*University of East Anglia* *School of Information Systems*

# Using Magnetic Positional Sensors to Assist in the Recognition of Human Gestures from Visual Imagery

**Roger Rowland – 9964924**

*July 2001*

# Table of Contents

# List of tables

# List of figures

# Abstract

Attempting to recognise and interpret human gestures using machine vision poses a number of difficult problems. There are many areas in which substantial progress is needed before the full benefits of computer-based gesture recognition could be realised in real world applications. These range from the need for some degree of language processing to the more practical limitations imposed by the level of ability of computer hardware.

When attempting to interpret human motion using a single-camera view, it is helpful to be able to estimate the pose of the subject in three dimensions. When dealing with the detailed movements necessary for a gesture recognition or sign-language application, it is necessary to closely examine the shape and orientation of the hand. The significant potential for (self) occlusion in such images makes this a particularly difficult problem.

For this research, we are employing a novel magnetic positioning system which allows the capture of accurate 3D positional information from sensors small enough to be placed at various sites on the hand. By capturing and analysing this positional data together with images from a single video camera, we are attempting to develop a viable method for reconstructing the shape of the hand using the video data alone. If successful, we hope that this will provide us with a way to disambiguate particularly hard-to-decipher gestures.

# Chapter 1 Introduction

## 1.1 Introduction

### *1.1.1 Overview*

In recent years, technological advances in hardware design and construction have provided a significant increase in the availability of computing power at a greatly reduced cost. One effect of this has been a proliferation of research in the area of computer vision.

Lately, there has been an increasing interest in the automated recognition and tracking of human motion. Potential applications include; motion capture - to assist computer-generated animation for the film industry or to analyse sporting performance, automated surveillance - for security purposes or behavioural analysis, and a number of subjects broadly covered by the term *gesture recognition*. The latter includes such areas as Human-Computer Interaction (HCI) and the recognition and interpretation of sign language.

This document describes the application of a new 3D magnetic positioning system - originally developed for producing three-dimensional images of medical endoscopes – to attempt to address some of the problems affecting research into machine based gesture recognition.

### *1.1.2 Structure of this document*

The remainder of this chapter gives a brief introduction to machine based gesture recognition and highlights some of the problems involved. At the end of this chapter, the aims and objectives of the project are stated.

Chapter 2 presents a review of existing research into gesture recognition and introduces some techniques which are of particular relevance to this work. These include the point distribution model (PDM) and the hidden Markov model (HMM).

Chapter 3 describes the methods used to conduct this research so far. These include the development of a new data glove using magnetic positional sensors and the implementation of software for building point distribution models and hidden Markov models.

Chapter 4 describes the current status of this project in terms of the results obtained so far, the problems encountered, and the proposed next steps for continuing the research.

## 1.2 Machine based gesture recognition

### *1.2.1 Introduction*

Recognising human gestures is a complicated task which involves inter-disciplinary aspects such as motion modelling, motion analysis, pattern recognition, artificial intelligence, and even psycho-linguistic studies.

A large part of the existing research [38] into machine based gesture recognition has involved various combinations of two techniques; the analysis of video images using one or more cameras, and/or some form of instrumentation of the subject using, for example, data gloves or colour coding. In addition, the analysis may be focused on purely static poses of the subject or may also involve a dynamic element by considering temporal changes. Both static and dynamic analyses typically entail the application of statistical models, for example active shape models [8], point distribution models [5], and hidden Markov models [37].

### *1.2.2 Problems and challenges*

Collecting suitable data for gesture recognition is not trivial – when analysing video images, the subject has first to be located in each image frame and possibly segmented from the background before any motion tracking or analysis of shape or pose can be attempted.

Many of the 2D image processing techniques required are widely used in computer vision applications [30] and are not in themselves a major problem. It has also been shown that it is possible to automatically track a moving object in

real time using a video camera by employing a contour tracking technique which gives good performance even in the presence of background clutter [15].

However, to produce a system which can recognise a wide range of human gestures, it is also necessary to be able to discover and interpret the motion and pose of the subject in *three* dimensions. When considering the human hand as playing an instrumental part in the execution of a gesture, then shape and orientation are as important as movement – for some gestures perhaps even more so. In such a highly articulated structure as a hand, this is a particularly difficult problem for a vision based system due to the high likelihood of occlusion.

## 1.3 Aims and objectives

### *1.3.1 Project aim*

The aim of this project is to develop a technique whereby as much information as possible about the shape and orientation of the human hand in three dimensions may be derived from a single video camera view. This information will then be used in an attempt to disambiguate some particularly hard-to-decipher gestures.

### *1.3.2 Objectives*

The detailed objectives of this project are as follows:

- *To review existing research into machine based gesture recognition and to describe some of the specific techniques which are of interest to this project.* — ***Chapters 1, 2***
- *To develop the hardware and software necessary for creating a data glove using magnetic positional sensors.* — ***Chapter 3***

- *To design and develop software to allow the capture of 2D and/or 3D data from the new data glove.*

- *To design and develop software to produce point distribution models using data captured by the glove and to explore the properties of these models.*

- *To design and develop software for implementing hidden Markov models and to use this to model the dynamics of captured gestures.*

***Chapter 3***

- *To re-assess the aims and objectives of this project in the light of results found so far and to propose alternative paths for progressing the research to some conclusion.*

***Chapter 4***

# Chapter 2 Gesture recognition

## 2.1 Background

### *2.1.1 Introduction*

The functions required for machine based gesture recognition could be broadly grouped into four areas:

- Gesture representation – what constitutes a gesture?
- Feature extraction – what features are important for recognising a particular type of gesture?
- Data collection – how can information about these required features be reliably and efficiently collected?
- Recognition – how the data collected is used to 'understand' gestures.

### *2.1.2 Gesture representation*

Gesture representation involves some description - possibly linguistic - of what constitutes a gesture in a given context. This can vary widely depending on the intended application.

In sign language for instance, signs may be classified by hand shape, hand orientation, position (usually with respect to the body), and movement [6]. In some HCI applications, hand shape and orientation may not be as relevant where gestures are intended more to control or manipulate real or virtual objects and less as a means of communication.

Generally, gestures requiring a more complex representation will be more difficult to recognise. Vogler and Metaxas [36] have demonstrated that three dimensional features are superior to two dimensional ones when attempting to recognise sign language, while Min et al [7] show that two dimensional images are sufficient for a gesture-driven editing system.

In a more general survey, Gavrila [11] supports this idea and classifies the techniques used in previous work into three (sometimes overlapping) areas:

- 2D approaches without explicit shape models,
- 2D approaches with explicit shape models, and
- 3D approaches

These are discussed in some detail in [11] and Gavrila concludes that the most suitable approach to pursue depends primarily on the intended application. A 2D approach is effective for applications where precise pose recovery is not required and represents the easiest solution for applications involving a single subject with constrained movement and a fixed viewpoint. 3D approaches are more effective where there are multiple subjects, cluttered backgrounds or occluded features.

### *2.1.3 Feature extraction*

As noted above, the intended application largely defines the features of interest. In a system where purely 'gross' movements of the hand-arm system are of interest [7], employing some form of segmentation of a 2D image – for example background subtraction or colour segmentation – reveals those features of interest, in this case, motion trajectories.

For static hand posture recognition, although some geometric features may be discernible (e.g. fingertips, contours), these are not always reliable due to self-occlusion and possibly difficult lighting conditions.

In such cases, where there is no instrumentation of the subject, the *whole* image may be taken as the input to a recogniser where features are selected implicitly or automatically. This is where some form of pattern matching may be used and often involves the use of some sort of shape model, thereby introducing some degree of foreknowledge about the application area. Kakadiaris and Metaxas [16] have recently shown how a model-based approach may be used to track upper-body extremities in three dimensions based on a spatio-temporal analysis of the subject's silhouette.

### *2.1.4 Data collection*

As the intended application defines the features of interest, so these features influence the methods of data collection. The use of video cameras is almost universal and some researchers use two or more cameras to circumvent problems with occlusion and cluttered backgrounds [26, 27, 28]. In many cases when dealing with the analysis of human movement, colour is an important aspect as the hues generated by skin tones fall in a quite well-defined range and ease the segmentation of the subject from the background.

### *2.1.5 Recognition*

Analysing the sometimes vast amounts of data collected by video cameras and/or data gloves can be a daunting task and consume inordinate amounts of computing

power. This is especially true where a three dimensional approach has been taken and as Gavrila [11] notes,

*"It is fair to say that the results of vision-based 3D tracking are still limited at this point. Few examples of 3D pose recovery on real data exist in the literature and most of these introduce simplifications (e.g. constrained movement, segmentation) or limitations (e.g. processing speed) that still require improvement with respect to robustness. Robust 3D tracking results have been particularly scarce for approaches using only one camera."*

It is clear that the ability to recover 3D pose from a single camera view would be very desirable in the search for a more robust gesture recognition system but that this will not be an easy problem to overcome.

Many existing researchers have employed some form of statistical model to assist in the recognition of objects from video images. Some of these techniques are described in detail in the next section.

## 2.2 Statistical models for gesture recognition

### *2.2.1 Introduction*

This section describes some techniques which are of particular interest to this research. Among these are the point distribution model (PDM) and the hidden Markov model (HMM). The way in which these models are developed and used in

this project is described in chapter 3. The remainder of this chapter describes the underlying principles.

### *2.2.2 The point distribution model*

Models are widely used in computer vision to classify, track or locate image features using some prior knowledge of object shape. As many objects are non-rigid, some sort of deformable model is often required in order to capture shape variability.

The point distribution model (PDM) [8] is one such model. It employs techniques based on linear statistics and so is particularly suited to modelling shape variation where the inherent differences between similar shapes can be represented by a linear function – for example, particular parts of the shape distort by moving only along straight lines. A PDM is typically constructed as follows:

If a non-rigid object is defined in terms of landmark points positioned strategically on various object features (and optionally at regular intervals in between) then, by labelling these points on a set of training examples of the object, the following approach may be used to discover the mean object shape and the major modes of shape variation.

Taking an example of a shape represented by a 2D contour, the $n$ landmark points chosen form a list of $n$ $(x, y)$ co-ordinates defining each example. Each of these examples, or poses, may instead be considered a vector composed of a concatenation of these co-ordinates, e.g. $(x_1, y_1, x_2, y_2, \ldots x_n, y_n)$ which effectively

defines a single point in a *2n*-dimensional space (often referred to as *shape space*). As these points in shape space are produced from similar training examples, they will tend to form a cluster, which in the ideal case will be uniform and hyper-elliptical in nature. Note that some pre-alignment of the training data (scaling, rotation, translation) is required to ensure that the best model is produced.

Having assembled a training set $E$ of $N$ example vectors, $x_i$ *(i=1, ..., N)*, the mean shape is calculated by finding the average vector. Principal component analysis (PCA) [12] is then performed on the deviation of the example vectors from the mean using eigenvector decomposition on the covariance matrix $S$ of $E$ where,

$$S = \frac{1}{N}\sum_{i=1}^{N}\left(x_i - \bar{x}\right)\left(x_i - \bar{x}\right)^T$$

However, where the dimensionality of the data is high – for example a contour of 300 (*x, y*) points giving a 600-dimensional training vector – then the calculation of the eigenvectors and eigenvalues of the covariance matrix can be lengthy and the physical memory used by the computer prohibitive.

To overcome this, Bowden [4] notes that it is not always necessary to solve a matrix for <u>all</u> eigenvectors. If the number of training examples, $N$, is less than the dimensionality, then the number of eigenvectors that can be extracted from the covariance matrix *cannot* exceed the number of training examples (*N-1*). In such cases, significant computational savings can be made by solving for a smaller $N \times N$ matrix derived from the same data.

As detailed in [4], if the covariance matrix, $S$, defined above is rewritten as

$$S = \frac{1}{N} DD^T$$

where $D$ is, in the above example, a $600 \times N$ matrix with the training samples as columns, then if a new matrix, $T$, is a smaller $N \times N$ matrix

$$T = \frac{1}{N} D^T D$$

and $e_i$ $(i=1, \ldots, N)$ are the unit, orthogonal eigenvectors of $T$ with the corresponding eigenvalues $\gamma_i$

$$Te_i = \gamma_i e_i \ (i=1, \ldots, N)$$

then

$$\frac{1}{N} D^T De_i = \gamma_i e_i$$

premultiplying by $D$ yields,

$$\frac{1}{N} DD^T De_i = \gamma_i De_i$$

and therefore $S(De_i) = \gamma_i(De_i)$

So if $e_i$ is an eigenvector of $T$, then $De_i$ is an eigenvector of $S$ and has the same eigenvalue. The $N$ unit eigenvectors of $S$ are then $v_i$ $(i=1, \ldots, N)$ where

$$v_i = \frac{1}{\sqrt{\gamma_i N}} De_i$$

with corresponding eigenvalues $\lambda_i = \gamma_i$.

In a compact model, it is usually found that a large percentage of the variability can be captured by the first few modes of variation which, together with the mean shape, gives a convenient way to describe any reasonable pose of the original model.

As described above, the $N$ unit eigenvectors of the covariance array $S$ are $v_i$ $(i=1, \ldots, N)$, so if we take the $t$ unit eigenvectors of $S$ corresponding to the $t$ largest eigenvalues, then it is possible to generate a deformed shape $x$ by adding weighted combinations of $v_i$ to the mean shape

$$x = \bar{x} + \sum_{i=1}^{t} b_i v_i$$

where $b_i$ is the weighting for the $i^{th}$ variation vector.

Cootes [8] expresses this in matrix form

$$x = \bar{x} + Pb$$

where $P = (v_1, v_2, \ldots, v_t)$ is a matrix of the first $t$ eigenvectors and $b = (b_1, b_2, \ldots, b_t)^T$ is a vector of weights.

Generally accepted ranges for the weights are $\pm 3\sqrt{\lambda_i}$ given that this is really equivalent to 3 standard deviations (the square root of $\lambda_i$ being the standard deviation of the variance along $v_i$) and so encompasses in excess of 99% of the total deformation.

So with a well-defined PDM, it is possible to explore almost the whole range of deformation indicated by the original training set, producing viable shapes which were not present in the training data. It is also possible to classify shapes using a much smaller number of parameters than those used to define each training shape. This is important when considering the recognition of previously-unseen shapes.

### *2.2.3 Non-linear point distribution models*

As the 'standard' PDM described above is based purely on linear statistics then, for any particular mode of variation, landmark points can move only along straight lines. Non-linear movement is achieved by combining two or more modes.

This situation is not ideal. Firstly because the most compact representation of shape variability might not be achieved, and secondly because implausible shapes can be generated due to the incorrect assumption that the variation modes are independent. The effect is particularly bad when the object being modelled can bend or pivot. Like the fingers of a hand, for example.

Also, when building a set of training data for a PDM, the accurate labelling of landmark points is very important. The landmark points on each training example should relate very closely to the *same* physical feature on the object as it is the relative movement of these points that is significant. Inaccurate identification of landmarks introduces additional non-linear behaviour into the model which again results in the model exhibiting deformations not present in the original training set.

However, when modelling an articulated shape like the hand, PDM's *can* still be useful. Sozou et al. [31] have proposed using polynomial regression to fit high-order polynomials to the non-linear axis of the training set and later [32], the same authors used a back-propagation neural network to perform non-linear PCA. Heap and Hogg [13] suggest using a log polar mapping to remove non-linearity from the training set *before* PCA followed by the reverse process afterwards. These authors have shown that this may be very useful in applications where the non-linearity is mainly due to pivotal motion.

All of the above attempt to overcome the problems caused by some non-uniform distribution of training points in shape space which - in extreme cases - may form into discrete 'clumps'. In such cases, some form of clustering technique may be used to analyse this distribution of training points. This allows the more simple calculations involved in the linear PDM to be retained and offers significant performance benefits over some of the alternative techniques, like polynomial regression [31].

In a situation like that described above, the distribution of points in shape space effectively form a number of smaller models, which may overlap to some greater or lesser extent. As a gesture is made, it will perhaps fit into one of these models more closely than the others and, as the gesture progresses, this model will gradually become less reliable until it becomes necessary to switch to a more suitable one. The problem then arises of deciding *when* best to switch between these models and in *what* order.

As there will inevitably be areas where this choice is ambiguous, some form of probabilistic matching would be appropriate to model these dynamic changes during the training process. One possible way of achieving this would be to use a hidden Markov model.

### *2.2.4 The hidden Markov model*

Hidden Markov models are a class of statistical models which attempt to characterise the properties of the observable output from a real-world process. This is particularly relevant where the observable output produced by the real-world process is subject to noise or distortion. In recent years, HMM's have been successfully applied to speech recognition systems [41].

To understand exactly what an HMM is, it is first necessary to understand a little about Markov chains, or first-order Markov models. The following description is largely abstracted from [23] and [25] although a more comprehensive coverage is given by [14].

#### Discrete Markov chains and HMM's

Consider a system which may be described at some point in time as being in one of a set of distinct states[1].

Suppose there are $N$ states such that the set of states $S = \{S_1, S_2, \ldots, S_N\}$.

[1] In this context, we are initially concerned only with regularly spaced discrete times (hence 'discrete' as opposed to 'continuous') and a *finite* number of distinct states. This may need to be reviewed.

At each discrete point in time, the system undergoes a change of state according to a set of probabilities associated with that state. Note that the change may possibly be back to the same state. In a first-order Markov model, the 'Markov assumption' (or Markov property) is said to hold for a system where the probability of moving from one state to another depends only on the current state, not on any part of the previous history of states. Also, in this application, we are only concerned with systems where the above probabilities do not vary over time (stationary models).

In these conditions, if the time instants associated with state changes are $t = 1, 2, \ldots$, and the actual state at time $t$ is $q_t$, then there will be a set of state transition probabilities $a_{ij}$ of the form

$$a_{ij} = P(q_t = S_j \mid q_{t-1} = S_i), \qquad 1 \leq i, j \leq N$$

with the state transition coefficients, obeying standard stochastic constraints, having the following properties

$$a_{ij} \geq 0 \qquad \text{and} \qquad \sum_{j=1}^{N} a_{ij} = 1$$

The above stochastic process could be called an 'observable' Markov model as the output of the process is just the state at each instant in time, where each state corresponds to an observable (physical) event.

In a hidden Markov model, the underlying states are *not* directly observable (hence 'hidden') but are observed through *another* set of stochastic processes associated with each state.

Suppose the system produces a number, $M$, of distinct observable 'symbols'. Let this set of symbols be denoted by $V = \{v_1, v_2, ..., v_M\}$. Then for each state, $j$, there will be an observation-symbol probability distribution $B = \{b_j(k)\}$, where

$$b_j(k) = P(v_k \text{ at } t \mid q_t = S_j), \qquad 1 \leq j \leq N \qquad 1 \leq k \leq M$$

This gives the probability that symbol $v_k$ will be observed given that the system is in state $S_j$ at time $t$.

One more probability distribution completes the elements characterising an HMM. This is the *initial* state distribution $\pi = \{\pi_i\}$ where

$$\pi = P(q_1 = S_i), \qquad 1 \leq i \leq N$$

This gives the probability that the system begins in a particular state.

To summarise, the probability measures characterising an HMM are as follows

| | | |
|---|---|---|
| $A = \{a_{ij}\}$ | - | the state transition probability distribution |
| $B = \{b_j(k)\}$ | - | the observation symbol probability distribution, and |
| $\pi = \{\pi_i\}$ | - | the initial state distribution |

While a complete specification of an HMM requires the specification of two model parameters ($N$ and $M$), specification of observation symbols, $V$, and the specification of the above three probability measures, for convenience the following compact notation is used to indicate the complete parameter set of the model

$$\lambda = (A, B, \pi)$$

#### Three problems

For a HMM to be useful in modelling a real-world process, there are three basic problems which need to be solved. These are :

1. Given the observation sequence $O = O_1, O_2, ..., O_T$, and a model $\lambda = (A, B, \pi)$, how do we efficiently compute $P(O \mid \lambda)$, the probability of the observation sequence given the model?

2. Given the observation sequence $O = O_1, O_2, ..., O_T$, and a model $\lambda$, how do we choose a corresponding state sequence $Q = q_1, q_2, ..., q_T$ which best explains the observations?

3. How do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximise $P(O \mid \lambda)$ ?

**Problem 1**, the 'evaluation problem', effectively asks for a method of scoring how well a particular model matches a given observation sequence. In a limited vocabulary speech (or gesture) recognition system, it could be that a different HMM would be created to 'recognise' each possible 'word' in the vocabulary.

The solution to this problem would provide a method of identifying which model best fits an unfamiliar word and so give an indication of what that word might be.

**Problem 2** attempts to uncover the 'hidden' part of the model. This is not easy to accomplish given that there may be a number of plausible underlying states explaining an observation sequence, when using different criteria for judging plausibility.

The solution to this problem may be useful in gaining insights about the structure of the model. Such insights might provoke changes in the number of states, for example, to optimise the suitability of a model for a particular purpose.

**Problem 3** addresses training. Given that we have an observation sequence (or a number of such sequences), how do we adjust the model to maximise the likelihood that it would produce this (these) sequence(s)?

The solution to this problem would allow a 'training' function to be built into a speech (or gesture) recognition system where, during training, the same word or phrase is delivered by different speakers and the output used to adjust the model to maximise the chance of it correctly identifying the same word or phrase when spoken by an unfamiliar voice.

#### Three solutions

The solutions to all three problems are well documented [25]. The solution to problem 1 is provided by the 'Forward' part of the '*Forward-Backward Procedure*'. This is an elegant and efficient algorithm which uses induction to avoid the brute force approach involving the enumeration of every possible state sequence of the same length as the observation. The *Forward* algorithm defines the 'forward variable' $\alpha_t(i)$ as

$$\alpha_t(i) = P(O_1 O_2 \ldots O_t, q_t = S_i \mid \lambda)$$

that is, the probability of the partial observation sequence $O_1, O_2, \ldots, O_t$ (until time $t$) *and* state $S_i$ at time $t$, given the model $\lambda$. The algorithm is implemented as follows -

Initialisation: $\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \leq i \leq N$

Induction: $\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \qquad 1 \leq t \leq T\text{-}1, \quad 1 \leq j \leq N$

Termination: $P(O \mid \lambda) = \sum_{i=1}^{N} \alpha_T(i)$

The solution to problem 2 is given by the *Viterbi Algorithm*. This algorithm is based on dynamic programming methods and is in many respects similar to the implementation of the *Forward* algorithm described above. A discussion of this algorithm along with a complete implementation is given in [23]. The main difference is that the summation performed in the induction step is replaced by a maximisation procedure.

Problem 3 is by far the most difficult to solve. There are a variety of optimisation methods in existence [23, 24], one popular technique being the *Baum-Welch* algorithm [25]. This algorithm utilises the 'forward variable' produced by the solution to problem 1 along with the 'backward variable' which is produced by the second part of the *Forward-Backward* procedure. The backward variable $\beta_t(i)$ is defined as

$$\beta_t(i) = P(O_{t+1}\, O_{t+2} \ldots O_T \mid q_t = S_i, \lambda)$$

that is, the probability of the partial observation sequence from $t + 1$ to the end, given state $S_i$ at time $t$ and the model $\lambda$. As with the calculation of the forward variable, induction is used to good effect -

Initialisation: $\beta_T(i) = 1, \quad 1 \leq i \leq N$

Induction: $\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j) \quad t = T-1, T-2, \ldots, 1 \quad 1 \leq i \leq N$

The forward and backward variables together may be used to compute the probability that the system was in state $S_i$ at time $t$ and state $S_j$ at time $t+1$ during the given observation sequence.

From this point, by summing over the time index, it is possible to arrive at a way of *counting* the number of times the system leaves a particular state, and the number of times the system transitions from state $S_i$ to state $S_j$.

The full implementation is described in [25], but it can be seen that the above information may be used to re-estimate the model parameters $\lambda$ as follows

$\pi_i$ = expected number of times in state $S_i$ at time $t = 1$

$$a_{ij} = \frac{\text{expected number of transitions from state } S_i \text{ to state } S_j}{\text{expected number of transitions from state } S_i}$$

$$b_j(k) = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j}$$

**Summary**

Hidden Markov models have been used extensively for speech recognition systems [41] and also by a number of researchers investigating gesture recognition, for example [33, 19, 37]. They will almost certainly form part of *this* project although at the time of writing, the exact details of their application have not been decided. This is discussed further in Chapter 4.

### *2.2.5 Recovering 3D pose from a 2D view*

As stated earlier, the ability to recover information about the pose of the hand in three dimensions using a 2D camera view is the main focus of this project. Other researchers have employed a number of techniques to achieve this aim (often with objects other than just the hand) with varying degrees of success. The methods used include the following.

Bowden et al. [5] have used a non-linear PDM to recover the 3D pose of a human torso (head, arms, body) based on a 2D view from a single camera. The method they used was to incorporate both 2D and 3D data into the vector defining the pose of each training example. This hybrid data was then used to produce a linear PDM using the method described in section 2.2.2.

The non-uniform distribution of training points in shape space was then analysed by a *k*-means clustering algorithm and the clusters identified were individually subjected to PCA. The results of this process were used to constrain the model.

After building a model using this technique with the hybrid 2D/3D data, the model was presented with an unfamiliar 2D object and allowed to deform[2] within the above constraints to fit the 2D elements of the model to this shape. As the 2D elements of the model deformed, so the 3D elements also deformed and – the authors suggest - produced the likely pose of the object in three dimensions.

This technique claims some success although the 3D data used to build the PDM was generated manually by *estimating* the *z* co-ordinates from a visual examination of each image frame. As the authors have noted, this lack of ground truth makes it difficult to make an objective evaluation of performance.

Di Bernado et al. [9] have taken a different approach to tracking a human arm (shoulder, forearm, hand position) in 3D using a single camera. They have developed software to model an arm in 3D and use an initial estimate of arm position to predict the arm projection in the image. The actual image from a single video camera is analysed to segment the arm from the background and the difference between the *predicted* arm image and the *actual* arm image is used as an error measurement to update the estimated arm position using a recursive estimator. This technique works well for a human arm where the area of interest is relatively easy to identify and there are no significant problems with occlusion.

Shimada and Shirai [29] and Ouhaddi and Horain [20] have attempted to apply modified versions of the above technique to the human hand. Both of these have

[2] As an active shape model (ASM) – see [8]

been hampered by the effects of occlusion in the image sequences resulting in inaccurate or ambiguous results in many cases.

Rehg and Kanade [26, 27, 28] use *multiple* cameras to provide three dimensional real time tracking of a hand model although this also suffers from problems caused by occlusion and cluttered backgrounds (remember, the intention for *this* project is to use a *single* video camera).

The proposed method for approaching this project is initially to use synchronised 2D images (from a single video camera) and 3D data (from magnetic positional sensors) of various single-hand gestures, to produce a hybrid PDM in a similar manner to [5]. This is described in more detail in the following chapter.

# Chapter 3 Methods

## 3.1 Overview

### *3.1.1 Introduction*

This chapter describes the methods used and software developed to capture and analyse the information we believe is required to further the aim of this project. These developments fall broadly into two areas; raw data capture and statistical modelling.

### *3.1.2 Data capture*

Section 3.2 describes the development of a new data glove using magnetic positional sensors. This covers the design of the glove and the software developed to capture data both from the glove itself and from a 'synchronised' video camera. This data forms the raw input to the modelling phase.

### *3.1.3 Statistical models*

Section 3.3 describes the software developed to allow the production of hybrid 2D/3D PDM's using the captured data. The techniques used follow those outlined in section 2.2.5.

Section 3.4 describes the development of software for building, training and evaluating hidden Markov models. As discussed in the previous chapter, HMM's will probably play *some* part in this project and this development results from that assumption.

## 3.2 Developing a new data glove

### *3.2.1 Magnetic positional imaging system*

The system used for obtaining 3D point data from the new data-glove was originally developed by Dr. John Bladen and colleagues for producing images of medical endoscopes [2, 3]. The system consists of a number of passive single-coil sensors placed in a generated magnetic field. As the characteristics of the generated field are altered by the positioning system, the responses of each of the sensor coils are analysed to derive the position and orientation of those sensors ($x, y, z, \theta, \phi$).

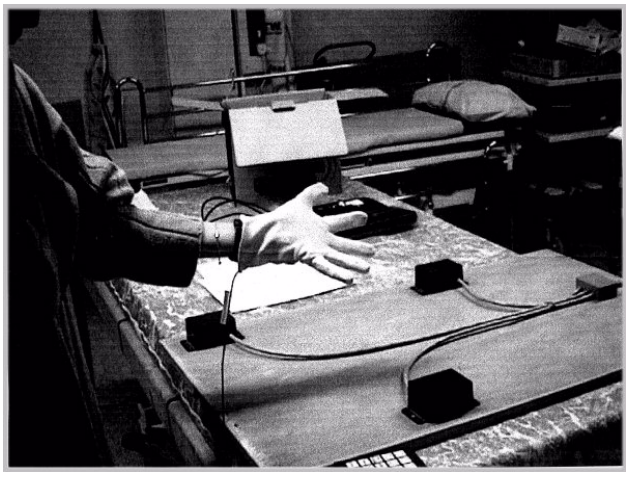

**Figure 3-1 - Prototype data glove**

In the data-glove application (Figure 3-1), a cotton glove is fitted with a number of sensors connected to the Bladen positioning system. The positioning system powers three magnetic field generator assemblies, each consisting of three orthogonally wound coils. These generators are fixed in a triangular configuration on a wooden board, over which the action must take place. Whilst this is physically a more restrictive environment than that possible when using a conventional data-glove, the system has been shown to have a high degree of accuracy [3, ch8] and, once set up, suffers no need for repeated re-calibration whilst in use.

The first prototype data-glove system using the Bladen positioning system is described in [10]. Some later developments, including an alternative medical application are described in [1] and [34].

The current data-glove uses eight sensors. Three of these are used to define the plane of the palm and the remaining five are placed one on the tip of each digit.

The use of just eight sensors per hand was envisaged as an ideal minimum, although certain assumptions are necessary about the way the hand moves if a realistic image of a hand is subsequently to be rendered by a computer.

Figure 3-2 shows the approximate positions and sizes of the sensor coils used in the current version of the data glove. All of the sensors are placed on the dorsal (back) surface of the hand.

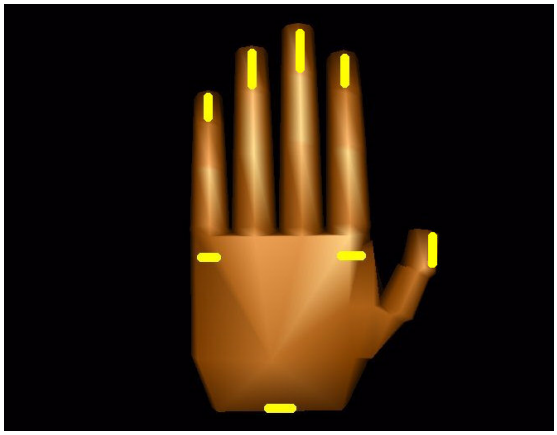

**Figure 3-2 – Data glove sensor layout**

### *3.2.2 Building a model of the hand*

#### Motivation

The need for a computer application to produce a 3D image of a hand is two-fold. Firstly, there is the need to validate the performance of the Bladen positioning system and data-glove. This is more easily performed at a subjective level if a visual image of a hand can be displayed in real time as the data-glove is used.

Secondly, such an application may be used to produce hand-shapes from data generated by a statistical model, for example a point distribution model. This would similarly assist the validation of any such PDM, which may potentially be useful both in gesture recognition and in 'driving' an avatar from an alternative source of data (e.g. text, speech).

#### Perceptual Control Theory

With the current version of the data glove, only the palm and fingertip positions are directly measured by sensors. The positions of individual articulated joints within each finger are not known. This does not give sufficient information to allow a 3D model of the hand to be rendered as a realistic image without additional processing.

In an attempt to simplify the calculation of joint angles from just the finger-tip positions, a number of ideas derived from Perceptual Control Theory (PCT) have been applied. This theory has its roots in the area of biological behaviour [21] although little published work for *this* type of simulation is available. Dr. Richard Kennaway at UEA has developed a simulation of six-legged insect using PCT and Java which performs remarkably well without the need for inverse kinematic calculations [18]. Dr. Kennaway's development uses a hierarchy of PID (proportional – integral – differential) controllers, where the higher levels send their output to the next level down and only the bottom level controllers affect the joint angles.

In the hand model developed for this project, the PCT implementation consists of a number of feedback controllers which incrementally adjust joint angles until the desired position of each fingertip is achieved. Unlike Dr. Kennaway's simulation, there is currently no hierarchy of control and this is possibly a drawback. The following sections describe this software in more detail.

#### 3D Hand Software

In order to facilitate integration with other development platforms, the hand imaging software has been implemented as an ActiveX control. This allows the graphic display of a hand and manipulation of its size and shape from within other Windows 9x/NT applications.

At present, the ActiveX version of the control has been tested using MS VC++, MS Internet Explorer, MS Visual Basic, MS Visual J++ and MATLAB.

The ActiveX control is written in Microsoft Visual C++ using the Active Template Library (ATL) and is accessed from other development environments as a COM object (Component Object Model) using the *IUnknown* and/or *IDispatch* interfaces.

Figure 3-3 overleaf shows the ActiveX control being used in a MATLAB session.

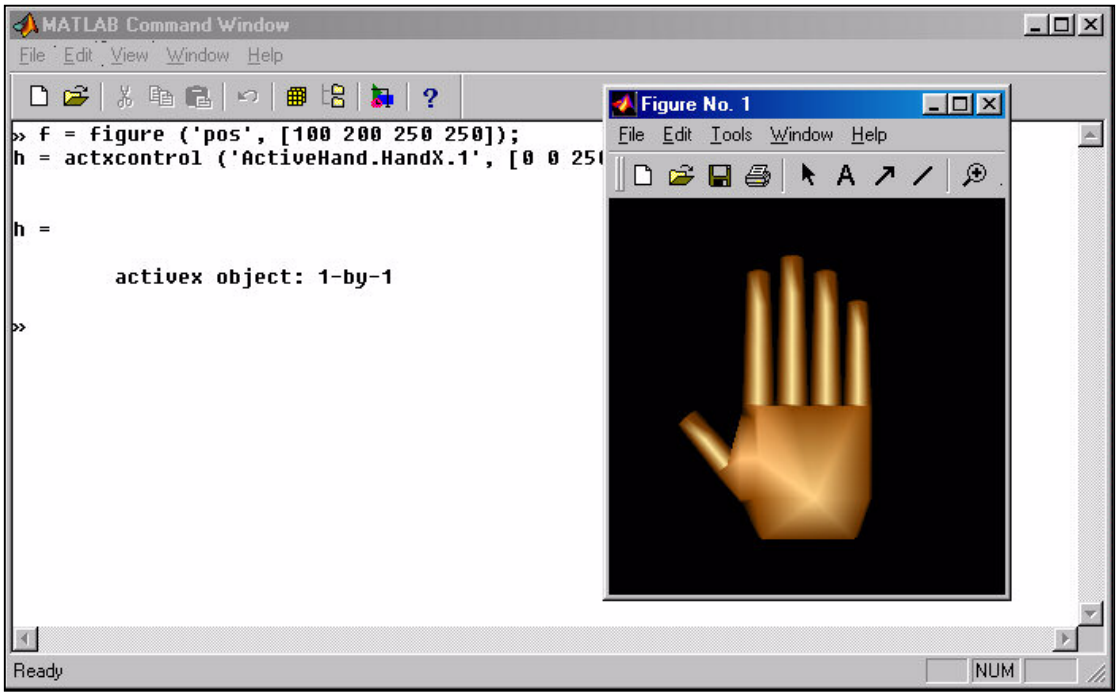


**Figure 3-3 – ActiveX Hand control in use by MATLAB**

### Software Design

The hand imaging software has been developed using an object-oriented design. OpenGL is used for rendering the graphics. The application utilises three main classes;

*CHand* Implements the complete hand model as a collection of vertices compatible for use with OpenGL. Class members include a collection of *CFinger* objects, each of which may be accessed through methods exposed by *CHand*. The *CHand* class handles generation of all surface normals and polygon definitions for the complete model and is capable of drawing itself in an OpenGL display list.

*CFinger* This class implements a single finger. A finger is able to calculate a set of vertices describing its configuration based on its current size and joint angles. These vertices are calculated relative to the finger's local origin and are made available to the owning *CHand* object. The finger also implements methods used for automatically

adjusting joint angles in order to place the fingertip at a given point relative to the local origin. This is used for tracking a moving sensor with the fingertip using techniques adapted from PCT.

*CThumb* A sub-class of *CFinger*, this class provides overridden methods for vertex generation and joint movement specific to a thumb.

The application also requires a higher level class, containing a *CHand* object, which implements the OpenGL interface and provides the means for the hand to be 'driven' in some fashion. This class (*ChandX*) provides the COM interface handling necessary for the application to operate as an ActiveX control. A test-bench application has been developed which drives the ActiveX control via GUI elements and/or from sensor data from a captured data-glove session. Figure 3-4 shows a screenshot from the completed hand imaging application.

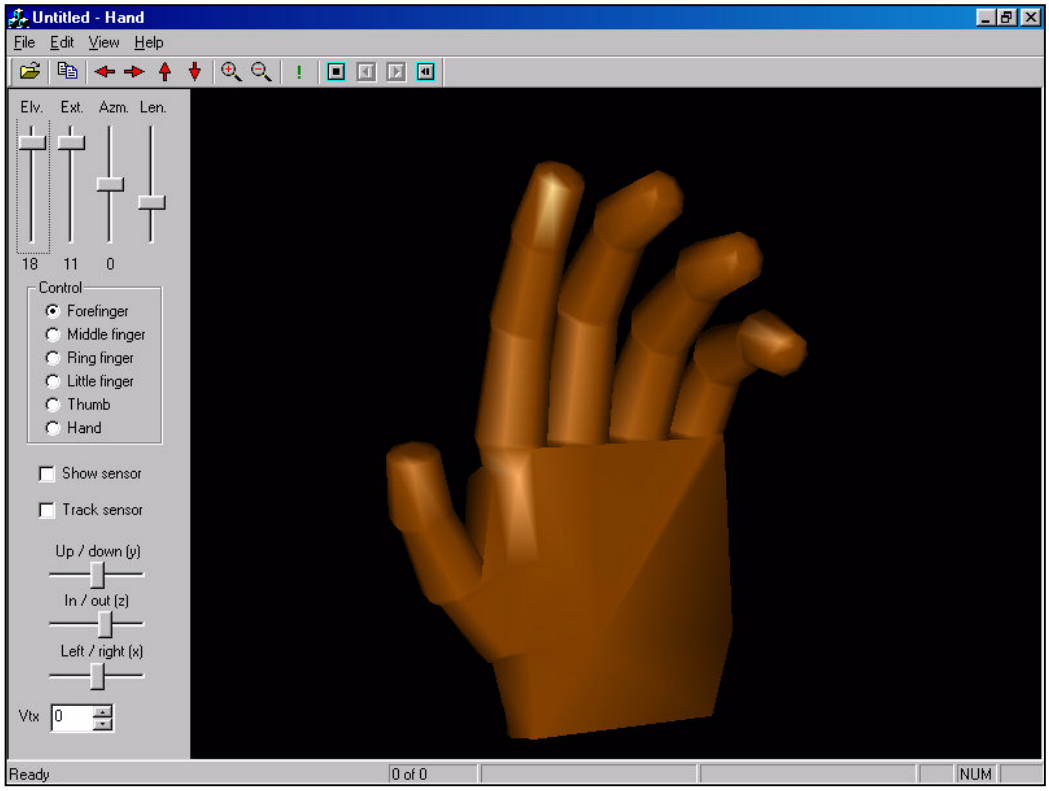


**Figure 3-4 – Hand imaging application**

#### Fingertip sensor tracking

Each finger has three PID controllers (at present these are error-integrators). One controls the finger extension by varying the top (distal) two joint angles, the second controls the finger elevation by varying the bottom (proximal) joint angle and the third controls the finger azimuth by varying the abduction angle.

These controllers operate together to bring the finger-tip to the position defined by the relevant sensor. The range of allowable angles for the finger joints permit the use of linear approximations for trigonometrical functions, which makes the controller functions very small and fast.

The only assumption made in using this technique is that the two distal-most joints of the finger tend to operate together when the finger bends in 'normal' usage. This assumption is based on observations of the movement of real fingers as there is little theoretical evidence to support this as a general principle. However, by adopting this assumption as a first model, these angles are equalised and together control the finger extension.

When used to track actual sensor positions provided by the data glove, some limitations have become apparent with this method of tracking. The model does not accurately reflect the shape or dynamics of a real hand. In the model, fingers bend in a plane normal to that of the palm and around axes parallel to the palm. In a real hand, this is not the case [17]. It is likely that the axis of rotation of finger joints changes slightly depending on the degree of inflexion. This and similar mismatches between the model and the real world means that sensor movements

are ‘seen’ which cannot be mirrored by the more constrained options provided by the model. As a result, the PCT controllers often appear to ‘jump’ from one configuration to another if the path of a fingertip sensor moves too far from that permitted by the inbuilt constraints.

#### Multi-threading

The hand imaging control is multi-threaded. The main thread of execution (the foreground thread) handles the interaction with the GUI and host application and performs most of the OpenGL interaction. Each fingertip control function is executed in a separate worker thread (background thread). These threads remain suspended until fired by an event from the *CFinger* object. This occurs in response to a *PointAt* message received by the owning *CHand* object. Once a controller thread becomes active, it will broadly perform the following actions:

1. Calculate elevation (*tgtElv*) and azimuth (*tgtAzm*) angles to the target point using a linear approximation.
2. Calculate the distance (*tgtExt*) from finger base to the target point.
3. Calculate elevation (*tipElv*) and azimuth (*tipAzm*) angles to the finger tip using a linear approximation.
4. Calculate the distance (*tipExt*) from finger base to the finger tip (i.e. the extension).
5. Modify proximal joint angle based on elevation error (*tgtElv – tipElv*).
6. Modify two distal-most joint angles based on extension error (*tgtExt – tipExt*).
7. Modify abduction angle based on azimuth error (*tgtAzm – tipAzm*).

8. If the fingertip has now moved by more than some threshold value, return to step 3.
9. Otherwise, activate Hand rebuild thread and then re-enter suspended state.

These five controllers thus work in the background, attempting to adjust joint angles in order to make the finger tips hit the target sensor positions as these are supplied. The controller threads contain logic to enable their operation to be interrupted (i.e. new target position received before current target reached) or terminated (i.e. application closed).

The remaining background 'worker' thread (activated in step 9 above) causes the whole hand to rebuild itself based on the new configuration of the finger(s). This results in the re-creation of the OpenGL display list containing the vertex and polygon data. Once this has been done, this thread fires a repaint in the main foreground thread to display the result on screen using the OpenGL interface.

Table 3-1 summarises the responsibilities and interactions of all the application threads.

| Description | Number of threads | Responsibilities |
|---|---|---|
| Foreground thread | 1 | User interface, application initialisation and termination, screen painting, manage OpenGL interface. |
| Hand rebuilder | 1 | Collect model vertices, calculate surface normals, build OpenGL display list. Fire repaint in foreground thread. |
| Fingertip tracking | 5 | Recalculate finger joint angles. Fire hand rebuild. |

**Table 3-1 – ActiveX Hand execution threads**

### *3.2.3 Capturing 2D and 3D data*

#### Interface with the positioning system

The Bladen positioning system is supplied with a COM (Component Object Model) interface which allows an application program to manipulate hardware calibration data and to interrogate sensor positions and orientations.

Two systems have been developed for capturing data from the glove using this interface. The first and simplest of these is designed to operate as quickly as possible and save just the 3D sensor information. This runs as a local application on the positioning system computer itself. The second system captures both 3D sensor data from the positioning system *and* 2D images from a USB (Universal Serial Bus) video camera. Due to the excessive overhead of running the video capture *and* positioning algorithms on a single processor, this system operates on two PC's communicating via a peer-peer network connection.

The two data capture systems are described in more detail in the following sections.

#### 3D Data Capture

The 3D data capture system (*GrabData*) runs on the positioning system computer. It is written in Pascal using the Borland Delphi compiler and uses OpenGL to display the eight sensor positions on the display during capture. Using this application, the capture rate is typically between 8 and 10 fps (frames per second). Figure 3-5 shows this diagrammatically.

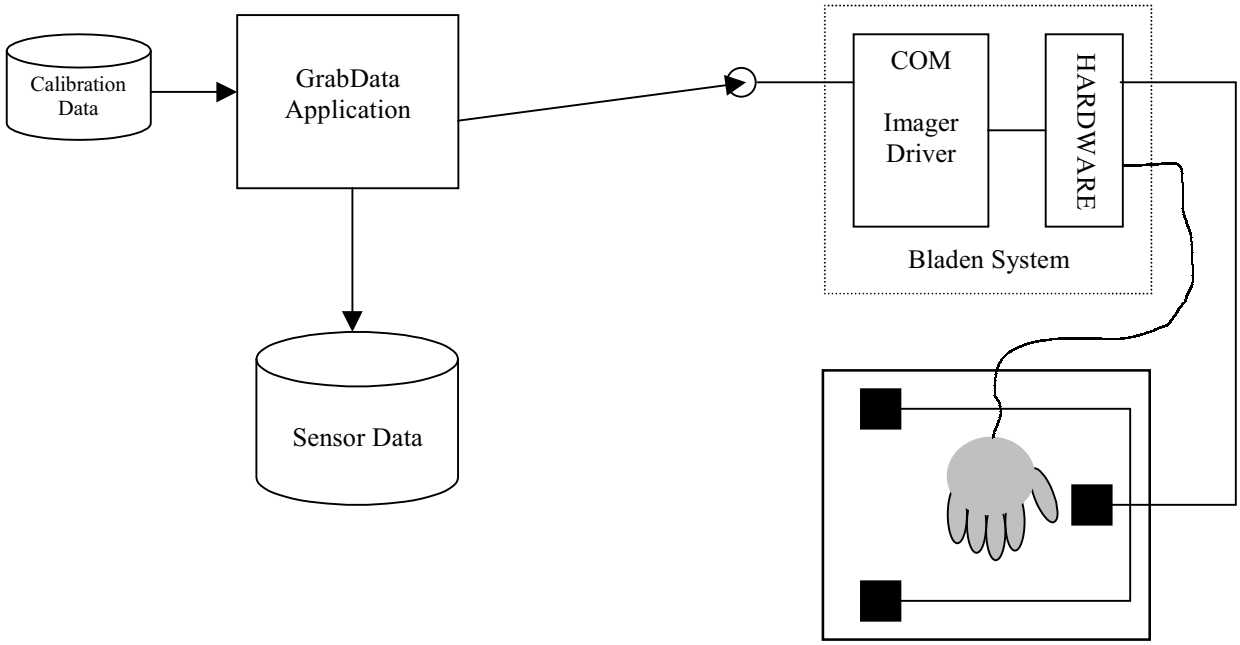


**Figure 3-5 – 3D Data Capture schematic**

The data format saved by the *GrabData* application is as follows:

The first entry in each file consists of 6 double-precision floating point values, being the $x$ and $y$ co-ordinates of the three generator assemblies (they are assumed to lie in the plane $z = 0$). These points are useful for establishing the initial size of the scene during subsequent off-line inspection.

Following the header record are a variable number of fixed-length records containing sensor data. Each record holds the position and orientations of all 8 sensors in a similar format to that returned directly by the positioning system.

The following tables show the details of these records.

**File Header – 1 per file, 48 bytes starting at byte 0**

| Offset | Length | Type | Description |
|---|---|---|---|
| 0 | 8 | double | Generator 0 – x co-ordinate |
| 8 | 8 | double | Generator 0 – y co-ordinate |
| 16 | 8 | double | Generator 1 – x co-ordinate |
| 24 | 8 | double | Generator 1 – y co-ordinate |
| 32 | 8 | double | Generator 2 – x co-ordinate |
| 40 | 8 | double | Generator 2 – y co-ordinate |

**Sensor Data – n per file, 456 bytes each, starting at byte 48**

| Offset | Length | Type | Description |
|---|---|---|---|
| 0 | 4 | DWORD | Timestamp – milliseconds from start |
| 4 | 8 | double | Sensor 0 – x co-ordinate |
| 12 | 8 | double | Sensor 0 – y co-ordinate |
| 20 | 8 | double | Sensor 0 – z co-ordinate |
| 28 | 8 | double | Sensor 0 – theta orientation (radians) |
| 36 | 8 | double | Sensor 0 – phi orientation (radians) |
| 44 | 8 | double | Sensor 0 – psi orientation (not used) |
| 52 | 1 | BYTE | Sensor 0 – Valid (0 if invalid) |
| 53 | 7 | BYTE | Filler |
| 60 | 56 | struct | Sensor 1 position / orientation as above |
| 116 | 336 | struct | Sensors 2 to 7 position / orientation |
| 452 | 4 | BYTE | Filler |

**Tables 3-2a and 3-2b – 3D Data Capture File Format**

#### 3D Data Review

Data captured by the *GrabData* application can be examined off-line by using the *TrackView* application. This application is written in MS VC++ and uses OpenGL to render a 3D view of the captured sensor data.

Figure 3-6 overleaf shows an example of the *TrackView* display. In this example, the generator positions are shown in blue, the fingertip sensors in green and the palm sensors in red.

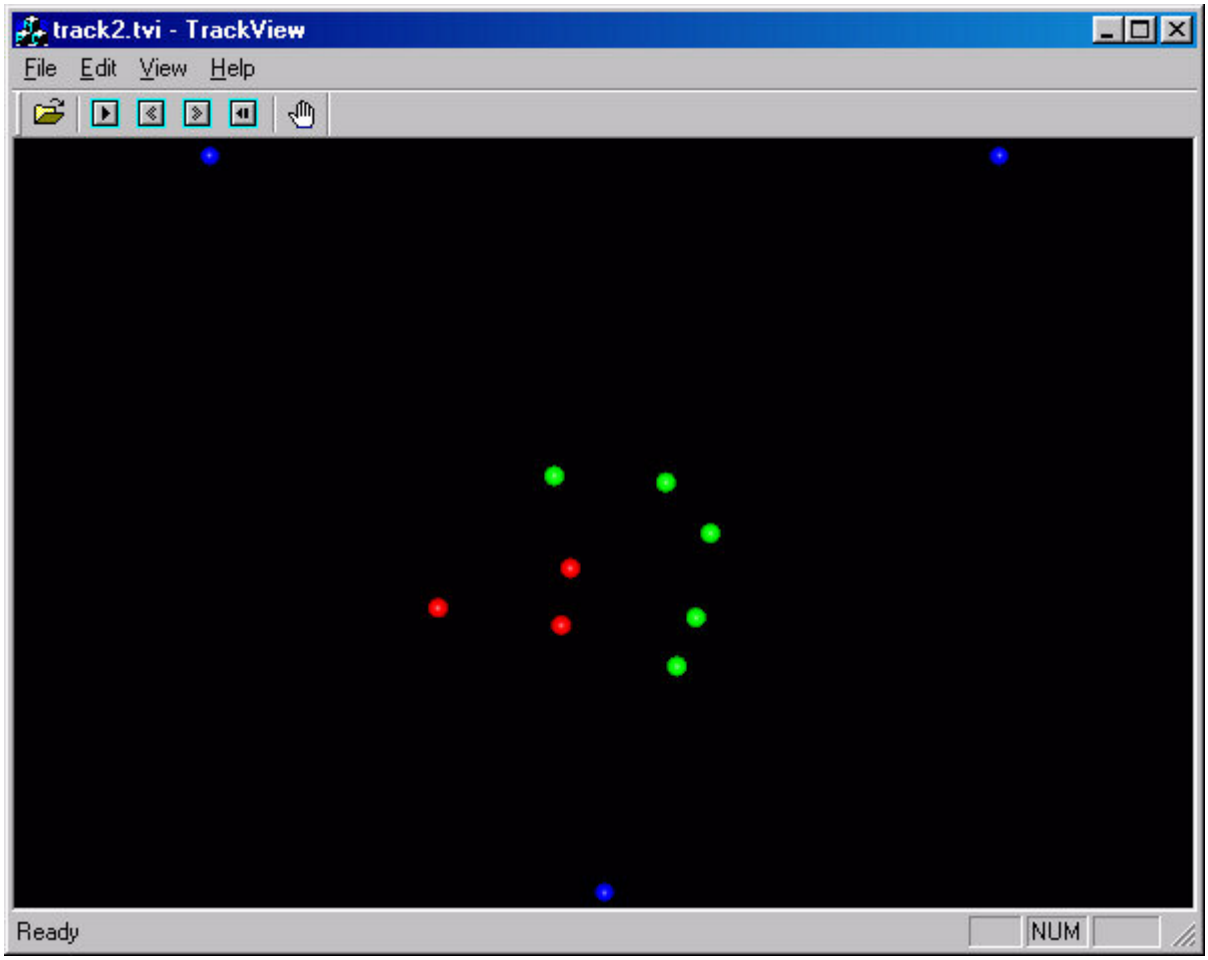


**Figure 3-6 – 3D Data Capture Review**

The application allows the captured data to be replayed continuously or stepped forward or backward frame by frame.

This application also implements a 'virtual trackball' which allows the scene to be rotated in three dimensions using the mouse. A menu option allows display of an optional trace of a configurable number of previous frames, so giving a visual indication of speed and movement. There is also a menu option providing the ability to export the captured positional data to a CSV (comma separated variable) file.

### 3D and 2D Data Capture

This system for data capture consists of two parts. The first of these (*ImagerServer*) runs on the positioning system computer. This application communicates on the one hand to the COM object controlling the positioning system hardware and on the other to a Windows Socket connection to provide the ability to converse using TCP/IP with a second computer over a local area network.

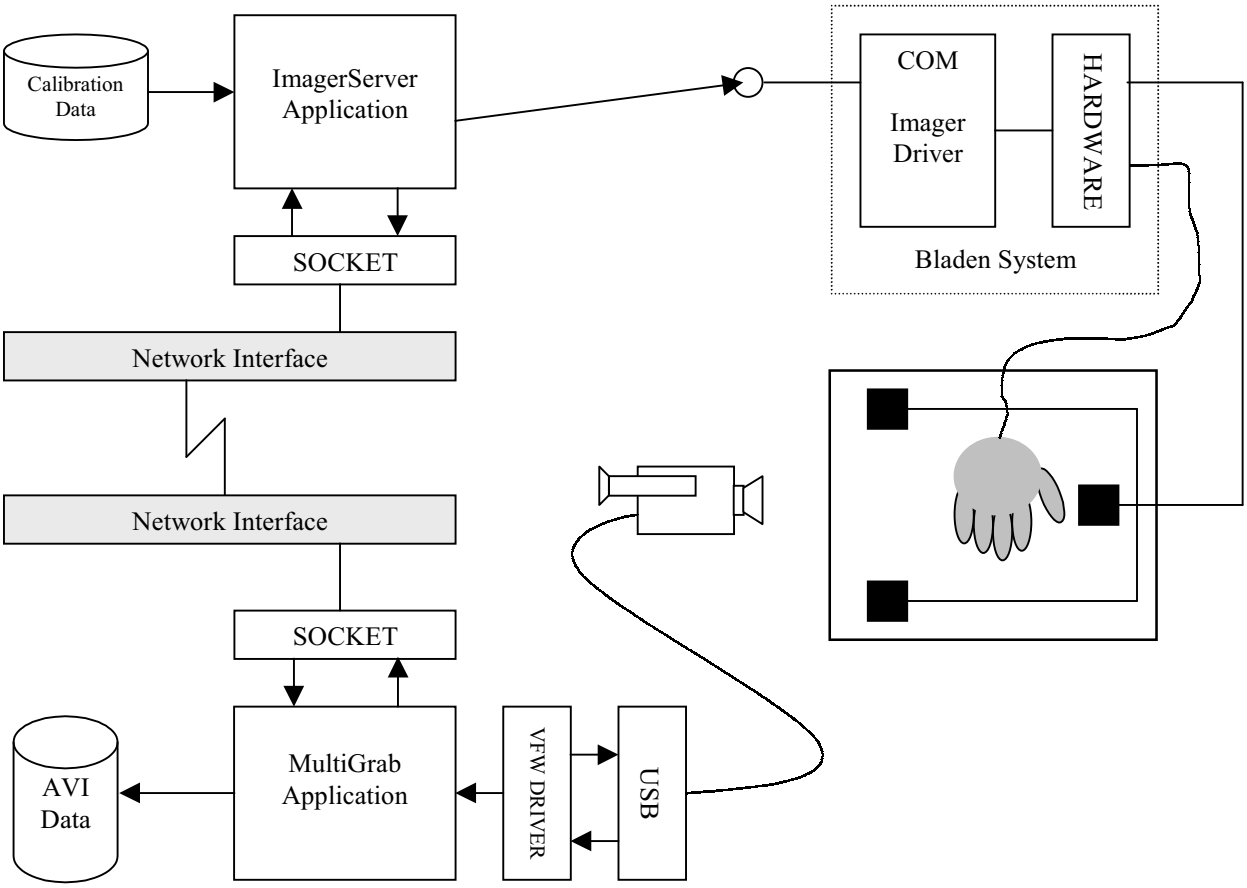


**Figure 3-7 – "Synchronised" 3D and 2D Data Capture**

The second part of this system (*MultiGrab*) runs on the aforementioned 'second' computer to which is also attached a USB video camera. This application implements a Windows Socket connection to receive the sensor data from the

*ImagerServer* on the positioning system computer and has an interface to the Video Camera via a Video For Windows (VFW) driver. The complete set-up is shown schematically in Figure 3-7.

The component parts of this system work together as follows –

After initialising the positioning system using stored calibration data, the *ImagerDriver* application instructs the positioning system to begin sampling. It then opens a socket and waits in 'listen' mode for an incoming connection.

On the second PC, the *MultiGrab* application initialises the VFW interface and displays a video picture in preview mode. A menu option initiates the socket connection to the positioning system PC. *ImagerServer* accepts the connection and awaits instructions.

Having established a connection to the *ImagerServer*, *MultiGrab* requests details of the generator positions to initialise the OpenGL scene and display the generator positions on screen. *ImagerServer* supplies this data from the calibration file used to initialise the positioning system.

*MultiGrab* now maintains a conversation with *ImagerServer*, repeatedly requesting sensor data and displaying the results. When *ImagerServer* receives such a request, the positioning system is instructed to pause sampling and return the sensor data. This data is transmitted to *MultiGrab* and the positioning system resumes sampling. On receipt of the sensor data, *MultiGrab* stores the sensor

positions in member variables, issues a request for more data, then refreshes the display to show the latest positions. This process occurs in a separate thread to the Video preview which is active concurrently.

When *MultiGrab* is instructed via a menu option to begin recording, the Video preview is stopped and a call-back function is used to intercept each video frame as it is received. This function writes the video data along with the currently active sensor data to an AVI (Audio Video Interleaved) file.

The AVI file contains two streams. The video stream is written in 24-bit RGB colour uncompressed format intended for playback at 30 fps. The 3D data stream, while updated less frequently (~10 fps) is written at the same time so each video frame has a corresponding 3D data record. Due to the difference in sampling rates and the lag involved in communicating with *ImagerServer*, the 2D and 3D data are not precisely synchronised although any such discrepancies are difficult to observe when the data is replayed.

The format of the 3D data stream is kept as simple as possible. Each frame consists of 30 single-precision floating point values. The first 6 of these contain the (*xy*) co-ordinates of the generators and the remaining 24 contain the (*xyz*) co-ordinates of the 8 sensors. The generator details are not necessary on each frame but this does make it easier to subsequently view individual frames as all the required 3D data is easily available.

Table 3-3 overleaf summarises the layout of the data in the 3D data stream.

| Offset | Length | Type | Description |
|---|---|---|---|
| 0 | 4 | float | Generator 0 – x co-ordinate |
| 4 | 4 | float | Generator 0 – y co-ordinate |
| 8 | 4 | float | Generator 1 – x co-ordinate |
| 12 | 4 | float | Generator 1 – y co-ordinate |
| 16 | 4 | float | Generator 2 – x co-ordinate |
| 20 | 4 | float | Generator 2 – y co-ordinate |
| 24 | 4 | float | Sensor 0 – x co-ordinate |
| 28 | 4 | float | Sensor 0 – y co-ordinate |
| 32 | 4 | float | Sensor 0 – 7 co-ordinate |
| 36 | 12 | struct | Sensor 1 position as above |
| 48 | 72 | struct | Sensors 2 to 7 positions as above |

**Table 3-3 – Combined 2D/3D Data Capture Format**

### 3D and 2D Data Review

The combined 2D and 3D data captured by *MultiGrab* may be reviewed offline by the *MultiView* application. Again, this is written in C++ and uses OpenGL and VFW (Video for Windows) to assist the display of the combined data.

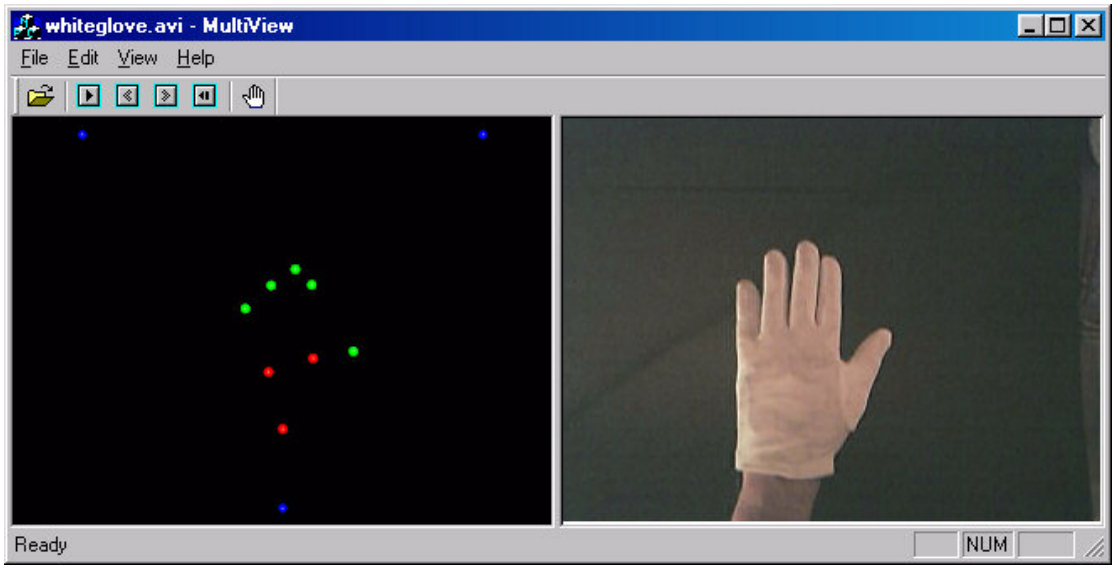


**Figure 3-8 – 2D/3D Capture Review**

Simple toolbar controls allow the data to be inspected frame by frame or for a captured sequence to be replayed at normal speed. As with *TrackView*, a virtual trackball may be used to examine the 3D sensor positions from any viewpoint.

The combined 2D and 3D data captured using this technique are used to build point distribution models (PDM's) to attempt to model the characteristics of particular hand shapes and movements. This is described below.

## 3.3 Modelling hand shape

### *3.3.1 Hybrid point distribution models*

With the ability to automatically capture reasonably synchronised 2D and 3D data from the new data glove, an obvious step was to attempt to reproduce this work of Bowden et al. [5] to assess the effect that the availability of accurate 3D point data might have on such a model. The software developed to pursue this course is described in detail in the following sections.

### *3.3.2 Building hybrid PDM's*

The software developed for the PDM functions is all written in MS VC++ and uses OpenGL for display of any 3D information. Three separate applications were developed, primarily to assist testing of each stage but also to allow hand-coded data to be introduced at various points to permit models to be built and manipulated using data from other sources or systems.

The three PDM applications provide the following functionality:

- Selection of 3D/2D frames for use as training data
- Building a PDM from training data
- Display and manipulation of shapes generated by a PDM

#### Selecting training data

The application developed to selected individual frames to form the training data is called *PDMSelect*. This uses as input an AVI file produced by the *MultiGrab* application described earlier.

*PDMSelect* allows individual frames to be selected as training examples for the PDM. Each selected frame is written out as a separate bitmap to a destination folder chosen at run time. Along with this set of bitmaps, a control file (*Control.txt*) is placed in the same folder. This file contains details of the 3D data points associated with each frame and is a text file in tab-delimited format.

Capturing data in this fashion allows the possibility of supplying data to the next stage from another source. Figure 3-9 shows *PDMSelect* in action.

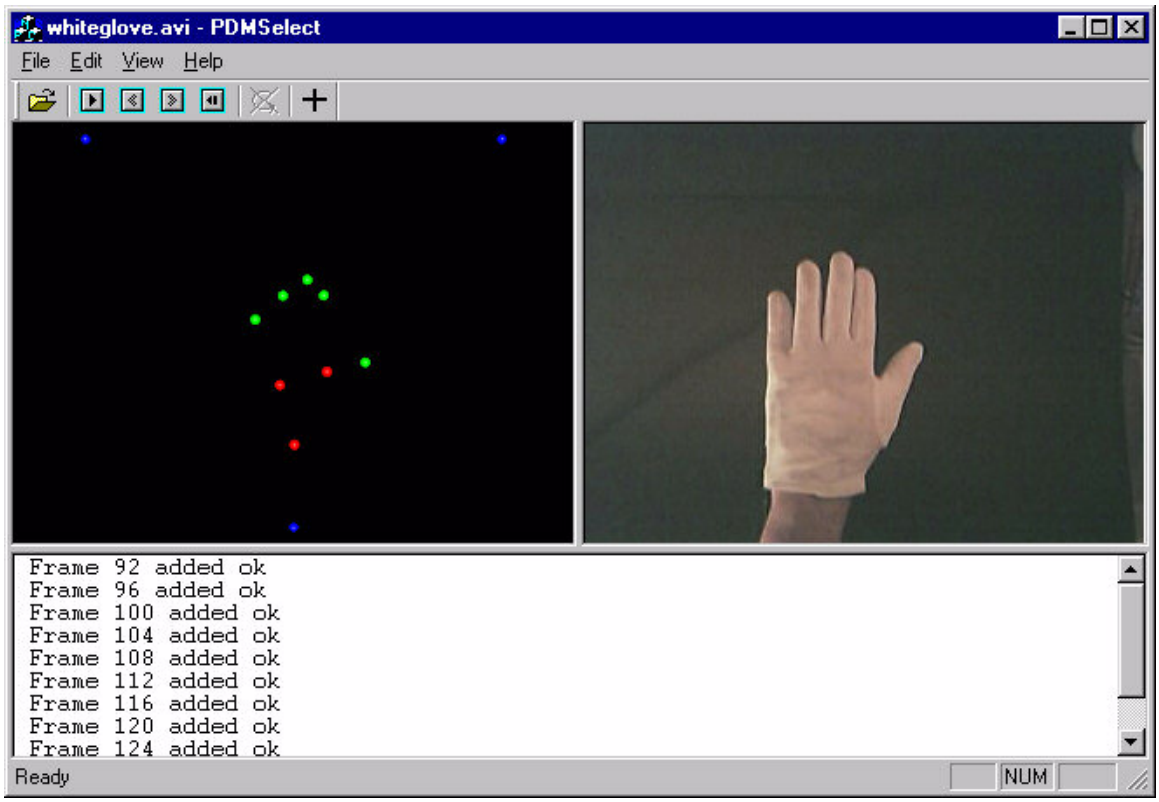


**Figure 3-9 – Selecting Training Data for the PDM**

Bitmaps are saved as uncompressed 24-bit RGB colour images 320 pixels wide by 240 pixels high. The tab-delimited control file contains one record for each sample, terminated by a CR/LF sequence. The fields in each record are as follows:

- Bitmap filename (*Framennn.bmp*) – name of the image file for this frame.
- Number of values – the number (*n*) of float values on this record
- Values – the 3D point values themselves in *x, y, z* order

When used with the *MultiGrab* data, there are always 24 values (i.e. x, y and z for each of 8 sensors) although the above format allows for different objects to be modelled if required.

#### Building the model

The application which builds the PDM from training data in the above format is called *PDMBuild*. This has a similar style to *PDMSelect* and implements the code necessary to produce 2D contours from the bitmap images and to perform PCA on the training set of combined 2D and 3D data by carrying out an eigenvector decomposition of the covariance matrix.

As the successful production of a 2D contour depends in part on the content and lighting conditions present for each frame, it is necessary to review the results for each training sample and, if necessary, adjust conditions to achieve a satisfactory result before the data is added to the training set. Figure 3-10 overleaf shows a view of a successfully contoured image and the statistics produced once the calculation of the model has completed.

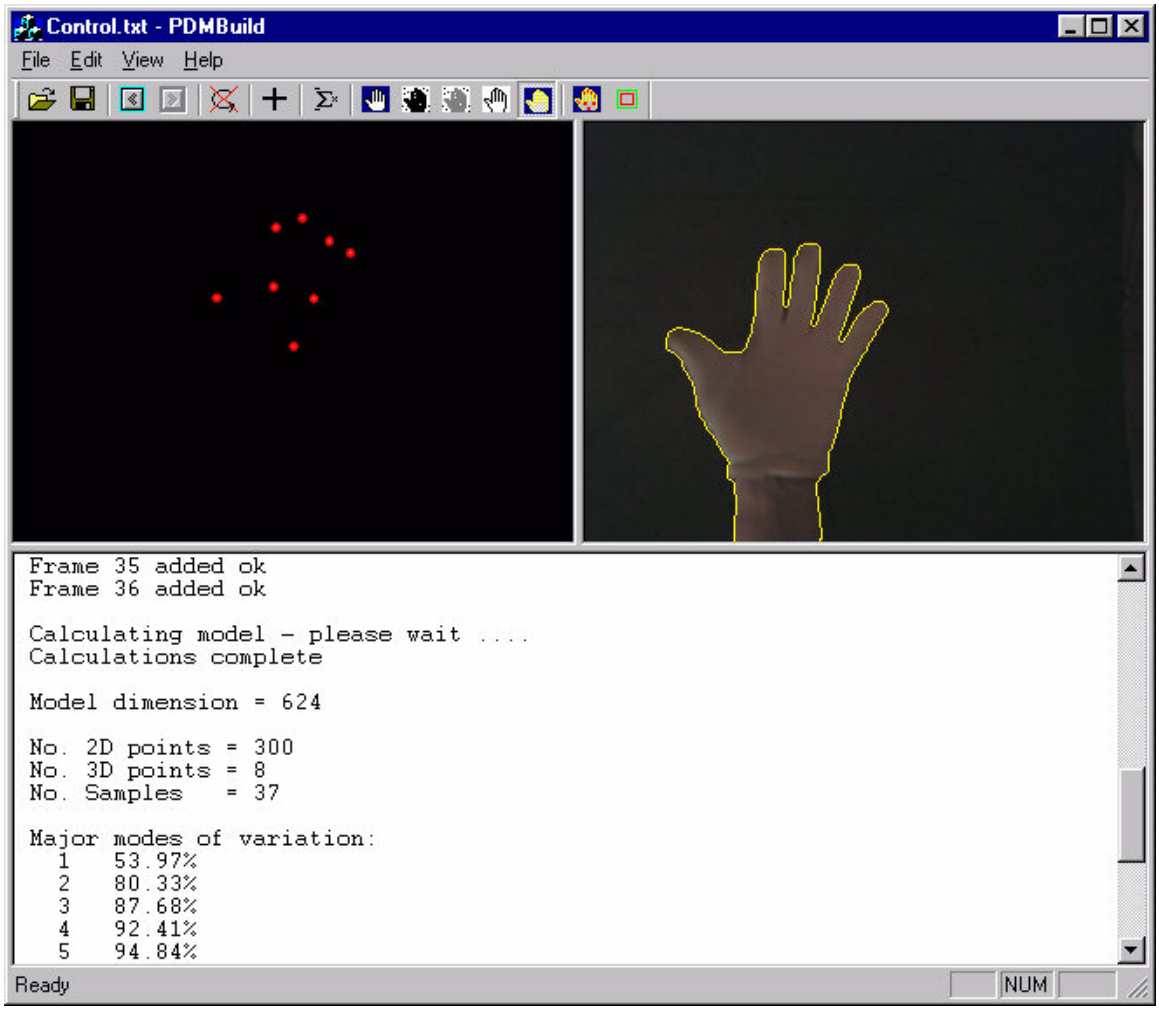


**Figure 3-10 – Building the PDM**

The process of producing a reasonable contour from the bitmap image involves several steps. These are as follows:

1. The RGB image is binarised using a simple thresholding algorithm based on the intensity of the red channel. This was found to be the most effective method of segregating the mainly white glove from the dark background. The threshold is adjustable using a modeless dialog box.

2. As a first step in removing any residual ‘noise’ from the binarisation, the position of the hand in the image is broadly located by identifying a large block of consecutive black pixels near the bottom of the image. Using this as a starting point, a flood-fill algorithm is used to initially isolate the hand shape as a grey region in the black and white image. Any areas untouched by the flood-fill are then set to white.

3. Next, to clean any residual noise from around the edges of the hand or fingers, a majority filter is applied using a square 3x3 window. This produces a reasonably smooth hand shape with no background noise.

4. A simple edge detection algorithm is next used to identify those areas where a grey/white transition occurs. Such points are marked black and any remaining grey is set to white. This results in a contour around the outside of the hand shape although this may still contain internal ‘edges’.

5. The final stage is a contour walking algorithm which, having identified the starting pixel of the contour in the bottom of the image, ‘rolls’ clockwise from pixel to pixel around the exterior of the hand shape until the bottom of the image is reached on the other side. The resulting contour is used as the basis for the 2D information in the training example.

The results of applying each of the above steps are shown in Figure 3-11 (a) – (f) overleaf.

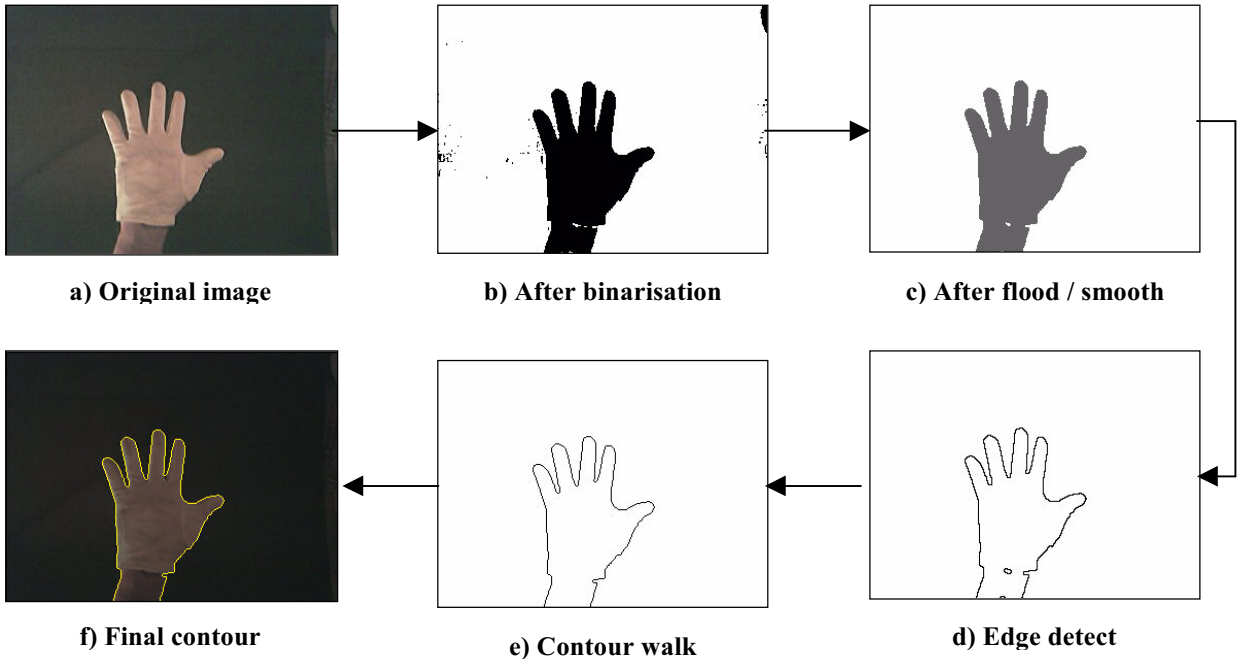


**Figure 3-11 – Producing a 2D Contour**

Figure 3-11 (f) shows the final contour superimposed over the original image to ensure that the contour is a fair representation of the original hand shape.

The points along the 2D contour are linearly resampled[3] for each training example in order to standardise their length. This is configurable within the application but is currently set at 300 (*x, y*) points per contour.

Each training vector is thus built up from a concatenation of the 24 values from the 3D data (8 sensors times x, y, z) and 600 values from the 2D data (300 points times x, y) which together identify a single point in a 624-dimensional space for each training example.

[3] As noted previously, this process will inevitably introduce non-linearity into the PDM as specific landmark points are not aligned between training examples. If this presents a problem, it may be possible to use a 2D projection of the 3D sensor data to identify points along the contour corresponding roughly to the implied positions of the (bent or unbent) fingers based on the positions of the fingertip sensors and to use these as more accurate landmarks.

Having assembled a training set of example vectors, the mean shape is calculated by finding the average vector and Principal component analysis (PCA) is performed on the deviation of the example vectors from the mean using eigenvector decomposition on the covariance matrix *S* of *E* as described in section 2.2.2.

As the dimensionality of the data (624) is high, the techniques described in section 2.2.2 have been implemented in *PDMBuild* and produce a 15-fold decrease in run-time for the calculation of a model with ~60 training examples. This calculation takes only a few seconds on a 800MHz Pentium III system with 64Mb RAM.

Once the model calculations are complete, *PDMBuild* sorts the eigenvectors into descending order of their eigenvalues and provides a graphical representation of the training data in shape space by projecting each training vector onto the first three eigenvectors to produce *x*, *y* and *z* co-ordinates. The 3D section of the display may be selected to show the original training sample or the complete shape-space projection by using a menu option.

An example of such a display is shown overleaf in Figure 3-12.

This feature is useful to gain an initial impression of the degree of non-linearity in any model as it is sometimes fairly obvious (as in the above example) that the training data are not uniformly distributed.

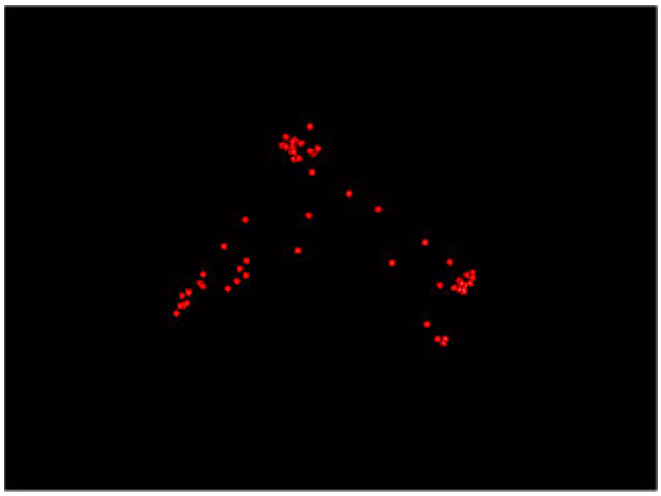

**Figure 3-12 – Training Data Projected from Shape Space**

As well as building a PDM from the *PDMSelect* output, *PDMBuild* also has an option to import combined or separate 2D and/or 3D data directly from a tab-delimited text file. This option was used to test the PCA calculations by feeding the program with generated data describing various poses of an anglepoise lamp. Whilst only a 2D model, this data is known to exhibit non-linearity due to its articulated structure and examples of the variation modes produced are available in a number of publications for comparison.

Having calculated the PDM from training data, *PDMBuild* can save the model in a compact form. The internally used format includes the mean shape, the eigenvector array and associated eigenvalues and also the original training data projections in shape space. This information may be used by the third of the PDM applications to generate and display shapes using the model. This is described in detail overleaf.

### Model display

The final PDM application is called *PDMDisplay* and takes as its input a model generated by *PDMBuild*. This application can handle models which include any combination of 2D and 3D information and is used to display generated shapes from the model. An example display is shown in Figure 3-13.

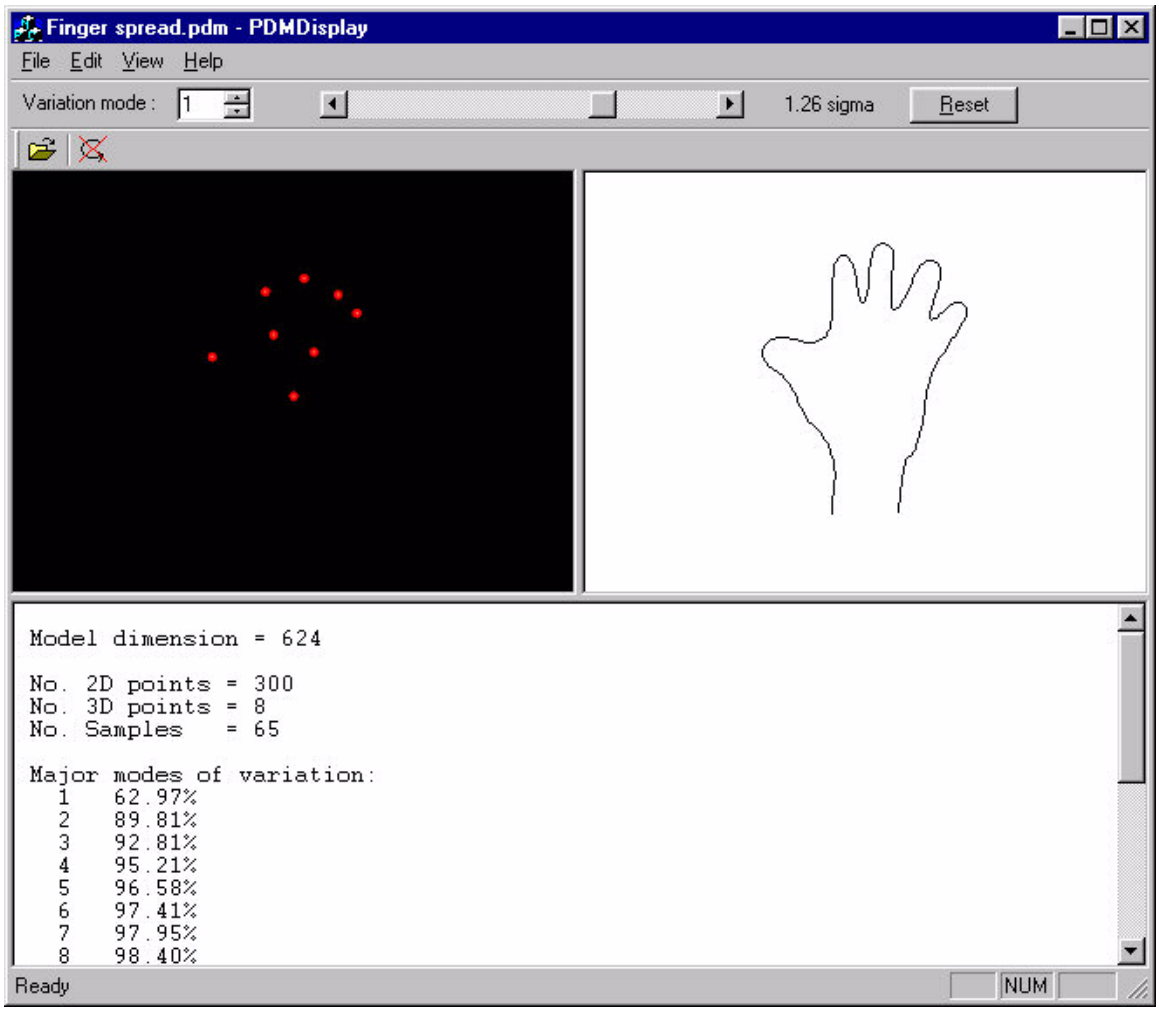


**Figure 3-13 – Generating Shapes from a PDM**

On loading a model, the familiar 3D / 2D split view initially displays the mean shape. A spinner control near the top of the display allows selection of the mode of variation and the adjacent slider permits the amount of variation to be set between ± 3 standard deviations.

In a similar manner to *PDMBuild*, a view of the training data projections in shape space may be selected to be shown in the left hand window. In such a display, the training points are shown in red and the corresponding position of the currently generated shape is shown in blue. By using the slider control to position the blue marker towards a training point, it can be seen that the 2D contour adopts a more plausible shape. Selecting a region of shape space in which there are no training examples always results in a badly deformed display. This is further evidence of the non-linear nature of the model.

*PDMDisplay* allows the variations of each mode to be set individually so that the cumulative effect may be observed in the generated shape. As the models built so far have large degrees of non-linearity, this application is so far only really useful for validating the correctness of the calculations.

## 3.4 Exploring hidden Markov models

### *3.4.1 Introduction*

As discussed in the previous chapter, HMM's are likely to feature at some point in this project. The motivation for developing the application described below, apart from being an exercise to fix an understanding of the principles, is to provide a method of creating, evaluating and manipulating HMM's in a user-friendly manner, primarily to prepare for applying these techniques to gesture recognition.

### *3.4.2 HMM Workbench Application*

This is a Microsoft C++ application using Microsoft Foundation Classes (MFC). It has been designed in such a way that it allows a "working set" of HMM's, observation sequences and algorithm outputs to be created and persisted.

The application provides implementations of the algorithms described in the previous section and also allows new observation sequences to be generated from existing models.

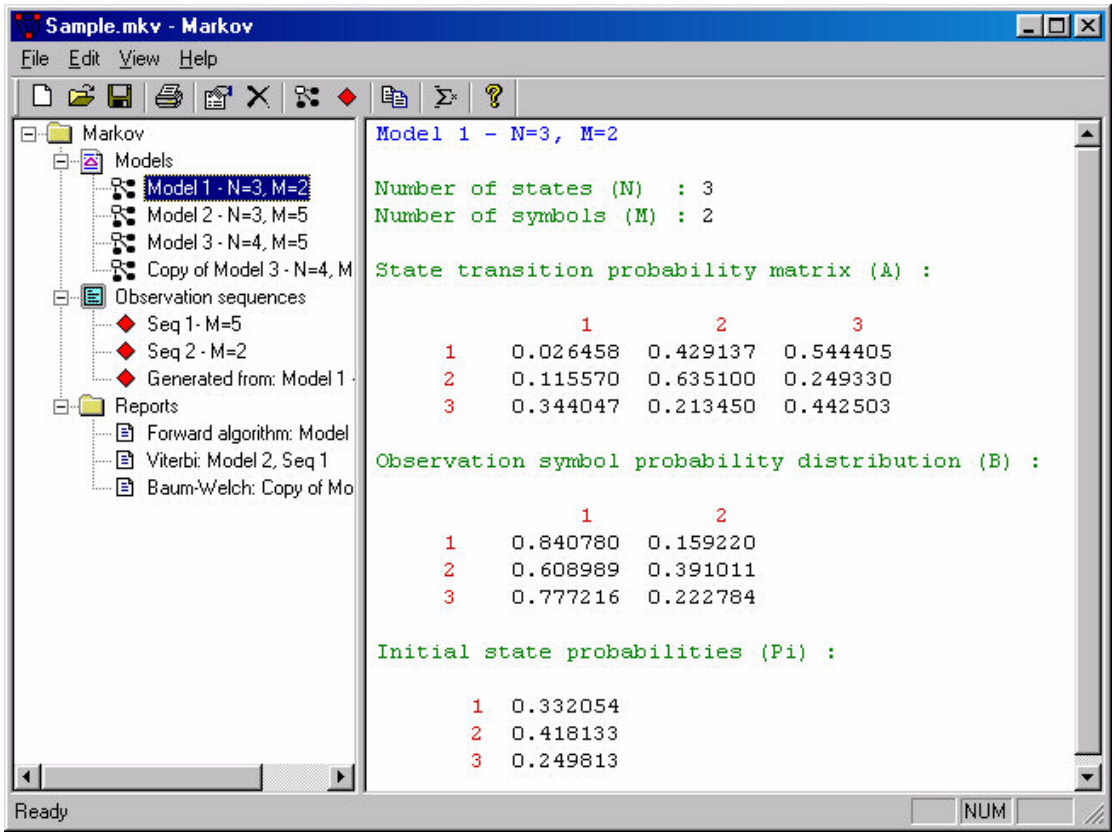


**Figure 3-14 – HMM Workbench application**

Rabiner [23, 25] details a number of implementation issues for HMM's, which affect the way the algorithms must be implemented in a computer application. Some of the techniques described have been implemented in the workbench application. Figure 3-14 shows an example of the application GUI.

#### Features

The HMM workbench currently provides the following basic features:

- Creation of HMM model parameters $\lambda$, with variable $N$ and $M$.
- Initialisation of probabilities using random values.
- Export of model parameters to MATLAB format .MAT file.
- Creation of random observation sequences $O_K$ with variable $K$, $M$ and $T$.
- Generation of sample observation sequences using a selected HMM.
- Implementations of *Forward*, *Viterbi* and *Baum-Welch* algorithms.
- On-screen or printed report of models, sequences, and algorithm results.

The implementations of the *Forward*, *Viterbi* and *Baum-Welch* algorithms use the scaling procedures described in [23].

The use of scaling, particularly in the forward and backward variables, is necessary to overcome the strong possibility that the limits of double-precision arithmetic will quickly be exceeded when calculating cumulative probabilities. For this reason, the final outputs from these algorithms are all calculated as *Log(P)* where $P$ is the probability. In this scenario, a lower number will imply a stronger probability.

Figure 3-15 shows the dialog used to apply the above algorithms to selected models and sequences.

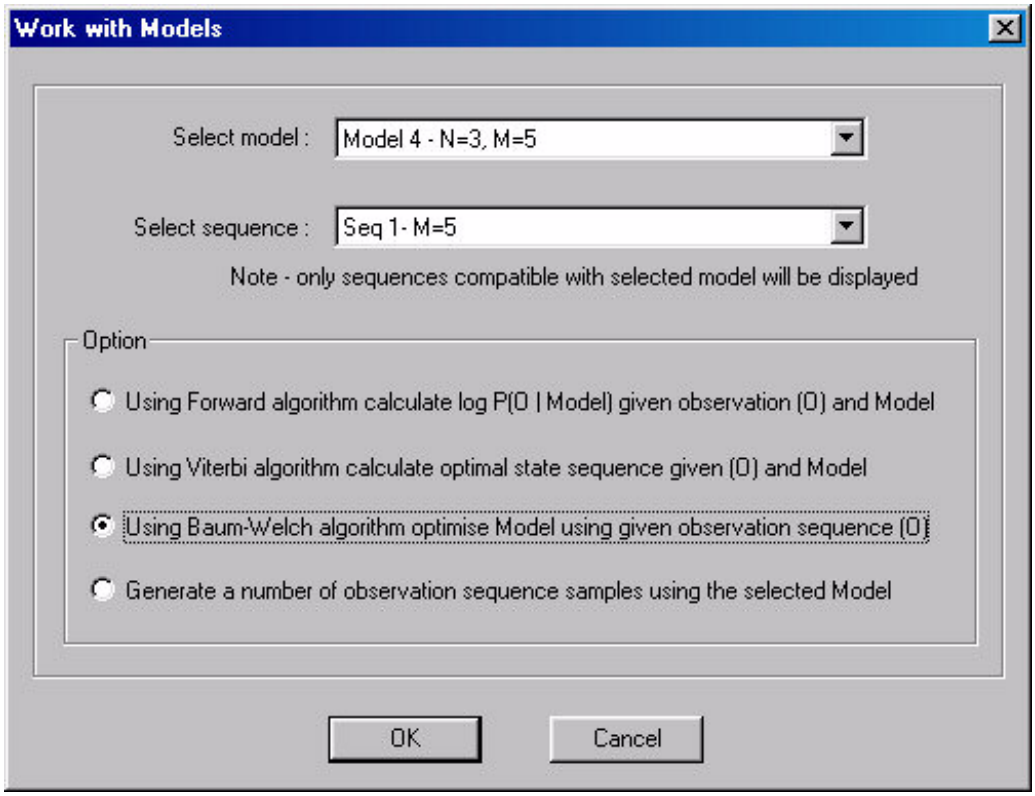


**Figure 3-15 – Applying HMM algorithms**

The report below shows a sample of the output produced after using the Baum-Welch algorithm to adjust model parameters based on an observation sequence.

```
Baum-Welch: Model 4 - N=3, M=5

Model parameters before optimisation:

Model 4 - N=3, M=5

Number of states (N)   : 3
Number of symbols (M) : 5

State transition probability matrix (A) :

              1         2         3
     1    0.191012  0.432064  0.376925
     2    0.327904  0.645133  0.026963
     3    0.328678  0.251862  0.419460

Observation symbol probability distribution (B) :

              1         2         3         4         5
     1    0.100160  0.280099  0.273195  0.159676  0.186870
     2    0.198439  0.164707  0.220740  0.098470  0.317643
     3    0.315664  0.010025  0.330917  0.103970  0.239424

Initial state probabilities (Pi) :

       1  0.211719
       2  0.138183
       3  0.650098

Initial log probability P(O|Model) = -17.349739
```

```
Final log probability P(O|Model)    = -11.060943
Number of re-estimation iterations = 19

Model parameters after optimisation:

Model 4 - N=3, M=5

Number of states (N)  : 3
Number of symbols (M) : 5

State transition probability matrix (A) :

              1         2         3
     1    0.001317  0.629388  0.371295
     2    0.506248  0.494747  0.001004
     3    0.001603  0.999390  0.001007

Observation symbol probability distribution (B) :

              1         2         3         4         5
     1    0.001001  0.627900  0.373092  0.001000  0.001006
     2    0.001278  0.001220  0.413292  0.001000  0.587210
     3    0.997783  0.001004  0.002074  0.001000  0.002139

Initial state probabilities (Pi) :

       1  0.001001
       2  0.001001
       3  0.999999

Generated sequence, M=5

Number of samples (K) : 1
Number of symbols (M) : 5

Sample k1:
Number of observations (T) : 12

           1  2  3  4  5  6  7  8  9 10 11 12
Symbols:   1  5  5  2  1  3  3  3  5  2  5  3
```

This report shows the model parameters before and after optimisation, the log probability before and after, and the observation sequence used for training.

#### Implementation

The main HMM related functionality of the application is provided by the following classes:

*CMarkovModel* - holds the model parameters $\lambda$ and the variables $N$ and $M$. Provides functionality for running the *Forward*, *Backward*, *Viterbi* and *Baum-Welch* algorithms given an observation sequence (*CMarkovSequence*). Also provides methods for

generating observations based on the model parameters and displaying itself on screen or printer.

*CMarkovSequence* - provides a way of holding a number of related observation samples (*CMarkovSample*). This class maintains the variable *M* (the alphabet size) and contains a collection of *CMarkovSample* objects. A method is provided to generate a number of random sample sequences of varying length. Also provides a method to display itself and associated samples on screen or printer.

*CMarkovSample* - represents an individual observation sequence *O* of length *T*. Contains *T* observation symbols (integer based) and allows storage of underlying states, one for each observation. The states are set either by the *Viterbi* algorithm or when the sequence is generated by a *CMarkovModel*. This class provides a method to format itself for screen or printer display.

*CMarkovReport* - represents the output from one of the HMM algorithms. Contains textual information and a variable number of *CMarkovModel*'s and/or *CMarkovSequence*'s. Provides the ability to display itself on screen or printer.

Collections of the above classes are held by a *CDocument* derived object and form the basis for the working view shown in Figure 3-14. All the classes are derived from *CObject* to support serialisation.

As the sizes of the models and sequences (in terms of number of elements) are variable, all probability distributions and observation sequences are held as dynamically allocated arrays of pointer variables using a variation of the techniques described in [22].

# Chapter 4 Current status

## 4.1 Data glove hardware

### *4.1.1 Limitations*

#### Effective Range

As the original application of the positioning system was to provide an image of a medical endoscope during colonoscopy, the design of the system was based around dimensions appropriate to that use. As a result the effective field of operation is roughly 1 $m^3$.

This range is fine for the purposes of experimenting with a single data glove but would not really be sufficient to cater for *two* gloves or for additional body sensors if the system were to be used as part of a more comprehensive motion capture application.

#### Number of Sensors

The Bladen positioning system allows for a maximum of 16 sensors. This easily covers the intended endoscopy application but is barely sufficient for working with more than one data glove at a time. One of the reasons for choosing an 8-sensor data glove is that this allows the possibility of implementing a two-handed system without the need for hardware changes.

#### Speed

The Bladen system currently available for testing is actually one of the original prototype systems developed by Dr. Bladen in 1993. Although the integral PC unit has been upgraded from an MS-DOS based Intel 486 system to a Windows 98 Pentium class system, the remaining hardware is nearly 8 years old. Given the increase in processing power achieved since then, the system performs very much slower than would be possible with more modern equipment. The maximum data capture rate possible with this original system appears in practice to be no more than 10 fps (frames per second).

### *4.1.2 Planned improvements*

A number of improvements to the Bladen positioning system are currently being pursued by colleagues at the University of Newcastle. These include the following:

#### Smaller Sensors

By using different materials for the sensor cores, the size of the original sensors has been reduced with no loss of sensitivity. The ability to use smaller sensors will give more flexibility in the choice of sensor sites and possibly allow the use of more sensors in close proximity to instrument the joints of the hand more precisely. This avenue is already being explored in Newcastle where an 11-sensor glove is being tested.

#### Larger Generators

More powerful field generators will possibly allow the generator assemblies to be sited further apart than at present, thus extending the effective range. This avenue is also being explored at the University of Newcastle.

#### Updated Hardware

David Geng, a PhD student at Newcastle, has recently proposed an updated hardware configuration for some critical parts of the system. This involves the introduction of an Intel ARM processor and a 48 channel A/D converter to allow the use of more sensors.

At present, these changes are at an early but Mr. Geng has given his opinion that the new configuration may be capable of providing real-time data capture at 50+ fps using 48 sensors. This would bring significant benefits were such improvements to materialise.

## 4.2 Statistical modelling

### *4.2.1 Limitations*

As discussed in section 2.2.3, using a linear PDM to model the non-linear movement of the hand is not ideal. The inability to locate matching landmark points on each 2D contour compounds these problems and the resulting distribution of points in shape space is decidedly non-uniform. At the current stage of development, it is not possible to use these PDM's without further analysis.

### *4.2.2 Planned improvements*

A number of options are available for managing and overcoming the effects of non-linearity in the models. These include:

- Using a clustering technique to segment data in shape space.
- Employing a 2D projection of the 3D sensor data to assist in the identification of matching landmarks between training images.

Some recently published work suggests it may be possible to avoid the problem of mismatching landmarks. This is an area which needs further investigation.

Allied to the above is the introduction of an HMM framework to the models. We have a basic tool available to assist in this although the most appropriate use may not become clear until the method for adapting the non-linear models has been decided and implemented. This is the part of the project under investigation at the time of writing.

## 4.3 Summary

### *4.3.1 Progress vs. objectives*

Progress to date has been reasonably good. The availability of accurate 3D point data gives an added impetus to the training of accurate statistical models. At the time of writing, no published work has been found indicating that this is an area currently being explored elsewhere.

While it is by no means clear that the current path will lead to achievement of the planned objectives, there are still many promising leads to follow.

Before more progress can be made in this direction, we first need to develop some method of quantifying the accuracy of the existing models. Once we can do this, it will be easier to establish where the boundaries should be between different models and this should lead into the introduction of an HMM framework to overlay everything. At that point a further review will be needed.

Until this stage, it will not be possible to put any of this work to the test with any serious recognition tasks but it may be that some comparable results can be obtained that will give us an indication of the strengths or weaknesses of these developments. This should be borne in mind as the work progresses to ensure that we make the most effective use of the 3D point data from the positioning system.

At present there is no evidence to suggest that the overall objectives cannot be achieved, at least to some worthwhile degree. Until such evidence presents itself, this project will continue in a spirit of optimism.

# Chapter 5 References


1. Allen CR, Su Y, Bell GD, Rowland RS. (2000) *Three-dimensional hand modelling using electromagnetic imaging for communication by sign language*. RecPad2000, 11th Portuguese Conference on Pattern Recognition, May 2000.

2. Bladen JS, Anderson AP, Bell GD , Rameh B and Evans B. (1993) *Non-radiological technique for three dimensional imaging of intestinal endoscopes*. Lancet 341:719-22.

3. Bladen JS. (1995) *Imaging Medical Endoscopes in three dimensions using magnetic fields*. PhD thesis University of Sheffield.

4. Bowden R. (1999) *Learning Non-linear Models of Shape and Motion*. PhD thesis, Brunel University, 1999.

5. Bowden R, Mitchell TA, Sarhadi M. (2000) *Non-linear statistical models for the 3D reconstruction of human pose and motion from monocular image sequences*. Image and Vision Computing, 18: 729 – 737.

6. Brien D, Ed.. (1992) *Dictionary of British Sign Language*. Faber and Faber, London. ISBN 0-571-14346-6.

7. Byung-Woo Min, Ho-Sub Yoon, Jung Soh, Tekeshi Ohashi and Toshiaki Ejima. (1999) *Visual Recognition of Static/Dynamic Gesture: Gesture-Driven Editing System*. Journal of Visual Languages and Computing, vol 10, pp 291-309.

8. Cootes TF, Taylor CJ, Cooper DH, Graham J. (1995) *Active Shape Models - their training and applications*. Computer Vision and Image Understanding, 61:1, pp 38-59.

9. Di Bernado E, Goncalves L, Perona P. (1998). *Monocular Tracking of the Human Arm in 3D*. In Cipolla R, Pentland A, ed. Computer Vision for Human-Machine Interaction, pp155-169. ISBN 0-521-62253-0.

10. Dogramadzi S, Allen C, Bell GD, Rowland RS. (1999) *An electromagnetic imaging system for remote sign language communication.* IMTC/99 Proceedings of the 16th IEEE Instrumentation and Measurement Technology Conference. Venice, Italy. Vol 3 May 1999. IEEE Catalog: 99CH36309 – ISBN: 0-7803-5276-9. 1443-1446.

11. Gavrila DM. (1999) *The Visual Analysis of Human Movement: A Survey*. Computer Vision and Image Understanding. vol 73;1 pp 82-98. Academic Press.

12. Hair JF (Jnr), Anderson RE, Tatham RL, Black WC. (1998) *Multivariate Data Analysis*. 5th Edition. Prentice Hall Intl. ISBN: 0-13-930587-4.

13. Heap T and Hogg DC. (1995) *Extending the PDM Using Polar Co-ordinates*. University of Leeds, Research report 95.5.

14. Isaacson DL and Madsen RW. (1976) *Markov Chains Theory and Applications*. John Wiley & Sons, Inc. ISBN: 0-471-42862-0.

15. Isard, M. and Blake, A. (1996) *Contour tracking by stochastic propagation of conditional density*. In Proc. European Conf. Computer Vision, vol 1 pp. 343–356, Cambridge UK.

16. Kakadiaris I and Metaxas D. (2000) *Model-Based Estimation of 3D Human Motion*. IEEE Transactions on Pattern Analysis and Machine Intelligence. Vol 22;12, December 2000, pp 1453- 1459.

17. Kapandji IA. (1982) *The Physiology of the Joints, Volume I Upper Limb*. Churchill Livingstone Inc. ISBN: 0-443-02504-5.

18. Kennaway JR. (1999) *Control of a multi-legged robot based on hierarchical perceptual control theory*. Journal of Perceptual Control Theory, vol.1, n.1.

19. Min BW, Yoon HS, Soh J, Yang YM and Ejima T. (1997) *Hand gesture recognition using hidden Markov models*. Proceedings of SMC'97*,* pp. 4232-4235.

20. Ouhaddi H and Horain P. (1999). *Hand tracking by 3D Model Registration. Virtual Reality and Prototyping*. June 1999, Laval (France).

21. Powers WT. (1974) *Behaviour: The Control of Perception*. Wildwood House, London. ISBN: 0-7045-00922.

22. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. (1997) *Numerical Recipes in C: The Art of Scientific Computing.* 2nd Edition. Cambridge University Press, ISBN: 0-521-43108-5.

23. Rabiner LR. (1989) *A tutorial on hidden Markov models and selected applications in speech recognition*. Proceedings of the IEEE 77**,** 257-286.

24. Rabiner LR, Wilpon JG, Juang BH. (1986) *A segmental k-means training procedure for connected word recognition*. AT&T Technical Journal, vol 65, no.3, pp 21-31.

25. Rabiner R, Juang BH. (1993) *Fundamentals of speech recognition*. PTR Prentice Hall. ISBN: 0-1301-51572.

26. Rehg J and Kanade T. (1994). *Digit Eyes: Vision-based hand tracking for human-computer interaction*. In Aggarwal J, Huang T, ed. Proc. of Workshop on motion of Non-Rigid and Articulated Objects, pp16-22. Austin, Texas: IEEE Computer Society Press.

27. Rehg J and Kanade T. (1994). *Visual Tracking of High DOF Articulated Structures: an Application to Human Hand Tracking*. Third European Conference on Computer Vision, Stockholm, Sweden, May 1994, pp35-46.

28. Rehg J and Kanade T. (1995). *Visual tracking of self-occluding articulated objects*. Proc. of 5th International Conference on Computer Vision, Cambridge MA, June 20-23 1995, pp612-617.

29. Shimada N and Shirai Y (19??). *3-D Hand Pose Estimation and Shape Model Refinement from a Monocular Image Sequence*. ????

30. Sonka M, Hlavac V, Boyle R. (1999) *Image Processing, Analysis and Machine Vision*. Second Edition. PWS Publishing. ISBN 0-534-95393-X.

31. Sozou PD, Cootes TF, Taylor CJ and Di-Mauro EC. (1994) *A non-linear generalisation of PDMs using polynomial regression*. In Proc. BMVC, volume II, pages 397-406, York, UK. BMVA Press.

32. Sozou PD, Cootes TF, Taylor CJ, Di-Mauro EC. (1995). *Non-linear Point Distribution Modelling Using a Multi-layer Perceptron*. In Pycock D, ed. British Machine Vision Conference 1995 (BMVA, Birmingham UK), pp107-116.

33. Starner TE and Pentland A. (1995) *Visual recognition of American sign language using hidden Markov models*. Proceedings of International Workshop on Face and Gesture Recognition, pp. 189 - 194.

34. Su Y, Geng D, Allen CR, Burn D, Bell GD, Rowland RS. (2001) *Three-Dimensional Motion System ("Data-Gloves"): Application for Parkinson's Disease and Essential Tremor*. IEEE International Workshop on Virtual and Intelligent Measurement Systems. Budapest, Hungary, May 19-20, 2001.

35. Utsumi A, Miyasato T, Kishino F, Nakasu R. (1995) *Real-time hand gesture recognition system*. Proceedings of the 2nd Conference on Computer Vision (II), pp. 249-253.

36. Vogler C and Metaxas D. (1998) *ASL Recognition Based on a Coupling Between HMMs and 3D Motion Analysis*. Proc. of the International Conference on Computer Vision. pp 363-369. Mumbai, India, January 4-7, 1998.

37. Wilson AD, Bobick AF. (1999) *Parametric Hidden Markov Models for Gesture Recognition*. IEEE Transactions on pattern analysis and machine intelligence. 21, 9: 884-891.

38. Ying Wu, Thomas S. Huang. (1999) *Vision-Based Gesture Recognition: A Review*. Lecture Notes in Computer Science: The 3rd Gesture Workshop, Gif-sur-Yvette, France.

39. Ying Wu and Thomas S Huang. (1999) *Capturing Articulated Human Hand Motion: A Divide-and-Conquer Approach*. Proc. of the International Conference on Computer Vision, Greece, 1999.

40. Ying Wu and Thomas S Huang. (2000) *View-independent Recognition of Hand Postures*. Proc. of the IEEE Conference on CVPR'2000. vol II pp 88-94.

41. Young SJ, Chase LL. (1998) *Speech recognition evaluation: a review of the U.S. CSR and LVCSR programmes*. Computer Speech & Language, October 1998, vol. 12, no. 4, pp. 263-279 (17)