GRAPHIC SYMBOL RECOGNITION TOOLKIT (HHreco) TUTORIAL

by

Heloise Hse and A. Richard Newton

Memorandum No. UCB/ERL M03/50

16 January 2003

GRAPHIC SYMBOL RECOGNITION TOOLKIT (HHreco) TUTORIAL

by

Heloise Hse and A. Richard Newton

Memorandum No. UCB/ERL M03/50

16 January 2003

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Graphic Symbol Recognition Toolkit (HHreco) Tutorial

Heloise Hse and A. Richard Newton
Department of Electrical Engineering and Computer Sciences
University of California at Berkeley
Berkeley, CA 94720, U.S.A.
{hwawen, newton}@eecs.berkeley.edu

1. Introduction

HHreco is a software library providing multi-stroke symbol recognition utilities. In addition to interfaces and core classes for developing symbol recognizers, the library contains a solid implementation of an adaptive multi-stroke recognition system. The recognizer is independent of stroke-order, -number, and -direction, as well as invariant to rotation, scaling, and translation of symbols. It is designed to be an off-the-shelf recognizer and can also be further customized to specific applications. The detailed description and the performance evaluation of this recognition system have been documented in another technical report [2].

2. Installation

HHreco can be downloaded from:
http://www.eecs.berkeley.edu/~hwawen/research/hhreco/index.html

HHreco is written entirely in Java and compiles with Java™ 2 SDK, Version 1.4.2 which can be downloaded from http://www.javasoft.com.

2.1. Installing and compiling the source release

1. Download the tar.gz file (hhreco.tar.gz) or the zip file (hhreco.zip), and save it in a directory.

2. Go to that directory and extract the files by typing:

   ```
   tar xvfz hhreco.tar.gz
   ```

   or

   use WinZip for the zip file on Windows

   This will create a directory called **hhreco** with all the recognition utilities in it.

3. Set the classpath to include hhreco, the SVM library it uses, and the Diva package (optional, see Section 6 for more details). For example, if you have downloaded **hhreco** into **d:/** directory, you would do:

   setenv CLASSPATH ".;**d:/**;**d:**/hhreco/lib/libsvm.jar;**d:**/hhreco/lib/diva.jar"

   If you are working in a Unix environment and **hhreco** is in /user direction, you would need to type:

```
setenv CLASSPATH .:/user:/user/hhreco/lib/libsvm.jar:/user/hhreco/lib/diva.jar
```

Note that on Unix, the path separator is : instead of ; and double quotes are not needed.

4. To compile the source code, execute the script, *compile*. Modify the path reference to *bash* on the first line of the script to reflect your own installation location of *bash*.

3. Package documentation

HHreco contains the following items in its directory.

Files:

COPYRIGHT	The copyright notice applies to all of the HHreco software and documentation.
README.txt	A short description of the package.
compile	A script to compile the source code.

Packages:

hhreco.apps	Two applications are included in this package. TrainApp.java is for creating gesture training files and TestApp.java is an interactive application that performs recognition on user sketched data.
hhreco.classification	This package provides data structures such as feature sets, training sets, as well as a classification framework for facilitating pattern classification tasks. It also includes 3 classifier implementations: SVM, Minimum Mean Distance classifier, and Nearest Neighbor. (Figure 1)
hhreco.lib	Jar files that hhreco uses are kept in this packages. We use libsvm.jar for its SVM implementation [1] and diva.jar for its sketch framework and graphical utilities [3].
hhreco.recognition	Interfaces and core classes for multi-stroke symbol recognition. (Figure 2)
hhreco.toolbox	This package contains routines specific to dealing with strokes and recognition. In this version, the package includes stroke preprocessing routines to approximate and interpolate strokes.
hhreco.util	General utility classes that can be used by any of the above packages. hhreco.util.aelfred is a package containing the Aelfred XML parser from Microstar. The copyright distributed with HHreco does not apply to hhreco.util.aelfred.

The designs of the major sub-packages in HHreco have been documented using UML class diagrams [4]. HHreco is designed with ease of use and extensibility in mind. The interfaces in the package are central to the extensibility of the system.

3.1 HHreco classification package

In the classification package, the *Classifier* and *TrainableClassifer* interfaces provide an abstraction between the implementation of the actual classification methods and any classes which use the classifiers. *ClassifierException* is thrown when an error occurs during classification, for example, when an inconsistency has been detected in the feature sets. The *DataRep* class is a data structure for modeling a particular class of data with a Gaussian density. A *FeatureSet* stores the feature values extracted from a symbol. Given a *TrainingSet* containing training feature vectors, a *TrainableClassifier* learns from the set and can be called to perform classification.

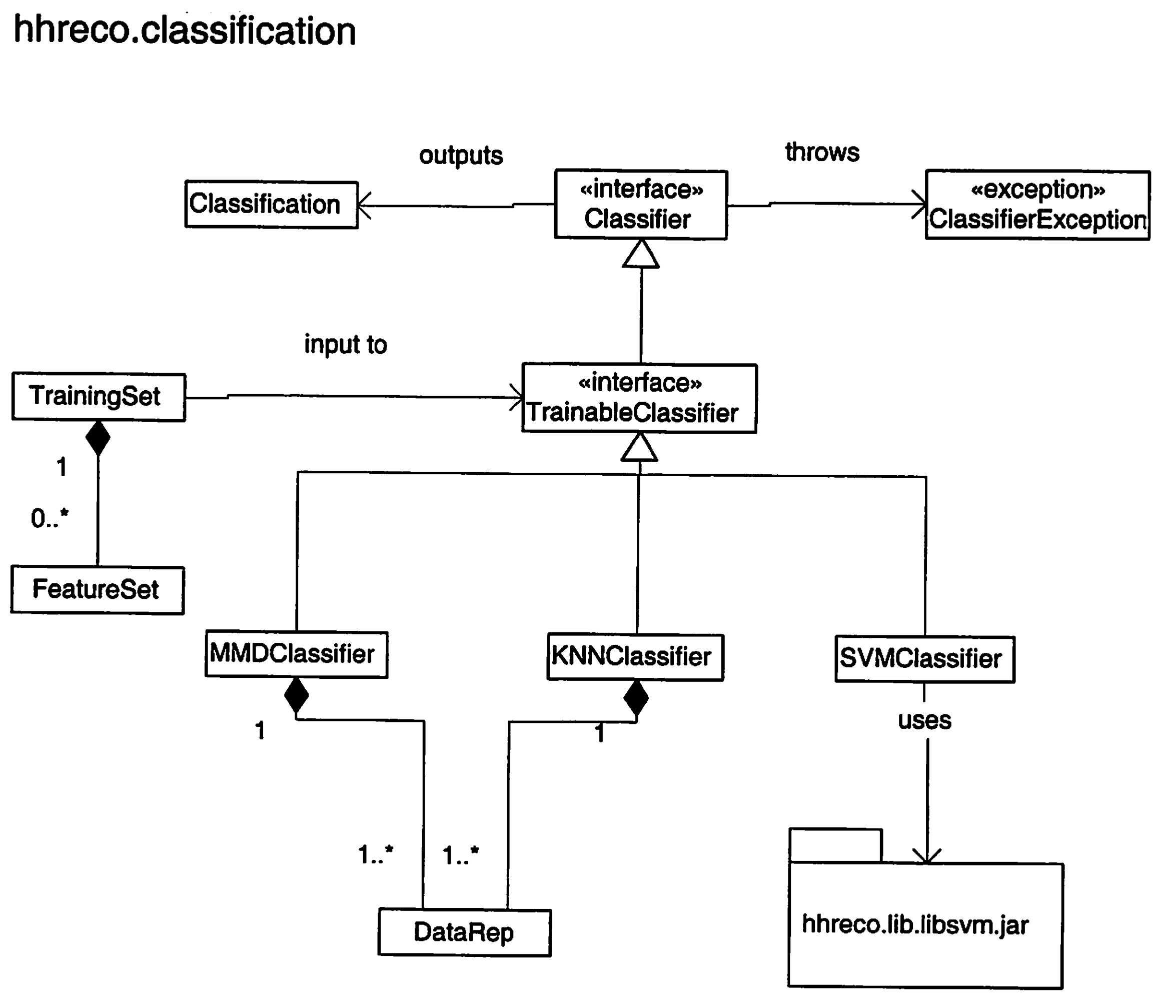


Figure 1. HHreco classification package UML

3.2 HHreco recognition package

The meat of this package is the *HHRecognizer* class. This class has been designed to use a set of *FeatureExtractors* and a *TrainableClassifier*, and the actual implementation of these elements is left to the user of the package. One can easily customize the feature extractors and classifiers to suit his or her application by extending the interfaces. In this particular implementation, we used Zernike moment feature extractor to generate shape descriptors from input strokes and a *SVMClassifier* to perform classification. A *TimedStroke* object records a sequence of sampled points from pen down to pen up, a multi-stroke symbol consists of one or more strokes, and an *MSTrainingModel* stores training symbols categorized by types. The recognition result is communicated through a *RecognitionSet* object consisting of zero or more *Recognitions*. A *Recognition* object records the type of a symbol (e.g. square) and the value associated with that type. Depending on the classification method, the value can be a distance measure, a probability, or a likelihood value. For the *SVMClassifier*, only one *Recognition* will be generated per classification using the max-win scheme. In a distance classifier, the values can be used to generate a ranked list of predictions.

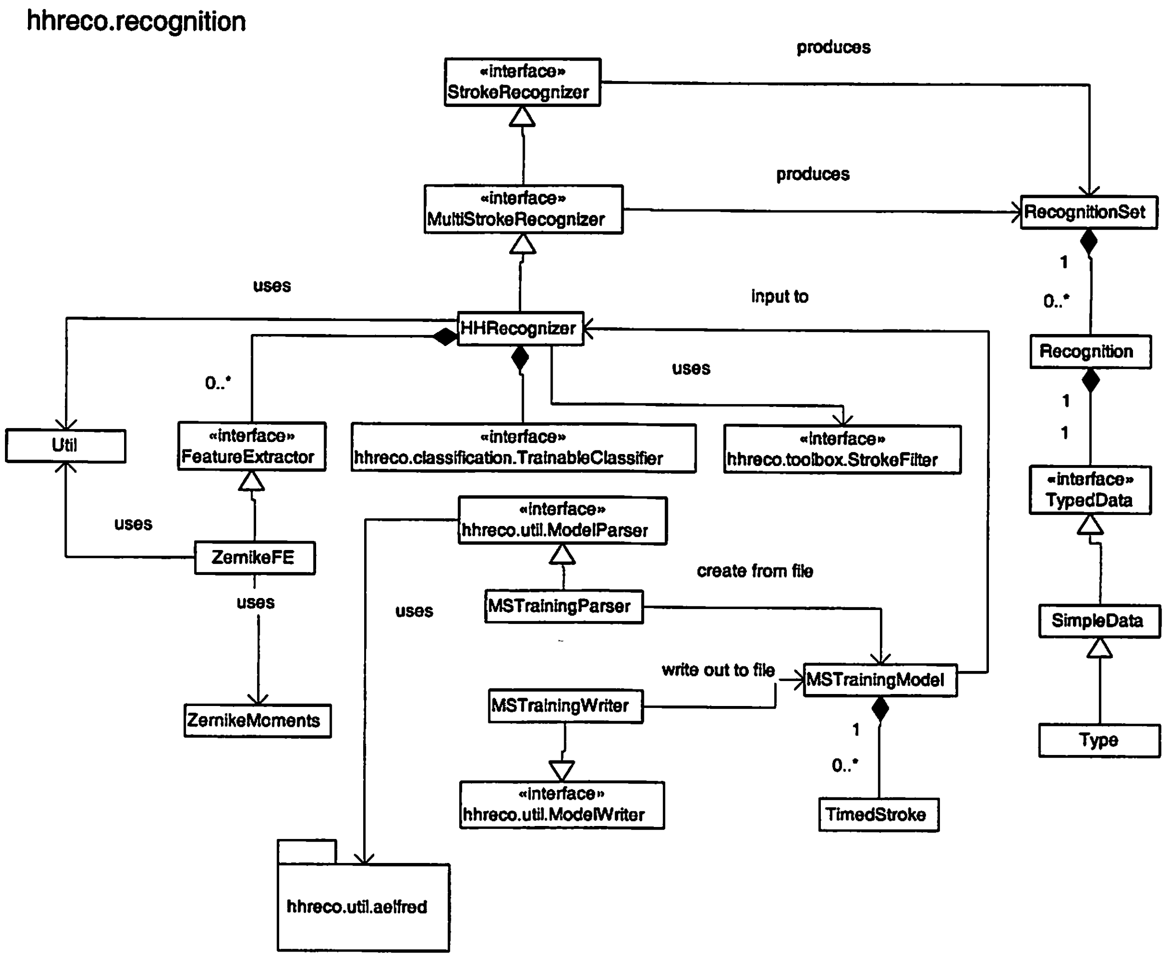


Figure 2. HHreco recognition package UML

Please see the Java API documentation for a more detailed description of each of the classes and their methods.

4. Using HHreco

In this section, we show a simple example on how to set up a recognizer to use in an application.

```
// import the necessary packages
import hhreco.recognition.*;
import hhreco.toolkit.*; //needed if you want to use the stroke filters
import java.io.*; //needed to read in training files

// instantiate a recognition engine using the default feature set (Zernike moments
// to the 8th order) and classifier (SVM).
HHRecognizer reco = new HHRecognizer();

// suppose the file, "joe.sml", contains a set of symbol examples sketched by a user.
// train the recognizer with this training data set.
// read in the file and parse it into a MSTrainingModel

BufferedReader br = new BufferedReader(new FileReader("joe.sml"));
MSTrainingParser parser = new MSTrainingParser();
MSTrainingModel trainModel = (MSTrainingModel)parser.parse(br);

// the strokes in the training set should be preprocessed for better recognition result, so
// let's set up some stroke filters.
ApproximateStrokeFilter approx = new ApproximateStrokeFilter(1.0);
InterpolateStrokeFilter interp = new InterpolateStrokeFilter(10.0);

// for each example in the training set, call HHRecognizer.preprocess and pass in the
// filters.
MSTrainingModel model = new MSTrainingModel();
for(Iterator iter = trainModel.types(); iter.hasNext();){
    String type = (String)iter.next();
    for(Iterator iter2 = trainModel.positiveExamples(type); iter2.hasNext();){
        TimedStroke[] strokes = (TimedStroke[])iter2.next();
        strokes = HHRecognizer.preprocess(strokes, approx, interp, null);
        model.addPositiveExample(type, strokes);
    }
}

// train the recognizer with the given symbol training set.
reco.train(model);
```

```
// at this point, the recognizer has learned the shape vocabularies and can be called
// at any time to perform recognition.

// call the recognizer to recognize a shape comprised of the given set of strokes.
// preprocess the strokes first.
// note that the preprocessing routine will normalize the scaling and translation
// of the shape. If you want to get the transformation done on the shape, pass
// in an AffineTransform object, otherwise just input "null" like below.
inputStrokes = HHRecognizer.preprocess(inputStrokes, approx, interp, null);
reco.sessionCompleted(inputStrokes);
```

Please take a look at hhreco/apps/TestApp.java for a complete, functional program that demonstrates how to set up and invoke a recognizer. The recognizer is adaptive such that examples can be added on-the-fly and the recognizer can be retrained with the incremented training set. This example illustrates how to implement incremental recognition:

```
// let strokes be the symbol you want to retrain the recognizer with
// preprocess it and then add to the training set
strokes = HHRecognizer.preprocess(strokes, approx, interp, null);
reco.addAndRetrain(strokes);
```

You can customize the recognizer by writing your own feature extractors, implementing the hhreco.recognition.FeatureExtractor interface. There are three classifier implementation provided in the hhreco.classication package. You can contribute different classification algorithms by implementing the hhreco.classification.Classifier interface. When setting up the recognizer, simply pass in your preferred feature extractors and classifier.

6. Running applications

In order to run the applications in **hhreco.apps** package, you would need to have Diva. You can download the latest snapshot from diva-22Apr03.zip
http://embedded.eecs.berkeley.edu/diva/release/index.html
And make sure to add its location to the classpath. For convenience, a Diva jar file is distributed with this version of HHreco. You can find it under hhreco/lib directory.

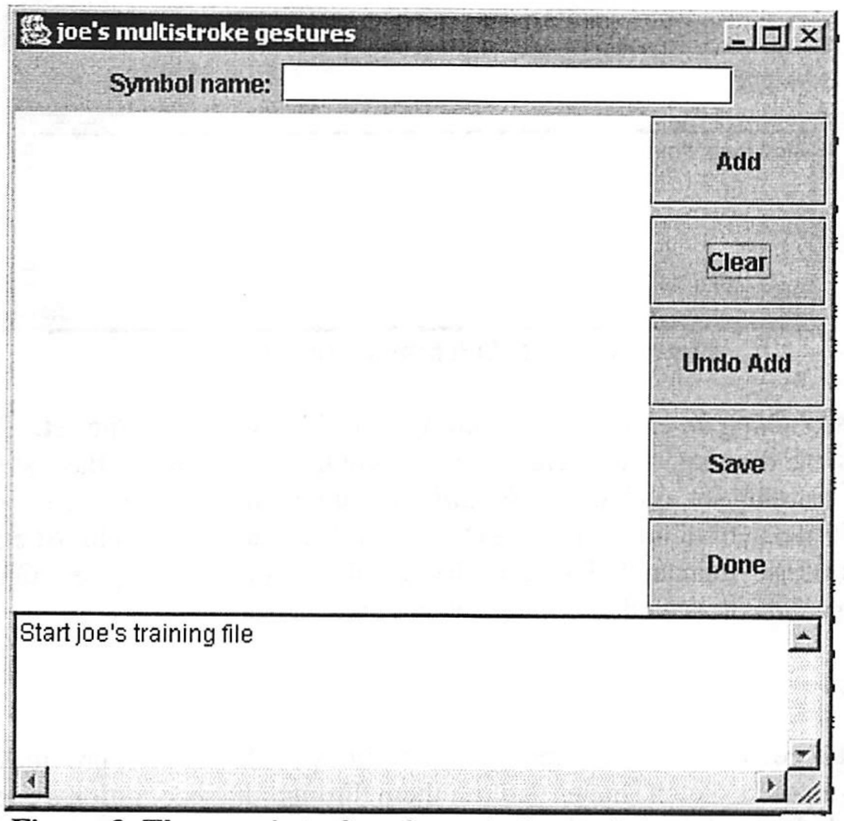
HHreco is designed to be a small, self-contained, off-the-shelf package for doing recognition. The two graphical applications in hhreco.apps, one for creating gesture file and one for demonstrating the recognition system, use Diva for its solid infrastructure for constructing sketch-based user interfaces. If you are interested in using the recognition engine alone as a plug-in to your own application, you can do so without including diva.jar.

6.1. TrainingApp

The TrainingApp program allows users to create customized gestures for their applications. The user can enter about 15 or more examples for each type of gesture that they want to train. The more examples, the better it is for recognition. Start the program by typing in:

```
java hhreco.apps.TrainingApp joe
```

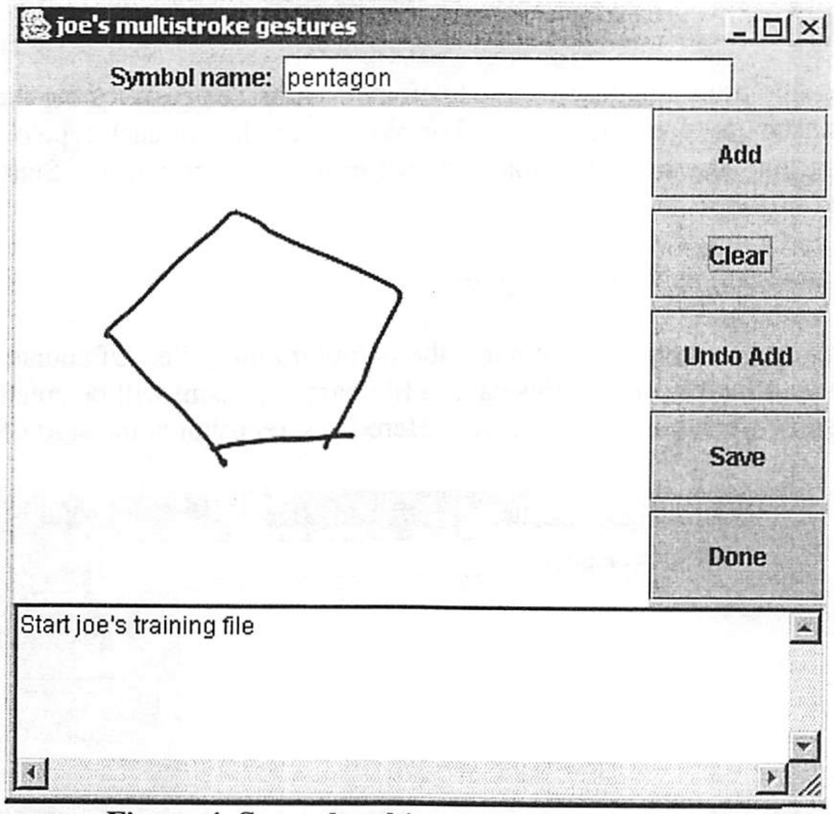
The argument "joe" will be used to name the output training file. Of course, you can substitute in your own name. In this case, a file named joe.sml will be created in the current directory when you click on Save. Here's a screenshot at the start of the program:



Start joe's training file

Figure 3. The user interface for entering training examples.

To start entering training examples, first type in the name of the symbol in the text field, and then start sketching the shape one example at a time.



Start joe's training file

Figure 4. Start sketching training examples.

When done sketching an example, click on *Add* to add it to the training set. The *Clear* button clears the drawing on the screen. If you would like to remove the last example added to the training set, click on *Undo Add*. If you would like to draw a different symbol, enter the symbol name in the text field, and continue to sketch. At any time during the process, you can click on *Save* to save the existing training set. Once you're done, click on *Done* to save the training set and exit.

6.2. TestApp

This application demonstrates the recognition utilities in **HHreco**. It provides an interface for users to sketch shapes and run them through the recognition system. The program can be executed by entering:

```
Java hhreco.apps.TestApp heloise.sml
```

Given a training file (in this case, heloise.sml), the application sets up a recognition system and trains it using the file. It displays the symbol classes that the recognizer has been trained on in the top row. If there were a lot of shape classes, you would need to maximize the window in order to see all the shapes displayed. The empty canvas below is where a user can sketch symbols, one at a time.

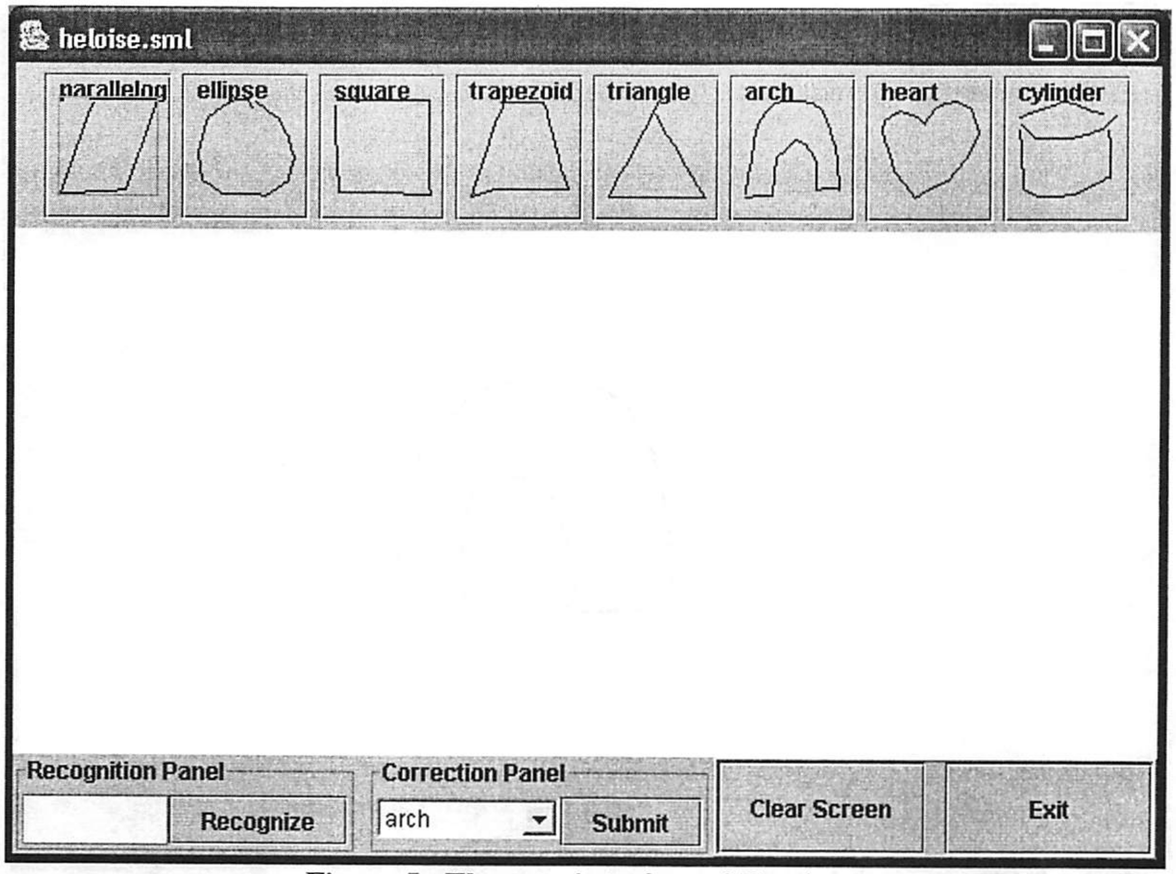


Figure 5. The user interface of TestApp.

After sketching a symbol, you can click on the *Recognize* button to invoke recognition. The recognition result will be shown in the text field under *Recognition Panel.* If the recognition result is incorrect, you can make corrections through the Correction Panel. First, select the correct label for the sketched shape from the choice box. The choices reflect the training set that you have given to the recognizer at the start of the program. Next, click on the *Submit* button. This will retrain the recognition system with the added sketched example and the correct label. You can retest the recognizer by clicking on the *Recognize* button again. To enter another shape, clear the screen first using the *Clear Screen* button, and then proceed. Click on the Exit button to exit the program. The corrections made during a session will be output to a file called corrections.sml. In order for the output file to be written and closed properly, be sure to exit the program by clicking on the X located at the upper right corner of the window or use the Exit button. If you simply do a Ctrl-C to kill the program, the closing tags will not be written to the SML file causing a breach in the file format. The corrected symbols are not appended to the initial training file so as to keep the original file intact. One can merge the two by using the program MergeFiles:

```
java  hhreco.apps.MergeFiles in1.sml in2.sml out.sml
```

If desired, developers can easily program it to do so by adding the corrections to the initial training model and writing out the training model at the end of the session.

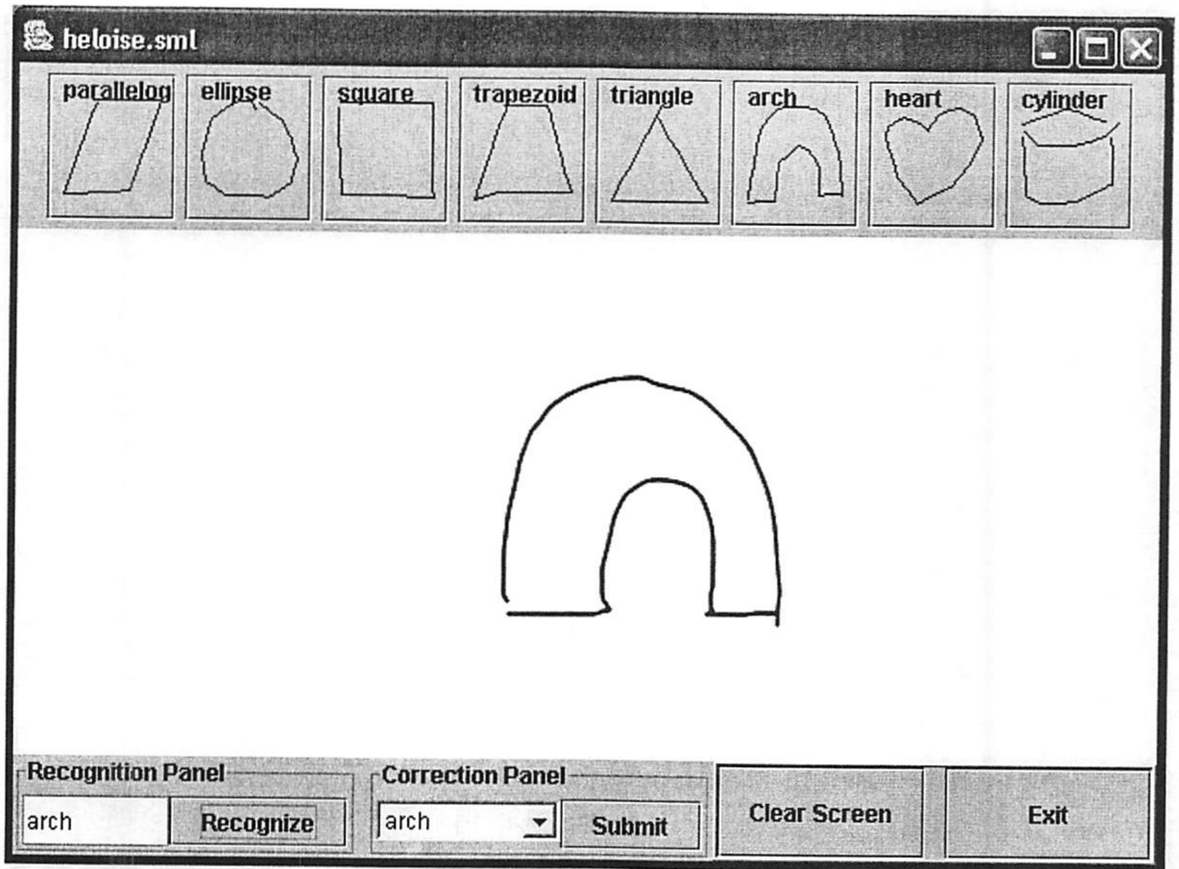


Figure 6. Sketch a symbol on a screen and click on the *Recognize* button to invoke the recognizer.

7. SML (Sketch Markup Language) file format

We are using a simple XML language, SML, for storing sketched data. SML is designed and developed by the original authors of the Diva project [3]. The files generated by TrainingApp are stored in this format. Here is the DTD for SML:

```
<!ELEMENT MSTrainingModel (type+)>
<!ELEMENT type (example+)>
<!ATTLIST type name CDATA #REQUIRED>
<!ELEMENT example (stroke+)>
<!ATTLIST example label CDATA #REQUIRED numStrokes CDATA #REQUIRED>
<!ELEMENT stroke EMPTY>
<!ATTLIST stroke points CDATA #REQUIRED>
```

The example below illustrates the file format. Starting with the heading, a training model is declared with <MSTrainingModel>. In between the start and the end of *MSTrainingModel* tags, one can define multiple symbol classes. A symbol class starts with <type name="..."> followed by examples of the class, and it ends with the </type> tag. Each example definition consists of a label, number of strokes, and the points in each stroke. A label specifies whether the example is a positive or a negative training data. Some classification algorithms would use both positive and negative examples. The stroke points are a sequence of *x*, *y*, and *timestamp* data sampled from the tablet on which the strokes have been captured. At the end of an example definition, close it with the </example> tag.

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE MSTrainingModel PUBLIC "-//UC Berkeley//DTD train 1//EN"
        "http://www.gigascale.org/diva/dtd/multiStrokeTrain.dtd">

<MSTrainingModel>
<type name="pentagon">
<example label="+" numStrokes="1">
        <stroke points="334.0 97.0 1034286896913 331.0 98.0 1034286897124 329.0
100.0 1034286897144 324.0 103.0 1034286897174 318.0 107.0 1034286897184 313.0
111.0 1034286897214 309.0 113.0 1034286897224"/>
</example>
<example label="+" numStrokes="1">
        <stroke points="..."/>
</example>
</type>
<type name="square">
...
</type>
</MSTrainingModel>
```

8. Acknowledgement

The authors wish to thank Michael Shilman, John Reekie, and Steve Neuendorffer for their support and collaboration. This work has been supported in part by Microsoft Research Laboratories. Their support is gratefully acknowledged.

9. Reference

[1] Chang, C. C. and Lin, C. J. http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/.

[2] Hse, H. and Newton, A.R. Sketched Symbol Recognition using Zernike Moments, Technical Memorandum UCB/ERL M03/49. Electronics Research Lab, Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, 2003.

[3] Reekie, J., Shilman, M., Hse, H., and Neuendorffer, S. http://embedded.eecs.berkeley.edu/diva.

[4] Rumbaugh, J., Jacobson, I. and Booch, G. The Unified Modeling Language Reference Manual. Addison-Wesley, 1999.