Hidden Markov Model Symbol Recognition for Sketch-Based Interfaces

Derek Anderson, Craig Bailey and Marjorie Skubic

Computational Intelligence Research Laboratory
University of Missouri-Columbia, Columbia, MO 65211
dtaxtd@missouri.edu / baileycw@missouri.edu / skubicm@missouri.edu

Abstract

A central challenge for sketch-based interfaces is robust symbol recognition. Artifacts such as sketching style, pixelized symbol representation and affine transformations are just a few of the problems. Temporal pattern recognition through Hidden Markov Models (HMM) can be used to recognize and distinguish symbols as pixel-driven gestures. The key challenges of such a system are the type and amount of necessary pre-processing, feature extraction, HMM parameter selection and optional post processing. In this paper, we describe a recognition strategy based on HMMs and include recognition results on twelve sketched symbols. In addition, we have successfully applied this methodology to a PDA sketch-based interface to control a team of robots. The symbol recognition component is used to identify sketched formations and issue commands that drive the behavior of the robot team.

Introduction

Our motivation for exploring sketch-based interfaces is to create intuitive interaction for controlling and communicating with one or more robots. We have constructed a prototype system that was interactively demonstrated at the 2004 AAAI robot competition in San Jose, CA. This interface was used to direct robot formations by communicating with a team of small mobile robots (Skubic et al., 2004). In another project, we have developed a sketch-based interface that is used to sketch a route map that is then translated into a set of spatial rules for a single mobile robot (Skubic, Bailey & Chronis, 2003; Chronis & Skubic, 2004). One similar factor between these two projects is the problem of recognizing a set of pre-defined symbols that can be drawn in a variety of different but similar ways and yet are interpreted robustly as control commands to the robot. These differences are due to affine transformations, user sketching styles, and problems associated with the pixelized symbol capture. In this paper, we present ongoing work towards a generalized method to recognize sketched symbols through the application of Hidden Markov Models (HMM).

Strategies to perform symbol recognition include fuzzy logic rule-based systems (Jorge & Fonseca, 1999), global and discriminate geometric features (Rubine, 1991), primitive geometric shape fitting to pixel sequences (Sezgin, Stahovich & Davis, 2001) and even hybrid systems such as decision trees and HMMs (Ou, Chen & Yang, 2003). The work performed by Sezgin et al. (2001) involves the discovery of vertices at the end of linear segments and the recognition of curved regions within a stroke. They work to generate a more abstract and compact description of a stroke through recognition of primitive geometric shapes. Ou et al. (2003) perform closed gesture symbol recognition with HMMs, but use decision trees in classifying open gestures. They extract a measure of curvature in a window of consecutive pixels and use these features as observations to train a HMM. A decision tree, constructed using the C4.5 algorithm, is used to classify open gestures by identifying a few attributes of the sketched symbol.

Of these approaches, the HMM seems to hold the most promise for a generalized and automatic framework for recognizing sketched gestures. However, a major challenge is finding a feature set that captures the essence of the gesture and yet is resilient to variability in scale, orientation, and sketching style. Other challenges include minimizing the number of training samples and computational complexity.

HMMs can be applied to pixel-driven symbol recognition through the following steps: (1) the identification of pre-processing procedures to handle some initial level of symbol variation, (2) identification and extraction of features from the gesture, (3) an appropriate application specific selection of HMM parameters, and (4) the optional application of post-processing methods for false alarm removal or additional symbol verification. In this work, we describe how we proceed with each step and address the topic of false alarm elimination through the process of post feature identification or heuristics. Another contribution of our work includes the discussion and explanation of a PDA (Personal Digital Assistant) sketch-based application for driving a robot team. In addition, we discuss both the strength and limitations of using HMMs for pixel-driven sketched gesture recognition.

Sketched Symbol Recognition

As a symbol is sketched, the sequence of pixel coordinates, which correspond to the two dimensional locations of the pen over time, are recorded. This temporal unfolding of the symbol in terms of pixel coordinates can

be viewed as a form of gesture recognition. The goal of the system is to compute and learn a set of model parameters that encapsulate symbol related discriminate temporal features. These trained models are finite and can be transferred to a PDA to generate the online likelihood that some future observation, a new sequence of pixels, was generated from some particular known model.

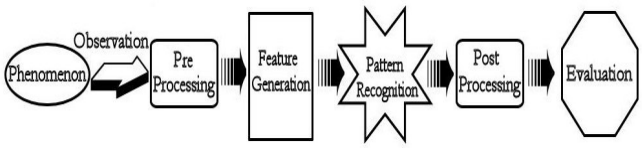


Figure 1. Pattern recognition procedure

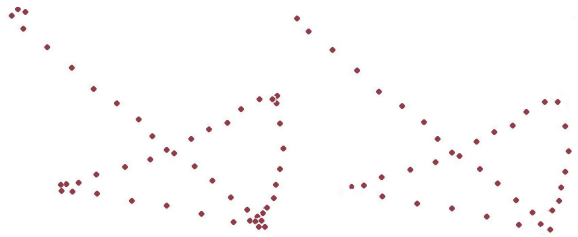
The overall goal is to develop one large classifier whose job is to recognize and classify sketched symbols. One example of a general pattern recognition system is illustrated in Figure 1 and encompasses the (1) observation and capturing of the data, (2) pre-processing of data, (3) extraction and identification of features, (4) application of the pattern recognition algorithms (typically some form of a classifier), (5) optional post processing procedures, and (6) final evaluation and interpretation of the results. The steps behind performing sketch-based symbol recognition include (1) the capture of pixel coordinates, (2) pre-processing of pixel data, (3) the extraction of features from the temporal pixel observations, (4) the application of HMMs to classify the symbol, (5) the application of post feature identification or heuristics to reduce false alarms, and (6) the final classification of a symbol into one of a known finite set.

Pre-Processing

It is a common approach to rely on pre-processing of the data to clean up and minimize erroneous behavior in the symbols. This includes linear and non-linear methods to resample and interpolate a symbol. Factors such as the amount of symbol variation, effects of pixelization, pixel sampling rate and speed which the user draws a symbol result in a possible introduction or elimination of features through pre-processing. In the case of many linear re-sampling methods, important areas such as edges can be greatly distorted. The non-linear methods are attractive but come at the expense of higher computational time.

We do not perform any elaborate pre-processing procedures, but rather attempt to later compensate for these factors within the model and its parameter selection. In order to reduce the pre-processing time, we only perform minimal pre-processing. The first step involves the removal of consecutive identical points. The next step is a method of minimum distance pixel sampling (to down-sample), which can be done quickly as the points are being captured, illustrated in Figure 2.

Figure 2 illustrates one additional artifact that primarily surrounds sketching style. As a user sketches symbols on a PDA screen, he can generate 'hooks' at the beginning or end of a symbol. These hooks depend on factors such as the symbol sampling rate and size of the sketched symbol. One can attempt to identify these hooks for removal in a pre-processing stage. The majority of our hooks were removed simply through the sequential removal of pixels within a fixed distance. The remainder of hooks were recognized and removed, in a similar fashion to Ou et al. (2003), by identifying large points of angular change at the beginning of a sketch.



(a) **(b)**

Figure 2. Fixed distance pre-processing. (a) Sketched symbol example before fixed point removal; (b) Same symbol after fixed point removal.

We strived for the fewest number of pre-processing procedures as possible. PDA devices typically have a low amount of system resources available and a recognizer is only one component of a larger application. Symbols sketched on a PDA are typically composed of a small number of observations or pixels. The fewer the number of pixels the harder it can become to recover the geometric characteristics and identifying features for a symbol. Any pre-processing procedure that modifies these already difficult to recognize symbols in a negative way can be devastating. Some of the pre-processing procedures that we considered computationally acceptable resulted in the unacceptable distortion of symbols in important feature regions or required too many points to have been generated for small symbols. An example of this is in the case of a fast sketched or small rectangle. Linear filtering or interpolation techniques can distort the important edge features and make their recognition extremely difficult

Feature Extraction

Two similar feature extraction methods were investigated in order to study their computational attractiveness, robustness and classification power. The first feature extraction procedure we analyzed was a local method of computing an angle from three consecutive pixels. Two vectors are computed, one from the past to present pixel and the other from the present to next pixel. The feature reflects the angle between these two vectors. This method is sensitive to the style, speed, and other factors that make discrimination and classification very difficult. The other side effect of this procedure was the occasional generation of very long observation sequences.

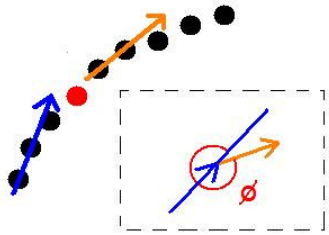


Figure 3. Seven-step, sliding window feature extraction procedure. The red pixel represents the observation step for which we are computing the present feature. The blue arrow represents the average back vector and gold arrow represents the average forward vector. The feature computed is the angle between the back and forward vectors.

The remainder of this paper reports symbol recognition results with respect to our second feature extraction procedure. Figure 3 illustrates this seven step feature extraction procedure. If there are N observations in the symbol, then the algorithm will compute (N-6) features. For every observation after the 3rd step and before the (N-3) step, the angle between a forward and a back vector is computed, where the forward and back vectors are computed averages as shown in Figure 3. The averaging effectively smoothes the curve and minimizes sketching differences due to distortion, varying sketching styles, and pixelization. The pixel that the feature is being computed for is not actually used in determining the angle, which has helped to maintain important information such as sharp turns.

Many symbols can be drawn in a variety of gesture specific sequences (e.g., clockwise vs. counterclockwise directions). Thus, each symbol may require multiple HMM models in order to capture the different gesture specific ways in which the symbol can be sketched. In some settings, a clockwise or counterclockwise sketched symbol can be represented as one HMM model by considering the absolute value of the angle and restricting the feature to be within the range of –180 to 180 degrees. However, in general, it may be useful to know how the user sketched the symbol. We are presently using the multiple model idea, i.e. not using the absolute value of the angle, in order to determine a clockwise or counterclockwise gesture.

This local feature extraction technique has the effect of introducing one additional problem that can be addressed through an optional pre-processing step or feature extraction procedure. If the feature extraction procedure is calculated in the standard way referenced above, then the angular features for a *large* ellipse can have the side effect of being mapped into just one symbol (the straight line). This local feature extraction method has the unwanted side effect of missing subtle geometric changes. If necessary, the components of the feature extraction procedure can be made a function of the symbol size and controlled. This means that features can be computed not for every pixel, but every D^{th} pixel. The number of past and next pixels that are used in calculation of the forward and back vectors can also be determined through the symbol size. We are presently not modifying the internals of the feature extraction procedure, but rather using heuristics to choose when to down sample pixels based on the size of the symbol.

One large predicament is how to develop a reliable and uniform pattern recognizer that works on multiple platforms with different processing speeds. Our method of fixed pixel distance removal helps to overcome some of the problems associated with denseness, but is not as robust for large and dense symbols. This is because the dropping criterion is determined through a heuristically selected threshold value. In these cases, a more formal method of re-sampling might be necessary. We are presently relying on an empirically selected heuristic value for fixed distance point removal.

Temporal Pattern Recognition

Hidden Markov Models (HMM) are an attractive approach to pixel-driven symbol recognition. HMMs are the classical workhorse for recognizing temporal patterns, such as natural speech recognition (Rabiner, 1989). An HMM assumes that the process being learned is composed of a finite sequence of discrete and stochastic hidden states. The states are assumed to be hidden but observable through features that the hidden states emit. Figure 4 is an illustration of a continuous observation HMM (COHMM). A COHMM is represented through a collection of hidden states (W_i), associated hidden state transition probabilities ($a_{(i,j)}$) and hidden state probability density functions (pdf's). For a formal and complete definition of HMMs refer to Rabiner (1989) or Bilmes (Bilmes, 1997).

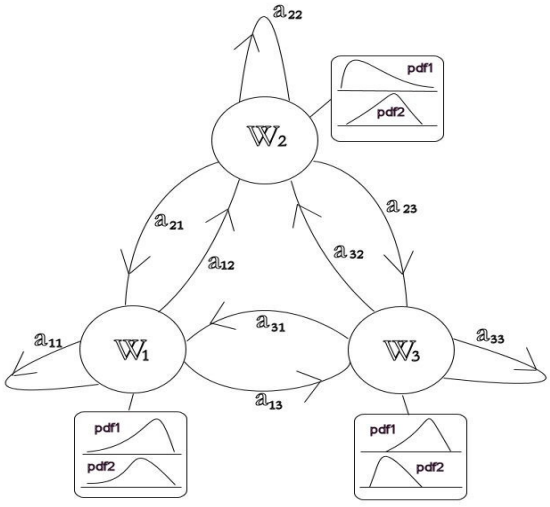


Figure 4. Hidden Markov Model illustration

The three classical problems associated with HMMs include:

1. *Classification Problem*: Given an observation sequence, which model is the most likely to have emitted the particular sequence?
2. *Most Likely State Sequence Problem*: What is the most likely sequence of hidden states which are believed to have emitted a given observation sequence?
3. *Training or Learning Problem*: Given one or more observation sequences, what are the model parameters associated which maximizing the likelihood of the observations?

HMM problem 3 takes the greatest computational time and can be performed offline for the particular case of sketch-based symbol recognition. Our HMM models are trained offline in Matlab. Even though the computational time of problem 3 is noted, it is not a factor because it is not part of the final PDA classifier and only has to be computed once offline. Step 1 is the only algorithm that needs to be implemented and transferred onto a PDA to predict and classify future observations (symbols).

We have applied the traditional Baum-Welch algorithm to iteratively re-estimate the parameters for a discrete observation HMM (DOHMM). Model verification can be done using the conventional forward and backward procedure with appropriate scaling (Rabiner, 1989). A discrete model has limitations and can induce problems into the learning process. Discretization of the features always results in some form of information loss, but it appears to be an acceptable level in this application. We discuss methods here to attempt to overcome and compensate for these problems by a careful selection of HMM parameters.

The angular features that we generate are mapped into 20 different discrete symbols. Note that the first symbol is reflective of the concept 'about zero' (0 +/- ((360°/N)/2), where N is the number of discrete symbols), as depicted in Figure 5. This is in contrast to the method of considering the first symbol to range between 0° to (360°/N). If the symbols are computed in this fashion, then straight gestures with a slight angle will be mapped into two different symbols, i.e. 1 and N.

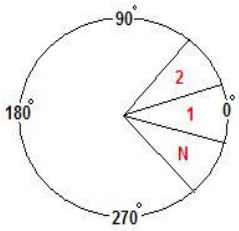


Figure 5. Discrete mapping of the angular features that we compute from the domain [0°, 360°] into one of (1 … N) discrete symbols. The numbers (3... N-1) have pictorially been left out.

In the ideal world of little symbol variation and perfect features there may be only two distributions (i.e. two hidden states) for an arrow. One distribution could represent the straight line feature, while the other could correspond to the sharp edges. We attempted to use a two state model initially, but the combination of sketching style, pixelized symbol representation and our local feature extraction procedure did not yield adequate classification rates. However, we found that we could learn a model to represent this symbol if we over specified the number of HMM hidden states. Figure 6 demonstrates our trained HMM for the arrow. We over specified the model by an additional two states. State 1 appears to learn the straight line feature, while state 3 appears to capture the sharp edge feature. States 2 and 4 appear to learn features that correspond to subtle changes in a straight line direction, which might relate to slightly curved regions.

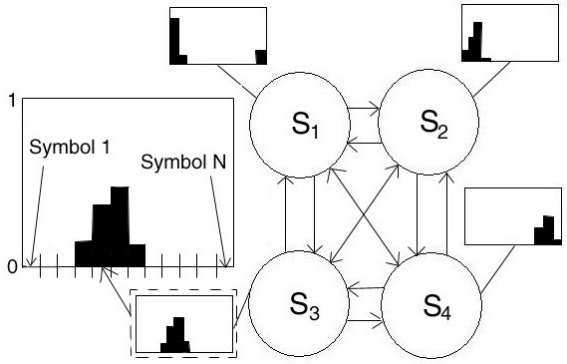


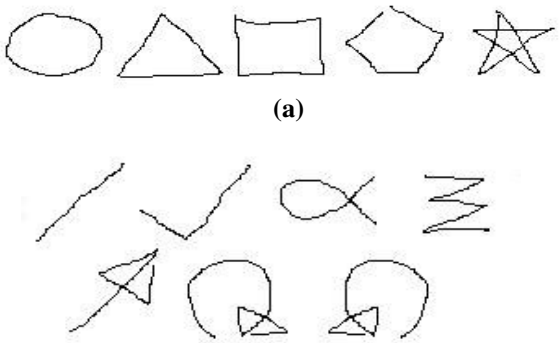
Figure 6. Trained HMM for the arrow gesture. Rectangular regions represent the state pdf's of the feature computed as in Figure 3.

We attempt to compensate and deal with some of the sketched symbol variation through an over specification in the number of HMM hidden states, selection of enough discrete observation symbols and an adequate number of training data. The over specification in the number of HMM hidden states has proven to handle slight inaccuracies that can arise in the local feature extraction procedure and also help to capture reoccurring symbol variation that appears to be natural for some symbols, such as not straight but slightly curved.

This idea of over specifying the number of hidden states is somewhat analogous to a mixture density HMM, which works to learn multiple pdf's per hidden state. Our initial hypothesis was that an over specification of the number of hidden states might help to learn difficult regions that arise from sketching style or compensate for the relatively simple feature extraction procedure. An appropriate selection of the number of discrete observation symbols helps in learning the pdf's associated with each hidden state. The number and type of training data is necessary in order to accurately capture and learn these attributes of interest. This different approach represents an

attempt to learn and account for some variation that arises when considering a large and robust set of training data from a variety of users through a local feature extraction procedure.

The primary problem in this system so far has not been misclassification but the false alarm rate. In nearly all misclassification cases the system failed to recognize the symbol as the most likely, but returned it as a second most likely symbol. After a symbol is classified, the system can compute a few global features or use heuristics to reject symbols that should not be recognized. One such post-processing feature in the case of a rectangle could be the number of distinguishable turn points (corners). In well-sketched rectangles these may be identified as transitions between HMM states. One could use the Viterbi algorithm to find the most likely sequence of HMM states and detect points at which the model transitions between states. It is also possible to devise and rely on heuristics for scenarios that do not fit such typical and expected behavior. This can be illustrated in the case of a rectangle that has no sharp edges but rather regions of slight and gradual change. It is easier to check for the existence of these attributes after the system believes it knows the symbol. At the moment, we are only using the open and closed global features in the results reported below.



(a)

(b)

Figure 7. (a) Closed symbols: Ellipse, Triangle, Rectangle, Pentagon and Star; (b) Open symbols: Line, Check, Cross, Delete, Arrow and Round Arrows.

Open vs. Closed Symbols

We identify and distinguish between open and closed gestures based on the relative location of the starting and ending points. Partially sketching a symbol can generate a likelihood value that results in a model match when there should not be one (e.g., a "c" vs. a circle). We address this problem by using the closed or open gesture criterion for a global feature. Figure 7(a) illustrates the 5 closed symbols and Figure 7(b) illustrates the 7 open symbols that we included in training and testing.

Results

The testing and training reflects the set of symbols that are shown in Figure 7, all collected on a PDA. An initial goal was to compare our classification results to those that were generated in the similar work by Ou et al. (2003). As a result, we selected the same set of symbols. Our results and methods differ in that they performed linear interpolation on the original sequence of pixels followed by a resampling of the symbol, used a more complex feature curvature extraction algorithm, worked on a tablet pc and used a decision tree to classify open gestures. These two works can also be expected to vary in the degree to which symbol variation was included in the testing and training data.

Generating comparable classification results is tricky. Simply reporting a few classification numbers is not sufficient. Variables within a testing environment include variation due to sketching style in the symbols, subjective factors such as when a symbol should no longer be classified as a symbol (fuzzy thinking), sampling rate of pixels, scaling of the symbols, and what combinations of factors are accounted for in terms of testing and training data.

Table 1 reflects the classification results from the entire set of symbols in Figure 7. The samples were collected from three different users. One user was very familiar with a PDA, the other had used a PDA before and the last user had never used a PDA. These samples varied in the size, relative starting point for the gesture, orientation and style. Ten training samples were collected from each user and the classifier was tested using twenty separate examples from each user. Therefore, the classification results are generated from 60 testing samples per symbol. Each classification result in Table 1 reflects the average of all models that are used to represent that one symbol. Hence, the ellipse classification results are the combination of both the clockwise and counterclockwise models.

Table 1. Classification Results

Symbols	Ou (2003)	Our Results
Ellipse	99.1%	92%
Triangle	100%	95%
Rectangle	89.2%	93%
Pentagon	96.4%	93%
Star	100%	97%
Line	100%	98%
Check	97.3%	93%
Cross	98.2%	92%
Delete	92.8%	89%
Arrow	94.6%	98%
Round Arrow	94.6% and 95.5%	96%

The classification results provided by Ou et al. (2003) are a little higher on some symbols, while slightly lower than ours on others. We included two sets of classification

results in Table 1 for the round arrow, to maintain uniformity with the results reported by Ou (2003). They report a classification percentage of 94.6% on the arrow, while we report 98%. They report higher and perfect classification percentages on the line, star and triangle. These two sets of results will differ due to training and testing symbol variation, pre-processing procedures, feature extraction and other factors that arise due to HMM usage and implementation.

We also collected samples of symbols that we would not expect any of the models to recognize. Figure 8(a) demonstrates an example of a correctly not-recognized symbol and Figure 8(b) shows a symbol that was marked as identified but is a false alarm. The false alarm is an example of what can happen with open gestures when they are not performed in their entirety. Some form of post-processing or heuristics, which we have discussed, can then be used to disqualify it if desired.

(a) **(b)**

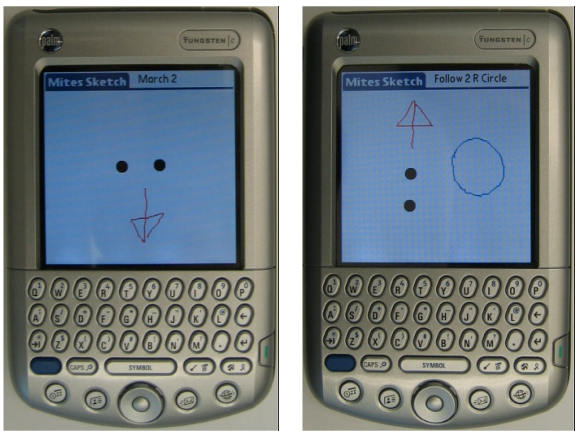
Figure 8. (a) Correctly not-recognized closed symbol; (b) Open symbol recognized as a false alarm.

Sketch Interface for Multi-Robot Formations

The sketch recognition strategy discussed above has been applied to a PDA interface for controlling a team of small robots. In this section, we briefly describe our sketch interface for driving multi-robot formations. See also (Skubic et al., 2004) for more detail.

The symbol recognition component of the robot sketch interface is responsible for identifying and distinguishing a fixed set of control symbols which are used to create multi-robot formations (Figure 9). The four symbols used in the system were the ellipse, rectangle, line and arrow. A user can also sketch "blobs", which represent robots. When a robot blob is recognized, the robot icon is displayed on the PDA screen as a solid black circle.

Based on previous work, we have also supported a set of editing capabilities to move and delete symbols (sketching an "X"). A single HMM was not used to recognize the delete command because it is the combination of two separate strokes. For the moment, our techniques can only handle the recognition of single stroke symbols. We perform the check for a delete command through the search for two consecutive lines being sketched, each one independently recognized by a HMM, and then perform a check for an intersection point. The blob, also based on previous work, is sketched as a "squiggle" (Skubic, Bailey, & Chronis, 2003).



(a) **(b)**

Figure 9. Robot formation directives sketched on a PDA. (a) Two robots sketched in a march formation; (b) Two robots in a follow-the-leader formation, with the leader moving in a circle.

The user can move and align the robot icons into configurations such as "follow the leader" and "march side by side" formations. While there are robot icons on the screen, a user can sketch an arrow; the orientation of the arrow with respect to the sketched robot icons is used to identify the robot formation. Follow the leader is identified when the robot icons are aligned in a linear formation parallel to the arrow. The march side by side formation is identified when the robot formation is perpendicular to the sketched arrow direction. After the command to march or follow the leader has been given, there is an option to sketch additional symbols such as ellipses, rectangles, or lines to change the geometric path of the robots. For example, a sketched clockwise ellipse after a follow the leader formation will result in the robot leader moving in a clockwise circular path with the remaining robot team following behind (Figure 9(b)).

The symbol recognition system has been implemented in PalmOS using C++. The sketch recognition application has been tested on several PDA platforms, including the Handspring Visor, the Palm m505 and the Palm Tungsten platforms. The recognition time of symbols is barely noticeable at all. The recognition time depends on the observation length, but on average is less than a second on the Visor and appears to be automatic in almost all cases on the Tungsten. When demonstrated at the AAAI Conference, the recognition system successfully identified symbols sketched by solicited participants. The team earned a technical innovation award at the AAAI Robotics Competition for the sketch-based interface.

Future Work

The existing system has several limitations that we intend to address through future work. The high false

alarm rate is currently being reduced through a post-processing phase using heuristics. Although this is an acceptable solution for the prototype system, it does not scale well for adding more types of symbols. This limitation will be addressed by investigating alternative features, or combinations of features. There is always a tradeoff in classification and recognition ability when considering whether to compute a local or global feature or both. Also, we propose to extend the system to support multi-stroke symbols.

Currently, there is no formal representation of hierarchical structures (i.e., configurations of recognized gestures). Configurations of gestures are handled in an ad hoc manner. We are going to explore hierarchical strategies and data structures for representing and reasoning about configurations of symbols distributed over space or time. To address the limited computational resources and screen size of the PDA platform, we also are investigating a port of the sketch-based interface from a PDA to a Tablet PC.

Concluding Remarks

The application of Hidden Markov Models to sketch-based symbol recognition as pixel-driven gestures is a very attractive method. Acceptable classification rates can be achieved through a moderate amount of pre-processing, identification of adequate but discriminatory features, appropriate selection or over specification of HMM parameters and additional post-processing and post-verification methods. False alarms are an undesirable and unavoidable aspect of this approach, but can be addressed through post-processing feature identification and heuristics. HMMs can act as an extremely important tool in the approach and design of sketch-based recognition systems, but should be considered as only one source of information. The outcomes of this classifier could be fused or merged with another type of algorithm, heuristics, or additional pattern recognition techniques to handle the natural ambiguity in sketch understanding.

Acknowledgements

This work is partially funded by the Naval Research Lab under grant N00173-04-1-G005.

References

Bilmes, J., 1997. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models, *Technical Report, University of Berkeley,* ICSI-TR-97-021.

Chronis, G., and Skubic, M., 2004. Robot Navigation Using Qualitative Landmark States from Sketched Route Maps. In *Proc. IEEE. Conf. Robotics and Automation*, New Orleans, LA, May: 1530-1535.

Jorge, J., and Fonseca, M., 1999. A Simple Approach to Recognize Geometric Shapes Interactively. *Proc. Third Intl. Workshop on Graphics Recognition*, Jaipur, India, Sept, 251-258.

Ou, J., Chen, X. and Yang, J., 2003. Gesture Recognition for Remote Collaborative Physical Tasks Using Tablet PCs, *ICCV Workshop on Multimedia Technologies in E-Learning and Collaboration*, Nice, France, Oct.

Rabiner, L., 1989. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. In *Proc. IEEE, 77(2):* 257-286.

Rubine, D., 1991. Specifying gestures by example. *Computer Graphics*, 25: 329-337.

Sezgin, M., Stahovich, T., and Davis, R., 2001. Sketch Based Interfaces: Early Processing for Sketch Understanding. *Proc. of PUI-2001,* Orlando, Florida, Nov, 1-8.

Skubic, M., Bailey, C., and Chronis, G., 2003. A Sketch Interface for Mobile Robots, In *Proc. IEEE Conf. SMC*, Washington, D.C., Oct, 918-924.

Skubic, M., Anderson, D., Khalilia, M., and Kavirayani, S., 2004, A Sketch-Based Interface for Multi-Robot Formations, *AAAI Mobile Robot Competition 2004: Papers from the AAAI Workshops*, San Jose, CA, July.

Yasuda, H., Takahashi, K., and Matsumoto, T., 2000. A Discrete HMM for Online Handwriting Recognition. *International Journal of Pattern Recognition and Artificial Intelligence,* 14(5): 675-689.