Online Handwritten Text Recognition

Project Goal
The main goal of our project is to design a text messaging system that is equipped with text recognition. It will utilize the XScale’s touch screen capability by allowing the user to handwrite text (non-cursive) with a stylus. The messaging system will transmit messages wirelessly between the XScale and a laptop.

Project Background

Motivation
Mobile devices have become commonly used in people’s everyday life. Cell phones, for example, are widely used communication devices and have become smaller and smaller such that it has become infeasible to use a keypad with ease and speed. Utilizing a touch screen would allow users to navigate a cell phone’s system easily and text recognition will enable users to input messages more conveniently.

Background Work and Current State of the Art
Currently, very few, if any, cell phones are equipped with text recognition. Voice recognition, on the other hand, is more commonly seen in cell phones.

There have been many academic researches going on and some commercial products on the handwriting recognition area [1] [2] [3]. According to the difference in handwriting input, there are two approaches to do handwriting recognition. One is the off-line approach, where handwritten information is captured as an image, for example, scanning the writing on a paper. Another is the on-line approach, which deals with processing of a message as it is written using a stylus that captures information about the pentip. Our interest focuses on the on-line handwriting recognition (O-HWR for abbreviation). In our application, the user uses stylus to input through the X-scale Touchscreen, and we try to recognize the input on-the-fly. There are a couple of statistical methods for text recognition and the Hidden Markov Model is the most commonly used.

Challenges
- The main challenge of the project is to balance the accuracy and the real-time constraint of the text recognition.
- Getting the MiniGUI to cross compile on the XScale
- Be able to receive input from the touch screen
- Integrating Rosetta as a backend

Limitations
We restrict the user input to English non-cursive letters and numbers.
Project Description

The project consists of two main programs: a text messaging GUI on the XScale and a text messaging GUI on the laptop. Both GUIs will be simple and have some means of establishing a wireless connection between each other. For the XScale’s program, the GUI will look like the following diagram:

![Diagram of XScale's GUI]

The red portions of the diagram indicate that the program will handle touch screen input. The top black box displays the messages received from and sent to the laptop GUI program. Below that is another black box that contains text that has been recognized by our text recognition system from the user’s input. The user writes his text with a stylus within the designated box labeled “User Input Text”. After the user has finished his message, touching the “Send Text” box can send the message. If the user wishes to clear his message and start over, he can touch the “Clear Text” box.

The GUI for the laptop is similar to the XScale’s without the touch screen capabilities. Instead of having a text box for recognized text and a user input text box, there will be just one box for the user input text box. The user merely input text by the keyboard and clicks the “Clear Text” and “Send Text” with a mouse.

We are using the MiniGUI [5] library to create the GUI programs. It reduces the development time for our project in order for us to focus more on the text recognition system. In addition, the MiniGUI library is light-weight with high performance and low resource consumption, which is ideal for embedded systems.

Tools

- **MiniGUI:**

  We did a survey on available GUI libraries for embedded system with which we build our GUI, including GGI [6], Qt/Embedded [8], GTK [7], MiniGUI. We finally decided to use MinGUI because it is compact, light-weight, less resource-consuming.
MiniGUI adopts a tired architecture as above. It uses an abstract layer to deal with the low-level input and output. The GAL (Graphical abstract layer) is used for displaying output, while the IAL (Input abstract layer) is for input. User can implement his own input engine for MiniGUI, for example, a touchscreen input engine.

- **Rosetta:**
  Due to the time constraint, the complexity of the statistical analysis of the text recognition, and GUI and I/O problems, we were not able to implement an engine from scratch. Instead, we used Rosetta [9] as the backend for our project. Rosetta is an open-source, online text recognition that supports multi-stroke recognition that runs on the X Window System. Multi-stroke recognition analyzes the user’s individual lines of the written characters and translates them statistically. We were not able to cross-compile the X Window System on the XScale, so the X portions of Rosetta had to be converted into MiniGUI API. Rosetta, therefore, became a pure backend to our project.

**Implementation**

1) **Porting MiniGUI to XScale**
   The biggest challenge with MiniGUI is that we encountered a lot of problems either building it in host PC or porting it to target. It has three different running modes, and numerous features when compiling. So we spent a lot of time to figure out which features should be enabled and which should be disabled to finally get it work.

2) **Porting Rosetta to XScale and Integrating with MiniGUI**
   Rosetta has two phases in its recognition. First, it needs to learn the writing of the user. The program will ask the user to write a set of characters. Since individuals write characters in arbitrary sizes, Rosetta will normalize the input by scaling it to a constant size for analysis. Afterwards, it statistical analyzes the characters by recording how many strokes were use to create the character, the number pixels of the character, and the angles of the curves in character. It saves this analysis into a stroke database. In the second phase of Rosetta, it does the actual text recognition. It will prompt the user to write text and then normalize the input as was done in the learning phase. Each individual
character is then compared with the database to determine the possible matches. The best match will be selected and outputted back to the user.

In our integration of Rosetta with the MiniGUI front-end, we encountered countless problems. The main issue was Rosetta’s strong dependency on the X Window System library functions. One of the issues we encountered was how Rosetta represented its characters using X’s encoding. We created a conversion table in order for encoding translation to work properly with MiniGUI. In addition, we rewrote the input and output procedures from the X to the MiniGUI libraries. The integration of Rosetta with the MiniGUI front-end was not a smooth process.

Results

We have successfully ported MiniGUI to XScale, developed a GUI application with the MiniGUI library, ported Rosetta to XScale and integrated Rosetta as a backend for the GUI frontend. All these things turned out to be non-trivial and required a lot of work, as opposed to what we have thought.

To use the system, the user can first do some training by writing a set of characters as prompted by the application, which will be stored in a stroke database. Afterwards, the user can write on the XScale touchscreen using a mouse, and the handwriting will then be matched against his original training data for recognition. The recognition rate is decent but not very stable.

Future work

Due to time constraint and unexpected problems encountered, we did not implement the touchscreen input and wireless communication. For future work, we would like to finish these parts.

We would like to implement touchscreen input engine for MiniGUI, which requires us to get familiar with touchscreen driver for XScale. There were difficulties in calibrating the positioning of the touchscreen coordinates correctly.

Lastly, we would like to implement the wireless communication between the XScale and a laptop to build a real text messaging system. From the XScale perspective, we have all the data ready for transmission. Ideally, this integration with wireless capability would be not too difficult.

References


