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

List of Figures

List of Tables

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.

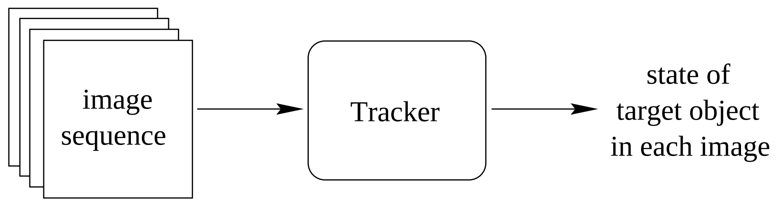


Figure 1.1: The task of tracking

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.

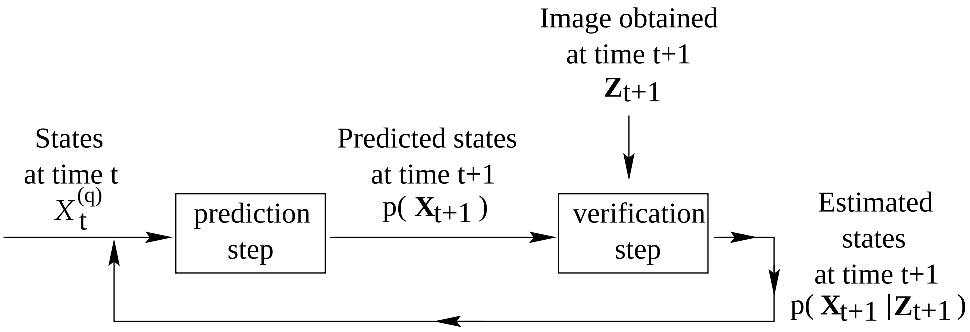


Figure 1.2: Tracking Framework

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 dynamics 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 $k = 1 \dots K$. 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

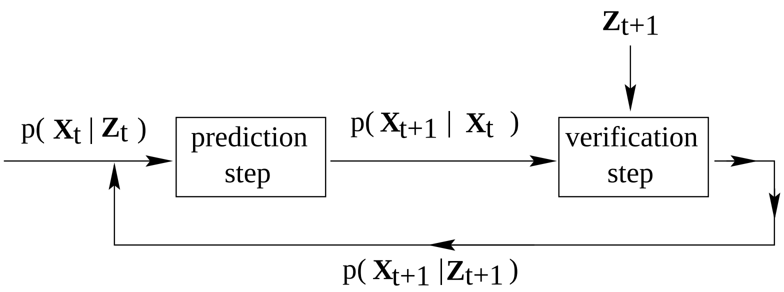


Figure 3.1: Formalised tracking framework

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

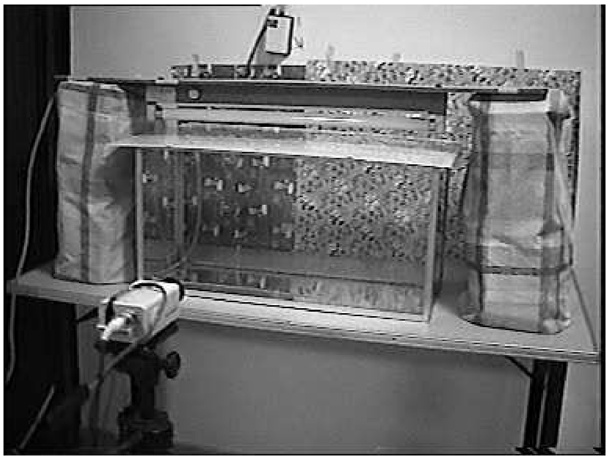


Figure 3.2: Setup of fish experiments

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:

$$\mathbf{Q} = \mathbf{W}\mathbf{O} + \bar{\mathbf{Q}} \tag{3.4}$$

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:

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

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

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

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.

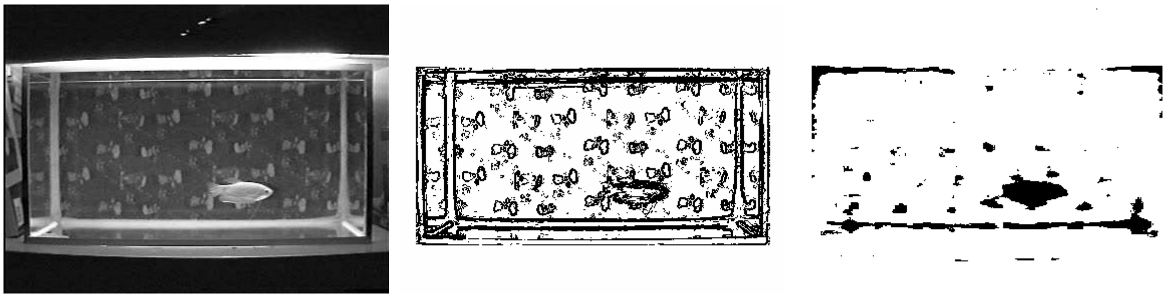
The first step in the fish tracking experiment was to capture video sequences of the fish swimming in a relatively clutter-free environment. These were used for gathering training data for the predictor. We manually recovered the shape and position of the fish in each video frame while making sure that 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.

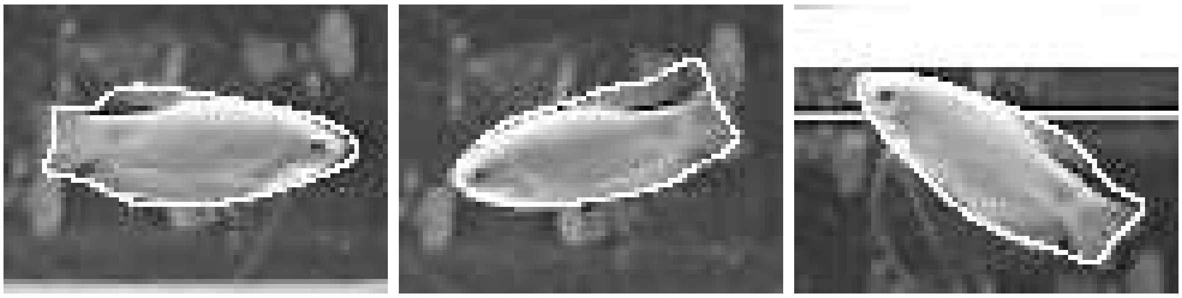
We manually initialised the tracker and set it to run on this sequence with $N = 5000$

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).

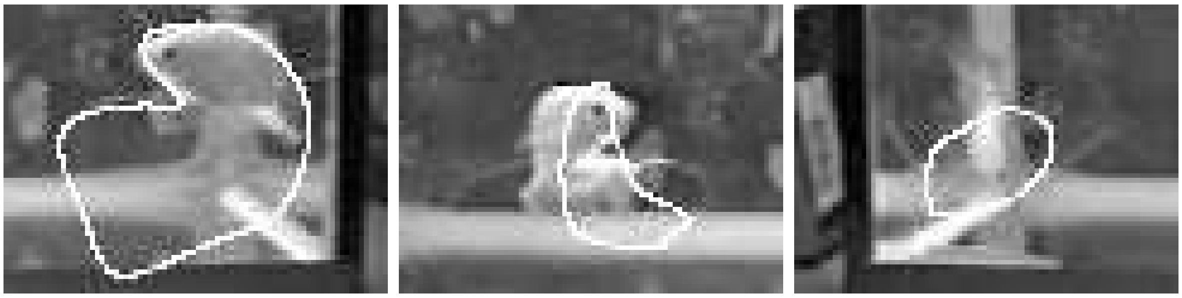
Figure 3.4(c) shows some frames where the tracker was temporarily distracted by clutter. This usually occurred near the edges of the tank where, as we have described seen in figure 3.3, the verification process was most vulnerable to clutter. Such situations tend to cause the posterior state density 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.

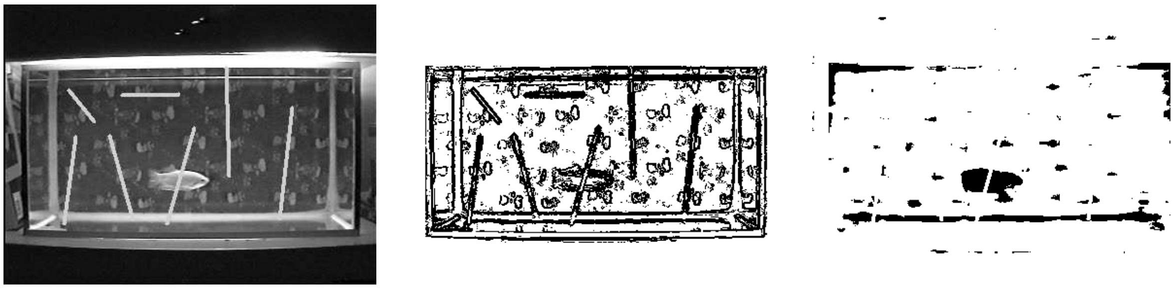
Sequence Name	Sequence Length	Tracking Accuracy
`vhsfish_swim2.Mpeg1`	568 frames (40s)	83%
`vhsfish_swim3.Mpeg1`	480 frames (35s)	73%
`vhsfish_twist.Mpeg1`	807 frames (60s)	86%

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.

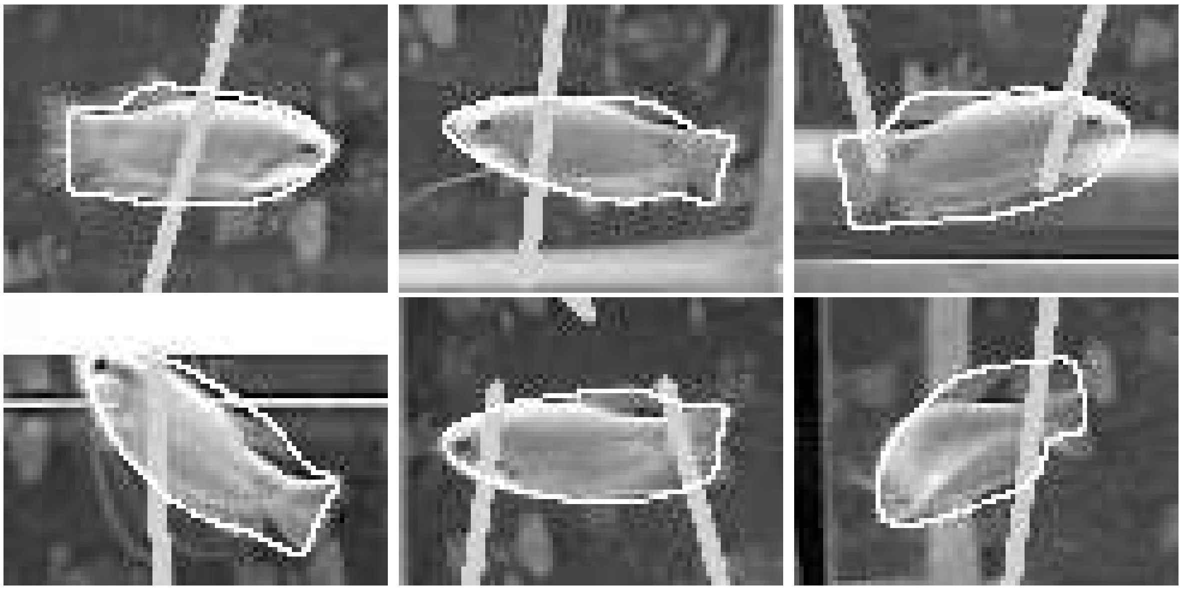


Figure 3.6: Results of fish tracking with occlusion. For clarity, the images shown are cropped to a 80×60-pixel region that is roughly centred on the fish. The white outline represents the mean of the posterior state density computed using (3.3) of the following frames in row major order: 43, 145, 204,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

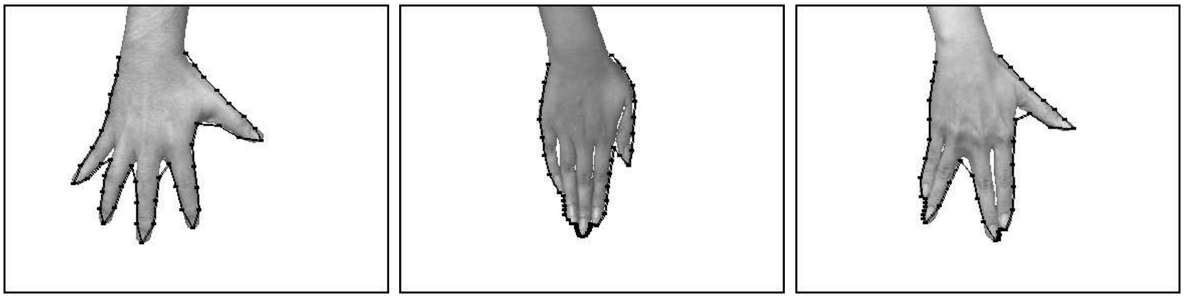
The next two sections will describe the new object model used to represent the hand and a simpler and faster verification process. This is followed by a description of the experiment procedure and results.

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:

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

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.

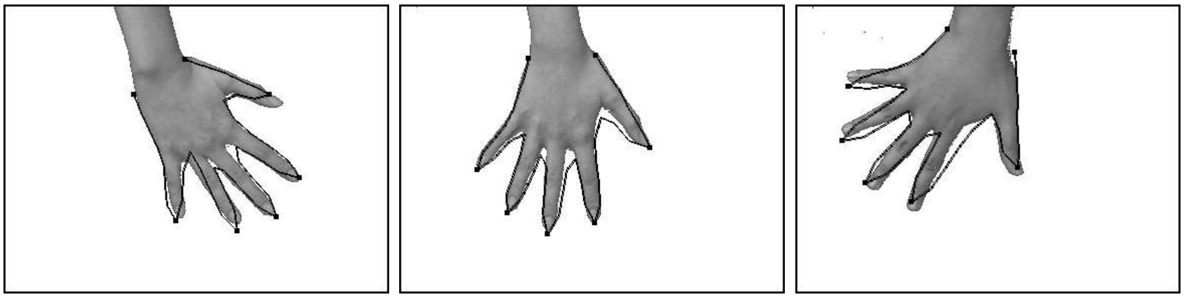
The tracker was initialised automatically by sampling from the unconditional prior density given in (2.16) and then allowed to run continuously using $N = 50$ samples to represent the posterior state density. Test subjects took turns to perform a range of hand movements in the view of the camera. During 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.

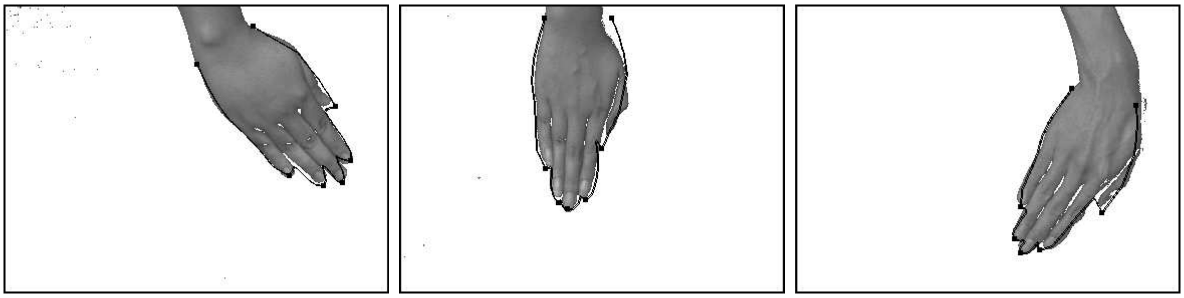
Figure 3.9 shows the three basic hand shapes at roughly $-45°$, $0°$ and $+45°$ degrees of rotation. Figure 3.10 shows larger versions of the same basic hand shapes. These larger hand shapes were not part of the original training data and were obtained only through artificial augmentation. The ability 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.

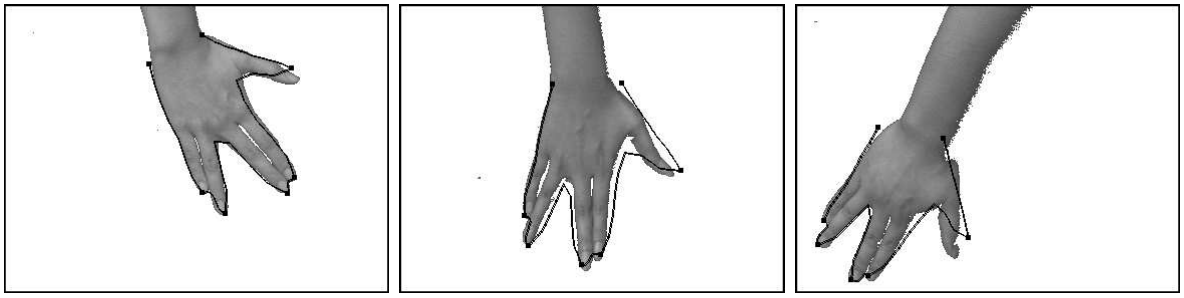
Finally, to quantify the results of this experiment, we captured a separate video sequence called `fingers_seq2.Mpeg1`. This sequence was 921 frames long and was captured at about 4Mhz to simulate the environment of its online counterpart. All the types of movements used to train the predictor were present 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%.



(a) Open palm



(b) Closed palm



(c) Vulcan hand salute

Figure 3.9: Results of hands tracking. Some points to note about these results are: (1) they show the three basic hand shapes, (2) they show rotations of about $\pm 45°$ from the vertical, and (3) these hands came from four different test subjects.

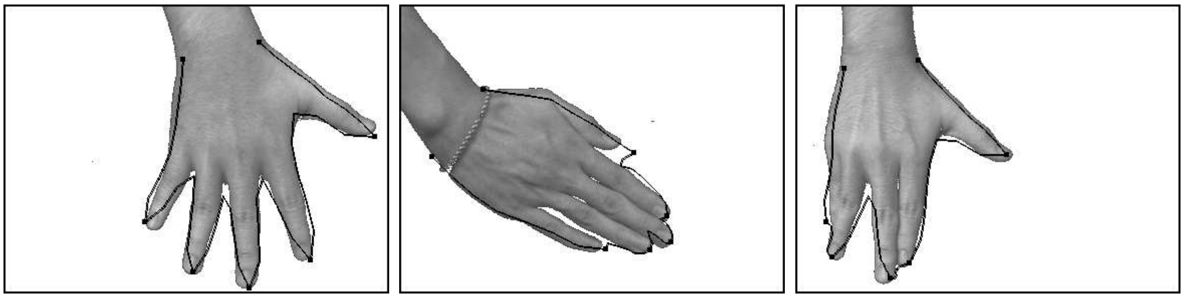


Figure 3.10: Results of hands tracking. These large hand contours show the ability of the predictor to handle change in hand size.

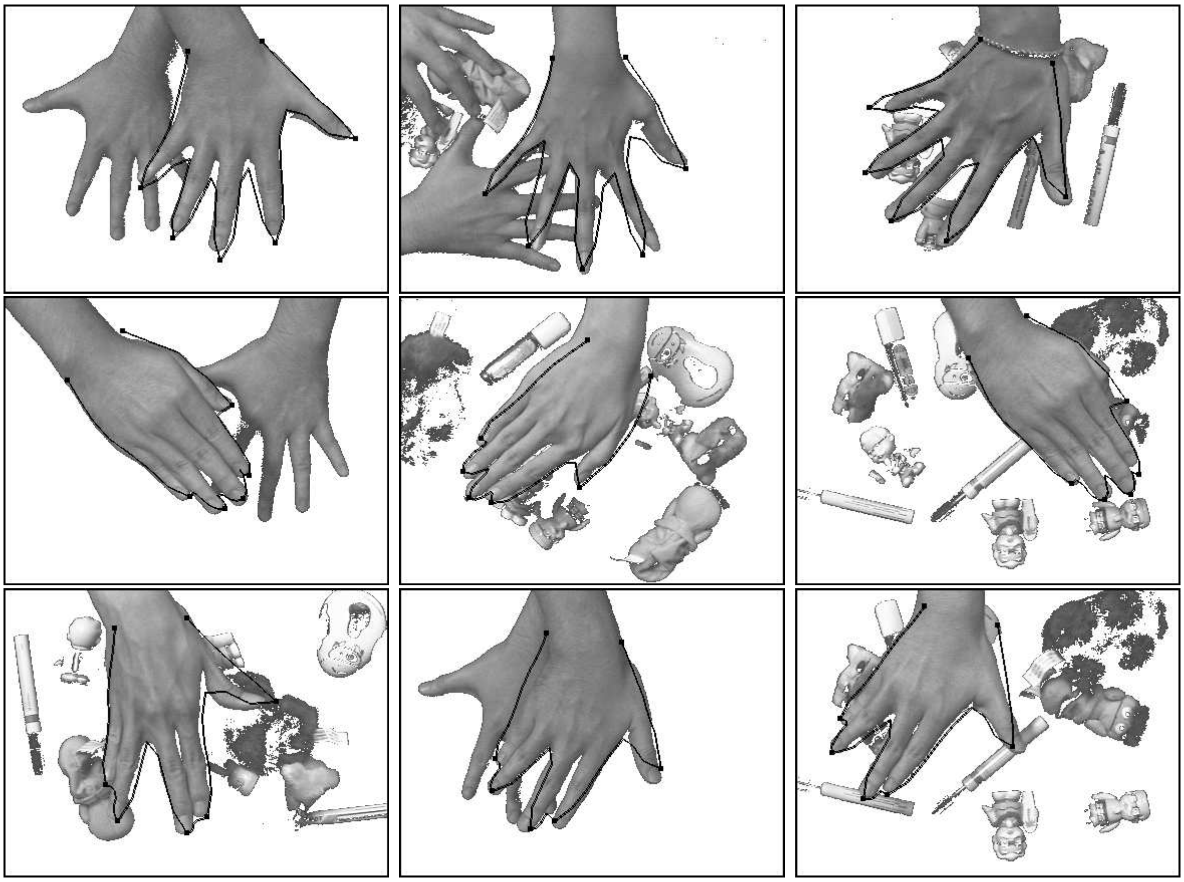


Figure 3.11: Results of hands tracking with clutter.

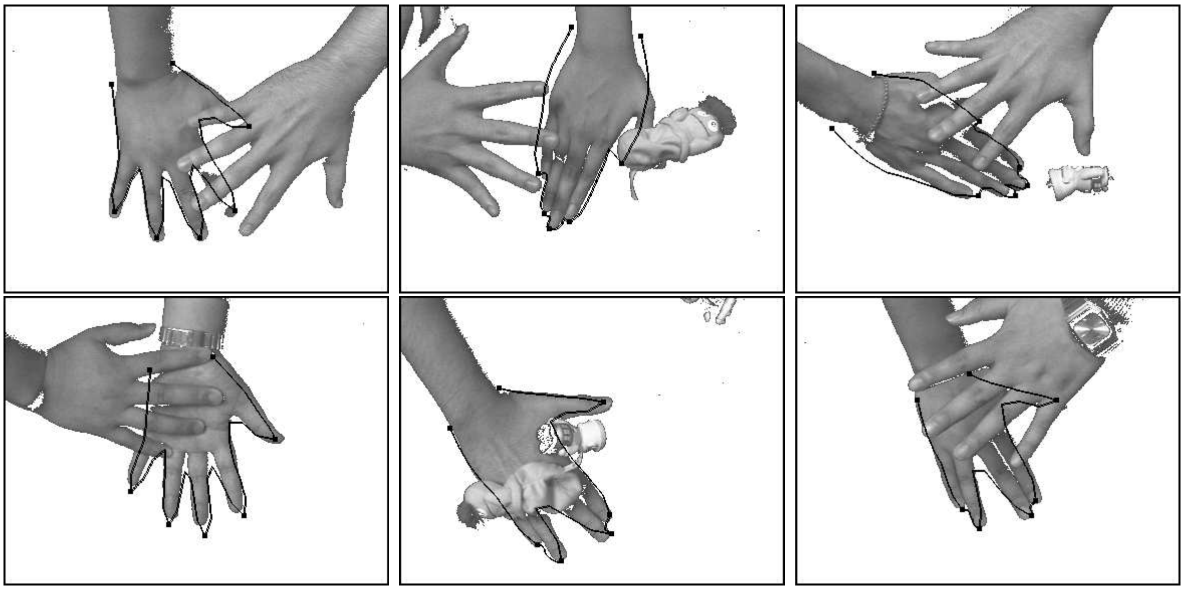


Figure 3.12: Results of hands tracking with occlusion.

Alternatively, since the experiment involved mainly tracking a complete hand, we can define tracking to have succeeded if the output hand shape (which includes rotation and size information) matched the one intended in the image frame. Some of the images shown in the previous four figures would have failed the former criterion for tracking success due to the failure of the object model to represent tiny variations in each person's hands. As such, finger tip localisation would not have been accurate. However, all of them would have passed the revised criterion, which only requires that the hand shape information be accurate. Using this criterion, we can then define the tracking accuracy to be the percentage of successfully tracked frames to the total number of frames, which turns out to be 91%. Regardless of the definition of tracking accuracy, the tracker recovered from all of the minor mistracks and tracking proceeded successfully till the end of the sequence.

Chapter 4

Analysis and Tuning of the Predictor

In the previous chapter, we have shown the effectiveness of the probabilistic predictor in a number of realistic tracking experiments. However, robust tracking results may come from other aspects of the tracking framework, such as the object model or the verification process. These aspects, however minimally effective we have made them in the previous chapter, would affect our ability to accurately quantify the effectiveness of predictor itself.

In this chapter, we present a methodology of quantifying the performance of the predictor without the use of a tracker and its associated components. A new experiment is presented in section 4.2, along with an analysis of the motivation for the design of this experiment. Section 4.3 presents and interprets the results of these experiments. A running time analysis of the predictor is also presented in section 4.4. Finally in section 4.5, we use the insights gained from the preceding analyses to suggest several methods of tuning the predictor for better performance.

4.1 Methodology of Analysing the Predictor

4.1.1 Parameters of Interest

There are two parameters that are of interest to the predictor. The first is the number of Gaussian components, K, used in (2.2). K controls the complexity of the prediction model. It may be tempting to think that a large K enables the predictor to model the underlying dynamics better. This is only partially true as, among other problems, increasing K also increases the problem of over-fitting on the training data. This in turn argues for using a smaller value of K. In this chapter, we investigate how K affects the modelling of dynamics and discuss the various issues to be considered when choosing a value of K. Ideally, we would like to choose a value of K that matches exactly the modelled dynamics. But since it is not always possible to know the ideal value of K, we show that K is relatively insensitive to this choice as long as it is large enough to approximate the true probability distribution and suitable precautions are taken to prevent over-fitting.

The second parameter of interest is N, the number of discrete samples used to approximate the posterior state density. Strictly speaking, N is a parameter of the tracking framework (in our case, the CONDENSATION algorithm) and not the predictor. Hence its setting does not affect the operation of the predictor. However a good predictor should reduce the size of the object state space and N provides our only measurable quantification of this size. Intuitively, we would like to see a tracker using the probabilistic predictor track successfully with a smaller N than one using a less effective predictor. We conduct experiments comparing the probabilistic predictor against the constant velocity prediction model to demonstrate the effects of having a good predictor.

4.1.2 Quantifying the Performance of the Predictor

Using notation similar to that used to describe the CONDENSATION algorithm in section 3.1, we present the quantification procedure we have developed to measure the predictor's performance. For some pre-determined value of K and N, it proceeds as follows:

1. Obtain data for the training set and a test set. Each data vector in these sets is made up of two components as shown in (2.3), namely a prior component $\mathcal{X}_t^{(q)}$ and a prediction component $\mathbf{X}_{t+1}$.

2. Perform augmentation on the training set. Train the predictor using the augmented training set with K Gaussian components.

3. Use the test set to evaluate the predictor. To do so, repeat the following steps for each data vector in the test set:

   (a) Generate N predictions using the prior component of the test vector. Denote these predictions as $\mathbf{s}_{t+1}^{(n)}$ for $n = 1, \ldots, N$.

   (b) Compute the Euclidean distance from each of these predictions to the true prediction, $\mathbf{X}_{t+1}$. The distance measure for the n^{th} measurement is expressed as:

$$D^{(n)}(\mathbf{X}_{t+1}) = ||\mathbf{s}_{t+1}^{(n)} - \mathbf{X}_{t+1}|| \tag{4.1}$$

   where $||\cdot||$ denotes the Euclidean distance measure. Each of these measurements represents the error of the prediction and is recorded for later analysis.

We can extract a number of statistics from the measurements collected in step 3b, e.g. the minimum, maximum and average distances as well as the standard deviation about the average distance. The aim is to use these statistics to develop a metric that will numerically quantify the performance of the predictor. To this end, we propose using the minimum distance statistic as the metric. This means that for each data vector in the test set, we first execute step 3a to obtain N predictions that approximate the prediction state density generated by the predictor. The minimum distance from these predictions to the true prediction is then computed as:

$$D_{min}(\mathbf{X}_{t+1}) = \min\left(D^{(1)}(\mathbf{X}_{t+1}), \ldots, D^{(N)}(\mathbf{X}_{t+1})\right). \tag{4.2}$$

This distance represents how well the predictor approximates the true prediction state density for one particular test vector. The same procedure is applied over the entire test set to obtain the overall predictor performance.

Among the various statistics that we have mentioned, we believe that the minimum distance statistic provides the best measure of predictor performance. The average distance statistic might also be a suitable choice if the prediction state density is unimodal. But since the probabilistic predictor claims to handle multimodal prediction state densities, and we have intentionally made our experiments exhibit this multimodality, the average distance statistic would be a poor judge of how well the predictor has modelled an object's dynamics. On the other hand, the minimum distance statistic implies that as long as one prediction is close to the true prediction, we consider the predictor as performing well. It could be argued that due to the stochastic nature of sampling from the prediction state density, the minimum distance statistic might "get lucky" to make one good prediction when in reality, the prediction state density modelled by the probabilistic predictor is far from the true one. However, the chances of this happening are very small, especially since the experiments were repeated many times. Hence we regard the minimum distance statistic to be reasonably reliable[1].

[1] An examiner of this thesis has questioned our use of the minimum distance statistic, saying essentially that it is too easy to "get lucky" and hence, give an over-optimistic evaluation of the predictor. He has suggested using the average distance statistic instead. In appendix A.4, we elaborate on our reasoning

In addition to the minimum distance statistic, we propose an *average minimum distance* statistic, $\bar{D}$, and the standard deviation, $\sigma_{\bar{D}}$, about this average. Mathematically, these are expressed respectively as:

$$\bar{D} = \frac{1}{N_T} \sum_{t=1}^{N_T} D_{min}(\mathbf{X}_{t+1}) \quad \text{and} \quad \sigma_{\bar{D}}^2 = \frac{1}{N_T} \sum_{t=1}^{N_T} \left[D_{min}(\mathbf{X}_{t+1}) - \bar{D} \right]^2 . \tag{4.3}$$

These provide an overall measure of how well the predictor has performed for a particular setting of K and N.

4.2 The "CHIME DIVA" Experiment

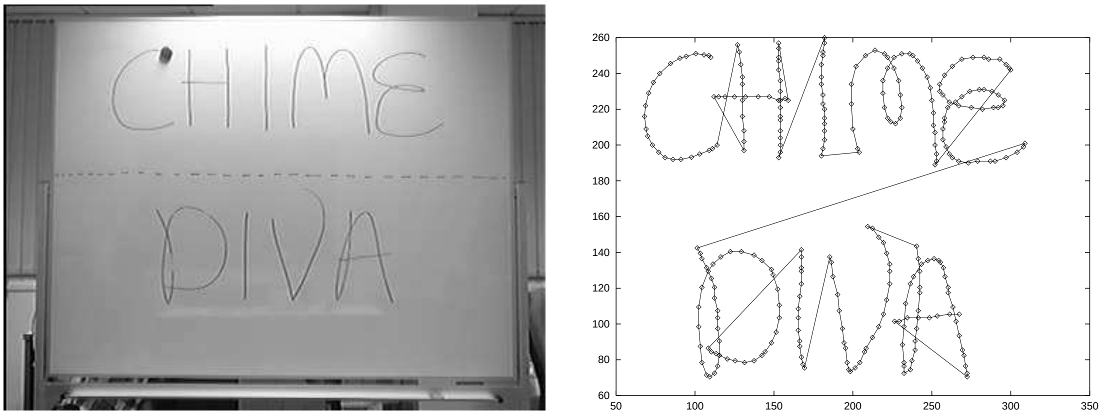
The "CHIME DIVA" sequence is the one we will use in the quantification analyses. It was generated by following the writing motion of the words "CHIME DIVA" on a white-board. We first describe the experiment setup and then discuss the reasons why this sequence was chosen.

4.2.1 Experiment Setup

Figure 4.1(a) shows the outlines of the words "CHIME DIVA" written on a white-board. This outline was traced with a coloured marker to simulate a person writing these words on the white-board with a pen. The resulting motion was captured as a video sequence. Colour segmentation was then used to recover the (x, y) image coordinates of the marker in each video frame. Several such sequences were captured by repeating the tracing process using the same outlines. Three of these repetitions were used for the training set and one was used for the test set.

Each image coordinate obtained can be classified as being in a "pen-down" or "pen-

as to why the average distance statistic is in fact a poor choice. We will also further explain our choice of the minimum distance statistic.



(a) (b)

Figure 4.1: "CHIME DIVA" experiment setup. (a) shows a white-board with an outline of the words. (b) shows a plot of the path traced out by the marker while it was in the "pen-down" position.

up" position corresponding to when the simulated pen is supposed to be in contact with the white-board and when it is not. For example, the letter "H" consists of 3 pairs of "pen-down" and "pen-up" position changes corresponding to the 3 straight lines that form the letter while the letter "C" can be drawn with only 1 such pair. Where the simulated pen was not supposed to be in contact with the white-board (i.e. "pen-up" position), e.g. when it was between letters, the (x, y) image coordinates of these frames were removed from the data set. The resulting data set consisted only of coordinates classified as being in the "pen-down" position. A plot of one such data set is shown in figure 4.1(b). Each data point shows the position of the marker in successive video frames. Observe that where the simulated pen was not in contact with the white-board, no data points are plotted.

Finally, the state of the marker at time t was chosen to be its (x, y) image coordinates, i.e. $\mathbf{X}_t = [x_t, y_t]^T$. And the joint vector was formed using the image coordinates of the marker at two consecutive time steps, expressed as $\mathbf{X}^T = [{\mathbf{X}_t}^T, {\mathbf{X}_{t+1}}^T]$, where we have set $q = 1$.

4.2.2 Analysis of Experiment Setup

The first notable point about this setup is that the resulting dynamics is complex, consisting of both continuous and discontinuous changes. The smooth writing motion corresponds to a continuous change in state space while discontinuous changes occur due to the "pen-up" coordinates being removed. This results in a jump between two "pen-down" coordinates, such as when the pen is moved between letters. The largest discontinuous change occurs between the end of the letter "E" and the beginning of the letter "D".

The next point to note is that the way the joint vector is formed results in multimodal prediction probability densities. This is caused by the writing motion of some letters, e.g. "M" and "A", having criss-crossing paths. Hence given the marker's position at time t, there is more than one possible marker position at time $t + 1$. Such multimodality may be removed if more time steps are included in the prior, i.e. setting $q \geq 2$. However, this was not done so as to demonstrate the predictor's ability to handle multimodal dynamics. Increasing q also increases the dimensionality of the state transition space which would make learning more difficult. For example in the fish experiments, the prior vector with $q = 1$ was 12-dimensional. Increasing q would quickly lead to a large state transition space that makes learning almost impossible due to the limited amount of training data.

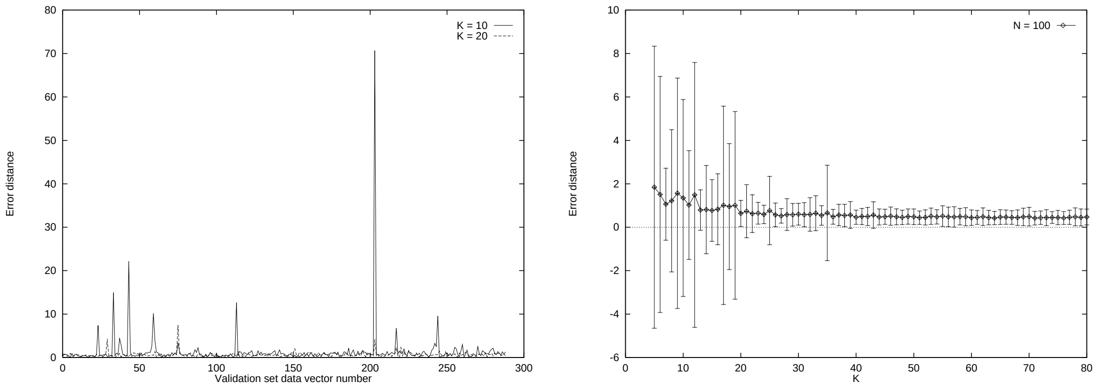
Since the writing motion in this setup is relatively simple, it could actually have been generated artificially by some drawing application. Some white noise could also be added to simulate jitter in the writing motion. However, the data collected from this process would be too synthetic, even with noise, since the underlying writing motion is deterministic. By capturing the writing motion the way we have described, we introduce real-world uncertainties into the training and test data. For example, the speed at which the words were traced vary. This generates data points that were sometimes closer to each other and sometimes further apart. The path traced also varied because the marker did not always follow the written words perfectly. These sort of jitter could not be added by any artificial augmentation procedure and would serve to make the "CHIME DIVA" sequence more realistic a test of the predictor.

Finally, we discuss the issue of using the “CHIME DIVA” sequence to analyse the predictor rather than simply running the quantification procedure on the fish experiments. After all, the fish experiments would provide a much more realistic test for the predictor compared to the relatively more controlled (and hence possibly more artificial) writing motion of the “CHIME DIVA” sequence. However unlike the “CHIME DIVA” sequence, it is difficult to ensure that the fish sequences exhibited all the types of dynamics that we wish to include in the test. Even if that was possible, and if the quantification procedure indicated that the predictor was not learning some dynamics well, it would be difficult to identify the exact type of dynamics that was causing it to fail. This is due to the complex object state used to describe the fish. In contrast, the simple image coordinates representation of the “CHIME DIVA” sequence make the above analysis easy. Lastly, and probably most importantly, the use of only the image coordinates in the state implies that the unit of the Euclidean distance measure of (4.1) is pixels, leading to a clear interpretation of the numerical error distances produced. This is not so for the fish sequences and we would not be able to properly quantify how small the distance should be in order to declare that the predictor is performing well.

4.3 Experiment Results

4.3.1 Effects of Varying K

In the previous few sections, we have discussed how to obtain the training and test sets. We have also discussed the procedure and the metric which will be used to analyse the predictor. We now execute the quantification procedure in section 4.1.2 using $N = 100$ for various values of K, specifically $K = 5, 6, \ldots, 80$. Uncorrelated Gaussian noise of standard deviation $\sigma = 2$ pixels was added to the data points to aid in generalisation and the predictor was trained. Figure 4.2 shows the results of this experiment, which we will



(a) (b)

Figure 4.2: "CHIME DIVA" experiment results: effects of varying K with fixed $N = 100$. (a) shows the minimum distance statistic computed over N predictions for each test vector. (b) shows the average minimum distance with error bars. See text for a more detailed description.

now discuss in more depth.

Figure 4.2(a) is a plot of the minimum distance statistic (4.2) for each test vector. For clarity, we only show the results for $K = 10$ and $K = 20$. Low values in the plot indicate that at least one prediction generated by the predictor was close to the true value. Conversely, large values that show up in the plot as spikes indicate that none of the N predictions generated lie close to the true value. These spikes occurred only for $K = 10$ and indicate that the type of dynamics represented by those test vectors were poorly modelled by the predictor. Checking these test vectors against the video sequence, we find that they tend to correspond to times when discontinuous dynamics occur. Discontinuous dynamics are usually harder to learn since they occur relatively infrequently in the training set compared to continuous dynamics. Furthermore, continuous dynamics, especially in the domain of image coordinates, are also easier to represent. In fact, we have previously mentioned in section 1.3.1 that many blob trackers have been successfully implemented with only simplistic predictors, testifying to how easy it is to represent such dynamics. So we consider it reasonable for the probabilistic predictor to favour modelling continuous dynamics when K, which controls the complexity of the predictor, is small. The plot for $K = 20$ shows that when K is large, all types of dynamics were well-modelled.

For each value of K that we ran the quantification procedure with (including those shown in figure 4.2(a)), we can use (4.3) to summarise the results into a succinct value: $\bar{D} \pm \sigma_{\bar{D}}$. This shows the average minimum distance and its standard deviation for the entire test set for each value of K. They are plotted in figure 4.2(b) with error bars representing $\sigma_{\bar{D}}$. Spikes in figure 4.2(a) translated into larger errors in the graph in figure 4.2(b). While the average minimum error may remain low even in the presence of spikes, the standard deviation of the minimum distances will be large and this is reflected by the long error bars. Since spikes indicate poorly represented dynamics, the occurrence of such dynamics in a tracking environment would likely result in tracking failure. So it is important to choose K carefully such that all types of dynamics are modelled well. Hence what we would like to see, in addition to a low average minimum error, are short error bars as K increases. And as figure 4.2(b) shows, this is indeed the case.

The use of image coordinates as the prediction state allows us to interpret the error distance measure in terms of pixels. For example, the error at $K = 22$ is 0.63 ± 0.87 pixels. Since image coordinates are quantised to the nearest pixel, it is reasonable to say that any error smaller than 1 pixel is negligible. Hence we can safely say that the complexity of the probabilistic predictor at, among other values of K, $K = 22$ is sufficient to represent the writing dynamics. The predictor is thus said to be performing well at this value of K.

Figure 4.2(b) also shows that the performance of the predictor is relatively insensitive to the exact choice of K once there are enough Gaussian components to approximate the complexity of the dynamics. For $K \geq 20$, the predictor maintains its good performance. This is important because for complex dynamics like fish movements, it is difficult to determine analytically how to set K. The empirical results presented here show that this method of modelling dynamics allows us some leeway in the choice of K without severely affecting performance.

The error bars in the plot in figure 4.2(b) also show that for $K = 21$, $K = 25$ and $K = 35$, there was a sudden increase in the prediction error. The cause of this increase is not a flaw in the predictor but merely an indication of the local nature of the learning algorithm. In iterative learning algorithms like EM, it is important to begin with a good guess of the ideal solution. The algorithm will then find the local optima closest to the initial guess. If the initial guess is bad, then the algorithm will likely converge to a poor solution. We attribute the poor results for these two values of K to bad initial guesses. In our implementation, the initial guess was generated by randomly selecting data points as the Gaussian centres with the covariances being randomly generated. Hence by re-running the learning procedure, we can re-initialise the initial Gaussian parameters and hopefully converge to a better solution.

Finally, we observe in figure 4.2(b) that setting K to too large a value no longer improves performance. This is reasonable since once the underlying dynamics have been suitably represented, additional complexity in the model is unneeded. In fact, it might even decrease predictor performance as over-fitting of the training data occurs. A more detailed discussion of over-fitting and how to reduce it is presented later in section 4.5.3. There is also a theoretical limit as to how large K can be. For the Gaussian components to have a full covariance matrix, each of them must have at least $(d(d+1))/2$ non-degenerate data points, where d is the dimensionality of the joint vector $\mathbf{X}$. The size of the training set must then be at least $(Kd(d+1))/2$ and this is assuming that the data points are distributed such that each Gaussian gets the proper number of data points. In practice, such ideal conditions rarely occur and much more data is usually needed to reliably estimate the covariance matrices. This argues for choosing a smaller value of K, preferably the smallest K that gives reasonable performance.

There is also another motivation for keeping K small, which is that it affects the time required to generate predictions. This is discussed in more depth in section 4.4.

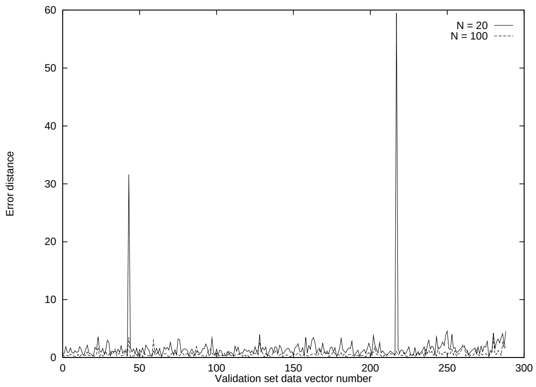
4.3.2 Effects of Varying N

We now examine the effects of varying N with K fixed at some reasonable value, which we have chosen to be $K = 20$. Recall that N allows us to measure how well a predictor has modelled the target dynamics. The smaller the value of N, the better the predictor. The performance of the probabilistic predictor on the "CHIME DIVA" sequence is shown in figure 4.3.

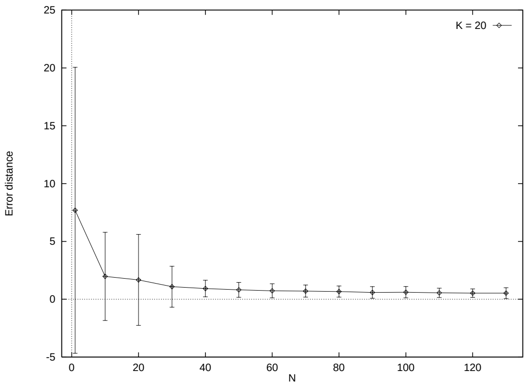
As with figure 4.2(a), figure 4.3(a) is a plot of the minimum distance statistic for each test vector, which for clarity, we have only shown the results for $N = 20$ and $N = 100$. Spikes in this plot indicate poorly modelled dynamics, which also contribute to long error bars in the summary plot in figure 4.3(b). It is not surprising that as N increases, the error in prediction decreases. However beyond a certain point, the decrease in error is negligible. As in the previous section, it is reasonable to define "negligible" as an error distance of less than 1 pixel. This first occurs at $N = 40$ with an average minimum error of 0.93 ± 0.72 pixels.

To give us an idea of how N quantifies the performance of a predictor, we compare the results obtained by the probabilistic predictor to that obtained by the constant velocity predictor given in (1.3). The inputs to the predictor are the object state and its instantaneous velocity at time t ($\mathbf{X}_t$ and $\dot{\mathbf{X}}_t$ respectively). Using a zero-mean Gaussian noise model to express uncertainty (ω), the output is the object state at time $t+1$ ($\mathbf{X}_{t+1}$). It is this object state that we use in (4.1) in place of $\mathbf{s}_{t+1}^{(n)}$, which we use to compare with the true object state at time $t+1$.

Figure 4.4(a) shows the results of testing the constant velocity predictor with $\sigma = 2$ pixels for Gaussian noise generation, which is the same as that used to obtain the results in figure 4.3(b) for the probabilistic predictor. It is clear that the performance of the constant velocity predictor is poorer than the probabilistic predictor for the same values



(a)

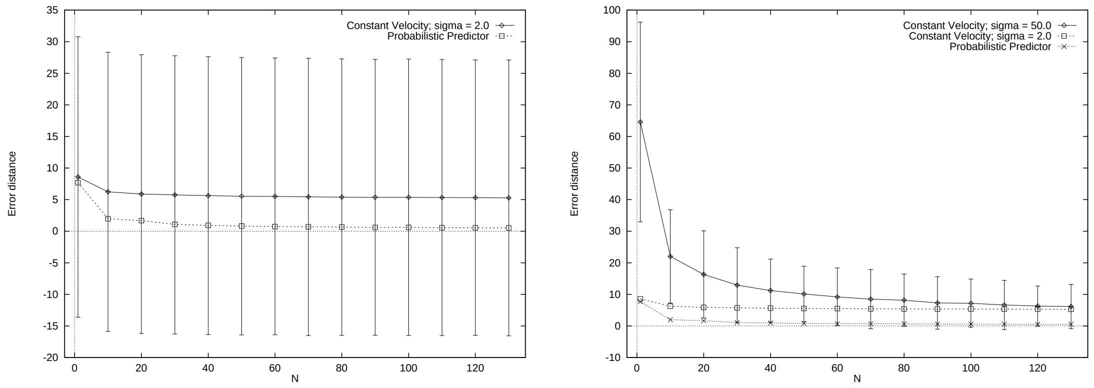


(b)

Figure 4.3: "CHIME DIVA" experiment results: effects of varying N with fixed $K = 20$. (a) shows the minimum distance statistic computed over N predictions for each test vector. (b) shows the average minimum distance with error bars. See text for a more detailed description.

of N. While the average minimum error is only slightly poorer, the error bars reveal that the standard deviation about this average is large. This was mainly caused by discontinuous dynamics in the sequence that were poorly represented by the constant velocity model. This is an example of a predictor being unable to deal with model failure. A tracker using this predictor would fail even if the verification process is perfect because possible dynamics in the environment were eliminated by the predictor.

The intuitive solution to this problem, and indeed many tracking implementations have done this, is to increase the process noise controlled by the noise parameter, σ. Based on visual inspection, we have chosen $\sigma = 50$ pixels, which is sufficient to cover all the discontinuous state changes within a 2σ radius except for the large one from the end of "E" to the beginning of "D". With this setting, we ran the quantification procedure again to obtain the results shown in figure 4.4(b). The predictor is now capable of handling discontinuous dynamics as can be seen from the shorter error bars. However, its performance is still poorer than the probabilistic predictor. For the constant velocity predictor to reach the accuracy attained by the predictor, N had to be increased drastically



(a) (b)

Figure 4.4: "CHIME DIVA" experiment results: comparing against the constant velocity predictor. (a) shows the results of the constant velocity predictor with $\sigma = 2$ pixels. The average minimum error of the probabilistic predictor shown in figure 4.3(b) is reproduced here (without error bars) for comparison. (b) shows the results of the constant velocity predictor with $\sigma = 50$ pixels. The two plots in (a) are reproduced here (without error bars) for comparison. Note: (a) and (b) have different scales along the vertical axes.

to $N = 8000$. Only then did it achieve an error of 0.91 ± 2.60 pixels. The probabilistic predictor, on the other hand, achieved this rate with only $N = 40$ samples showing that it was capable of properly constraining the prediction state space.

It is also interesting to note that for low values of N, the constant velocity predictor with $\sigma = 2$ outperformed the same predictor with $\sigma = 50$. This can be attributed to the tight constraints that the small amount of noise places enforces. However, model failure caused by discontinuous dynamics kept it from performing better even at higher values of N. In fact at $N = 8000$, the constant velocity predictor with $\sigma = 2$ had an error rate of 4.78 ± 21.47. This value is approximately the same as its error rate at $N = 130$. This shows the danger of model failure and it is a problem that changing N will not fix. This provides a strong motivation for the topic we will discuss in section 4.5.3, i.e. the topic of preventing over-training in the learning process.

Finally, we observe in figure 4.4(b) that the error rate of the constant velocity predictor with $\sigma = 50$ converges quite fast to that obtained by the same predictor with $\sigma = 2$. This effect is not surprising since the constant velocity model approximates continuous dynamics well. However, the large N that it requires to sufficiently sample the prediction state space is a limiting factor. As dimensionality increases, this space explodes exponentially and so will the number of samples needed to represent it properly. The probabilistic predictor was able to properly constrain this prediction state space such that only $N = 40$ was needed to represent it.

4.4 Running Time Analysis

A detailed study of the algorithm for generating predictions presented in section 2.1.3 reveals that the time taken to generate one prediction is in the order of

$$O\left[K(m^2 + m) + \log(K) + (mn + m + n) + n + n^2 + n\right] \tag{4.4}$$

where m and n are the dimensionalities of the prior and prediction states respectively. See section A.3 for the derivation of this result. If we assume that the prior is formed by concatenating q prediction states where $q \geq 1$, then we have $m \geq n$, which leads to the following simplified running cost:

$$O\left[Km^2\right] \tag{4.5}$$

This is the largest execution cost in the entire prediction procedure and it comes from computing the K v_k values. As mentioned at the end of section 4.3.1, this result further motivates keeping the value of K small.

4.5 Tuning the Predictor

Based on the insights gained from the analysis experiments in the previous sections, we now present several methods that can be used to tune the predictor to achieve better performance.

4.5.1 Automatic Bootstrapping

In section 2.3.4, we proposed a method of bootstrapping by running the tracker on the training image sequences and re-training the predictor for states where prediction errors occur. This was the method we used when training the predictor for the fish experiments. However, this method is time-consuming requiring much user intervention. In this section, we propose an alternative method of bootstrapping that is more automated.

Earlier, we observed that spikes in the graphs of figures 4.2(a) and 4.3(a) represent poorly modelled dynamics. Assuming that the training set is free from outliers, it follows that the data vectors causing these spikes are the data that the predictor needs to be re-trained on. The idea is then to develop a bootstrapping procedure that will automatically find these poorly modelled dynamics by looking for such spikes and to re-train the predictor on these poorly represented dynamics.

It remains to be defined how these spikes can be automatically found. For an experiment like the "CHIME DIVA" sequence where the prediction state has easily interpretable units (in this case, pixels), a meaningful threshold could be used to define which error distances indicate poorly represented dynamics and which do not. However for complex prediction states that comprise various shape, pose and other parameters, such a method is not feasible. Instead, we propose an alternative heuristic for detecting spikes using the average distance statistic computed over the entire training set. This is slightly different from the average distance statistic mentioned in section 4.1.2 in that previously, the statistic was computed over a set of N predictions for *one* single vector in the training set. Here, we compute the statistic over the set of N predictions for *all* vectors in the training set. The statistic that we propose is expressed as:

$$\tilde{D} = \frac{1}{N(T-1)} \sum_{t=1}^{T-1} \sum_{n=1}^{N} D^{(n)}(\mathbf{X}_{t+1}). \tag{4.6}$$

While, in general, average distance statistics provide poor measures of predictor performance, this statistic can be used to provide an upper bound on the minimum distance statistic and can be used to determine when a spike occurs. The automatic bootstrapping procedure can now be described as follows.

1. Run the quantification procedure using the training set only and record the distance measurements as described in step 3b. Compute the overall average distance statistic using (4.6).

2. Compute the minimum distance statistic of each data vector using (4.2). Find all poorly represented training vectors by checking for the following condition: $D_{min}(\mathbf{X}_{t+1}) > \tilde{D}$.

3. For all such training vectors, perturb them with noise and add them to the training set. Train the predictor on this new training set. Repeat this bootstrapping procedure until all training vectors are well represented by the predictor.

There are two points to note when using this bootstrapping procedure. Firstly, observe that the training set was used to both train the predictor as well as to drive the quantification procedure. This means that the test set is the same as the training set. Secondly, we have assumed that the training set is free from outliers. Hence the predictor should represent all training vectors well. However if this is not the case, then the automatic bootstrapping procedure will cause the predictor to learn these outliers, which may adversely affect its performance later. This is a caveat to keep in mind when using this procedure.

4.5.2 Choosing K

Model selection is the process of finding a suitable complexity for the learning architecture for it to best represent the training data. In our context, this refers to choosing an optimal value of K, which can be done as follows:

1. Divide the training set into two disjoint sets, namely an *estimation set* and a *validation set*. It was suggested in [25] that approximately 90% of the training set be used for the estimation set while 10% be used for the validation set.

2. Run the quantification procedure we have described earlier using the estimation set for training and the validation set for testing. This would produce a graph similar to that of figure 4.2(b). The optimal empirically-determined value of K can then be chosen by selecting the value of K where the error is smallest. We re-iterate that small values of K are favoured because it aids in better generalisation as well as being computationally less expensive. Hence the heuristic to use in the selection process would be to find the smallest value of K that gives an acceptable error margin.

4.5.3 Early Stopping to Prevent Over-fitting

It is important that during learning, the predictor does not over-fit on the training data, which will lead to poor generalisation performance. This point was reinforced by the empirical evidence shown in section 4.3.2 where a predictor that over-constrains the prediction state space (the constant velocity predictor with $\sigma = 2$ pixels) is unable to perform properly. That experiment also showed that a predictor that under-constrains the prediction state space (the constant velocity predictor with $\sigma = 50$ pixels) can be said to be a better choice than one that over-constrains it because by increasing N, there is still some hope of obtaining reasonable performance.

In this section, we describe the method known as *early stopping* of learning that prevents the predictor from over-fitting on the training data and hopefully allows it to have good generalisation performance. The idea behind this method is that beyond a certain point in the learning process, the learner will begin to learn the noise and peculiarities present in the training data. Hence it makes sense to stop the learning process before this occurs. Amari *et al.* [1, 2] have shown that if the amount of training data is less than the number of free parameters to be learnt (specifically that the number of training vectors is less than 30 times the number of free parameters), early stopping yields better generalisation performance. In our probabilistic predictor, the free parameters are the various w_k, μ_k and Σ_k of the Gaussian mixture, which numbers in the order of $O(Kd^2)$ where d is the dimensionality of the joint vector $\mathbf{X}$. This number grows very fast and easily exceeds the amount of training data available. For example, the exact number of free parameters for the fish experiments was:

$$K\left[1+d+\frac{d(d+1)}{2}\right] = 35\left[1+24+\frac{24(25)}{2}\right] \tag{4.7}$$

$$= 11375. \tag{4.8}$$

The number of independent training vectors, including those obtained from correcting mistracks, was only 7531. Hence early stopping is needed to prevent over-fitting.

The learning process using early stopping can be described as follows, where we have made use of the quantification procedure from section 4.1.2:

1. Divide the training set into an estimation set and a validation set. It was suggested in [1, 2] that the split be

$$r = \frac{\sqrt{2W-1}-1}{2(W-1)} \tag{4.9}$$

where W is the number of free parameters and r is the proportion of the training data to be used for the validation set.

2. Run the learning algorithm on the estimation set. At each iteration, run step 3 of the quantification procedure using the validation set to obtain the error rate given by (4.3). As long as this error rate decreases, continue running the learning algorithm. Terminate the learning algorithm when the error rate increases.

Chapter 5

Conclusion and Future Directions

In this thesis, we have developed a new prediction model based on probabilistic principles. This predictor has the advantage that it is capable of modelling a wide range of object state dynamics. Conceptually, it is also more general than existing predictors in that one need not rely on prior human knowledge to use this predictor. Furthermore, we have shown how this conceptual prediction framework can be implemented in a tractable manner so that predictions can be generated efficiently. This predictor was then integrated into the CONDENSATION framework and tracking experiments were performed to show its practical applicability. A detailed analysis of this predictor was also presented, which led to several suggestions for tuning the predictor being made.

In section 2.2, we discussed the various interesting aspects of the probabilistic predictor and its advantages over existing predictors. However, there are a number of benefits arising from the conceptual novelty of this prediction model that we have not yet exploited. We will explore these issues now and discuss potential future work that may be derived from this.

5.1 Dynamics Classification

Observe from the description of the predictor in chapter 2 that prediction is a two-step process, requiring first modelling of the object state dynamics, and then reproducing these dynamics. With this observation, we note that it is possible that after the object dynamics are modelled, there may be applications for it other than to generate predictions. One important application is the classification of dynamics. While this relationship has not been explicitly noted, the literature contains many examples of techniques originally developed for prediction being used (possibly after modification) for classifying dynamics [21, 9, 37].

In our work, this two-step process is explicit. The first step involves modelling the joint density (2.2) as described in section 2.1.2, which is used to represent knowledge about the object dynamics. The second step involves deriving the prediction state density (2.6) as described in section 2.1.3, which is used to generate predictions. Hence the idea for future work is to replace the second step with one more suitable for building a classifier of object dynamics.

This direction is the subject of current research. Although the work is still incomplete, we have had some promising results. The insight in this work is that (2.2) is one implementation of a radial-basis function network [30]. The Gaussians are the radial-basis functions representing the hidden layer of the network and the weights w_k implement a linear classifier in the space of the hidden units. In our probabilistic predictor, the weights have been estimated so that the equation represents the joint probability density between the prior and the prediction. By modifying these weights using a supervised learning algorithm, we can construct a linear classifier that can discriminate between different types of dynamics.

The same benefits said about the probabilistic predictor apply to a classifier constructed as described above too. Classifiers constructed from parametric models of dynamics require prior human knowledge that may not always be reliable. With sufficient reliable training data, our classifier is able to model more general classes of dynamics. Its classification complexity is constrained only by the assumption that these classes are linearly separable in the space of the hidden units.

Preliminary experiments on classifying fish dynamics have shown promising results. In these experiments, we classified fish dynamics as being "interesting" or "uninteresting", where "interesting" dynamics occur when the fish performs turning and twisting movements while "uninteresting" dynamics occur when it mainly translates with little deformation. The classifier achieved a recognition rate of 83% for the training set and 74% on the test set. Despite having seemingly low recognition rates, this result is encouraging because among other problems, the classes of dynamics we chose were complex and may not be linearly separable. This is supported by the fact that the recognition rates for the training and test sets are reasonably close. Also unlike classification of static patterns, classification of time-varying patterns usually involve hierarchical processing at different levels, of which our dynamics classifier is only one of them. Just like in speech recognition where it is common to divide a sentence into words and words into phonemes [34], dynamics can be divided into simpler movements, termed "movemes" in [9], that can be recognised separately and then combined at a higher level to form a complex movement. The current direction of this work is to develop this hierarchical recognition framework using our dynamics classifier at the lowest level to classify "movemes" with additional steps at higher levels to deal with the movements' syntactical structure.

5.2 Better Modelling of the Joint Density

In (2.2), we have chosen an additive Gaussian mixture representation. This has the effect of being able to model the effects of "positive" data, where "positive" refers to a training example that indicates an occurrence of such object state dynamics. However, it may be useful to be able to represent "negative" data as well, where such data represents knowledge that the object dynamics represented by it *does not* occur. The use of both types of data allow for a more refined specification of the joint probability density, which would lead to better predictor performance.

One method of addressing this problem is to use "negative" Gaussians, which are Gaussians that are learnt from "negative" data. The joint probability density in (2.2) can then be modified to become:

$$p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)}) = \sum_{k=1}^{K_+} w_k \, g(\mathbf{X}; \mu_k, \Sigma_k) - \sum_{k=K_+ +1}^{K} w_k \, g(\mathbf{X}; \mu_k, \Sigma_k) \tag{5.1}$$

where $1 \leq K_+ < K$ and the expression allows for an additional set of Gaussians to represent "negative" data. In practice, a representation involving both "positive" and "negative" Gaussians has been shown to be more effective in carving out state spaces [38].

However, this representation leads to the problem that the prediction state density, $p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)})$, is no longer a solely additive mixture of Gaussians. As such, we can no longer use the method described in section 2.1.3 to sample from it. One solution is to use the rejection method [33] instead, but this method is inefficient and the time taken to generate a prediction is not deterministic. One research direction would be to find a more efficient method of sampling from the new prediction state density.

An alternative solution is to use only "positive" Gaussians, as in (2.2), but to modify the learning algorithm to take into account "negative" data. This would allow the same method of generating predictions to be used but at the same time, have a more refined joint probability density. This would be another possible direction for future work.

5.3 Tracking Multi-Part Objects

It is easy to see that if there is reliable human knowledge indicating that one attribute in the prediction state is independent of another, then the problem of estimating the joint probability density can be simplified by learning and predicting these two attributes separately. The advantage of our predictor is that where these attributes are dependent but in a way that cannot be reliably described by human knowledge, the predictor is able to learn this dependency from training examples.

One application of this insight is the tracking of multi-part objects, such as the various parts of a human body, where each part constitutes some attributes of the prediction state. Each part may move in a manner dependent on some other part of the body. A common dependency is the set of rigid transformations, such as the turning of the head leading to a rotation of the facial features. This dependency can still be reasonably described with human knowledge. Other dependencies like a smiling face corresponding to an outreached hand during a handshake would be harder to describe. This would require a more advanced method of modelling dependency, such as that used in our probabilistic predictor. A possible research direction would be to investigate the feasibility of using the predictor for such a purpose.

Appendix A

Derivations

A.1 Deriving the Prediction State Density Function

We begin by expanding the Gaussian function in (2.2) to get:

$$p(\mathbf{X}_{t+1}, \mathcal{X}_t^{(q)}) = \sum_{k=1}^{K} w_k \frac{\exp\left\{-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1}(X-\mu_k)\right\}}{\sqrt{(2\pi)^{m+n}|\Sigma_k|}} \tag{A.1}$$

where the dimensionality of $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$ are m and n respectively. Substituting (A.1) into (2.1), we get:

$$p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)}) = \sum_{k=1}^{K} w_k \frac{\exp\left\{-\frac{1}{2}(X-\mu_k)^T \Sigma_k^{-1}(X-\mu_k)\right\}}{p(\mathcal{X}_t^{(q)})\sqrt{(2\pi)^{m+n}|\Sigma_k|}}. \tag{A.2}$$

At this point, the aim is to simplify this expression by separating the prior and prediction components of $\mathbf{X}$. The parts involving the prior will be constant and can then be incorporated as part of the constant v_k, leaving only the prediction components in the Gaussian representation. We show how to do this for only one Gaussian; the results we derive can similarly to be applied to all the Gaussians. Let us first break up the mean and covariance matrices as in (2.5). We then compute the inverse of the covariance matrix expressed as:

$$\Sigma_k^{-1} = \begin{bmatrix} U_k & V_k \\ V_k^T & W_k \end{bmatrix}. \tag{A.3}$$

It is useful to obtain the elements of the inverse in terms of those in (2.5), and this is done by forming $\Sigma_k^{-1}\Sigma_k = \mathbf{I}$ and equating the elements of both sides as follows:

$$\begin{gathered} \begin{bmatrix} U_k & V_k \\ V_k^T & W_k \end{bmatrix} \begin{bmatrix} A_k & B_k \\ B_k^T & C_k \end{bmatrix} = \mathbf{I} \\ \begin{bmatrix} U_k A_k + V_k B_k^T & U_k B_k + V_k C_k \\ V_k^T A_k + W_k B_k^T & V_k^T B_k + W_k C_k \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \end{gathered} \tag{A.4}$$

where $\mathbf{0}$ and $\mathbf{I}$ are the zero and identity matrices of appropriate size respectively. Comparing these terms, we get:

$$U_k B_k + V_k C_k = \mathbf{0} \tag{A.5}$$

$$U_k A_k + V_k B_k^T = \mathbf{I} \tag{A.6}$$

$$V_k^T A_k + W_k B_k^T = \mathbf{0} \tag{A.7}$$

$$V_k^T B_k + W_k C_k = \mathbf{I}. \tag{A.8}$$

Rearrange (A.7) to get

$$V_k = -A_k^{-1} B_k W_k \tag{A.9}$$

where we have made use of the fact that A_k^{-1} and W_k are symmetric. Substituting this into (A.8), we get:

$$W_k = (C_k - B_k^T A_k^{-1} B_k)^{-1}. \tag{A.10}$$

Now substituting (A.9) into (A.6) and rearranging, we get:

$$U_k = A_k^{-1} + A_k^{-1} B_k W_k B_k^T A_k^{-1}. \tag{A.11}$$

To summarise, we gather the results of (A.11), (A.9) and (A.10) here. Each solution also has a dual if the derivation had started with (A.5) instead of (A.7), which are also displayed here. Thus, we can now express Σ_k^{-1} in terms of the elements of Σ_k.

$$U_k = A_k^{-1} + A_k^{-1} B_k W_k B_k^T A_k^{-1} \quad = (A_k - B_k C_k^{-1} B_k^T)^{-1} \tag{A.12}$$

$$V_k = -A_k^{-1} B_k^T W_k \quad = -U_k B_k C_k^{-1} \tag{A.13}$$

$$W_k = (C_k - B_k^T A_k^{-1} B_k)^{-1} \quad = C_k^{-1} + C_k^{-1} B_k^T U_k B_k C_k^{-1}. \tag{A.14}$$

Proceeding with the derivation of the prediction state density, we focus our attention only on the Mahalanobis distance M in the Gaussian in (A.2) and use (2.5) to break up the mean and covariance matrices:

$$\begin{aligned} M &= (X - \mu_k)^T \Sigma_k^{-1} (X - \mu_k) \\ &= (\mathcal{X}_t^{(q)} - p_k)^T U_k (\mathcal{X}_t^{(q)} - p_k) + 2(\mathcal{X}_t^{(q)} - p_k)^T V_k (\mathbf{X}_{t+1} - q_k) \\ &\quad + (\mathbf{X}_{t+1} - q_k)^T W_k (\mathbf{X}_{t+1} - q_k) \end{aligned} \tag{A.15}$$

and then substituting the "left" equations of (A.12), (A.13) and (A.14) into this equation:

$$\begin{aligned} M = {} & (\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1} (\mathcal{X}_t^{(q)} - p_k) + (\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1} B_k W_k B_k^T A_k^{-1} (\mathcal{X}_t^{(q)} - p_k) \\ & - 2(\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1} B_k^T W_k (\mathbf{X}_{t+1} - q_k) + (\mathbf{X}_{t+1} - q_k)^T W_k (\mathbf{X}_{t+1} - q_k). \end{aligned} \tag{A.16}$$

Notice the structure of this last result. The first term is a Mahalanobis distance for the Gaussian for the prior, which is constant given $\mathcal{X}_t^{(q)}$. The next three terms form a quadratic expression that can be factorised as follows:

$$\left[(\mathbf{X}_{t+1} - q_k) - B_k^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k)\right]^T W_k \left[(\mathbf{X}_{t+1} - q_k) - B_k^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k)\right]. \quad \text{(A.17)}$$

If we let

$$\begin{aligned} \mu'_k &= q_k + B_k^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k) \\ \Sigma'_k &= W_k^{-1} = C_k - B_k^T A_k^{-1} B_k \end{aligned} \quad \text{(A.18)}$$

and substitute these back into M, we get:

$$M = (\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k) + (\mathbf{X}_{t+1} - \mu'_k)^T (\Sigma'_k)^{-1}(\mathbf{X}_{t+1} - \mu'_k). \quad \text{(A.19)}$$

Substituting this into (A.2), we get:

$$\begin{aligned} p(\mathbf{X}_{t+1}|\mathcal{X}_t^{(q)}) &= \frac{1}{p(\mathcal{X}_t^{(q)})} \sum_{k=1}^{K} w_k \frac{\exp\left\{-\frac{1}{2}(\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1}(\mathcal{X}_t^{(q)} - p_k)\right\}}{\sqrt{(2\pi)^m |A_k|}} \\ &\quad \frac{\exp\left\{-\frac{1}{2}(\mathbf{X}_{t+1} - \mu'_k)^T (\Sigma'_k)^{-1}(\mathbf{X}_{t+1} - \mu'_k)\right\}}{\sqrt{(2\pi)^n |\Sigma_k|/|A_k|}} \\ &= \frac{1}{p(\mathcal{X}_t^{(q)})} \sum_{k=1}^{K} w_k g(\mathcal{X}_t^{(q)}; p_k, A_k) g(\mathbf{X}_{t+1}; \mu'_k, \Sigma'_k) \\ &\propto \sum_{k=1}^{K} v_k g(\mathbf{X}_{t+1}; \mu'_k, \Sigma'_k). \end{aligned} \quad \text{(A.20)}$$

which is exactly what was stated in (2.6) where

$$v_k = w_k g(\mathcal{X}_t^{(q)}; p_k, A_k). \quad \text{(A.21)}$$

In these last few lines of this derivation, we have assumed that

$$|\Sigma'_k| = \frac{|\Sigma_k|}{|A_k|}. \tag{A.22}$$

To prove this, we first observe that by applying LU decomposition [33] to Σ_k, we can express Σ_k as a multiplication of two matrices: one lower-triangular and the other upper-triangular. With reference to the form of Σ_k defined in (2.5), we define these two matrices to be:

$$\Sigma_k = \begin{bmatrix} A_k & B_k \\ B_k^T & C_k \end{bmatrix} = \begin{bmatrix} A_k & \mathbf{0} \\ B_k^T & L \end{bmatrix} \begin{bmatrix} \mathbf{I} & N \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{A.23}$$

$$= \begin{bmatrix} A_k & A_k N \\ B_k^T & L + B_k^T N \end{bmatrix} \tag{A.24}$$

for some matrix L and N. $\mathbf{0}$ and $\mathbf{I}$ are zero and identity matrices, though possibly of different dimensionalities. Comparing both sides of the equation, we obtain:

$$C_k = L + B_k^T N \tag{A.25}$$

$$\text{and} \quad B_k = A_k N. \tag{A.26}$$

From (A.26), we get

$$N = A_k^{-1} B_k \tag{A.27}$$

and substituting this into (A.25) and rearranging, we get

$$L = C_k - B_k^T A_k^{-1} B_k. \tag{A.28}$$

To compute the $|\Sigma_k|$, we use (A.23) and the fact that A_k and L are both square matrices to get

$$\begin{aligned}|\Sigma_k| &= |A_k||L| \\ |L| &= \frac{|\Sigma_k|}{|A_k|}.\end{aligned} \tag{A.29}$$

Comparing (A.28) and (2.8), we see that $L = \Sigma'_k$, hence

$$|\Sigma'_k| = |L| = \frac{|\Sigma_k|}{|A_k|} \tag{A.30}$$

which proves our earlier assumption.

A.2 Deriving the Backward Prediction Equations

Using the "right" equations in (A.12), (A.13) and (A.14), the "reverse-time" analogue of the predictor can be derived. This modifies (A.16) to be:

$$\begin{aligned}M = (\mathbf{X}_{t+1} - q_k)^T C_k^{-1}(\mathbf{X}_{t+1} - q_k) + (\mathbf{X}_{t+1} - q_k)^T C_k^{-1} B_k^T U_k B_k C_k^{-1}(\mathbf{X}_{t+1} - q_k) \\ - 2(\mathcal{X}_t^{(q)} - p_k)^T U_k B_k C_k^{-1}(\mathbf{X}_{t+1} - q_k) + (\mathcal{X}_t^{(q)} - p_k)^T U_k (\mathcal{X}_t^{(q)} - p_k).\end{aligned} \tag{A.31}$$

Similar to the result we obtained before, the first term is the Mahalanobis distance for the Gaussian for $\mathbf{X}_{t+1}$. The following three terms can be factorised to produce:

$$\left[(\mathcal{X}_t^{(q)} - p_k) - B_k C_k^{-1}(\mathbf{X}_{t+1} - q_k)\right]^T U_k \left[(\mathcal{X}_t^{(q)} - p_k) - B_k C_k^{-1}(\mathbf{X}_{t+1} - q_k)\right]. \tag{A.32}$$

Redefining μ'_k and Σ'_k to be

$$\begin{aligned}\mu'_k &= p_k + B_k C_k^{-1}(\mathbf{X}_{t+1} - q_k) \\ \Sigma'_k &= U_k^{-1} = A_k - B_k C_k^{-1} B_k^T\end{aligned} \tag{A.33}$$

and substituting these back into M and then into (A.2), we manipulate the equations in the same way we did earlier to obtain:

$$p(\mathcal{X}_t^{(q)}|\mathbf{X}_{t+1}) = \sum_{k=1}^{K} v_k g(\mathcal{X}_t^{(q)}; \mu'_k, \Sigma'_k) \tag{A.34}$$

where v_k is now defined as

$$v_k = w_k g(\mathbf{X}_{t+1}; q_k, C_k). \tag{A.35}$$

A.3 Deriving the Time Complexity of the Predictor

In this section, we present a detailed derivation of the time complexity of the predictor. To do so, we analyse the prediction procedure presented in section 2.1.3 on page 18 and derive its time complexity step by step.

Before we begin, we discuss some conventions that we have used in this derivation. The notation we use here is the same as that used previously: K is the number of Gaussian components and m and n are the dimensionalities of the $\mathcal{X}_t^{(q)}$ and $\mathbf{X}_{t+1}$ states respectively. We also assume that where possible, quantities are pre-computed to save on run-time costs. Furthermore, we have assumed that a matrix multiplication is implemented in the naive method it is defined by. This means that the cost of multiplying a $p \times q$ matrix with a $q \times r$ matrix is $O(pqr)$. Faster methods of performing this operation will lower the upper-bound running time produced in this derivation. With this, we analyse each step of the prediction procedure as follows:

1. Selecting the Gaussian component can be done easily using cumulative weights of v_k leading to a $O(\log K)$ complexity. The more costly operation is to compute the K v_k values given the prior $\mathcal{X}_t^{(q)}$. This means evaluating (2.9) for $k = 1, \ldots, K$. To

make this evaluation more explicit, (2.9) can be expanded as:

$$v_k = w_k \frac{1}{\sqrt{(2\pi)^m |A_k|}} \exp\left\{ -\frac{1}{2}(\mathcal{X}_t^{(q)} - p_k)^T A_k^{-1} (\mathcal{X}_t^{(q)} - p_k) \right\}. \tag{A.36}$$

The evaluation of the Mahalanobis distance in the Gaussian is the most expensive operation costing $O(m^2 + m)$ due to the matrix subtraction and the two multiplications. Combined, this step yields a $O(K(m^2 + m) + \log K)$ complexity.

2. Computing μ'_k requires two matrix addition operations that cost $O(n)$ and $O(m)$, and a matrix multiplication that costs $O(mn)$. The total cost is $O(mn + m + n)$.

3. Obtaining and diagonalising the posterior covariance matrix is independent of $\mathcal{X}_t^{(q)}$. Hence this step does not contribute to the run-time cost.

4. This step involves using the Box-Muller method n times to generate the vector of Gaussian random variables. The Box-Muller method runs in constant time, which leads to a total cost of $O(n)$.

5. Obtaining $\mathbf{X}_{t+1}$ requires a matrix multiplication costing $O(n^2)$ and a matrix addition costing $O(n)$.

Summing these individual costs gives the expression shown in (4.4).

A.4 Further Justification for Using the Minimum Distance Statistic

An examiner of this thesis wrote:

> Page 59 states that minimum distance is used as a measure of performance. It tries to justify the use on the ground that “the experiments are repeated

> many times", so the [chance] of the algorithm "getting lucky" is small. This justification seems incorrect. Suppose that out of 100 trials, 99 of them return a distance of 0.9 but one of them returns a distance of 0.1, then the minimum distance would be 0.1 which is not at all representative of the actual performance, which is nearer to 0.9. So, the use of the minimum distance should be further justified.

In a separate comment, he suggests using the average distance statistic instead. In the main thesis, we have already stated our reasons for using the minimum distance statistic, as opposed to the average distance statistic. We continue to believe that our reasons are valid, though perhaps they have not been described in sufficient detail. In this section, we will elaborate on why the use of the average distance statistic is in fact incorrect. We will then argue that among the statistics proposed, the minimum distance statistic is the best choice. We concede that there may exist better statistics that might perform better than any of those discussed in this thesis. However, they would tend to be complicated to compute, and may not be without faults of their own either. Hence for simplicity, we still conclude that we are justified in using the minimum distance statistic.

A.4.1 Average distance statistic is a poor choice

As we have already explained in the thesis, the average distance statistic assumes that the prediction state density is unimodal. This assumption is most certainly violated in our experiments since we have intentionally made the prediction state density multimodal. So even if the predictor has perfectly modelled the dynamics, this statistic may still report that the predictor is performing poorly due to the violation of the statistic's underlying assumption.

Consider an example where the true prediction state density has m modes, where $m \geq 2$. Furthermore, suppose that a predictor has modelled this density perfectly. Ideally, we would want the chosen statistic to report this success. However, a particular test vector (specifically the $\mathbf{X}_{t+1}$ portion of the test vector) would be located at only one of these m modes. On the other hand, the N predictions generated by the predictor would be scattered over all m modes. This is proper behaviour for prediction because given a fixed prior, the predictor is expressing its belief that the prediction state may be close to any of these m modes. But the average distance statistic will produce a large value due to many of the N generated predictions being far away from the test vector, which will incorrectly indicate that the predictor is performing poorly.

We consider the average distance statistic to be a worse choice than the minimum distance statistic because we can already qualitatively described how it will fail. As such, this statistic should never be used in our quantification experiments, or any experiments where its unimodality assumption will be violated.

A.4.2 The minimum distance statistic is the best choice

In section 4.1.2, we suggested that several statistics can be obtained from the measurements collected in step 3b of the quantification procedure. The maximum distance statistic is obviously not useful in measuring how well a predictor is performing. In the previous section, we have argued that the average distance statistic is a poor choice as well. We are then left with the minimum distance statistic.

The use of the minimum distance statistic can be interpreted as using a tracker with a perfect verification step. This is because in such a tracker, as long as one prediction lies close to the true prediction state, the perfect verification step will be able to identify it and weight it appropriately for use in the next time step. In the same way, the minimum distance statistic only requires one prediction to be close to the true prediction state for it to report that the predictor is performing well. It may be unrealistic to expect a perfect verification step in a tracker, but we should be able to expect one that is reasonably accurate. As such, the use of the minimum distance statistic still bears a closer resemblance to normal tracking than any of the other proposed statistics. Hence, we believe that it is still the best statistic to use.

A.4.3 Other statistics may not be perfect either

Our explanation so far has focussed on comparing the minimum distance statistic with the other proposed statistics. However, it can be argued that there might exist statistics other than those proposed here that may be more suitable. For example, instead of choosing just one minimum distance value, we can take an average of the M smallest distance values and use this as the statistic. The question then would be how to choose M. A small M may receive the same criticism as the minimum distance statistic; a large M will increase the strength of the unimodality assumption and possibly skew our results. We have no satisfactory method of choosing M and any choice we make will be arbitrary, which will again come under criticism. Hence, we have chosen to fix $M = 1$, which is the minimum distance statistic we have described.

We concede that there may be other suitable statistics, but we believe that these will come at a price of increased complexity, which might require extensive justification for their use. So for the sake of simplicity, we have chosen to focus only on the easily computable statistics mentioned in section 4.1.2.

Bibliography

[1] Shun-ichi Amari, Noboru Murata, Klaus-Robert Muller, M. Finke, and Howard Hua Yang. Statistical Theory of Overtraining - Is Cross-Validation Asymtotically Effective? In D. S Touretzky, M. C Mozer, and M. E Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 176 – 182, 1996.

[2] Shun-ichi Amari, Noboru Murata, Klaus-Robert Muller, M. Finke, and Howard Hua Yang. Asymptotic Statistical Theory of Overtraining and Cross-Validation. *IEEE Transactions on Neural Networks*, 8(5):985 – 996, September 1997.

[3] R. Bartels, J. Beatty, and B. Barsky. *An Introduction to Splines for use in Computer Graphics and Geometric Modeling.* Morgan Kaufmann, 1987.

[4] D. Beymer, Amnon Shashua, and Tomaso Poggio. Example Based Image Analysis and Synthesis. A.I. Memo No. 1431, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1993.

[5] S. Birchfield. Elliptical Head Tracking Using Intensity Gradients and Color Histograms. In *Proc. IEEE Computer Vision and Pattern Recognition*, pages 232 – 237, 1998.

[6] A. Blake and M. Isard. *Active Contours.* Springer-Verlag, 1998.

[7] A. Blake, M. Isard, and D. Reynard. Learning to Track the Visual Motion of Contours. *Artificial Intelligence*, 78:101 – 133, 1995.

[8] A. Blake, B. North, and M. Isard. Learning Multi-Class Dynamics. In *Advances in Neural Information Processing Systems*, volume 11, 1999.

[9] Christoph Bregler. Learning and Recognizing Human Dynamics in Video Sequences. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 568 – 574, 1997.

[10] Christoph Bregler and Stephen M Omohundro. Surface Learning with Applications to Lipreading. In *Advances in Neural Information Processing Systems*, volume 6, 1994.

[11] P. J. Brockwell and R. A. Davis. *Introduction to Time Series and Forecasting.* Springer, 1996.

[12] T.F. Cootes, D. Cooper, C. J. Taylor, and J. Graham. Active Shape Models - Their Training and Application. In *Computer Vision and Image Understanding*, volume 61, pages 38 – 59, 1995.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, 39(1):1 – 38, 1977.

[14] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, 1973.

[15] G. J. Edwards, C. J. Taylor, and T. F. Cootes. Learning to Identify and Track Faces in Image Sequences. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 317 – 322, 1998.

[16] P. Fieguth and D. Terzopoulos. Color-Based Tracking of Heads and Other Mobile Objects at Video Frame Rates. In *Proceedings of the IEEE Computer Vision and Pattern Recognition*, pages 21 – 27, 1997.

[17] T. Heap and D. Hogg. Improving Specificity in PDMs using a Hierarchical Approach. In *Proceedings of the British Machine Vision Conference*, 1997.

[18] T. Heap and D. Hogg. Wormholes in Shape Space: Tracking through Discontinuous Changes in Shape. In *Proceedings of the Sixth International Conference on Computer Vision*, pages 344 – 349, 1998.

[19] John W Fisher III, Alexander T Ihler, and Paul A Viola. Learning Informative Statistics: A Nonparametric Approach. In S A Solla, T K Leen, and K.-R. Mller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 2000.

[20] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *Proc. European Conf. Computer Vision*, pages 343 – 356, 1996.

[21] M. Isard and A. Blake. A mixed-state CONDENSATION tracker with automatic model-switching. In *Proc. 6th Int. Conf. Computer Vision*, 1998.

[22] M. Isard and A. Blake. CONDENSATION – conditional density propagation for visual tracking. *Int. Journal Computer Vision*, 29(1):5 – 28, 1998.

[23] M. Isard and A. Blake. ICondensation: Unifying Low-level and High-level Tracking in a Stochastic Framework. In *Proc. 5th European Conf. Computer Vision*, volume 1, pages 893 – 908, 1998.

[24] I. T. Jolliffe. *Principal Component Analysis*. Springer-Verlag, 1986.

[25] M Kearns. A Bound on the Error of Cross Validation Using the Approximation and Estimation Rates with Consequences for the Training-Test Split. In D. S Touretzky, M. C Mozer, and M. E Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 183 – 189, 1996.

[26] Genshiro Kitagawa. Monte Carlo Filter and Smoother for non-Gaussian Nonlinear State Space Models. *Journal of Computational and Graphical Statistics*, 5(1):1 – 25, 1996.

[27] Alberto Leon-Garcia. *Probability and Random Processes for Electrical Engineering.* Addison-Wesley Publishing Company, 1994.

[28] S J Maybank, A D Worrall, and G D Sullivan. Filter for Car Tracking Based on Acceleration and Steering Angle. In *Proceedings of the 7th British Machine Vision Conference*, 1996.

[29] Jerry M Mendel. *Lessons in Estimation Theory for Signal Processing, Comunications, and Control.* Prentice Hall, 1995.

[30] Tomaso Poggio and Frederico Girosi. Networks for Approximation and Learning. *Proceedings of the IEEE*, 78(9):1481 – 1497, 1990.

[31] Tomaso Poggio and Thomas Vetter. Recognition and Structure from one 2D Model View: Observations on Prototypes, Object Classes and Symmetries. A.I. Memo No. 1347, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[32] Dean Pomerleau. *Neural Network perception for Mobile Robot Guidance.* PhD thesis, Carnegie Mellon University, February 1992.

[33] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C.* Cambridge University Press, second edition, 1992.

[34] L. Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition.* Prentice Hall, 1993.

[35] R. A. Redner and H. F. Walker. Mixture Densities, Maximum Likelihood and the EM Algorithm. *SIAM Review*, 26(2):195 – 239, April 1984.

[36] D. Reynard, A. Wildenberg, A. Blake, and J. Marchant. Learning Dynamics of Complex Motions from Image Sequences. In *Proc. European Conf. Computer Vision*, volume 1, pages 357 – 368, 1996.

[37] J. Rittscher and A. Blake. Classification of Human Body Motion. In *Proceedings of the Seventh International Conference on Computer Vision*, volume 1, pages 634 – 639, 1999.

[38] Kah Kay Sung. *Learning and Example Selection for Object and Pattern Detection.* PhD thesis, Massachusetts Institute of Technology, December 1995.