Image and Vision Computing 20 (2002) 993–1007

www.elsevier.com/locate/imavis

Real-time gesture recognition system and application

Chan Wah Ng, Surendra Ranganath*

Department of Electrical and Computer Engineering, National University of Singapore, 4 Engineering Drive 3, Singapore 117576, Singapore

Abstract

In this paper, we consider a vision-based system that can interpret a user's gestures in real time to manipulate windows and objects within a graphical user interface. A hand segmentation procedure first extracts binary hand blob(s) from each frame of the acquired image sequence. Fourier descriptors are used to represent the shape of the hand blobs, and are input to radial-basis function (RBF) network(s) for pose classification. The pose likelihood vector from the RBF network output is used as input to the gesture recognizer, along with motion information. Gesture recognition performances using hidden Markov models (HMM) and recurrent neural networks (RNN) were investigated. Test results showed that the continuous HMM yielded the best performance with gesture recognition rates of 90.2%. Experiments with combining the continuous HMMs and RNNs revealed that a linear combination of the two classifiers improved the classification results to 91.9%. The gesture recognition system was deployed in a prototype user interface application, and users who tested it found the gestures intuitive and the application easy to use. Real time processing rates of up to 22 frames per second were obtained. © 2002 Elsevier Science B.V. All rights reserved.

Keywords: Real-time gesture recognition; Hand segmentation; Hidden Markov models; Neural networks

1. Introduction

As computers become more pervasive in society, facilitating natural human–computer interaction (HCI) will have a positive impact on their use. Hence, there has been growing interest in the development of new approaches and technologies for bridging the human–computer barrier. The ultimate aim is to bring HCI to a regime where interactions with computers will be as natural as interactions between humans, and to this end, incorporating gestures in HCI is an important research area [1].

We are interested in developing a vision-based system which can interpret a user's gestures in real time to manipulate windows and objects within a graphical user interface (GUI). Various works by Kadobayashi et al. [2], Pavlovic et al. [3], Freeman et al. [4] and Kjeldsen et al. [5] indicate that there is keen interest among current researchers to incorporate gestures into traditional HCI interfaces. Our work expands on their ideas and also looks into the possibility of using two-handed gestures while imposing fewer constraints on the users.

Much of the research on real-time gesture recognition has focused on the space-time trajectory of the hand without considering the shape or posture of the hand [6–8]. These works utilized only relative or oscillatory motion of the hand to recognize the gesture. However, in many situations, the meaning of gestures depends very much on the hand posture, in addition to the hand movement. Hence, our work incorporates hand posture as well as hand motion to recognize gestures. Also, unlike other works where users are required to wear artificial devices like data gloves [9] or green markers [10], it is our aim to allow the users to perform gestures in a natural and unencumbered manner.

In the work by Kjeldson and Kender, gestures were incorporated into a windowing user interface to manipulate windows [5]. The hand was segmented by using a neural network whose inputs were images coded in the hue-saturation-intensity (HSI) color model. Another neural network was trained to classify each pose. Gesture interpretation was performed by a state machine which implemented a gesture grammar. Their work demonstrated the feasibility of using gestures in a modern GUI. Our system differs from Ref. [5] in a few ways. We defined a slightly larger gesture set, and our system is specifically designed to allow the user to employ two-handed gestures, if he wishes. Moreover, our system does not need to be re-trained for every new user; it needs only to be trained once to achieve a relatively high level of user-invariance. Finally, the processing steps are quite different.

In the following, we present an overview of the system in Section 2. In Section 3, we describe the segmentation

* Corresponding author. Tel.: +65-6874-6538; fax: +65-6779-1103.
E-mail address: elesr@nus.edu.sg (S. Ranganath).

procedure to locate the hand(s) in the image. We then discuss a wrist-cropping method to isolate the segmented hand from the rest of the arm in Section 4. Next in Section 5, we describe pose classification using RBF networks, with Fourier descriptors of the segmented hand boundary as input features. We develop and compare the performance of the gesture recognizer based on hidden Markov Models (HMM) and recurrent neural networks (RNN) in Section 6. We also consider enhancing the recognition performance by combining the classifiers. In Section 7, we present and discuss the recognition results. A prototype GUI application that was developed to run in real time is described in Section 8, and Section 9 concludes the paper.

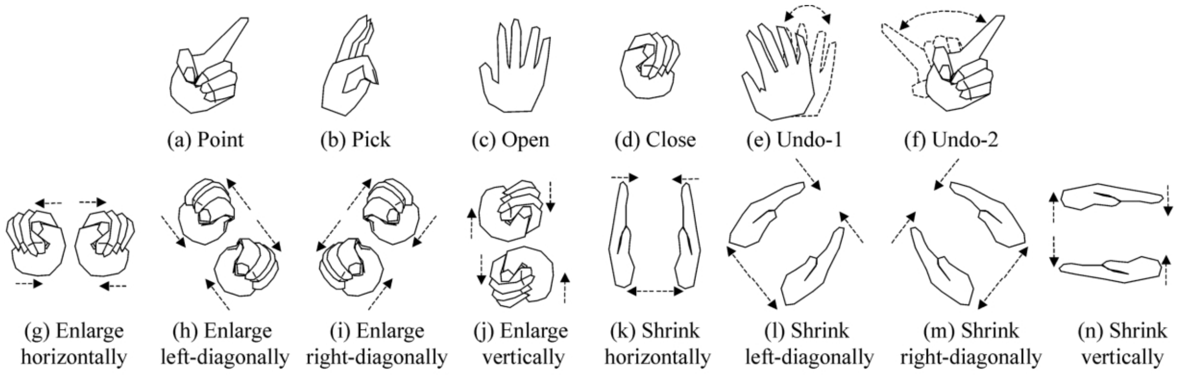
2. System overview

2.1. System description

Fourteen gestures shown in Fig. 1 were defined for controlling the windowing system. The *Point* gesture is used to move the cursor on the screen. The user can select a window/object to manipulate using the *Pick* gesture. Windows can be minimized with the *Close* gesture and restored with the *Open* gesture. The size of the window/object can be varied in different directions using different *Enlarge* and *Shrink* gestures. The *Undo* gestures can be used to reverse the previous action. These gestures are decomposed into the five basic, static hand poses shown in Fig. 2. The poses are first recognized in each frame and are used along with motion information to recognize the gestures. Poses are used to perform gesture recognition in this system for two reasons. First, most of the gestures are composed of distinct poses. Second, using pose as the intermediate recognition step, as will be described later in this paper, reduces system complexity and helps to increase recognition speed. The system was implemented on a Sun workstation. Images were captured at a resolution of 320×240 pixels in 24-bit color.

2.2. Processing stages

Every frame of the acquired image sequence undergoes five different processing stages as shown in Fig. 3 to recognize the gesture. The recognized gesture is input to a state machine which controls the user interface application. *Segmentation* utilizes color information and motion cue to transform the captured color images into binary hand images. The segmentation algorithm we have developed can handle one or both user hands in the original image and outputs each hand as a singly connected blob. The segmented hand blob(s) may include the hand and the lower arm, depending on whether the user is wearing a short-sleeves or long-sleeves shirt. To remove this variability, it is necessary to use a *wrist-cropping* procedure to isolate the hand from the lower arm. Rotation, scale and translation invariant features based on Fourier descriptors are then extracted from each wrist-cropped hand blob, and are input to a radial-basis function (RBF) network to obtain the likelihood of the five basic hand poses. If an image contains two hand blobs, the pose likelihood outputs of each hand blob are combined via a third RBF network. *Gesture recognition* is implemented using two independent classifiers, HMM chains and RNNs. Inputs to these classifiers are the hand pose likelihood vectors from each frame and the motion vector of the hands between the current and previous frames. Outputs from the two classifiers are combined by using a linear output layer to give the final gesture recognition result. The gesture label and the cursor position



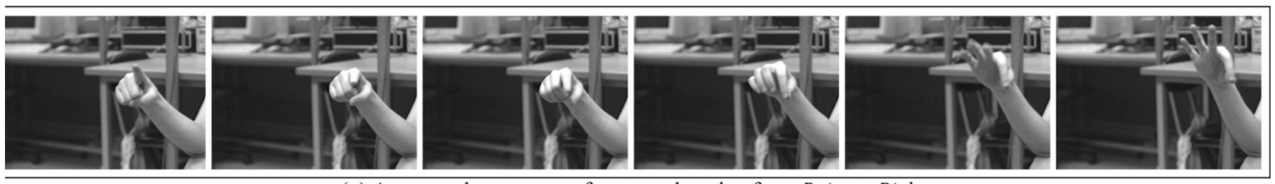

(o) An example sequence of gesture changing from *Point* to *Pick*.

Fig. 1. The fourteen gestures defined in the system are illustrated in (a)–(n). (o) shows an example sequence of gesture changing from *Point* to *Pick*.

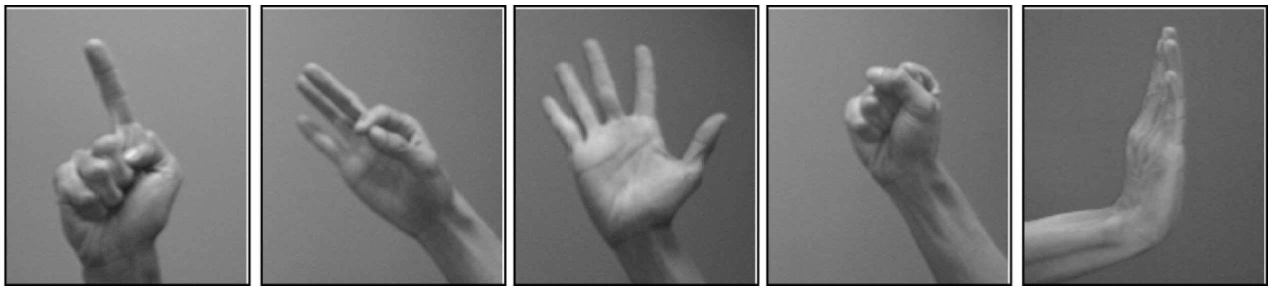


Fig. 2. The five basic hand poses making up the system gestures: *Point*, *Pick*, *Open*, *Close*, *Push*.

on the screen are input to a state machine to control the GUI application.

3. Segmentation

Fig. 4 depicts the segmentation process, which uses color and motion cues. The camera's field of view may contain objects moving in the background, but it is assumed that hands are the only skin-colored objects in the view, to simplify their extraction. Background differencing is used to isolate the moving object region, followed by a segmentation process to extract the skin-colored objects (hand and arm). A wrist-cropping operation is next used to separate the hand from the segmented arm. Details of the various steps in the segmentation process are described below.

3.1. Determination of object bounding box

A background image $B(x,y)$ is obtained from the initial frames (we assume that at least the first frame does not contain the hand). This background image is then used to obtain the

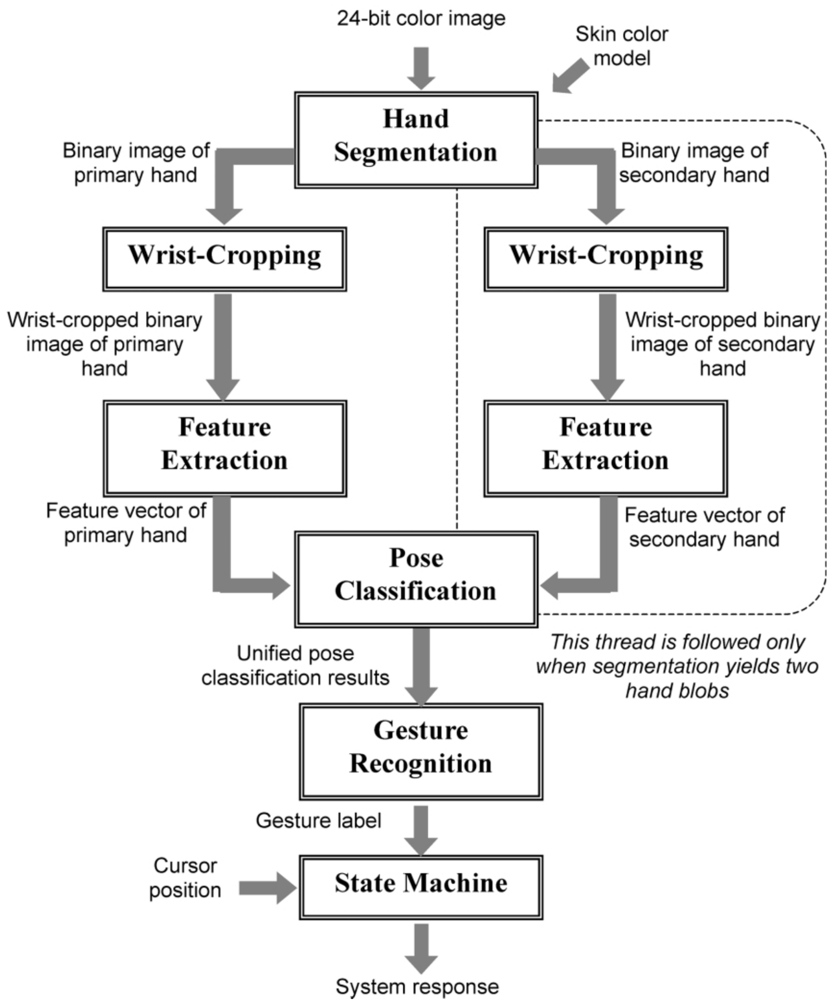

24-bit color image
Skin color model
Hand Segmentation
Binary image of primary hand
Binary image of secondary hand
Wrist-Cropping
Wrist-Cropping
Wrist-cropped binary image of primary hand
Wrist-cropped binary image of secondary hand
Feature Extraction
Feature Extraction
Feature vector of primary hand
Feature vector of secondary hand
Pose Classification
This thread is followed only when segmentation yields two hand blobs
Unified pose classification results
Gesture Recognition
Gesture label
Cursor position
State Machine
System response

Fig. 3. Overview of processing steps.

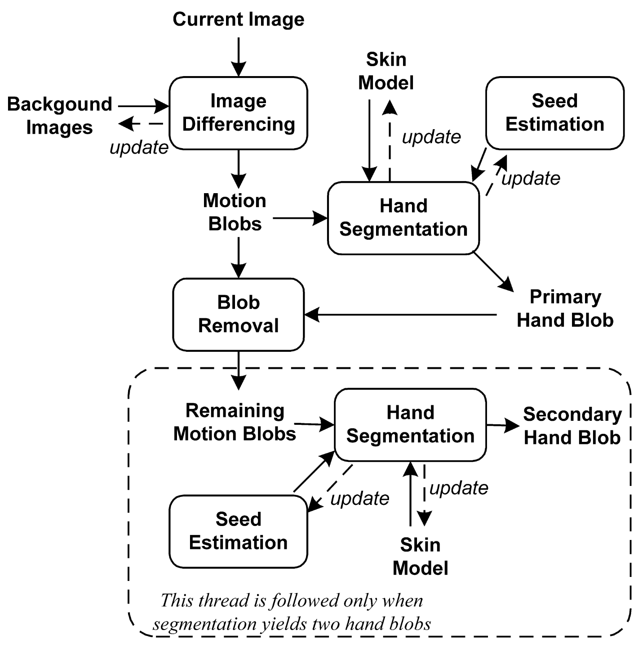


Fig. 4. Segmentation process for the hand.

object pixels in subsequent frames by image differencing. However, one would expect the background pixels to have small variations over an extended period of time due to changes in illumination, making it desirable to adapt the background image. This can be done by identifying images in the sequence which contain only background, and using these to continuously update $B(x,y)$. Background images can be identified by thresholding the absolute difference between two consecutive frames, and checking the number of pixels that have changed. If this is below a pre-determined threshold, the current frame is considered to be static, and further, if no hand blobs can be extracted from the static frame (using a segmentation algorithm to be presented later), then the frame is a background image. The background image is updated by the average of the previous and current background images. A foreground object $O(x,y,t)$ is then obtained by thresholding the absolute difference between the background image and current frame. The object pixels are then enclosed within the smallest bounding rectangle. Skin color segmentation is then carried out within this bounding box.

3.2. Skin color model

We represent skin color by a Gaussian distribution with mean color vector **m** and covariance matrix **S** given by

$$\mathbf{m} = [m_r, m_g]^{\mathrm{T}}, \qquad \mathbf{S} = \begin{bmatrix} \sigma_{rr}^2 & \sigma_{rg} \\ \sigma_{rg} & \sigma_{gg}^2 \end{bmatrix} \tag{1}$$

where m_r and m_g are the means of normalized red and green values of skin color, respectively, and the elements of **S** are the variances and correlations of the normalized colors in the usual notation. The parameters of the skin color model were initialized by calculating the skin color statistics of 10 different subjects under varying illumination conditions from 160 images. As such, the initial model is general, with large variances that can accommodate to skin color of different users. As the segmentation algorithm proceeds, the skin color statistics of the current user are used to adapt this general model [11]. Here, we averaged the means and covariance matrices of the skin color in the previous three segmented frames to estimate skin statistics for the current frame.

3.3. Seed estimation and segmentation

To start the segmentation procedure, a seed pixel within the hand is needed. The seed may be chosen as the centroid of $O(x,y,t)$. However, the centroid may not always be within the hand. To solve this problem, a set of five potential seed pixels is obtained. One of the seed pixels is chosen as the centroid of $O(x,y,t)$, while the other four seeds are selected by predicting the location of the hand in the current frame. In the first few initial frames where the hand is detected, as there is lack of prior information to allow accurate prediction, the four seed pixels are chosen to be uniformly distributed within the bounding rectangle of $O(x,y,t)$. In subsequent frames, the remaining four seed pixels are obtained as the (i) center of the bounding rectangle, (ii) centroid of the previously segmented hand, (iii) Kalman filter prediction of the centroid of the hand using constant velocity model; and (iv) Kalman filter prediction of the centroid of the hand using the constant acceleration model.

We obtain a globally thresholded region of interest (ROI) $GT(x,y)$ as

$$GT(x,y) = \begin{cases} 1, & \text{if } |r(x,y) - \hat{m}_r| < T_r \ \& \ |g(x,y) - \hat{m}_g| < T_g \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

where $\hat{m}_r$ and $\hat{m}_g$ are the estimated means of normalized red and green values for skin color in the current frame, $T_r = 2\hat{\sigma}_r$, $T_g = 2\hat{\sigma}_g$, and $\hat{\sigma}_r$ and $\hat{\sigma}_g$ are the estimated standard deviations of normalized red and green values. The scale factor of 2 in T_r and T_g was empirically found to be suitable over 160 training images. The globally thresholded ROI determines whether a pixel located at (x,y) with normalized color values $r(x,y)$, $g(x,y)$ falls within the Gaussian skin color model. To improve the segmentation, we form another locally thresholded ROI that determines whether the pixel at (x,y) belongs to the same object as its 4-neighbor pixel (x',y'). This is given as

$$LT(x,y) = \begin{cases} 1, & \text{if } |r(x,y) - r(x',y')| < d_r \ \& \ |g(x,y) - g(x',y')| < d_g \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

where $d_r = \hat{\sigma}_r$ and $d_g = \hat{\sigma}_g$, are local thresholds.

The local segmentation algorithm is a region growing procedure which needs a valid seed from among the five

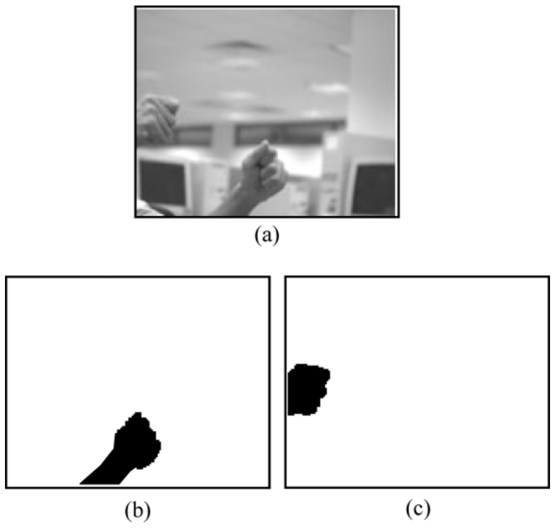


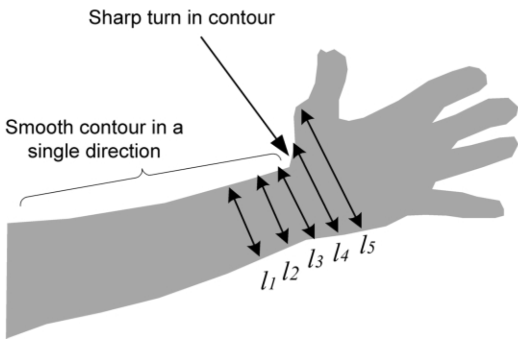
Fig. 5. Segmented images of a frame with two hands.

seeds described previously to initiate the process. A seed is considered to be valid, if its normalized color values r_s and g_s satisfy $GT = 1$ in Eq. (2). If all five seeds are invalid (below the global threshold), then segmentation is declared to have failed for this frame. When any seed pixel qualifies, the other seed pixels are discarded and segmentation begins with this seed. The recursive region growing procedure evaluates the four neighbors of the center pixel. Every neighbor pixel where $GT = 1$ and $LT = 1$ is considered to be a hand pixel, and is used as the center pixel in the next recursion.

The procedure described above has the following advantages: (1) it utilizes frame-to-frame adaptation of skin color statistics so that temporal variations are taken into account, (2) it utilizes local thresholding techniques so that spatial variations in skin and background color statistics are taken into account, and (3) the segmented image contains only one singly connected blob. It is simple to extract the second hand, if present. The same procedure is used again after removing all the pixels of the first hand. Fig. 5 shows typical segmentation results for an image containing two hands. The two hand blobs thus obtained are fed separately (and independently) through the wrist-cropping and feature extraction modules.

4. Wrist-cropping

The binary image obtained by segmentation is further processed to isolate the hand from the rest of the lower arm, by a wrist-cropping procedure. This is necessary because the segmented image may or may not include the lower arm depending on whether the user is wearing long-sleeves shirt, watches or other wrist ornaments. This can result in significantly different features being extracted for the same hand pose, and lead to increased


Sharp turn in contour
Smooth contour in a single direction
l_1 l_2 l_3 l_4 l_5
Fig. 6. A sudden increase in width and a sharp turn in contour at the wrist are used to mark the wrist location.

complexity in the pose classifier. To avoid this, we performed wrist-cropping so that only the hand is used for pose classification. Two different methods were implemented: one based on the width of the wrist and the other based on contour segmentation.

The width-based wrist-locating algorithm [12] is based on the observation that there is a sudden increase in the hand width as one moves from the lower arm to the hand. This is illustrated in Fig. 6. Before the wrist, the width of the lower arm is quite constant (i.e. $l_1 \approx l_2 \approx l_3$). At the wrist, however, the width increases suddenly ($l_5 > l_4 > l_3$). Detection of this serves as an indication of wrist location. Alternatively, the wrist can be located by a contour-based method. We observed that the contour of the hand is smooth from the arm to the wrist. At the joint where the thumb and wrist meet, however, there is a sharp turn in contour. This is also shown in Fig. 6. To decide between the two wrist estimates, i.e. the length-based estimate and the contour-based estimate, a parameter, $\hat{S}_p$, is defined to predict the palm size in the current frame. The estimated wrist location that yields a palm size closest to $\hat{S}_p$ is selected. $\hat{S}_p$ is determined by averaging the sizes of segmented hands in previous frames. Fig. 7 shows a wrist estimate, and how the palm is extracted by removing the lower part of the hand.

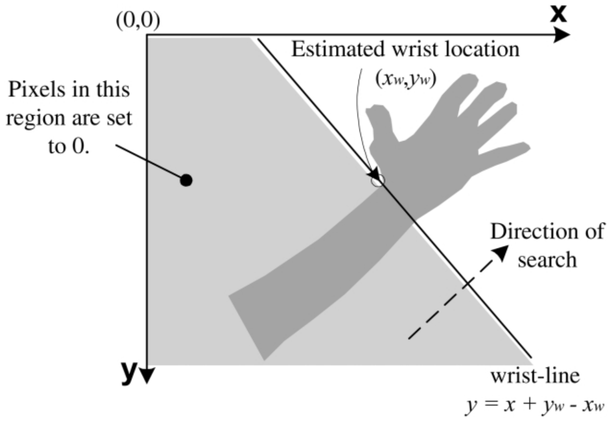


Fig. 7. Determining the hand region to be cropped.

5. Feature extraction and pose classification

5.1. Fourier descriptors

We used Fourier descriptors [13] to represent the boundary of the extracted binary hand as the set of complex numbers, $b_k = x_k + jy_k$, where $\{x_k, y_k\}$ are the boundary pixels. This is re-sampled to a fixed length sequence, $\{f_k, k = 0, 1, \ldots N-1\}$, for use with the discrete Fourier transform (DFT). Denoting $\{F_n\}$ as the DFT coefficients, the set of (rotation, scale, and translation invariant) Fourier descriptors is given by

$$A_n = \frac{|F_n|}{|F_1|}, \qquad 2 \le n < N. \tag{4}$$

We used a set of 56 Fourier descriptors, resulting from using a 64-point DFT. F_0 contains the positional information of the hand, and hence A_0 is discarded to remove positional sensitivity. After normalization by F_1 to remove scale sensitivity, $A_1 = 1$, and hence this descriptor is discarded. Six high frequency descriptors were also discarded to reduce sensitivity to noise.

5.2. Pose classification

Pose classification yields the likelihood of the current image belonging to one of the five defined poses, based on the extracted feature vector. This was implemented using RBF networks. The RBF network, RBF-FD, was trained with the boundary Fourier descriptors as input vectors, and consisted of 56 input nodes, 38 hidden layer nodes and five output nodes. Training of the networks followed a hybrid learning process [14]: selection of RBF centers, followed by supervised learning of linear weights for the output layer.

The activation function of the jth hidden node given an input vector $\mathbf{x}$ is

$$\varphi_j(\mathbf{x}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{c}_j\|^2}{2\sigma_j^2}\right) \tag{5}$$

where $\mathbf{c}_j$ is the center and σ_j is the spread of $\varphi_j(\mathbf{x})$. The RBF centers were selected based on a supervised *k-means* clustering algorithm. Here, the number of clusters m is assumed to be known a priori, and m initial cluster centers are needed. To specify this, we divided the training samples into five pose groups, and within each group, the samples were visually inspected to determine the number of sub-classes needed. This was done so that out-of-plane rotations and other variations in hands of the same pose could be taken into account by the system. A total of 27 initial sub-classes were identified visually. Fig. 8 illustrates the diversity of variations within each hand pose.

The k-means algorithm [15] was then used separately on training data of each pose to ensure that training samples from different poses would not be grouped into the same sub-class. One sample from each sub-class was chosen as the initial cluster center. After clustering all five groups of training samples, all the cluster centers obtained were used to classify the entire training set using the nearest neighbor rule. If the number of misclassified samples from any pose, say A, was substantial, it suggested that an additional sub-class may be needed for pose A. The k-means algorithm was then repeated over the training samples of pose A with an additional cluster center chosen from one of the misclassified samples. The entire process was repeated until the number of misclassifications was reduced to a tolerable fraction (10%). In this way, 38 sub-classes were identified for RBF-FD. The cluster means and standard deviations were chosen as the RBF parameters $\{\mathbf{c}_j\}$ and $\{\sigma_j\}$,

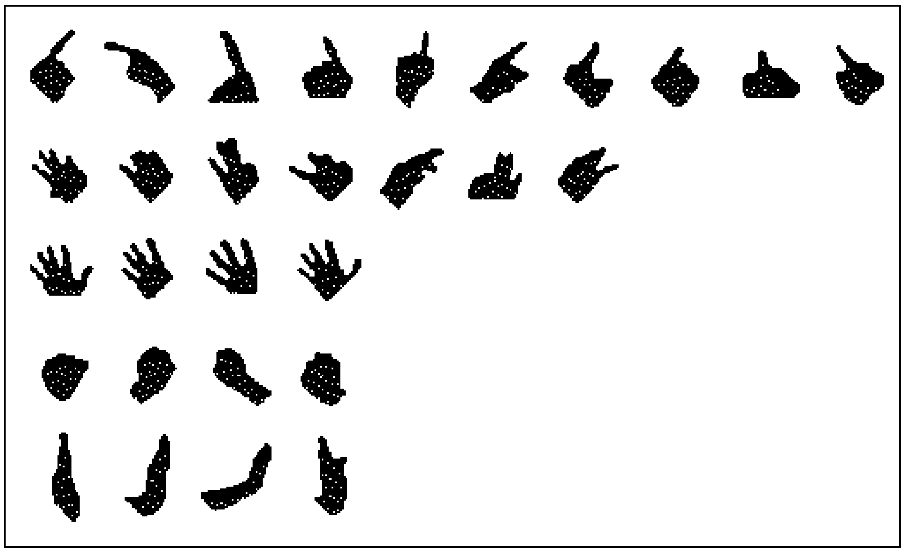


Fig. 8. Examples of sub-classes identified for each pose group for use in supervised k-means clustering.

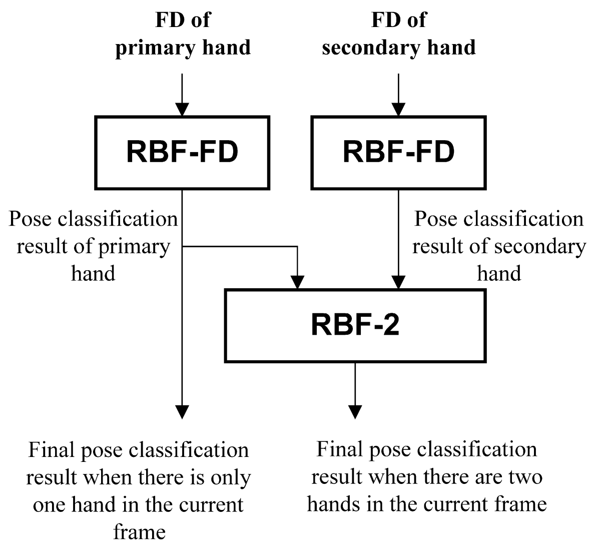

FD of primary hand
FD of secondary hand
RBF-FD
RBF-FD
Pose classification result of primary hand
Pose classification result of secondary hand
RBF-2
Final pose classification result when there is only one hand in the current frame
Final pose classification result when there are two hands in the current frame

Fig. 9. Overview of the RBF pose classifier.

respectively. Subsequently, the pseudoinverse method [16] was used to compute the weights of the output layer.

It is possible for a single image to contain two hands. When two hand blobs are obtained from the segmentation module, feature extraction and pose classification are done independently on each hand blob and then combined using a third neural network, RBF-2, to give a unified pose classification. RBF-2 consisted of 10 input nodes for the two hand classification vectors, 10 hidden layer nodes and five output nodes. Training of RBF-2 follows the same hybrid learning process as the RBF-FD network, i.e. k-means clustering to determine the RBF centers and pseudoinverse method to train the output layer. Fig. 9 shows the architecture of the pose classifier. In images with only one segmented hand blob, RBF-2 is by-passed and the pose output from the RBF-FD is used. When there are two segmented hand blobs, the combined pose output is obtained from RBF-2. Note that we do not distinguish between the left and right hand. It is not necessary since Fourier descriptors are mirror-image invariant [13]. Instead, we differentiate between the two hands by identifying a *primary* and *secondary* hand. *Primary* hand is the one that the user uses when making single-handed gestures (eg. *Point*, *Pick*). Typically, a user will use his/her *primary* hand extensively to move the cursor and select objects, with the *secondary* hand entering the field of view only when making gestures with two hands (eg. *Enlarge*, *Shrink*).

6. Gesture recognition

Gesture recognition uses the pose classification results and motion information of the centroids of the segmented hand(s) to classify the current frame as belonging to one of the fourteen predefined gestures. Input to the gesture recognition module is a nine element input vector $\mathbf{u} = [u_0 \dots u_8]^T$, which consists of five elements $[u_0 - u_4]$ from the pose classifier, and four additional elements $[u_5 - u_8]$, which encode the hand centroid motion and location. If the centroids of the primary hand and secondary hand in the nth frame are given by $(x_{p,n}, y_{p,n})$ and $(x_{s,n}, y_{s,n})$, respectively, then the elements $[u_5 - u_8]$ for the n-th frame are obtained as $u_5 = x_{p,n} - x_{p,n-1}$, $u_6 = y_{p,n} - y_{p,n-1}$, $u_7 = x_{p,n} - x_{s,n}$, and $u_8 = y_{s,n} - y_{s,n}$. If the nth frame contains only one hand, u_7 and u_8 are set to zero. If the $(n-1)$th frame does not have any hand blobs detected, then u_5 and u_6 are set to zero. Hence u_5 and u_6 encode hand motion, while u_7 and u_8 encode separation between the two hands.

We investigated both HMMs and RNNs to recognize the 14 gestures. We used a set of 14 HMM chains and 14 RNNs for each of the gestures. The gesture label was taken to correspond to the HMM or RNN that gave the maximum output. In the case of HMMs, we considered discrete as well as continuous models.

6.1. Gesture recognition using HMM

The discrete HMM (HMM-D) we tested was a set of fourteen HMM chains (one chain per gesture). Each chain consisted of five states in a pure feed-forward topology, as depicted in Fig. 10. Quantization of the input vector $\mathbf{u}$ is necessary for the HMM-D. Here, we employed a simple quantization scheme. The elements $[u_0 - u_4]$, were quantized by simply setting the maximum of the five pose likelihood to 1 and the remaining four to 0. Each of the remaining four elements of $\mathbf{u}$ related to motion were quantized to one of the symbols in the set {'+', '0', '−'}. Thus, the input vector $\mathbf{u}$ can be mapped to 405 distinct symbols. This implies that for each of the 14 HMM chains, training needs to identify a 5-element initial probability vector π, a 5×5 state transition probability matrix $\mathbf{A}$, and a 5×405 observation probability matrix $\mathbf{B}$. Training of the HMM-D follows the Baum–Welch re-estimation formulas [17].

The 14 continuous HMM (HMM-C) again had five states in a pure feed-forward topology. Here, the input vector $\mathbf{u}$ was directly used as the observation. Therefore, for each chain, apart from the 5-element initial probability vector π and 5×5 state transition probability matrix $\mathbf{A}$, the parameters that needed to be identified were a set of five observation mean vectors, $\{\boldsymbol{\mu}_j\}$, each of length 9, and a set of five 9×9 observation covariance matrices, $\{\sigma_j\}$.

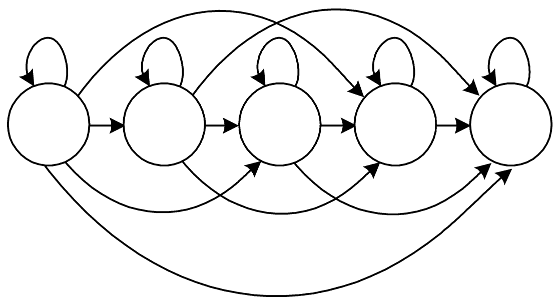


Fig. 10. Five state HMM with forward chaining.

Training of the HMM-C followed the Baum–Welch re-estimation formulas.

6.2. Gesture recognition using RNN

To compare gesture recognition performance with HMMs, we also investigated RNNs. Training the RNNs with the standard real-time recurrent learning (RTRL) algorithm [18] is computationally quite intensive [19–21]. An RNN with N feedback neurons and M total neurons in a fully connected architecture has NM weights, and requires $O(N^3M)$ operations to perform one cycle of the RTRL algorithm [21]. To reduce the computational complexity, several methods have been proposed. These include teacher-forcing [22], subgrouping strategy [21], and Mode-Exchange RTRL (MERTRL) [23,24]. We synthesized an effective training approach using these methods as described below.

6.3. Training the RNN classifiers

14 RNNs were each trained to recognize a different gesture. 14 RNNs were used instead of one RNN with 14 output neurons to make training of the RNNs more manageable. Attempts to train a single RNN with 14 output neurons failed to yield converging results. Each of the 14 RNNs consisted of nine external input nodes for the input vector $\mathbf{u}$, nine hidden processing neurons and one output neuron, yielding a 20×10 weight matrix. The standard unipolar logistic function was used as activation function for the processing neurons. To prevent bias, each RNN was trained with equal numbers of positive and negative examples, with the negative examples randomly selected from the training sets of other gestures. The desired output $d(n)$ was chosen either as an ascending or descending ramp function depending on the training example. For the positive examples, the RNNs were trained to give a ramp response starting from 0.5 in the initial frame to 1.0 in the last frame of the gesture sequence. For negative examples, the RNNs were trained to ramp down from 0.5 to 0.0. The ramp response is desirable instead of a flat response because certain gestures can be recognized with reasonable confidence only after a few frames have been processed. For example, the *Undo* gesture has a hand pose that is similar to the *Point* gesture. It is only when the oscillatory nature of the hand is recognized that we can be confident that it is the *Undo* gesture.

Training was done in batch mode, with each RNN assigned a random initial weight matrix. One training epoch consists of presenting the RNN with all its training sequences in random order. At the start of each training sequence, responses of the hidden processing neurons were re-initialized to 0, and the response of the output neuron was re-initialized to 0.5. Training of each RNN was carried out in two distinct modes, static and dynamic, as in the MERTRL. However, unlike the MERTRL where all recurrent weights are frozen except the self-recurrent synapse in the static mode, we trained the RNN with the RTRL algorithm using teacher-forcing and subgrouping strategy. The dynamic mode removes the subgrouping and trains the RNN with the full RTRL algorithm with teacher-forcing. Teacher-forcing [22] is a variant of the RTRL algorithm whereby the actual output $y_j(n)$ is replaced by the desired response (teacher signal) $d_j(n)$ in the subsequent computation of the behavior of the network, whenever such a value for $d_j(n)$ exists. Subgrouping [21] divides the RNN into smaller groups, and performs the RTRL training separately (and concurrently) on different subgroups.

The idea behind adopting this approach was to speed up the learning process without sacrificing performance. Subgrouping with teacher forcing in the static mode allows the RNN to quickly form localized sub-optimal solution to learning tasks. However, because the RNNs used have only one output neuron, it has to be replicated when subgrouping is used. In the static mode, two subgroups were used with each subgroup containing five neurons: four hidden neurons and one output neuron. The two output neurons had identical desired response. Each subgroup treated the feedback signals from the processing neurons in the other subgroup as external input signals. In the dynamic mode where there is no subgrouping, we removed the redundancy of replicating the output neuron and allowed the additional hidden neuron (which was previously a replicated output neuron) to be trained with fewer constraints.

For both the static and dynamic modes of training, the learning rate was selected to start from a small initial value (0.01) and was gradually reduced as training progressed, as recommended in Ref. [18]. The learning rate was reduced by a factor of half when the total squared error for three consecutive training epochs did not differ by more than a tolerable fraction, which was set to 0.01. Also, when three consecutive reductions of the learning rate did not cause the total squared error for an epoch to drop, the training mode

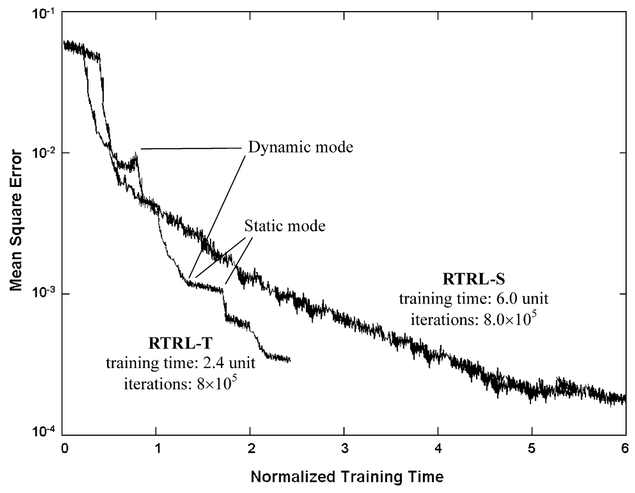


Fig. 11. Training curves for standard RTRL (RTRL-S) and the training method we adopted (RTRL-T).

was switched from dynamic to static or vice versa. The entire training process was terminated when switching between the two modes did not cause the total squared error to be reduced further.

Fig. 11 shows the learning curves for the standard RTRL algorithm (RTRL-S) and the two-mode approach (RTRL-T) we employed. We see that the latter training curve has distinct phases of gradual and sharp drop in MSE (corresponding to the static and dynamic phases, respectively). The training time required for RTRL-T was 0.4 of the time required for RTRL-S.

6.4. Combining HMM and RNN

Experiments showed that the performance of the HMM-D was worse than the HMM-C, and RNN classifiers, whose performance was quite good. Hence, we combined the latter two classifiers with a linear layer of weights with a view to improve the recognition results even further. Since the HMM-C responses are in log probabilities while the RNN outputs are in the range of 0–1, normalization is necessary for the HMM and RNN responses to have equal weight. Here, we used a softmax function to normalize the HMM and RNN responses to yield

$$y_{x,i}(n) = \frac{\exp[v_{x,i}(n)]}{\sum_{j=1}^{14} \exp[v_{x,j}(n)]}, \qquad x = \mathrm{HMM}, \mathrm{RNN} \tag{6}$$

where $v_{HMM,j}(n)$ is the nth log probability output from the HMM chain for the jth gesture, and $v_{\mathrm{RNN},j}(n)$ is the nth output from the RNN network for the jth gesture.

The input vector (from frame n) to this linear layer is the concatenated vector given by $\mathbf{y}(n) = [\mathbf{y}^{\mathrm{T}}_{\mathrm{HMM}}(n)\ \mathbf{y}^{\mathrm{T}}_{\mathrm{RNN}}(n) - 1]^{\mathrm{T}}$, where the last element is inserted as bias. The combined gesture recognition output is obtained from this layer as the vector $\mathbf{z}(n) = \mathbf{W}^{\mathrm{T}}\mathbf{y}(n)$, where $\mathbf{W}$ is a 29×14 weight matrix, and $\mathbf{z}(n) = [z_1(n), z_2(n), \ldots, z_{14}(n)]^{\mathrm{T}}$. The output of the combined gesture recognizer for frame n is then the label $j(n)$, where

$$j(n) = \arg \max_{i=1\ldots 14} \{z_i(n)\} \tag{7}$$

Since this is the last stage in the entire gesture recognition system, the criterion function that is minimized is not the total square error; rather it is the total square of misclassification errors given by

$$J(n) = \frac{1}{2} \sum_{k \in \Im} e_k^2(n), \tag{8}$$

where $\Im$ is the set of indices k such that the output $z_k(n)$ misclassifies of the input vector $\mathbf{y}(n)$, $e_k(n)$ is the kth element of the error vector $\mathbf{e}(n) = \mathbf{d}(n) - \mathbf{z}(n)$, $\mathbf{d}(n)$ is the desired output vector for the input vector $\mathbf{y}(n)$ such that its kth element $d_k(n)$ is 1 if $\mathbf{y}(n)$ belongs to gesture k, and is zero otherwise. To determine the set $\Im$, recall that the labeling of gesture involves determining the maximum output $z_j(n)$, $j = \arg\max\{z_i(n)\}$. Hence, in order for the input vector $\mathbf{y}(n)$ from the gesture class i to be misclassified, there must exist at least one j such that $z_j(n) > z_i(n)$. The set $\Im$ is then given by the set. The criterion function is optimized by an iterative gradient descent method. The training was terminated when the number of correct classification for an entire epoch of training gesture sequences did not increase over three consecutive epochs.

7. Results and discussion

In this section, we present results and discussion on the different components of the system, as well as overall gesture recognition performance.

7.1. Performance of the pose classifier

In addition to using Fourier descriptors as features to represent the hand boundary, as described in Section 5.1, we also compared their performance with Zernike moments [25] which have also been used in the literature as scale, rotation and translation invariant features. We found that Zernike moments of order up to 22 (yielding 142 features) were required for good performance. A neural network, RBF-Z was trained with the Zernike moments of the segmented hands from the same set of training images, and consisted of 142 input, 32 hidden and five output layer nodes. After training the two RBF networks with the two types of features, we evaluated their performance by comparing their classification results, and computational speed. Table 1 shows the performance of the two networks on 887 training vectors, and 495 unseen test vectors. It is apparent that the classification results of using the two types of features are comparable, but using Fourier descriptors is significantly faster. Hence the Fourier descriptors were subsequently used to provide the feature vectors for pose classification.

The performance of the entire pose classification module (shown in Fig. 9) using Fourier descriptors was evaluated in detail with 692 training images and 329 test images. Of the 692 training images, 195 contained two hands and 497 contained one hand. The 195 images with two hands were used to train RBF-2. Of the 329 test images, 166 contained two hands and 163 contained one hand. Table 2 shows the classification results. The pose classification module

Table 1
Comparison of classification results from Fourier descriptors and Zernike moments

Network	Training results (%)	Test results (%)	Computation time (ms)
RBF-Z	95.9	89.9	250
RBF-FD	95.7	90.7	5

Table 2
Pose classification results

	Training results		Test results	
	No. of images	Percentage correct	No. of images	Percentage correct
Single hand	497	96.0	163	90.8
Two hands	195	96.4	166	91.0
Combined	692	96.1	329	90.9

Table 3
Classification results for different hand poses on test images

Poses	No. of images	% Correct classification
POINT	90	87.8
PICK	58	82.8
OPEN	51	100.0
CLOSE	60	93.3
PUSH	70	92.9
TOTAL	329	90.9

achieved an overall correct classification rate of 96.1% for the training images and 90.9% for the test images.

Table 3 shows the individual classification accuracy for different poses. OPEN pose had the highest correct classification percentage (100.0%) and PICK pose had the lowest (82.8%). The relatively poor classification accuracy for the PICK pose was expected since it was most affected by out-of plane rotations. This is illustrated by example in Fig. 12. Due to the imaging angle, the PICK pose appears to be the POINT pose in the segmented binary image (Fig. 12(a)) while the PICK resembles the PUSH pose in the binary image (Fig. 12(b)). The POINT pose also suffers from effects of out-of-plane rotations. As shown in Fig. 12(c), because the index finger in the POINT pose was pointing almost directly at the camera, the corresponding binary image is very similar to the CLOSE pose. On the other hand, images of the OPEN, CLOSE and PUSH poses were relatively less affected by out-of-plane rotations. This is especially true for the CLOSE pose. However, the OPEN and PUSH poses, are still distinguishable from the others as long as the out-of-plane rotation is less than 90°. The approach taken to handle out-of-plane rotations was to divide each pose into sub-classes so that images from different angles of the same hand pose would be adequately represented by these sub-classes. This method alleviates the problem by using a large training set but cannot completely solve the problem of images of two different hand poses appearing similar due to the imaging angle. This would require the use of multiple cameras.

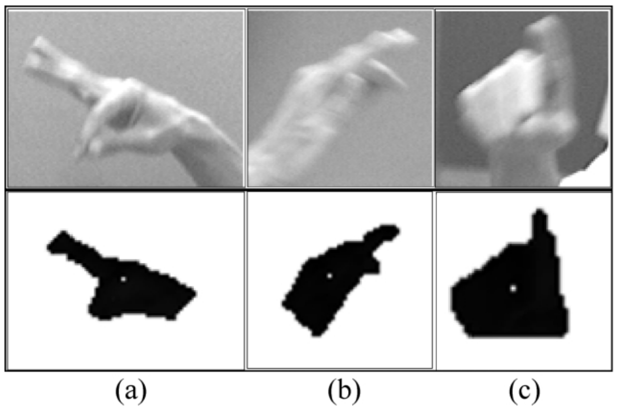


Fig. 12. Example images of hand poses which suffer from out-of-plane rotation problems.

7.2. Performance of the gesture recognizers

In this Section, we compare the isolated gesture recognition performance of the HMM-D, HMM-C and RNN classifiers, and the effect of combining the HMM-C and RNN classifier. However, in the final gesture recognition application considered in this paper, gestures are fed to the recognizers in a concatenated, continuous stream that requires temporal segmentation, as will be described in Section 8.

The Viterbi algorithm was used to obtain the accumulated log probabilities of each HMM chain. For the RNNs, the network response was used directly. Classification was done on a frame by frame basis where the frame was assigned to gesture class *A* if the HMM chain (RNN) corresponding to gesture class *A* had the highest accumulated log probability (output) among all the 14 HMM chains (RNNs) for that frame. Recognition of gesture from the sequence was deemed to be correct if more than 75% of the frames in the sequence were classified correctly. This criterion is used merely to facilitate commensurate performance analysis of the HMM and RNN classifiers. In actual deployment, HMM classifications would be obtained at the end of a sequence corresponding to the chain with the highest accumulated log probabilities, instead of on a frame-by-frame basis. However, RNN classifications are only obtained on a frame-by-frame basis. Thus to have an unbiased comparison, the two types of classifiers are compared using the same classification criterion.

Table 4 shows the recognition results of the three classifiers for 392 training and 295 test gesture sequences of varying lengths. From the recognition results, it is clear that HMM-C performed best for both training (93.6%) and test (90.2%) sets, whereas the HMM-D was the worst, with 92.5% for the training set and 86.4% for the test set. This is expected since the quantization scheme used for HMM-D is a very simple one. From experience with HMMs, it is

Table 4
Recognition results for the different recognizers

	HMM-D (%)	HMM-C (%)	RNN (%)	Combined (%)
Training	92.5	93.6	93.6	94.9
Test	86.4	90.2	89.8	91.9

Table 5
Breakdown of recognition results

HMM-D	HMM-C	RNN	Training	Test
Wrong	Wrong	Wrong	21	24
Correct	Wrong	Wrong	0	0
Wrong	Correct	Wrong	2	6
Correct	Correct	Wrong	2	0
Wrong	Wrong	Correct	4	5
Correct	Wrong	Correct	0	0
Wrong	Correct	Correct	2	5
Correct	Correct	Correct	361	255

expected that a better quantization scheme may improve results, but still unlikely to outperform HMM-C.

We next investigated whether there was any advantage to combining the classifiers by examining the breakdown of recognition results which are shown in Table 5. It is observed that in every case where the HMM-D classification is correct, so is the HMM-C classification; and when HMM-C classifies incorrectly, so does the HMM-D. This shows that the HMM-D cannot be used to supplement the HMM-C. This is expected since both type of recognizers are HMM-based, and trained in a similar manner. On the other hand, there are cases where RNN classified a sequence correctly when HMM-C did not, and vice versa. This is especially true for the test set, where there were five sequences that the RNN classified correctly but the HMM-C did not, and there were six sequences that the HMM-C recognized correctly, but the RNN did not. Hence, these classifiers were combined to supplement each other. Table 4 shows that by combining these two classifiers, the results for the training and test sets were improved to 94.9% and 91.9%, respectively.

7.3. Comparison between different recognition criteria

In the above discussion of the different gesture recognizers, the criterion we used to denote successful recognition of gesture i was that the output of the ith element of the recognizer is the maximum for 75% of frames in the sequence. The 75% criterion seems reasonable, since we cannot expect the recognizer to classify a sequence correctly for every frame in the sequence. Recognition of gesture is performed based on the temporal variations in hand pose. As such, the possibility of wrong classification for the initial frames in a sequence maybe very high. For instance, consider the case of the *Enlarge* gesture *vis-a-vis* the *Close* gesture. In both gestures, the hand pose corresponds to CLOSE. Certainly, the recognizer will not be able to distinguish between these two gestures in the first few frames. It can only do so when it recognizes the oscillatory motion of the hand in the *Enlarge* gesture, or the lack of it in the *Close* gesture. Thus recognition will only be accurate in the later frames of the gesture sequence.

For comparison, the recognition results for the test set using different values for the recognition criterion are shown in Table 6. We note here that this criterion is useful only for presentation and comparison of recognition results. When the gesture recognition is used in an application (Section 8), this criterion is irrelevant.

Table 6
Classification results using different criteria

	Percentage correct classification using different criteria				
	100%	90%	75%	60%	50%
Point	82	97	100	100	100
Pick	73	93	97	100	100
Close	71	95	100	100	100
Open	79	96	96	100	100
Undo-1	0	75	92	100	100
Undo-2	0	61	89	94	100
Shrink-V	34	72	83	89	100
Shrink-H	40	77	89	94	100
Shrink-LD	45	72	89	100	100
Shrink-RD	34	66	83	94	100
Enlarge-V	0	51	92	100	100
Enlarge-H	0	45	83	89	100
Enlarge-LD	0	34	89	94	100
Enlarge-RD	0	51	89	100	100
Total	39.5	75.5	91.9	97.2	100

7.4. Reduction in computation complexity

We have presented a modular approach to gesture recognition, where the gestures to be recognized were decomposed into a smaller set of basic hand poses. A pose classifier was then developed to recognize these poses in each frame, and the pose sequence was used to perform gesture recognition. This approach was devised not only to achieve more robust gesture recognition, but also to reduce computation time, both in recognition and in training.

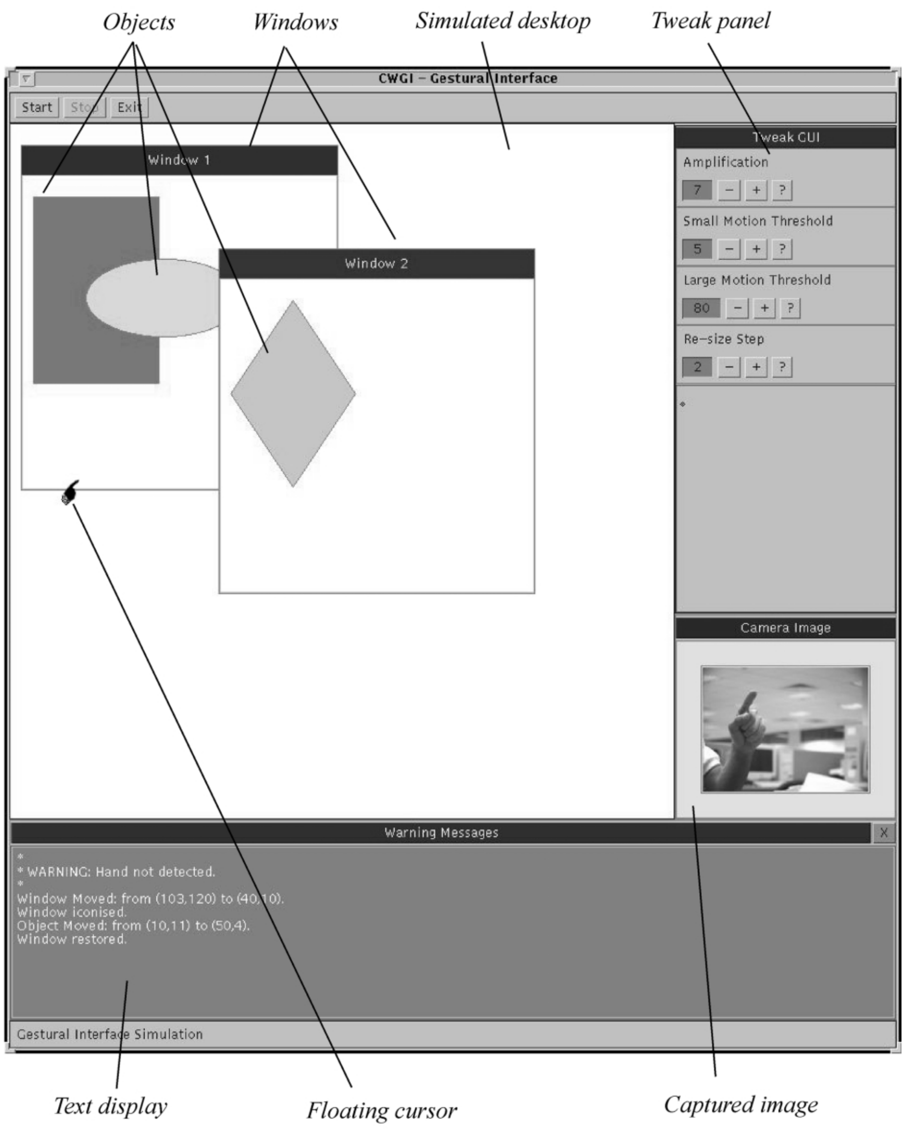
For simplicity, let the gesture recognizer consist of the pose classifier, and the set of 14 RNNs. Pose classification involves computing the five output responses of a RBF network with 56 input and 38 hidden nodes. This requires a computational time proportional to $56 \times 38 + 38 \times 5 = 2318$ units per frame. Gesture recognition involves 14 RNNs each having a 20×10 weight matrix. The computation required is thus $14 \times 20 \times 10 = 2800$ units. Hence, the total computation is 5118 units per frame. If, on the other hand, the pose classification stage was not implemented, we would expect to input 60 features (56 Fourier descriptors and four motion features) to 14 RNNs, each with say one output and nine hidden nodes (in practice, more hidden nodes will be needed). This implies that each RNN has a 71×10 weight matrix. The amount of computation is thus $14 \times 71 \times 10 = 9940$ units per frame. Indeed, the inclusion of an intermediate pose classification stage reduces the amount of computation required to perform recognition by about 50%, which is useful for real-time applications (an example application described in the next section runs at 22 frames per second (fps)). Needless to say, training time is significantly less when the task is decomposed into pose recognition followed by gesture recognition.

8. Application

The prototype application simulates a windowing GUI driven by gesture. The processing rate for frames acquired in real time is 22 fps. Fig. 13 shows a screen shot of the application. A major portion of the screen is the *simulated desktop*, where the user can manipulate windows and objects as in any other GUI desktop environment. To the right of the simulated desktop is a *tweak panel* for users to adjust the simulation parameters according to personal preference. Just below it is an image display showing the captured image from the camera. This allows the user to see and keep his/her hand within the camera's field of view. At the bottom is a text display screen reporting the system status.

8.1. State machine

The prototype application is controlled by a state machine. Fig. 14 shows the state transition diagrams, where the IDLE state refers to the system at rest. It must be noted that in Fig. 14, only one ENLARGE and one SHRINK state are shown for simplicity, instead of showing the ENLARGE and SHRINK states for all the directions. The conditions for state transitions are usually based on the output from the gesture recognition module. For example, the system will move from the IDLE state to POINT when the gesture recognition module outputs a *Point* label. In addition to gesture labels, another input that affects some state transitions is the item the floating cursor is currently on top of. For instance, a state change from POINT to PICK.x



Amplification 7
Small Motion Threshold 5
Large Motion Threshold 80
Re–size Step 2

*
* WARNING: Hand not detected.
*
Window Moved: from (103,120) to (40,10).
Window iconised.
Object Moved: from (10,11) to (50,4).
Window restored.

Gestural Interface Simulation

Fig. 13. Screen shot of the prototype GUI application.

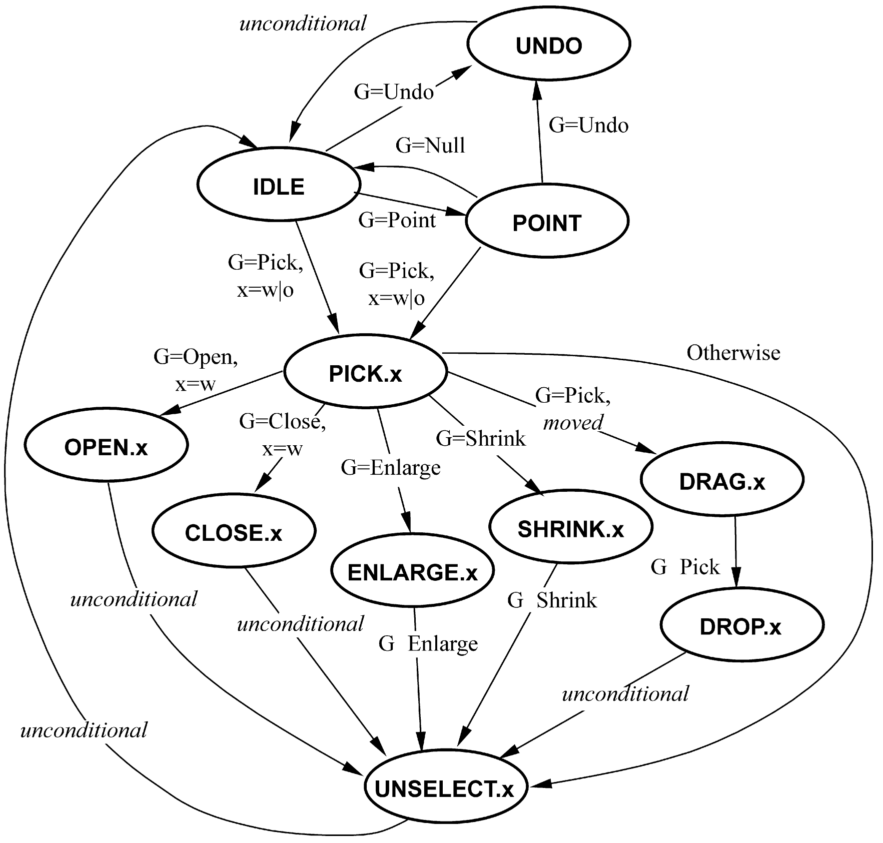


Fig. 14. State diagram for gesture-based GUI application. In the above diagram, G represents the label output from the gesture recognition module, x represents the item under the cursor, which can be window (w) or object (o). Note that for simplicity, only one ENLARGE state and one SHRINK state are shown.

will occur only when the gesture label is *Pick* and the cursor is over a window or object. On the other hand, some state transitions are unconditional. One example of such a transition is from UNSELECT.x to IDLE.

8.1.1. Operation of the state machine

The gesture output controls the operation of the state machine, and in many cases, the state changes depend only on gesture output. This causes the system to behave erratically when the raw gesture output is incorrect, for example, at the transition from one gesture to another. The nature of both the HMM chains and RNNs are such that their outputs are accurate only after a few frames when sufficient data is available to allow a classification with fairly high confidence. Thus during the transition between gestures, the output from the gesture recognition module is likely to be incorrect, with the possibility of erroneous state transition. To address this issue, the state transition decisions are made for a group of five consecutive frames, and an overall gesture label is assigned to each group. In each group, the appearance of each of the 14 gesture labels is counted. When a gesture label x appears more than twice in a group of frames, the overall gesture label for this group is taken to be x. Otherwise, the overall gesture label is *Null*.

The above scheme regulates and reduces erroneous state transitions. In choosing the number of frames in a group, accuracy of the overall gesture label and the response time of the system was taken into account. A larger group will generally result in better accuracy, since this has the effect of smoothing the gesture outputs. On the other hand, if the group size is larger, the response time of the system is adversely affected. The group size of five was found to be an optimal choice by testing different group sizes from 3 to 10. At an output frame rate of about 22 fps, state transitions made at five frame intervals have a maximum delay of about 250 ms, which was perceived to be tolerable.

8.1.2. Temporal segmentation

The gesture recognition module has no means to perform temporal segmentation, that is, to decide which frame is the start of a new gesture. This task is shifted to the state machine. Temporal segmentation is essential because the outputs of both the HMM chains and RNNs are cumulative. In HMM chains, the outputs are the accumulated log probabilities. In order to recognize new gestures, these accumulated probabilities need to be reset to zero. In RNNs, a reset of feedback nodes is required before recognizing new gestures, since that is how they were trained: the outputs from the feedback nodes were reset to their initial values before the networks were trained with a new gesture sequence.

Temporal segmentation can be simply performed by resetting the accumulated probabilities of the HMM chains and the feedback nodes of the RNNs whenever the overall gesture label for a group of frames differs from the previous one. This change can be taken as an indication that the user is changing the gesture, and temporal segmentation can be done at this point. In addition, temporal segmentation is also carried out whenever the overall gesture output is null. This is the case when the gesture output is relatively volatile, such as when the user is changing his/her gestures.

8.2. Discussion

The demonstration prototype was tested by six users who did not train the system. They were asked to perform some simple operations such as moving and re-sizing windows and objects, and opening/closing of windows. The feedback from the users on the ease of use of the system is summarized below.

Each of the users was asked to perform a series of simple tasks, all of which included at least the following: (i) minimizing and restoring windows, (ii) moving objects from one window to another, (iii) shrinking and enlarging of windows/objects, and (iv) undoing a previous command. All of the users were able to perform all required tasks after a brief demonstration on how to control the system using various gestures. Most of the users found the choice of gestures intuitive, especially the *Close* and *Open* gestures used to minimize and restore the windows. The *Pick* gesture to select windows and objects also had a natural appeal to most users.

The response time of the system was found to be adequate, with the exception of the *Undo* gesture. Four out of six users found it slow as the system took more than one second to register their *Undo* commands. This is due to the *Undo* gesture having a slightly longer recognition time compared to other gestures. An analysis of the recognition results showed that for the first few frames of the *Undo* gesture, the classifiers will generally classify it to some other gesture (like *Point* or *Open*). This is because the *Undo* gesture is composed of similar hand poses as these gestures. The only difference is the oscillatory motion in the *Undo* gesture, and this oscillatory motion can only be identified after a few frames.

One area where most users found it disadvantageous to use gesture commands was the limited precision of control they had. In particular, they found it difficult to move the cursor to the precise location they wanted. In addition, re-sizing windows and objects to the desired sizes often required several trials. This was attributed to segmentation and wrist-cropping errors that caused the estimation of the hand centroid to be noisy, and thereby affected cursor position. Furthermore, the fact that the camera resolution is 320×240 in contrast to the desktop size of 800×600, adds to the difficulty of precise control over the cursor position.

The constraint imposed by the limited view of the camera makes it necessary to repeat hand motion in the *Point* gesture in order to move the cursor by large distances. This would also have been the case when users performed two-handed gestures since it is more likely for one of the hands to go out of the field of view. Fortunately, the system is designed so that all two-handed gestures (i.e. the *Shrink* and *Enlarge* gestures) can also be interpreted using the one hand in view. This not only gives users the freedom to choose a more natural form of gesture, but also allows the system to perform normally when one of the user's hands accidentally moves out of the camera's vision when performing two-handed gestures.

In order to resolve the problems related to moving the cursor and re-sizing of objects/windows, a few options are provided to the user. The *Tweak* panel in the application interface allows the user to adjust the system parameters which control the cursor movements and re-size speed. The user can set the amount by which a window/object is re-sized at each step, causing the items to re-size at a speed that suits the user. Users can also set the scale factor by which the motion of the hand centroid is mapped to the cursor movement. A large value will cause the cursor to be moved a larger distance when the user moves his/her hand, and vice versa.

Here, we hoped to demonstrate the feasibility of integrating natural gesture into traditional user interfaces, rather than replace the mouse by gesture. Users who tested the application found the gestures intuitive. Though it is impossible to move the cursor with a precision that devices such as the mouse can provide, it was found to be adequate. Moreover, this shortcoming can be overcome by using other HCI devices, such as the touch screen where users can specify the exact position by touching the screen. Gesture lends itself readily to be incorporated into such interfaces.

9. Conclusions

In this paper, we considered a vision-based system that can interpret a user's gestures in real time to manipulate windows and objects within a graphical user interface. Every frame from the acquired image sequence was processed through five different stages, viz. hand segmentation, wrist-cropping, feature extraction, pose classification, and gesture recognition. Output from the gesture recognition module was used in an application to control windows and objects in a simulated GUI.

Hand segmentation utilized color and motion information to transform the captured color images into binary hand blobs. The segmentation algorithm developed can handle one or both user's hands in the image, and output each hand as a singly connected blob. The segmented hand was isolated from the rest of the arm by using a wrist-cropping algorithm. This algorithm made use of width and contour heuristics to estimate the location of the wrist in the segmented hand blob, and the hand was extracted from the estimated wrist location. Fourier descriptors were used to represent the segmented, wrist-cropped, hand blobs. These features were presented to a pose classification network comprising of radial-basis function (RBF) networks. The output network was essentially the likelihood of the hand being in each of the five defined hand poses. The pose classification network was capable of combining two hand blobs from a single image to yield a unified pose classification output.

Gesture recognition was then performed based on the pose and motion information. We also explored different methods for gesture recognition. HMMs with discrete and continuous densities were trained and tested. It was found that the continuous HMM performed better than the discrete HMM, with a test result of 90.2% correct classification, versus the 86.4% for the discrete HMM. RNNs were also used to recognize gestures and their performance was compared with HMMs. The merits of using different training algorithms for the RNN were explored, and an efficient learning procedure was developed based on a suitable modification of the real-time recurrent learning algorithm. The RNN gave a test result of 89.8% correct classification. The outputs from the continuous HMM and the RNN were combined using a linear layer of weights, as it was found that the two independent classifiers were able to supplement each other, to yield a combined classification accuracy of 91.9%. We also showed that the inclusion of an intermediate pose classification stage resulted in reduced computations in training and recognition, an important factor to consider for real time applications.

Output from the gesture recognition module was used with a state machine to control the windows and objects in an example GUI application. Real time processing rates of up to 22 fps were obtained. Users who tested the application were able to perform different tasks using gestures with ease, and they found the use of gestures to be intuitive. The main disadvantage of the system was the imprecision in positioning the cursors and resizing of objects, and this was addressed to some extent with a *tweak panel* to set interface parameters. Future research work may involve the use of modern HCI devices (such as the touch screen) to augment the use of gestures in HCI. Other natural forms of HCI, such as speech, may be incorporated so as to achieve a natural, multi-modal approach in the way humans interact with the computer.

References

[1] V.I. Pavlovic, R. Sharma, T.S. Huang, Visual interpretation of hand gestures for human–computer interaction: a review, IEEE Trans. Pattern Anal. Mach. Intell. 19 (7) (1997) 677–692.

[2] R. Kadobayashi, K. Nishimoto, K. Mase, Design and evaluation of an immersive walk-through application for exploring cyberspace, Proc. Third Int. Conf. Autom. Face Gesture Recogn. (1998) 534–539.

[3] V.I. Pavlovic, R. Sharma, T.S. Huang, Gestural interface to a visual computing environment for molecular biologists, Proc. Int. Conf. Autom. Face Gesture Recogn., Killington, Vt October (1996) 30–34.

[4] W.T. Freeman, K. Tanakan, J. Ohta, K. Kyuma, Computer vision for computer games, Proc. Int. Conf. Autom. Face Gesture Recogn. October (1996) 100–105.

[5] R. Kjeldsen, J. Kender, Towards the use of gesture in traditional user interface, Proc. Int. Conf. Autom. Face Gesture Recogn. October (1996) 66–71.

[6] C.J. Cohen, L. Conway, D. Koditschek, Dynamical system representation, generation, and recognition of basic oscillatory motion gestures, Proc. Int. Conf. Autom. Face Gesture Recogn., Killington, Vt October (1996) 60–65.

[7] S. Nagaya, S. Seki, R. Oka, A theoretical consideration of pattern space trajectory for gesture spotting recognition, Proc. Int. Conf. Autom. Face Gesture Recogn., Killington, Vt October (1996) 72–77.

[8] B. Raytchev, O. Hasegawa, N. Otsu, User-independent online gesture recognition by relative motion extraction, Pattern Recogn. Lett. January (21) (2000) 69–82.

[9] R. Liang, M. Ouhyoung, A real-time gesture recognitiion system for sign language, Proc. Third Int. Conf. Autom. Face Gesture Recogn. (1998) 558–567.

[10] M. Hoch, A prototype system for intuitive film planning, Proc. Third Int. Conf. Autom. Face Gesture Recogn. (1998) 504–509.

[11] J. Yang, A. Waibel, Tracking Human Faces in Real Time, Technical Report CMU-CS-95-210, Department of Computer Science, Carnegie Mellon University, 1995.

[12] E.S. Koh, Pose Recognition System, BE Thesis, National University of Singapore, 1996.

[13] G.H. Granlund, Fourier preprocessing for handprint character recognition, IEEE Trans. Comput. 21 (1972).

[14] J.E. Moody, C.J. Darken, Fast learning in networks of locally-tuned processing units, Neural Comput. 1 (1989) 281–294.

[15] R. Schalkoff, Pattern Recognition: Statistical, Structural And Neural Approaches, Wiley, New York, 1992.

[16] D.S. Broomhead, D. Lowe, Multivariable functional interpolation and adaptive networks, Complex Syst. 2 (1988) 321–323.

[17] L.E. Baum, An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov process, Inequlities 3 (1972) 1–8.

[18] R.J. Williams, D. Zipser, A learning algorithm for continually running fully recurrent network, Neural Comput. 1 (1989) 270–280.

[19] A.W. Smith, D. Zipser, Learning sequential structure with the real-time recurrent learning algorithm, Int. J. Neural Syst. 1 (2) (1989) 125–131.

[20] R.J. Williams, J. Peng, An efficient gradient-based algorithm for on-line training of recurrent network trajectory, Neural Comput. 2 (1990) 490–501.

[21] D. Zipser, A subgrouping strategy that reduces complexity and speeds up learning in recurrent networks, Neural Comput. 1 (1989) 552–558. MIT.

[22] R.J. Williams, D. Zipser, Experimental analysis of the real-time recurrent learning algorithm, Connection Sci. 1 (1989) 87–111.

[23] Y.L. Lu, M.W. Mak, W.C. Siu, Diminish the computational burden of real time recurrent learning algorithm by constrained sensitivity, Proc. Int. Conf. Neural Netw. Signal Process. (1995) 223–226.

[24] Y.L. Lu, Improved Recurrent Learning Algorithm, Masters Thesis, The Hong Kong Polytechnic University, 1996.

[25] F. Zernike, Refraction theory of the cutting method and its improved form, the phase contrast method, Physica 1 (1934) 689–704.