PROBABILISTIC LEARNING AND MODELLING OF OBJECT DYNAMICS FOR TRACKING

TAY HAN BOON TERENCE

(B.Sc. (Hons), NUS)

A THESIS SUBMITTED

FOR THE DEGREE OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2000

Acknowledgements

I would like to thank my thesis supervisor, Dr Sung Kah Kay, for his patient guidance and supervision, without which this work would not have been possible.

I am also grateful to the many people who have helped me in the course of this work. To Annie Hui for the insights she has given me. To Lee Lik Wee for the many interesting discussions and the loan of his fish tank. To Handoko Setyawan for the loan of his video camera, and to Rini Setiadarma, Jack Yeo, Fan Lixin, Tang Mengting and everyone else for literally lending me a hand with my experiments.

In memory of

Sung Kah Kay

Table of Contents

Table of Contents iv
List of Figures vii
List of Tables viii
Summary ix

1 Introduction and Background **1**
1.1 Definitions and Notations 2
1.2 The Predict-Verify Framework for Tracking 4
1.3 Existing Work on Prediction 5
1.3.1 Parametric Prediction Models 6
1.3.2 Non-parametric Prediction Models 10
1.4 Thesis Organisation and Contribution 12

2 A Probabilistic Prediction Model **14**
2.1 Developing the Probabilistic Predictor 14
2.1.1 Modelling Dynamics as a Density Estimation Problem 14
2.1.2 Modelling the Joint Probability Density 15
2.1.3 Using the Predictor 16
2.2 Features of the Predictor 18
2.3 Implementation Issues 21
2.3.1 Choice of the Form of $\mathbf{X}$ 21

2.3.2 Artificial Augmentation of Training Data 23
2.3.3 Estimating the Joint Density Parameters 24
2.3.4 Bootstrapping 26
2.3.5 Initialisation Strategy 27
2.3.6 Recipe for Using the Predictor 28

3 Experiments **30**
3.1 The CONDENSATION Algorithm 31
3.2 Fish Tracking Experiments 33
3.2.1 Object Model 34
3.2.2 Verification Process 36
3.2.3 Rationale for Using Fish Tracking 38
3.2.4 Experiment Procedure 39
3.2.5 Experiment Results 41
3.2.6 Experiment Results of Fish Tracking with Occlusion 45
3.3 Hands Tracking Experiments 47
3.3.1 Object Model 48
3.3.2 Verification Process 49
3.3.3 Experiment Procedure 50
3.3.4 Experiment Results 51

4 Analysis and Tuning of the Predictor **56**
4.1 Methodology of Analysing the Predictor 57
4.1.1 Parameters of Interest 57
4.1.2 Quantifying the Performance of the Predictor 58
4.2 The "CHIME DIVA" Experiment 60
4.2.1 Experiment Setup 60
4.2.2 Analysis of Experiment Setup 62

4.3 Experiment Results 63
4.3.1 Effects of Varying K 63
4.3.2 Effects of Varying N 67
4.4 Running Time Analysis 70
4.5 Tuning the Predictor 71
4.5.1 Automatic Bootstrapping 71
4.5.2 Choosing K 73
4.5.3 Early Stopping to Prevent Over-fitting 73

5 Conclusion and Future Directions **76**
5.1 Dynamics Classification 77
5.2 Better Modelling of the Joint Density 78
5.3 Tracking Multi-Part Objects 80

A Derivations **81**
A.1 Deriving the Prediction State Density Function 81
A.2 Deriving the Backward Prediction Equations 86
A.3 Deriving the Time Complexity of the Predictor 87
A.4 Further Justification for Using the Minimum Distance Statistic 88
A.4.1 Average distance statistic is a poor choice 89
A.4.2 The minimum distance statistic is the best choice 90
A.4.3 Other statistics may not be perfect either 91

Bibliography **92**

List of Figures

1.1 The task of tracking 1
1.2 Tracking Framework 5

3.1 Formalised tracking framework 32
3.2 Setup of fish experiments 34
3.3 Image information as seen through the eyes of the verification process . . 41
3.4 Results of fish tracking 43
3.5 Setup for fish tracking with occlusion 45
3.6 Results of fish tracking with occlusion 46
3.7 Setup of hands experiments 47
3.8 Point distribution model (PDM) 49
3.9 Results of hands tracking: basic hand shapes 53
3.10 Results of hands tracking: large hands 54
3.11 Results of hands tracking with clutter. 54
3.12 Results of hands tracking with occlusion. 55

4.1 "CHIME DIVA" experiment setup 61
4.2 "CHIME DIVA" experiment results: effects of varying K with fixed $N = 100$ 64
4.3 "CHIME DIVA" experiment results: effects of varying N with fixed $K = 20$ 68
4.4 "CHIME DIVA" experiment results: comparing against the constant velocity predictor 69

List of Tables

3.1 Tracking accuracy of fish experiments 44

Summary

Given an image sequence, the aim of tracking is to find the target object in each of these images by exploiting temporal coherence information. Let $\mathbf{X}_t$ be the target object's state at time t, which can be a set of parameters specifying the object's position, shape, pose, image appearance and other attributes. Finding the object means recovering this object state at all time steps. This problem can be decomposed and independently addressed as two sub-tasks:

1. **Prediction step**. Given a set of the most recent q object states up to time t, $\mathcal{X}_t^{(q)} = \{\mathbf{X}_{t-q+1}, \ldots, \mathbf{X}_t\}$, predict the next object state at time $t+1$, denoted by $\mathbf{X}_{t+1}$. In probabilistic terms, one can view $\mathbf{X}_{t+1}$ as an a-priori hypothesis on the object's next state based on an assumed temporal coherence model of object state dynamics.

2. **Verification step**. Given a next-state hypothesis $\mathbf{X}_{t+1}$, take image measurements $\mathbf{Z}_{t+1}$ to compute the observation likelihood $p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1})$ at time $t+1$. When combined with results from the prediction step, this leads to an a-posteriori goodness measure or distribution on next object states, given by: $p(\mathbf{X}_{t+1}|\mathbf{Z}_{t+1}) \propto p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1})p(\mathbf{X}_{t+1})$.

The verification step involves building good object models and similarity measures that can discriminate well between the target object and scene clutter, possibly even when the object is partially occluded. While interesting and challenging in its own right,

this topic is not within our scope of discussion. The prediction step involves building good models of object state dynamics that accurately and tractably reflect the object's a-priori next-state distribution $p(\mathbf{X}_{t+1})$ given $\mathcal{X}_t^{(q)}$ or its distribution $p(\mathcal{X}_t^{(q)})$. This is the focus of this thesis.

We have developed a probabilistic prediction model that offers the following benefits. Firstly, it treats modelling dynamics as a joint density estimation problem in $(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$-space, so that prediction can be directly treated as computing $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$ and, in principle, be implemented non-parametrically. This means that with sufficient reliable training data, one can choose not to rely on possibly incomplete and unreliable prior human knowledge to build parametric models of dynamics, as most existing predictors have done. There is also an efficient Gaussian mixture representation to implement this conceptual prediction framework so that predictions can be generated in a tractable fashion during operation. Furthermore, it fits nicely into a wide variety of tracking frameworks, including the CONDENSATION algorithm [20].

In this thesis, we present the development of this probabilistic prediction model. We also describe how a tracker can be built using this predictor and the experiments we have conducted. These experiments show that the use of the predictor leads to robust tracking performance. In addition, we also present a detailed analysis of the predictor as well as suggestions for tuning it to improve performance. This thesis concludes with a discussion of future work.

Chapter 1

Introduction and Background

In computer vision, the problem of tracking can be defined as follows: given an image sequence, find the target object in each image frame by exploiting temporal coherence information. The idea is to build an automated system that is capable of following a target object in an image sequence, which it may be obtained from some pre-recorded video footage of the target scene or may be obtained in real time from a camera. Figure 1.1 depicts this task diagrammatically. The output of the tracker is the *state* of the target object (which we will define more formally in section 1.1) in each image. This object state will most likely include the object's (x, y) coordinates in the image, and probably a number of other parameters such as those specifying the object's shape, pose, image appearance and other attributes as well. Finding the target object in each image frame means recovering this object state in all image frames.

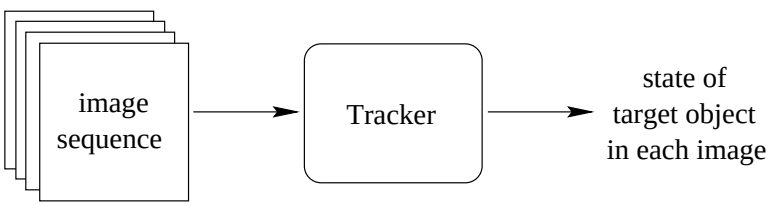


Figure 1.1: The task of tracking

There are many applications where a good tracker would be useful. An example would be traffic monitoring, where a camera can be positioned to capture images of passing traffic. With the states obtained by the tracker through processing these images, useful information about traffic flow such as statistics about the speed of the vehicles, the amount of traffic congestion and the spotting of reckless drivers can be derived. Another example of the use of tracking would be in a surveillance system such as that of a bank. Instead of tracking vehicles, the tracker would now track people with the goal of analysing human traffic characteristics. It may also be desirable to be able to track a person's hands, which has applications in interpreting gestures such as those in sign language.

These potential applications provide a motivation for the development of a good tracker. In the next two sections, we will discuss a generic framework that can be used to solve the tracking problem. This framework has two key components, namely the prediction process and the verification process (also indicated in figure 1.1). While we will speak of both processes and of tracking in general, the main focus and contribution of this thesis is in the prediction process. The remainder of this chapter will present a conceptual view of the goal of the prediction process and discuss existing work on prediction and their limitations. We conclude this chapter by describing the organisation and contributions of this thesis.

1.1 Definitions and Notations

Before we proceed, we introduce some basic definitions and notations. First, an *object model* is an internal representation of an object in the real world. For example, one object model for describing a human is a curve representing its 2-dimensional contour as seen in an image. Another object model for a human is one that breaks the various limbs of the human body into cylindrical components that are connected at the joints. Naturally, some models are better than others at describing the object, just like the "cylindrical limbs"

representation is better than the "2-D contour" representation. However, depending on the intended tracking application, a simple object model could go some way in simplifying the problem and making the solution tractable using existing hardware.

In any object model, an object can be represented by an *object state*, which is a set of parameters describing the object of interest. These include parameters affecting position, shape, pose, appearance and other attributes of the object. This state is denoted by the vector $\mathbf{X}_t$, where t references time in discrete units. The space spanned by these object attributes is called the *object state space*.

The next term we define is the *prediction state*, which is the output of the prediction process. The space spanned by the elements of the prediction state is called the *prediction state space*. The prediction state (and hence, the prediction state space as well) is often the same as the object state (object state space), though as we shall see in the experiments, they need not be. This distinction is not significant in our conceptual understanding of this thesis, hence we may assume the prediction state to be the same as the object state and denote both by $\mathbf{X}_t$. In this thesis, unless otherwise indicated, we also follow the convention that the current time step is referenced as t and future time steps are as expressed as $t+1$, $t+2$ and so forth. A prediction state generated at time t for time $t+1$ is then denoted as $\mathbf{X}_{t+1}$.

The final term we define is the *prior state*, which is the input into the prediction process. The prior state at time t is denoted by $\mathcal{X}_t^{(q)}$ and it is described by a set of the most recent q prediction states as follows:

$$\mathcal{X}_t^{(q)} = \begin{bmatrix} \mathbf{X}_{t-q+1} \\ \vdots \\ \mathbf{X}_t \end{bmatrix}. \tag{1.1}$$

This definition is somewhat restrictive but again, it is sufficient for the conceptual understanding of the predictor. Later in section 2.3.1, we will modify this definition to take

on a more general form.

1.2 The Predict-Verify Framework for Tracking

The problem of tracking can be decomposed and independently addressed as two subtasks as shown in figure 1.2 and described as follows:

1. **Prediction step**. Given the prior at time t, $\mathcal{X}_t^{(q)}$, predict the object state at time $t+1$, $\mathbf{X}_{t+1}$, or its distribution, denoted by $p(\mathbf{X}_{t+1})$.

2. **Verification step**. Given a next-state hypothesis $\mathbf{X}_{t+1}$, take image measurements $\mathbf{Z}_{t+1}$ to compute the observation likelihood $p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1})$ at time $t+1$.

Intuitively, we can view the prediction step as attempting to guess the state of the object at $t+1$ given the state history of the object up to time t. In probabilistic terms, we can view $\mathbf{X}_{t+1}$ (or its distribution $p(\mathbf{X}_{t+1})$) as an a-priori hypothesis on the object's next state based on an assumed temporal coherence model of object state dynamics. The verification step then computes the accuracy of the prediction (or set of predictions in the case of $p(\mathbf{X}_{t+1})$) by taking image measurements. When this is combined with results from the prediction step, it leads to an a-posteriori goodness measure or distribution on next object states, given by: $p(\mathbf{X}_{t+1}|\mathbf{Z}_{t+1}) \propto p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1})p(\mathbf{X}_{t+1})$. This two-step process can then be iterated to achieve tracking over time.

The effectiveness of the verification step is dependent on two factors. The first is the ability to build good object models and the second is the ability to extract useful information from an image. These two factors allow the verification process to define a metric that can discriminate well between the target object and scene clutter, possibly even when the object is partially occluded. While interesting and challenging in its own right, this topic is not within the scope of our discussion.

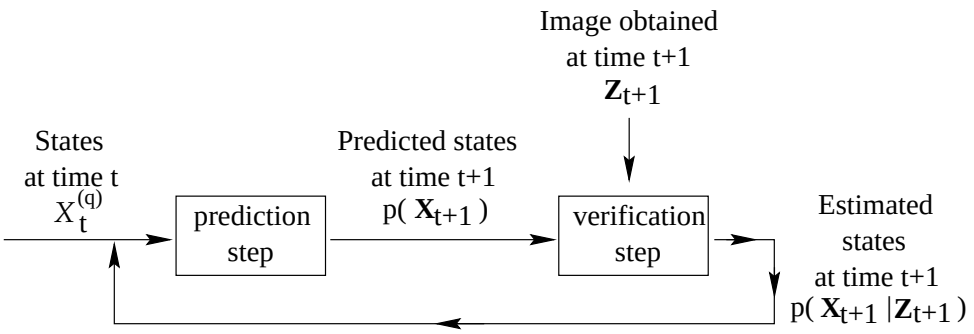


Figure 1.2: Tracking Framework

The prediction process involves building good models of object state dynamics that accurately and tractably reflect the object's a-priori next-state distribution $p(\mathbf{X}_{t+1})$ given $\mathcal{X}_t^{(q)}$ or its distribution $p(\mathcal{X}_t^{(q)})$. A good model of object state dynamics allows the predictor to properly exploit temporal coherence information to constrain the object state space at time $t+1$. This has the effect of discounting clutter that lies outside the constrained object state space as well as compensating for the loss of image measurements where the target object is occluded. Hence the prediction process is able to complement the verification process in the task of building a robust tracker. It is the goal of this thesis to present the construction and analysis of a novel prediction model.

1.3 Existing Work on Prediction

Generally, research on prediction can be divided into two broad categories: parametric and non-parametric models. In sections 1.3.1 and 1.3.2, we will discuss these two categories. A detailed understanding of these predictors will also aid in the discussion of our prediction model found in later chapters.

1.3.1 Parametric Prediction Models

A large number of existing prediction models fall into this category. In general, the process these predictors use to generate predictions, or equivalently to estimate $p(\mathbf{X}_{t+1})$, from the current state history involves two steps:

1. Construct a functional mapping $F : \mathcal{X}_t^{(q)} \mapsto \mathbf{X}_{t+1}$ that computes a most probable next-state hypothesis from the current state history.

2. "Smear" the most probable $\mathbf{X}_{t+1}$ from step 1 with an appropriately chosen probability distribution to express uncertainty in the output hypothesis.

As we shall see in the following subsections, the specific form of $F()$ in step 1 is often derived from prior human knowledge of the underlying object state dynamics [16, 5, 15, 7], with possibly a small number of free parameters to be inferred from actual data. In cases where multiple independent physical causes can influence object dynamics, $F()$ can be an arbitrated collection of multiple functional mappings, where the number and specific form of these mappings are often (but need not be) pre-defined using prior human knowledge [21]. For step 2, most existing work assume an appropriate Gaussian "smearing" form, where the main, and perhaps sole, purpose of expressing next-state information as a distribution $p(\mathbf{X}_{t+1})$ is to capture and quantify effects of uncertainty in the predicted output hypothesis.

Simplistic Prediction Models

The first class of parametric prediction models we discuss is what we have termed *simplistic prediction models*. They are described as simplistic not because they lack technical sophistication but rather that their understanding of object state dynamics are primarily derived from prior human knowledge. Little or no learning from training examples is involved.

One such predictor is the *constant position model*, which assumes that the object state at time t is close to the state at time $t+1$. This way, the object state space at $t+1$ is constrained to a narrow region near t. This is expressed by:

$$\mathbf{X}_{t+1} = \mathbf{X}_t + \omega \tag{1.2}$$

where ω is a multivariate random variable incorporating noise into the prediction. A slightly smarter method, known as the *constant velocity model*, is an expression of Newton's first law of inertia by assuming a constant velocity between two time steps. This is given by:

$$\mathbf{X}_{t+1} = \mathbf{X}_t + \dot{\mathbf{X}}_t + \omega \tag{1.3}$$

where $\dot{\mathbf{X}}_t$ is the instantaneous velocity of the object at time t. There are many other variations of predictors in this class, e.g. those assuming constancy in higher order moments and more sophisticated ones like that in [28] where intricate details of a car's steering controls were used to construct this predictor. Various simplistic predictors can also be combined if prior human knowledge deemed this more suitable for describing the dynamics of the target object [16, 5, 15].

In these models, a suitable form of ω must also be chosen to represent the uncertainty of the prediction. One method is to define ω as a vector of independent uniform random variables, $\mathcal{U}(0, \delta)$, where $[0, \delta]$ defines the range of numbers to be generated. δ can be interpreted as controlling the size of the region in object state space to be searched. Another noise model is the normal distribution given by:

$$\omega \sim \mathcal{N}(\mu, \Sigma). \tag{1.4}$$

where μ and Σ are the mean and covariance of the distribution respectively.

There are many variants of simplistic predictors, each combining different attributes of the predictor form (e.g. constant position, constant velocity, etc) with an appropriate noise model (e.g. uniform, Gaussian, etc) to represent uncertainty. The exact form is not as important as the fact that they are parametric prediction models of the form described earlier: the predictor form is the functional mapping in step 1 of the parametric predictor and the noise model is the smearing process in step 2. Both these steps require a large amount of prior human knowledge to construct and use, such as in the choice of sensible settings for δ, μ and Σ. Where the object state space is easily interpretable, suitable settings of these parameters can be chosen. An example of this is if the object state comprises only the (x, y) image coordinates of the object. However, such interpretable object states are rare. Object states that include parameters describing shape, pose and appearance often do not have such direct interpretations, making it difficult to determine suitable values for the parameters.

Stochastic Differential Equations

Another form of a parametric predictor is one using stochastic differential equations (SDE), specifically the class of linear SDEs [11]. The linearity assumption is a great convenience to the model for several reasons. Firstly, generating predictions using this model can be done efficiently via a q-order auto-regressive (AR) process, expressed as:

$$\mathbf{X}_{t+1} = \left(\sum_{i=1}^{q} \mathbf{A}_i \mathbf{X}_{t-i+1} \right) + \mathbf{D} + \mathbf{B}\omega \tag{1.5}$$

where the distribution of the noise model is again normal, only this time, $\mu = \mathbf{0}$ and $\Sigma = \mathbf{I}$. The various $\mathbf{A}_i$ represent the parameters of the q-order linear differential equations in matrix form and $\mathbf{B}$ is a matrix specifying the correlation between the elements of the vector ω. $\mathbf{D}$ is a constant vector representing the object state's constant drift.

The second advantage of the linearity assumption is that it allows for an analytical

solution to the problem of automatically estimating the parameters of the AR process [7, 36] from training examples. Provided that the assumption that the object dynamics can be modelled by an SDE holds, this prediction model has exhibited good performance [20, 7, 6].

However, failure of such an assumption is not uncommon. A simple example would be in the case of a bouncing ball where its motion under the effects of gravity and its motion due to colliding with the ground correspond to two different modes of dynamics. This prevents such dynamics from being modelled by a single AR process. The natural extension of this model is to use multiple AR processes, each corresponding to one mode of dynamics. This is the essence of the prediction model described in [21]. Switching between these n AR processes is done automatically using an $n \times n$ discrete transition matrix:

$$\mathbf{T} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & a_{ij} & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} \tag{1.6}$$

where the element a_{ij} is the probability of switching from AR process i to j. These elements are constrained so that each row sums to 1, i.e. $\sum_{j=1}^{n} a_{ij} = 1$. All the parameters of this extended model, including that of the transition matrix, can still be learnt efficiently as described in [8].

Prediction models based on differential equations are certainly an improvement over simplistic predictors. They are capable of modelling a wider variety of dynamics and have been shown to work with state spaces comprising parameters of shape as well as position [20]. However, the assumption that the object dynamics can be modelled by linear differential equations is still a restrictive one. For one thing, it assumes that these dynamics are smoothly-varying, which may not necessarily hold for complex state spaces. Furthermore, the multiple AR process model also assumes that the number of modes in

the object dynamics is known. In the example of the bouncing ball, the number of modes is two, corresponding to the effects of gravity and the effects of ground collision. In complex tracking environments, it may be difficult to identify the number of modes, which may lead to failure of these models.

In general, parametric predictors rely on prior human knowledge to choose the predictor form and the noise model to represent uncertainty effects. Such predictors are acceptable if the human knowledge is reliable and complete, such as for the case of well-understood object dynamics of simple physical systems. However, such knowledge is often incomplete for complex physical systems causing parametric predictors to fail.

1.3.2 Non-parametric Prediction Models

Broadly speaking, a non-parametric prediction model is one that cannot be described by the two-step conceptual process of a parametric predictor in section 1.3.1. More specifically, we are referring to a class of prediction models that do not explicitly construct the mapping $F()$ to compute the most probable prediction state. To the best of our knowledge, only two prediction models can be categorised as being non-parametric. The first is the discrete-state Markov process used in [18], which we discuss next, and the other is the new prediction model that we have developed in this thesis.

Discrete-State Markov Processes

The idea of this prediction model is to divide the object state space into many discrete patches, each described by its centre and its principal axes of variation [10]. Object state dynamics can then be modelled by modelling the probability of transitions between these patches. This prediction model can be implemented using a probability transition matrix similar to that of (1.6). The element a_{ij} is the probability of an object state moving from patch i to j. This transition matrix can be learnt from training examples [18] and applied as follows:

1. Given the current object state $\mathbf{X}_t$, determine the patch, i, to which this state belongs. A simple way of doing this is to find the patch whose centre is closest to $\mathbf{X}_t$. Other methods of doing this can also be used, if deemed applicable.

2. Given the source patch, i, sample from the probability transition matrix to determine the destination patch, j. Many destination patches may be possible, allowing for uncertainty in the value of j.

3. To obtain the next-state hypothesis $\mathbf{X}_{t+1}$, two cases are considered.

   (a) If $i = j$, then $\mathbf{X}_{t+1} = \mathbf{X}_t + \omega$, where ω is a random variable and has the task of performing zero-mean Gaussian "smearing", where the covariance of the Gaussian is defined using the principal axes of patch j.

   (b) If $i \neq j$, then $\mathbf{X}_{t+1} = \mu_j + \omega$, where μ_j is the centre of patch j and ω has the same purpose and implementation as in the case of $i = j$.

This prediction model was developed by Heap and Hogg [18] to cope with the limitation that the use of SDEs assume that the object state dynamics must be smoothly-varying. Since each patch in the object state space is capable of making the transition to any other patch, discontinuous dynamics can be modelled by the discrete-state Markov process. To a certain extent, continuous dynamics are also captured because each predicted shape is perturbed by noise.

The problem with such a model is that it assumes that the object states cluster in discrete patches. Where continuous dynamics occur, a large number of patches must be used to represent them in a discrete form. Dynamics in image space, i.e. 2-D image motion, is a simple example of continuous dynamics and would not be represented well by a discretised prediction model. Alternatively, continuous dynamics can be modelled separately using some other prediction model, e.g. an AR process, but this makes the overall prediction process complicated and cumbersome to use. Prior human knowledge

must be used to decide how to partition the object states into those attributes that exhibit continuous or discontinuous changes. Finally, and probably the most limiting factor of this prediction model is that it was designed specifically for the hierarchical point distribution model (HPDM) [17], which is the object model used by Heap and Hogg. The implementation of the noise model requires the knowledge of the principal axes of each patch, which is readily available in the HPDM, but may not be if a different object model is used. Such specificity makes it difficult to apply this prediction model to trackers using other object models. Further analysis of the problems of this prediction model will be presented in section 2.2 when we compare it to the model we have developed.

1.4 Thesis Organisation and Contribution

In this thesis, we present a novel non-parametric prediction model built from probabilistic principles. Chapter 2 presents the construction of the probabilistic predictor. It then analyses the features that such a predictor exhibits as well as its advantages over existing predictors. The major implementation issues that need to be considered are also discussed. In chapter 3, we integrate the probabilistic predictor with the CONDENSATION algorithm [20] to build a tracker. Several tracking experiments were performed to show how our predictor leads to robust tracking performance and its suitability to be used in a practical application.

However, tracking performance is the result of many components, such as a good object model and an appropriate verification process. In chapter 4, we describe a different approach to analyse the effectiveness of the predictor and to measure the effects of varying its only user-adjustable parameter. Building on this information, we also suggest several methods of tuning the predictor to improve performance. In the concluding chapter, we present several directions that future work can take. An appendix is also included to present the details of the derivation and analysis of the predictor that was beyond the

scope of the main thesis.

To conclude, the key contribution of this thesis is a conceptual understanding of the prediction process that has led to the casting of the problem of modelling dynamics into one of probability density estimation. Using this conceptual understanding, a novel prediction model has been derived from probabilistic principles. Given sufficient reliable training data, this prediction model is capable of modelling a wide range of dynamics. It is also efficient, simple, and does not need to rely on prior human knowledge. Finally, its generality allows it to fit nicely into a wide variety of tracking frameworks.

Chapter 2

A Probabilistic Prediction Model

The ability to understand and model a wide variety of object state dynamics is a key aspect of a good predictor. We begin this chapter by describing a novel method of doing this by casting this problem into a problem of probability density estimation. We then show how this leads to a predictor that can be used efficiently. In section 2.2, we describe various important aspects of our predictor and discuss its advantages over parametric and other non-parametric predictors. The various implementation issues of using the predictor are dealt with in section 2.3, which concludes with a step-by-step guide to using the predictor.

2.1 Developing the Probabilistic Predictor

2.1.1 Modelling Dynamics as a Density Estimation Problem

In line with the view that modelling object state dynamics is about modelling the uncertainty of a next-state hypothesis, prediction can be seen as computing the *prediction state density*, denoted by $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$, which is the probability of the state $\mathbf{X}_{t+1}$ occurring at time $t+1$ given that the prior $\mathcal{X}_t^{(q)}$ has occurred in the previous time steps. Using Bayes'

rule, we have:

$$p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)}) = \frac{p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})}{p(\mathcal{X}_t^{(q)})}. \tag{2.1}$$

where $p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$ is the joint probability density between the recent state history and the prediction state. This density represents the a-priori knowledge about the dynamics of the target object and can be learnt from training examples. This interpretation of the prediction state density casts the problem of modelling dynamics into one of probability density estimation. The task is then to model and learn $p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$ which can then be used with (2.1) to give the prediction state density, $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$, assuming that $\mathcal{X}_t^{(q)}$ is available.

2.1.2 Modelling the Joint Probability Density

In general, any form of probability density estimation framework can be used to approximate $p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$. The parameters of this framework can be learnt from examples obtained from training sequences of the object to be tracked, hence obviating the need for restrictive assumptions about the dynamics. Here, we present a Gaussian mixture approach using the EM algorithm [13, 35] to estimate its parameters. This approach has two advantages:

1. It is a universal approximator capable of modelling a wide range of functional mappings, and hence object dynamics as well. If the number of Gaussian components is large, this approach can be described as being non-parametric.

2. Generating predictions using this approach is computationally efficient and technically sound. See section 2.1.3 for details.

This Gaussian mixture approach to modelling the joint probability density is expressed as follows:

$$p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)}) = \sum_{k=1}^{K} w_k \, g(\mathbf{X}; \mu_k, \Sigma_k), \tag{2.2}$$

where

$$\mathbf{X} = \begin{bmatrix} \mathcal{X}_t^{(q)} \\ \mathbf{X}_{t+1} \end{bmatrix}. \tag{2.3}$$

K is the number of Gaussian components used and $g(\cdot; \mu_k, \Sigma_k)$ is a Gaussian function with joint mean and covariance matrices μ_k and Σ_k respectively. w_k are the weights controlling the mixing proportions of these components under the constraints that

$$w_k \geq 0 \quad \text{and} \quad \sum_{k=1}^{K} w_k = 1. \tag{2.4}$$

Note that in some situations, e.g. when there is limited training data available, Σ_k need not be a full covariance matrix and can be a regularised covariance matrix with fewer than $d(d+1)/2$ free parameters, where d is the dimensionality of $\mathbf{X}$.

2.1.3 Using the Predictor

For tracking applications, applying the predictor means that given $\mathcal{X}_t^{(q)}$ or its distribution, $p(\mathcal{X}_t^{(q)})$, we wish to obtain a set of samples for $\mathbf{X}_{t+1}$ that is representative of its predicted next-state distribution, $p(\mathbf{X}_{t+1})$. We first consider how to do this for the case of a fixed $\mathcal{X}_t^{(q)}$, or equivalently where $p(\mathcal{X}_t^{(q)})$ is a delta function. The extension to a more general $p(\mathcal{X}_t^{(q)})$ is a simple modification that we will describe later.

Using the predictor involves the following two main steps. The first step is that given $\mathcal{X}_t^{(q)}$ at time t, a closed-form expression of $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$ is computed. To do this, begin

by breaking each μ_k and Σ_k into sub-matrices as follows:

$$\mu_k = \begin{bmatrix} p_k \\ q_k \end{bmatrix} \quad \text{and} \quad \Sigma_k = \begin{bmatrix} A_k & B_k \\ B_k^T & C_k \end{bmatrix} \tag{2.5}$$

where p_k and A_k are the mean and covariance matrices of the $\mathcal{X}_t^{(q)}$ portion of $\mathbf{X}$ respectively. Similarly, q_k and C_k are the mean and covariance matrices of the $\mathbf{X}_{t+1}$ portion respectively. The remaining term, B_k, is a rectangular matrix containing the remaining elements in Σ_k and it is square if $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$ have the same dimensionality. This term represents the cross-correlation between the elements of the $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$.

We now substitute (2.2) and (2.5) into (2.1) to obtain the prediction state density as a mixture of K Gaussians:

$$p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)}) \propto \sum_{k=1}^{K} v_k \, g(\mathbf{X}_{t+1}; \mu'_k, \Sigma'_k), \tag{2.6}$$

where

$$\mu'_k = q_k + B_k^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k), \tag{2.7}$$

$$\Sigma'_k = C_k - B_k^T A_k B_k, \tag{2.8}$$

$$\text{and} \quad v_k = w_k \, g(\mathcal{X}_t^{(q)}; p_k, A_k). \tag{2.9}$$

The details of this derivation are well-known [29], which we have also reproduced in the appendix for completeness. Note that the prediction state density is a closed-form expression, and hence can be efficiently obtained.

The second main step is to sample from $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$ in (2.6) to generate hypotheses about the object state at time $t+1$. There are two well-known methods of doing this, namely the rejection method and the transformation method [33]. The rejection method has the advantage of being applicable to any probability density function. However, it tends to be computationally expensive and the time required to generate a prediction

varies. The transformation method, on the other hand, imposes some restrictions on the form of the probability density function, but it is efficient and it is able to generate predictions in a tractable fashion. Fortunately since the prediction state density in (2.6) is an additive mixture of Gaussians, these restrictions are satisfied. The following describes the procedure to generate predictions given the prior, $\mathcal{X}_t^{(q)}$:

1. Select a Gaussian component based on the weights v_k in a uniformly random manner. Let the chosen component be k.

2. Compute μ'_k using (2.7).

3. Obtain the matrix Ψ_k such that $\Sigma'_k = \Psi_k \Psi_k^T$. This can be done by diagonalising the covariance matrix to obtain $\Sigma'_k = \Phi_k \Lambda_k \Phi_k$ where Φ_k and Λ_k are the eigenvectors and eigenvalues of Σ'_k respectively. Then Ψ_k can be obtained as $\Psi_k = \Phi_k \Lambda_k^{1/2}$ [27].

4. Use the *Box-Muller* method [33] to generate a vector $\mathbf{U}$ of zero-mean unit-variance Gaussian random variables.

5. Transform the vector of random variables, $\mathbf{U}$, to have the appropriate mean and covariance using $\mathbf{X}_{t+1} = \Psi_k \mathbf{U} + \mu'_k$ [27]. $\mathbf{X}_{t+1}$ is the prediction generated.

In the case where $p(\mathcal{X}_t^{(q)})$ is available and can be sampled from in a tractable fashion, one can first obtain a representative set of samples of $\mathcal{X}_t^{(q)}$ before applying the above two main steps to all elements in this set. The final outcome is also a set of next-states that is representative of the distribution $p(\mathbf{X}_{t+1}) = \int p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)}) p(\mathcal{X}_t^{(q)}) d(\mathcal{X}_t^{(q)})$ (see [22] for details).

2.2 Features of the Predictor

There are several interesting aspects about our probabilistic prediction model that merit elaboration. Firstly at the conceptual level, we treat the problem of modelling dynam-

ics as a joint density estimation problem in $(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$-space. Prediction can then be directly treated as computing $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$ and, in principle, be implemented non-parametrically. This means that with sufficient training data, one can choose *not* to rely on prior human knowledge to decide on a particular parametric form of predictor and/or noise model. Instead, both of these can be inferred from the training data.

At the implementation level, we used the Gaussian mixture representation, which has several interesting properties with respect to modelling dynamics.

1. Firstly, the Gaussian mixture representation used in (2.2) belongs to a class of universal approximators. With sufficient training data and when K is sufficiently large, one can always obtain an approximation sufficiently close to the true $p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$. This allows the predictor to model complex dynamics having an arbitrary form and number of modes.

2. The use of Gaussians as the generative components in (2.2) implicitly enforces a smoothness constraint on the joint probability density function. This means that our model has the ability to generalise to dynamics not found in the training data. Smoothness along the prediction axis $\mathbf{X}_{t+1}$ represents the uncertainty of the predictions, which is analogous to the addition of noise in the various predictors discussed in section 1.3. Smoothness along the prior-state axis $\mathcal{X}_t^{(q)}$ means that prior states close to that of training examples cause a similar next-state to be predicted. Both of these constraints are reasonable and desirable in real-world tracking environments, hence lending support to our choice of the density model.

3. The resulting prediction state density in (2.6) is also a Gaussian mixture, which means that the process of sampling from it to generate predictions can be done in a tractable fashion during operation.

4. A characteristic of using the Gaussian mixture representation is that the joint probability density does not distinguish between the "input" and the "output" of the

predictor. A "reverse-time" analogue of (2.6), denoted by $p(\mathcal{X}_t^{(q)}|\mathbf{X}_{t+1})$, can be easily derived as shown in appendix A.2. An application of this is in a "reverse-time tracker", which can be used to smooth the posterior state density, as described in [26].

At the conceptual level, our probabilistic predictor is related to Heap and Hogg's discrete state Markov process prediction model [18] in that both are attempting to model object state dynamics by modelling the joint probability density, $p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)})$. However, our probabilistic predictor does so in a more general form without assuming that the object state space can be broken into discrete patches, or even what the object model is. Also, to use Heap and Hogg's predictor, the prior state $\mathcal{X}_t^{(q)}$ must first be quantised to belong to some patch. Our predictor does not require this, which leads to two related advantages:

1. The prediction state density, $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$, in (2.6) was derived in a principled manner based on our Gaussian mixture representation. This is different from Heap and Hogg's predictor in that their prediction state density was derived heuristically through the use of the prediction algorithm described in section 1.3.2. The heuristics used were (i) the method of determining the patch that the prior belongs to (step 1) and (ii) the method of computing $\mathbf{X}_{t+1}$ (steps 3a and 3b). The use of heuristics requires some amount of prior human knowledge to work well.

2. The fact that $\mathcal{X}_t^{(q)}$ is not quantised during the computation of $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$ means that the prediction state density will be specified more accurately. This would better constrain the search for the target object, leading to more resilience against clutter and occlusion as well as greater computational efficiency.

In addition to the benefits of the probabilistic predictor mentioned so far, it also makes no assumptions about the joint probability vector $\mathbf{X}$. This vector can be of arbitrary

dimensionality and can contain any form of prior and prediction states. Hence we are free to choose whatever representation that best suits the tracking environment. The overall generality of our predictor allows it to fit nicely into a wide variety of tracking frameworks, including the CONDENSATION algorithm [20] which we have used for the tracking experiments presented later.

To conclude, all of the above features are achieved with one unified model, as opposed to complicated extensions of other models. This makes the probabilistic prediction model powerful yet simple and succinct. The interpretation of modelling state dynamics as a density estimation problem also opens us to a wealth of existing work done in the field. Techniques like the Gaussian mixture representation and the EM algorithm are well studied, and whose benefits extend naturally to solving the problem of modelling dynamics. Other techniques that can be borrowed from existing work include bootstrapping and the use of virtual examples, both of which will be discussed in the next section.

2.3 Implementation Issues

In the previous sections, we have described the probabilistic prediction model as well as its features. In this section, we delve deeper into the predictor and flesh out the issues that need to be considered when using it. We will conclude this section with a step-by-step summary of how to use our predictor.

2.3.1 Choice of the Form of X

The first issue we consider is the form of $\mathbf{X}$ in (2.3), which in turn means the form of its two constituent parts, $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$. It is common to make the first-order Markov assumption here and set $q = 1$ so that $\mathcal{X}_t^{(q)} = \mathbf{X}_t$. This means that the prediction depends only on the immediately preceding state. In many practical situations, such an assumption is valid and including more information in the prior does not necessarily

improve the prediction. Instead, it increases the dimensionality of $\mathbf{X}$ which makes the density estimation problem more difficult and requiring much more training data. So we must be careful when using values of $q > 1$ since this can quickly lead to an exponential explosion in the difficulty of density estimation.

Another factor to consider is the contents of the prior and prediction states. As mentioned in section 1.1, the prediction state is often the same as the object state. And when $q = 1$, the prior state is also the same as the object state. However, this scenario is overly restrictive and other forms of the prior and prediction states are possible too. For example, it is sometimes desirable to reproduce dynamics that is invariant to the absolute position of the target object in the image. This has applications in, say, tracking a bouncing ball where its motion under the force of gravity is independent of its absolute position in the image. Hence even though the object state contains the ball's absolute position in the image, a better choice for the prior and prediction states may be the ball's instantaneous velocity instead. This causes the prior and prediction states (and hence the prior and prediction state spaces) to be different from the object state (and object state space), and this is the distinction mentioned in section 1.1. This difference is important because it allows informative variables to be used in the predictor; if these variables are poorly chosen, then the predictor will be poor, regardless of the amount of reliable training data used.

In line with this, we can further generalise so that the prior $\mathcal{X}_t^{(q)}$ need not be a concatenation of the q most recent prediction states as was expressed in (1.1). A more general form of $\mathcal{X}_t^{(q)}$ is to allow it to be arbitrarily derived from the recent prediction states, expressed as:

$$\mathcal{X}_t^{(q)} = \begin{bmatrix} \phi_1(\mathbf{X}_{t-q+1}, \ldots, \mathbf{X}_t) \\ \vdots \\ \phi_m(\mathbf{X}_{t-q+1}, \ldots, \mathbf{X}_t) \end{bmatrix} \tag{2.10}$$

where each $\phi_i(\cdot)$ is a function of the q most recent prediction states and m is the dimensionality of $\mathcal{X}_t^{(q)}$. Observe that if the prior is formed by concatenating q n-dimensional prediction states, then $m = qn$. However the new form of $\mathcal{X}_t^{(q)}$ in (2.10) allows for $m < qn$, which has the effect of summarising the recent prediction states to produce an informative prior. Hence, one is free to use more states in the prior (i.e. increase q) where necessary, while avoiding the problem of combinatorially exploding the difficulty of density estimation. If sufficient prior human knowledge is available, meaningful functions for $\phi_i(\cdot)$ can be defined. However, in line with the goal of not relying on possibly unreliable human knowledge, a better method would be to learn them from training examples, which can be interpreted as a feature extraction problem. One solution to this problem is to find the various $\phi_i(\cdot)$ that maximises the mutual information between $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$, which is described in more detail in [19].

2.3.2 Artificial Augmentation of Training Data

In order to get good performance from the predictor, it is necessary to obtain a large amount of training data. Unfortunately, real-world constraints limit the amount of training data that can be obtained. Here, we propose the method of artificially augmenting the training set to cope with this problem. This method of generating virtual examples has also been used elsewhere for object recognition [31], face analysis [4], mobile robot navigation [32] and pattern detection [38]. Note that this method can be seen as imposing some prior human knowledge into the predictor, similar in concept to what the parametric prediction models discussed in section 1.3.1 do. However, we re-iterate that our prediction framework does not need this where sufficient reliable training data is available. What we present here are only suggestions to cope with situations where such reliable data is not available.

We propose two methods of augmenting the training set. The first method is simply to duplicate the training set and to add noise to the duplicated data. This corresponds to

increasing the uncertainty of each training data point and prevents the density estimation procedure (in our case, the EM algorithm) from over-fitting on the training data. One type of noise to add is uncorrelated Gaussian noise of some fixed variance, σ^2.

The second method of augmentation is more principled and involves domain-specific knowledge about the object state and its dynamics. It works by exploiting known forms of invariance in these characteristics. For example, if the size of the target object is known to vary but the underlying dynamics governing its motion are unchanged, then the training set may be enlarged by generating the same set of dynamics for an object of a different size. Or if the object is known to move at different speeds but in a way that is already captured in the original training data, the training set can again be augmented to reflect this. In these examples, the invariant parameters are the object size and speed respectively. Other invariant parameters such as the symmetry in the dynamics may also be used. This augmentation method can be seen as incorporating prior human knowledge about the tracking environment into the predictor by way of the training data.

At this point, we would like to point out that artificial augmentation is just one method of incorporating prior human knowledge into the predictor. Choosing an appropriate form of $\mathbf{X}$ that accounts for certain invariant parameters (e.g. translation invariance) is another such method. And there could be other analytical ways of accounting for such invariances that are equivalent to what we have discussed here. For the probabilistic predictor, the exact method used is not important as long as it is clear that these methods are meant to deal with situations where insufficient training data is available. Where such data is available, most of these methods are unnecessary.

2.3.3 Estimating the Joint Density Parameters

The parameters of (2.2) can be summarised as $\Theta = \{\mu_k, \Sigma_k, w_k | k = 1, \ldots, K\}$. Given the N_T training examples, $\{\mathbf{Y}_t | t = 1, \ldots, N_T\}$, these parameters can be estimated with

the *maximum-likelihood principle*:

$$\hat{\Theta} = \arg\max_{\Theta} \left(\prod_{t=1}^{N_T} p(\mathbf{Y}_t|\Theta) \right) \tag{2.11}$$

Many methods exist for solving this equation. The exact choice of which is not crucial to our probabilistic prediction model. We have chosen to use the well-known *expectation-maximisation* (EM) algorithm [13], whose application to solving (2.11) was discussed extensively in [35]. Briefly, the algorithm consists of two key steps which are executed iteratively.

The first step is the "E-step", which estimates the probability of the training vector $\mathbf{Y}_t$ belonging to the Gaussian component k. At the i^{th} iteration, it is computed as:

$$h_{k,t}^{(i)} = \frac{w_k^{(i)} g(\mathbf{Y}_t; \mu_k^{(i)}, \Sigma_k^{(i)})}{\sum_{j=1}^{K} w_j^{(i)} g(\mathbf{Y}_t; \mu_j^{(i)}, \Sigma_j^{(i)})} \tag{2.12}$$

This is followed by the "M-step", which generates an estimate of Θ as follows:

$$w_k^{(i+1)} = \frac{\sum_{t=1}^{N_T} h_{k,t}^{(i)}}{\sum_{k=1}^{K} \sum_{t=1}^{N_T} h_{k,t}^{(i)}} \tag{2.13}$$

$$\mu_k^{(i+1)} = \frac{\sum_{t=1}^{N_T} h_{k,t}^{(i)} \mathbf{Y}_t}{\sum_{t=1}^{N_T} h_{k,t}^{(i)}} \tag{2.14}$$

$$\Sigma_k^{(i+1)} = \frac{\sum_{t=1}^{N_T} h_{k,t}^{(i)} (\mathbf{Y}_t - \mu_k^{(i+1)})(\mathbf{Y}_t - \mu_k^{(i+1)})^T}{\sum_{t=1}^{N_T} h_{k,t}^{(i)}} \tag{2.15}$$

The EM algorithm can be shown to converge at a local maximum of the likelihood

function. The quality of this maxima depends on the initial guesses of the parameters. In our implementation, we initialise the algorithm by randomly choosing training vectors as the means, $\mu_k^{(0)}$ for $k = 1 \ldots K$. The covariance matrices, $\Sigma_k^{(0)}$, are initialised to be a random multiple of the identity matrix and the weights were set to $w_k^{(0)} = 1/K$. The number of iterations that the algorithm was run for depended on the experiment; 30 iterations usually produced reasonable performance.

2.3.4 Bootstrapping

Bootstrapping is a procedure whereby a small amount of training data is initially obtained and used for learning, in our case, to estimate the parameters of the predictor. The predictor can then be used with a tracker, though errors are expected since the predictor was poorly trained. These errors are manually corrected and used to expand the training set. The predictor is then trained on this new training set and used with the tracker again. Errors are again collected and corrected and this process repeats until the predictor works well. This strategy has been applied to other areas like pattern detection [38] and mobile robot navigation [32] with good results.

Bootstrapping improves tracking performance by selecting examples that were previously poorly modelled by the predictor. While this alters the statistical properties of the training set so that it is no longer representative of the true distribution, the training set is made more useful to solving the tracking problem.

As a final note, this method of bootstrapping involves a considerable amount of user intervention. The tracking experiment must be run, prediction errors must be collected, corrected and used for retraining. It may also be necessary to repeat this process several times. This is the price of bootstrapping in this manner. However, there is another way of bootstrapping the predictor that requires little user intervention. It may not work as well as the method described here but it is automatic and can be used as an "initial guess" to correct the more obvious problems in the training set before the method described here

is applied. This will help reduce the amount of user intervention needed. A complete description of this automatic bootstrapping method is deferred until section 4.5.1, after we have developed a suitable quantification procedure that is needed by this method.

2.3.5 Initialisation Strategy

To generate a prediction, we sample from (2.6) where the prior vector $\mathcal{X}_t^{(q)}$ is assumed to be known. This is not true for $t \leq q$ where $\mathcal{X}_t^{(q)}$ cannot be obtained from previous prediction states. Hence, there is a need for a method to initialise the predictor. During tracking, it also may be desirable to allow for some small chance of re-initialisation to help the tracker recover from tracking failures.

We propose two strategies for initialisation. The first is to manually initialise the predictor, where the user would provide the information needed to form up the first prior vector. This is a very simple method which may be applicable in some cases. But since it requires user intervention, this solution is not useful as a method to automatically re-initialise the tracker during tracking failures.

The second strategy is to automatically generate the prior by sampling it from some prior probability density function. Fortunately, the probabilistic predictor provides a convenient way of specifying this density: the density is simply the unconditional prior density expressed as follows:

$$p(\mathcal{X}_t^{(q)}) = \sum_{k=1}^{K} w_k \, g(\mathcal{X}_t^{(q)}; p_k, A_k) \tag{2.16}$$

where w_k was previously defined in (2.2) and p_k and A_k were previously defined in (2.5). These parameters have already been estimated while training the predictor so we get to use them here "for free".

If $q = 1$, then the prior can be considered as a prediction at $t = 1$ that can be measured directly against the image data. However for $q \geq 2$, it would probably be more

convenient to sample from (2.2) to obtain the object state (prediction state) needed for the verification process and the prior state needed for predict-verify framework to proceed to the next time step.

The information from the prediction process alone may not be enough to build a good initialisation process. Image information can also be exploited here, possibly by using importance sampling [33], which allows low-level blob information to be combined with higher-level object state information generated by the predictor to produce a reasonable distribution of the object state. This method was used partly as a re-initialisation strategy with the CONDENSATION algorithm in [23] with good results.

2.3.6 Recipe for Using the Predictor

We have discussed many issues regarding the use of the probabilistic prediction model. We now summarise these into a step-by-step guide of how to use the predictor for tracking. This involves two phases, namely a training phase where the predictor learns the dynamics of the target object, and an application phase where the predictor is used in the prediction step of the tracking framework. The following describes these phases in more detail.

Training Phase

§1. The predictor has two free components. They are the form of the joint probability vector, $\mathbf{X}$, and the number of Gaussian components to use to estimate the joint probability density of (2.2), K. The first step is to decide on these parameters. Choosing the form of $\mathbf{X}$ was discussed in section 2.3.1. Though section 4.5.2 suggests a heuristical procedure to choose a good value of K, a common method to do so is through trial-and-error.

§2. Obtain the training data in the form of the joint probability vector decided upon in step §1. Augment the training data set as desired (see section 2.3.2).

§3. Learn the parameters of the prediction model, w_k, μ_k and Σ_k for $k = 1, \ldots, K$ given in (2.2) (see section 2.3.3).

Application Phase

§4. Generate predictions by sampling from (2.6) using the procedure described in section 2.1.3.

Chapter 3

Experiments

We now demonstrate the utility of our prediction model with several tracking experiments. Recall the framework from figure 1.2 that uses prediction and verification to do tracking. We have already discussed prediction in great detail in the previous chapter. To complete the tracker, we now describe the verification process, as well as a conditional probability density propagation technique known as CONDENSATION[22] to glue the two steps together. However, we re-iterate that the probabilistic predictor can be applied to any predict-verify tracking framework, of which CONDENSATION is only one of them. It is also independent of the choice of the verification step, which is a result of its generality.

Sections 3.2 and 3.3 will each present a set of experiments. The first set is on fish tracking and the second is on hands tracking. Our approach to describing these experiments are generally similar and each section will describe the following details:

- The object model. We have intentionally chosen different models for the two experiments to demonstrate the probabilistic predictor's independence of this choice.
- The verification process. As with the object model, we have chosen different verification processes.
- The motivation for the experiment. Each set of experiments is intended to show

certain interesting or useful aspects of the predictor.

- The experiment procedure. A basic procedure for setting up the experiment is described here, including how the training data was gathered, manipulated and used to train the predictor.

- The experiment results. The various results obtained from the experiments are presented and analysed here.

We begin by describing the CONDENSATION algorithm.

3.1 The CONDENSATION Algorithm

The tracking framework shown in figure 1.2 can be formalised as follows. First, the prediction step propagates the conditional density from time t to time $t+1$:

$$p(\mathbf{X}_{t+1}|\mathcal{Z}_t) = \int p(\mathbf{X}_{t+1}|\mathbf{X}_t)p(\mathbf{X}_t|\mathcal{Z}_t)d\mathbf{X}_t \tag{3.1}$$

where $\mathbf{Z}_t$ is the image measurement at time t and $\mathcal{Z}_t = \{\mathbf{Z}_1, \ldots, \mathbf{Z}_t\}$ is the set of all image measurements up to time t. The posterior state density is obtained after image measurements at time $t+1$ are incorporated into the system, giving

$$p(\mathbf{X}_{t+1}|\mathcal{Z}_{t+1}) \propto p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1})p(\mathbf{X}_t|\mathcal{Z}_t). \tag{3.2}$$

This formalism is shown pictorially in figure 3.1. See [22] for more details.

There are many methods of implementing this tracking framework. The Kalman filter is one such method if $p(\mathbf{X}_{t+1}|\mathcal{Z}_{t+1})$ is assumed to be Gaussian. However, such an assumption is often violated in the presence of clutter. The CONDENSATION algorithm [20, 22], on the other hand, does not make such an assumption. It maintains and propagates conditional probability densities using a set of N discrete samples and is capable of handling

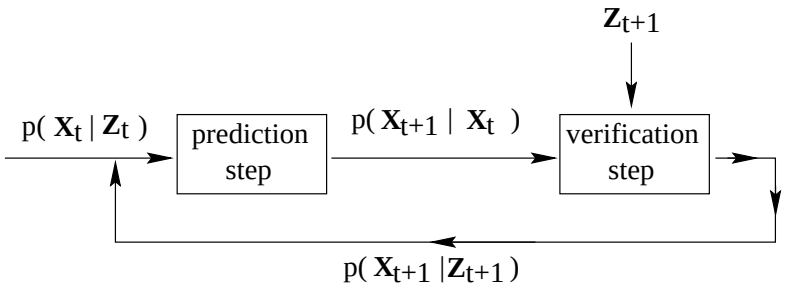


Figure 3.1: Formalised tracking framework

non-Gaussian densities. It is this method that we use in our tracking experiments and the rest of this section will discuss it in more detail.

As mentioned, CONDENSATION maintains a set of N samples approximating the posterior state density, $p(\mathbf{X}|\mathbf{Z})$. Each sample consists of a $(\mathbf{s}_t^{(n)}, \pi_t^{(n)})$ pair where $n = 1, \ldots, N$, $\mathbf{s}_t^{(n)}$ represents the state of the sample at time t and $\pi_t^{(n)}$ represents its corresponding probability. These samples are propagated over time using the prediction and verification steps of (3.1) and (3.2) respectively. Briefly, the algorithm proceeds by iterating the following steps for $t = 0, 1, \ldots$:

1. To construct the sample set at time $t+1$, perform the following for each sample n:

   (a) **Prediction Step** First, randomly select a sample, $\mathbf{s}_t^{(n')}$, from $p(\mathbf{X}_t|\mathbf{Z}_t)$ with probability $\pi_t^{(n')}$. Predict its state at time $t+1$, $\mathbf{s}_{t+1}^{(n)}$, by sampling from $p(\mathbf{X}_{t+1}|\mathbf{X}_t = \mathbf{s}_t^{(n')})$.

   (b) **Verification Step** Compute a goodness value of the prediction $\mathbf{s}_{t+1}^{(n)}$, which we denote with $a_{t+1}^{(n)}$.

2. Normalise the goodness values to obtain the posterior sample probabilities, expressed by $\pi_{t+1}^{(n)} = p(\mathbf{Z}_{t+1}|\mathbf{X}_{t+1} = \mathbf{s}_{t+1}^{(n)})$, as follows: $\pi_{t+1}^{(n)} = a_{t+1}^{(n)} / \sum_{n=1}^{N} a_{t+1}^{(n)}$, We now have $p(\mathbf{X}_{t+1}|\mathbf{Z}_{t+1})$ approximated by the sample set $\{(\mathbf{s}_{t+1}^{(n)}, \pi_{t+1}^{(n)})\}$.

At each time step, statistics about the posterior state density may be computed to provide an output of the tracking. A common statistic to use is the mean of the density,

which can be computed as follows:

$$\bar{s}_t = \sum_{n=1}^{N} \pi_t^{(n)} s_t^{(n)} \tag{3.3}$$

because we have already normalised all the weights $\pi_t^{(n)}$ at each time step to sum to unity. The use of this statistic assumes that the posterior state density is unimodal, which may not be true. However, it is simple enough to compute and gives a reasonable output most of the time, which is why we have used this in all of our experiments.

A few other details remain before using CONDENSATION. First, there needs to be a way to initialise the tracker, specifically the predictor. This can be done using the methods we have suggested in section 2.3.5. We would also need to define an object model to describe the target object as well as a similarity measure for use in the verification step. These two are experiment-specific and will be discussed in due course of discussing the experiments themselves.

3.2 Fish Tracking Experiments

The first experiment that we present is the fish tracking experiment. The reason we have chosen fish as the target object is because it exhibits the complex movements that we believe will suitably demonstrate the ability of our probabilistic predictor. We will elaborate on the reasons for our choice in section 3.2.3.

Our experimental setup consists of a fish in the constrained environment of a fish tank. A camera was set up facing the side of the tank. Fluorescent lamps above the tank shone through a translucent white board to provide a diffuse lighting. To create clutter in some sequences, the back of the fish tank was covered with textured paper. Since our main modes of image information are edges and colour, the textured paper was chosen to exhibit features that mimic the edges and colour of the fish. Figure 3.2 shows a picture

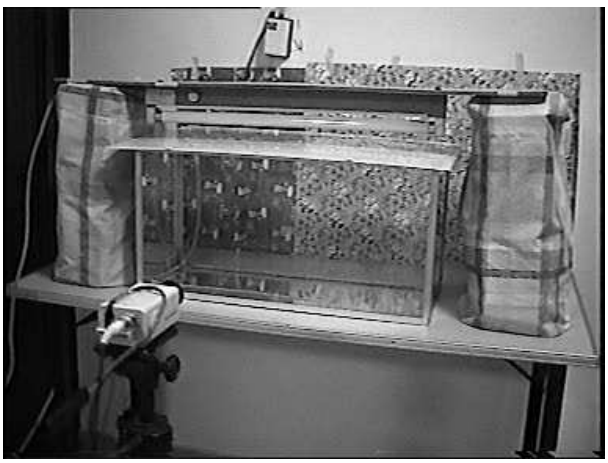


Figure 3.2: Setup of fish experiments

of this setup.

In the next two sections, we complete the tracker by describing the object model used to represent the fish and the verification process used to measure the accuracy of each prediction. We then analyse the reasons for using fish as the target object, and finally present the actual experiment procedure and results.

3.2.1 Object Model

The fish was described by its 2-dimensional outline. This outline was represented by a closed B-spline curve [3] parameterised by a set of N_Q control points $\mathbf{Q}$, where $\mathbf{Q}^T = [\mathbf{Q}_x^T, \mathbf{Q}_y^T]$ and $\mathbf{Q}_x$ are the x-coordinates of the control points and $\mathbf{Q}_y$ are the corresponding y-coordinates. The unconstrained $2N_Q$-dimensional space spanned by these control points, however, may represent shapes other than those that are "fish-like". Hence, we would like to find a suitable linear subspace that does represent these "fish-like" shapes. We do this by defining an $2N_Q \times N_O$ shape matrix $\mathbf{W}$, where $N_O \leq 2N_Q$, that contains the basis vectors of this linear subspace. The resultant valid shapes are then expressed

as:

$$\mathbf{Q} = \mathbf{WO} + \bar{\mathbf{Q}} \tag{3.4}$$

where $\bar{\mathbf{Q}}$ represents the mean fish shape and $\mathbf{O}$ is an $N_O \times 1$ vector parameterising this lower dimensional subspace. In the terminology defined in section 1.1, $\mathbf{O}$ is the object state, which may be the same as the prediction state, $\mathbf{X}_{t+1}$.

$\mathbf{W}$ and $\bar{\mathbf{Q}}$ may be obtained in a number of ways. One way is to manually set them based on analytical derivations [6]. However, this is tedious and may yield poor results for deformable shapes like fishes. So, we have chosen the more effective method of learning them from examples. We do this by applying principal component analysis (PCA) [24] to a set of example fish shapes. This process begins by computing a mean shape, $\bar{\mathbf{Q}}$. For all the fish shapes, the residual deviation from $\bar{\mathbf{Q}}$ is computed and PCA proceeds to produce a set of eigenvectors and eigenvalues spanning this residual space. From these, we select the eigenvectors corresponding to the largest $N_O - 2$ eigenvalues, denoted by $\mathbf{v}_i$ for $i = 1, \ldots, N_O - 2$. These eigenvectors span a lower dimensional subspace that best captures the fish shape deformation [24]. The specification of this representation is almost complete; the remaining problem with it is that it represents translation as part of a shape change. It is usually more effective to be able to represent translation independent of shape. This leads us to complete the representation by forming up the projection matrix as follows:

$$\mathbf{W} = \left[\begin{array}{cc|ccc} \mathbf{1} & \mathbf{0} & & & \\ \mathbf{0} & \mathbf{1} & \mathbf{v}_1 & \cdots & \mathbf{v}_{N_X-2} \end{array}\right] \tag{3.5}$$

where $\mathbf{0}$ and $\mathbf{1}$ are $N_Q \times 1$ vectors of zeros and ones respectively. This completed projection matrix spans what is called the *shape space*. Correspondingly, the first two values of $\mathbf{O}$ are the (x, y) location of the fish shape in an image while the other $N_O - 2$ values are the

coefficients of the eigenvectors. Geometrically, we can then interpret $\mathbf{O}$ as controlling the deviation of the target fish shape from its mean, $\bar{\mathbf{Q}}$.

Our choice of the object model follows closely that which has been commonly used with the CONDENSATION algorithm and we refer the interested reader to [6] for a more extensive description of it. But we also hasten to point out that this choice was heuristically chosen based on what we believe would be reasonable for representing the fish contour and that there are other object models that may work just as well. Neither the probabilistic predictor nor CONDENSATION requires this particular object model to be used.

3.2.2 Verification Process

The input into the verification process is the object state, $\mathbf{X}_{t+1}$, obtained from the prediction state. The aim of this process is to check this hypothesized object state against the image data, denoted by $\mathbf{Z}_{t+1}$, to produce a numerical value of how good the prediction was. To do this, we begin by converting the object state into a closed B-spline curve as described in (3.4) with $\mathbf{O} = \mathbf{X}_{t+1}$. A set of M evenly-spaced discrete points, $\{\alpha_m|\mathrm{m} = 1, \ldots, \mathrm{M}\}$, on this curve is then obtained, where M is set by hand before the tracking experiment begins. At each point, two pieces of information are used in the verification: edge information $Q_e(\cdot)$ and colour information $Q_c(\cdot)$. On top of that, the detection of the eye of the fish $Q_i(\cdot)$ is also used. The overall quality of the prediction is then computed as follows:

$$a_{t+1}^{(n)} = Q_i(\mathbf{O}) \prod_{m=1}^{M} Q_e(\alpha_m) Q_c(\alpha_m). \tag{3.6}$$

Using the following step function,

$$h(x; x_\tau, y_l, y_h) = \begin{cases} y_h & \text{if } x > x_\tau \\ y_l & \text{otherwise,} \end{cases} \tag{3.7}$$

we now discuss each of these components in detail.

The first component that we discuss is the edge metric. Let r and θ denote the magnitude and orientation of the edge at α_m respectively. These are then used to define the edge metric as:

$$Q_e(\alpha_m) = h(r; R, 0.1, 1.0)\ g(\theta; \mu_\theta(\alpha_m), \sigma_\theta) \tag{3.8}$$

where R is a threshold of the magnitude, which is set to $R = 10.0$, and $g(\theta; \mu, \sigma^2)$ is a Gaussian function with mean μ and variance σ^2 which we use to evaluate θ. $\mu_\theta(\alpha_m)$ is the orientation of B-spline curve at α_m and we set $\sigma_\theta = \pi/6$.

In the colour metric component, we make use of Fish's *(sic)* linear discriminant [14], denoted by $\mathbf{F}$, to encourage the pixels in the interior of the B-spline to be "fish-coloured" and the exterior to be "non-fish-coloured". Let $\mathbf{I}_l(\alpha_m)$ be the RGB-vector of the pixel l units perpendicularly away from α_m (negative values indicate the interior of the B-spline) and $L = 4$ be the maximum distance to search. Now using σ_c as a threshold of what is considered "fish-coloured" and what is not, we define the colour metric as

$$Q_c(\alpha_m) = \left[\prod_{l=-L}^{-1} h(\mathbf{F} \cdot \mathbf{I}_l(\alpha_m); \sigma_c, 0.3, 1.0)\right] \left[\prod_{l=1}^{L} (1 - h(\mathbf{F} \cdot \mathbf{I}_l(\alpha_m); \sigma_c, 0.3, 1.0))\right]. \tag{3.9}$$

Finally, we describe the component detecting the eye of the fish. Let $S(\mathbf{O})$ be a set of coordinates indicating possible locations for the centre of the fish-eye and $\mathcal{E} * I(\mathbf{x})$ be a normalised correlation operation between the fish-eye template $\mathcal{E}$ and the image at

position specified by $\mathbf{x}$. The fish-eye detector metric is then defined as

$$Q_i(\mathbf{O}) = \min_{\mathbf{x} \in S(\mathbf{O})} [\mathcal{E} * I(\mathbf{x})] . \tag{3.10}$$

In this experiment, the linear discriminant $\mathbf{F}$, and the fish-eye template $\mathcal{E}$, were both learnt from examples. All other parameters, including $S(\cdot)$, were set experimentally.

3.2.3 Rationale for Using Fish Tracking

There are several reasons why we believe this experiment to be complex and suitable to demonstrate the ability of the probabilistic predictor.

1. One reason for choosing fish as the tracking target was that fish shapes are deformable, and would thus generate complex state dynamics. Visually, we can see that there is a pattern in how the fish moves and deforms, yet this pattern is difficult to capture in any parametric form.

2. Fish movements generate both continuous and discontinuous dynamics. Continuous changes occur when the fish is deforming smoothly as it swims. In situations where the fish turns, its shape may deform quite suddenly leading to discontinuous changes. Such is the form of dynamics that a good predictor should handle.

3. The choice of our object model described earlier showed that we were representing a 3-dimensional fish with only a 2-dimensional contour. This is an inherently ambiguous representation. Furthermore, the fish has flexible dorsal and anal fins which were difficult to model well. It also had two thread-like pelvic fins that were too flexible to be modelled but from time to time, showed up in the image sequences. Even without clutter, these highly deformable parts make it a challenging task for any tracker. Furthermore, the verification process is crude involving thresholds set primitively by trial-and-error. As such, it is easily confused, especially since clutter

was also added into the test sequences. All these factors lead to more reliance on the predictor for successful tracking.

4. In gathering training examples, we were careful to maintain correspondences between the B-spline control points and certain parts of the fish as much as possible. For example, one particular control point always followed the mouth of the fish while another three points always represented its tail. In return for the extra effort put in to establish these point correspondences, we require that a tracker not only recover the shape of the fish but maintain these correspondences during tracking as well. Such correspondences may be lost in cases such as when the fish is twisting, where its mouth or tail may be occluded by its body. In such difficult situations, a tracker with sufficient knowledge about these correspondences is required. In the case of the fish experiments, successful tracking would show how the predictor was able to capture this knowledge and use it during tracking.

5. Finally, the fact that the fish is an autonomous biological object means that it cannot be coerced to move in pre-determined ways. This puts the onus on the predictor to properly understand and faithfully reproduce its dynamics.

3.2.4 Experiment Procedure

We have described all the components needed to complete the predict-verify tracking framework. We now describe how the experiment was carried out as well as the results obtained.

The first step in the fish tracking experiment was to capture video sequences of the fish swimming in a relatively clutter-free environment. These were used for gathering training data for the predictor. We manually recovered the shape and position of the fish in each video frame while making sure that correspondence was maintained between the 13 B-spline control points and various parts of the fish. PCA was applied to these B-spline

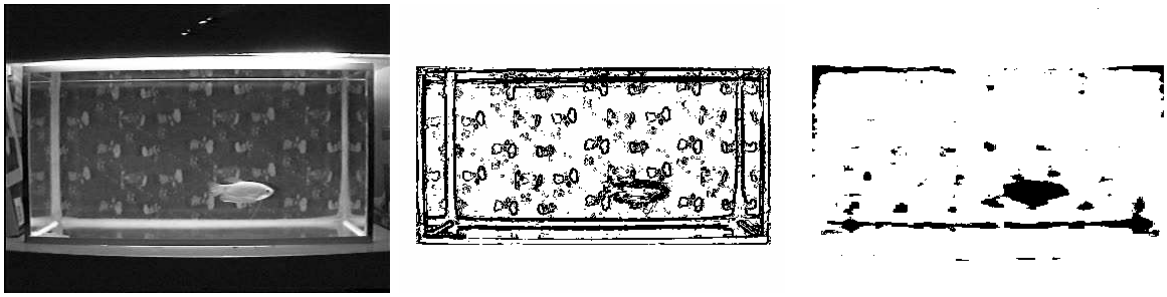
curves and the eigenvectors corresponding to the largest 10 eigenvalues were retained to represent the shape space of the fish. Combined with two translational components as previously described, the shape matrix $\mathbf{W}$ of (3.5) was produced, where $N_O = 12$.

Once $\mathbf{W}$ was constructed, we obtained all the B-spline curves in terms of the lower dimensional parameterisation $\mathbf{O}$. We then used this to form up the data that would be used to train the predictor. The key issue at this point is the form of the joint probability vector, $\mathbf{X}$, previously discussed in section 2.3.1. For the fish experiments, the prediction state comprised two components representing the translational velocity of the fish in the image, while the other components are the coefficients of the 10 eigenvectors used to parameterise the shape space of the fish[1]. This choice reflected some human knowledge about a representation that would intuitively work well: we believed that the translational velocity of the fish was more informative than its absolute position in the image.

Each training vector was then formed up as a 24-dimensional vector, $\mathbf{Y}_t^T = [\mathbf{X}_t^T, \mathbf{X}_{t+1}^T]$ for $t = 1, \ldots, N_T$, where $N_T = 2095$ is the number of training vectors and we have made the first-order Markov assumption of $q = 1$. The training data was then augmented by first incorporating the prior knowledge that the same fish dynamics are possible regardless of the distance of the fish from the camera. This was done by duplicating and resizing the B-splines encoded in the training vectors. These training vectors were further augmented by corrupting duplicated sets of the data with uncorrelated Gaussian noise of $\sigma = 3.0$ pixels. Finally with $K = 35$ components, the joint probability density was estimated using the entire collection of training data.

Due to the limited amount of training data we could obtain by hand, tracking failures often occurred initially, even on the training video sequences. These failures were manually corrected and used to re-train the predictor in the manner described in section 2.3.4.

[1] This means that the prediction state is different from the object state. This difference does not affect the conceptual understanding of our probabilistic predictor and only requires some minor implementation modifications to deal with.



(a) (b) (c)

Figure 3.3: Image information as seen through the eyes of the verification process. (a) shows frame 60 from the test sequence. Black regions in (b) show where the edge magnitude exceeds the threshold R. Black regions in (c) show areas that are classified as being "fish-coloured".

3.2.5 Experiment Results

Clutter was added into the scene and a test sequence was captured. This sequence, called `vhsfish_swim2.Mpeg1` for lack of a more meaningful name, was 568 frames long, spanning about 40 seconds of footage captured at roughly 13 frames per second. The sequence consisted of a variety of fish movements ranging from simple translation to twists and turns.

Figure 3.3 shows three views of the same frame from the test sequence. The first is a gray scale version of the original coloured image. The middle image is approximately how the verification process sees the image in terms of edges. Black regions indicate areas where the edge magnitude exceeds the threshold R. Orientation information, which could not be displayed here, would reduce the amount of noise seen in this view. Finally, the black regions in the third image are those classified as being "fish-coloured" while the rest are "non-fish-coloured". Observe how the edges of the fish tank contributed substantially to the amount of clutter as their silvery-white colour was one of the likely colours of the fish.

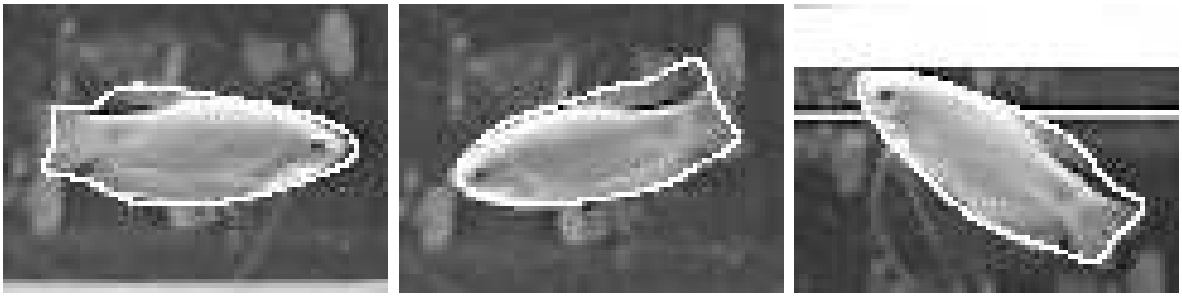
We manually initialised the tracker and set it to run on this sequence with $N = 5000$

samples to represent the posterior state density in (3.2) at each time step. At each time step, the expected value of the posterior state density, expressed in (3.3), was computed to provide the visual output on the progress of the tracker. Figure 3.4 shows this output at various points in the sequence.

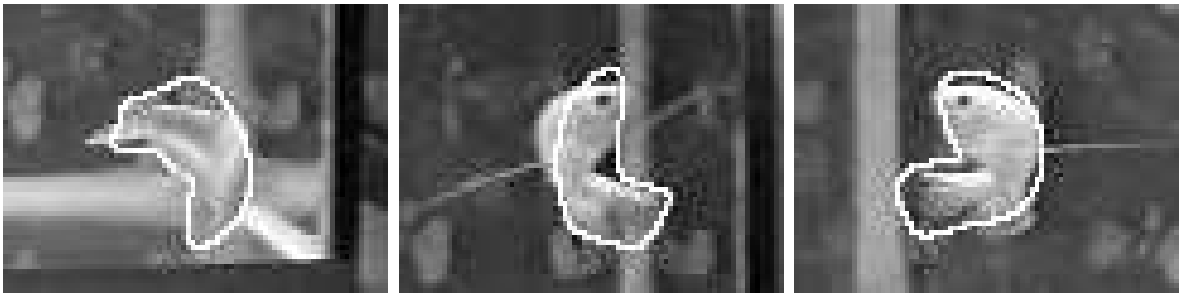
In the fish experiments, the most difficult dynamics to represent were dynamics of the fish turning and twisting. The reason for this is the extremely limited amount of training data we had of these forms of dynamics. They also caused discontinuous changes in the fish shape as well as the loss of point correspondence due to self occlusion. However through the bootstrapping process, the predictor managed to learn such difficult dynamics. It imposed sufficient constraints on the tracker that tracking was maintained, as shown in figure 3.4(b).

Figure 3.4(c) shows some frames where the tracker was temporarily distracted by clutter. This usually occurred near the edges of the tank where, as we have described seen in figure 3.3, the verification process was most vulnerable to clutter. Such situations tend to cause the posterior state density to split into multiple peaks corresponding to different possible true states. Hence the output mechanism of (3.3) would average between the peaks producing an erroneous output. However, we note that tracking lock was recovered as soon as sufficient disambiguating image information was available. This shows that the predictor had guessed the correct state transitions and it was the crude verification process that failed to perform. This is reasonable behaviour and we point out again that the predictor was able to understand fish dynamics well enough that the tracker was not hopelessly lost due to clutter.

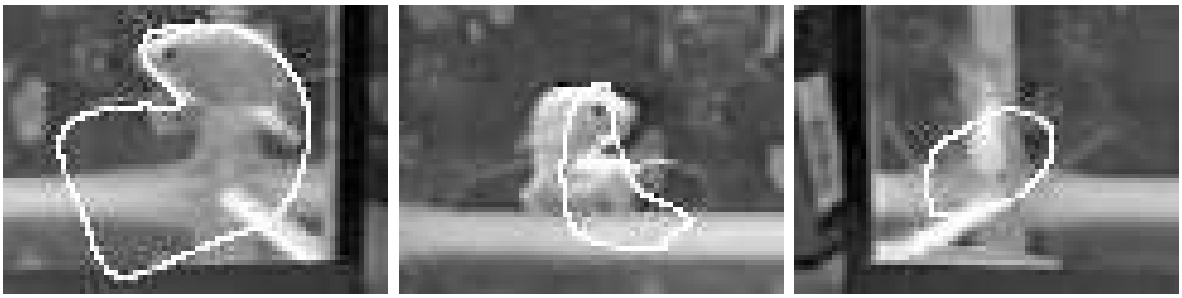
In addition to `vhsfish_swim2.Mpeg1`, we have applied the same tracker to two other video sequences of swimming fish. Their experiment setup were exactly the same as `vhsfish_swim2.Mpeg1` and they both contained the same types of fish movements too. The results produced were similar to those shown in figure 3.4, with their fair share of distraction due to clutter. To quantify the tracking results, we classified the fish contour



(a) Translational motion: frames 60, 305 and 479



(b) Twisting and turning motion: frames 115, 265 and 559



(c) Temporarily distracted by clutter: frames 135, 340 and 543

Figure 3.4: Results of fish tracking. For clarity, the images shown are cropped to a 80×60-pixel region that is roughly centred on the fish. The white outline represents the mean of the posterior state density computed using (3.3) at various time steps.

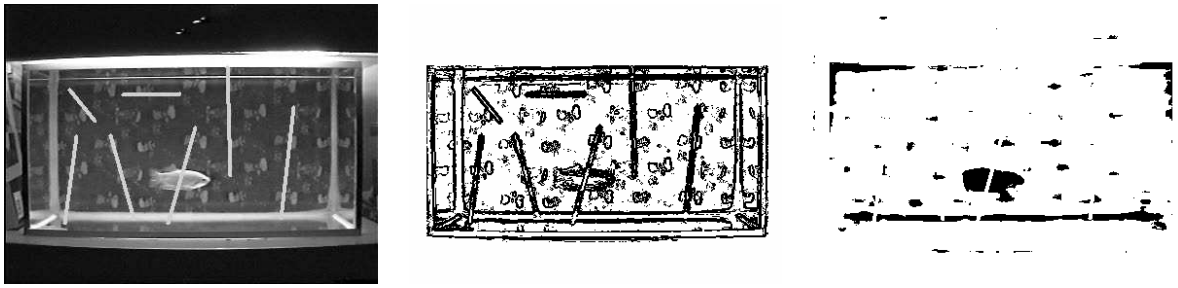
Sequence Name	Sequence Length	Tracking Accuracy
`vhsfish_swim2.Mpeg1`	568 frames (40s)	83%
`vhsfish_swim3.Mpeg1`	480 frames (35s)	73%
`vhsfish_twist.Mpeg1`	807 frames (60s)	86%

Table 3.1: Tracking accuracy of fish experiments, where accuracy is defined as the percentage of "good" contours in the entire video sequence.

of each frame as being "good" or "bad". Since the fish contour is represented by a B-spline curve, a fish contour is considered "good" if more than half of its 13 B-spline control points are less than or equal to 5 pixels (or 25% of the length of the minor axis of elongation of an average-looking fish) away from their manually-determined ideal positions; otherwise, they are classified as "bad". The accuracy of tracking is then defined as a percentage of the number of "good" contours in the entire video sequence. The results of these experiments are summarised in table 3.1 and all three sequences showed reasonably good performance.

We suggest two words of caution when examining these results. Firstly, there should not be an over-emphasis on these results. The fish contour of each frame was computed using (3.3), which as explained earlier, assumes the posterior state density to be unimodal. Clutter in the scene will violate this assumption and possibly lead to a poor visual output that will skew the results. Secondly, tracking accuracy is reflective of all components of the tracker, not just the predictor. This must be taken into consideration when attempting to decide the effectiveness of the predictor. Chapter 4 provides a more detailed analysis of the predictor independent of the other tracking components. In our opinion, the best indicator of tracking effectiveness in a Monte Carlo tracking framework like CONDENSATION is the fact that tracking lock is maintained over the entire video sequence. This was true for all three sequences we have presented here.

As a concluding remark to this experiment, we would like to point out that it can be argued that any prediction model could have predicted these fish dynamics (or any difficult dynamics for that matter) if it had increased its process noise. At the extreme,



(a) (b) (c)

Figure 3.5: Fish tracking with occlusion. (a) Occlusion is added into each image frame in the form of 4-pixel wide opaque strips randomly placed in the image. (b) Occlusion causes more clutter in terms of edges. (c) Black regions show areas that are classified as "fish-coloured". Observe that the occlusion is classified as "non-fish-coloured", as desired.

a purely random predictor could also predict the correct object state if it simply chose to randomly guess every possible state. However, our predictor was able to guess intelligently such that only 5000 samples was needed at each time step to represent the 12-dimensional object state of the fish in all the test sequences.

3.2.6 Experiment Results of Fish Tracking with Occlusion

To make the things more challenging, we added occlusion into the `vhsfish_swim2.Mpeg1` sequence used in the previous section. This was done by artificially drawing opaque strips, each about 4 pixels wide, at certain positions in each image frame. The positions of these strips were chosen randomly, but once chosen, they remained fixed for the entire video sequence. The colour of these strips was also chosen so that they would be treated as being "non-fish-coloured", hence creating the desired effect of occluding the fish. The first image of figure 3.5 shows a frame that has been corrupted as described. The middle image shows that the occlusion has increased the amount of edge clutter in the scene. And the third image shows that these strips are treated as being "non-fish-coloured", as desired.

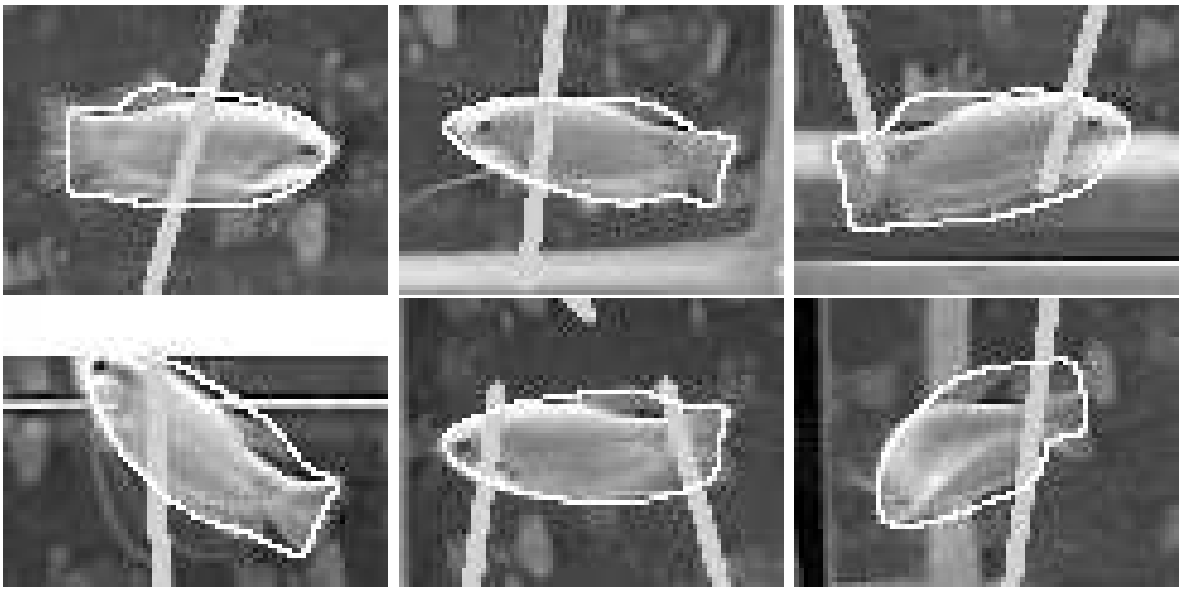


Figure 3.6: Results of fish tracking with occlusion. For clarity, the images shown are cropped to a 80×60-pixel region that is roughly centred on the fish. The white outline represents the mean of the posterior state density computed using (3.3) of the following frames in row major order: 43, 145, 204, 476, 508 and 532.

With $N = 7000$ to cope with the increased difficulty of the tracking environment, the tracker was run on this partially-occluded sequence. The ability of the tracker to deal with occlusion is shown in a series of images in figure 3.6. The tracking accuracy of this occluded sequence is 81%, which is only slightly worse than the 83% accuracy obtained from the original sequence. The important thing is that tracking lock was maintained throughout the entire sequence, and we credit this to the ability to the predictor to properly constrain the prediction state space so that the tracker was not overly confused by the occlusion. However, we also caution that the predictor alone is not a substitute for proper occlusion reasoning. The task of the predictor is ultimately to understand the target dynamics only while the effects of occlusion is a problem that can be more appropriately addressed by a more sophisticated verification process.

Figure 3.7: Setup of hands experiments

3.3 Hands Tracking Experiments

In this experiment, we illustrate the feasibility of using the probabilistic predictor in a real-time environment. In addition, we have used a different object model and verification process to achieve successful tracking results, demonstrating the generality of our predictor.

This experiment involves tracking a human hand. An application of such a tracker is a virtual desktop environment, where a user's hand movements is captured and translated into commands that can be used to manipulate objects in a virtual environment. The experimental setup in figure 3.7 shows a camera mounted on a stand. The camera is constrained to face downwards so as to obtain a relatively clutter free tracking environment.

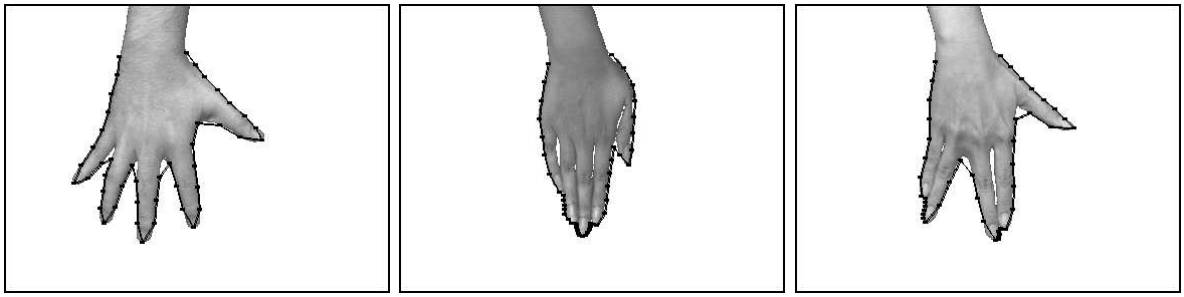
The next two sections will describe the new object model used to represent the hand and a simpler and faster verification process. This is followed by a description of the experiment procedure and results.

3.3.1 Object Model

The hand was described by its 2-dimensional contour, the same as that used for the fish. However, the difference lies in the representation of this contour. While a B-spline was used for the fish experiments, we rely on a point distribution model (PDM) [12] to represent the contour of the hand. A PDM is constructed by first describing the target object with a series of markers. The locations of these markers, e.g. (x, y) image coordinates if these markers are in 2-D image space, then span a high-dimensional space of all possible combinations of marker locations. Principal components analysis (PCA) [24] is then used to obtain a linear subspace approximating the space of valid marker locations.

For our purposes, the PDM was constructed by using a series of discrete points to represent the continuous outline of a hand, as shown in figure 3.8. Some or all of these points may be positioned to correspond to certain physical features. These points are stored as a vector of (x, y) coordinates in 2-D image space. To obtain a linear lower-dimensional parameterisation representing the space of "hand-like" PDMs, PCA [24] was performed on example "hand-like" PDMs and combined with two translational components to obtain a shape matrix, $\mathbf{W}$, similar to that of (3.5). Valid shapes in this subspace are then expressed in the same way as that used for splines in (3.4) except that $\mathbf{Q}$ and $\bar{\mathbf{Q}}$ now contain the (x, y) coordinates of the discrete points of a PDM.

The example hand shapes were obtained from several people to get a set of data representative of human hands in general. However, we instituted two restrictions for the purpose of this experiment. The first was that only the right hand was used. The second restriction was that we allowed only three basic hand shapes: the open palm, the closed palm and the Vulcan hand salute. These are also shown in figure 3.8.



(a) Open palm (b) Closed palm (c) Vulcan hand salute

Figure 3.8: Point distribution model (PDM): black dots represent the discrete points used to represent the continuous hand contour.

3.3.2 Verification Process

Due to the relatively clutter-free environment of this experiment, the verification process we have used is simpler than that used for the fish experiments. Let $\{\alpha_m | m = 1, \ldots, M\}$ be the set of discrete points in the PDM, where M is the number of points used. The quality of the prediction is then computed as follows:

$$a_{t+1}^{(n)} = g\left(\frac{1}{M}\sum_{m=1}^{M} h(Q_p(\alpha_m); 20.0, Q_p(\alpha_m), 100.0); 0.0, 4.0\right) \tag{3.11}$$

where $g(\cdot; 0.0, 4.0)$ is a Gaussian function with mean 0.0 and variance 4.0. $h(\cdot)$ is the step function defined in (3.7) that penalises the prediction heavily should any one of its points exceed the desired threshold. Finally, $Q_p(\alpha_m)$ is the distance of the nearest point on the hand boundary normal to the curve at α_m. Colour information can be used to find the hand boundary or in a simple environment like ours, intensity thresholding was a reasonable substitute.

3.3.3 Experiment Procedure

The first step in this experiment was to gather training data. Several human subjects performed a range of hand movements using the three basic hand shapes. These movements included deforming the hand to change from one hand shape to another as well as rotating the hand about $\pm 45°$ from the "frontal" positions shown in figure 3.8. The discrete points on the hand contour were semi-automatically obtained: the user specified certain control points to correspond to certain parts of the hand while the other points were automatically extracted using image segmentation and are evenly-spaced along the hand contour between corresponding control points. Of a total of 44 points used to discretise the hand contour, seven of them were control points: five of which corresponded to the five finger tips and the remaining two corresponded to the sides of the wrist.

This semi-automatically recovered data was used to construct the PDM shape space as described in section 3.3.1 with $N_O = 10$, where the eigenvectors having the $N_O - 2$ largest eigenvalues were kept to represent the space of hand deformations. To prepare the data for training the predictor, it was first projected into the lower dimensional shape space of hands. We then defined the prediction state to comprise only the coefficients of the $N_O - 2$ eigenvectors; the translational components of the shape space was modelled separately by a constant velocity predictor. This simplifies the experiment, allowing for fewer Gaussian components and a smaller training set to be used. This definition assumes that translational motion of the hand is independent of its contour. Since we have more prior human knowledge about the hands tracking environment than in the case for the fish, we believe this assumption to be valid. Now setting $q = 1$, the training vectors were then constructed as 16-dimensional vectors, $\mathbf{Y}_t^T = [\mathbf{X}_t^T, \mathbf{X}_{t+1}^T]$ for $t = 1, \ldots, N_T$ where $N_T = 1408$ is the number of tracking vectors. This data was then augmented using the prior knowledge of size invariance (same as that for the fish experiments) allowing for similar hand movements at different distances from the camera. With $K = 10$ components, the joint probability density was estimated using the entire collection of

training data.

3.3.4 Experiment Results

The hardware used in this experiment was a 400Mhz Sun Ultra 2 workstation using the SunVideo subsystem to capture and display images from the camera setup shown in figure 3.7. This setup allowed the experiments to run at about 4 frames per second. While this figure is not large enough to claim real-time capability, it is reasonable enough to show its potential.

The tracker was initialised automatically by sampling from the unconditional prior density given in (2.16) and then allowed to run continuously using $N = 50$ samples to represent the posterior state density. Test subjects took turns to perform a range of hand movements in the view of the camera. During the change of test subjects, tracking failure would occur and the same strategy of sampling from the unconditional prior density was used to re-initialise the tracker automatically. Four of the test subjects had previously contributed training data for building the object model and training the predictor. The remaining three were yet unseen by the tracker.

The mean of the posterior state density was used as the visual output of the tracker at each time step. The following figures show images captured during the experiment. The discrete points of the PDM were connected to form a contour of the hand and these are the black outlines seen in the following figures. The black squares indicate the location of the control points. Observe that they correspond closely to the five finger tips and the sides of the wrist, demonstrating that the object state can be used to localise the fingers. In a virtual desktop environment, such information can be used to convert the fingers into pointing devices. In addition, these results also show that the tracker is able to deal with the hands of different people, which can partly be attributed to a good predictor.

Figure 3.9 shows the three basic hand shapes at roughly $-45°$, $0°$ and $+45°$ degrees of rotation. Figure 3.10 shows larger versions of the same basic hand shapes. These larger

hand shapes were not part of the original training data and were obtained only through artificial augmentation. The ability of the tracker to handle these shapes attest to the usefulness of appropriate generation of virtual examples for use in training the predictor.

Even though only $N = 50$ samples were used to represent the posterior state density, figures 3.11 and 3.12 show surprisingly robust tracking performance. The miscellaneous objects in the background and foreground, some of which are hands, easily distracted the crude verification process we used. However, under the constraint that the hand movements are slow enough (how slow it should be is dependent on the frame rate), the predictor was able to suitably constrain the search of the object state space such that tracking proceeded successfully. These figures also show roughly what the verification process saw; post-processing of these images for presentation purposes did modify them slightly.

Finally, to quantify the results of this experiment, we captured a separate video sequence called `fingers_seq2.Mpeg1`. This sequence was 921 frames long and was captured at about 4Mhz to simulate the environment of its online counterpart. All the types of movements used to train the predictor were present in this sequence, including transitions between the three basic hand shapes, rotation and change in hand size. Robustness to clutter and occlusion was also tested later in this sequence. The tracker was run offline on this sequence using $N = 200$ samples and a simple skin colour detector was used to find the hand boundary in the verification step. At each frame, we used the mean of the posterior state density as the visual output. We consider tracking to have succeeded if the control points corresponding to the five finger tips are located less than or equal to 10 pixels from their designated positions. Since the size of the hand in the image is at least twice the size of the fish in the previous experiment, the choice of 10 pixels as the threshold allows for an approximate comparison between the results of the two experiments. The tracking accuracy for this sequence, defined as the percentage of successfully tracked frames to the total number of frames, is 81%.