# State of the Research – Recognition

Jessi Peck and Miranda Parker

Summer 2011

## **Goals**

Our primary goal this summer was to enhance the accuracy of the recognition algorithms. Last year’s project sacrificed accuracy for speed, and we wanted to maintain both properties in the code.

We also planned to implement a framework to allow the system to adapt to user-specific drawing styles. This was critical to improving gate recognition owing to the disparity of user drawing styles. We hoped that this would produce a dramatic improvement in recognition over time as any particular user worked with the system.

After several years’ worth of research students working on this project, the code has become cluttered. We wanted to continue last year’s work developing streamlined system architecture that is easy for future researchers to follow.

## **State of the Code and Activities**

### **Stroke Classification and Grouping**

We improved stroke classification and grouping by replacing the hard-coded decision trees with Weka-based trees. We implemented Weka-based J48 decision trees with AdaBoost to improve the accuracy over the regular decision tree classifier. In the past, the system transitioned from Weka algorithms to simpler decision trees because loading an external program was overly time consuming. We used IKVM to translate the Java-based Weka into a C#-readable .dll file, eliminating the need to run an external program and thereby maintaining a reasonable speed.

The stroke classifier labels each stroke in your drawing as either “Gate”, “Label” or “Wire”. To do this, the stroke classifier combines the 27 features listed at <https://www.cs.hmc.edu/twiki/bin/view/Sketchers/GroupingStrokes> and described in Eric Peterson’s paper “Grouping Strokes into Shapes in Hand-Drawn Diagrams”. It is pre-trained and saved as .arff and .model files, which are loaded when the program starts.

The stroke grouper considers each similarly labeled pair of strokes and decides if they should be joined into a shape. To make this decision, the stroke grouper uses 12 features described in Eric Peterson’s paper and on the Sketchers wiki (see above). Gates and labels are each grouped by their own decision tree. Wires are never grouped, but instead dealt with by the connector. As with the stroke classifier, the stroke groupers are pre-trained and loaded from .arff and .model files when the program starts.

### **Shape Recognition**

The system implements three separate algorithms to recognize shapes. The wire recognizer simply gives each wire the type “wire.” The text recognizer was mostly out of our hands, as it uses Microsoft Ink software to recognize handwriting. The gate recognizer was the primary focus of our research. When we first got the system, it used a bloated combination recognizer to assign gates to their types. The combination recognizer utilized four different sub-recognizers, and then combined their results using a Bayes net. The primary issue with the combination recognizer was that, with all of the different recognizers, it was very difficult to track down the recognition problems. When we began working with the system, the recognition accuracy was very poor and we hoped to dramatically improve it throughout the summer.

*We now are able to expose the recognition model to the user, thanks to our collaboration with the UI improvements team. After recognizing a sketch, the user can enter a mode that allows them to hover over a gate and see the template that was used to recognize that shape. This helps users understand the system and the recognition process better. If users are frustrated, they can take this knowledge to help the system help them!*

We transitioned the recognition of the system from a bloated combination recognizer to an image recognizer based on the model described in “An image-based, trainable symbol recognizer for hand-drawn sketches” by Kara and Stahovich. The new image recognizer allows the user (and future programmers) to understand the recognition model, whereas the ComboRecognizer required extensive prior knowledge to understand. Furthermore, the image-based recognizer is more accurate than the combination recognizer. This disparity would likely be lessened if the combination recognizer was properly debugged and trained. We believe that the image recognizer is still a better option because it offers comparable recognition accuracy and is a much simpler model.

### **Adaptive Image Recognizer**

We also added a framework to the image recognizer to allow it to learn from user error correction. When the user re-labels a gate, the recognizer adapts by adding the drawing to its database of templates. However, the program runs increasingly slowly as you add templates. Therefore, we implemented a “phase out” process that removes gates when we accumulate too many gates of that type. To ensure that we do not remove the templates that are most helpful to recognition, the image recognizer keeps track of a “usefulness” value for each template that is increased every time the template is used to recognize a shape and decreased whenever a user indicates that the template led to incorrect recognition.

## **Results**

### **Stroke Classification and Grouping**

To assess the accuracy of the stroke classifier and grouper, we ran tests on the labeled sketches from the UCR/HMC user study. There were 480 sketches, collected as part of the Circuit User Study which is described in “The Effect of Task on Classification Accuracy: Using Gesture Recognition Techniques in Free-Sketch Recognition” by Martin Field, Sam Gordon, Eric Peterson, Raquel Robinson, Thomas Stahovich, and Christine Alvarado.

We tested our Weka-based classifier and grouper using user holdout cross-validation. The classifier and grouper were trained on all the users present, except for one, and then tested on the excluded user. This procedure was repeated for all users, and then the results were compiled before the accuracy was calculated.

The original stroke classifier and grouper were poorly documented, so we were unable to determine what data it had been trained on or find a way to retrain it. To test it, we simply used the existing decision trees to classify and group the data described above. Obviously, this leads to a small amount of discrepancy in our testing results, but we were still able to show that the Weka implementation produces consistently better results.

### **Shape Recognition**

To compare our image-based recognizer to the old combination recognizer we ran tests on the data described above from the Circuit User Study. Even without adaptive capabilities, our image recognizer is a dramatic improvement and thus the base of what we currently use in the code.

To obtain these results, we tested on the data described above under Stroke Classification and Grouping. The combination recognizer and the image recognizer were trained on half of the set, then tested on the whole set. The “perfect” image recognizer was trained on five ideal drawings for each gate, and then similarly tested on the entire Circuit User Study data set.

### **Adaptive Image Recognizer**

When a user corrects an error, our adaptive image recognizer adds a template to its database. However, each additional template slightly slows down the system, so we wanted to limit the maximum number of templates stored for each gate. We attempted to find the optimal balance between accuracy and speed, but we did not have sufficient data to conclude anything significant about the best number of templates. We tested on all of the available sketches from a single user, incrementally changing the maximum number of templates per gate from five to fifteen.

Our graph below documents our results from testing the adaptive image recognizer on the data collected from this year’s user study. We developed a testing procedure that simulates users working with the program and correcting errors over time. See the details of our testing at: <https://www.cs.hmc.edu/twiki/bin/view/Sketchers/AdaptiveTestingProcedure>

Although we have not done any extensive analysis of our data, it seems clear that as we add templates, the recognition accuracy quickly increases.

## **Future Work**

* The recognition process is currently fully functional. Of course, recognition accuracy can always be improved.
* Continue research to make sure we’re using the most up-to-date recognition algorithms.
* Speed up the processes of training and loading the stroke classifier and grouper. Currently, this takes up a considerable amount of time.
* Recognition could be extended to other domains pretty easily, with alterations in the following areas regarding recognition:
  + ShapeType
  + Templates
* Find ways to improve the recognition of the orientation of templates. This is currently the most frequently wrong part of the recognition algorithm.
* Possibly allow removal of individual templates. This would give the user the option to remove gates that they drew poorly, so as to not impair the recognition accuracy. However, we should do more research into whether or not we want to allow the user this much control.
* Find a better way to normalize the values that we obtain from the sub-recognizers. Currently we are doing this linearly, but a neural net would be an interesting alternative.