CHAPTER-3
METHODOLOGY

3.1 Introduction to the Problem

Automatic human activity Recognition is a challenging problem in computer vision. One of its main goals is the understanding of the complex human visual system and the knowledge of how humans represent human activity in order to discriminate different identities with high accuracy. Two basic and conceptually independent problems have to be addressed by this kind of systems namely human activity detection and recognition of the detected human activity.

Work on the recognition stage, takes the detected human activity values as input to the algorithm. This stage can be separated in two steps: feature extraction, where important information for discrimination is saved, and the matching step, where the recognition result is given with the aid of a human activity database.

This work presents a biometric system performing identification of automatic human activity recognition. This system is based on Gabor features extraction using Gabor filter bank construction. For feature extraction the input image is convolved with Gabor filter bank to select a set of informative and non-redundant Gabor features. The extracted features are again subjected to discrete Radon transform (DRT) to extract a sequence of feature vectors from an image. The HMM (Hidden Markov Model) is used for matching the input face image to the stored images.

The HMM-based system developed matches the feature set (observation sequence) for a test image with an HMM of the claimed image, through Viterbi alignment. A distance measure is obtained by calculating negative log likelihood. The purpose of this research is to develop a novel, accurate and efficient human activity verification system.

3.2 Problem Definition and Proposed Solution

Automatic human activity Recognition is a challenging problem in computer vision. One of its main goals is the understanding of the complex human visual system and the knowledge of how humans represent human activity in order to discriminate different identities with high accuracy. Two basic and conceptually independent problems have to be addressed by this kind of systems.

- Human activity detection and
- Recognition of the detected human activity

Focus of this work is on the recognition stage i.e. taking the detected human activity as the input to the algorithm. This stage can be separated in two steps:

1. Feature extraction, where important information for discrimination is saved, and
2. Matching step, where the recognition result is given with the aid of a human activity database.

3.3 Proposed Solution

Activity recognition is an exciting ground for the development of robust machine learning techniques, as applications in this field typically require to deal with high-dimensional, multimodal streams of data that are characterized by a large variability (e.g. due to changes in the user's behavior or as a result of noise). However, unlike other applications, there is lack of established benchmarking problems for activity recognition. Typically, each research group tests and reports the performance of their algorithms on their own datasets using experimental setups specially conceived for that specific purpose. For this reason, it is difficult to compare the performance of different methods or to assess how a particular technique will perform if the experimental conditions change (e.g. in case of sensor failure or changes in sensor location). We intend to address this issue by setting up a challenge on activity recognition aimed at provide a common platform that allows the comparison of different machine learning algorithms on the very same conditions. We call for methods for tackling key questions in activity recognition

such as classification based on multimodal recordings, activity spotting and robustness to noise.

For this system, we used Gabor filter for selecting Gabor features for human activity recognition. A small subset of Gabor features capable of discriminating from other human activity images that are stored in the database. In this work the system developed uses the hidden Markov model (HMM) to match a test human activity image with an appropriate reference image. This system use the Gabor filter to extract the sequence of informative Gabor features from the given human activity image. The extracted features are again subjected to discrete Radon transform (DRT) to extract a sequence of feature vectors from a image. The HMM-based system developed in this paper matches the feature set (observation sequence) for a test image with an HMM of the claimed image, through Viterbi alignment (Bastos, L.C.et al.,1997). A distance measure is obtained by calculating a negative log likelihood (Bracewell, R.N 1995) & (Coetzer et al. 1994).

This work presents an overview of state-of-the-art methods in activity recognition using semantic features. Unlike low-level features, semantic features describe inherent characteristics of activities. Therefore, semantics make the recognition task more reliable especially when the same actions look visually different due to the variety of action executions. We define a semantic space including the most popular semantic features of an action namely the human body (pose and pose let), attributes, related objects, and scene context. We present methods exploiting these semantic features to recognize activities from still images and video data as well as four groups of activities: atomic actions, people interactions, human–object interactions, and group activities. Furthermore, we provide potential applications of semantic approaches along with directions for future research.

3.4 System Design

1. Feature Extraction: - The systems developed in this paper use similar feature extraction techniques. The bulk of the image processing and feature extraction involves the calculation of the discrete Radon transform (DRT) of

each image. The DRT is obtained by calculating projections of each image at different angles. After some further image processing (normalization), each of these projections constitutes a feature vector in an observation sequence. These features are classified as global features, since they are not extracted at stroke or sub-stroke level.

2. Image Modeling: - The systems developed in this dissertation use two very different approaches to model a specific human activity image. In the case of the HMM-based system, each facial image is modeled by an HMM of which the states are organized in a ring.HMM based system is developed in this dissertation (Cardot et al., 1994).
3. Matching: - The distance between a test image and a model for the claimed image is obtained as follows. The HMM-based system developed in this paper matches the feature set (observation sequence) for a test image with an HMM of the claimed image, through Viterbi alignment. A distance measure is obtained by calculating negative log likelihood.
4. Verification: - When a claim is made that a test image belongs to a specific person, the extracted observation sequence is first matched with a model of the image, so that a distance measure is obtained. This distance measure is then normalized in order to compensate for the variation in the human activity image. The variation in the human activity image is estimated by matching all of the training images with the image model. In this way several distance measures are obtained. Statistics of these distance measures are then used to estimate the variation in the image training set. A global threshold, that is a threshold which is the same for all image, can therefore be used. Test images, for which the normalized distance measure is less than this threshold are accepted and others are rejected. A schematic representation of the systems developed in this dissertation is given in figure 3.1.

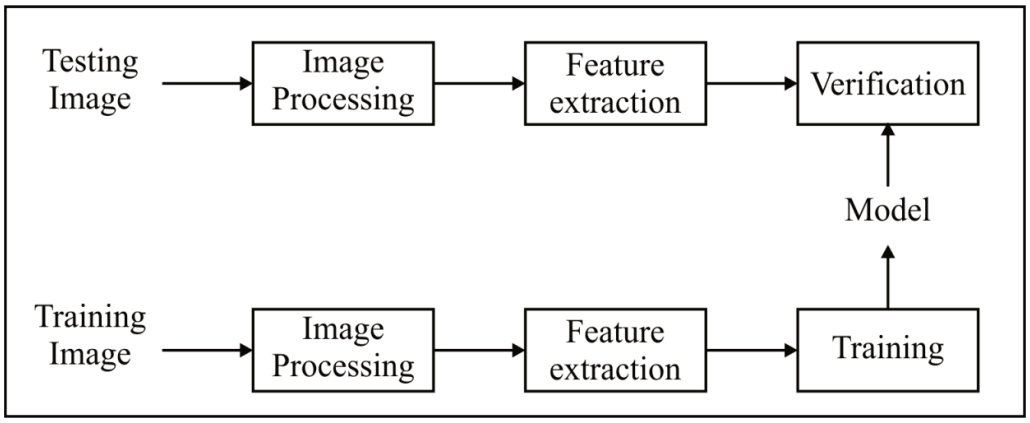

Figure 3.1 A schematic representation of the systems.

3.5 Filter Bank Constructions

In order to cover the frequency spectrum effectively, a range of both scales and orientations of the Gabor filters must be considered. The overall aim is to provide an even coverage of the frequency components of interest while maintaining a minimum of overlap between filters so as to achieve a measure of independence between the extracted co-efficient.

3.5.1 Gabor Filter

In the spacial domain, the 2D Gabor filter is a Gaussian kernel modulated by a sinusoidal plane wave

$$\phi\Pi(f,\theta,\gamma,\eta)(x,y) = (f^2/\pi\gamma\eta)e^{-(\alpha^2 x'^2+\beta^2 y'^2)} e^{j2\pi f x'} \quad (3.1)$$

$$x' = x\cos\theta + y\sin\theta,$$

$$y' = -x\sin\theta + y\cos\theta,$$

where f (cycles/pixel) is the central frequency of the sinusoidal plane wave, θ is the anticlockwise rotation of the Gaussian and the plane wave, α is the sharpness of the Gaussian along the major axis parallel to the wave, and β is the sharpness of the Gaussian minor axis perpendicular to the wave. $\gamma = f/\alpha$ and $\eta = f/\beta$ are defined such that the ratio between frequency and sharpness is constant.

3.5.2 Gabor Feature Representation

Once Gabor filters have been designed, image features at different locations, frequencies, and orientations can be extracted by convolving the image I(x, y) with

the filters: $O\Pi(f,\theta,\gamma,\eta)(x, y) = I(x, y) * \phi\Pi(f,\theta,\gamma,\eta)(x, y)$.A number of Gabor filters at different scales and orientations are usually used. We designed a filter bank with 8*8 and 8 orientations for feature extraction.

3.6 Image Processing

Each image is scanned into a binary image at a resolution of 300 dots per inch, after which median filtering is applied to remove speckle noise. The image dimensions are not normalized. Subsequently, the DRT of each image is calculated. Each column of the DRT represents a projection or shadow of the image at a certain angle. After these projections are processed and normalized, they represent a set of feature vectors (observation sequence) for the image in question. The DRT of an image is calculated as follows.

Assume that each image consists of Ψ pixels in total, and that the intensity of the i^{th} pixel is denoted by I_i, $i = 1, \ldots, \Psi$. The DRT is calculated using β non overlapping beams per angle and Θ angles in total. The cumulative intensity of the pixels that lie within the jth beam is denoted by R_j , $j = 1, \ldots, \beta\Theta$. This is called the j^{th} beam sum. In its discrete form, the Radon transform can therefore be expressed as follows:

$$R_j = \sum_{i=1}^{\Psi} w_{ij} I_i, \; j = 1,2,...,\beta\Theta, \tag{3.2}$$

where w_{ij} indicates the contribution of the i^{th} pixel to the j^{th} beam sum (see Figure 3.2). The value of w_{ij} is found through two-dimensional interpolation. Each projection therefore contains the beam sums that are calculated at a given angle.

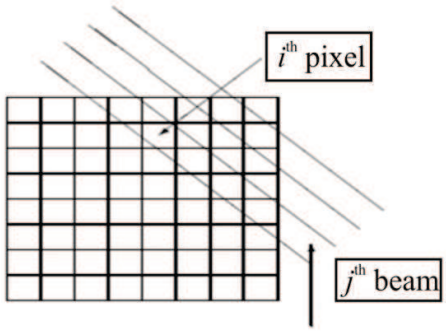

Figure 3.2: Discrete model for the Radon transform.

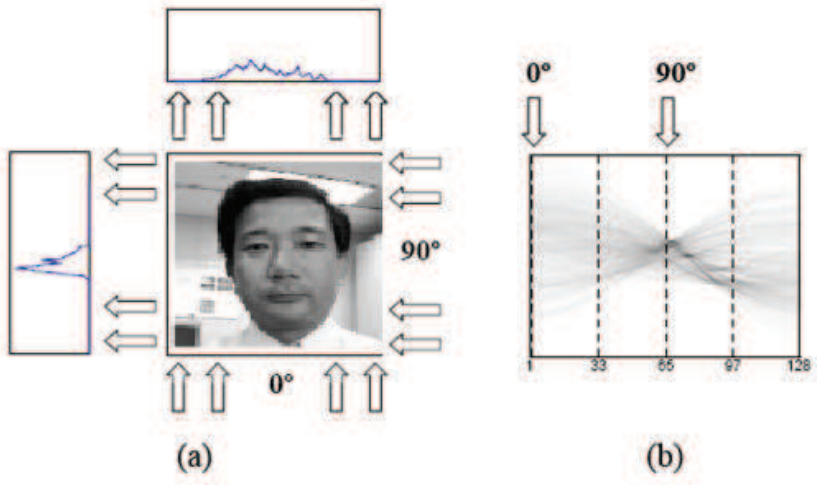

Figure 3.3: A image and its DRT. (a) A image and its projections calculated at angles of 0° and 90°. (b) The DRT displayed as a gray-scale image. This image has NΘ = 128 columns, where each column represents a projection.

The accuracy of the DRT is determined by Θ (the number of angles), β (the number of beams per angle), and the accuracy of the interpolation method. Note that the continuous form of the Radon transform can be inverted through analytical means. The DRT therefore contains almost the same information as the original image and can be efficiently calculated with an algorithm by Bracewell (Bracewell, 2005). Our system calculates the DRT at Θ angles. These angles are equally distributed between 0° and 180° (See figure 3.3).

The dimension of each projection is subsequently altered from β to d. This is done by first decimating all the zero-valued components from each projection. These decimated vectors are then shrunk or expanded to a length of d through interpolation. Although almost all the information in the original image is contained in the projections at angles that range from 0° to 180°, the projections at angles that range from 180° to 360° are also included in the observation sequence. These additional projections are added to the observation sequence in order to ensure that the sequence fits the topology of our HMM.

Since these projections are simply reflections of the projections already calculated, no additional calculations are necessary. An observation sequence

therefore consists of T = 2Θ feature vectors, that is, XT_1 = {x1, x2. . . xT}. Each vector is subsequently normalized by the variance of the intensity of the entire set of T feature vectors. Each image pattern is therefore represented by an observation sequence that consists of T observations, where each observation is a feature vector of dimension d.

The DRT, as a feature extraction technique, has several advantages. Although the DRT is not a shift invariant representation of a image, shift and scale invariance is ensured by the subsequent image processing. Each image is a static image and contains no dynamic information. Since the feature vectors are obtained by calculating projections at different angles, simulated time evolution is created from one feature vector to the next, where the angle is the dynamic variable. This enables us to construct an HMM for each image.

The DRT is calculated at angles that range from 0° to 360° and each observation sequence is then modeled by an HMM of which the states are organized in a ring. This ensures that each set of feature vectors is rotation invariant. Our system is also robust with respect to moderate levels of noise. These advantages are now discussed in more detail.

3.7 Advantages of DRT

3.7.1 Simulated Time-Evolution.

Each image is a static image and contains no dynamic information. Since the feature vectors are obtained by calculating projections at different angles, simulated time-evolution is created from one feature vector to the next, where the angle is the dynamic variable. This enables the HMM-based system to construct an HMM for each image.

3.7.2 Rotation Invariance.

Since the augmented DRT is periodic, with a period of 360°, we are able to achieve rotation invariance in an elegant and robust way. The HMM-based system, for example, represents each image with an HMM of which the states are organized in a ring. The HMM is constructed in such a way that it is equally likely to enter the

HMM at any state. This guarantees rotation invariance. For the DTW-based system, rotation invariance is only achieved after some further processing.

3.7.3 Shift Invariance.

Although the DRT is not a shift invariant representation of a image, shift invariance is ensured by the subsequent image processing. The zero-valued components of each projection are decimated and the corresponding feature vector is constructed from the remaining components only.

3.7.4 Noise

We explained earlier in this section that the zero-valued components of each projection are decimated before the remaining non-zero components are shrunk or expanded through interpolation. In this way, a feature vector with the required dimension is obtained. The decimation of the zero-valued components ensures that moderate levels of noise (which are represented by a few additional small-valued components within certain projections) are "attached" to the other nonzero components before the decimated vector is shrunk or expanded.

3.7.5 Scale Invariance

For each projection, scale invariance has to be achieved in the direction perpendicular to the direction in which the image is scanned, that is, perpendicular to the beams, and in the direction parallel to the beams. Scale invariance perpendicular to the beams is ensured by shrinking or expanding each decimated projection to the required dimension. Scale invariance parallel to the beams is achieved by normalizing the intensity of each feature vector. This is achieved by dividing each feature vector by the variance of the intensity of the entire set of feature vectors.

3.8 Background Subtraction

Background subtraction, also known as Foreground Detection, is a technique in the fields of image processing and computer vision wherein an image's foreground is extracted for further processing (object recognition etc.). Generally an image's regions of interest are objects (humans, cars, text etc.) in its foreground.

Background subtraction is a widely used approach for detecting moving objects in videos from static cameras. The rationale in the approach is that of detecting the moving objects from the difference between the current frame and a reference frame, often called “background image”, or “background model (see figure 3.4).

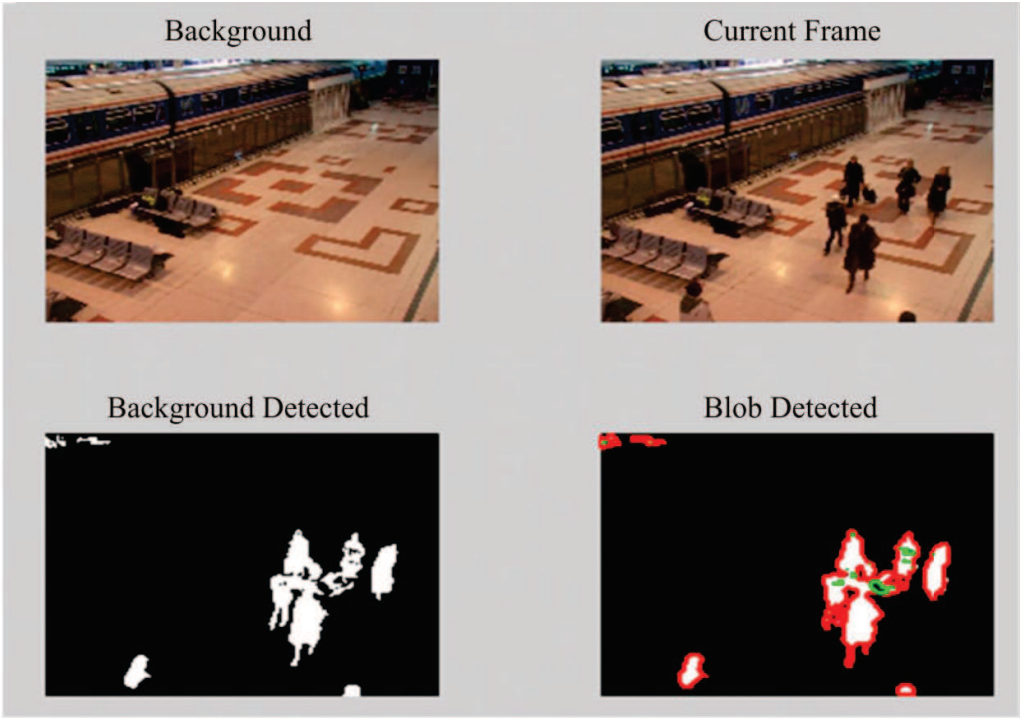

Figure 3.4: Background subtraction using MATLAB

3.9 Images Modeling Using HMM

HMM-based system developed uses a continuous first order HMM to represent each image. The HMM-based and DTW-based systems use similar verification protocols A pattern recognition system, which is based on HMMs, typically uses an HMM to represent each pattern class. Each of these HMMs is used to model an observation sequence, as well as the relationship between the individual observations. HMMs are therefore constructed in such a way that time-evolution is assumed from one observation in the sequence to the next. Since speech signals and dynamic (on-line) images also contain temporal information, it is possible to extract a continuous observation sequence from these signals in a very intuitive way. For this reason HMMs are especially well-suited for modeling these type of signals. This is not the case for static (off-line) images. Consequently, feature vectors have to be

extracted from off-line images in such a way that time-evolution is simulated from one observation to the next.

In this dissertation we use a grid to segment an image into local square cells. From each cell, the pixel density is computed, so that each pixel density represents a local feature. Each image is therefore represented by a sequence of feature vectors, where each feature vector represents the pixel densities associated with a column of cells. The HMM-based system developed in this dissertation simulates time-evolution from one observation to the next by calculating the DRT of each image during the feature extraction process. Before we discuss the HMM-based image model, we first present the notation in the following section.

Notation

We use the following notation for a sequence of T continuous observations,

$$X_{T1} = \{x_1, x_2, \ldots, x_T\}, \quad (3.3)$$

Where x_i, i = 1, 2, . . . T denotes the ith feature vector in the sequence.

We use the following notation for a continuous, first order HMM λ:

We denote the N individual states as

$$S = \{s_1, s_2, \ldots, sN\}, \quad (3.4)$$

and the state at time t as q_t.

The initial state distribution is denoted by

$$\pi = \{\pi_i\}, \text{ where } \pi_i = P(q_1 = s_i), i = 1, \ldots N. \quad (3.5)$$

The state transition probability distribution is denoted by A = {ai,j}, where

$$a_{i,j} = P(q_{t+1} = s_j \mid q_t = s_i), i = 1, \ldots N, j = 1, \ldots, N. \quad (3.6)$$

The PDF, which quantifies the similarity between a feature vector x and the state sj, is denoted by

$$f(x|s_j, \lambda), j = 1, \ldots, N. \quad (3.7)$$

The similarity between an observation sequences X and a model λ is denoted by

$$f(X|\lambda). \quad (3.8)$$

3.9.1 HMM Topology

The HMM-based system developed represents each image with an HMM of which the states are organized in a ring (Figure 3.5).This model is equivalent to the popular left-to-right model, but a transition from the last state to the first state is allowed. Since the HMM is constructed in such a way that it is equally likely to enter the model at any state, and the feature vectors are obtained from all the projections, that is, the projections calculated at angles ranging from 0° to 360°.The ring topology of the HMM guarantees that the images are rotation invariant.

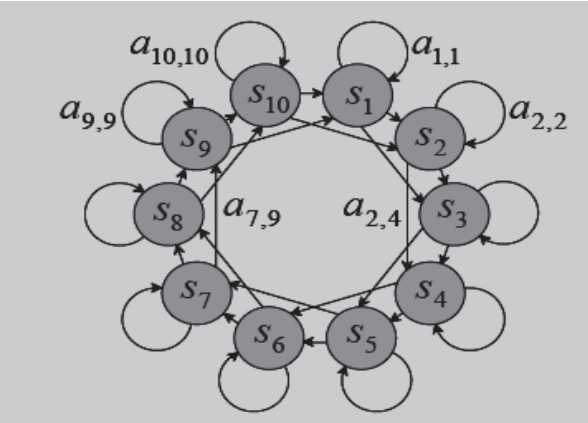

Figure 3.5 An example of a HMM with a ring topology.

This model has ten states with one state skipped. One state skip is equivalent to two allotted forward links. Each state in the HMM represents one or more feature vectors that occupy similar positions in a d-dimensional feature space. This implies that the HMM groups certain projections (columns of the DRT) together. It is important to note that this segmentation process only takes place after some further image processing has been conducted on the original projections. The state transition probabilities and the parameters for the PDFs, that represent the individual states, are estimated during training.

When a test sequence is matched with a trained ring-structured HMM, it is possible that the HMM is exited before the entire ring has been traversed. In other words, it is possible that the HMM is exited before the state immediately preceding the emitting state, which was initially visited, is encountered for the definitions of emitting and non-emitting states. It is also possible that the ring is traversed more than once.

In order to ensure that the entire ring is traversed, and only once, one may for example use N HMMs, where each HMM has N states and left-to-right topology. Figure 3.6 illustrates this configuration for N = 6. Note that the first HMM, that is λ_1, does not have a transition between S_6 (the last emitting state) and S_1 (the first emitting state).

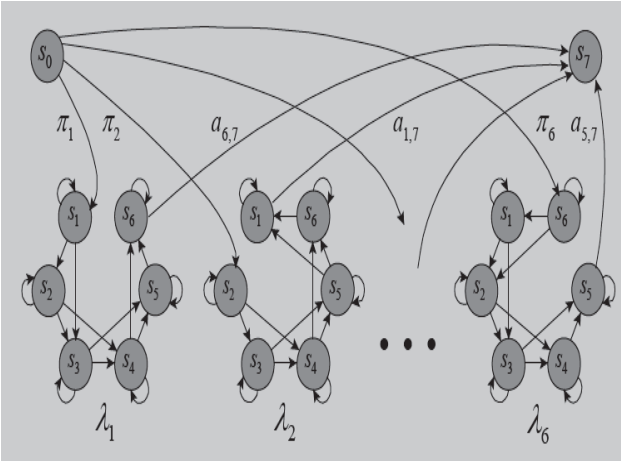

Figure 3.6: An example of a model that consists of six individual HMMs..

Each of these individual HMMs has six states with one state skip and a left-to-right topology. The initial (non-emitting) state is denoted by s_0 and the terminal (non-emitting) state by s_7. With this configuration it is still equally likely to enter the model at any state, but it is guaranteed that the entire ring will be traversed.

The second HMM, that is λ_2, does not have a transition between S_1 (the first emitting state) and S_2 (the second emitting state). Corresponding states within these HMMs, for example S_2 in λ_1 and S_2 in λ_2, share a common probability density function. The initial and terminal (non-emitting) states are denoted by S_0 and S_7, respectively. The probability to make a transition from the initial (non-emitting) state to S_1 in λ_1, or to S_2 in λ_2, or to S_3 in λ_3, etc., is the same, that is $\pi_1 = \pi_2 = \pi_3 = ... = \pi_6 = 1/6$.

When a transition is made from the initial (non-emitting) state to S_1 in λ_1, it is only possible to reach the terminal (non-emitting) state from S_6. Similarly, when a transition is made from the initial (non-emitting) state to S_2 in λ_2, it is only possible to reach the terminal (non-emitting) state from S_1, etc.

In this way it is still equally likely to enter the model at any state, but it is guaranteed that the entire ring will be traversed. However, when the above model is used with N HMMs and N states per HMM, we have N_2 states and $3N_2$ transitions, in contrast to the original N states and 3N transitions. This enlarges the computational requirements considerably. We therefore did not implement this model.

The HMM-based system developed in this dissertation utilizes a single ring-structured HMM, like the one shown in figure 3.5.

3.9.2 Initial Estimates

The HMM-based system developed in this dissertation uses uniform estimates for the initial state and state transition probabilities. The PDFs, which represent the individual states, are estimated by first assigning an equal number of observations to each state. The average of the observations within each state is then calculated.

3.9.3 Training

Each model is trained using the Viterbi reestimation technique. The dissimilarity between an observation sequence X and a model λ can therefore be calculated as follows (Rabiner, L.R.,1989).

$$D(X, \lambda) = -\ln(f(X|\lambda)). \tag{3.9}$$

In real-world scenarios, each writer can only submit a small number of training samples when he or she is enrolled into the system. Since the algorithm uses feature vectors with a high dimension, the estimated covariance matrix of the PDF for each state is not reliable and may even be singular.

A Mahalanobis distance measure can therefore not be found. Consequently these covariance matrices are not estimated and are initially set to 0.5I, where I is the identity matrix. Only the mean vectors are estimated, which implies that the dissimilarity values are based on an Euclidean distance measure. We assume that training images are available for only a limited number of images, that is for those images in the database used. Assuming that there are Ω image in the database, the

training images for each writer are used to construct an HMM, resulting in Ω models, that is {λ1, λ2. . . λΩ}.

When the training set for writer ω is denoted by

$$\{X_1^{(\omega)}, X_2^{(\omega)}, \dots\dots\dots\dots, X_N^{(\omega)}\} \tag{3.10}$$

Where Nω is the number of samples in the training set, the dissimilarity between every training sample and the model is used to determine the following statistics for the image,

$$\mu_\omega = \frac{1}{N_\omega}\sum_{i=1}^{N_\omega} D\left(X_i^{(\omega)}, \lambda_\omega\right) \tag{3.11}$$

$$\sigma_\omega^2 = \frac{1}{N_\omega - 1}\sum_{i=1}^{N_\omega}\left(D\left(X_i^{(\omega)}, \lambda_\omega\right) - \mu_\omega\right)^2 \tag{3.12}$$

These statistics are used to obtain a threshold distance, which is subsequently used to authenticate an input (test) image. The mean provides a reference distance. The standard deviation σω measures the variability of the image.

Note that the mean μ_ω represents the average dissimilarity between the observation sequences in the training set and the HMM, λ_ω. Also note that λ_ω was trained with these observation sequences. This implies that the mean μ_ω, in conjunction with providing a reference distance, also measures the variability of the image. The trained model for the image ω therefore consists of the trained HMM, λ_ω, in conjunction with one or both of the statistics defined in above equations.

3.9.4 Verification

When a system aims to detect only subsets of other image, training sets can be used to model system. This is called “impostor validation” and can be achieved through strategies like test normalization. These techniques enable one to construct verifiers that detect random false image very accurately. Our verifier is constructed as follows. When a claim is made that the test pattern X(w) Test belongs to image w, the pattern is first matched with the model λw through Viterbi alignment. This match is quantified by f (X(w) Test|λw). The dissimilarity between the test pattern and the model is then calculated as follows:

$$d\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) = -\ln\left(f\left(X_{Test}^{(\omega)}|\lambda_{\omega}\right)\right) \quad (3.13)$$

In order to use a global threshold for all image, Dolfing (J. G. A. Dolfing, 1998) suggests that every dissimilarity value in (3.13)) is normalised, using the statistics of the claimed image, that is,

$$d_{Mah}\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) = \frac{d\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) - \mu_{\omega}}{\sigma_{\omega}} \quad (3.14)$$

Where $d_{Mah}\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right)$denotes the normalized dissimilarity between the test pattern and the model of the claimed image. This normalization is based on the assumption that the dissimilarity value in (J. G. A. Dolfing, 1998) is based on a Mahalanobis distance measure.

When only the mean vectors are estimated though, the dissimilarity value in (3.13) is based on an Euclidean distance measure. When this is the case, we found that significantly better results are obtained when the standard deviation of the dissimilarities of the training set, that is, σ_w in (3.14), is replaced by the mean μ_w, that is,

$$d_{Eud}\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) = \frac{d\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) - \mu_{\omega}}{\mu_{\omega}} \quad (3.15)$$

A sliding threshold τ, where $\tau \in (-\infty, \infty)$, is used to determine the error rates for the test patterns. When

$d_{Eud}\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) < \tau$ that is

$$d_{Eud}\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) < \mu_{\omega}(1 + \tau) \quad (3.16)$$

The claim is accepted, otherwise, the claim is rejected. When $\tau = 0$, all the test patterns for which $d\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) \geq \mu_w$ are rejected. When $\tau \rightarrow \infty$ all the test patterns, for which $d\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right)$ is finite are accepted.

1) Finding the best state sequence using the Viterbi algorithm.

To find the single best state sequence $QT* 1 = \{q*1, q*2, \ldots, q*T\}$ for the given observation sequence $OT_1 = \{o_1, o_2, \ldots, o_T\}$, we need to define the quantity

$$\delta t(i) = \max_{AllQ_1^{t-1}} p(q_t = s_i, O_1^t|\lambda),$$

that is the best score along a single path, which accounts for the first t observations, and ends in state si. By induction we have that

$$\delta t(i) = \max_{i=1,\dots,N}[\delta_t(i)a_{ij}]b_j(O_{t+1}),$$

To retrieve the state sequence, we need to keep track of the argument which maximized for each t and j. We do this via the array ψt(j). The complete procedure for finding the best state sequence (Viterbi algorithm) can now be stated as follows:

Initialization

$$\delta_1(i) = \pi_i b_i(o_1), i = 1, 2, \dots\dots\dots, N$$

$$\psi_1(i) = 0$$

Recursion

$$\delta_t(j) = \max_{j=1\dots N}[\delta_{t-1}(i)a_{ij}]b_j(o_t), t = 2, \dots T, j = 1, \dots N$$

$$\psi_t(j) = \operatorname{argmax}_{j=1\dots N}[\delta_{t-1}(i)a_{ij}], t = 2, \dots T, j = 1, \dots N$$

Termination

$$P^* = \max_{i=1,\dots N}[\delta_T(i)]$$

$$q_t^* = \operatorname*{argmax}_{i=1,\dots N}[\delta_T(i)]$$

Backtracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*), t = T-1, T-2, \dots, 1$$

Here P* = P (OT_1, QT* 1 |λ) is an approximation to the forward or the backward method.

Each state in the HMM used in this dissertation is represented by a PDF for which only the mean vector is estimated. The corresponding covariance matrix is kept fixed. The dissimilarity between an observation sequence and the HMM is therefore based on an Euclidean distance measure for example, a threshold of τ =

0.16 is selected, equation (3.16) implies that all the test patterns for which $D\left(X_{Test}^{(\omega)}, \lambda_{\omega}\right) \geq 1.16\mu\omega$ are rejected the other pattern are accepted.

For various values of observations, states and feature lengths, it is clear that when the dimension of the feature vectors is decreased from d = 512 to d = 256 or even to d = 128, the performance of the system does not decrease significantly. The performance of the HMM-based system is generally enhanced when the number of feature vectors, or the number of states in the HMM, that is N, is increased. The best results are obtained when only one forward link is allowed in the HMM.

3.10 Tools Used: MATLAB

The name MATLAB stands for MATrix LABoratory. MATLAB was written originally to provide easy access to matrix software developed by the LINPACK (linear system package) and EISPACK (Eigen system package) projects. MATLAB (The MathWorks Inc. MATLAB 7.0 (R14SP2), 2005) is a high-performance language for technical computing. It integrates computation, visualization, and programming environment. Furthermore, MATLAB is a modern programming language environment: it has sophisticated data structures, contains built-in editing and debugging tools, and supports object-oriented programming. These factors make MATLAB an excellent tool for teaching and research. MATLAB has many advantages compared to conventional computer languages (e.g., C, FORTRAN) for solving technical problems. MATLAB is an interactive system whose basic data element is an array that does not require dimensioning. The software package has been commercially available since 1984 and is now considered as a standard tool at most universities and industries worldwide. It has powerful built-in routines that enable a very wide variety of computations. It also has easy to use graphics commands that make the visualization of results immediately available. Specific applications are collected in packages referred to as toolbox. There are toolboxes for signal processing, symbolic computation, control theory, simulation, optimization, and several other fields of applied science and engineering.

Documentation about MATLAB and MATLAB commands is available from within the program itself. If you know the name of the command and need more

information about how it works, you can just type "help <command name>" at the MATLAB prompt. In the same way, you can get information about a group of commands with common uses by typing "help <topic name>". I will show examples of using the command-line help feature below.

The MATLAB desktop contains a help browser covering both reference and tutorial material. To access the browser, click on the Help menu and choose MATLAB Help. You can then choose "Getting Started" from the table of contents for a tutorial introduction to MATLAB, or use the index to find specific information.

As mentioned above, MATLAB has many capabilities, such as the fact that one can write programs made up of MATLAB commands. The simplest way to use MATLAB, though, is as an interactive computing environment (essentially, a very fancy graphing calculator). You enter a command and MATLAB executes it and returns the result.

Starting MATLAB After logging into your account, you can enter MATLAB by double-clicking on the MATLAB shortcut icon (MATLAB 7.0.4) on your Windows desktop. When you start MATLAB, a special window called the MATLAB desktop appears. The desktop is a window that contains other windows. The major tools within or accessible from the desktop are:

- The Command Window
- The Command History
- The Workspace
- The Current Directory
- The Help Browser
- The Start button

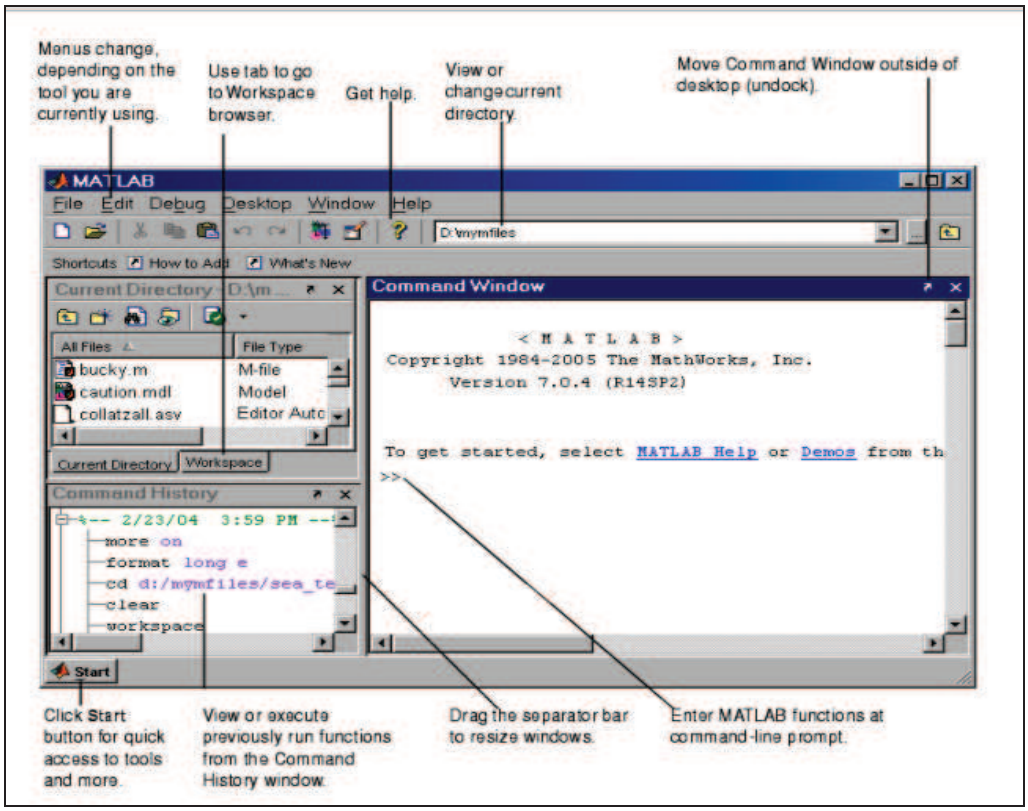

Figure 3.7: MATLAB Screen to show various MATLAB Windows

When MATLAB is started for the first time, the screen looks like the one that shown in the Figure 3.7. This illustration also shows the default configuration of the MATLAB desktop. You can customize the arrangement of tools and documents to suit your needs. Now, we are interested in doing some simple calculations. We will assume that you have sufficient understanding of your computer under which MATLAB is being run. You are now faced with the MATLAB desktop on your computer, which contains the prompt (>>) in the Command Window. Usually, there are 2 types of prompt: >> for full version EDU> for educational version Note: To simplify the notation, we will use this prompt, >>, as a standard prompt sign, though our MATLAB version is for educational purpose.

MATLAB is a high-level technical computing language for research and interactive environment for algorithm development, data visualization, data analysis, and numerical computation. It can be used in a wide range of applications, including design, analysis and optimization of engineering problems. MATLAB benefit's

faculty involved in research and development and contents will be supplemented with case studies.

Key Features

- High-level language for numerical computation, visualization, and application development
- Interactive environment for iterative exploration, design, and problem solving
- Mathematical functions for linear algebra, statistics, Fourier analysis, filtering, optimization, numerical integration, and solving ordinary differential equations
- Built-in graphics for visualizing data and tools for creating custom plots
- Development tools for improving code quality and maintainability and maximizing performance
- Tools for building applications with custom graphical interfaces
- Functions for integrating MATLAB based algorithms with external applications and languages such as C, Java, .NET, and Microsoft® Excel®

The MATLAB Language

- The MATLAB language provides native support for the vector and matrix operations that are fundamental to solving engineering and scientific problems, enabling fast development and execution.
- With the MATLAB language, you can write programs and develop algorithms faster than with traditional languages because you do not need to perform low-level administrative tasks such as declaring variables, specifying data types, and allocating memory. In many cases, the support for vector and matrix operations eliminates the need for for-loops. As a result, one line of MATLAB code can often replace several lines of C or C++ code.

Development Tools

MATLAB includes a variety of tools for efficient algorithm development, including:

- Command Window - Lets you interactively enter data, execute commands and programs, and display results
- MATLAB Editor - Provides editing and debugging features, such as setting breakpoints and stepping through individual lines of code
- Code Analyzer - Automatically checks code for problems and recommends modifications to maximize performance and maintainability
- MATLAB Profiler - Measures performance of MATLAB programs and identifies areas of code to modify for improvement
- Additional tools compare code and data files, and provide reports showing file dependencies, annotated reminders, and code coverage.

MATLAB add-on products provide functions in specialized areas such as statistics, optimization, signal analysis, and machine learning (see figure 3.8).

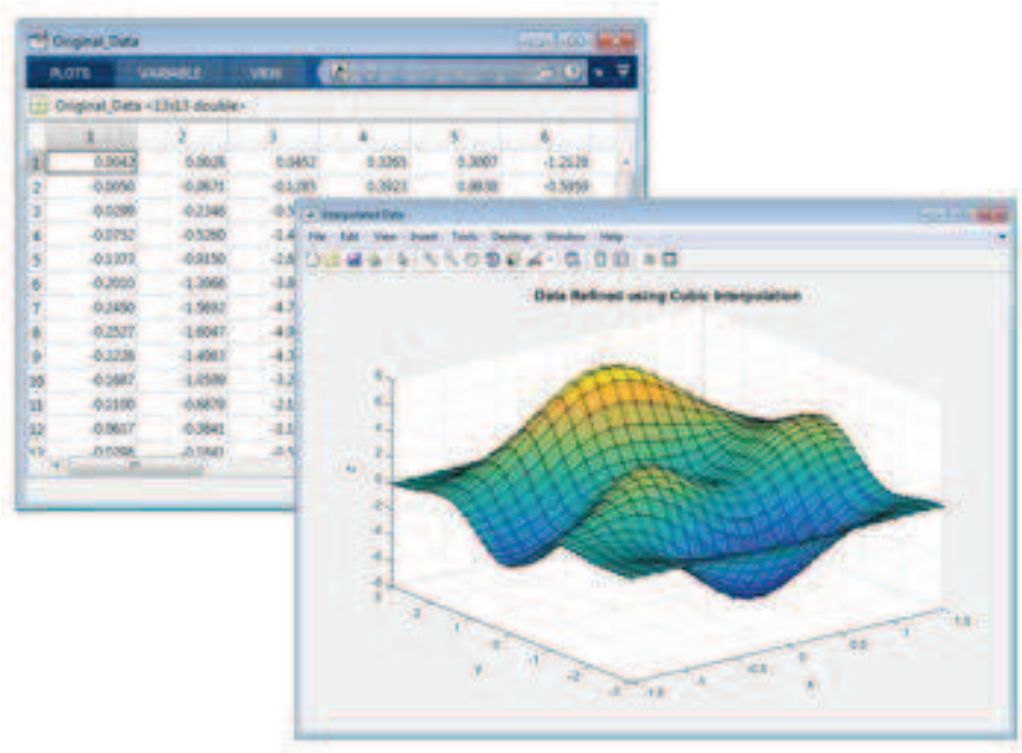

Figure 3.8: MATLAB Graphs

Data Analysis and Visualization

MATLAB provides tools to acquire, analyze, and visualize data, enabling you to gain insight into your data in a fraction of the time it would take using spreadsheets or traditional programming languages. You can also document and share your results through plots and reports or as published MATLAB code.

Analyzing and Visualizing Data with MATLAB Explore, visualize, and model your data with MATLAB®.

Acquiring Data

MATLAB lets you access data from files, other applications, databases, and external devices. You can read data from popular file formats such as Microsoft Excel; text or binary files; image, sound, and video files; and scientific files such as net CDF and HDF. File I/O functions let you work with data files in any format.

Using MATLAB with add-on products, you can acquire data from hardware devices, such as your computer's serial port or sound card, as well as stream live, measured data directly into MATLAB for analysis and visualization. You can also communicate with instruments such as oscilloscopes, function generators, and signal analyzers.

What is a UI?

A user interface (UI) is a graphical display in one or more windows containing controls, called *components* that enable a user to perform interactive tasks. The user does not have to create a script or type commands at the command line to accomplish the tasks.

Unlike coding programs to accomplish tasks, the user does not need to understand the details of how the tasks are performed. UI components can include menus, toolbars, push buttons, radio buttons, list boxes, and sliders just to name a few. UIs created using MATLAB® tools can also perform any type of computation, read and write data files, communicate with other UIs, and display data as tables or as plots.

The following figure 3.9 illustrates a simple UI that we can easily build yourself.

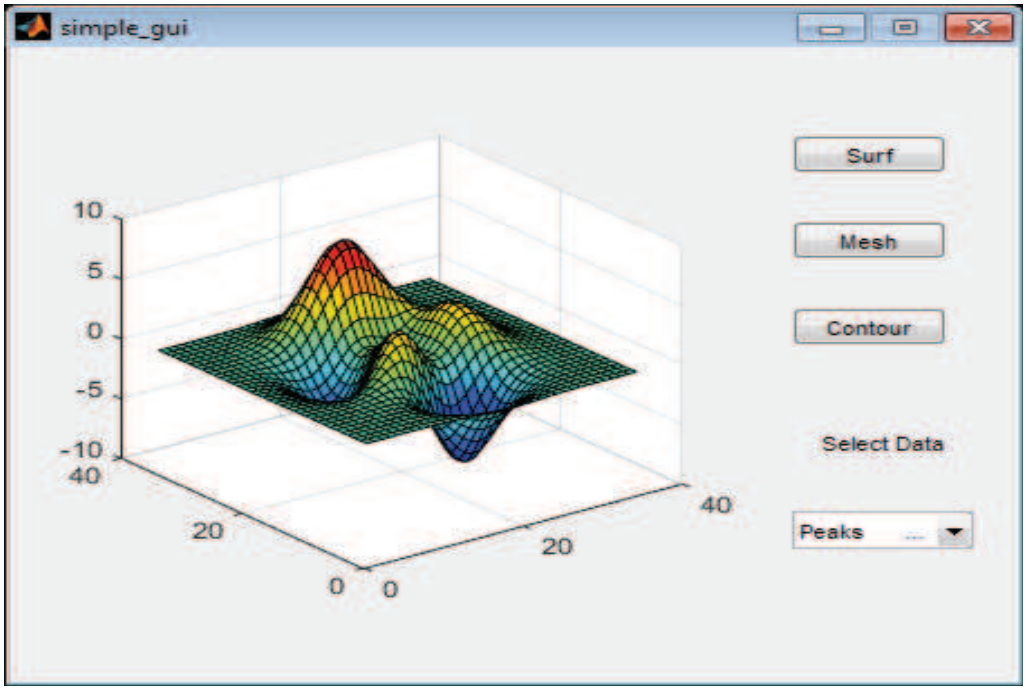

Figure 3.9 UI Window

The UI contains these components:

- An axes component.
- A pop-up menu listing three data sets that correspond to MATLAB functions: peaks, membrane, and sinc.
- A static text component to label the pop-up menu.
- Three buttons that provide different kinds of plots: surface, mesh, and contour.

When you click a push button, the axes component displays the selected data set using the specified type of 3-D plot.

How Does a UI Work?

Typically, UIs wait for a user to manipulate a control, and then respond to each user action in turn. Each control, and the UI itself, has one or more *callbacks*,

named for the fact that they "call back" to MATLAB to ask it to do things. A particular user action, such as pressing a screen button, or passing the cursor over a component, triggers the execution of each callback. The UI then responds to these *events*. We, as the UI creator, write callbacks that define what the components do to handle events. This kind of programming is often referred to as *event-driven* programming. In event driven programming, callback execution is *asynchronous*, that is, events external to the software trigger callback execution. In the case of MATLAB UIs, most events are user interactions with the UI, but the UI can respond to other kinds of events as well, for example, the creation of a file or connecting a device to the computer.

We can code callbacks in two distinct ways:

- As MATLAB language functions stored in files
- As strings containing MATLAB expressions or commands (such as 'c = sqrt(a*a + b*b);'or 'print')

Using functions stored in code files as callbacks is preferable to using strings, because functions have access to arguments and are more powerful and flexible. We cannot use MATLAB scripts (sequences of statements stored in code files that do not define functions) as callbacks. Although we can provide a callback with certain data and make it do anything we want, we cannot control when callbacks execute. That is, when wer UI is being used, we have no control over the sequence of events that trigger particular callbacks or what other callbacks might still be running at those times. This distinguishes event-driven programming from other types of control flow, for example, processing sequential data files.

Ways to Build MATLAB UIs:

A MATLAB UI is a figure window to which we add user-operated components. We can select, size, and position these components as we like. Using callbacks we can make the components do what we want when the user clicks or manipulates the components with keystrokes. We can build MATLAB UIs in two ways:

- **Create the UI using GUIDE**

This approach starts with a figure that we populate with components from within a graphic lawet editor. GUIDE creates an associated code file containing callbacks for the UI and its components. GUIDE saves both the figure (as a FIG-file) and the code file. We can launch wer application from either file.

- **Create the UI Programmatically**

Using this approach, we create a code file that defines all component properties and behaviors. When a user executes the file, it creates a figure, populates it with components, and handles user interactions. Typically, the figure is not saved between sessions because the code in the file creates a new one each time it runs. The code files of the two approaches look different. Programmatic UI files are generally longer, because they explicitly define every property of the figure and its controls, as well as the callbacks. GUIDE UIs define most of the properties within the figure itself.

They store the definitions in its FIG-file rather than in its code file. The code file contains callbacks and other functions that initialize the UI when it opens. We can create a UI with GUIDE and then modify it programmatically. However, we cannot create a UI programmatically and then modify it with GUIDE. The approach we choose depends on our experience, our preferences, and our goals.