustering data is to derive a measure of similarity or a distance between a pair of elements.

In a first attempt to cluster the pathlets into groups, we used this approach. We built a similarity measure based on the dynamic time warping algorithm commonly used in speech recognition [76]. A clustering algorithm was then used to find the different groups. The algorithm we used for this approach is described in section 5.4.2.

The approach leads to the potential problem of outliers. Grouping the pathlets with our spatiotemporal model of pathlets requires that no outlier is used during the model building. If an outlier is used, then the mean pathlet is shifted and the distribution around the mean pathlet cannot be properly modelled by a Gaussian. This effect leads to an improper pathlet model for the group containing the outlier.

Furthermore, we need pathlet models that are compact; groups with a wide range of different pathlets are unlikely to be modelled properly. The way pathlets

are grouped affects the quality of the resulting pathlet models. One has to group the pathlets so that the models of the resulting groups are “good”. Since we want compact models of groups that do not contain outliers, we define a “good” pathlet model as being a model where the variance of the underlying Gaussian is low. We have derived a new grouping algorithm that takes this specific criterion into account. It is presented in section 5.4.3.

5.4.2 Grouping algorithm based on dynamic time warping and normalised cuts

We derive a similarity measure based on the dynamic time warping algorithm, then cluster the pathlets using the normalised cut algorithm.

5.4.2.1 The pathlet similarity measure

We wish to compare the pathlets in space only. The difference of timings are modelled by the spatiotemporal model, so the similarity measure needs only to assess the difference of shape between two pathlets.

Two points from different pathlets might have different speeds. The points on two similar pathlets might not correspond to each other directly while still describing the same curve in space. This problem of matching pathlets is similar to the problem of matching phonemes in speech recognition. The phonemes can be pronounced with different speed so the same point in time will not correspond to the same part of the phoneme.

Dynamic time warping is commonly used to match phonemes [50]. It is a sequence alignment algorithm which non-linearly warps the timing of two sequences to find the optimal match between the two sequences (see appendix A).

5.4.2.2 The normalised cut algorithm

We now have a similarity measure to compare two pathlets. We wish to cluster the pathlets using this similarity measure. Clustering algorithms have been used in a wide range of applications. However, our similarity measure does not verify the triangular inequality. So clustering algorithms like the k-means algorithm [70] cannot be used.

We require clustering algorithm based on a similarity measure and not a distance measure. We also need the algorithm to be efficient as the number of

pathlets might be large for long training video sequences. We use the normalised cut algorithm from Shi and Malik [88], described in appendix B.

The normalised cut algorithm returns a set of groups of pathlets. Each resulting group of pathlets is modelled by a pathlet model.

5.4.3 The greedy algorithm

As we show later in this chapter, the algorithm based on the dynamic time warping and the normalised cut sometimes groups pathlets that look different. This results in the creation of a poor pathlet models. This problem is due to the fact that we are comparing pathlets in pairs using a similarity measure that is not necessarily compatible with our grouping model. A global similarity measure for several pathlets at a time would be much more appropriate since it can ensure that a group is coherent, a property that is not guaranteed with a similarity measure for only two pathlets. A global similarity measure leads to a groupwise clustering rather than the pairwise clustering of the dynamic time warping and normalised cut algorithm.

Furthermore, we want the similarity measure to be compatible with our grouping model. One logical choice for the similarity measure is the variance of the group of pathlets. Indeed several pathlets are properly grouped if the variance of the group is small. A large variance denotes a group containing pathlets that look different to each other.

Unfortunately, we cannot use the same clustering algorithm. The normalised cut algorithm is based on the use of a similarity matrix, which cannot easily be created for a groupwise similarity measure. Instead, we use a greedy algorithm to group the pathlets. At each step, the algorithm makes the best decision for the grouping.

The algorithm first assumes that each pathlet given by the trajectory segmentation initially forms a group by itself. Let \mathcal{S} be this initial set of groups.

For every pair (g_i, g_j) of elements of \mathcal{S} , we compute the variance of the group $g_{i \cup j}$ that is built using the pathlets from both g_i and g_j . We select the pair of groups (g_a, g_b) that gives the lowest variance and merge those two groups to form only one group. We delete g_a and g_b from \mathcal{S} and insert $g_{a \cup b}$ instead.

We iterate the process until we reach a given number of clusters n_c .

We can also stop the algorithm if we reach a given variance so that we can control the quality of the model. However, if we set this variance too low, we

stop the algorithm too quickly and most of the resulting groups will only contain one pathlet, making the use of pathlet models superfluous in our framework. We prefer to control the number of pathlet models so that we can control the computational complexity of the remaining steps in the framework.

Figure 5.11 summarises the greedy algorithm.

Create a pathlet group set $\mathcal{S} = \{g_i\}_{1 \leq i \leq N}$
 containing empty pathlet groups.
 For each i , add the i^{th} pathlet to the group g_i .
 While $|\mathcal{S}| > n_c$ do:
 For all $(g_i, g_j) \in \mathcal{S}^2$ compute the variance of $g_{i \cup j}$
 Select the lowest variance $g_{a \cup b}$
 Add $g_{a \cup b}$ in \mathcal{S}
 Remove g_a and g_b from \mathcal{S}

Figure 5.11: The greedy algorithm.

5.5 The results

5.5.1 Results of clustering based on dynamic time warping and normalised cut

This section presents the pathlet groups extracted from original trajectories. The normalised cut algorithm has been used for the grouping while the dynamic time warping algorithm has been used for the similarity measure.

Tables 5.2 and 5.3 show pathlet groups extracted from the trajectory T1.

For each pathlet group, the nodes extracted from the original trajectory are shown with black dots; the pathlets that form the group are drawn between the nodes. The nodes have been generated using algorithm A4, as previously illustrated in table 5.1 page 73.

Below those pictures, we represent the pathlet model by the generation of one hundred random pathlets. This is done by sampling pathlets from the group, with a Gaussian distribution around the mean pathlet, in order to represent the variance of that group. Only dots representing the positions of the generated points are shown for the reader to appreciate the modelling of timings.

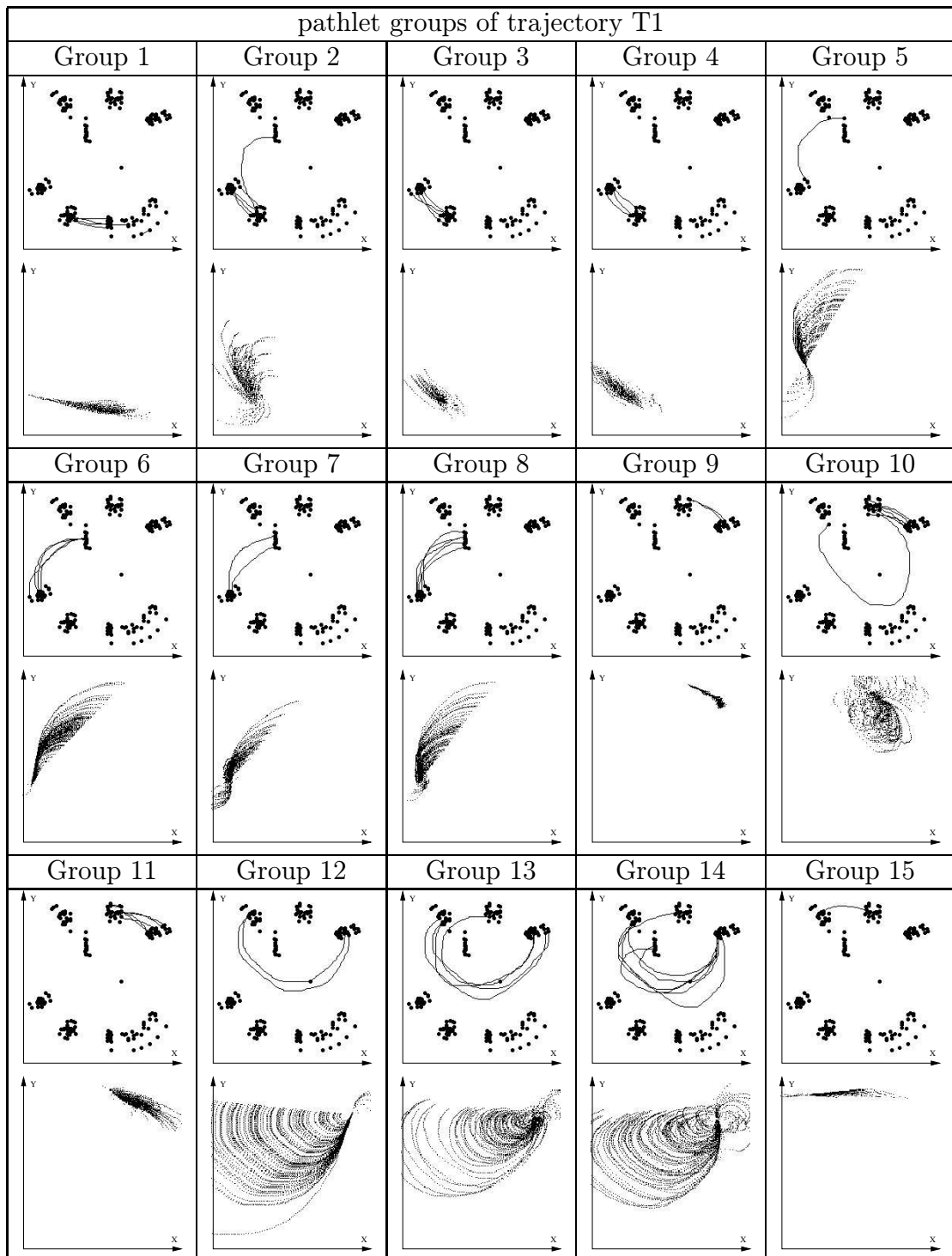


Table 5.2: Pathlet groups for trajectory T1 (groups 1 to 15) based on dynamic time warping and normalised cut clustering. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model, one hundred generated pathlets are drawn in the lower figure.

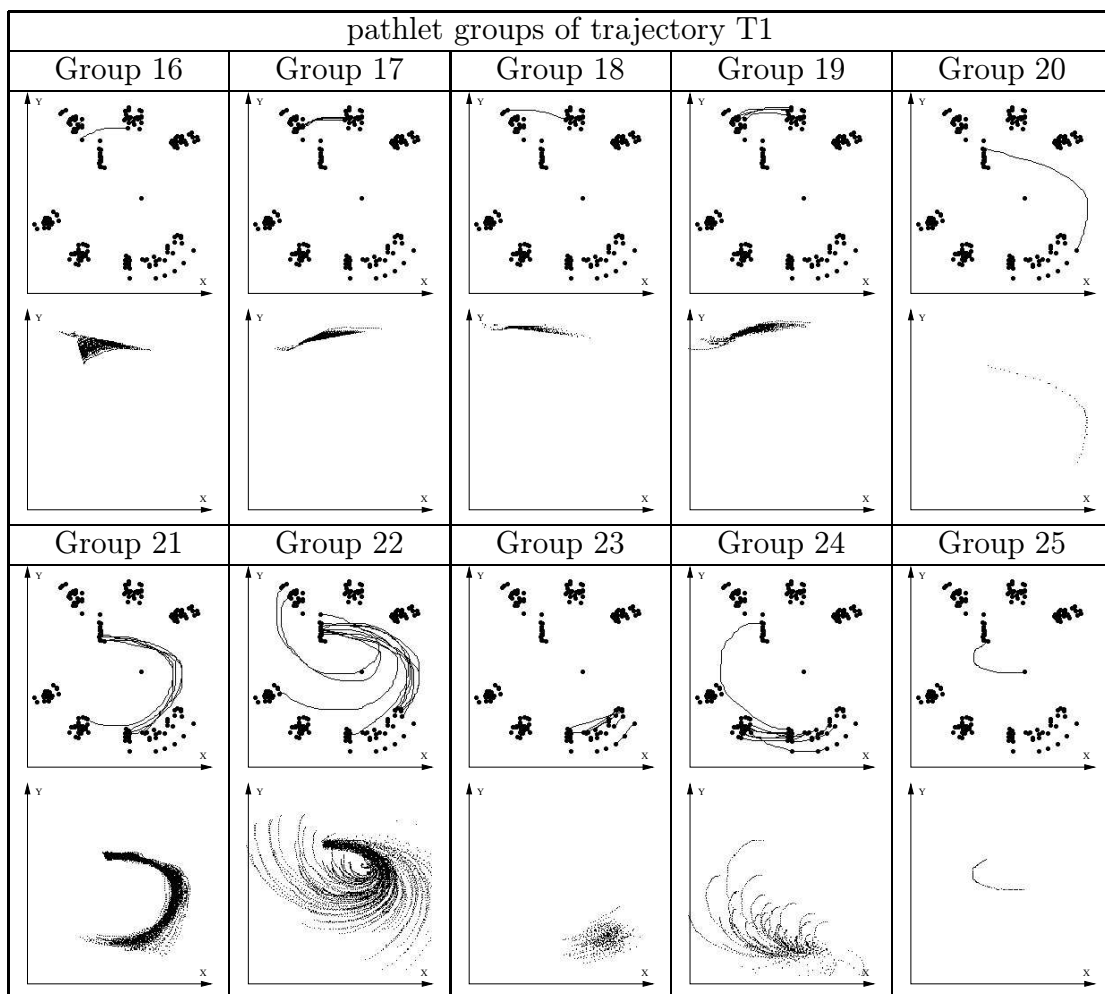


Table 5.3: Pathlet groups for trajectory T1 (groups 16 to 25) based on dynamic time warping and normalised cut clustering. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model, one hundred generated pathlets are drawn in the lower figure.

Note that one can find four different types of pathlet models:

- pathlet models that model properly the variations observed with several similar pathlets (models learnt from groups 3, 9, 11 or 21 in tables 5.2 and 5.3, for instance).
- pathlet models that have been build with only one pathlet. Those models can only generate one pathlet since the variance in its training set of pathlets is null; $\mathbf{Q}_{s,\delta}$ from equation 5.6 is therefore the null matrix (models learnt from groups 20 and 25 in table 5.3, for instance).
- pathlet models that have been built with an outlier in their training set of pathlets (models learnt from groups 2, 5, 6, 10, 12 or 24 in tables 5.2 and 5.3, for instance).
- pathlet models which have been trained on pathlets that do not look totally similar (models learnt from groups 14 and 22 in table 5.2 and 5.3, for instance).

Similar type of pathlet models can be observed when applying the algorithms to trajectory T2 (tables 5.4 and 5.5) and trajectory T3 (table 5.6).

For instance for trajectory T2, the pathlet models corresponding to groups 3, 14 or 21 have been learnt on a set of pathlets containing outliers. The groups 1, 8, 23, or 25 are correctly modelled, while the pathlet models corresponding to groups 12 or 13 have been built using only one pathlet. Finally pathlet models corresponding to the groups 6, 19 or 26 have been learnt on a set of pathlets that do not look similar to each other.

Finally, the same types of pathlet models can be observed when extracting pathlet groups from the trajectory T3 (see table 5.6).

Note that tables 5.2, 5.3, 5.4, 5.5 and 5.6 shows all the possible generated pathlets. They include the pathlets that have been generated with negative timings. This often occurs where the pathlet groups contain outliers. For instance the pathlet model corresponding to group 5 extracted from trajectory T1 can generate such impossible pathlets. Figure 5.12 shows pathlets generated from the pathlet model of group 5 extracted from the trajectory T1. Pathlets generated with negative timings are circled with a dotted circle. As mentioned in section 5.3.2, pathlets that have timings going backward in time are discarded and new

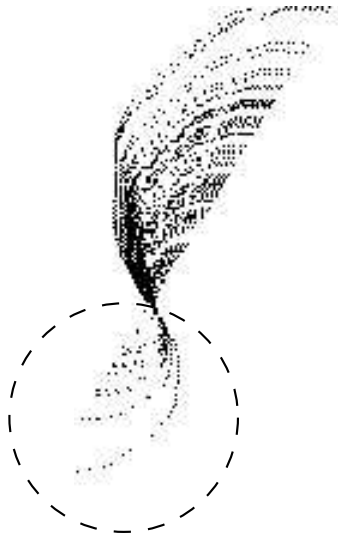


Figure 5.12: Generation of pathlets with negative timings. A hundred pathlets have been generated from the pathlet model learnt from the pathlet group 5 extracted from trajectory T1. The pathlets with negative timings have been circled with a dotted circle. In practice we ignore such pathlets and sample again.

pathlets are generated instead. We can see that this rejection of impossible pathlets improves the output of the pathlet model. However the model still remains weak since it can generate other unlikely pathlets.

Ideally, we prefer to have pathlet models that model the original data properly. We want to avoid pathlet models modelling outliers as well as those modelling pathlets that do not look similar to each other. The two other types of pathlet models are able to model properly the original data.

Pathlet models built from pathlet groups that only contain one pathlet reproduce exactly the pathlet they have learnt. Since we want to avoid using original frames to create the output video of our system, we would like to avoid such models. However, we cannot discard those groups. Since we cannot generalise from a group of pathlets with only one example, we have to copy them to be faithful to the original data.

5.5.2 Results of clustering based on the greedy algorithm

This section presents the pathlet groups extracted from the original trajectory using the greedy algorithm. Nodes are extracted from the trajectory using the

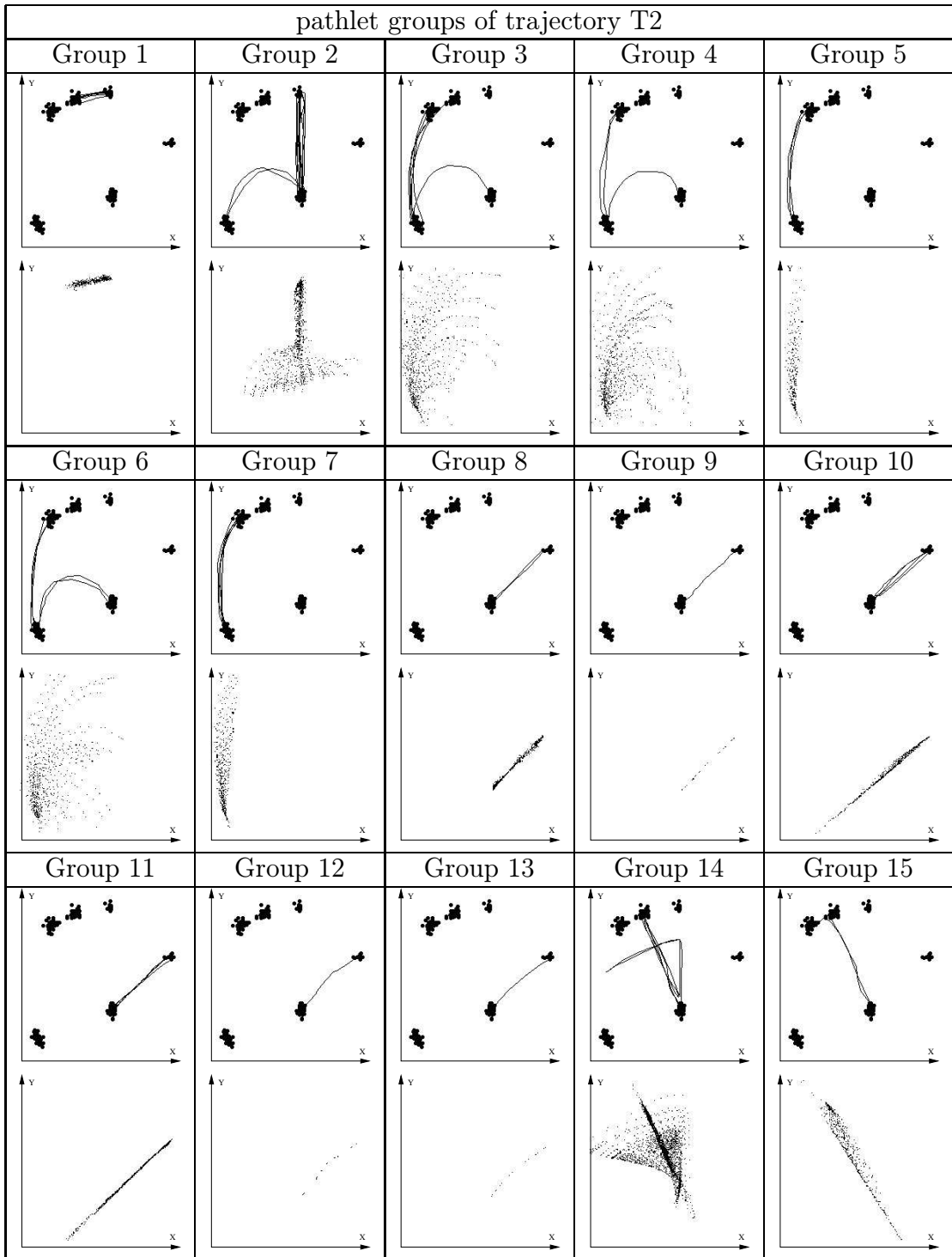


Table 5.4: Pathlet groups for trajectory T2 (groups 1 to 15) based on dynamic time warping and normalised cut clustering. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model learnt from that group, one hundred generated pathlets are drawn in the lower figure.

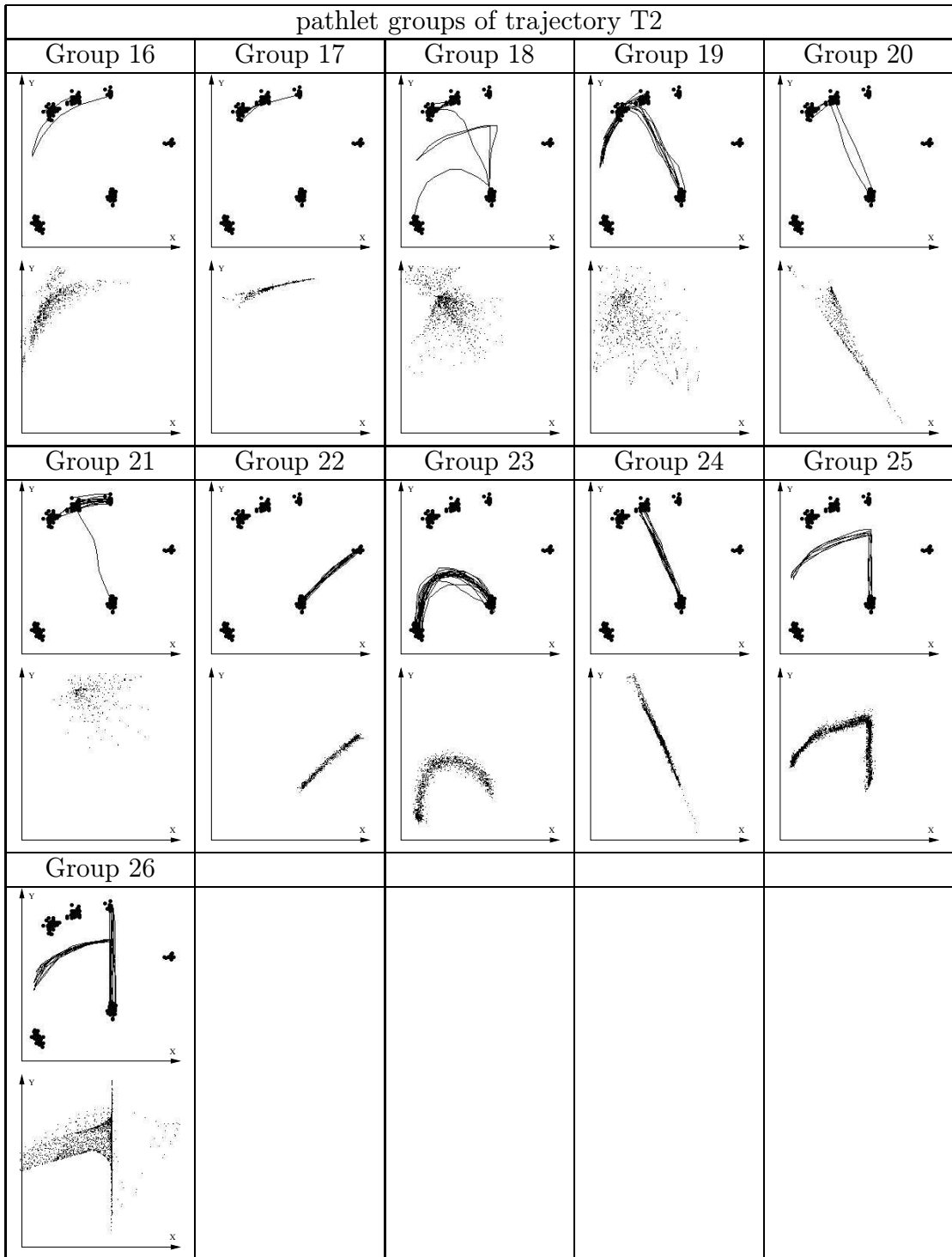


Table 5.5: Pathlet groups for trajectory T2 (groups 16 to 26) based on dynamic time warping and normalised cut clustering. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model learnt from that group, one hundred generated pathlets are drawn in the lower figure.

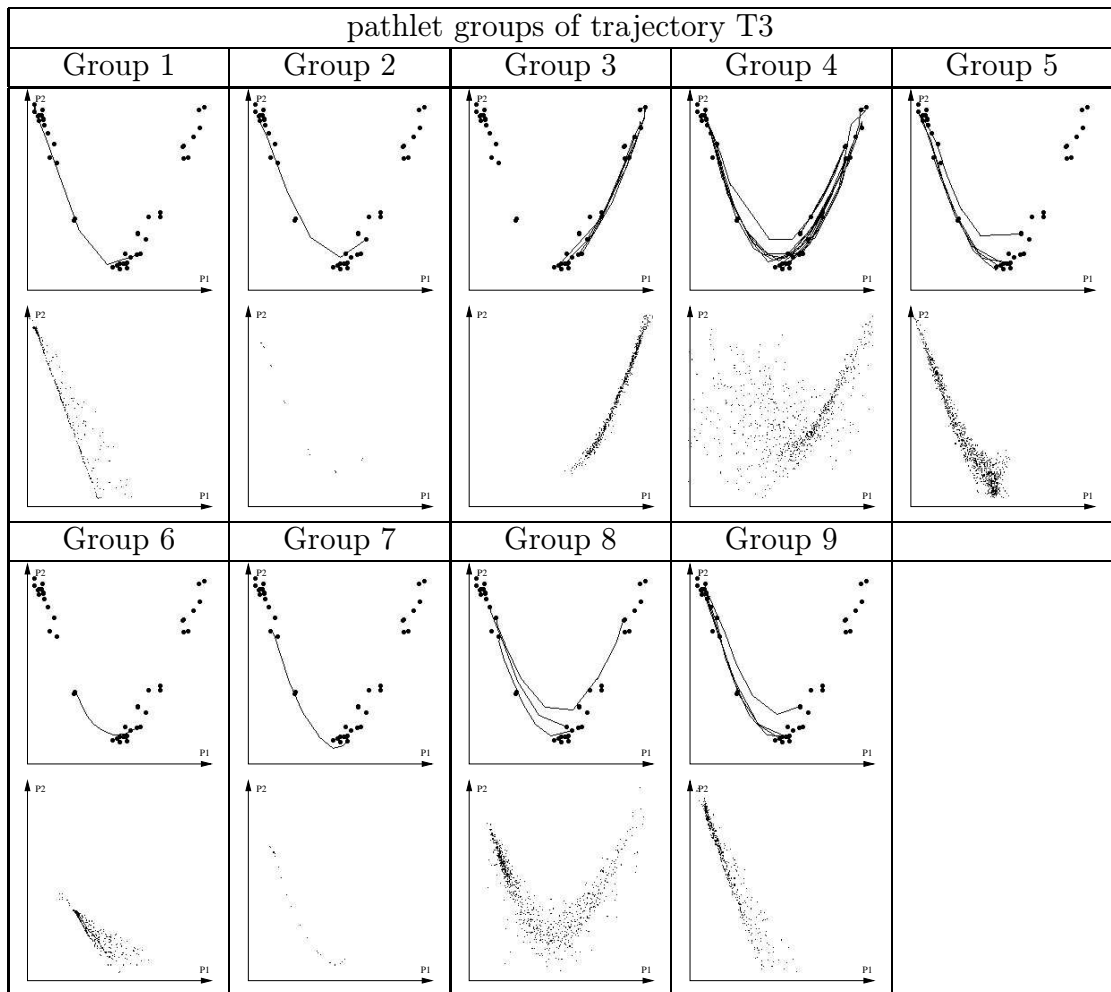


Table 5.6: Pathlet groups for trajectory T3 (groups 1 to 9) based on dynamic time warping and normalised cut clustering. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model learnt from that group, one hundred generated pathlets are drawn in the lower figure.

mean shift based algorithm (A4) as in the previous section.

Table 5.7 shows the pathlet groups extracted from trajectory T1. As in the previous section, for each cell of the table, the figure at the top represents the pathlets used to create the pathlet group along with the nodes extracted from the original trajectory, while at the bottom one hundred pathlets have been generated using the pathlet model. Most types of pathlet model can be found.

For instance, sets of pathlets have been correctly grouped (group 4 or group 11). A group containing a single pathlet can also be seen (group 12). Some groups contain outliers (group 1 contains two pathlets that end in different locations to the other pathlets in the group). However, the associated pathlet model seems to model the original data quite faithfully. No pathlet that looks impossible has been generated from the pathlet model learnt from group 1.

Pathlet groups 2 or 8 are also amongst those that contain an outlier. The corresponding model sometimes generates pathlets with negative timings. As in the previous section, this is due the fact that very short pathlets have been included in the model. However, we can see that this only happens if the other pathlets in the group are short themselves. After discarding a pathlet with negative timings, the new sampled pathlet is likely to be reasonable.

We can argue that group 6 in table 5.7 contains pathlets that do not look like each other, but once again the model represents the original pathlets rather well.

We can see on tables 5.8 and 5.9 that the pathlet models encode the same structures as the original pathlets used to build the models. Only similar pathlets are grouped together. The groups that have been created with only one pathlet only contains the outliers. Indeed, pathlet groups like group 20, 21, 22 or 23 contain pathlets that are not common. This is mainly due to errors in the selection of the nodes in the original trajectory. For instance the pathlet in group 21 should have been split into 3 pathlets.

The greedy algorithm is able to cope with imperfections in the node selection process. This quality of the greedy algorithm is useful since the node selection algorithm is not always perfect. The normalised cut algorithm, along with the similarity measure based on the dynamic time warping algorithm, is not always able to recover proper pathlet groups in the case of a failure to select good nodes in the original trajectory. For instance, the pathlets in the group 19 on table 5.9 have been clustered has a part of the group 2 in table 5.4. The pathlets in group 19 on table 5.9 should have been split into two pathlets each with extra

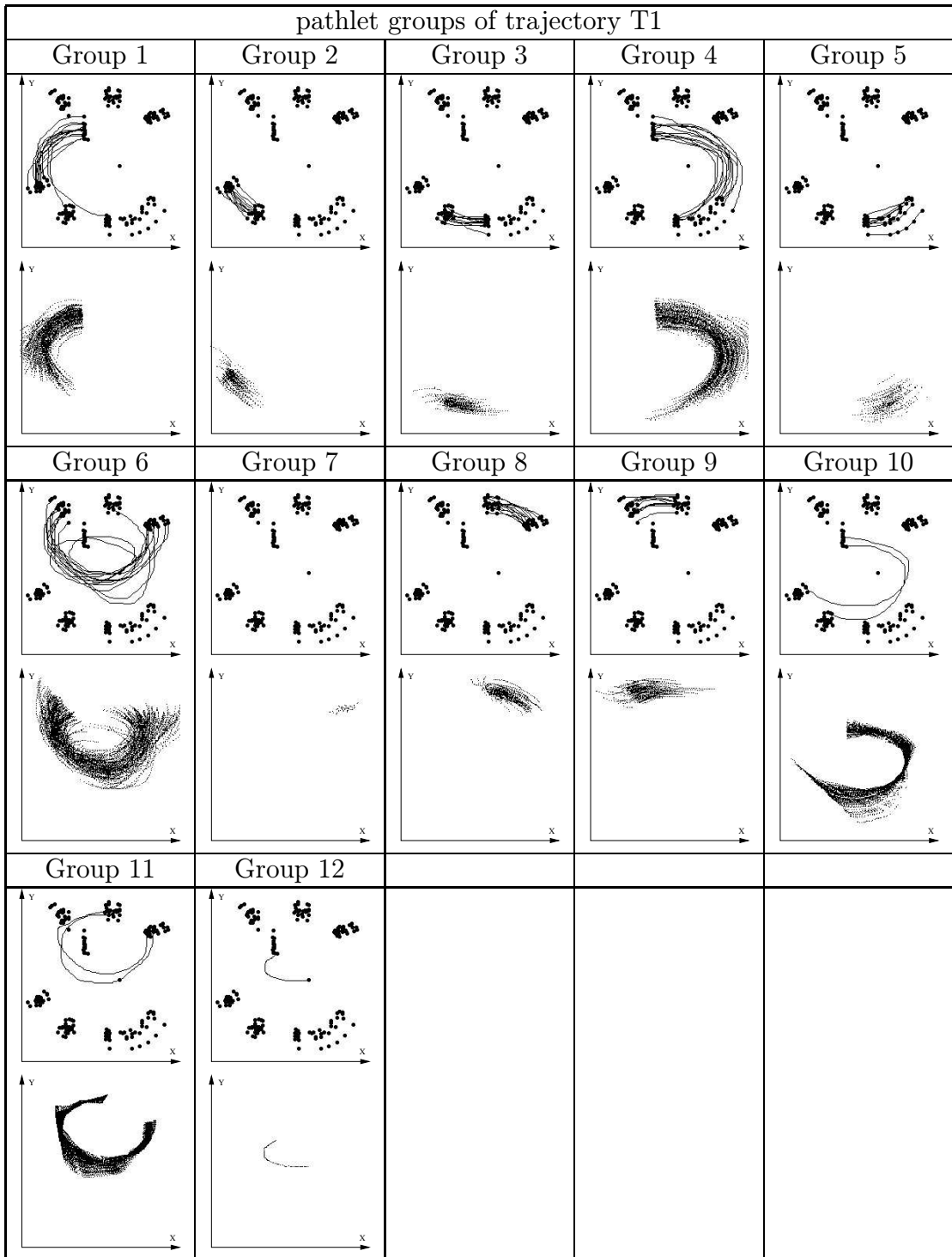


Table 5.7: Pathlet groups for trajectory T1 (groups 1 to 12) based on the greedy algorithm. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the pathlet model learnt from that group, one hundred generated pathlets are drawn in the lower figure.

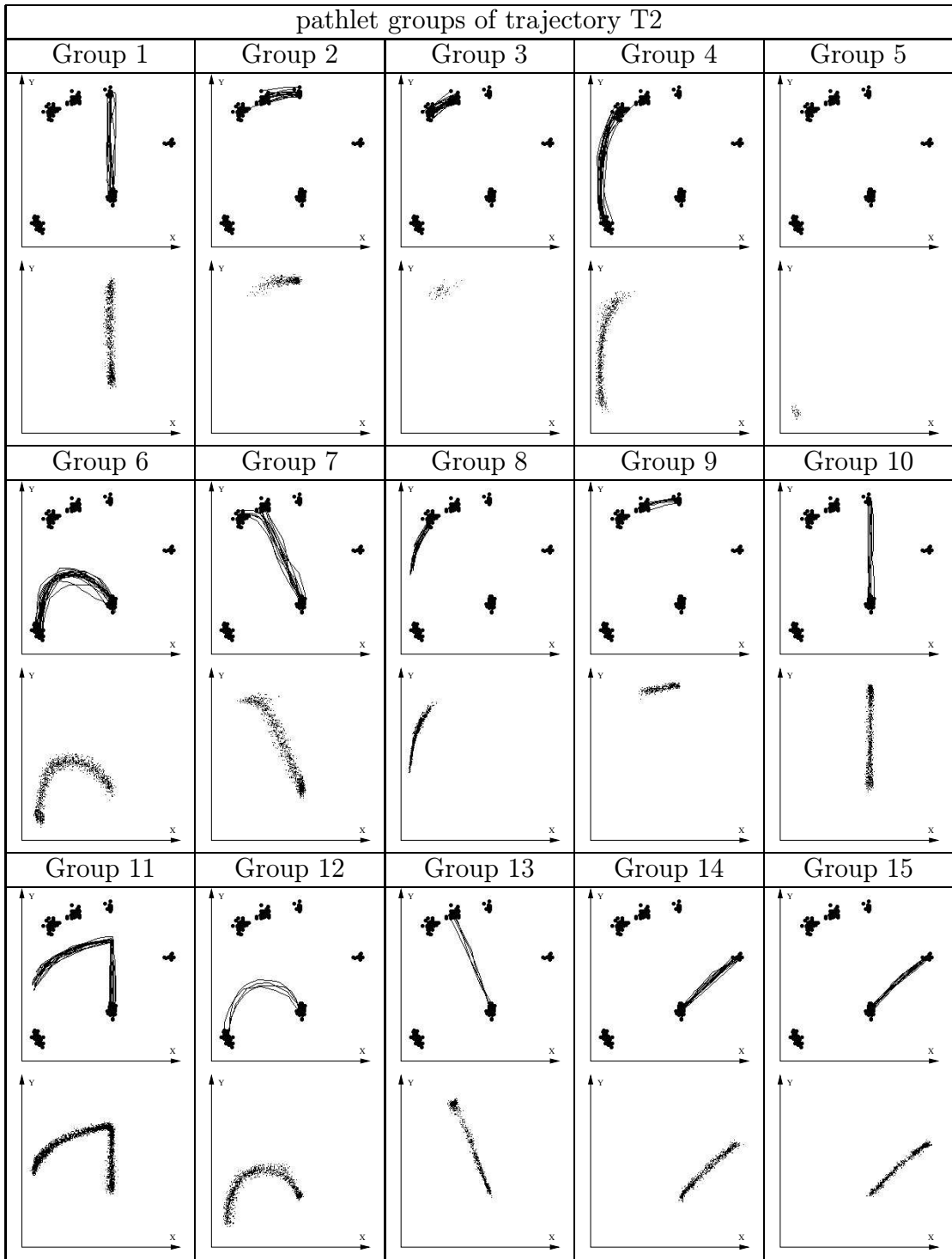


Table 5.8: Pathlet groups for trajectory T2 (groups 1 to 15) based on the greedy algorithm. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the corresponding pathlet model, one hundred generated pathlets are drawn in the lower figure.

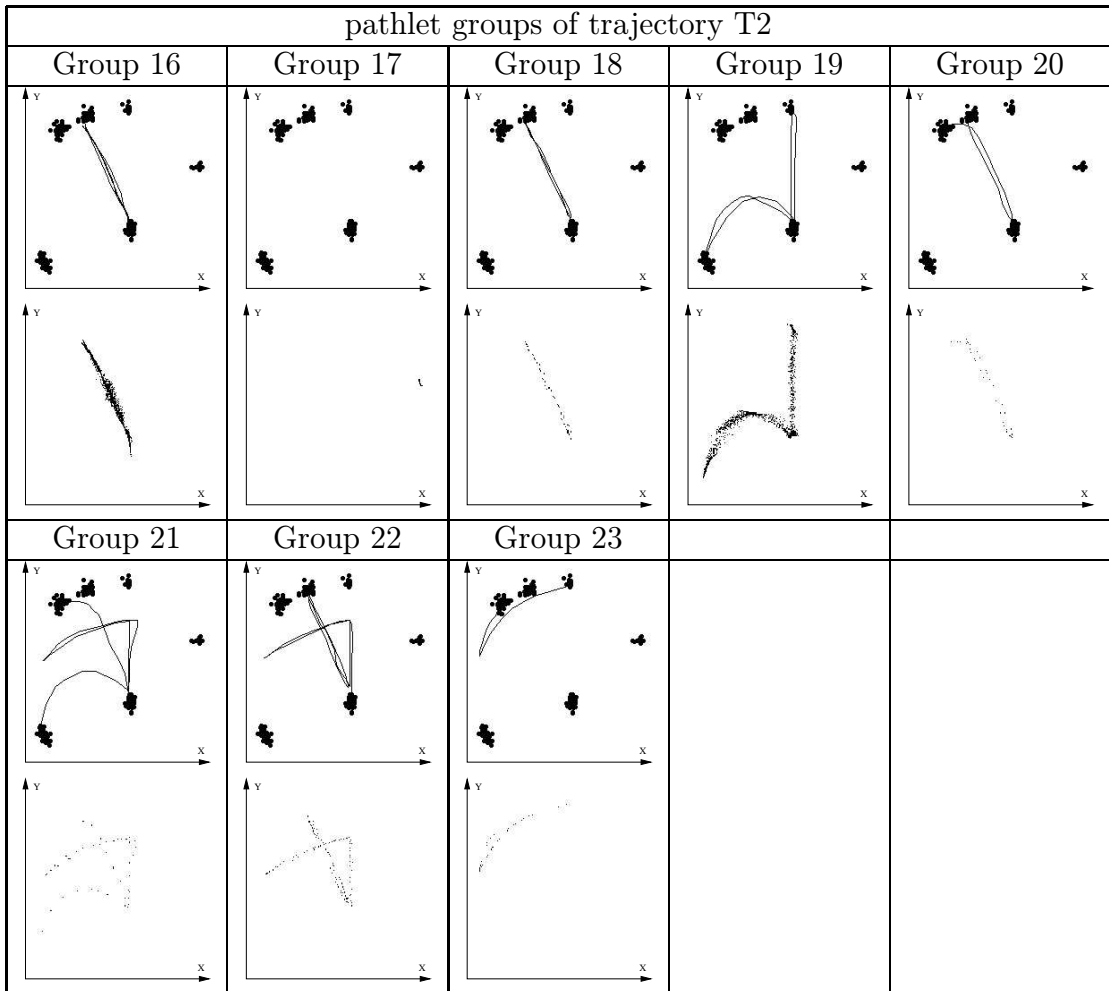


Table 5.9: Pathlet groups for trajectory T2 (groups 16 to 23) based on the greedy algorithm. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the corresponding pathlet model, one hundred generated pathlets are drawn in the lower figure.

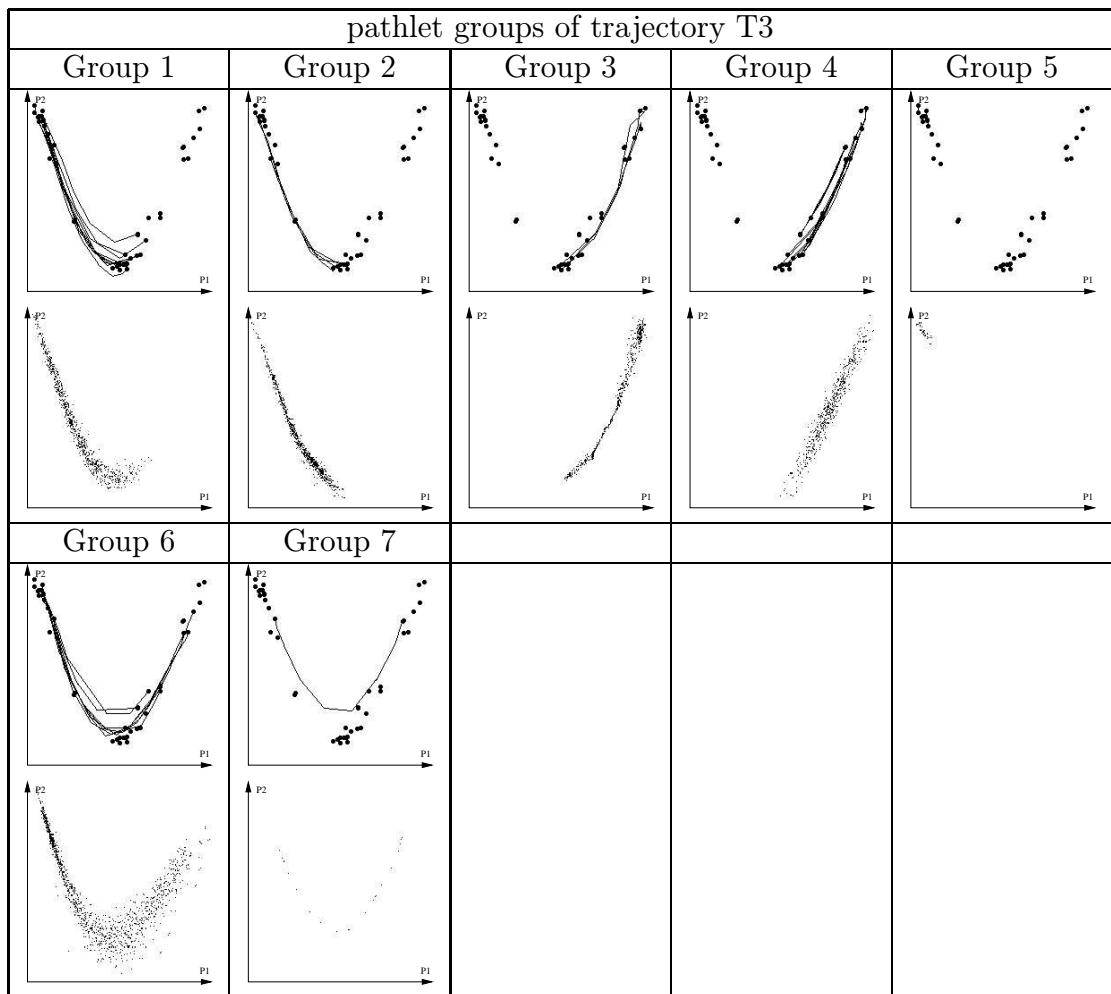


Table 5.10: Pathlet groups for trajectory T3 (groups 1 to 7) based on the greedy algorithm. For each group, the nodes are drawn in the upper figure with the pathlets forming that group. Using the corresponding pathlet model, one hundred generated pathlets are drawn in the lower figure.

nodes. The wrong clustering in table 5.4 is due to the fact that the dynamic time warping tends to match those pathlets with one of the two pathlets that would have been used instead. It compares badly extracted pathlets with pathlets that are similar to parts of those badly extracted pathlets.

This is a general problem of the dynamic time warping algorithm: it matches parts of pathlets and shrinks the time of the remainder of the pathlets to zero for the comparison, thus forcing wrong matchings.

Table 5.10 shows that the greedy algorithm performs correctly on data extracted from a video of a face (trajectory T3 corresponding to the video sequence V1 described in section 4.3.1).

We can also note that, for each trajectory (T1, T2 or T3), the greedy algorithm requires fewer pathlet groups than the algorithm based on the normalised cuts and the dynamic time warping algorithms for visually similar pathlet quality.

Finally, we can note that, with the greedy algorithm, some pathlet groups appears very similar. Two corresponding models generate pathlets in opposite directions. Similar pathlets with two different directions cannot be modelled properly by the same pathlet model. Examples of such corresponding pathlet groups include the pathlet groups 1 and 10 and the pathlet groups 6 and 12 in figure 5.8. Pathlet groups extracted from real video sequences also show such correspondences as we can see on figure 5.10: the pathlet group 1 corresponds to the pathlet group 2 and the pathlet group 3 corresponds to the pathlet group 4.

5.6 Conclusion

This chapter explained the approach used to model fragments of behaviour. We split the original trajectory into pathlets by extracting nodes from the trajectory. These are grouped, and each group is modelled with a Gaussian spatiotemporal representation.

Several algorithms have been investigated for node selection. For the remainder of the thesis, we use algorithm A4 (page 76). This algorithm, based on the mean shift algorithm, gives a better distribution of splitting nodes in the trajectory.

A spatiotemporal model of a pathlet group has been derived. This model has been extended to model the residuals as well. However, it is not clear that this extended model improves the quality of generated pathlets. In chapter 8, we

assess the whole framework with and without the residual modelling.

We have developed two algorithms for grouping pathlets. We have shown visually how the greedy algorithm of section 5.4.3 outperforms the clustering of pathlets based on the normalised cut algorithm with a similarity measure based on the dynamic time warping algorithm.

The greedy algorithm is able to create pathlet groups whose models best represent the data, that is the groups they have been trained on. The approach using the normalised cuts algorithm and the dynamic time warping algorithm has more trouble creating proper pathlet models. The extracted models can generate unlikely pathlets. The main reason for that is due to the similarity measure which is not strict enough when comparing the pathlets. It can favour grouping of pathlets that only have small parts that match together.

Even if the grouping of pathlets seems to give good clusters⁴, it does not mean that those groups are well modelled by a linear pathlet model. For grouping, the algorithm, and especially the criterion used for comparing pathlets in this algorithm, has to be adapted to the situation as we adapted the node extraction process to the model of a group⁵.

In the remainder of the thesis, we use the greedy algorithm to extract pathlet groups from a trajectory.

In the next chapter, we describe the variable length Markov model and how it can be used to model the sequence in which each pathlet model should be visited.

⁴as it is the case for the clustering algorithm based on the normalised cuts and the dynamic time warping algorithms.

⁵remember that we select nodes that are close to each other to have better chances to group the extracted pathlets together by a linear model.

Chapter 6

Variable length Markov model

6.1 Introduction

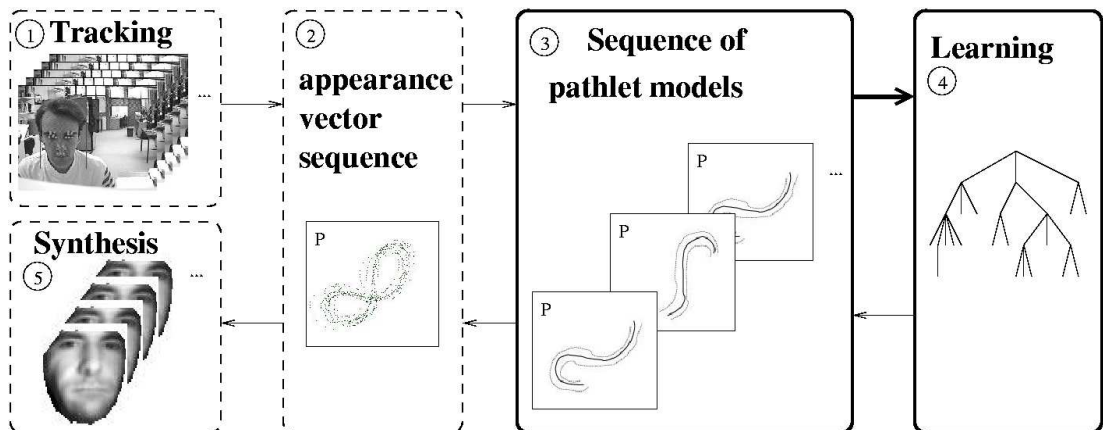


Figure 6.1: Overview of the components of the model. This chapter explains how to model temporal relationship between pathlet models.

The previous chapter demonstrated how to extract a sequence of pathlet models from a trajectory of active appearance model parameters. We now describe how a variable length Markov model can be used to model the sequence.

In the following, we describe the VLMM along with the different components that can be used within its learning scheme. Figure 6.1 shows how the VLMM module fits into the whole framework of our model.

6.2 Encoding transition probabilities

6.2.1 The storage of transitions

Let \mathcal{S} be a sequence of states $\{\mathcal{G}_n, \mathcal{G}_{n-1}, \dots, \mathcal{G}_2, \mathcal{G}_1\}$. We want to predict or generate the next state \mathcal{G}_0 given the history \mathcal{S} . In order to do that, we need the probability:

$$P(\mathcal{N} | \mathcal{G}_n, \mathcal{G}_{n-1}, \dots, \mathcal{G}_2, \mathcal{G}_1)$$

for all possible states \mathcal{N} .

We can use Markov models to learn such probabilities from typical sequences [77]. In this case, as we want to predict a pathlet model given an history of n pathlet models, we are interested in n -th order Markov models or n -th order hidden Markov models.

Unfortunately, high order Markov models can be vastly more expensive than their first-order counterparts. The memory required to store all the transition probabilities and the processing time required to train them are serious issues especially for large number of states.

Here the states are defined by the pathlet models. We call them: “pathlet states”. Depending on the method used to find the pathlet models (see chapter 5) and the training video sequence, the number of pathlet models found can potentially be huge. The size of the pathlet model set can even be larger if we wish to model interaction¹. Some form of dimensionality reduction is thus required.

The variable length Markov model provides an efficient way of storing the transition probabilities [81]. It is an approximation of a n -th order Markov model where the transitions that do not bring useful information are discarded.

6.2.2 Towards a more effective storage of transitions

A Markov model of N states needs to store a matrix of transition probabilities of size N^2 .

Furthermore, in a standard Markov model, with our kind of data, many transition probabilities will be either zero or insignificant. Indeed, in a video sequence

¹If we model interaction by modelling the joint probability of the two speakers, the cardinality of the resulting pathlet model set could be as large as the product of the cardinality of the pathlet model sets modelling each speaker individually.

of an object, the trajectory of the object is often continuous and does not jump randomly from one position to another. When the object is in a position corresponding to one pathlet model, only a few pathlet models can be used for the next position in the sequence. In the transition matrix, all of the impossible configurations are modelled by a probability of zero.

In some other situations, the transition probability is very low. Such situations can include atypical behaviours that happened only once during the training stage. In this case, we may want to discard these transitions since they do not reflect representative situations. A low probability transition can also come from the hardware used to capture the video sequence. A slow computer may drop some frames when the user is running a resource consuming process. This will result on sudden jump of the object within the sequence, and so an atypical behaviour. The same effect can be achieved if a recorded video sequence has not been correctly encoded in digital format.

It would be better not to store these values as they are generally unnecessary. The variable length Markov model gives a solution to this problem because it discards small probabilities and also transitions that do not add any useful information to the model. The transition probabilities storage needs to be rearranged for efficiency.

In the VLMM model, the transition probabilities are encoded in a tree. Figure 6.2 shows such a tree. Figure 6.2(a) represents the standard storage in VLMM as described in [81]. Figure 6.2(b) represents the way we have stored the probabilities. We can easily map between the two. Our storage is more efficient in retrieving probabilities for our application, but the gain is negligible compared to the other sources of computation.

Each branch of the tree represents a transition probability. In this case, the set of pathlet models is $\{A, B, C\}$. A pathlet model, a position in the history and a vector of probabilities is associated with each node. The size of the vector is the same as the size of the set of pathlet models. Each element in the vector corresponds to a immediate child. If the value in the vector is null, the corresponding child is not drawn on the tree. On figure 6.2(b), it stores the joint probability of having this child and all the pathlet models seen so far on their respective positions in the history.

For instance, the node B_2 corresponds to the pathlet model B at the position 2 in the history. The second element in the vector associated with the node

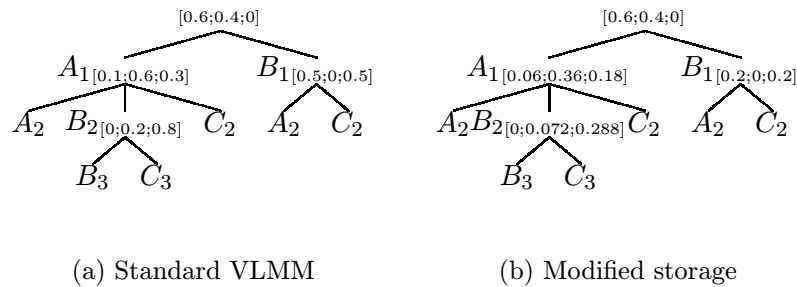


Figure 6.2: Example of storage of transitions in a VLMM model. The two way of storing the probabilities are equivalents. 6.2(a) stores conditional probabilities while 6.2(b) stores joint probabilities. From 6.2(a) we can deduce for instance the probabilities $P(\mathcal{G}_3 = B | \mathcal{G}_2 = B, \mathcal{G}_1 = A) = 0.2$, or $P(\mathcal{G}_2 = C | \mathcal{G}_1 = A) = 0.3$. From 6.2(b) we can deduce the probabilities $P(\mathcal{G}_3 = B, \mathcal{G}_2 = B, \mathcal{G}_1 = A) = 0.072$, or $P(\mathcal{G}_2 = C, \mathcal{G}_1 = A) = 0.18$.

B_2 correspond to the probability of B (the second pathlet model) being at the position 3 (one step further in the history) given that the total sequence ends with the subsequence B, A . Using mathematical notation, the second element in the vector associated with the node B_2 is the probability $P(\mathcal{G}_3 = B, \mathcal{G}_2 = B, \mathcal{G}_1 = A)$ that is the probability of having the sequence B, B, A . The end of this sequence corresponds to the nodes B_2 and A_1 that need to be retrieved from the tree to reach B_3 .

On figure 6.2(a), the vector stores the probability of having a child at its corresponding position in the history given that all the nodes seen so far are placed on their respective positions in the history.

For instance, the second element in the vector associated with the node B_2 corresponds to the probability of B being the third in the history given that B is the second and A is the first, that is: $P(\mathcal{G}_3 = B | \mathcal{G}_2 = B, \mathcal{G}_1 = A)$.

This tree representation of the storage is also called a prediction suffix tree. It is proved in [82] that a prediction suffix tree is equivalent to a finite state automata. It is also equivalent to a transition matrix as used by Markov models.

6.3 Training a VLMM

6.3.1 Definitions

Variable length Markov models (VLMM) were designed to learn text, that is sequence of letters. We will keep the original terminology and map it to our problem. In the next sections, we will use the following definitions.

An alphabet Σ is a set of predefined distinct entities. In the case of a sequence of letters, the entities would be letters. In the case of a sequence of a face, the entities would be the pathlet models. We will refer to these entities as being letters to stick to the original definitions given by Ron *et al.* [81]. $|\Sigma|$ denotes the size of the alphabet.

A string is a sequence of letters and is denoted by $s = s_1s_2s_3 \dots s_n$ where n is the length of the sequence. $|s|$ denotes the number of letters in the string s . We denote by e the empty string ($|e| = 0$). Σ^* is the set of all strings over Σ , Σ^L is the set of all strings of length L over Σ , and $\Sigma^{\leq L}$ is the set of all strings of length at most L over Σ .

A prefix of a string s of length n is denoted by $prefix(s) = s_1s_2s_3 \dots s_{n-1}$. In the same manner, a suffix of a string s of length n is denoted by $suffix(s) = s_2s_3 \dots s_n$. The set of all suffixes of a string s of length n is $suffix^*(s) = \{s_i \dots s_n | 1 \leq i \leq n\} \cup e$.

A string p is a suffix extension of s if and only if s is a suffix of p , that is $s \in suffix^*(p)$.

6.3.2 The learning algorithm

6.3.2.1 General idea

The idea behind the learning algorithm is that for different sequences, we need different lengths of memory for the prediction of the next element. For instance, if the learning has been done with many instances of sequences such as $\{a,c,b,d,e,f\}$, $\{e,c,b,d,e,f\}$ or $\{d,c,b,d,e,f\}$ then we would like to use a memory of four letters: $\{c,b,d,e\}$ in order to predict the next letter “f”. If only sequences like $\{a,l,k,j,e,f\}$, $\{s,d,l,l,e,f\}$ or $\{s,l,o,a,e,f\}$ occurs, then we want to use only the necessary information to predict “f” that is the sequence of one letter: $\{e\}$. We do not model the patterns in the sequence that are not important for the prediction.

In a more formal way, we wish to learn a model of the distribution of probabilities of strings in the sequence as suggested by the figure 6.2(b) but with a minimum number of nodes. The idea is to build a tree that gives the same probability values as a previously chosen probability measure on the learning sequence. The section 6.3.3 describes the probability measures that we have tried on a simplified example and how they performed. For now we can use the notation \tilde{P} to denote this probability measure.

6.3.2.2 The pruning of nodes

The construction of the tree is iterative and begins with the root node. An empty string e is assigned with the root node. We then add candidate nodes up to a depth L . If the probability of a string is small then the node should not be in the tree. We decide whether a probability is small by comparing it with a predefined threshold that represents the accuracy of the model. This threshold is defined by the user. We denote it by ϵ .

There is another reason not to add a node in the tree: if the addition of the node gives a new tree that is statistically equivalent to the previous one. We need a way of checking if two distributions of probability are equivalent. In section 6.3.4 we describe the different measures of distance between probabilities that we have investigated and how they behave. For now, we denote by $D(P_1||P_2)$ a measure of the distance between two probability measures P_1 and P_2 over the observation space.

The distance D can measure the amount of additional information we gain by adding the suffix σs to the tree instead of pruning the node σ and keeping the suffix s to describe the distribution. Given this measure of distance between two probabilities $\tilde{P}(\cdot|\sigma s)$ and $\tilde{P}(\cdot|s)$, we can construct a statistical error measure by weighting it with the prior probability of observing the suffix σs . Indeed, the distance $D(\tilde{P}(\cdot|\sigma s)||\tilde{P}(\cdot|s))$ can be large even if the probability of observing the sequence σs is very small. Weighting the distance by $\tilde{P}(\sigma s)$ allows us to neglect those cases. The statistical difference is then measured by

$$Err(\sigma s, s) = \tilde{P}(\sigma s) \cdot D(\tilde{P}(\cdot|\sigma s)||\tilde{P}(\cdot|s))$$

Err is called the statistical error measure.

6.3.2.3 The algorithm

Given the statistical error measure, we can construct the tree using the following algorithm:

- Initialisation step:
 - The tree T is initialised to a single node: e .
 - The set of candidate sequences S is initialised to $\{\sigma | \sigma \in \Sigma, \tilde{P}(\sigma) \geq \epsilon\}$.
- While there is still a sequence to be tested (that is, $S \neq \emptyset$), do:
 - 1) Pick the first candidate sequence $s \in S$ and remove it from the set S .
 - 2) If $Err(s, suffix(s)) \geq \epsilon$ then add to T the node corresponding to s .
 - 3) If $|s| < L$ then for every sequence $\sigma \in \Sigma$, if $\tilde{P}(\sigma s) \geq \epsilon$ add σs to the end of the candidate sequence set S .

6.3.3 The estimation of observed probabilities

One problem that arises when one deals with finite sequences is that of estimating the probabilities. The true underlying probability is not known. The probabilities must be estimated. We investigated several ways of estimating probabilities from sequences of letters (or pathlet models respectively). The details of the different laws of succession mentioned here can be found in [80].

We denote by n_s the number of times the sequence s is observed as being a subsequence of the training sequence. The training sequence is supposed to represent the population of sequences we will have to deal with, that is samples from the probability distribution we want to model. We denote by N_s the number of possible subsequences of size $|s|$ in the training sequence, that is:

$$N_s = \sum_{s' \in \Sigma^{|s|}} n_{s'}$$

Note that $N_s = L + 1 - |s|$, where L is the size of the training sequence.

Furthermore, the conditional probability $P(\sigma|s)$ is by definition:

$$P(\sigma|s) = \frac{P(s\sigma)}{P(s)}$$

We only need to estimate the probabilities of a subsequence s within a training sequence. The different laws of succession give us such estimates.

6.3.3.1 Laplace's law of succession

This is Laplace's first attempt to estimate probabilities of events occurring consecutively in a sequence. Its goal is to estimate the probability of an event occurring $n + 1$ times in succession given that the event occurred n times in succession.

The law can be found using the following reasoning:

Suppose that we have $N + 1$ coins that we label from 0 to N . Each coin has a different probability of landing heads up. The coin i lands heads up with a probability of i/N . We assume that the coins can be chosen with a uniform probability. We can then choose a coin and toss it again and again.

The probability that the first n toss are all heads for this particular coin is given by:

$$P(n \text{ heads} | \text{coin} = i) = \left(\frac{i}{N}\right)^n$$

So, the probability of observing n heads in a row in this experiment is:

$$F_{N,n} = P(n \text{ heads}) = \sum_{i=0}^N P(\text{coin} = i) \cdot P(n \text{ heads} | \text{coin} = i) = \frac{1}{N+1} \sum_{i=0}^N \left(\frac{i}{N}\right)^n$$

Furthermore:

$$P(n+1 \text{ heads} | n \text{ heads}) \cdot P(n \text{ heads}) = P(n \text{ heads} | n+1 \text{ heads}) \cdot P(n+1 \text{ heads})$$

If the first $n + 1$ toss are all heads, the first n toss are all heads too, so

$$P(n \text{ heads} | n+1 \text{ heads}) = 1$$

and

$$P(n+1 \text{ heads} | n \text{ heads}) = \frac{F_{N,n+1}}{F_{N,n}}$$

We want this to be true in general and not only for a finite number of possible probabilities $\left\{\frac{i}{N}\right\}_i$. In order to do that, we have to take an infinite number of coins with probabilities describing the possible range of probabilities $[0, 1]$. The probability we are looking for becomes:

$$P(n + 1 \text{ heads} | n \text{ heads}) = \lim_{N \rightarrow \infty} \frac{F_{N,n+1}}{F_{N,n}}$$

The properties of integrals give us:

$$\lim_{N \rightarrow \infty} F_{N,n} = \int_0^1 x^n dx = \left[\frac{x^{n+1}}{n+1} \right]_0^1 = \frac{1}{n+1}$$

So we can express the solution with this simple formula:

$$P(n + 1 \text{ heads} | n \text{ heads}) = \frac{n+1}{n+2}$$

This is a special case of the Laplace's law of succession. The key observation is that a uniform probability prior is required to apply the formula². In practice we do not know the distribution of the priors. Furthermore, the priors may depend on the context, so we cannot compute them and thus we have to make a choice. It has also been proved (see [46] for instance) that the Laplace's formula is general and gives the same result with other forms of prior probabilities.

In our case, the formula can be extended and we can compute the probability of a sequence s being sampled next given that we observed a sequence of N_s sequences [46]. An estimate of the probability of the sequence s is³:

$$P(s) = \frac{n_s + 1}{N_s + |\Sigma|}$$

This way of estimating the probabilities was used by Ron *et al.* to train a variable length Markov model [81].

6.3.3.2 The maximum likelihood estimate

Another empirical probability measure that can be used is the maximum likelihood estimate:

$$P(s) = \frac{n_s}{N_s}$$

²We used a uniform probability for the coin selection.

³The particular case demonstrated above computes the probability of having the event "the next toss is a head" given that we have observed the sequence of event where all the previous n tosses observed were heads. Thus we have $n_s = n$ because all the tosses observed were heads, $N_s = n$ because n toss were observed and $|\Sigma| = 2$ because there were 2 possible events for each toss: heads or tails.

This estimate is frequently used because of its simplicity, and because it generally gives correct results.

It was used by Guyon and Pereira [39] to train a variable length Markov model on strings. It has also been used by Galata, Johnson and Hogg [33] to train a variable length Markov model on the pathlet models derived from flow vectors.

6.3.3.3 Lidstone's law of succession

The Lidstone's law of succession is given by:

$$P_\lambda(s) = \frac{n_s + \lambda}{N_s + |\Sigma|\lambda}$$

where the parameter λ is in the range $[0, +\infty[$. It has been shown that this class of probability estimates is in fact a linear interpolation between the maximum likelihood estimate given in section 6.3.3.2 and the uniform prior $\frac{1}{|\Sigma|}$. Indeed, we can define a new constant μ by:

$$\mu = \frac{N_s}{N_s + |\Sigma|\lambda}$$

We then have a new form for the estimate given by Lidstone's law of succession:

$$P_\mu(s) = \mu \frac{n_s}{N_s} + (1 - \mu) \frac{1}{|\Sigma|}$$

It is interesting to consider particular cases of the Lidstone's law of succession:

- $\lambda = 0$ gives the maximum likelihood estimate.
- $\lambda = 1$ gives Laplace's law of succession.
- if λ tends to ∞ then we have the uniform estimate $\frac{1}{|\Sigma|}$.

λ thus represents the trust we have in relative frequencies. $\lambda < 1$ implies more trust in relative frequencies than the Laplace's law of succession while $\lambda > 1$ represents less trust in relative frequencies. In practice, people use values of λ in the range $[\frac{1}{32}, 1]$, a common value being $\lambda = \frac{1}{2}$ [81].

6.3.3.4 The natural law of succession

The natural law of succession presented in [81] is a more complex law of succession. It is based on the fact that simple sequences are more probable than complex

sequences. The probabilities are then estimated using a more appropriate subset of the alphabet. Alphabets are usually large, and so natural sequences do not include all the elements of the alphabet. The number of possible subsequences found in the observed sequence is given by:

$$q = |\{s | s \in \Sigma^*, n_s > 0\}|$$

The formula can then be derived as follow:

$$P(s) = \begin{cases} \frac{n_s+1}{N_s+|\Sigma|} & \text{if } q = |\Sigma| \\ \frac{(n_s+1)(N_s+1-q)}{N_s^2+N_s+2q} & \text{if } q < |\Sigma| \text{ and } n_s > 0 \\ \frac{q(q+1)}{(|\Sigma|-q)(N_s^2+N_s+2q)} & \text{otherwise} \end{cases}$$

It has been proved both in theory and in practice that this law of succession outperforms the previous ones.

Unfortunately, from a practical point of view, the natural law of succession is too computationally expensive. Indeed, the computation of the formula requires the value of q . The computation of q is done by counting the number of all sets of similar subsequences of any size in the observed sequence. Furthermore, the variable length Markov model learning algorithm uses extensively the estimation of observed probabilities. We cannot afford a loss of performance for this estimation so we will not consider the law of succession in the rest of this thesis.

6.3.4 Comparison of the probability distributions

The tree in the VLMM approximates a probability distribution. When building the tree, we need to test whether adding a particular node makes a significant difference to the accuracy of the approximation. We thus need methods of comparing distributions.

6.3.4.1 The Kullback-Leibler divergence

Let p and q describe two measure of probabilities, the Kullback-Leibler divergence is given by:

$$D_{KL}(p||q) = \int p(x) \ln \left(\frac{p(x)}{q(x)} \right) dx$$

In our case we have modelled the probabilities \tilde{P} by discrete values on a known alphabet Σ , so:

$$D\left(\tilde{P}(\cdot|\sigma s)\|\tilde{P}(\cdot|s)\right) = \sum_{\sigma' \in \Sigma} \tilde{P}(\sigma'|\sigma s) \ln \left(\frac{\tilde{P}(\sigma'|\sigma s)}{\tilde{P}(\sigma'|s)} \right)$$

Thus:

$$Err(\sigma s, s) = \sum_{\sigma' \in \Sigma} \tilde{P}(\sigma s \sigma') \ln \left(\frac{\tilde{P}(\sigma s \sigma') \tilde{P}(s)}{\tilde{P}(\sigma s) \tilde{P}(s \sigma')} \right)$$

This measure of a distance between probabilities has been used in [81] together with the Laplace's law of succession in order to correct corrupted texts using a variable length Markov model.

6.3.4.2 The Matusita distance

The Matusita measure between two probability distributions p and q is given by:

$$D_B(p\|q) = \int \left(\sqrt{p(x)} - \sqrt{q(x)} \right)^2 dx = 2 - 2 \int \sqrt{p(x) \cdot q(x)} dx$$

where x describe the whole space again. The term $\int \sqrt{p(x) \cdot q(x)} dx$ is called the Bhattacharyya measure [1]. In our case the distance becomes:

$$D\left(\tilde{P}(\cdot|\sigma s)\|\tilde{P}(\cdot|s)\right) = 2 - 2 \sum_{\sigma' \in \Sigma} \sqrt{\tilde{P}(\sigma'|\sigma s) \cdot \tilde{P}(\sigma'|s)}$$

Thus:

$$Err(\sigma s, s) = 2\tilde{P}(\sigma s) - 2\tilde{P}(\sigma s) \sum_{\sigma' \in \Sigma} \sqrt{\frac{\tilde{P}(\sigma s \sigma') \tilde{P}(s \sigma')}{\tilde{P}(\sigma s) \tilde{P}(s)}}$$

Unlike the Kullback-Leibler divergence, the Bhattacharyya term is symmetric and is invariant to scale in the case of two Gaussian probability density distributions. The Matusita measure inherits these properties.

6.4 Prediction using VLMM

Variable length Markov models provide compact models of sequences that have the same characteristics as traditional Markov models. In particular we can generate stochastic synthetic sequences from a model. In order to do that, we only have to sample a new element after the sequence s from the distribution

given by the set of probabilities:

$$\{\tilde{P}(\sigma|s)|\sigma \in \Sigma\}$$

where every element is proportional to the probabilities in the set:

$$\{\tilde{P}(s\sigma)|\sigma \in \Sigma\}$$

A maximum likelihood generation is also possible by taking the element that maximise the probability $\tilde{P}(\sigma|s)$.

In order to be able to compare similar measurements, we need to use the same observed history s for each probability computed for a prediction. It is not meaningful to compare probabilities of sequences that encode different lengths of history. We set the size of s to a predefined value. The sequences should be long enough to take all the useful information into consideration. A good value would be L , the maximum depth of the VLMM tree generated by the learning algorithm.

If the sequence $s\sigma$ is encoded in the VLMM tree, the corresponding probability is given by the vector associated with the parent node (the last node of the sequence $suffix(s)\sigma$). For instance, the probability of having the sequence $\{B, B, A\}$ in figure 6.2(b) is given by the second value of the vector associated with B_2 , that is 0.072. But what about the sequence $\{B, C, A\}$? Indeed, this sequence is not represented in the tree, so we cannot find the probability directly in one of the vector of the tree. In that case, we cannot really compute the value of the probability because we do not have enough information to do so. However, we can assume that the probabilities of the missing elements are uniform. Elements are missing because either:

- they have not been observed in this context in the training sequence, in that case we cannot know their real probabilities and we have to decide an empirical probability, or
- they have been pruned during the training algorithm because they did not bring any valuable information. We can chose a uniform probability in order to compare them to elements that might have brought information.

So, in order to compute the probability of the sequence $\{B, C, A\}$, we multiply the probability $\frac{1}{|\Sigma|}$ of having the element B followed by the sequence $\{C, A\}$ by

the probability of having the sequence $\{C, A\}$ (0.18 in our case).

If we require a longer generated trajectory, we can continue adding pathlet states to the sequence B, C, A in the same way to get a sequence of pathlet states \mathcal{S} of length n .

This algorithm needs initialising with a sequence of states. One way of doing it can be to assume a null history at the beginning. In practice, we just generate a random sequence of states of size n . We then add n states to that sequence and remove the n first states, giving us the sequence \mathcal{S} . We then only use the last n elements of \mathcal{S} to add the states to the sequence. If we choose n to be the maximum number of levels in the VLMM tree, all the other elements of \mathcal{S} will be associated with a uniform probability since they will not appear in the VLMM tree⁴. This is simpler to implement in practice since the history has a constant size. The states added to \mathcal{S} at the beginning are not very likely but become more and more likely as the last n elements of \mathcal{S} form a likely sequence of states.

Note that another possible generation method involves sampling directly branches of the VLMM tree. Even if it is a cheap way a generating sequences, it is not correct because it does not take the history into account. Histories should be used when the VLMM is used for prediction. We want to generate behaviour, so we need to look at the history for every element generated. Otherwise the generated sequence will look like small pieces of behaviour concatenated one after the other, without any natural link.

6.5 VLMM results

This section presents the results we obtained in the evaluation of the variable length Markov model. All results have been obtained using texts as learning sequences, allowing quantitative assessment of the performance of the algorithms. The results are summarised in tables 6.2 and 6.3 (page 124).

6.5.1 Comparison of the Lidstone probability estimation with the maximum likelihood probability estimation

We have compared the effect of using the Lidstone and the maximum likelihood estimation of probability during model training. The comparison is done twice:

⁴In practice we choose n as well as the depth of the VLMM tree to be 10.

once in conjunction with the Matusita distance and a second time in conjunction to the KL divergence. The experiments have been done on sequences of letters. The VLMM has been trained on English texts. The results are presented in a form of a tree where every element that has a probability higher than 0.003 is shown. The sequences should be read from the leaves to the root of the tree. For instance figure 6.3 shows the sequences “rea”, “ha”, “la” and so on, from left to right.

6.5.1.1 Comparison of trees built using the Matusita distance

For these experiments, the VLMM has been trained on a text (2144 letters). The training text tells the story of penguins and other animals.

Figures 6.3, 6.4, 6.5 and 6.6 represents the resulting tree when trained using the Matusita distance. The Lidstone estimation of probability has been used with λ varying from 0 to 1. The case $\lambda = 0$ corresponds to the maximum likelihood estimate and the case $\lambda = 1$ correspond to Laplace’s law of succession.

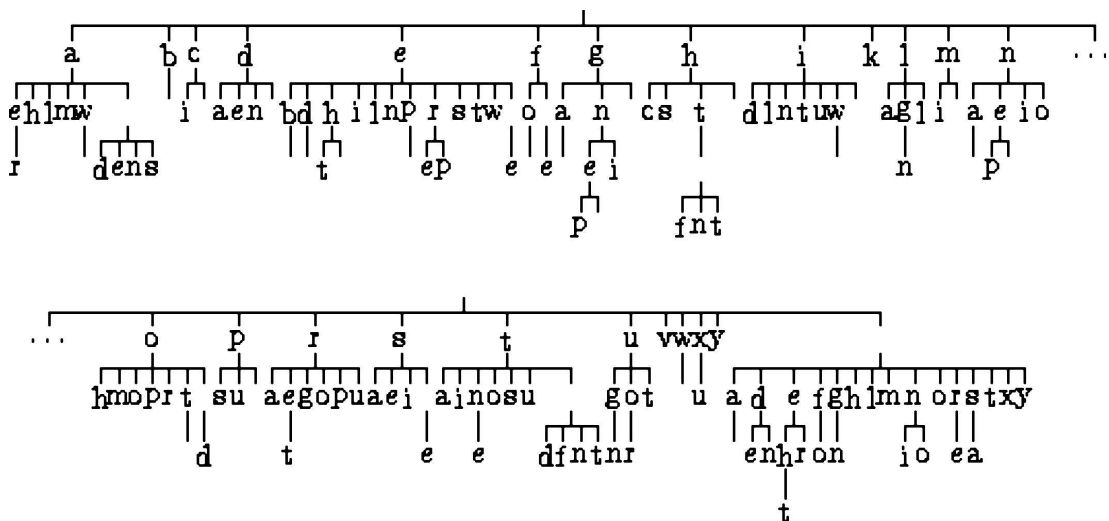


Figure 6.3: VLMM tree learnt using the maximum likelihood estimation of probability and the Matusita distance. ϵ is set to 0.003 and only the nodes with probabilities that are greater than 0.003 are shown on the graph.

We can see that the topology of the tree evolves a lot from the case $\lambda = 0$ to the case $\lambda = 1$.

For the case $\lambda = 1$ (figure 6.6), the depth of the tree is one. That means that the model does not take histories into account. It only models the probability

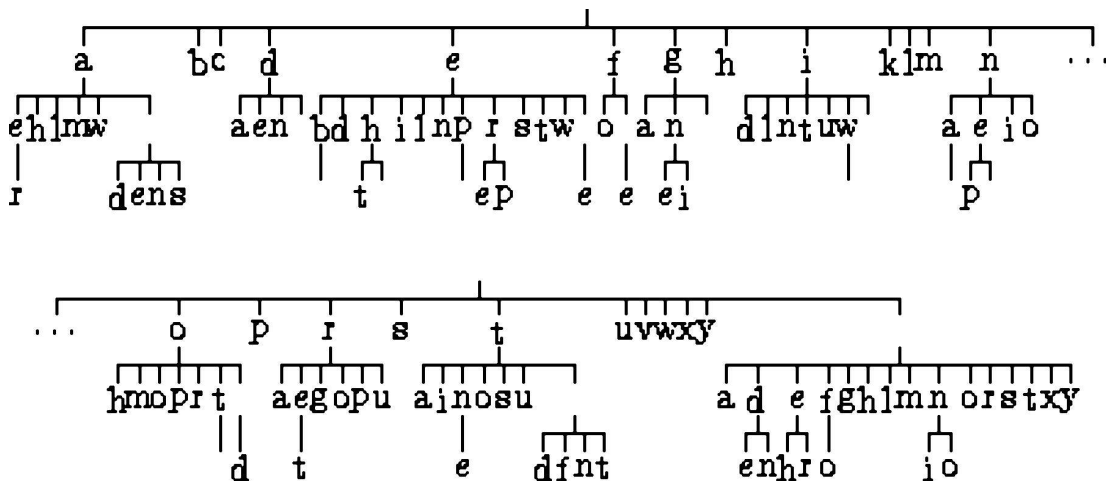


Figure 6.4: VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.05$ and the Matusita distance. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

of having a letter in the alphabet. We can notice that some letters like “j” or “q” are not in the tree because of their low probability in English texts, and in particular in our example text. As we have seen before, this is the problem with Laplace’s law of succession because it is built on the assumption of a uniform prior.

In the case $\lambda = 0$ (figure 6.3), the tree seems to have learnt the text in a more appropriate manner. We can find parts of some words like “peng” which comes from “penguin” (main subject of the text). We can also find small words such as “the”, “of”, “in”, “on” or “as”.

The two other cases are a transition between those two extremes.

This qualitative evaluation seems to be favorable to the maximum likelihood estimation. A quantitative evaluation of these trees is given later.

6.5.1.2 Comparison of trees built using the Kullback-Leibler divergence

We repeated the experiment using the KL divergence to compare probability densities. Figures 6.7, 6.8 and 6.9 show the resulting trees.

The method of estimation of probability seems to exert less influence on the result. In the case of the maximum likelihood estimate the tree does not change significantly compared to the corresponding case in the previous section. The

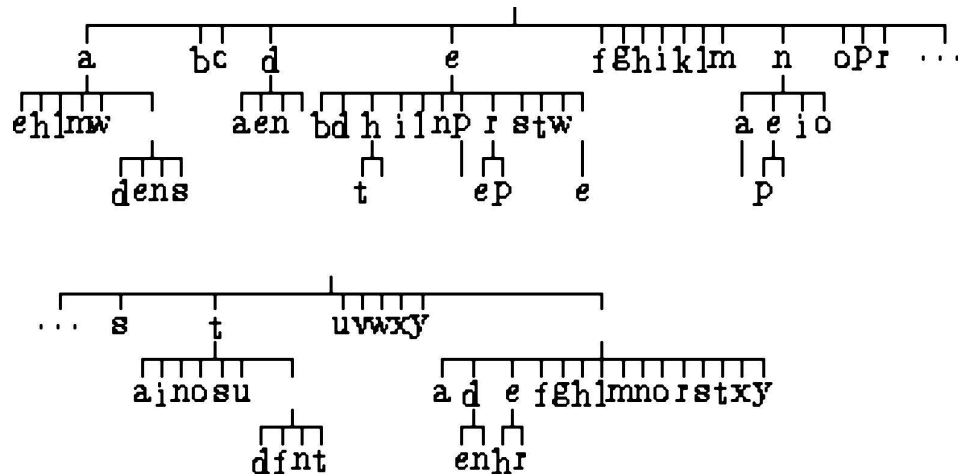


Figure 6.5: VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.1$ and the Matusita distance. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

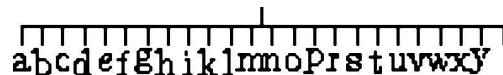


Figure 6.6: VLMM tree learnt using the Laplace estimation of probability and the Matusita distance. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

two other trees have grown.

The aim of the variable length Markov model is to reduce the number of links required to model the probability distribution. The size of the tree influences directly the learning because the more nodes there are in the tree, the more nodes the learning algorithm has to check. So it is possible that the KL divergence gives us a less efficient tree.

Due to the small amount of data used to construct these trees, the learning using the KL divergence in the two last cases can give such a tree because the text has been over learnt. In order to make sure that it is not the case, a further experiment has been done.

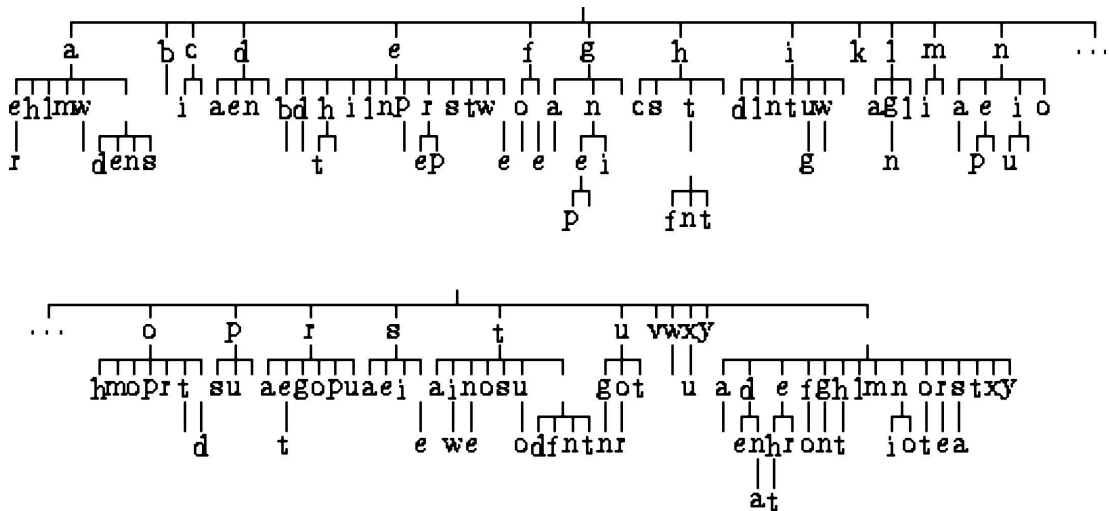


Figure 6.7: VLMM tree learnt using the maximum likelihood estimation of probability and the Kullback-Leibler divergence. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

6.5.2 Comparison of trees built using the Matusita measure with trees built using the Kullback-Leibler divergence for large texts

In order to have a correct qualitative comparison of the Matusita measure and the KL divergence, we trained a VLMM tree on a much larger text. The text was a compilation of journalistic style articles from the news, 70000 characters long.

Figures 6.10 and 6.11 show the learnt trees for a VLMM using a KL divergence and a Matusita distance respectively.

We can see that the two trees look similar. The tree learnt with a Matusita distance has a depth of six and the KL divergence gave a tree with a depth of three⁵. The method using the Matusita distance is able to encode more history in the tree while still not using so many nodes. It encodes small parts of words, but also short words such as “said”, “and”, “the”, “of” or “to”. The method using the KL divergence is not able to find words in the text, it only encodes parts of words.

The fact that humans group letters into words suggest that a word is a coherent sequence of information and the letters at the extremity of the word are less

⁵note that the depth of trees learn in section 6.5.1 was limited to four in order to limit the computational cost

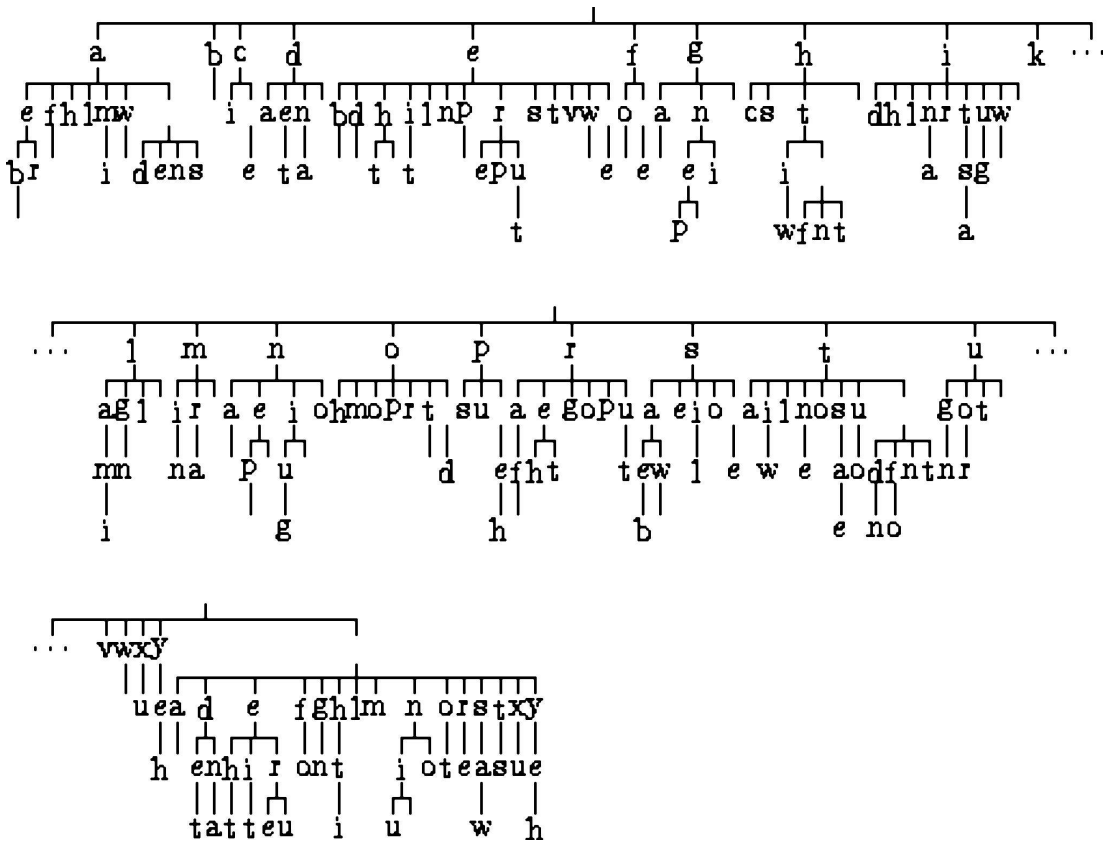


Figure 6.8: VLMM tree learnt using the Lidstone estimation of probability with $\lambda = 0.5$ and the Kullback-Leibler divergence. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

linked to the other words than to the word itself. This suggests that the method using the Matusita measure is better at representing groups of natural sequences.

6.5.3 Quantitative assessment of the prediction

In order to assess the prediction properties of the variable length Markov model, we used a test text and predict each letter using the sequence of previous letters. So, a sequence corresponding to the beginning of the test text is given to the VLMM and the prediction of the next letter is compared to the next letter in the test text. We records the proportion of correctly predicted letters.

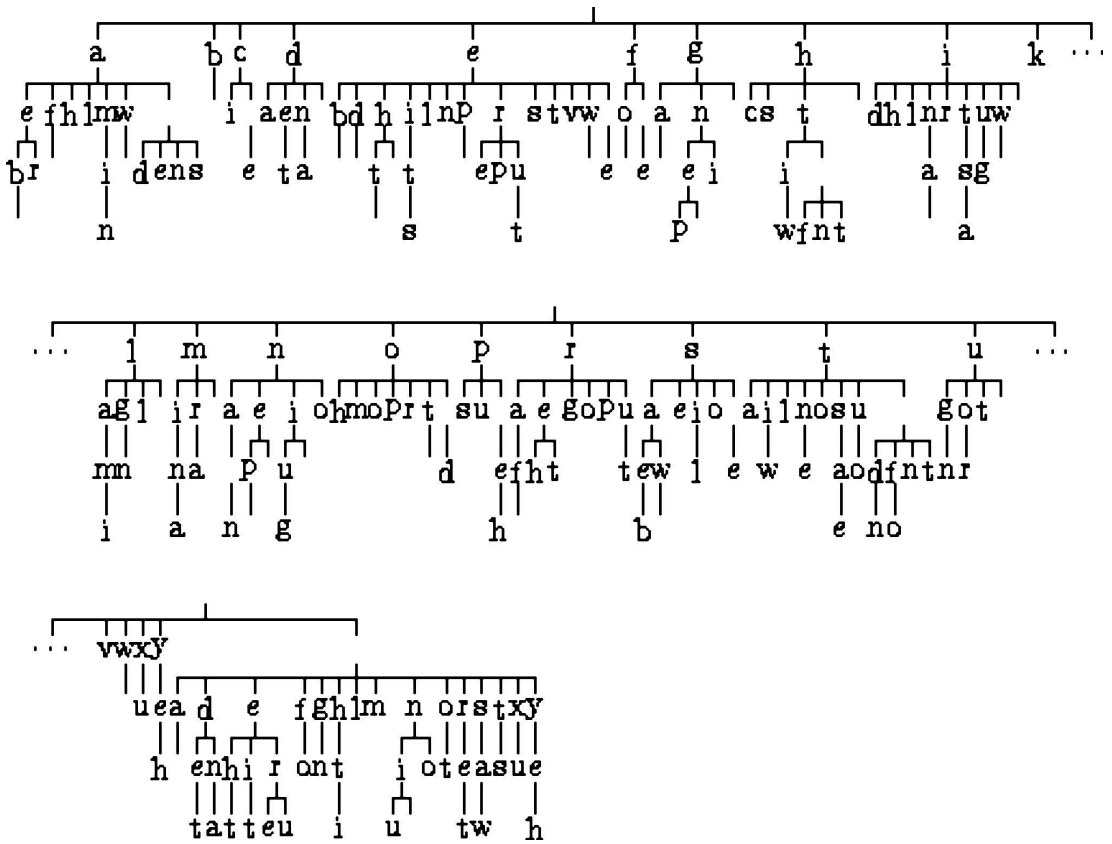


Figure 6.9: VLMM tree learnt using the Laplace estimation of probability and the Kullback-Leibler divergence. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

6.5.3.1 Performance given a large training set

In this section, the learning text E_1 is 70000 letters long. The learnt VLMM has been tested on 3 texts:

- E1 - the training text (70000 characters).
- E2 - a text similar in style and content (5218 characters).
- E3 - a completely different text (2144 characters).

The prediction capabilities (percentage of letters correctly predicted) are reported on table 6.1. The results show similar performances on each text, but the Matusita measure gives markedly better predictions than the KL divergence (note that a flat prior would achieve about $\frac{1}{|\Sigma|} \approx 3\%$).

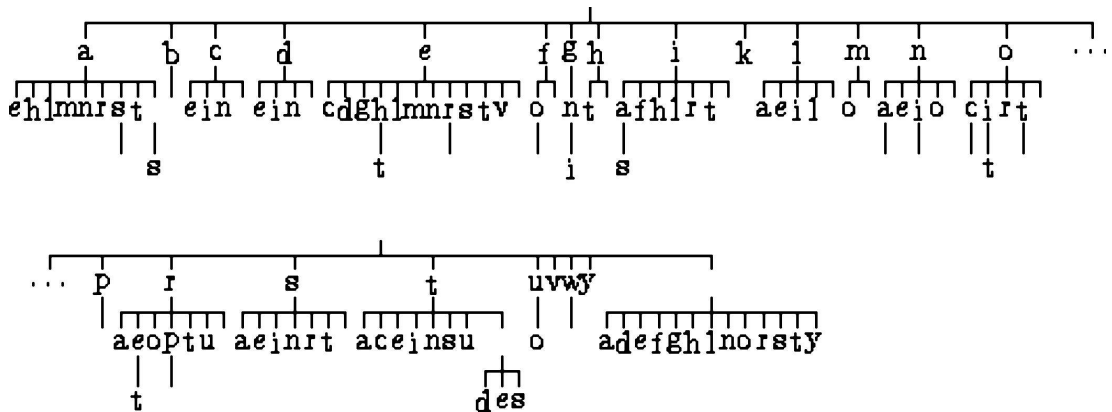


Figure 6.10: VLMM tree learnt using the maximum likelihood estimation of probability and the KL divergence. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

	E_1	E_2	E_3
KL	25.7%	26.4%	24.9%
Matusita	32.9%	34.8%	32.7%

Table 6.1: Comparison of the prediction of a VLMM learnt from a long text. The VLMM has been trained using the text E_1 .

The small differences between the percentage of correct guesses of the three different texts tends to suggest that the VLMM did not “over learn” the text, as its performance on the quite dissimilar test set E_3 was close to that on the training set E_1 .

6.5.3.2 Performance given a small training set

The test was repeated training on the shortest text E_3 . Results are reported in tables 6.2 and 6.3 for the Matusita distance and the KL divergence respectively.

The method using the KL divergence performs better this time. Associated with the Laplace’s law of succession, it is able to predict about 30% of a text dealing with another subject (E_1 and E_2).

In the case of the Matusita distance, the best performance is achieved with $\lambda = 0.05$. It seems that this time, the maximum likelihood estimate could not explain the data on its own. A small amount of uniform prior was important. This is due to the fact that there were insufficient training data. We cannot trust the frequencies counted in the training sequence as we could with the training on

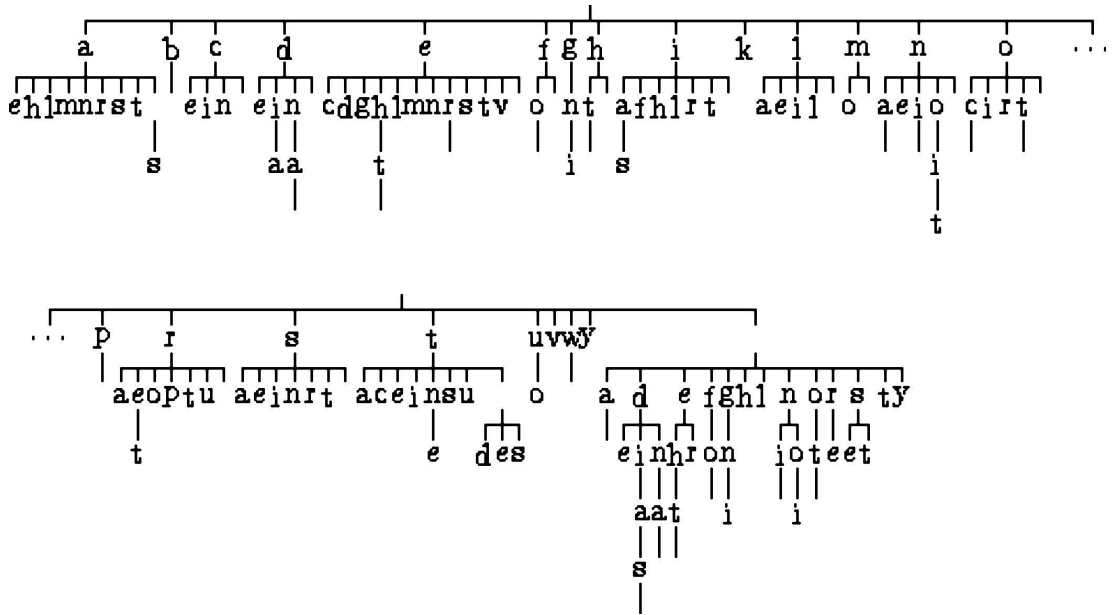


Figure 6.11: VLMM tree learnt using the maximum likelihood estimation of probability and the Matusita distance. ϵ is set to 0.003 and only the probabilities that are greater than 0.003 are shown on the graph.

a large dataset, so we need to include a part of uniform estimation which is done by increasing the value of λ .

6.6 Conclusion

This chapter has described the theory behind the variable length Markov model and the learning algorithm. One problem of the VLMM is the choice of ϵ . It would be better to be able to estimate it and thus to have a parameter free algorithm.

Tiño and Dorffner [45] try to solve this problem by using a completely different way of constructing the tree in the learning algorithm. They use a metric that represents how close two sequences are. This metric demands a parameter that can be interpreted as a learning parameter used in the same way as the learning parameter of a temporal difference learning [70]. A vector quantisation is then used to cluster subsequences into clusters that share the same suffix structure according to the metric. Unfortunately this approach does not eliminate the use of a parameter. However, it may be easier to find the right parameter for this

λ	E_1	E_2	E_3
0	20.2%	22.1%	30.8%
0.05	23.7%	25.5%	35%
0.1	24.0%	24.8%	32.7%
1	16.6%	16.5%	18.2%

Table 6.2: Comparison of the prediction of a VLMM learnt from a short text using the Matusita distance. The VLMM has been trained using the text E_3 . It has been tested on the three texts E_1 , E_2 and E_3 . The results are reported for Lidstone's law of succession for different values of λ . The case $\lambda = 0$ corresponds to the maximum likelihood estimate. The case $\lambda = 1$, which corresponds to Laplace's law of succession, completely failed since it only predicted spaces.

λ	E_1	E_2	E_3
0	20.9%	23%	31.6%
0.5	28.3%	30.1%	52.4%
1	28.3%	30.1%	52.4%

Table 6.3: Comparison of the prediction of a VLMM learnt from a short text using the Kullback-Leibler divergence. The VLMM has been trained using the text E_3 . It has been tested on the three texts E_1 , E_2 and E_3 . The results are reported for Lidstone's law of succession for different values of λ . The case $\lambda = 0$ corresponds to the maximum likelihood estimate. The case $\lambda = 1$ corresponds to Laplace's law of succession.

learning algorithm. Experiments show that the result is sometimes better than the VLMM algorithm and sometimes equivalent.

Different measures of probability and different distances to compare probabilities have been described. We have seen some examples of generated text as well as generated trajectories in the appearance parameter space.

It has been shown that the use of the Matusita distance in the learning algorithm and the use of the maximum likelihood produced the best estimates of the probability distribution. However, for short training sets, we cannot trust totally the observed data, thus a combination of Kullback-Leibler divergence and Laplace's law of succession is better.

Chapter 7

Generation of new behaviour movies

7.1 Introduction

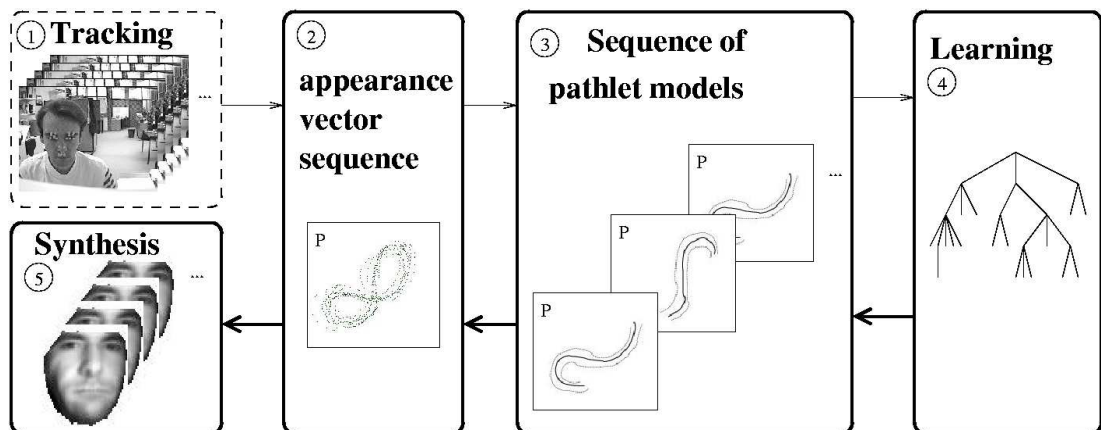


Figure 7.1: Overview of the components of the model. This chapter explains how to generate a new video sequence of facial behaviour given the models learnt in the previous chapters.

In chapters 3, 5 and 6, we explained how we are modelling the behaviour of a face from a video sequence. In this chapter, we show how we can use this model to generate a new video sequence of the same face that mimics its original behaviour. Figure 7.1 shows how this chapter fits into our framework.

In this chapter, we first show how the VLMM can explain the behaviour seen in a video sequence and how we can use it to generate new sequences of pathlet

models. We then describe how to generate a new trajectory in the appearance parameter space from that sequence. Finally, we show how to synthesise the resulting video sequence.

7.2 The VLMM tree for pathlet states: an explanation of the behaviour

One of our aims is to be able to understand the behaviour of the face in the original video sequence, by just looking at the model. The model should be able to explain the behaviour in an explicit way.

In this section, we show how the behaviour can be graphically represented. We take trajectory T2 as an example.

We have seen in section 5.5.2 that pathlet models can be represented effectively by generating a hundred pathlets from the pathlet model. We can then use this representation to draw a VLMM tree that stores the probabilities of pathlet model sequences. Such a tree is drawn on figure 7.2.

Each node on that tree stores a probability of a pathlet state, that is the probability of a sequence of pathlet models¹. In a way similar to the VLMM trees of letters shown in chapter 6.5, the tree has to be read from the leaves to the root node. For instance, the node A1 in figure 7.2 stores the probability of the sequence of pathlet models referred as A, which is composed of the pathlet models A1, A2 and A3 in that order. Similarly the pathlet model B1 stores the probability of the sequence of pathlet models B, composed of the pathlet models B1, B2, B3 and B4 in that order.

With this representation of the VLMM tree, we can see what the probabilities stored in the tree represent. The global shape on the part of trajectory encoded in each node can be seen directly from the tree. We only need to concatenate generated pathlets from the pathlet models represented in nodes A1, A2 and A3 to understand the possible trajectories generated from the sequence of pathlet models A.

Figure 7.3 illustrates the way the VLMM tree has to be read for the sequence of pathlet models B. The possible trajectories described by the sequence of pathlet models B is the concatenation of generated pathlets from the pathlet models

¹Note that the probabilities are not drawn on the tree of figure 7.2 for simplicity only

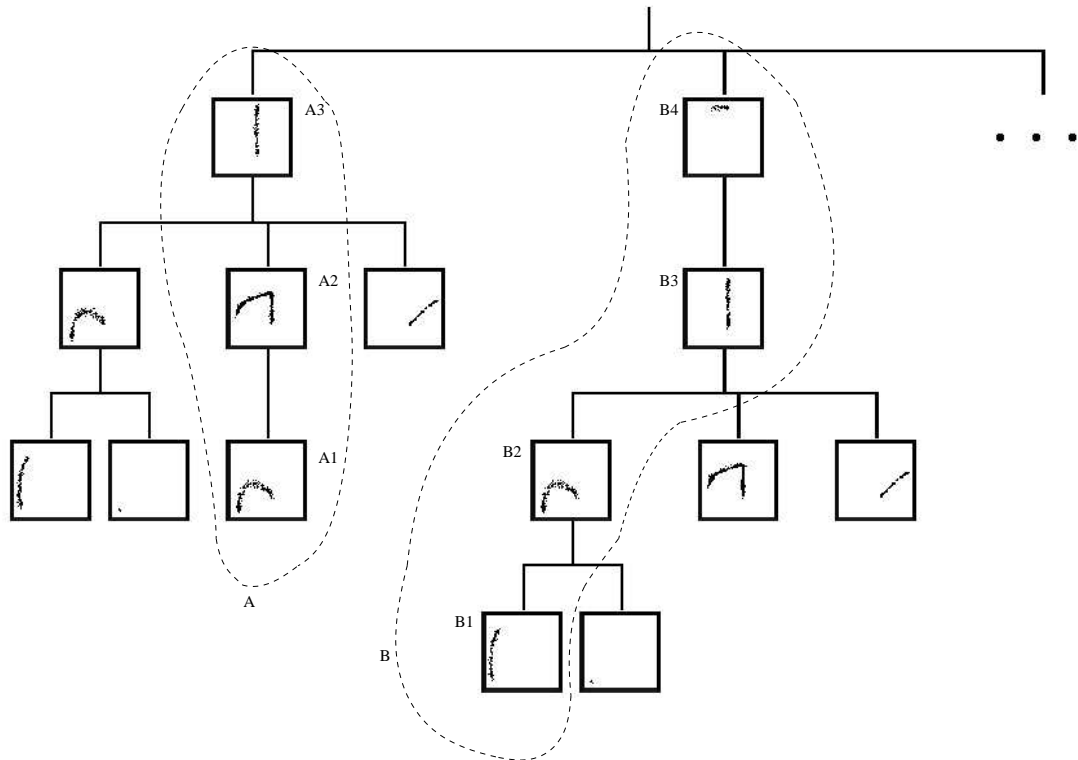


Figure 7.2: VLMM tree of pathlet states. The tree has been generated by learning the sequence of pathlet models extracted from trajectory T2. Only a part of the tree is shown.

B1, B2, B3 and B4. The arrows represent how the trajectory evolves through time. First we use points generated with the pathlet model B1, then points generated with the pathlet model B2 and so forth. Since one hundred pathlets have been generated from each groups on figure 7.3, the resulting drawing represents the distribution of possible trajectories that can be generated from the sequence B (the arrows show its direction). Another example of a sequence of pathlets extracted from the VLMM tree of figure 7.2 is shown on figure 7.4.

Note that on figures 7.3 and 7.4, the directions of the pathlets generated from the pathlet models have been highlighted with arrows. The representation of the pathlet models in figure 7.2 does not show those directions. It could be added to the representation in figure 7.2 by drawing the direction of the mean pathlet of the model. The direction could also be encoded in colour.

However, in this simple example, it is easy to guess the direction of the generated pathlets. The end of a pathlet should correspond to the beginning of the

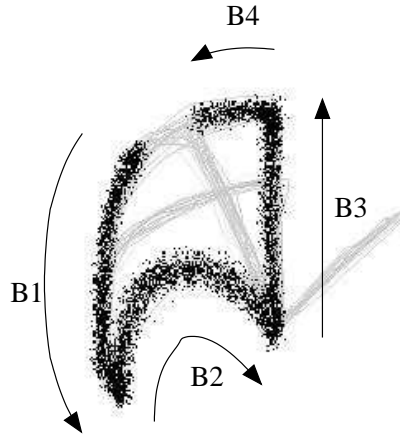


Figure 7.3: Representation of the sequence B, highlighted in figure 7.2. The trajectory generated by this sequence is a concatenation of pathlets generated by the groups B1, B2, B3 and B4. The arrows show how the trajectory evolves through time following pathlets from the groups B1, B2, B3 and then B4. The original trajectory used to teach the model is drawn in grey.

next pathlet. We can deduce the directions of pathlets by matching beginnings and ends of generated pathlets.

Looking at the probabilities stored in the VLMM tree, we can easily explain the behaviour encoded in the original trajectory. This gives an advantage over other models of behaviour that usually do not provide such an explicit and structured modelling, for instance models based on the autoregressive process.

7.3 Generating a new sequence of pathlet models

Suppose that we have already generated a sequence of pathlet states

$$\mathcal{S} = \{\mathcal{G}_n, \mathcal{G}_{n-1}, \dots, \mathcal{G}_2, \mathcal{G}_1\} \quad (7.1)$$

We want to extend that generated sequence with another pathlet state \mathcal{G}_0 .

In order to do that, we test every possible new sequence. For each pathlet state \mathcal{G}_0 , we extract the probability of the sequence $\{\mathcal{G}_n, \mathcal{G}_{n-1}, \dots, \mathcal{G}_1, \mathcal{G}_0\}$. This gives us a probability distribution for the choice of the pathlet state \mathcal{G}_0 . We can then stochastically sample from that probability to find the pathlet state to add.

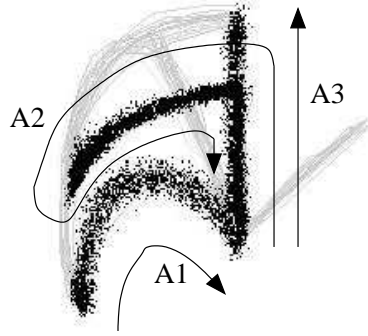


Figure 7.4: Representation of the sequence A highlighted in figure 7.2. The trajectory generated by this sequence is a concatenation of pathlets generated by the models A1, A2 and A3. The arrows show how the trajectory evolves through time following pathlets from the models A1, A2 and then A3. The original trajectory used to teach the model is drawn in grey.

Figures 7.2 and 7.5 show this approach (the two figures show parts of the same generated tree). Suppose \mathcal{S} is composed of the pathlet states depicted in the nodes A1 and A2 of the tree (figure 7.2). We extend the trajectory generated by the sequence \mathcal{S} with one pathlet.

For each node on the first level of the tree, we look for the history \mathcal{S} in the next couple of levels. For instance, for the node A3, the history \mathcal{S} can be found in the tree and the whole resulting sequence is depicted by A in figure 7.2. The probability for that sequence is read in the node A1.

For the node B4, the history \mathcal{S} cannot be found. The last elements of \mathcal{S} cannot be found either, so the probability associated with the sequence we are looking for is read from the node B4 itself, and we multiply that value by a uniform probability for each element of \mathcal{S} . The resulting probability is therefore:

$$\frac{P_{B4}}{|\Sigma|^2} \quad (7.2)$$

where P_{B4} is the probability read in the node B4 and $|\Sigma|$ is the number of pathlet models.

For the node C2 in figure 7.5, only the last element can be observed in the tree. Indeed the node C1 encodes the same pathlet state as the last element of \mathcal{S} which is the pathlet state seen in node A2. In that case, a uniform probability is chosen for the remaining pathlet state in the sequence \mathcal{S} and the probability of

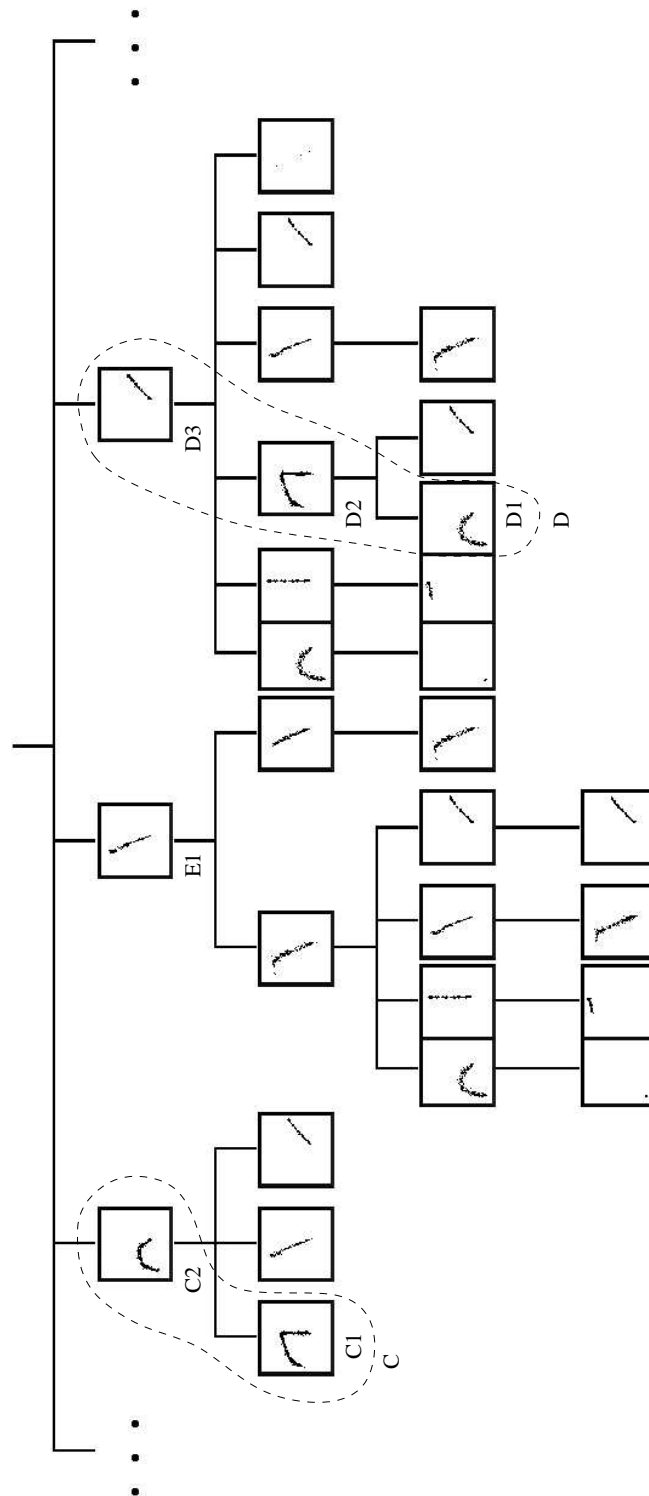


Figure 7.5: VLMM tree of pathlet states for the trajectory T2. Only a part of the tree is shown.

the sequence we are looking for is:

$$\frac{P_{C1}}{|\Sigma|} \quad (7.3)$$

where P_{C1} is the probability read in the node C1.

For node E1 in figure 7.5, the history cannot be found in the tree. The probability is:

$$\frac{P_{E1}}{|\Sigma|^2} \quad (7.4)$$

where P_{E1} is the probability read in the node E1.

Finally, for node D3 in figure 7.5, the history can be found in the tree and the probability can be read directly from the node D1.

We are computing such probabilities for all the remaining possible pathlet states. A random sampling for the computed probabilities gives us the way to extend \mathcal{S} , as described in section 6.4.

7.4 Generating a new trajectory

Once we have a sequence of pathlet states, we have a sequence of associated pathlet models and we can generate a new trajectory in the appearance parameter space.

For each pathlet model, we can generate a new pathlet. The generated trajectory in the appearance parameter space is the concatenation of the pathlet generated from each group in the order it appears in the pathlet model sequence. We have two ways of generating the pathlets given a model. We can use the spatiotemporal model of pathlets (equation 5.6 in section 5.3.2) or we can use the spatiotemporal model with linear residuals (equation 5.8 in section 5.3.3).

7.4.1 Generation without the residual model

For each group in the sequence of pathlet models, we can sample a new pathlet using equation 5.6 as described in section 5.3.2. The generated trajectory is then the concatenation of the generated pathlets. Note that for each generated pathlet, the timings used to reconstruct the pathlet are added to the timing at the end of the previous pathlet, so that the difference of time between the first point of a pathlet and the last point of the previous pathlet is the same as the difference of

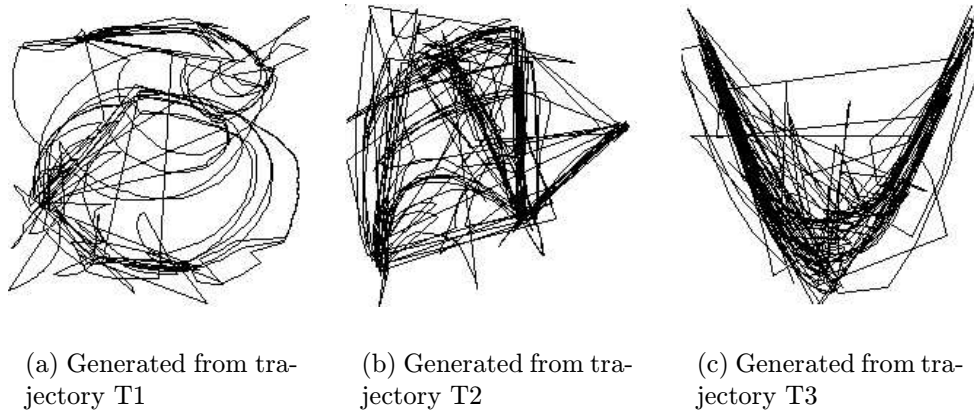


Figure 7.6: Trajectories generated with the normalised cut algorithm and the dynamic time warping algorithm.

time between two consecutive points within a pathlet. This is needed to ensure a continuous representation of the timings through the whole generated trajectory.

Figure 7.6 shows generated trajectories for sequences of pathlet models extracted from trajectories T1, T2 and T3 with the normalised cut algorithm. Figure 7.7 shows generated trajectories for sequences of pathlet models extracted from trajectories T1, T2 and T3 with the greedy algorithm.

One can notice some jumps in the generated trajectories, especially on figure 7.6. Since the points are linked by lines on figures 7.6 and 7.7, those jumps correspond to straight lines. Some of those jumps (the larger ones) correspond to problems of prediction of the VLMM. Generating a wrong pathlet state sequence generates a discontinuous trajectory since the ends of pathlets do not match.

We can see that the VLMM has more trouble modelling the sequence of pathlet models when we use the normalised cut algorithm and the dynamic time warping than when we use the greedy algorithm. This is due to the fact that the well formed pathlet groups extracted with the greedy algorithm give a more structured sequence of pathlet models, for structured trajectories. Outliers appearing in the pathlet groups extracted with the normalised cut algorithm tend to give less structured pathlet model sequences. If an outlier is modelled in a pathlet model, it generates an unlikely temporal relationship with the other pathlet models. A VLMM being trained on an unlikely sequence of pathlet models has more chance of selecting unsuitable pathlet models.

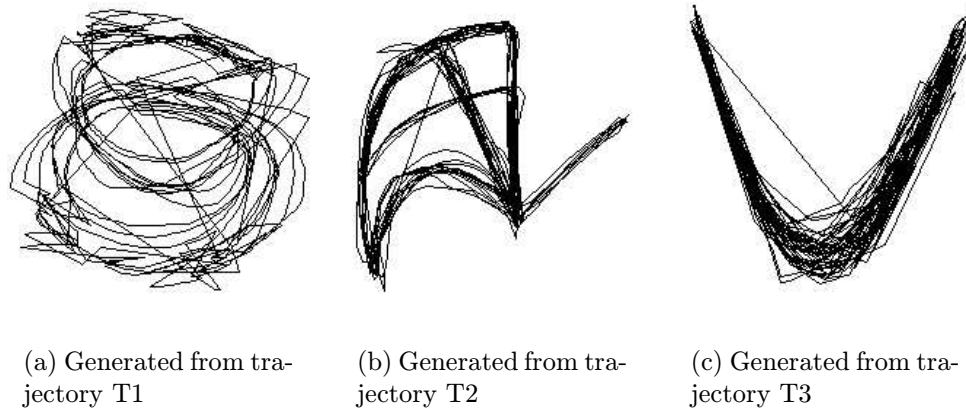


Figure 7.7: Trajectories generated with the greedy algorithm.

7.4.2 Generation with a linear residual model

If we use the spatiotemporal model without modelling the residuals, the ends of generated pathlets might not always match each other. Indeed, even if the mean pathlets of two consecutive pathlet models join properly, it does not mean that sampled pathlets from those two groups join properly. This is a problem for the corresponding generated video sequences since one can see sharp changes of the movement of the face when generating parts of the video corresponding to changes of pathlet. This creates an impression of a face jumping which is easily spotted by people.

In order to reduce this effect, we tried to add constraints on pathlet models. Unfortunately, it leads to an over-constrained problem. Figure 7.8 illustrates the problem. It shows a previously generated pathlet on the left. We want to generate the next pathlet. Dotted lines show possible pathlets we can generate from the next pathlet model. Due to the small number of modes describing the possible generated pathlets, the beginnings of those pathlets describe a subspace of the appearance parameter space. This happens when the number of modes of the pathlet model is lower than the dimension of the appearance parameter space². Matching the end of the previous generated pathlet is impossible if this point does not belong to the subspace described by the beginnings of possible generated pathlets.

²Since we aim for compact pathlet models, this case is frequent.

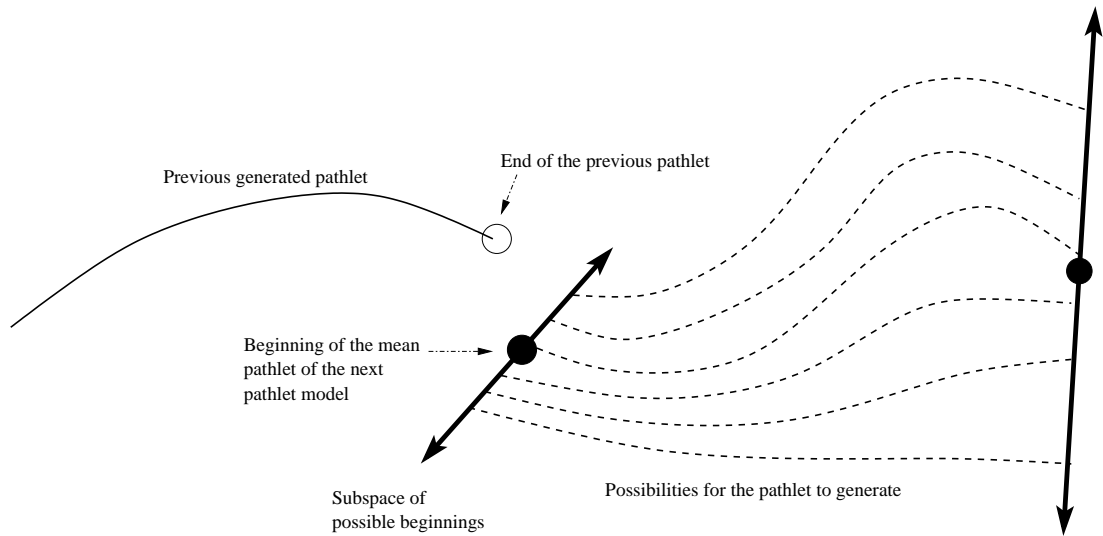


Figure 7.8: Constraining the generation of pathlets. The beginning of any generated pathlet cannot match the end of the previous generated pathlet.

Since adding constraints did not work, we use the linear residual model described in section 5.3.3. For each pathlet model, a pathlet is generated using equation 5.6 as in the previous section. The beginning of each pathlet is then forced to match the end of the previous one. The other points of the pathlet are moved according to equation 5.9 in section 5.3.3.

Figures 7.9 and 7.10 show generated trajectories using the normalised cut algorithm and the greedy algorithm respectively. The linear model for the residuals has been used to generate both figures.

7.5 Synthesising the new video sequence

Once a trajectory has been synthesised in the appearance parameter space, we can generate a new video sequence. Each point on the trajectory corresponds to one frame in the video sequence. For each point, we generate an image of the face by using the appearance parameter \mathbf{c} in equation 3.10 and use the result \mathbf{b} to reconstruct the shape (equation 3.7) and the shape-free texture (equation 3.8 page 50). The shape-free texture is then warped to the computed shape.

Examples of generated video sequences can be seen on the accompanying CD-ROM. The files:

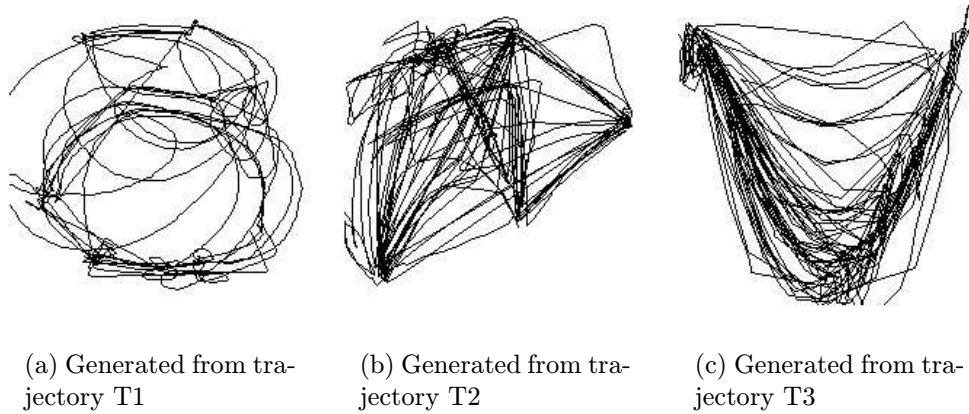


Figure 7.9: Trajectories generated with the normalised cut algorithm, the dynamic time warping algorithm and a linear residual model. Figure 7.9(a) shows a trajectory generated from the trajectory T1. Figure 7.9(b) shows a trajectory generated from the trajectory T2. Finally, figure 7.9(c) shows a trajectory generated from the trajectory T3.

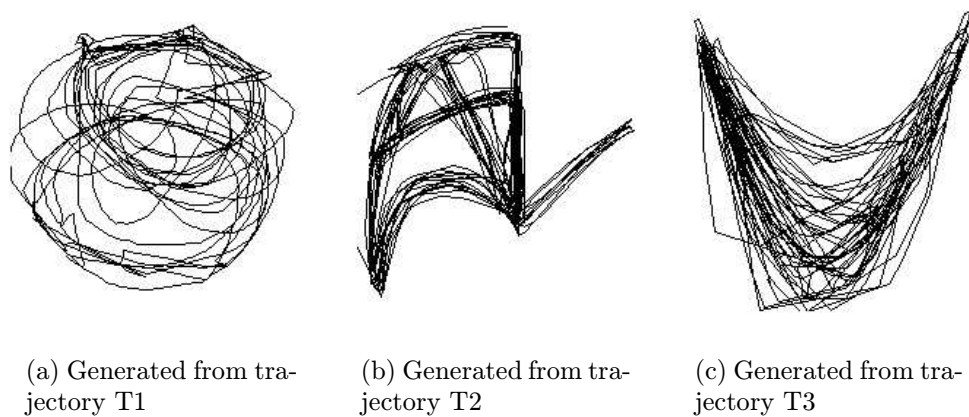


Figure 7.10: Trajectories generated with the greedy algorithm and a linear residual model. Figure 7.7(a) shows a trajectory generated from the trajectory T1. Figure 7.7(b) shows a trajectory generated from the trajectory T2. Finally, figure 7.7(c) shows a trajectory generated from the trajectory T3.

- `examples/V1/V1_wor.m1v`,
- `examples/V2/V2_wor.m1v`,
- `examples/V3/V3_wor.m1v`

have been generated without using the linear residual model while the files:

- `examples/V1/V1_wr.m1v`,
- `examples/V2/V2_wr.m1v`,
- `examples/V3/V3_wr.m1v`

have been generated using the linear residual model. All those files have been generated using the greedy algorithm.

7.6 Conclusion

In this chapter we have explained how to generate new video sequences of facial behaviour using our model. First, new sequences of pathlet states are sampled from the VLMM. Then pathlets are sampled from each corresponding pathlet model. The pathlets are concatenated together with an optional smoothing performed by using the residual component of the pathlet model. This gives us a sequence of appearance parameters. Finally, the new video sequence is generated by synthesising each frame in turn using the appearance model.

Using the linear residual model avoids jumps in generated trajectories. But, as we can see by comparing figures 7.7 and 7.10, it also seems to generate less likely trajectories. We need to be able to assess how good the generated trajectories are. In the next chapter, we develop a measure of comparison between generated trajectories in the appearance parameter space and use it to assess our model by comparing it to an alternative. A psychophysical experiment is also used to assess how people judge the generated video sequences.

Chapter 8

Qualitative and Quantitative Results

8.1 Introduction

This chapter assess our model of behaviour both qualitatively and quantitatively. Our model is compared to an alternative based on an autoregressive process. This model is described in section 8.2.

A measure of comparison between behaviour models is derived in section 8.3. It is used to compare our model to the one based on autoregressive process.

Finally, a psychophysical experiment has been set up. People were asked to compare generated videos. The experiment is described and the results are reported.

8.2 A model based on an autoregressive process

In [17], Campbell *et al.* introduce another way of generating video sequences of faces based on an existing video clip without direct reuse of the original frames. They encode frames from the original sequence in a way similar to our method. An appearance model is used to model the face through the video sequence and a trajectory is obtained in the appearance parameter space. This trajectory is then used to train a second order autoregressive process.

An autoregressive process predicts the position of a point \mathbf{y}_k in the appearance parameter space, given the two previous points \mathbf{y}_{k-1} and \mathbf{y}_{k-2} where \mathbf{k} represents

the frame number. This prediction is produced by the equation:

$$\mathbf{y}_k - \bar{\mathbf{y}} = \mathbf{A}_2 (\mathbf{y}_{k-2} - \bar{\mathbf{y}}) + \mathbf{A}_1 (\mathbf{y}_{k-1} - \bar{\mathbf{y}}) + \mathbf{B}_0 \mathbf{w}_k \quad (8.1)$$

where $\bar{\mathbf{y}}$ is the limit of the mean value of \mathbf{y}_k as k tends to infinity, \mathbf{w}_k contains white noise ($\mathbf{w}_k \sim \mathcal{N}(0, 1)$), \mathbf{A}_2 , \mathbf{A}_1 and \mathbf{B}_0 are parameter matrices. $\bar{\mathbf{y}}$, \mathbf{A}_2 , \mathbf{A}_1 and \mathbf{B}_0 can be learnt from the original data set. The learning method used in this work is due to Reynard *et al.* and is described in [79] and [8].

Given two initial points in the parameter space, a new trajectory can be generated by repeatedly applying equation 8.1. This new trajectory in the parameter space fed into an appearance model to generate a video sequence. Figure 8.1 shows an example of a video clip generated by an autoregressive process. The original video sequence V3 is shown on figure 8.2. For comparison figure 8.3 shows the generation of similar video sequences with our model.



Figure 8.1: Frames taken every 4 seconds from the video sequence generated with an autoregressive process.



Figure 8.2: Frames taken every 4 seconds from the original long video sequence.



Figure 8.3: Frames taken every 4 seconds from the video sequence generated with our model and the linear residual model.

The corresponding video sequences can be seen on the accompanying CD-ROM. The file `examples/V3/V3_orig.m1v` shows the original video sequence. The file `examples/V3/V3_arp.m1v` shows the video sequence generated with the autoregressive process. The file `examples/V3/V3_wr.m1v` shows the video sequence generated with our model if we use the linear model for the residuals. The file `examples/V3/V3_wor.m1v` shows the video sequence generated with our model when we do not use the residuals.

Another example can be seen on figures 8.4(a), 8.4(b), 8.4(c) and 8.4(d). Those figures describe the trajectory T3 (figure 8.4(a)) as well as the corresponding generated trajectory using our model with or without the linear model of residuals (figures 8.4(c), 8.4(b) respectively). Finally figure 8.4(d) shows the trajectory generated by the autoregressive process in the appearance parameter space. The corresponding video sequences can be seen on the accompanying CD-ROM. The file `examples/V1/V1_orig.m1v` shows the original video sequence. The file `examples/V1/V1_arp.m1v` shows the video sequence generated with the autoregressive process. The file `examples/V1/V1_wr.m1v` shows the video sequence generated with our model if we use the linear model for the residuals. The file `examples/V1/V1_wor.m1v` shows the video sequence generated with our model when we do not use the residuals.

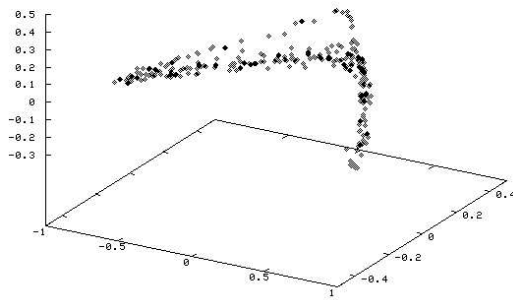
8.3 A measure of quality of generated videos

8.3.1 The comparison measure

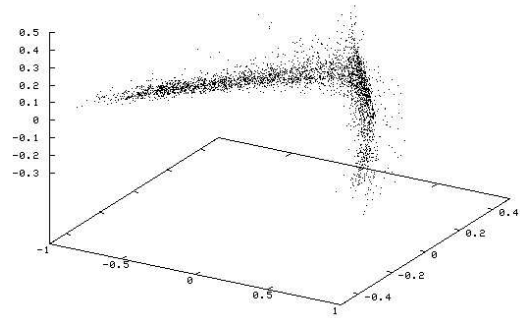
In order to compare the two models, we need some measure of behavioural similarity between two video sequences. Our approach is to compare the distribution of the generated points in the parameter space with the distribution of points extracted from the original video sequence.

We construct two dimensional histograms to approximate the distribution of points in the parameter space for each pair of dimensions. We choose two dimensional histograms instead of p dimensional histograms because we rarely have sufficient data to fill the latter, and because of the computational complexity of generating p dimensional histograms.

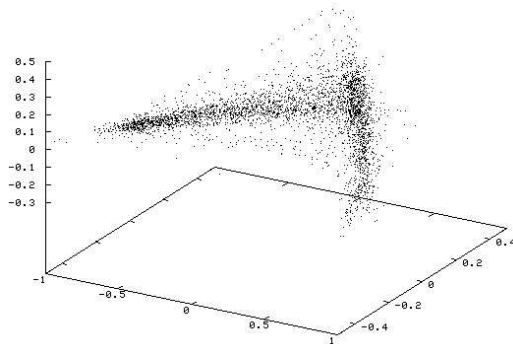
In order to compare the original and generated sequences of points using histograms, particular care has to be taken on the selection of the bin width



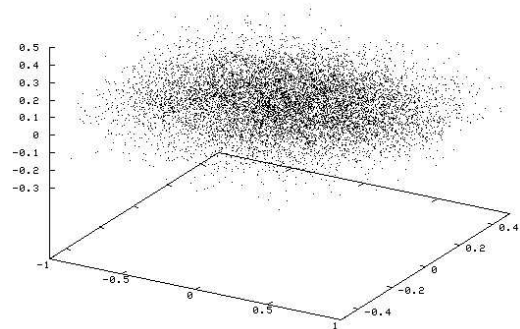
(a) Training sequence extracted from video V1.



(b) Generated sequence with our model without using the residuals.



(c) Generated sequence with our model and a linear model for the residuals.



(d) Generated sequence with an autoregressive process.

Figure 8.4: Comparison of generated trajectories. Figure 8.4(a) represents points on the trajectory that corresponds to the original video sequence. Figure 8.4(b) represents points on the trajectory generated by our model if the residuals used are set to zero. Figure 8.4(c) represents the equivalent with a linear model for residuals. Figure 8.4(d) represents points on the trajectory generated by an autoregressive process.

used to compute those histograms. Indeed, a too large bin width will smooth the original data while a too small bin width will result in an over-fitting of the data. In order to solve this problem, and for reproducibility of the comparison method, we used Scott's rule to select the bin size [97]. The bin size for each dimension is given by the formula:

$$h = 3.5\xi N^{-\frac{1}{3}} \quad (8.2)$$

where N is the total number of points in the original sequence and ξ is the standard deviation of the original data computed with respect to the selected dimension.

The two dimensional histograms of a reference and a generated set of points are then compared using a Bhattacharyya overlap $\mathcal{B}(i, j)$, where i and j are the dimensions used to compute the histograms. $\mathcal{B}(i, j)$ can be computed for each pair of dimensions (i, j) . The final similarity measure m is computed by averaging the quantities $\mathcal{B}(i, j)$ using the formula:

$$m = \frac{1}{p^2} \sum_{i=1}^p \sum_{j=1}^p \mathcal{B}(i, j) \quad (8.3)$$

where p is the dimension of the parameter space P . The standard error can also be computed to represent our confidence in the result.

m represents how close the point distributions of the generated and the reference sequences are. A value of 1 corresponds to a perfect match of the distributions. A value of 0 corresponds to two totally different distributions.

The facial behaviour models are assessed with respect to the original video sequence used to create those models. The distributions of points in the appearance parameter space should match with the original one if the model performs its task correctly.

Figures 8.5 and 8.6 show how this similarity measure can be applied to find the best generated trajectory in the appearance parameter space. Figure 8.5 shows a similarity result of 0.898 while figure 8.6 shows a similarity result of 0.965. Thus, the trajectory generated using our model has a greater similarity to the original trajectory than the trajectory generated using the autoregressive process. Our model performs better on this example. The next session describes the results obtained on the videos sequences V1, V2 and V3 introduced in section 4.3.

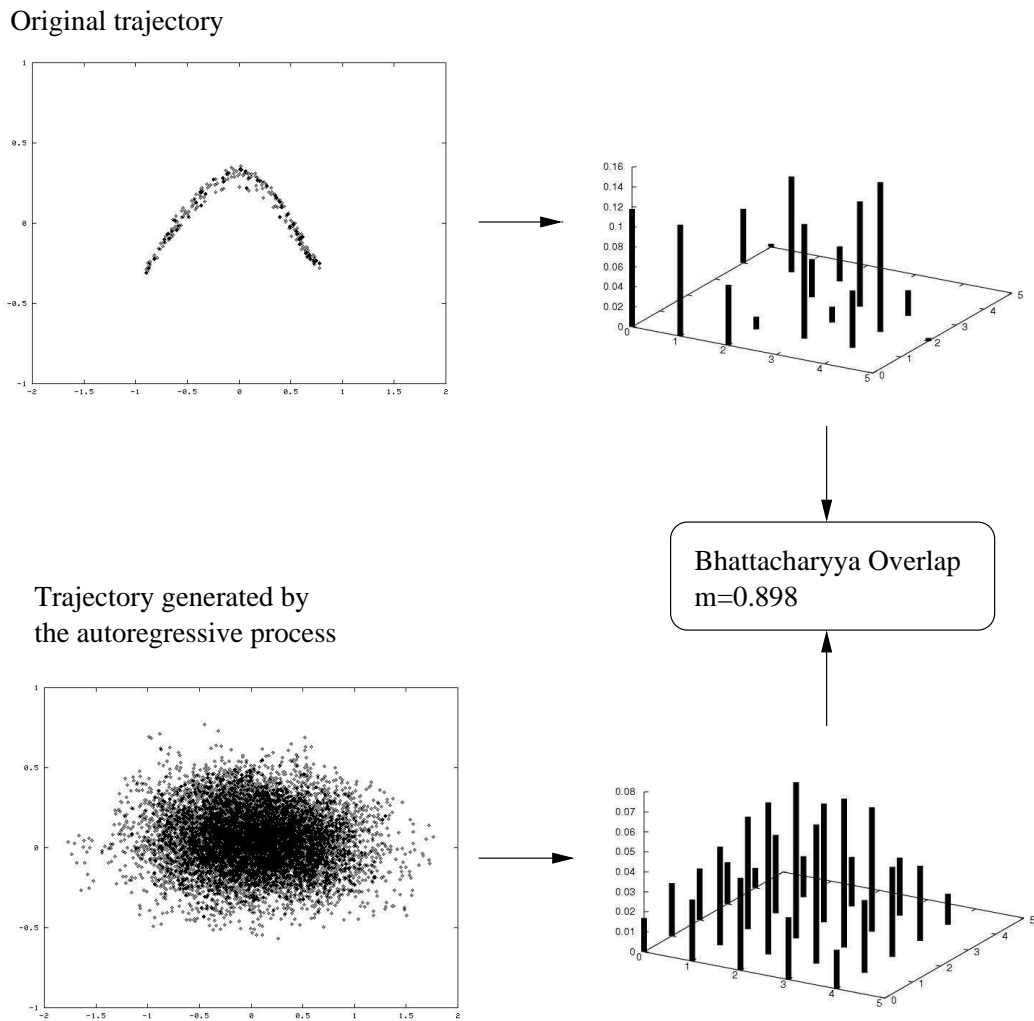


Figure 8.5: Comparison between the original trajectory and a trajectory generated by the autoregressive process. The histograms represent densities of points extracted from regularly spaced rectangular areas of the appearance parameter space. The areas are the same for both trajectories and are computed using the original trajectory.

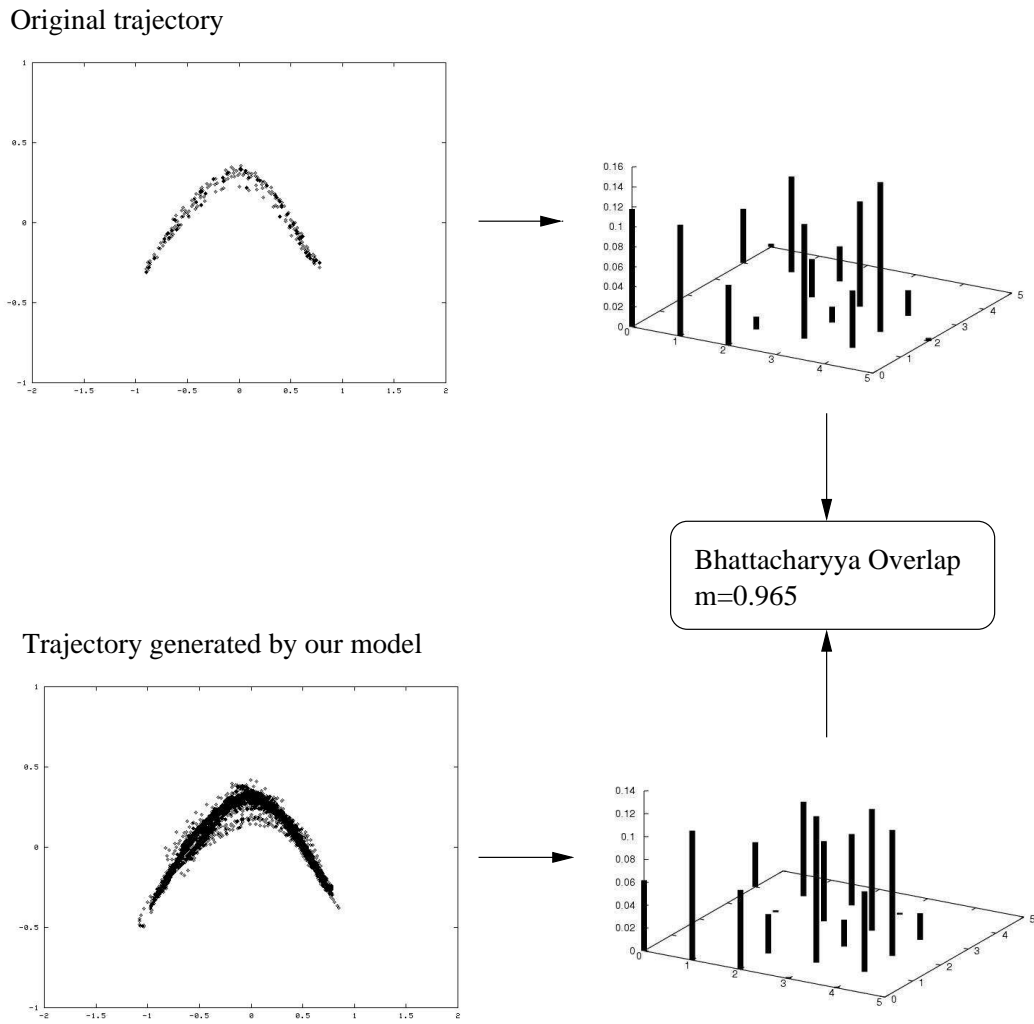


Figure 8.6: Comparison between the original trajectory and a trajectory generated by our model. The histograms represent densities of points extracted from regularly spaced rectangular areas of the appearance parameter space. The areas are the same for both trajectories and are computed using the original trajectory.

8.3.2 Results with the comparison measure

Table 8.1 presents the results of the comparisons for the videos generated from the video sequences V1, V2 and V3. Each training video is compared with each model using the measure described in the previous section for both position and speed data. The bold labels represent lines where our model is significantly better than the autoregressive process. We can see that our model outperforms the autoregressive process for structured videos, while maintaining good results for the unstructured video.

The videos corresponding to the entries in table 8.1 can be found on the accompanying CD-ROM. The videos:

- `examples/V1/V1_arp.m1v`,
- `examples/V1/V1_wr.m1v`,
- `examples/V1/V1_wor.m1v`

are generated from the video V1 using the autoregressive process, our model with and without linear residuals respectively. Similarly the videos:

- `examples/V2/V2_arp.m1v`,
- `examples/V2/V2_wr.m1v`,
- `examples/V2/V2_wor.m1v`

are generated from the video V2 and the videos:

- `examples/V3/V3_arp.m1v`,
- `examples/V3/V3_wr.m1v`,
- `examples/V3/V3_wor.m1v`

are generated using the video V3 using the same models respectively.

A visual inspection of the generated video sequences shows that our model performs best when we use the linear model for the residuals. Indeed, it produces smooth video sequences that exhibit realistic behaviours. If the residuals are not used, then the generated videos contain perceptible jumps of the face, thus giving a worse overall effect. Finally, the autoregressive process produces a video with





















Video V1: Shaking		ARP	position: $\mu = 0.898, \sigma = 0.010$
		ARP	speed: $\mu = 0.863, \sigma = 0.008$
		ARP	position: $\mu = 0.898, \sigma = 0.010$
Video V1: Shaking		WOR	position: $\mu = 0.965, \sigma = 0.003$
		WOR	speed: $\mu = 0.910, \sigma = 0.004$
		WOR	position: $\mu = 0.944, \sigma = 0.004$
Video V1: Shaking		WR	speed: $\mu = 0.896, \sigma = 0.005$
		WR	position: $\mu = 0.828, \sigma = 0.011$
		WR	speed: $\mu = 0.782, \sigma = 0.011$
Video V2: Expressions		WOR	position: $\mu = 0.829, \sigma = 0.009$
		WOR	speed: $\mu = 0.901, \sigma = 0.005$
		WOR	position: $\mu = 0.865, \sigma = 0.006$
Video V2: Expressions		WR	speed: $\mu = 0.897, \sigma = 0.005$
		WR	position: $\mu = 0.896, \sigma = 0.003$
		WR	speed: $\mu = 0.870, \sigma = 0.002$
Video V3: Dialog		WOR	position: $\mu = 0.931, \sigma = 0.001$
		WOR	speed: $\mu = 0.864, \sigma = 0.002$
		WOR	position: $\mu = 0.813, \sigma = 0.006$
Video V3: Dialog		WR	speed: $\mu = 0.903, \sigma = 0.002$
		WR	position: $\mu = 0.813, \sigma = 0.006$

Table 8.1: Comparison of the autoregressive process (ARP) with our model (WOR and WR). WOR is our model when we are not using residuals while WR is our model with the linear model for residuals.

many perceptible jitters, due to the noise present in equation 8.1, which also give a worse effect.

Jitters and jumps in the generated video sequences are not taken into account in our measure of comparison. Indeed, this measure is only a crude way to assess the quality of generated video sequence.

Since comparing generated video sequences is a complicated task and depends on the psychology of humans, we have set up a psychophysical experiment to perform the assessment.

8.4 The psychophysical experiment

8.4.1 The aim of the experiment

In order to assess how people judge generated video sequences, we set up a psychophysical experiment.

The experiment should answer the following two questions:

- are people able to distinguish between a generated video sequence and the original video sequence?
- are people able to distinguish between two video sequences generated with two different models? If so, which model produces the most realistic behaviour?

In order to answer those questions we implemented a forced choice experiment.

8.4.2 Description of the experiment

After a short description of the experiment (see figure 8.7), people are asked to answer 52 questions. Each question consists of a comparison between two video sequences. The user is asked to choose the most realistic video between the two videos displayed (see figure 8.8).

Tables 8.2 and 8.3 summarise the content of the questions. The questions are the same for each person completing the experiment. They have been selected in such a way that:

- each type of video is shown the same number of times.

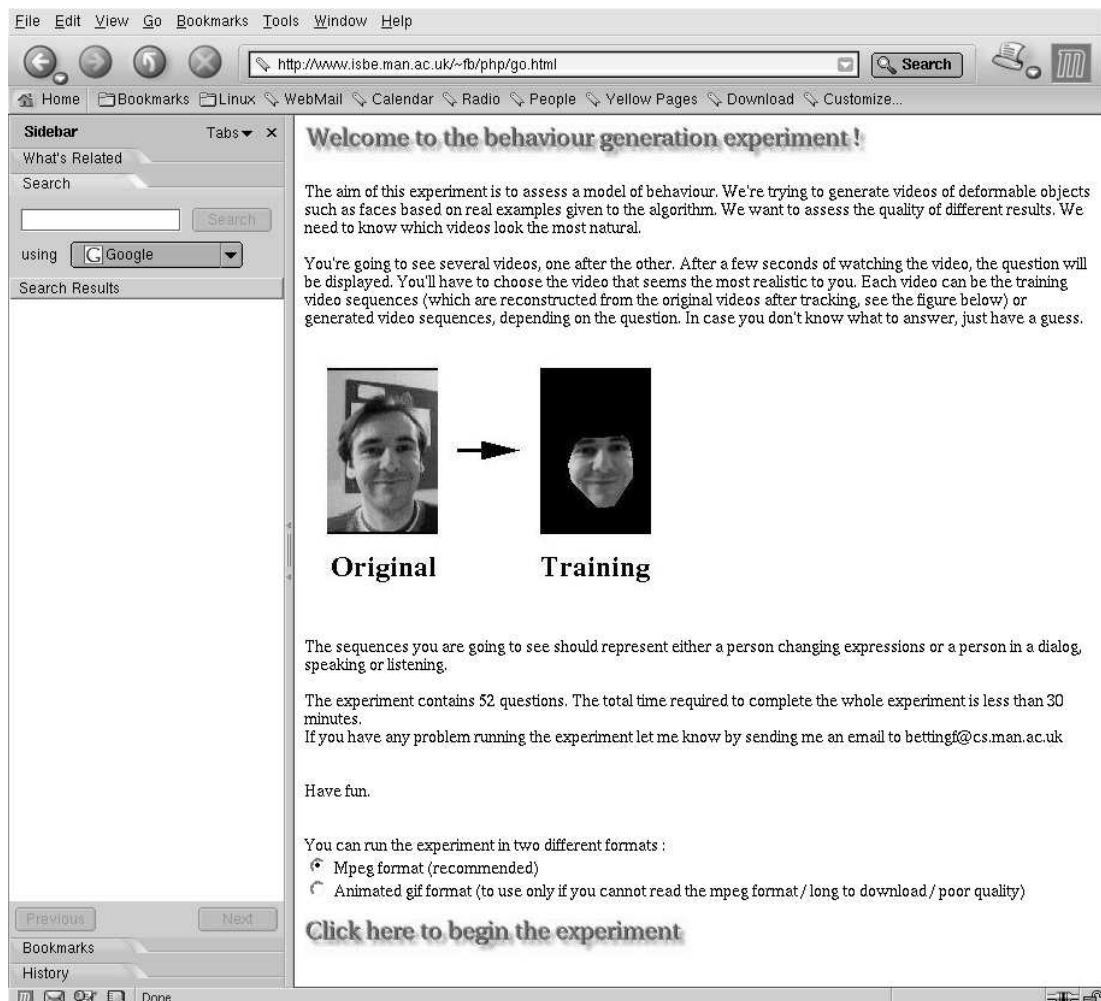


Figure 8.7: Screen capture of the description of the experiment.

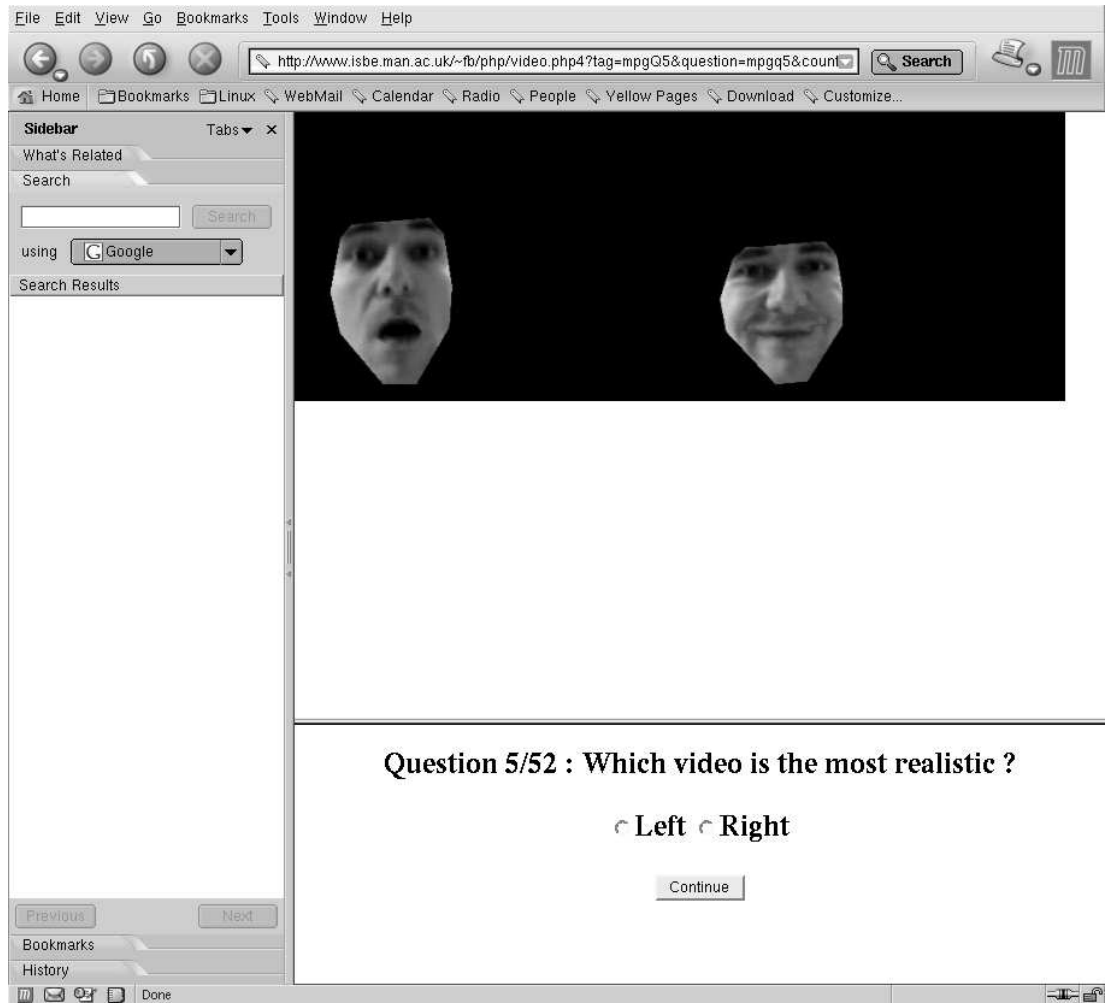


Figure 8.8: Screen capture of a question during the experiment.

- each type of video appears the same number of times on the right as on the left.
- generated videos are compared to the original video.
- videos generated with one model are compared to video generated with all the other models.
- for each pair of generated videos another pair is shown with the same models in the same order.
- for each pair of generated videos another pair is shown with the same models in a different order.

In addition to those questions, people are asked to compare four times the original video to itself. Those questions are used to verify whether people prefer the right or the left given no real difference. The questions are presented in random order to avoid potential psychological effects on sequences of similar questions.

The experiment has been done online to allow a wide range of persons to participate. Two video formats have been provided (mpeg and animated gif). The mpeg format provided the best quality and people were advised to use it in the presentation of the experiment.

The accompanying CD-ROM contains the video files used for the experiments. The files `exp/m1v/q01.m1v` to `exp/m1v/q52.m1v`, corresponding to the videos used for questions 1 to 52 respectively, are encoded using the mpeg format, while the files `exp/gif/q01.gif` to `exp/gif/q52.gif` are the corresponding videos encoded in the animated gif format. The mpeg files are each 405 frames long. This correspond to a runtime of about 16 seconds¹. The animated gif version contains half of the frames with half the frame rate². So each person completing the experiment to the end has seen about 14 minutes of video sequences (either generated or original videos). This time appeared to be globally too long for people to concentrate, which explains why only video V2 and video V3 have been included in the experiment. Video V2 represents a structured video while the video V3 is unstructured.

¹15 seconds correspond to the actual video displayed while the remaining corresponding to a countdown inserted at the beginning of the video.

²The full frame rate could not be used due to technical reasons. This also reduces the size of the video by half thus reducing the download time.

Question number	Video V2: expressions				Video V3: dialog			
	Original	WR	WOR	ARP	Original	WR	WOR	ARP
1					←	→		
2						→		←
3					↔			
4					←			→
5	←	→						
6		←	→					
7	→			←				
8		←		→				
9			←	→				
10	←	→						
11							←	→
12						→	←	
13						←		→
14					←		→	
15	↔							
16	→			←				
17					←		→	
18	←		→					
19						←	→	
20					←			→
21					→	←		
22		←	→					
23		←		→				
24			→	←				
25			→	←				
26	→		←					
27					←	→		
28						→		←
29					→	←		
30	→	←						
31					→		←	
32	→		←					
33					→		←	
34		→	←					

Table 8.2: Summary of the content of the questions in the experiment (questions 1 to 34). Arrows pointing to the left means that the video is shown on the left while arrows pointing on the right means that the video is shown on the right. An arrow pointing in both directions means that the clips used on the right as well as the one on the left are extracted from the same video.

Question number	Video V2: expressions				Video V3: dialog			
	Original	WR	WOR	ARP	Original	WR	WOR	ARP
35							→	←
36	←			→				
37						←		→
38					→			←
39		→	←					
40					↔			
41						→	←	
42					→			←
43		→		←				
44	↔							
45	←			→				
46							←	→
47		→		←				
48	←		→					
49							→	←
50	→	←						
51			←	→				
52						←	→	

Table 8.3: Summary of the content of the questions in the experiment (questions 35 to 52). Arrows pointing to the left means that the video is shown on the left while arrows pointing on the right means that the video is shown on the right. An arrow pointing in both directions means that the clips used on the right as well as the one on the left are extracted from the same video.

Finally, the time taken to answer each question has been recorded although no indication about timings were given to the volunteers.

8.4.3 Statistics

Since only two choices are available for each question, the statistics of the set of answers describe a binomial distribution. If we are comparing videos generated from a model A with videos generated from a model B, the probability $P(r; p, n)$ of choosing the videos generated by the model A r times out of n trials is given by the equation:

$$P(r; p, n) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \quad (8.4)$$

where p is the hypothetical probability of choosing A for each trial.

We want to answer the question: "can people distinguish between two different models by looking at generated videos from the two models"?

We consider the null hypothesis that the videos are indistinguishable. In this case, $p = 0.5$ and the distribution of r is given by:

$$P(r; p, n) = \frac{n!}{2^n r!(n-r)!} \quad (8.5)$$

We reject the null hypothesis if:

$$\sum_{i=\min(r, n-r)+1}^{\max(r, n-r)-1} \frac{n!}{2^n i!(n-i)!} > 0.95 \quad (8.6)$$

i.e. there is a 95% chance that we would have obtained a smaller difference if the videos had been indistinguishable.

Similar equations can be used to check whether people prefer the left or the right when choosing videos from two choices extracted from the original sequence.

8.4.4 Results of the psychophysical experiment

Of the 112 people who volunteered, only 43 completed all the questions. Technical problems and the length of the experiment deterred the rest. In the following we present an analysis based on the 43 complete surveys.

Figure 8.9 shows a graph of the time used to answer the questions. We can see three different types of response on this graph:

- answers before the end of the video.
- answers at the end of the video.
- answers after playing the video a second time.

The three different types of response are separated by dotted lines on the graph, corresponding to the end of the video either played once or twice.

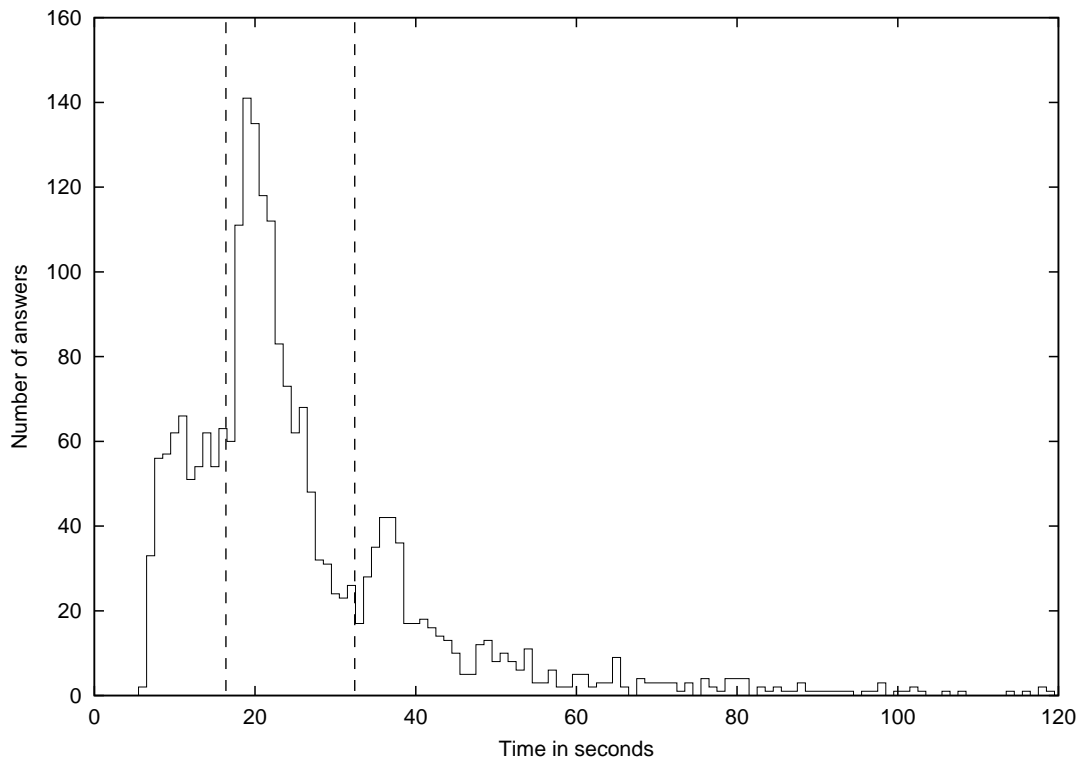


Figure 8.9: Time taken to answer the questions in the experiment. One dotted line corresponds to the time taken by the video. The other one corresponds to twice that time.

Since most answers belong to the two last categories, it appears that the experiment has been done carefully by the persons who took the time to complete it.

When comparing two videos clips both extracted from the original sequence, people tend to choose the video on the right rather than the video on the left. Indeed, 72 videos have been selected on the left while 99 videos have been selected on the right. This gives a significant bias towards the right (the probability being 0.9535).

	Original	WR	WOR	ARP
Original		121 (35%)	125 (36%)	86 (25%)
WR	223 (65%)		168 (49%) *	124 (36%)
WOR	219 (64%)	176 (51%) *		158 (46%) *
ARP	258 (75%)	220 (64%)	186 (54%) *	

Table 8.4: Psychophysical experiment answers' summary. WR (respectively WOR) is our model with (respectively without) a linear residual model. ARP is the autoregressive process. The results are reported for the whole set of questions. Each cell represents the number of answers selecting the model in the column as being more realistic than the model in the row. A star (\star) has been added to non-significant results.

The bias on the selected side is also present if we use all the questions. This bias is the main reason why we have only kept answers from the people completing the whole experiment. The side on which the generated videos appears should be balanced to compensate for the bias introduced in the case of random choice. If the experiment is completed, each pair of models is displayed an equal number of times on each side.

The results of the experiments can be seen on table 8.4. The table shows the number of answers given for each pair of models, for all the videos and all the volunteers. It shows that for each model, people were able to distinguish between the original video sequence and the generated ones. There is still room for improving any of the models. This result is in agreement with the results of Hack who did a similar experiment [41] and found that no model so far tested in his experiment can confound the volunteers.

The results of our experiment also show that our model performs better than the autoregressive process if the linear model of residuals is used to smooth the output. We cannot conclude anything when comparing the other possible pairs of models since the results are not significant (cases annotated by a \star in table 8.4).

Table 8.5 shows the results for video V2 (expressions). It shows similar results. We can also conclude that, for video V2, our model is also significantly better than the autoregressive process even when we do not use the linear model of residuals.

Table 8.6 shows the results for video V3 (dialog). This time, only the original videos have been successfully spotted by the volunteers. There are no significant

	Original	WR	WOR	ARP
Original		58 (34%)	66 (38%)	25 (15%)
WR	114 (66%)		91 (53%) *	44 (26%)
WOR	106 (62%)	81 (47%) *		69 (40%)
ARP	147 (85%)	128 (75%)	103 (60%)	

Table 8.5: Psychophysical experiment answers' summary for video V2. WR (respectively WOR) is our model with (respectively without) a linear residual model. ARP is the autoregressive process. Each cell represents the number of answers selecting the model in the column as being more realistic than the model in the row. A star (\star) has been added to non-significant results.

	Original	WR	WOR	ARP
Original		63 (37%)	59 (34%)	61 (35%)
WR	109 (64%)		77 (45%) *	80 (47%) *
WOR	113 (66%)	95 (55%) *		89 (52%) *
ARP	111 (65%)	92 (53%) *	83 (48%) *	

Table 8.6: Psychophysical experiment answers' summary for video V3. WR (respectively WOR) is our model with (respectively without) a linear residual model. ARP is the autoregressive process. Each cell represents the number of answers selecting the model in the column as being more realistic than the model in the row. A star (\star) has been added to non-significant results.

differences between any other pairs of models. However, the results suggest that our model does not perform worse than the autoregressive process when we use the linear model of residuals.

8.5 Conclusion

In this chapter, we have developed a measure of comparison of facial behaviour models to assess quantitatively our model of behaviour. This measure is based on a comparison of trajectories in the appearance parameter space. A Bhattacharyya overlap is used to compare histograms of densities of points in the appearance parameter space. This measure shows that our model outperforms an alternative on structured videos sequences while maintaining a good performance on unstructured video sequences.

A psychophysical experiment was performed to assess qualitatively our model

of facial behaviour. It showed that our model is significantly better than the model based on an autoregressive process when it models structured video sequences. We do not have any significant results for unstructured video sequences. However, the figures suggest that our model does not perform worse than the autoregressive process in that case.

Agreements between the two methods of evaluation suggest that our quantitative measure of similarity between two facial behaviour models follows the natural choice of people.

Furthermore, people seem to have trouble to distinguish between our model with and without the linear model of residuals. However, in structured video sequences, the experiment shows that our model is better than the autoregressive process even if we do not use the residual model.

Chapter 9

Conclusions

This thesis has described a practical approach to modelling the behaviour of faces by learning from video sequences. The assumption is that people repeat fragments of behaviour, and that it is possible to learn both the general form of the fragments and the order in which they tend to happen.

The fragments are represented using models of pathlets in the parameter space of a facial appearance model. A VLMM is trained to generate the order of transitions between pathlets over both short and long timescales.

9.1 Summary of contributions

Though similar approaches have been used to model human motion [34], as far as we are aware, this is the first time the approach has been applied to facial behaviour.

As well as this, the contributions include :

- 1) **A heuristic for breaking a trajectory into pathlets.** An efficient heuristic has been developed to break a trajectory into pathlets. By observing that similar pathlets should have similar beginning and similar ends, points of high density in the parameter space are used to split the trajectory.
- 2) **A compact pathlet model.** By modelling the pathlet groups using a multivariate Gaussian, it has been shown how timing information can be included in the model, thus allowing modelling groups of pathlets of different speeds.

- 3) **An effective pathlet grouping algorithm.** Two pathlet grouping algorithms have been presented. The most effective one is based on a greedy merging strategy. At each step, the variance of the pathlet groups are kept as low as possible, leading to compact models.
- 4) **Improvements to the performance of the VLMM.** We have shown how different ways of estimating the probabilities of samples can affect the performance of the VLMM. Improvement in performance has been achieved by replacing the Kullback-Leibler measure by the Bhattacharyya measure for comparing the probability distributions.
- 5) **A quantitative similarity measure for the behaviour model.** A quantitative similarity measure has been developed to assess the quality of different models of behaviour. Although the measure is crude, it has been shown that it gives similar results than the intuitive comparisons done by people.
- 6) **A demonstration of improved performance.** A psychophysical experiment has been set up to assess the performance of our model. It shows an improvement of the performance for structured behaviours compared to an alternative technique, while performance for unstructured behaviours remains similar.

9.2 Future work and extensions

Creating a convincing synthetic talking head is a very ambitious project, well beyond the scope of a single PhD. This thesis demonstrates some progress towards such a goal, but much further work remains to be done.

The psychophysical experiment suggests that, although our method generally outperforms a relatively straightforward alternative such as the autoregressive process, it is not yet good enough to be indistinguishable from the original video. There is clearly further work required to improve performance. We would also like to compare our approach with recently described methods such as that of Hack and Taylor [40], who model facial behaviour using various modifications of the hidden Markov models.

One of the major drawbacks of our approach, is the number of thresholds that have to be set by the user. The threshold values usually depend on the

data that we want to process. For instance, as explained in section 5.2.2, if the parameter d is too small or too large for the dataset, we take the risk of not selecting enough nodes to split the trajectory in the appearance parameter space. We need a method for selecting the parameters automatically.

So far, our model has only been used to synthesise faces. In theory a good model should be able to help in tracking by predicting likely facial movements. Adapting our model for this task remains to be done.

Clearly, a convincing talking head would have to integrate speech synthesis, as well as to be able to both observe a human user and in some way understand them so that it could react to them appropriately. Merging of results in speech synthesis, speech recognition, behaviour modelling and eventually complex artificial intelligence methods could be investigated to create a talking head displaying emotions believably.

9.3 Final conclusions

This thesis described an approach for modelling the behaviour of deformable objects from training video sequences. The resulting model is able to explicitly explain behaviours and more importantly synthesise new videos of behaviour.

The method has been applied on the movement of the face. Though the results do not yet always convince every human observer, the method is able to generate believable facial behaviour and could form the basis for an effective talking head.

Bibliography

- [1] F. J. Aherne, N. A. Thacker, and P. I. Rockett. The Bhattacharyya metric as an absolute similarity measure for frequency coded data. *Kybernetika*, 32(4):001–007, 1997.
- [2] Okan Arikan, David A. Forsyth, and James F. O’Brien. Motion synthesis from annotations. In *Proceedings of SIGGRAPH*, 2002.
- [3] G. Bailly. Audiovisual speech synthesis. In *ETRW on Speech Synthesis*, Perthshire, Scotland, 2001.
- [4] A. M. Baumberg and D. C. Hogg. An efficient method for contour tracking using active shape models. Technical Report 94.11, School of Computer Studies, University of Leeds, April 1994.
- [5] F. Bettinger and T. F. Cootes. A model of facial behaviour. In *Proceedings of the 6th International Conference on Automatic Face and Gesture Recognition*, pages 123–128, Seoul, Korea, May 2004.
- [6] F. Bettinger, T. F. Cootes, and C. J. Taylor. Modelling facial behaviours. In Paul L. Rosin and David Marshall, editors, *British Machine Vision Conference*, pages 797–806, Cardiff, UK, September 2002.
- [7] M. Black and A. Jepson. Recognizing temporal trajectories using the condensation algorithm. In *Proceedings of the 3rd IEEE International Conference on Automatic Face and Gesture Recognition*, April 1998.
- [8] Andrew Blake and Michael Isard. *Active Contours*. Springer, 1998.
- [9] A. Bobick and J. Davis. An appearance-based representation of action. In *13th International Conference on Pattern Recognition*, Vienna, Austria, August 1996.

- [10] A. Bobick and J. Davis. The representation and recognition of action using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, March 2001.
- [11] Richard Bowden. Learning statistical models of human motion. In *IEEE Workshop on Human Modelling, Analysis and Synthesis, CVPR*, Hilton Head Island, July 2000.
- [12] M. Brand. Coupled hidden Markov models for modeling interacting processes. Technical Report TR-405, MIT Media lab Perceptual Computing / Learning and Common Sense, Cambridge, November 1996.
- [13] M. Brand and V. Kettner. Discovery and segmentation of activities in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):844–851, August 2000.
- [14] M. Brand, N. Oliver, and A. Pentland. Coupled hidden Markov models for complex action recognition. In *IEEE CVPR97*, 1997.
- [15] Christoph Bregler. Learning and recognizing human dynamics in video sequences. In *IEEE Conference on Computer Vision and Pattern Recognition*, June 1997.
- [16] Christoph Bregler, Michele Covell, and Malcom Slaney. Video rewrite: Driving visual speech with audio. In *Proceedings of SIGGRAPH*, 1997.
- [17] Neill W. Campbell, Colin Dalton, David Gibson, and Barry Thomas. Practical generation of video textures using the auto-regressive process. In Paul L. Rosin and David Marshall, editors, *British Machine Vision Conference*, pages 434–443, September 2002.
- [18] Robert T. Collins and Yanxi Liu. On-line selection of discriminative tracking features. In *Proceedings of the 9th International Conference on Computer Vision (ICCV-03)*, pages 346–352, Nice, France, 2003.
- [19] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *Seventh International Conference on Computer Vision*, pages 1197–1203, 1999.
- [20] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

- [21] T. Cootes and C. Taylor. Statistical models of appearance for computer vision. Technical report, University of Manchester, 1999.
- [22] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In H. Burkhardt & B. Neumann Ed's, editor, *European Conference on Computer Vision*, volume 2, pages 484–498. Springer, 1998.
- [23] T. F. Cootes and C. J. Taylor. A mixture model for representing shape variation. In BMVA Press, editor, *Proceedings of British Machine Vision Conference*, pages 110–119, 1997.
- [24] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, 1997.
- [25] Darren Cosker, David Marshall, Paul Rosin, and Yulia Hicks. Video realistic talking heads using hierarchical non-linear speech-appearance models. In *Proceedings of Mirage*, France, March 2003.
- [26] J. Davis. Hierarchical motion history images for recognizing human motion. In *IEEE Workshop on Detection and Recognition of Events in Video*, pages 39–46, Vancouver, Canada, July 2001.
- [27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm (with discussion). *Journal of the Royal Statistical Society series B*, 39:1–38, 1977.
- [28] Vincent E. Devin and David C. Hogg. Reactive memories: An interactive talking-head. In *British Machine Vision Conference*, pages 603–612, September 2001.
- [29] Ian L. Dryden and Kanti V. Mardia. *Statistical Shape Analysis*. John Wiley & Sons, 1998.
- [30] G. Edwards, C. Taylor, and T. Cootes. Learning to identify and track faces in image sequences. In *8th British Machine Vision Conference*, pages 130–139, Colchester, UK, 1997.
- [31] P. Eisert, S. Chaudhuri, and B. Girod. Speech driven synthesis of talking head sequences. In *3D Image Analysis and Synthesis*, pages 51–56, Erlangen, November 1997.

- [32] D. J. Fleet, M. J. Black, Y. Yacoob, and A. D. Jepson. Design and use of linear models for image motion analysis. *Int. Journal of Computer Vision*, 36(3):171–193, 2000.
- [33] Aphrodite Galata, Neil Johnson, and David Hogg. Learning structured behaviour models using variable length markov models. Technical Report 1999.10, University of Leeds, May 1999.
- [34] Aphrodite Galata, Neil Johnson, and David Hogg. Learning variable-length Markov models of behavior. *Computer Vision and Image Understanding: CVIU*, 81(3):398–413, March 2001.
- [35] B. Le Goff and C. Benoit. A text-to-audiovisual-speech synthesizer for french. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Philadelphia, USA, October 1996.
- [36] Gene H. Golub and Charles F. Van Loan. *Matrix computations*. John Hopkins Press, 1989.
- [37] Hans Peter Graf and Eric Cosatto. Sample-based synthesis of talking heads. In *ICCV-RATFG-RTS*, pages 3–7, 2001.
- [38] Lisa Gralewski, Neill Campbell, Barry Thomas, Colin Dalton, and David Gibson. Statistical synthesis of facial expressions for the portrayal of emotion. In *Proceedings of the International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE 2004)*, Singapore, June 2004.
- [39] I. Guyon and F. Pereira. Design of a linguistic postprocessor using variable memory length markov models. In *3rd International Conference on Document Analysis and Recognition*, pages 454–457, 1995.
- [40] C. A. Hack and C. J. Taylor. Modelling 'talking head' behaviour. In *British Machine Vision Conference (BMVC03)*, 2003.
- [41] Craig Hack. *Modelling 'Talking Head' Behaviour*. PhD thesis, University of Manchester, 2004.

- [42] Pengyu Hong, Matthew Turk, and Thomas S. Huang. Gesture modeling and recognition using finite state machines. In *Proceedings of the Fourth International Conference on Automatic Face and Gesture Recognition*, pages 410–415, March 2000.
- [43] Pengyu Hong, Zhen Wen, and Thomas S. Huang. An integrated framework for face modeling, facial motion analysis and synthesis. In *Proceedings of the 9th ACM International Conference on Multimedia*, pages 495–498, Ottawa, Ontario, Canada, September 2001.
- [44] Ying Huang, Xiaoqing Ding, Baining Guo, and Heung-Yeung Shum. Real-time face synthesis driven by voice. In *Proceedings of CAD/Graphics*, 2001.
- [45] Peter Tino and Georg Dorffner. Building predictive models from fractal representations of symbolic sequences. In *Advances in Neural Information Processing Systems*, 2000.
- [46] E. T. Jaynes. Monkeys, kangaroos and n. In J. H. Justice, editor, *Maximum Entropy and Bayesian Methods in Applied Statistics*, page 26. Cambridge University Press, 1986.
- [47] T. Jebara and A. Pentland. Parametrized structure from motion for 3d adaptive feedback tracking of faces. In *Proceedings of Computer Vision and Pattern Recognition*, 1997.
- [48] T. Jebara and A. Pentland. Action reaction learning: Automatic visual analysis and synthesis of interactive behaviour. *Lecture Notes in Computer Science*, 1542:273–292, 1999.
- [49] T. Jebara and A. Pentland. The generalized CEM algorithm. *Advances in Neural Information Processing Systems*, 12, 1999.
- [50] F. Jelinek. *Statistical Methods for Speech Recognition*. MIT Press, 1997.
- [51] N. Johnson, A. Galata, and D. Hogg. The acquisition and use of interaction behaviour models. In IEEE Computer Society Press., editor, *CVPR*, pages 866–871, 1998.
- [52] N. Johnson, A. Galata, and D. Hogg. The acquisition and use of interaction behaviour models. In IEEE Computer Society Press., editor, *Proc. IEEE*

- Computer Society Conference on Computer Vision and Pattern Recognition - CVPR'98*, pages 866–871, 1998.
- [53] F. Jurie and M. Dhome. Real time robust template matching. In *British Machine Vision Conference*, pages 123–132, September 2002.
- [54] V. Kettner and M. Brand. Minimum-entropy models of scene activity. In *Proceedings of the IEEE Computer Science Conference on Computer Vision and Pattern Recognition (CVPR-99)*, pages 281–286, Los Alamitos, June 23–25 1999. IEEE.
- [55] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of SIGGRAPH*, pages 473–482, San Antonio, Texas, 2002.
- [56] Takeshi Kurata, Takashi Okuma, Masakatsu Kourogi, and Katsuhiko Sakaue. The hand mouse: Gmm hand-color classification and mean shift tracking. In *Proceedings of the IEEE ICCV Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems (RATFG-RTS'01)*, pages 119–124, July 2001.
- [57] A. Lanitis, C. J. Taylor, and T. F. Cootes. An automatic face identification system using flexible appearance models. In BMVA Press, editor, *Proceedings of the British Machine Vision Conference*, pages 66–75, 1994.
- [58] Yan Li, Tianshu Wang, and Heung-Yeung Shum. Motion texture: A two-level statistical model for character motion synthesis. In *Proceedings of SIGGRAPH*, 2002.
- [59] D. Magee and R. Boyle. Feature tracking in real world scenes (or how to track a cow). In *IEE Colloquium on Motion Analysis and Tracking*, May 1999.
- [60] D. R. Magee. *Machine Vision Techniques for the Evaluation of Animal Behaviour*. PhD thesis, The University of Leeds, School of Computing, October 2000.
- [61] D. R. Magee and R. D. Boyle. Detecting lameness in livestock using 're-sampling condensation' and 'multi-stream cyclic hidden Markov models'. In *BMVC2000 11th British Machine Vision Conference Volume 1*, Bristol, September 2000.

- [62] D. R. Magee and R. D. Boyle. Spatio-temporal modeling in the farmyard domain. In *Proceedings IAPR International Workshop on Articulated Motion and Deformable Objects*, pages 83–95, 2000.
- [63] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley series in probability and statistics. John Wiley & Sons, 1999.
- [64] Dimitrios Makris and Tim Ellis. Spatial and probabilistic modelling of pedestrian behaviour. In Paul L. Rosin and David Marshall, editors, *British Machine Vision Conference*, pages 557–566, Cardiff, UK, September 2002.
- [65] Duncan Marsh. *Applied Geometry for Computer Graphics and CAD*. Springer-Verlag, 1999.
- [66] Richard J. Martin. A metric for ARMA processes. *IEEE Transactions on Signal Processing*, 48(4):1164–1170, April 2000.
- [67] D. W. Massaro, J. Beskow, M. M. Cohen, C. L. Fry, and T. Rodriguez. Picture my voice: Audio to visual speech synthesis using artificial neural networks. In D. W. Massaro, editor, *Proceedings of AVSP'99, International Conference on Auditory-Visual Speech Processing*, pages 133–138, Santa Cruz, CA., August 1999.
- [68] Iain Matthews and Simon Baker. Active appearance models revisited. Technical Report CMU-RI-TR-03-02, The Robotics Institute, Carnegie Mellon University, 2002.
- [69] Iain Matthews, Takahiro Ishikawa, and Simon Baker. The template update problem. In *Proceedings of the British Machine Vision Conference*, September 2003.
- [70] Tom M. Mitchell. *Machine Learning*. Computer Science. McGraw-Hill, 1997.
- [71] Jeffrey Ng and Shaogang Gong. Learning intrinsic video content using Levenshtein distance in graph partitioning. *Lecture Notes in Computer Science*, 2353:670–684, 2002.

- [72] Jun-Yong Noh and Ulrich Neumann. Talking faces. In *IEEE International Conference on Multimedia and Expo*, volume 2, pages 627–630, 2000.
- [73] S. Ouni, D. W. Massaro, M. M. Cohen, K. Young, and A. Jesse. Internationalization of a talking head. In *15th International Congress of Phonetic Sciences (ICPhS'03)*, Barcelona, Spain, August 2003.
- [74] David Oziem, Lisa Gralewski, Neill Campbell, David Gibson, and Barry Thomas. Synthesising facial emotions. In *Proceedings of Theory and Practice of Computer Graphics (TPCG'04)*, pages 120–127, Bournemouth, United Kingdom, June 2004.
- [75] M. Pantic and L. J. M. Rothkrantz. Automatic analysis of facial expressions: The state of the art. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1424–1445, December 2000.
- [76] L. Rabiner and B. Juang. *Fundamentals of speech recognition*. Prentice Hall, 1993.
- [77] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77:257–286, 1989.
- [78] E. Raudsepp. Body language speaks louder than words. *Machine Design*, 65(19):85–89, September 1993.
- [79] David Reynard, Andrew Wildenberg, Andrew Blake, and John A. Marchant. Learning dynamics of complex motions from image sequences. In *ECCV (1)*, pages 357–368, 1996.
- [80] Eric Sven Ristad. A natural law of succession. Technical Report TR-495-95, Princeton University, Computer Science Department, July 1995.
- [81] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia. In Jack D. Cowan, Gerald Tesauro, and Joshua Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 176–183. Morgan Kaufmann Publishers, Inc., 1994.
- [82] Dana Ron, Yoram Singer, and Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. *Machine Learning*, 25:117–149, 1996.

- [83] S. T. Roweis and L. K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *SCIENCE*, 290:2323–2326, 2000.
- [84] Payam Saisan, Gianfranco Doretto, Ying Nian Wu, and Stefano Soatto. Dynamic texture recognition. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, pages 58–63, 2001.
- [85] Lawrence K. Saul and Michael I. Jordan. Boltzmann chains and Hidden Markov Models. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 435–442. The MIT Press, 1995.
- [86] Arno Schödl, Richard Szeliski, David H. Salesin, and Irfan Essa. Video textures. In Kurt Akeley, editor, *Siggraph 2000, Computer Graphics Proceedings*, pages 489–498. ACM Press / ACM SIGGRAPH / Addison Wesley Longman, 2000.
- [87] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Conf. Computer Vision and Pattern Recognition*, June 1997.
- [88] Jianbo Shi and Jitendra Malik. Self inducing relational distance and its application to image segmentation. In *Proceedings of the 5th European Conference on Computer Vision (ECCV'98)*, pages 528–543, Freiburg, Germany, June 1998.
- [89] Jianbo Shi and Jitendra Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, August 2000.
- [90] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.
- [91] M. B. Stegmann. Object tracking using active appearance models. In Søren I. Olsen, editor, *10th Danish Conference on Pattern Recognition and Image Analysis*, volume 1, pages 54–60, Copenhagen, Denmark, July 2001. DIKU.
- [92] N. Sumpter and A. Bulpitt. Learning spatio-temporal patterns for predicting object behaviour. In *Proc. British Machine Vision Conference*, pages 649–658, 1998.

- [93] Barry J. Theobald, Andrew Bangham, Iain Matthews, J. R. W. Glauert, and G. C. Cawley. 2.5d visual speech synthesis using appearance models. In Richard Harvey and Andrew Bangham, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, 2003.
- [94] Barry J. Theobald, J. Andrew Bangham, Iain Matthews, and Gavin C. Cawley. Visual speech synthesis using statistical models of shape and appearance. In *Proceedings of Auditory-Visual Speech Processing*, 2001.
- [95] M. Walter, A. Psarrou, and S. Gong. Auto clustering for unsupervised learning of atomic gesture components using minimum description length. In *ICCV-RATFG-RTS*, pages 157–162, 2001.
- [96] Michael Walter, Alexandra Psarrou, and Shaogang Gong. Data driven gesture model acquisition using minimum description length. In *British Machine Vision Conference*, pages 673–683, September 2001.
- [97] M. P. Wand. Data-based choice of histogram bin width. *The American Statistician*, 51(1):59, February 1997.
- [98] Tianshu Wang, Yan Li, Ying-Qing Xu, and Heung-Yeung Shum. Learning kernel-based hmms for dynamic sequence synthesis. *Graphical Models*, 65(4):206 – 221, July 2003.
- [99] J. Yang, R. Stiefelhagen, U. Meier, and A. Waibel. Visual tracking for multimodal human computer interaction. In *Proceedings of ACM CHI 98 Conference on Human Factors in Computing Systems*, volume 1 of *About Faces*, pages 140–147, 1998.
- [100] Bo Yu. Recognition of freehand sketches using mean shift. In *Proceedings of the 2003 International Conference on Intelligent User Interfaces*, pages 204–210, Miami, Florida , USA, January 2003.

Appendix A

Dynamic time warping

The dynamic time warping algorithm compares two sequences of points. A cost function is defined between each pairs of points. In our case, the cost function f is the square of the euclidian distance between the points. The total cost of comparing two sequences is the sum of the cost for each corresponding point defined by the warping of time. We want to find the best warping that gives the lowest cost for comparing two sequences of points, that is two pathlets.

Since dynamic time warping assumes continuity and monotonicity of the time, the minimum cost can be computed recursively by computing the minimum cost between two sub-sequences starting at the origin, then stepping forward in time¹, either on one sequence or the other or both and computing the minimum cost between the new sequences by adding the cost of the newly matched points. Figure A.1 summarises the dynamic time warping algorithm.

```
For each line  $i$  of the grid from 1 to  $I$ 
  For each row  $j$  of the grid from 1 to  $J$ 
    compute  $D(i, j) = \min_{\substack{i-1 \leq k \leq i \\ j-1 \leq l \leq j \\ (k, l) \neq (i, j)}} \{D(k, l)\} + cost(i, j)$ 
return  $D(I, J)$ 
```

Figure A.1: The dynamic time warping matching algorithm.

It is possible to retrieve the warp of time used to compute the minimum cost

¹We are only stepping forward one step due to the monotonicity condition.

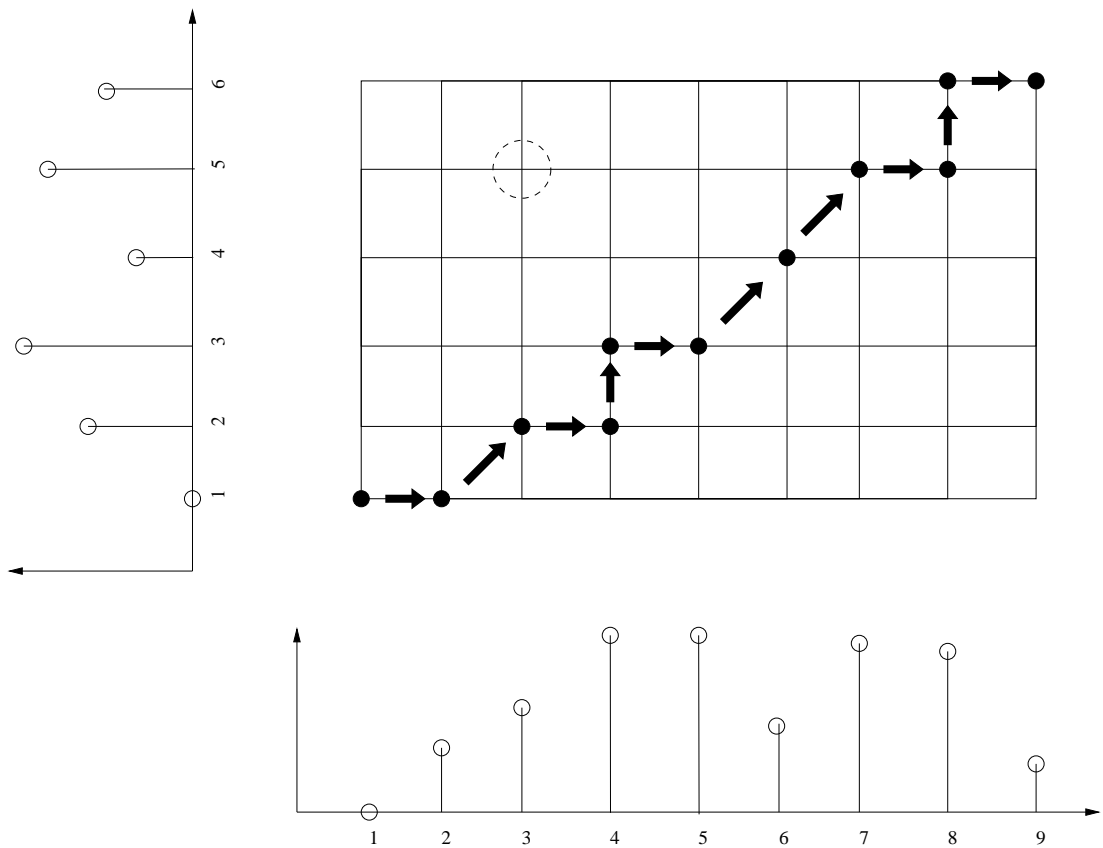


Figure A.2: Grid representation of the optimal warp. The nodes of the grid corresponds to pair of points of the two sequences we want to match. They store the cost of the optimal warp up to that point. The arrows and black dots represent the optimal warp found for matching the two sequences.

of matching two point sequences. Figure A.2 shows an example of the usual representation of that warp. The grid nodes store the similarity values between two subsequences built using the first elements of the point sequences. For instance the grid node circled in figure A.2 represents the similarity between subsequences built using the 5 first elements of one sequence and the 3 first elements of the other one. The arrows represent the optimal warp found between the two sequences. It is the path of lowest cost between the bottom-left corner of the grid to the top-right corner. However, we are only interested in the final cost, which is the value of the similarity between the two point sequences. This value is stored at the top-right node of the grid.

Appendix B

The normalised cut algorithm

The normalised cut algorithm [88] can cluster a set of elements based only on the values of a similarity measure between all possible pairs of elements. The approach is a spectral clustering method. It is based on the properties of eigenvectors from a matrix computed using the similarities between each pairs of elements.

In the normalised cut scheme, the clustering is seen as a graph partitioning problem. The nodes of the graph are the elements and the weights on the graph edges connecting two nodes are the similarities measured between the two corresponding elements. We use the similarity measure described in the previous section. We seek to partition the graph into subgraphs with high similarities between the nodes of the same subgraphs and a low similarities between nodes from different subgraphs.

Let G be that graph with nodes V and an adjacent graph weight matrix W . The element $W(u, v)$ of W is the similarity measured between the elements represented by the nodes u and v . We can break G into two disjoint sets A and B so that $A \cup B = V$ and $A \cap B = \emptyset$ by simply removing the edges connecting the two parts. The cost of removing those edges is computed as the total weight of the edges that have been removed:

$$cut(A, B) = \sum_{u \in A, v \in B} W(u, v) \tag{B.1}$$

This cost is called a “cut” in the graph theory language. We want to minimise this cut value when partitioning G .

However, minimising the cut value encourages cutting isolated nodes in G .

To avoid that problem, the cost of removing edges from A to B is considered as a fraction of the total weight of connections from nodes in A to all nodes in the graph. This leads to a new cost measure called the normalised cut:

$$Ncut(A, B) = \frac{cut(A, B)}{asso(A, V)} + \frac{cut(A, B)}{asso(B, V)} \quad (\text{B.2})$$

where $asso(A, V) = \sum_{u \in A, v \in V} W(u, v)$ is the total connection weight from all the nodes in A to all the nodes in V . $asso(B, V)$ is defined in a similar way. Our aim is now to find a partition of G that minimise the normalised cut between the two parts.

Let \mathbf{x} be a $|V|$ dimensional vector, that represents the partition of V into two sets A and B . $x_i = 1$ if the node i is in A and $x_i = -1$ if the node i is in B . We define:

$$Ncut(\mathbf{x}) = Ncut(A, B) \quad (\text{B.3})$$

Let $d_i = \sum_{j \in V} W(i, j)$, the total weight of edges between the node i and all the other nodes in the graph. Let $D = \text{diag}(d_i)$ and let $b = \frac{\sum_{x_i > 0} d_i}{\sum_{x_i < 0} d_i}$. It is proven in [87] that:

$$\min_{\mathbf{x}} Ncut(\mathbf{x}) = \min_{\mathbf{y}} \frac{\mathbf{y}^T (D - W) \mathbf{y}}{\mathbf{y}^T D \mathbf{y}} \quad (\text{B.4})$$

subject to the constraints $y_i \in \{1, -b\}$ and $\mathbf{y}^T D \mathbf{1} = 0$. The change of variable:

$$\mathbf{y} = \frac{\mathbf{1} + \mathbf{x}}{2} - b \frac{\mathbf{1} - \mathbf{x}}{2} \quad (\text{B.5})$$

has been used. So $y_i = 1$ if the node i is in A and $y_i = -b$ if the node i is in B .

$\frac{\mathbf{y}^T (D - W) \mathbf{y}}{\mathbf{y}^T D \mathbf{y}}$ is called the Rayleigh quotient [36, 63]. The minimisation of equation B.4 can be approximate by relaxing \mathbf{y} to take real values and solving the generalised eigenvalue system:

$$(D - W) \mathbf{y} = \lambda D \mathbf{y} \quad (\text{B.6})$$

Equation B.6 can be rewritten as a standard eigensystem:

$$D^{-\frac{1}{2}} (D - W) D^{-\frac{1}{2}} \mathbf{z} = \lambda \mathbf{z} \quad (\text{B.7})$$

with $\mathbf{z} = D^{-\frac{1}{2}} \mathbf{y}$.

By observing the following:

- $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is a symmetric diagonally dominant real matrix with non-negative diagonal entries so it is positive semidefinite; it has therefore only positive eigenvalues
- $D^{-\frac{1}{2}}\mathbf{1}$ is an eigenvector of $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$; its corresponding eigenvalue is 0
- $D^{-\frac{1}{2}}(D - W)D^{-\frac{1}{2}}$ is symmetric so its eigenvectors are orthogonal to each other, in particular the eigenvector z_1 that corresponds to the second smallest eigenvalue is orthogonal to $D^{-\frac{1}{2}}\mathbf{1}$ and thus satisfy the constraint $\mathbf{y}_1^T D \mathbf{1} = 0$ with $\mathbf{z}_1 = D^{\frac{1}{2}}\mathbf{y}_1$

we can use the following theorem [63]: Let A be a $n \times n$ real symmetric matrix with eigenvalues $\lambda_1 \leq \lambda_2 \leq \dots \leq \lambda_n$. The Rayleigh quotient $\frac{\mathbf{x}^T A \mathbf{x}}{\mathbf{x}^T \mathbf{x}}$ is minimised by the j^{th} eigenvector, under the constraint that \mathbf{x} is orthogonal to the eigenvector associated with the $j - 1$ smallest eigenvalues of A . The minimum is equal to the λ_j .

In our case, we can conclude that \mathbf{z}_1 minimises $\frac{\mathbf{z}^T D^{-\frac{1}{2}}(D-W)D^{-\frac{1}{2}}\mathbf{z}}{\mathbf{z}^T \mathbf{z}}$ under the constraint $\mathbf{z}^T D^{\frac{1}{2}}\mathbf{1} = 0$. So, $\mathbf{y}_1 = D^{-\frac{1}{2}}\mathbf{z}_1$ minimises $\frac{\mathbf{y}^T (D-W)\mathbf{y}}{\mathbf{y}^T \mathbf{y}}$ under the constraint $\mathbf{y}^T D \mathbf{1} = 0$.

We need only to compute the eigenvector associated with the second smallest eigenvalue. We use the Lanczos algorithm to compute that vector. This algorithm iteratively approximates the eigenvectors until convergence [36]. In contrast to singular value decomposition, only two eigenvectors have to be computed to get the eigenvector we are looking for. This makes the Lanczos algorithm quicker, especially for large matrices.

The solution \mathbf{y}_1 of the relaxed problem can be used as an approximate solution to the original discrete problem. We need an extra condition on the components of the solution \mathbf{y}_1 : $y_i \in \{1, -b\}$. In the ideal case, the components of the eigenvector takes only two discrete values, which represents the two classes. If it is not the case, we need to choose a splitting point to partition the values of the components of the eigenvector. Here we simply choose 0 as a splitting point. All the positive components represent elements from one class and all the negatives ones represent elements from the other class. Other methods exist for choosing the splitting point [89].

We can then separate the set of elements into two sets, one containing the elements corresponding to positive values in the eigenvector and the other one

containing elements corresponding to negative values in the eigenvector. We recursively cluster the resulting two element sets until we reach a given number of clusters.

This number of resulting clusters is usually not a power of two as one would expect given our description of the algorithm. This is due to the fact that the normalised cuts algorithm does not always separate the groups into two for each steps. It is possible that the second eigenvector given by the Lanczos algorithm has only positive or only negative values. This can be due either to rounding errors in the computation of the eigenvector or to cases where the choice of 0 as a threshold value for selecting the two groups is a bad choice (the solution of the normalised cuts algorithm is only an approximation of the continuous case). It can also be due to the fact that the group we are trying to split only contains one element (due to previous splits). In that case, we do not split the group so the count of the number of group will not be a power of two.