Gesture Recognition for Remote Collaborative Physical Tasks Using Tablet PCs

Jiazhi Ou, Xilin Chen, Jie Yang
School of Computer Science
Carnegie Mellon University, Pittsburgh, PA, USA
{jzou,xlchen,yang+}@cs.cmu.edu

Abstract

In this paper, we present effective and efficient gesture recognition algorithms for building a system to support remote collaborative physical tasks using tablet PCs. We discuss problems of gesture recognition in detail. We use a variable window to extract curvature changes to form invariant local features. We then employ a hierarchical classifier that consists of hidden Markov model (HMM) and decision tree classifiers. HMMs are utilized to classify closed gestures, while decision tree is used to classify open gestures. Experiment results show that the overall accuracy for all gestures is 96.4%. Accuracies for closed gestures and open gestures are 96.9% and 96.1% respectively. We have developed a prototype system integrating gesture and live video to support collaboration on physical tasks. Besides normal gesture recognition, the system also supports gesture fitting, freehand drawing, and the combination of the two. The system can support both human to computer interaction and human to human communication.

1. Introduction

Collaborative physical tasks refer to tasks in which two or more people interact with real objects in the 3D world. They play an important role in many domains, such as education, industry, and medicine. As working force becomes increasingly distributed, there is a critical need for technologies to support collaborative physical tasks. Prior studies of physical collaboration suggest that people's speech and actions in this context are inherently multimodal, intricately related to the position and dynamics of objects, other people, and ongoing activities in the environment [1, 2, 3, 5, 7]. In particular, communication during physical tasks combines both speech and gesture. During verbal communication, people use several types of gestures to clarify or enhance their messages [1, 6]. Pointing gestures are used to refer to task objects and locations. Representational gestures, such as hand shapes and hand movements, are used to represent the form of task objects and the nature of actions to be used with those objects, respectively.

In face-to-face collaboration on physical tasks, people can readily combine speech and gesture because they share the same environment. Combining speech and gesture is more complicated in remote collaboration because of the need to reference external objects. Previous studies of video systems to support remote collaboration on physical tasks (e.g., [2, 3]) have repeatedly observed that remote participants have difficulty communicating because of their inability to gesture or point at objects in the workspace. These communication problems have negative effects on performance, in that remote performance on physical tasks takes longer than performance when the collaborators are co-located. To facilitate remote communication on physical tasks, it is thus necessary to provide a tool that allows remote collaborators to use both speech and gesture in the same way they would do so if co-located.

The majority of previous systems for remote collaboration, however, have paid little attention to supporting activities that must refer to the external spatial environment. Consequently, gestural communication is not explicitly supported by most existing computer supported cooperative work (CSCW) technologies. The objective of this research is to develop technologies to support communication through speech and gesture during collaborative physical tasks. In the current work, we aim to develop an inexpensive multimodal system that can be easily incorporated into existing video conferencing systems. Our goal is to allow remote collaborators to communicate about their physical world through speech and gesture with the same ease as they can do so when co-located.

We approach the problem by using pen-based gesturing over video stream. The video stream plays a dual purpose in the proposed paradigm: (1) it establishes remote communication among collaborators, and (2) it provides gestural communication media. The system allows collaborators to share the workspace through video connections. It also provides remote support for gesture by overlaying pen-based gestures over video streams. Our goal is to devise a system, using desktop PC and Tablet PC platforms, that enables speakers and listeners to produce and interpret both pointing and representational gestures as readily as they do in face-to-face settings. The preliminary idea has been evaluated by implementation of a cursor pointing device [4]. User studies concluded that cursor pointing is valuable for collaboration on physical

tasks, but that additional gestural support will be required to make performance using video systems as good as performance working side-by-side.

In this paper, we present a system supports remote interaction using gestural communication over video streams using video cameras, tablet PCs, and desktop PCs. The system allows collaborators to share the workspace through video connections. It also provides remote support for pointing and representational gesture by overlaying pen-based gestures on video streams. Our objective is to find an effective and efficient way to recognize and fit gestures. Therefore, we employ a hierarchical scheme consists of hidden Markov models (HMM) and decision trees.

2. Pen-based Gesture Recognition

With the progress in hardware, touch screen has been widely used in various computational devices, such as PDA and Tablet PC, etc. This highly stimulates the researchers' interests in Pen-based technologies. We use a pen-based interface to implement gestural communication over video streams. In a pen-based interface, a gesture is represented by the trajectory of moving points. The task of gesture recognition is to classify a sequence of points into different predefined classes.

The problem we address in the paper differs from online handwriting recognition [11, 13, 16]. Although the number of gestures defined in our system is smaller than those in online handwriting recognition systems, the drawing style of pen-based gesture recognition is more arbitrary. In a handwriting recognition system, input samples will have almost the same orientation and size, while in our system, gestures are identical under affine transformation. For example, straight arrows may represent the command to move a camera, and their lengths and directions are parameters of each command. While most online handwriting recognition algorithms use x/y coordinates of sample points as input features, we could not apply those technologies directly in gesture recognition because they are not invariant of rotation and scaling.

Technically pen-based gesture recognition can be viewed as a graph recognition or classification problem if we do not take their scopes and commands into account. The earliest pen-based work is Sutherland's Sketchpad [18], which is also the first graphic user interface. [8] described a gesture-based interface called GRANDMA. GRANDMA specifies single-stroke gestures drawn by mouse movement, beginning with the press of a mouse button. He used a statistical method for gesture recognition. First, thirteen locally and globally geometrical features are extracted to represent the input stroke. Then, the feature vector is classified as one of the C possible gestures via a linear evaluation function. Finally, a closed formula is used to calculate the weights in the function.

Jorge and Fonseca used decision tree and fuzzy logic method for online graphics recognition [9]. The recognition process starts from the first pen-down event until a set timeout value after the last pen-up. First, global geometric properties of the input stoke are extracted as features. Second, a decision tree is applied to filter out unwanted shapes using distinctive criteria. Third, fuzzy logic is used to associate degrees of certainty to recognized shapes.

Jin et al. [10] proposed an on-line sketchy graphics recognition algorithm. There are four pre-processing steps. First, it removes redundant intermediate points using polygonal approximation. Second, agglomerate points filtering is employed to reduce hooklets at the end of the lines and circlets at the turning corners. Third, end point refinement is used to delete extra points for a self-crossed stroke and extend endpoints for an open stroke. Fourth, convex hull is calculated to select n vertexes to represent the original line. After the pre-processing, m points from the original n vertexes are selected with a recursive vertex combination algorithm. The closed-shape graph is classified according to the number m.

The problem of supporting gesture recognition in remote collaboration on physical tasks differs from what has been explored in previous research in many ways. Technologies for supporting gesture communication in CSCW must be different from those supporting human-computer interaction (HCI). In HCI, a gesture-based interface, which translates input gestures to coded data, is designed to implement human-computer communication through human-like styles. Humans are in the human computer interaction loop. The gesture recognition system recognizes the predefined gestures. On the other hand, the function of a gestural tool in CSCW systems is to mediate human-human communication. Instead of the human in the loop, we have put the computers into the human communication loop. The role and functions of the computer have been changed. Furthermore, a gesture tool might ideally have both HCI and CSCW functions. As an HCI tool, gestures can be used as an input device for camera control (pan, tilt, zoom). As a CSCW tool, gestures are intended to communicate meaning to a remote partner. In design of our system, we have fully considered how gesture recognition can be implemented to facilitate both to enhance interpersonal communication and as a camera control device. Unlike existing gesture recognition systems used for human computer interaction, which support recognition only of predefined gestures, our system supports recognition of predefined gestures, freehand drawing, and a combination of the two. We propose to use a hierarchical structure of classifiers that consist of hidden Markov models (HMMs) and decision trees to achieve good performance for the gesture

recognition task. We discuss our gesture recognition algorithms in detail below.

2.1. Preprocessing

Like many other pattern recognition tasks, preprocessing is necessary for enhancing robustness and recognition accuracy. We have performed two different preprocessing techniques before the feature extraction.

A user will draw a gesture at different speeds. This means that the sampling rate for the same gesture is not a constant, i.e., for a given period of time, the number of samples is changeable. Several methods can be used for this task, such as linear, B-spline, Bezier interpolation, etc. In the current system we apply linear interpolation before resampling the sequence of points.

Most of the people draw gestures with hooklet-like segments, either in start or in the end. [10] states that the hooklet-like segments happen at the end of the sketchy lines, but we find that the hooklets are more likely to happen at the beginning of gestures. Therefore, if we find a sharp curvature change after a few points from the start, we remove those points, as shown in Figure 1.

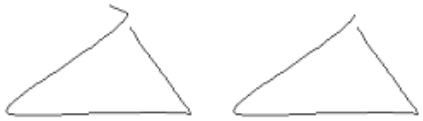

Figure 1. An example of removing the hook at the beginning of the gesture.

2.2. Feature extractions

Hand written signal recognition systems uses either local features ([11, 13, 14]) which include x (y) offsets, slope angles, curvatures, etc. or global features ([8, 9, 10, 11]). In our system we first extract local features, which are more informative. Because we want to extract features that are insensitive to affine transformation, we use the maximum curvature within certain window as the feature to describe the gestures. To obtain a stable feature, we use a window W, which contains a gesture segment C_s ($\overset{\frown}{AB}$) with length of L, as shown in Figure 2. We measure the curvature θ of the segment as following:

1. Select points C, and D on the segment, so that

$$\int_A^C dC_s = \int_B^D dC_s = c\int_A^B dC_s \quad , c \in (0,0.5] \text{ is a constant}$$

2. Calculate the angle θ between AC and BD to measure the curvature of the segment. The larger the angle θ is, the larger the curvature.

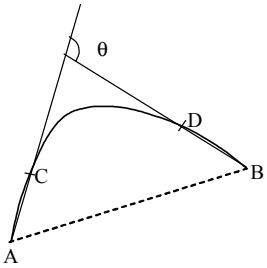

Figure 2. An illustration of curvature calculation.

When we compute the feature, we need to consider that the size of window and the coefficient c. Note that the curvature may change with the scale of the gesture. Therefore a small window may focus only on the detail of the curve and ignore the real important shape information. Similar to dynamic time wrapping (DTW) in speech recognition, we use a variable window size to calculate the curvature. We define the window size as:

$$L = \frac{\int dC_{gureture}}{SegmentNo},$$

where $SegmentNo \in [MinNo, MaxNo]$.

The curvature sequence $\{\theta_i,\ i = 1,\ldots n\}$ is the feature that we use for classification. In order to differentiate the rotation of the slope change (clockwise or counter-clockwise), we add a sign to each θ_i. In addition, some gestures are easy to classify with global features. We will introduce them in detail specifically in Section 2.3.2.

2.3. Hierarchical Classifier

Gesture classification is a key part for gesture recognition. Researchers have employed different classification technologies for online pen-based input, such as decision tree, HMM, and Bayesian network, etc. Our design principle is simple, high accuracy, and robust. In order to balance simplicity, accuracy, and robustness, we combine decision tree and HMM into a classifier in a hierarchical structure.

Our system recognizes 12 predefined gestures, which can be classified into two categories by the distance between the first point and the last point of the gesture. The first category is called *Closed Gestures*, which includes Ellipse, Triangle, Quadrangle, Pentagon, and Star (Figure 3). The other is called *Open Gestures*, which includes Straight Line, Check, Cross, Delete, Arrow, and Round Arrows (Figure 4).

Ellipse Triangle Quadrangle Pentagon Star

Figure 3. Closed gestures.

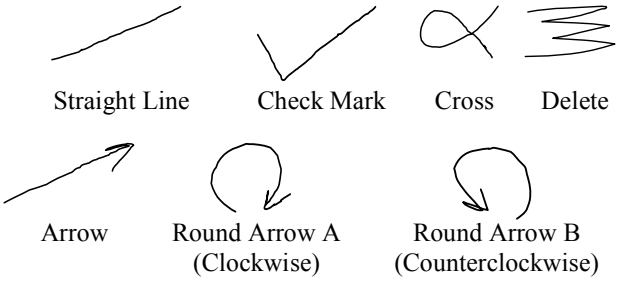

Figure 4. Open gestures.

To further classify gestures within closed gesture set and open gesture set, we use different classifiers. Given an input gesture, if we know the number of vertexes (large curvature changes) and their orders, we can classify it by simple rules. Vertexes can be detected by sharp curvature changes, which can be achieved by the threshold method. However, for the closed gestures, the concept of 'angle' is quite vague sometimes. Curvature changes around obtuse angles are small. Arcs with low curvatures (flat arcs) are close to lines, arcs with high curvatures (sharp arcs) are close to angles, and round angles are close to arcs (seeing Figure 5). An explanation on this is that when a user draws several continues corners, they could not control the distribution of curvature change well. Hence ellipses with sharp arcs, quadrangles with round angles, and polygons with obtuse angles are hard to classify correctly. Furthermore, if we rely on the angles detected, the performance is very sensitive to the features. Mis-detection of one or more angles due to a slightly noisy feature will cause a wrong classified result. Therefore, we model lines, angles and arcs statistically with HMMs, thus local features are fully utilized. And we will present a novel constrained HMM in the next section.

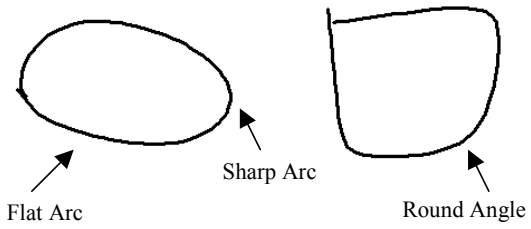

Figure 5. Examples of ambiguous angles.

Open gestures, on the other hand, have more significant angles, and most of them are not so sensitive to the noisy data. If we miss an angle for Arrow or Delete gestures, they still can be recognized correctly by the other detected angles. We use a decision tree, which is fast and intuitive, to classify gestures in this category.

The diagram of this hierarchical is shown in Figure 6:

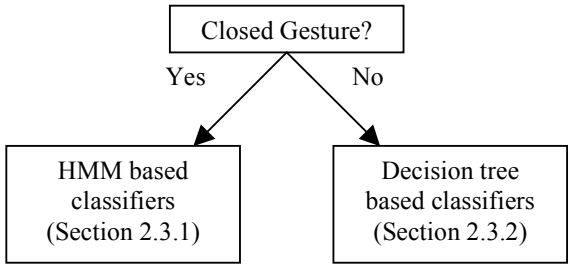

Figure 6. A hierarchical classifier.

2.3.1. HMM Based Classifiers for Closed Gestures. Hidden Markov models are well known to model sequential data and were successfully applied to speech recognition, handwriting recognition, and other pattern recognition tasks [12, 13]. We found that it produced satisfactory results to classify closed gestures.

Observable Symbols

As described in Section 2.2, given an input gesture, we extract curvature information θ_i, with sign that decides the direction of the drawing (clockwise or counter-clockwise). Because we don't care this direction for the closed gestures we defined in our task, we use the absolute values as input features:

$$O = \{ o_1, o_2, \ldots, o_n \}, \text{ where } o_i = |\theta_i|.$$

Discrete HMMs were used as generative models to generate the features. Continuous value o_i is quantized with 20 levels (0 to 19) evenly when it is smaller than a threshold. Values greater than this threshold are quantized to one level (20).

2-State HMMs

Two 2-state HMMs were constructed to model polygons (Triangles, Quadrangles, Pentagons, and Stars) and Ellipses respectively. The rationale behind it is states for polygons represent edges and angles respectively, and states for ellipse represent flat arcs and sharp arcs respectively. Training was processed using traditional Baum-Welch algorithm. Initial state probability vector π, transition matrix A, and output probability matrix B were updated. The distribution of output probabilities for each state (B matrix) is shown in Figure 7. We can see that edge state and flat arc state tend to generate features correspond to low curvatures. While angle state and sharp arc state tend to generate features that correspond to higher curvatures.

We have no prior knowledge on the probabilities of these gestures, and assume they have equally the same occurring probabilities. Therefore, by using the Bayesian decision rule to classify polygons and ellipses, we have:

"Polygon": If $\Pr(O \mid \text{Polygon HMM}) > \Pr(O \mid \text{Ellipse HMM})$

"Ellipse": Otherwise

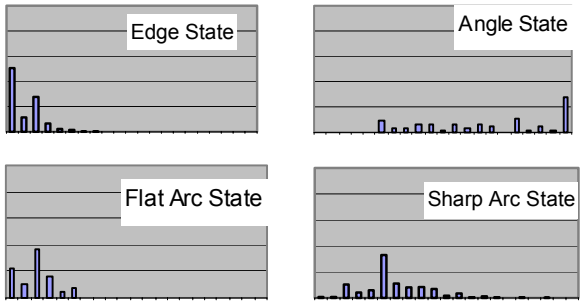

Figure 7. An illustration of output probabilities for each state. The x-axises are the curvatures and the y-axises are the probabilities.

Constrained HMMs for Polygons

To find the number of edges or angles generated by the polygon HMM, one possible way is to track the Viterbi path of the feature sequence and count the number of transitions between states, the other way is to model this number of transitions explicitly. The latter was implemented in our system because it has more fertile information.

First we define $PATH_m$ as the set of state paths that have exactly $2\times m$-1 transitions between edge state and angle state in the polygon HMM. More specifically, paths in $PATH_m$ should repeat {edge state, edge state, …, edge state, angle state, angle state, …, angle state} m times. We model the probability of the number of edges given the input feature as:

$$\Pr(m \quad edges \mid O) \propto \sum_{path \in PATH_m} \Pr(O \mid path, HMM) * \Pr(path \mid HMM) \quad (2)$$

To calculate (2), we expand the polygon HMM manually to a left to right model HMM′ (Figure 8).

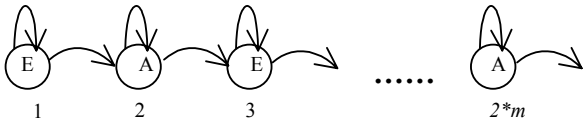

Figure 8. The expanded left to right model HMM′. *E* and *A* represent edge state and angle state respectively.

And equation (2) becomes:

$$\sum_{path \in PATH_m} \Pr(O \mid path, HMM) * \Pr(path \mid HMM)$$

$$= \Pr(O, last \quad state = 2 * m \mid HMM') \quad (3)$$

Note that probability calculated in equation (2) is an approximate value. It's biased because when m is larger, there are more paths. Therefore, we penalize the model with more edges. That is, the model with m edges has a positive factor *penalty(m)*, which satisfies

$$penalty(i+1) > penalty(i), i = 3,4,...$$

The classification of gestures in polygon set is:

$$\arg\max_m \{\log \Pr(m \quad edges \mid O) - penalty(m)\}$$

The penalty factors were tuned using the training data.

Both Pentagon and Star have five edges. We need a global feature called *crossing* to differentiate them quickly. It is defined as whether there exist two separated segments that intersect each other. We use the way presented in [11] to calculate *Crossing*. Obviously Star has enough *Crossing* and Pentagon doesn't have any.

This method can be easily extended to other polygons or folded lines that have a fixed number of edges.

2.3.2. Open Gestures

Higher Level Features

While we don't have the 'blurred angle' problem in the open gesture category, we could use a threshold to detect vertexes, that is, an angle is marked once the curvature is larger than the threshold. But instead, we want to learn the thresholds from the training examples. Therefore, we first specify a set of thresholds $\{thred_k \mid k=1, 2, 3, 4$ and $thred_i < thred_{i+1}\}$. For each threshold $thred_k$, the number of vertexes V_NUM^k and their positions pos^k_1, pos^k_2, …, pos^k_v are recorded as higher level features. To decide whether it contains curves or not, we use the following ratio to see how well the detected vertexes fit the original gesture:

$$Curve^k = \frac{\sum_{i=1}^{n-1} Dist((x_i, y_i), (x_{i+1}, y_{i+1}))}{\sum_{i=0}^{V\_NUM^k} Dist((x_{pos^k_i}, y_{pos^k_i}), (x_{pos^k_{i+1}}, y_{pos^k_{i+1}}))} \quad (4)$$

Where (x_i, y_i) and resampled points, $pos^k_0=1$ and $pos^k_{v+1}=n$.

To detect arrows, we measure the ratio between the lengths of the first and the second line segments:

$$Ratio^k = \frac{Dist((x_{pos^k_0}, y_{pos^k_0}), (x_{pos^k_1}, y_{pos^k_1}))}{Dist((x_{pos^k_1}, y_{pos^k_1}), (x_{pos^k_2}, y_{pos^k_2}))} \quad (5)$$

Features that are not dependant on the value of thresholds are *Crossing* described in last section and the majority of signs of curvatures *Sign*.

Decision Tree Classifier

In the training phase, for each gesture we extract (V_NUM^1, …, V_NUM^4, $Curve^1$, …, $Curve^4$, $Ratio^1$, …, $Ratio^4$, *Crossing*, *Sign*) as feature vector. C4.5 algorithm ([17]) is *applied* to select the most distinguishable attributes and construct the decision tree. Attributes included in the output decision tree are shown in Table 1.

Table 1. Attributes of the output decision tree

V_NUM^1	Discrete
V_NUM^3	Discrete
$Curve^2$	Continuous
$Ratio^1$	Continuous
$Ratio^2$	Continuous
Crossing	Discrete
Sign	Discrete

In the test phase, only 7 attributes (out of 14) listed in Table 1 are needed.

2.4. Gesture Fitting

To better support human gestural communication, we provide a gesture fitting tool. Our system recognizes the intentional shape a user is drawing and regularizes it. We would like users to be able to draw arbitrary sketches, not restricted by the graphs we can fit. While many free hand sketching interfaces aim at accurate approximation to the input strokes (e.g. [14]), we feel it is unrealistic in our task if parts or all of a single gesture are potentially free hand drawings. To address this situation, we proposed to combine gesture fitting with freehand drawing.

A single gesture is segmented by the vertexes we have detected. We verify a line segment using the most intuitive way. If the distance between their end points is shorter than the original length to some extend, we reject it. Otherwise we connect them with a straight line. Instead of trying to approximate the curves, which may be intractable in some situations, we leave them as free hand drawings connected with other line segments (either recognized gestures or free hand drawings).

3. A Prototype System

We have incorporated the gesture recognition scheme into a system to facilitate gesturing over video within the context of an instructional collaborative physical task where two or more people interact with real objects in the 3D world. The architecture of the system is shown in Figure 9. The workspace is visually shared through video cameras and equipped with tablet PCs, desktop PCs or other handheld devices. Real-time video streams from these cameras are sent to collaborators' computing devices in the workspace. A helper can make freehand drawings and pen-based gestures on the touch sensitive screen of a computing device, overlaid on the video stream, just like using a real pen on a piece of paper in a face-to-face setting. The results are observable by all collaborators on their own monitors. Details of the implementation are discussed in the remainder of this section.

Since we want to overlay gestures over video streams and display them together, we need two running threads: one is for video communication, the other is for gesture communication. Because they are concurrent procedures—i.e., the order of these two threads are undetermined—displaying them directly on the screen will have flashing effect. Therefore, an image buffer is prepared before the ultimate image is displayed.

Video cameras are essential to facilitate remote collaboration. In order to reduce potential network delay caused by the video server, we opted to use network IP cameras, which are inherent servers, to solve the problem of distributing network traffic. Each network IP camera is a server and connected to the network independently; other computers on the network can be its clients. Once started, a network IP camera opens a TCP/IP port and waits for its clients. When a connection is established, the server's status message and the client's authentication messages will be exchanged. If the client is authenticated, video data will be sent in JPEG format upon a client's image request message. By using this technique, the video flow and the process overhead is shared by all network IP cameras. Furthermore, because jitter is more likely to happen in an Internet environment because of a higher chance of collision, we establish a local area network (LAN) for our preliminary tests of the system. A wireless router is used to connect network IP cameras, workers', and helper's computers. The devices communicate with each other locally, isolated from the Internet. In this way, we can minimize effects caused network delay. Disruption of remote gestures by network jitter was investigated by Gutwin [15], and is an important issue we will be addressing in future work.

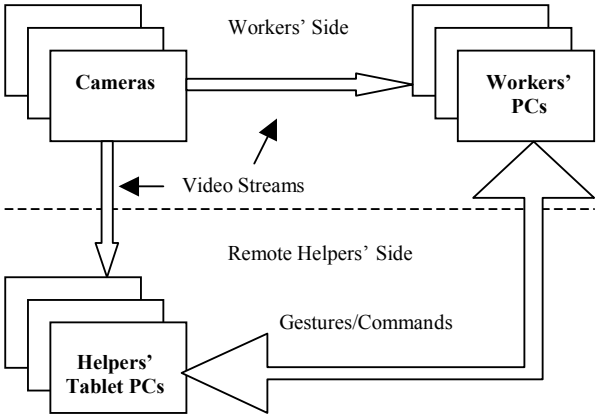

Figure 9. Overview of system architecture.

After connecting to network IP cameras, the communication among collaborators' computing devices is also in client-server mode. For example, the worker's computer can be a server and the helper's computer can be clients. A socket is created on the worker's computer. It waits and accepts client sockets from the helper's computer. After the establishment of a connection, a helper can send remote gestures and commands through socket communication, or vice versa. The trajectories of freehand drawing and gesture recognition results are observable on all collaborators' monitors.

Pen-based gesture and freehand drawing consist of sequences of points. Each sequence starts from the pen touching the screen and ends when the pen is lifted. When the helper is drawing, the sequence of points will be added to a link list of the current gesture and sent to the workers' computers simultaneously. While drawing, the helper can choose among freehand drawing, gesture

recognition, or drawing normalization. In freehand drawing, what is sketched will be shown exactly as drawn on the screen. In gesture recognition mode, a predefined gesture will be recognized and a certain command will be executed. In the drawing normalization mode, the current sequence of points will be sent to a gesture recognition module immediately after the user lifts the pen from the screen. The recognition module recognizes the shape that the user is trying to draw (e.g., arrow, circle) and returns a set of parameters to approximate the recognized shape. The interface, on the other hand, will use these parameters to synthesize and display the normalized shapes. There are several parameters that a user can set for sketching, including pen width and color of the drawing.

In the current experiment setting, a user can make two sets of commands besides sketching. The first set of commands concern erasing gestures already drawn. A user can choose remove all gestures, the first gesture, or the latest gesture. In addition, we are testing an automatic fade-out function, in which each gesture fades out after a predefined time.

The second set of commands is "undo/redo". There is a pair of buttons and a user can always undo the last action (i.e., drawing or erasure) or redo what is undone. Inverse action is taken after each undo/redo command as shown in Table 2.

Table 2. Actions to Take for Undo/Redo

Last Action	Undo	Redo
Draw a Gesture	Erase Last Gesture	Resume Last Gesture
Erase Last Gesture	Resume Last Gesture	Erase Last Gesture
Erase First Gesture	Resume First Gesture	Erase First Gesture
Erase All Gestures	Resume All Gesture	Erase All Gestures

The third set of commands is to take a 'snapshot'. If a user wants to keep an image at any time, he/she can use the snapshot command to save the image as a JPEG file on the local disk.

4. Experimental Results

4.1. Experiment Setup

In order to demonstrate the feasibility of the proposed methods, we have performed experiments to evaluate accuracy of gesture recognition. We collected a total of 1337 gestures from 14 people. 666 gestures from 7 people were used as training data to train HMMs and tune the thresholds. 671 gestures from other 7 people were used as test data. Then we switch training data to test data and test data to training data. The performance was evaluated with the test data in these two experiments.

We evaluated the gesture recognition accuracy in macro level, which is computed by first calculating the accuracy for each gesture individually, then averaging the accuracy of each class. The overall accuracy of 12 gestures is 96.4%.

4.2 Results of Closed Gestures

The accuracy of closed gestures is 96.9%. Results of individual gestures are shown in Table 3.

Table 3. Recognition Results for Closed Gestures

Gestures	Accuracy	Gestures	Accuracy
Ellipse	99.1%	Triangle	100.0%%
Quadrangle	89.2%	Pentagon	96.4%
Star	100.0%		

4.3. Results of Open Gestures

The accuracy of open gestures is 96.1%. Results of individual gestures are shown in Table 4.

Table 4. Recognition Results for Open Gestures

Gestures	Accuracy	Gestures	Accuracy
Straight Line	100.0%	Check Mark	97.3%
Cross	98.2%	Delete	92.8%
Arrow	94.6%	Round Arrow A	94.6%
Round Arrow B	95.5%		

4.4. Gesture Fitting

We also tested combination of freehand drawing and gesture fitting together. An example of gesture fitting combined with freehand drawing is illustrated in Figure 10.

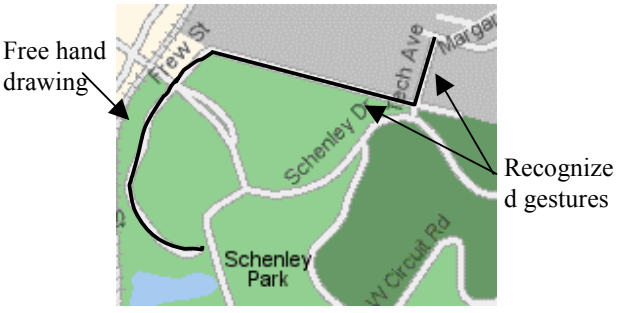

Figure 10. An example of gesture fitting. Part of a single gesture is free hand drawing.

5. Summary

We have developed a system support gestural communication over live video stream for remote collaborative physical tasks using tablet PCs. Our task differs from other gesture recognition systems in the way that it supports not only human to computer interaction but also human to human communication. Our current system support recognition of 12 predefined gestures, gesture fitting, freehand drawing, and combination of two. In gesture recognition, we have used a variable window to extract curvature changes as local features for representing input gestures. We have presented a novel hierarchical classifier that consists of hidden Markov models and a decision tree. HMMs handle the problems of ambiguous angles of closed gestures statistically. And the constrained HMMs can be extended to gestures with fixed number of edges and angles. While open gestures have more significant higher level features, we propose to extract features with different thresholds. And a decision tree is used select the most distinguishable features. We have demonstrated feasibility of the proposed algorithms through experiments. Recognition results have indicated promising performance of our algorithms. The overall accuracy of 12 gestures is 96.4%. Accuracies for closed gestures and open gestures are 96.9% and 96.1% respectively.

6. Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant Nos. 9980013 and 0208903. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. We'd like to thank Rong Yan, Yan Liu, and Tal Blum for many useful discussions on classifies.

7. References

[1] Bekker, M. M., Olson, J. S., & Olson, G. M. (1995). Analysis of gestures in face-to-face design teams provides guidance for how to use groupware in design. Proceedings of DIS 95. NY: ACM Press.

[2] Flor, N. V. (1998). Side-by-side collaboration: A case study. International Journal of Human-Computer Studies, 49, 201-222.

[3] Fussell, S. R., Kraut, R. E., & Siegel, J. (2000). Coordination of communication: Effects of shared visual context on collaborative work. Proceedings of CSCW 2000 (pp. 21-30). NY: ACM Press.

[4] Fussell, S., Setlock, L., Parker, E., & Yang, J. (2003). Assessing the value of a cursor pointing device for remote collaboration on physical tasks. Proceedings of CHI '2003. NY: ACM Press.

[5] Kuzuoka, H., & Shoji, H. (1994). Results of observational studies of spatial workspace collaboration. Electronics and Communications in Japan, 77, 58-68.

[6] McNeill, D. (1992). Hand and mind: What gestures reveal about thought. Chicago: University of Chicago Press.

[7] Tang, J. C. (1991). Findings from observational studies of collaborative work. International Journal of Man-Machine Studies, 34, 143-160.

[8] Rubine, D. (1991). Specifying gestures by example. *Computer Graphics, 25*, 329-337.

[9] Jorge, J., & Fonseca, M. (1999). A Simple Approach to Recognise Geometric Shapes Interactively. *Proceedings of GREC'99* (pp. 266-276).

[10] Jin, X., Liu, W., Sun, J., & Sun, Z. (2002). On-line Graphics Recognition. *Proceedings of PG'02,* pp. 256-265.

[11] Hu, J., Rosenthal, A. S., & Brown, M. K. (1997). Combining High-Level Features with Sequential Local Features for On-Line Handwriting Recogintion. ICIAP (2) 1997, 647-654.

[12] Rabiner, L. R., A tutorial on Hidden Markov Models and Selected Applications inSpeech Recognition, Proc. of the IEEE, Vol. 77, No. 2, pp. 257-286, 1989.

[13] Yasuda, H., Takahashi, K., and Matsumoto, T., (2000). A Discrete HMM for Online Handwriting Recognition. International Journal of Pattern Recognition and Artificial Intelligence, Vol. 14, No. 5, pp. 675-689.

[14] Sezgin, M., Stahovich, T., & Davis, R. (2001). Sketch based interfaces: Early processing for sketch understanding. *Proceedings of PUI-2001*. NY: ACM Press.

[15] Gutwin, C., & Penner, R. (2002). Improving interpretation of remote gestures with telepointer traces. *Proceedings of CSCW 2002*. (pp.49-57). NY: ACM Press.

[16] Connell, S. D., & Jain, A. K. (2000). Template-based Online Character Recognition. Pattern Recognition Volume 34, Issue 1, 1 January 2000, Pages 1-14.

[17] Quinlan, J. R. (1993). *C4.5.: Programs for Machine Learning*. San Mateo, CA: Morgan Kaufmann.

[18] Sutherland, I., (1963). Sketchpad: A Man-Machine Graphical Communication System. PhD thesis, Department of Electrical Engineering, MIT, January 1963.