ECEN 4024 Capstone Design Project:
Text Recognition for an Automated Test System

Ben Jespersen
Student
Electrical and Computer Engineering Department
Oklahoma State University
Stillwater, OK, USA

Matt Litchfield
Student
Electrical and Computer Engineering Department
Oklahoma State University
Stillwater, OK, USA

Chengjie Lin
Student
Electrical and Computer Engineering Department
Oklahoma State University
Stillwater, OK, USA

Abstract*—Testing is a vital component of engineering design. Sometimes this can be a tedious task that keeps engineers busy and delays designing of new features and systems. This is particularly true in the medical industry, where regulations require devices to be tested extremely thoroughly to ensure they pose no threat to human health. One such medical device is a tablet computer used to program an implanted nerve stimulator developed by Cyberonics. Up until now, Cyberonics has tested this device by hand, requiring people to physically tap the tablet screen and verify that it performs as expected. The company would like to be able to test it automatically; however, certain constraints in the tablet's software prevent this from being done through the traditional method of inserting additional test features into the main software. To solve this problem, Tietronix Software Inc. is currently in the process of developing an Automated Test System to test the tablet using a robotic stylus and camera instead. This system will test the tablet in a completely non-intrusive way, without requiring extra software on the tablet. It has great potential to streamline the engineering process by allowing engineers to work on other tasks while testing is carried out automatically. As part of this effort, Tietronix recruited a Capstone Design team at Oklahoma State University to develop a software module for the system. This module uses image processing technology to recognize specific strings of text on the tablet screen. The goal was for a test engineer to be able to input a specific text string by hand or from a file prepared in advance. The string recognition module will then search for this text on the tablet screen. When a string is found, the software module also taps the string on the tablet screen using the robotic stylus. This allows the test system to navigate through an application on the tablet without requiring an engineer to be present. The module also logs the results of each search to a file that can be referenced later. This allows an engineer to automatically run a sequence of text searches to determine if the tablet is working properly and displaying the correct information. The engineer need not be present at all; once the program is started, he can work on other things and check the output log file at his convenience. This module was successfully prototyped, and the details of its design will be discussed in this report.

***Index Terms*—Automated Test System (ATS); Capstone Design; Image Processing; Levenshtein; Optical Character Recognition (OCR); Tesseract**

Contents

I. Introduction

The purpose of this report is to describe in detail a project undertaken by Ben Jespersen, Matt Litchfield, and Chengjie Lin for the electrical and computer engineering Capstone Design course at Oklahoma State University (OSU). The overall project goals and purpose will be described, followed by details of the subsystems designed by each team member. Methods of testing will also be discussed, along with the results thereof. Finally, possible future expansion of the project will be explored.

Graders for ECEN 4024 may wish to refer to Appendix A while reading. This appendix lists which sections of the report were authored by each member of the team.

II. Project Description

A. Overview

Unlike most of the projects undertaken for Capstone Design, this one was sponsored by a company outside OSU. Tietronix Software Inc. is working on an Automated Test System (ATS) to test a tablet developed by Cyberonics for use in the medical field. This tablet is used by physicians to read data from and reprogram electronic devices implanted in patients. Due to the critical timing constraints involved in communicating with the implanted device, the programming tablet cannot be tested using traditional automation methods involving extra software running alongside the primary application. The tablet must be tested externally, with no additional software installed. Traditionally this has been done by test engineers who operate the tablet manually and record the results. However, Tietronix is in the process of developing the ATS to do this job automatically instead. The ATS consists of a robotic XY Plotter with a stylus attached and a camera mounted overhead. The tablet is placed under the system, and the system is able to take pictures of the tablet, process them, and tap certain locations on the screen in response. This method of testing could greatly improve the efficiency of development for the medical tablet because it will allow engineers to work on other tasks while the long, repetitive work of testing is being performed.

As part of this effort, Tietronix contacted OSU to recruit a Capstone Design team to add a software feature to the ATS. This new feature is a text recognition module, and it is the project detailed in this report. The goal was for the module to take in a string of text from a user or an input file, capture an image of the tablet screen with the camera, and process the image to determine if the input text appears on the tablet screen. If the text does appear, the ATS should tap the screen at the location of the text. The system should also log the results of the search to a file that a test engineer can read to analyze the tablet's performance.

This software module has great potential for usefulness in the ATS. First, it can allow automated testing of new text features when they are added to the tablet. An example might be adding support for a new language. One reason this would need to be tested is to ensure that a word in the new language does not become so long that it overlaps the edges of a button on the tablet screen. Another use for the module comes from the fact that the ATS taps the screen at the location of the text. If the text being searched for appears on a button, then this feature can be used to navigate to different pages in the tablet software or activate features of the tablet. This allows the ATS to automatically navigate throughout the software simply by having a test engineer specify a sequence of text strings to tap. A tester would not need to set up specific (x,y) coordinates to tap on the screen in advance; he could simply prepare a file containing a list of the text on each button he wants pressed.

B. Design Specifications

The following design specifications were defined early in the project. Many of these were given to the team directly by Tietronix, though some of them were defined by the team with the help of its mentor, Dr. George Scheets. The specifications were selected such that the work load would be appropriate for a single-semester capstone project for three students.

1. The string recognition shall accept string input from a language file or user input.
2. The string recognition shall locate a given string within a region of interest based on the string input. The region of interest may be a subsection of the whole screen.
3. The string recognition shall support the English language and at least one other language from the following list: Danish, Dutch, French, German, Italian, Norwegian, Spanish and Swedish.
4. The string recognition shall positively identify the string by activating (lowering) the stylus at the location of the string.
5. The string recognition shall be able to produce a similarity value that represents how likely the string was located expressed as a percentage from 0 to 100%.
6. The string recognition shall be able to translate the string coordinates on an image to a physical (x,y) position on the XY Plotter.

7. The string recognition shall navigate four layers deep into the physician programming software application.
8. The string recognition shall take less than 30 seconds to identify the string at runtime.
9. The string recognition shall be able to function under typical indoor lighting conditions with minimal glare on the tablet screen.
10. The string recognition shall be able to recognize strings with the following font characteristics:
    a. High-contrast (font color is significantly different than background color).
    b. Size between 12 point and 20 point.
    c. Standard serif or sans-serif style (not script, gothic, or other atypical font styles).
11. The string recognition shall provide the ability to log "pass/fail" results to a text file to indicate if an expected string was presented on a given static screen.
12. The string recognition shall be able to run on one of the following operating systems: Windows, Linux/GNU, or UNIX.

III. Design Details

A. Work Breakdown

For this project, none of the members had any prior experience with image processing or stepper motor control. Because of this, blocks were made based on preliminary research and our requirements list. It made sense to estimate the difficulty of each task then group them together in blocks that are closely related, such that the overall difficulty would be about even.

The image enhancement and optical character recognition blocks made sense to group together. Because there was no prior experience with OCR, enhancing an image to yield the best results from Tesseract would require a lot of trial and error. It would be very difficult to have two people try and work on those blocks independently. Ben became in charge of these two blocks.

The next grouping was the image acquisition, user input, and OCR interface. These made sense to group together because they all happen before the image enhancement and OCR. In order to balance the workload, the similarity calculation was added to this group. This group was tasked to Matt.

The last grouping was the blocks needed to be done after an image was processed. They mainly make the plotter move to the correct location and make the log file after the cycle is complete. Chenjie worked on these blocks.

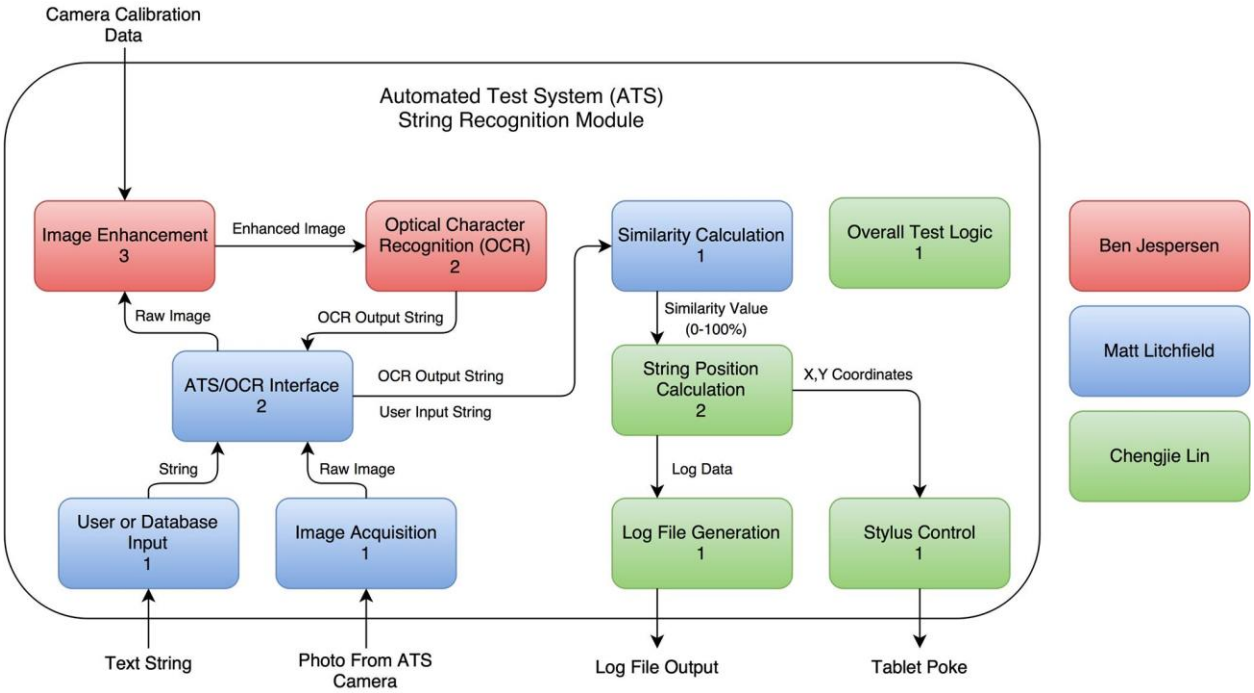


Figure 1: Functional Block Diagram

B. Image Enhancement

Before the text on the tablet can be recognized, the image from the camera must be cleaned up and prepared. This is the job of the Image Enhancement subsystem. The subsystem consists of the following functions:

- detectLetters
- eraseBackground
- getAvgBoxHeight
- isolateText

All of these functions utilize an open source image processing suite called OpenCV. This package was already in use by Tietronix in the existing ATS software, so the team chose to continue using it for compatibility. It is also an extremely popular package, so there are many resources available for learning to use it.

The first important function in the image enhancement subsystem is detectLetters. This function is a modified version of code provided by user dhanushka at stackoverflow.com, and its purpose is to determine regions of the image where text appears. The function takes in an image and an expected font size and returns a set of rectangles, where each rectangle represents a region of the screen which is likely to contain text. Each region is as small as possible without cutting off portions of letters.

The core operation of the function detectLetters is unchanged from its original version on stackoverflow.com, but some changes were made to fit the needs of this project. First, the function was translated to Java from its original language, C. Second, some constants were changed to better fit the type of image being processed in this project. Third, a feature was added to allow the function to search for text differently based on the expected font size. Finally, another feature was added to merge regions produced by the function if they are close to one another. This helps to filter out overlapping regions and combine regions containing individual words into single lines of text.

The function detectLetters works in the following way: first, the image is converted to grayscale. A new image is then constructed in which each point represents the gradient of the original image at that point. The result is that locations on the original image which contain sudden changes from dark pixels to light pixels (or vice-versa) show up as light points on the new image, while all other points show up as dark on the new image. A threshold is then applied to the new image to convert it to pure black and white instead of grayscale. The resulting image is therefore black with white pixels arranged along the outlines of objects from the original input image. These outlines can represent text or other objects that have sharp, contrasting edges.

Next, a morphological operation called “close” is performed on the image. This converts all pixels which are adjacent to white pixels to white, then immediately converts all pixels adjacent to black pixels to black. If the expected font size parameter is large, then the close operation extends 11 pixels to the sides instead of only affecting immediately adjacent pixels. If the font size is small, the operation extends only 7 pixels to the sides. The effect is that if multiple small objects were arranged close to one another horizontally in the original image, then the objects will be merged into wide, horizontal, predominantly white regions. This occurs most often in regions of text, because the letters are small objects arranged horizontally. Because the characters in text are so close together, the close operation tends to overlap white pixels between characters, resulting in a single white region instead of multiple small ones. It also tends to remove black pixels within lines of text. Non-text objects, however, tend to have more than 11 black pixels on either side of them. Since this is beyond the range of the close operation, the white in these objects is widened and then immediately shrunk back to the original size, resulting in no net change.

Next, the contours of all the white regions in the closed image are calculated. The contours represent the outlines of each white region. For each contour, the bounding box is calculated, resulting in a rectangular box around each white region. If the pixels inside a box are 35% white or more, if the box is at least eight pixels wide, and if the box is between 12 and 200 pixels high, then the box is saved as a region of detected text. All boxes which do not fit these criteria are discarded. This tends to filter out the objects which were not expanded by the close operation because there are insufficient white pixels in these regions for them to be saved.

Finally, the boxes are checked to determine if any of them are very close together. Any boxes which are close to each other horizontally and which have very similar heights in the image are likely to represent text that appears on the same line. These boxes are therefore merged into one so that each single line is represented as a single box. After this, all the boxes which remain are returned as the set of detected text regions.

The next function to discuss is eraseBackground. This function takes in an image and the set of detected text regions as inputs. It sets all pixels in the image to white if they are outside the regions of text. Pixels inside the regions are left unchanged. The resulting image is then returned. This erases all background elements which are not contained within the regions of text.

The third function is getAvgBoxHeight. This function takes in the set of detected text regions and calculates the average height of the regions. This average height is then used outside the function to determine what font size should be expected.

Finally, the last function to discuss is isolateText. This is the primary function which is called to execute the image enhancement subsystem. It takes in a raw image captured from a camera and a rectangular region of interest as inputs, and it returns an enhanced image as its output.

The function begins by increasing the contrast of the image and calling detectLetters with a large expected font size. The heights of the resulting boxes representing text regions are then averaged using getAvgBoxHeight. If the average height is smaller than a specific threshold, then detectLetters is called again with a smaller expected font size to improve its accuracy.

Next, the image is converted to grayscale and everything outside the region of interest is erased (set to white). This ensures that any text which appeared outside the region of interest is not recognized. The resolution of the image is then doubled. While this does not increase the sharpness or focus of the image, it does increase the pixel density, which improves the accuracy of morphological operations which follow.

Once the resolution has been increased, a set of sub-images is created, where each sub-image is the portion of the full image inside one of the detected text regions. Each region in the set has a corresponding sub-image created. Note that these sub-images still exist as part of the full image; in other words, changing a sub-image also affects the corresponding location in the full image.

For each sub-image, a basic threshold is applied to convert the image from grayscale into plain black and white. This is known as a binary image. If the outline of the resulting sub-image is predominantly black, then the region must have contained light text on a dark background. The sub-image is then inverted to change it to black text on a white background. If instead the binary sub-image was predominantly white to begin with, then the region was already dark text on light background. Instead of inverting, the threshold is re-done as an adaptive threshold. This improves the clarity of the text by accounting for the fact that different sub-images may have different brightness and contrast. The adaptive threshold automatically selects a threshold value to fit the characteristics of each sub-image.

The next step is to erase the background outside the text regions, which still appears as shades of gray. This is done by calling eraseBackground, which was described previously. At this point, the full image consists only of black text on a plain white background. However, because of inherent inaccuracies in the thresholding process, there is still noise in the image which must be filtered out for the character recognition to work properly.

To filter out noise, the morphological operations “close” and “open” are performed. The close operation was described above; any pixel adjacent to a white one is converted to white, then any pixel adjacent to a black one is converted to black. The result is that small black specks of a few pixels or less are erased completely from white regions. This filters out noise in the white background of the image. The open operation is exactly the opposite; the conversion to black is done first, then the conversion to white. This has exactly the same effect except that it removes small white specks from black regions. This filters out noise which appears within the black text.

At this point, the enhanced image is complete. The raw input image has been converted to plain black text on a white background. Both light text on dark background and dark text on light background have been converted to plain black text. Some non-text objects may still appear in the image, but by this point they are few enough that they will not impede the character recognition. Noise has also been filtered out to produce very clean, sharp text that can be interpreted easily.

Figure 2 demonstrates the operation of the image enhancement. The left half is a raw input image taken directly from the ATS camera, while the right half is the result of running the image enhancement. It can be seen that the edges of the tablet, the table, and the non-text elements on the tablet screen have been filtered out, leaving only the text remaining.

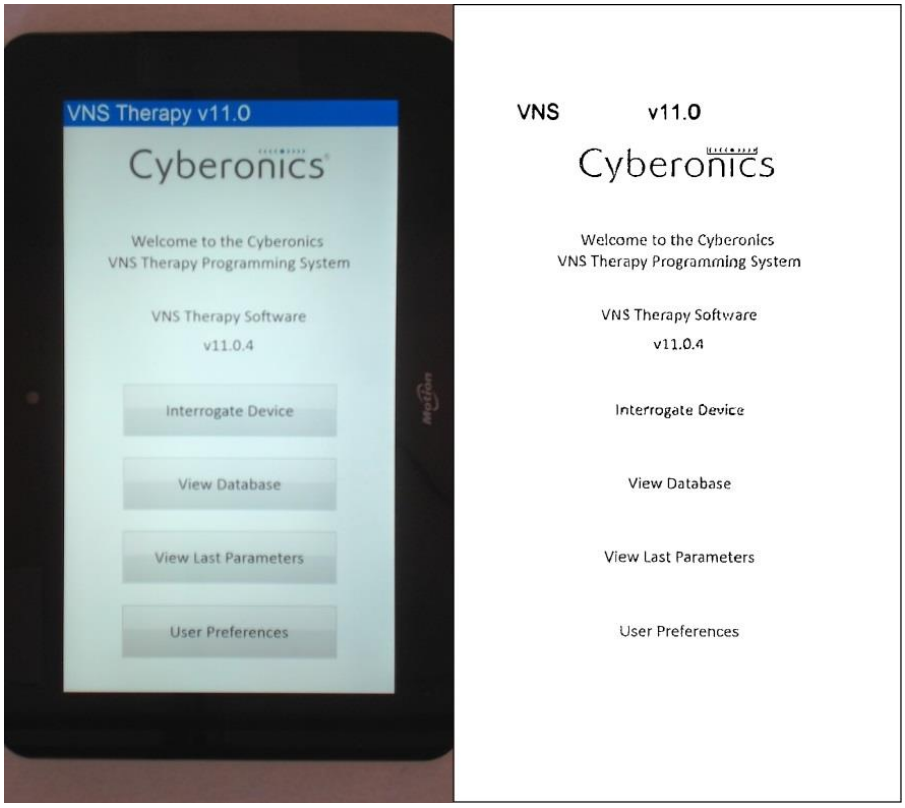


Figure 2: Image Enhancement Example

C. Optical Character Recognition (OCR)

The OCR subsystem is responsible for converting the enhanced image into a text file that can be read by the program. It is also responsible for determining the location of a specific string of text on the image.

Converting an image into a text file is a very complex task that has already been solved by large engineering teams working for companies. For this reason, an existing open-source program was selected to perform this step. This program is called Tesseract, and it is currently maintained by Google. It supports a multitude of languages, including English and Dutch, which were chosen as the two languages to be supported by this project. Users of the Object Recognition System described in this report will need to install Tesseract on their systems. Once this is done, the user still only needs to run the Object Recognition System program, and this program will handle using Tesseract correctly. The user never needs to interact with Tesseract aside from the initial installation.

The black and white image produced by the enhancement subsystem is passed to Tesseract as an input. Tesseract then produces two files made up of plain text. The first is a basic .txt file which contains all the text from the image. Tesseract breaks the text up into separate lines in an attempt to give it the same appearance and relative positioning as the text in the original image. The second file is a custom .box file. This file lists each individual character contained in the .txt file. The coordinates of a rectangle appear next to each character; this rectangle represents the region where that character appears in the image.

Once the conversion performed by Tesseract is complete, the resulting .txt file is passed to the similarity calculation subsystem, which will be described later. This subsystem returns the string of text from the .txt file which is most similar to the desired string. This occurs only if a string was found which meets a minimum similarity threshold.

After this string has been returned to the OCR subsystem, the .box file is searched for this specific string by calling the function getLocationInImage. In this function, the rectangles representing the locations of the first character in the string and the last character in the string are extracted from the file. From these, a new rectangle is created which extends from the upper left corner of the first character to the lower right character of the last character. This results in a rectangle that represents the location of the entire string. Finally, the center point of this new rectangle is calculated. This center point is considered to be the

location of the string in the image. This location is then passed to the string position calculation subsystem in order for it to be converted to a physical position in the ATS work area.

Figure 3 shows the Tesseract conversion in action. The left half of the image is an example of an image that has already gone through the enhancement process. The right half of the image is the corresponding text file generated by Tesseract. It can be seen that the text recognition by Tesseract is very accurate once the image has been well enhanced.

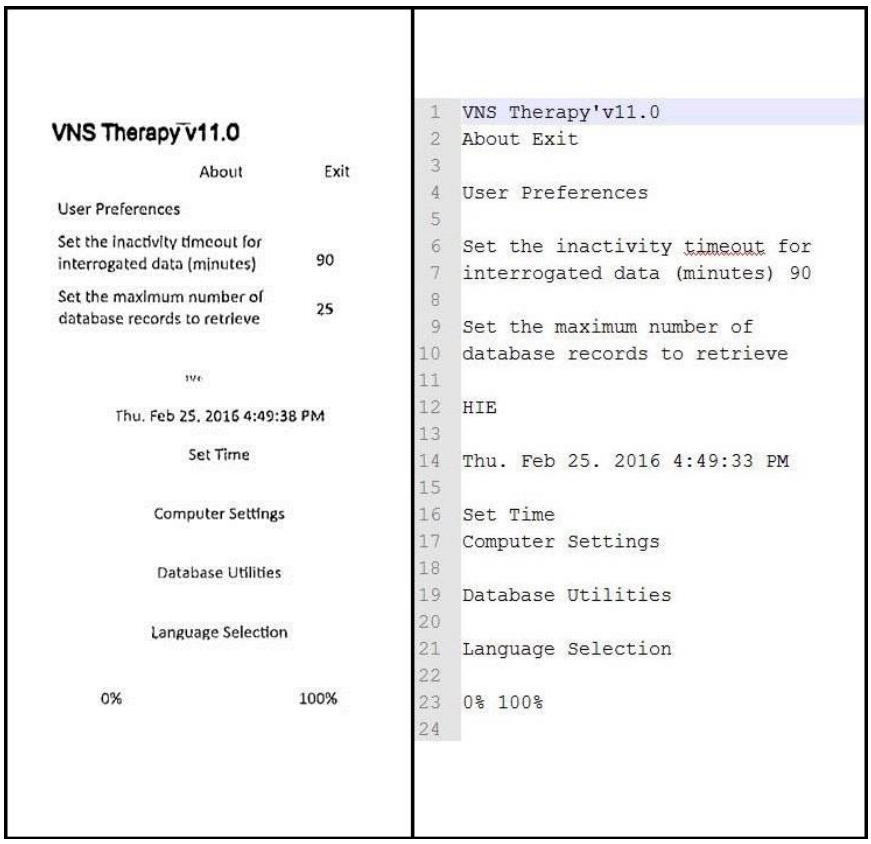


Figure 3: Optical Character Recognition Example

D. Image Capture

Before any attempt can be made to recognize a string, a picture must be taken with the provided webcam. Because OpenCV is used for other parts of the program, it made sense to continue using it. There are already built in objects for instantiating a web camera and capturing an image. One problem we did face was focusing the camera. In order to have a clear enough image to enhance, a better focus than what was being provided by the built in autofocus of the camera would have to be achieved. Logitech provides a program that can give manual control of focusing the camera. This program has to be run before ours, but only on a redeployment of the system that might have caused loss of focus.

E. ATS/OCR Interface

Tesseract, the program that is being used to perform the OCR, is a command line utility. As such, it needs a way to be called from the Java runtime environment. Because all that is needed is a simple command call with arguments, the Java Runtime library can be used. First, an instance of the current runtime must be created. Then the exec() command with a string as its function argument can be called. If Tesseract is not installed or the PATH environmental variable does not include the directory where Tesseract was installed, trying to run the program will produce an error because the command could not be found.

F. Similarity Calculation

To determine how similar strings are, the Levenshtein distance was calculated and subtracted

from the length of the two strings and then divided by the length of the smaller of the two strings. Levenshtein distance determines the number of character insertions, deletions, or substitutions required in order to make one string into another. The reason that it was chosen was that it allows for comparisons between different sized strings, and there was not a need for transpositions because that is not a common problem with Tesseract. It is a very common computer science problem, and the actual implementation will be left as an exercise to the reader.

Once we have the text from the processed image and the string we are looking for, we begin a windowed slide looking for a match above our threshold. For example, if the string we were looking for was 4024 and the text from the image was ECEN 4024, a window of 4 would be used.

In the current implementation, the first iteration that returns a similarity above the threshold will return the windowed string. As a possible improvement, a best fit search could be implemented instead of a first fit method.

There are three methods in the LevenshteinDistance class. All are implemented statically. The first returns the Levenshtein distance as an integer. This is the method where the Levenstein algorithm is implemented. An iterative version was chosen for its clarity. The next returns the confidence value as a double. The final method is the window slide that is explained above.

Iteration	To Find	Windowed String	Levenshtein Distance	Similarity
1	4024	ECEN	4	0%
2	4024	CEN	4	0%
3	4024	EN 4	4	0%
4	4024	N 40	4	0%
5	4024	402	2	50%
6	4024	4024	0	100%

Table 1: Levenshtein Distance

G. User Input

The goal of the user input block was to allow both manual input and a test bench file. This was accomplished by constructing a graphical user interface (GUI). The GUI had input boxes for various search functions along the bottom and a test bench selector under the “File” tab. Among the search functions were a lot of options such as language and boundaries that allowed for more exhaustive testing. The image is 1920 x 1080, so entering values between those numbers bounds the search to the box drawn by the two sets of coordinates.

The GUI also displayed three images. From left to right: the raw captured image, the enhanced image with the text identified and non-text stripped away, and the original image with a red indicator dot showing where the string was found. Creating a full GUI was beyond our specs; however, it does a good job of showing the overall process, as well as creating an easy to use product.

The GUI was constructed from a base JFrame. In the constructor the frame was configured as a BorderLayout. JTextFields and JLabels were placed in a JPanel and placed in the south area. Three JLabels were constructed and placed in the middle three regions. They were given the three different images to show how the image was processed to the user. In the north section, a JMenu was made for starting the JFileChooser and calibration window.

In order for the images to be updated, a thread was started that polled the image files at a set time. The buttons also needed ActionListeners so that they would know what to do when pressed. The frame implemented the ActionListener interface and assigned itself to be the ActionListener for all of its objects. A case statement was used to determine what to do on each button press. In the case of preforming a test, a threaded call was made to the string finding code. All of the threads were written as anonymous inner classes because the calls were not very lengthy.

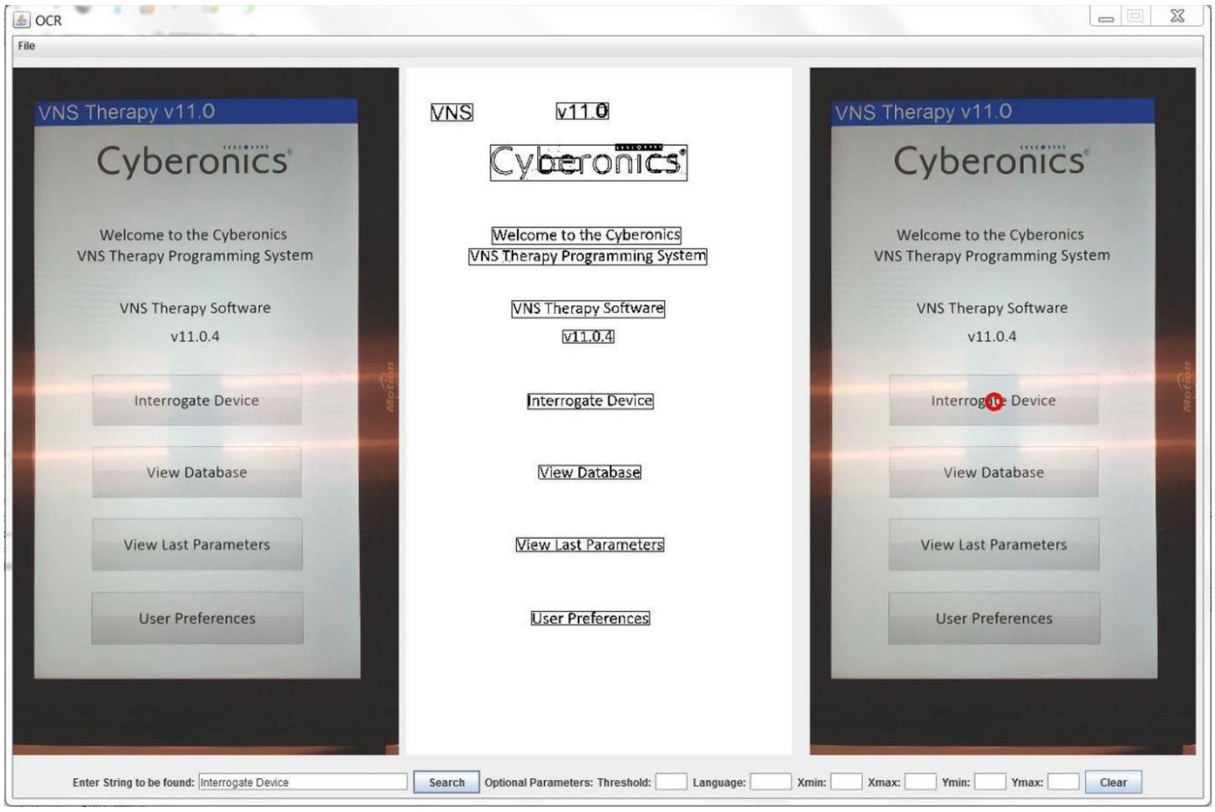


Figure 4: Graphical User Interface

H. XY Plotter

1) Introduction to XY Plotter

The XY plotter we use is provided by Tietronix, the Makeblock XY-Plotter Robot Kit v2.0. It has the following characteristics. XY Plotter Robot Kit is a drawing robot that can move a pen or other instrument to draw digital artwork on flat surface. It can also be developed into a laser engraver by adding the Laser Engraver Upgrade Pack.

- Frame: Anodized aluminum
- Physical Dimensions: 620mm x 620mm x 140mm
- Working area 310x390mm
- XY Accuracy 0.1mm
- Max Working Speed: 50mm/s
- Power: 100-240 V~50/60Hz AC/DC Power adapter, 12V/3.0A
- Main Controller Board: Makeblock Orion (Arduino Leonardo)
- Makeblock A4988 stepper motor driver
- Makeblock K-Power 9g Micro Servo
- Makeblock 42BYG Stepper Motor
- USB A-Male to B-Male Cable

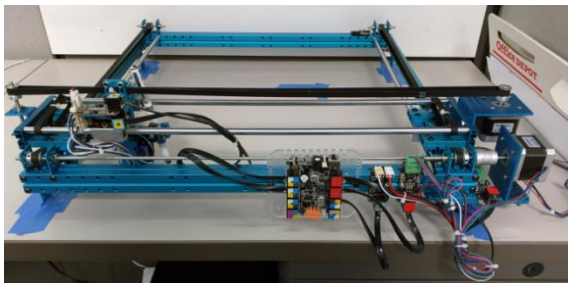


Figure 5: XY Plotter Side View

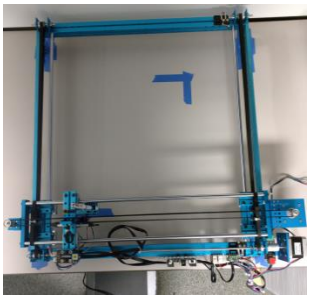


Figure 6: XY Plotter Top View

2) Configuring the XY Plotter

We need to configure the XY plotter first to make it prepared for this project. First, we plug in the

power and turn on the XY plotter. Second, connect Arduino Leonardo to your laptop. After that, we need to download Tietronix's firmware codes for XY plotter and open the codes with Arduino IDE. Finally, we can upload the codes to Arduino Leonardo. By doing this, the XY plotter is configured for this project.

I. Stylus Control

To control the stylus, first of all, we need to establish the serial communication between laptop and Arduino Leonardo. To move the stylus, you will need to send G code to Arduino Leonardo from your laptop. The easiest way to do this is to use Arduino IDE, but it is not really what we want because the whole project is Java written. So we need to develop the serial communication codes for Java working environments so the Java codes can also send G code to Arduino Leonardo. Luckily, the Tietronix's baseline codes provide this function and we need to modify the codes for our projects.

The baseline codes is like the following:

```
private void setPenPosition(float Z) {
        if(isConnected){
           String cmd = "G01 Z" + Z;

 while(!CM.sendCmd(cmd,SetPenPosCommand
 Timeout));
           current_Z = Z;
        }
        else{
           System.out.println("XYPlotter: Not
 Connected!");
        }
 }
```

Then we need to translate from coordinate on image to physical coordinate to XY plotter so that XY plotter can use the physical coordinate to go to the correct point. To do that, first, we need to set the tablet on a correct position, which we can make sure by using the Calibration box in the camera view. We need to put the tablet in a position that fit into the Calibration box.

Then we move the stylus around to go to the four corners of the tablets and record the physical coordinate for the four corners, which will be as following:

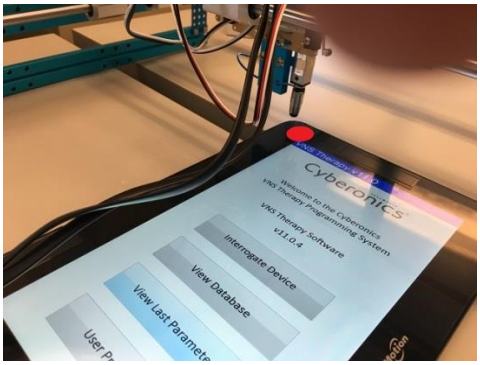


Figure 7: Upper Left Corner's Physical Coordinates: (105,340)

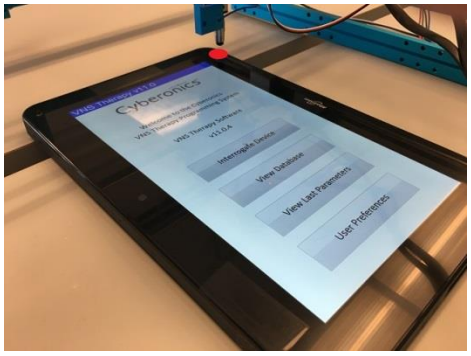


Figure 8: Upper Right Corner's Physical Coordinates: (290,340)

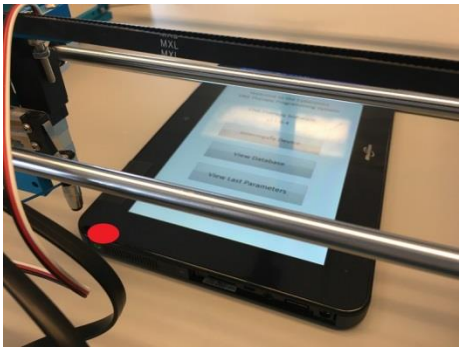


Figure 9: Lower Left Corner's Physical Coordinates: (105,35)

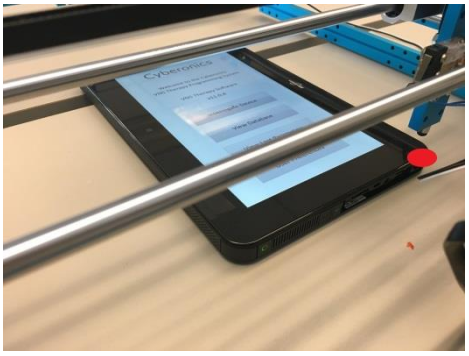


Figure 10: Lower Right Corner's Physical Coordinates: (290,35)

So we know the physical coordinates for the tablet's four corners. It will be rather easy to translate the coordinates on the image into the physical coordinates.

For example, we get a coordinate on image which is (x1,y1) and it is also known that the image is 1920*1080 pixels and the physical coordinate assumed to be (x2,y2). We know that the (0.0) on the image correspond to (105,35) on the XY plotter and (1080.1920) corresponds to (290,340) on the XY plotter. Then we calculate the relative pixels distance between (x1,y1) and (0,0) and translate the pixel distance into the physical distance and add to the (105,35) physical coordinate. Then formula is as follows:

```
Double x1=a.strLocation.x;
Double y1=a.strLocation.y;
double boundX=1080;
double boundY=1920;
Double x2=(x1/boundX)*185+105;
Double y2=0.0;
if(centery<300)
{
  y2=(1-y1/boundY)*305+35;
}else{
  y2=(1-y1/boundY)*305+35+10
}
```

Notice that if y2 is greater than 300, we need to add 10 more because in the experience, the higher point you move your stylus to, the more error you will have. So in order to reduce the error, it is needed to add 10 to the original formula to have a offset and the issue is solved to a great extent.

To make the stylus tap the screen for an appropriate amount of time, we need to change the ZDown parameter to a smaller number.

Original:

```
private float ZDown = 100;
```

Modified:

```
private float ZDown = 65;
```

And in the XYPlotter.java, remove the old dependencies with Tietronix's baseline classes and add dependency with the classes of this project.

J. *Log File Generation*

To make it more user friendly, we decided to generate the log files to keep track of things and record the status.

The Log file generation is an easy part. We will generate log file every time we process a cycle. The contents of the log file will be as followings:

The contents of the log file are:

Time stamp: When this process begins
Success or failure: Is it a success or a failure
Target string: Strings we are looking for
Location of the string: The position of the string on the image
Confidence value: Confidence of the correctness of this process
Confidence threshold: Confidence we set as a threshold to recognize the string.

The example of log file is as below, as can be seen, it has all the elements to show useful information about the status of the process.

```
Test run on 2016-04-14 at 11:07:47.582
Success
String searched for: Interrogate Device
String located at image position: (515.75, 955.5)
Confidence value: 100.0%
Required confidence threshold: 70.0%

Test run on 2016-04-14 at 11:08:02.634
Success
String searched for: Cancel
String located at image position: (520.0, 1624.25)
Confidence value: 83.33333333333334%
Required confidence threshold: 70.0%

Test run on 2016-04-14 at 11:08:18.946
Success
String searched for: View Database
String located at image position: (515.0, 1154.5)
Confidence value: 100.0%
Required confidence threshold: 70.0%

Test run on 2016-04-14 at 11:08:38.457
```

Figure 11: Log File

K. Overall Logic

Finally, we need to clean up the project into different function blocks and each of them contains several classes. For the String Recognition block, it is in charge of recognizing the desire string in the processed image and output the location of the string.

It has the following classes: FindString.java, LevenshteinDistance.java, RunCommand.java and StringRecognizer.java

As for the Utilities block, it provides utilities to the generate log file, move stylus, coordinate transformation and so on. The Classes used for this block are AbstractExternalDevice.java, Attributes.java, CommwithArduino.java,

CommunicationManager.java, DeviceManager.java, Serial.java, XYplotter.java and Util.java

For the GUI block, it serves as the GUI for user to make it more user friendly. The Class list is FileTreeView.java, JImagePanel.java and SingleRootFileSystemView.java

Last but not least, we have Image preprocess block. It helps to preprocess image for the string recognition. It has the following classes: ImageInput.java, OCRFrame.java, CalibrationFrame.java and Enhance.java

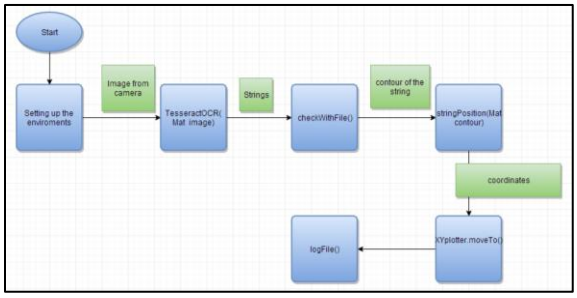


Figure 12: Overall Logic Flowchart

IV. Test Methods and Results

Once the project was integrated and complete, it was important to perform final tests to ensure it met the specifications agreed upon. Some of the specifications can be verified simply by observing the program functioning, while some required slightly more careful testing. While reading this section, the reader may find it useful to periodically turn back to Section II.*B* of this report to reference the project specifications.

The GUI described previously contains a text field where a user can enter text for the program to search for. In addition, it also contains a menu in the upper left corner labeled "File." One of the options in this menu is to load a file saved on the user's computer. This allows the user to open a file containing a sequence of text strings that he or she has prepared in advance. The program will then run through these strings in order, searching for each one. This satisfies specification 1, which requires the program to accept user or file input.

Specification 2 requires the program to search within a region of interest that may be defined by the user. This was met by including entry fields on the GUI to allow entry of a rectangular region of interest. Because the GUI displays all text which was recognized in each run, correct operation of the region of interest was verified by observing that the image of recognized text was blank everywhere outside the region of interest.

Specification 10 requires the program to recognize high-contrast font between 12 and 20 point size. To test this, a sheet was printed with both black text on white background and white text on black background. Both colors were also printed in the maximum and minimum required font sizes, resulting in four test regions on the page. Text was successfully recognized in all four regions when tests were run, verifying these requirements.

The remaining specifications were tested using an input file specifying a sequence of text strings to search for automatically. Almost all of the text in this sequence represents buttons, so the sequence navigates through the tablet software as it executes. The sequence begins on the tablet's home page, which contains four buttons. The sequence detects the text on each button in turn, which navigates the tablet to the corresponding page. When each page is opened, the test sequence searches for "Exit" or "Cancel" to move back to the home page in preparation for searching for the next button. This results in navigation through four different screens as required by specification 7. The fourth screen of the tablet software includes a language selection option. Instead of cancelling out of this screen, the test sequence searches for and taps the appropriate text to change the language from English to Dutch. It then backs out to the home screen, then returns to the language selection to verify that recognition of Dutch text is successful. The sequence then changes the language back to English, backs out to the home screen, and searches for a final non-button string of text.

This test sequence runs successfully. The program is able to navigate through all the desired screens and options using only text recognition. A log file was successfully generated during the test, and it was analyzed after the run. It correctly displayed the date and time of each text search, a pass/fail indicator showing whether the text was found, an assurance percentage of how likely it was that the correct text was found, the location in the image where the text was found, and the threshold assurance percentage required for the recognition to count as a pass. The test sequence was used multiple times for demonstration at the end of the course, allowing the program to run automatically while the team discussed it with observers. This very successfully demonstrated the goal of the project; the program ran automatically without requiring intervention from the team until the test sequence ended.

V. Possibilities for Further Development

As this project is merely one component of a larger work in progress, it is important to identify

potential ways in which it can be improved and expanded as it is added into the full system. Throughout the design process, the team has identified areas which can be improved or expanded upon but which were not able to be addressed as part of this project due to time constraints.

The first area of improvement relates to the accuracy of the OCR subsystem. Tesseract has a feature called training, which allows it to be specifically tailored for certain fonts and text styles. This is done by taking several sample images of the text that will be recognized and telling Tesseract exactly what all the text says. If this is done with several different images, Tesseract learns how to recognize that type of text more accurately. The team considered doing this for the project, but there was also another alternative for increasing accuracy: improving the image enhancement subsystem. There was not time in a single semester to do both, and the decision was ultimately made to spend what time was available on the image enhancement rather than training Tesseract. However, the system can benefit greatly from both, so training could provide a very significant improvement.

The next area of improvement relates to comparing the text found on the tablet to the text being searched for. As previously mentioned, the current implementation locates the first string on the tablet which has a similarity percentage above the threshold defined by the user. It is possible, however, for more than one string on the screen to meet the threshold. If the first string which meets the threshold is not quite as similar to the desired string as the second string which meets the threshold is, it may be preferable to return the string which is more similar rather than simply the first one found. This is a relatively simple feature that could be added in the future.

Finally, the accuracy of the XY plotter's taps on the tablet screen can be improved as well. The current implementation uses a simple linear transformation from the string's coordinates on the camera image to its physical coordinates in the XY plotter's area, and this transformation results in some error along the y-axis. A more robust transformation could be developed to ensure the stylus always taps directly in the center of the located string.

VI. Conclusion

This project ended with great success. While there is certainly room for expansion and improvement, the team was able to fully demonstrate proof of concept for using image processing and text recognition technologies in a testing environment. Because the end product is a standalone module that will be integrated into the Tietronix ATS, there is a wide range of ways it could be used, including verification of text, simple automated navigation through software screens, and more.

With any design project, it is important to ensure that the end product delivered meets the expectations and needs of the client for whom it was made. About two weeks before the final course deadline, the team was able to demonstrate the working design to Chris DuPont, the Director of Product and Business Development in the Life Science Division at Tietronix, who had been the primary point of contact with the team throughout the project. After the demonstration, the team requested that Mr. DuPont provide a brief statement describing his opinion of what he had seen. The following quote is the statement he gave:

> In the medical device industry, the overhead of design controls, testing, and regulatory compliance can be daunting. The challenge for this Senior Design Team was to take a "real-life" problem of designing and implementing a working prototype to passively test a commercial FDA Class III Physician Programmer using a robotic system which would ultimately relieve the human of the burdensome task of repetitive testing. The Physician Programmer interrogates and programs a human implantable pulse generator used to treat neurological disorders. This team, mentored by Dr. Scheets, met the task head-on and made steady progress each week. The team never missed a goal and our weekly status meetings always started on time - which I much appreciated. I was most impressed as I personally observed a demonstration of the fully working prototype last week in Engineering South. A BIG thanks goes out to OSU, Dr. Scheets, and this Senior Design Team - Ben Jespersen, Matt Litchfield, Chengjie Lin - for a "Job Well Done!"

VII. References

[1] A. Gohr, "Linux OCR Software Comparison", Splitbrain.org, 2010. [Online]. Available: http://www.splitbrain.org/blog/2010-06/15-linux_ocr_software_comparison. [Accessed: 26- Jan- 2016].

[2] A. Wilson, "Developers Look to Open Sources for Vision Algorithms", Vision-systems.com, 2014. [Online]. Available: http://www.vision-systems.com/articles/print/volume-19/issue-3/features/developers-look-to-open-sources-for-

vision-algorithms.html. [Accessed: 25- Jan- 2016].
[3] C. Hodges, “Automated Test System (ATS) Use Cases and Features,” Tietronix Software, Inc., Houston, TX, Tech. Rep. 1.0, May 2015.
[4] Cognitiveforms.com, "CuneiForm". [Online]. Available: http://cognitiveforms.com/ru/products_and_services/cuneiform. [Accessed: 26- Jan- 2016].
[5] Embedded Vision Alliance, "SimpleCV: Is An ‘OpenCV For The Masses’ Necessary?", 2011. [Online]. Available: http://www.embedded-vision.com/news/2011/09/30/simplecv-opencv-masses-necessary. [Accessed: 25- Jan- 2016].
[6] "Extracting text OpenCV", *Stackoverflow.com*, 2014. [Online]. Available: http://stackoverflow.com/questions/23506105/extracting-text-opencv. [Accessed: 29- Apr- 2016].
[7] Opencv.org, “OpenCV”. [Online]. Available: http://opencv.org/. [Accessed: 24-Jan-2016].
[8] P. Harker et al., “Object Recognition For a Robotic User Interface Automated Test System,” Stillwater, OK, Tech. Rep. Dec. 2015.
[9] Researchgate.net, "Which is the best opencv or matlab for image processing?". [Online]. Available: https://www.researchgate.net/post/Which_is_the_best_opencv_or_matlab_for_image_processing. [Accessed: 25- Jan- 2016].
[10] S. Dhiman and A. Singh, "Tesseract Vs Gocr A Comparative Study", International Journal of Recent Technology and Engineering, vol. 2, no. 4, 2013.
[11] Simplecv.org, "SimpleCV". [Online]. Available: http://simplecv.org/. [Accessed: 25- Jan- 2016].

APPENDIX A: REPORT AUTHORSHIP

The purpose of this appendix is to indicate which members of the team wrote each section of this report. This was a requirement for the Capstone Design course to facilitate grading. Each team member is listed below, along with the sections of the report he wrote.

Ben Jespersen:

- Abstract
- I: Introduction
- II.*A*: Project Description
- II.*B*: Design Specifications
- III.*B*: Image Enhancement
- III.*C*: Optical Character Recognition (OCR)
- IV: Test Methods and Results
- V: Possibilities for Further Development
- VI: Conclusion
- Appendix A: Report Authorship
- Appendix B: Progress Reports

Matt Litchfield:

- III.*A*: Work Breakdown
- III.*D*: Image Capture
- III.*E*: ATS/OCR Interface
- III.*F*: Similarity Calculation
- III.*G*: User Input

Chengjie Lin:

- III.*H*: XY Plotter
- III.*I*: Stylus Control
- III.*J*: Log File Generation
- III.*K*: Overall Logic

APPENDIX B: PROGRESS REPORTS

The following pages contain the progress reports given in meetings the team held with its mentor, Dr. Scheets, and with representatives from Tietronix Software Inc. Early in the semester, these reports took the form of simple meeting notes and were used only for the team's benefit. Later, in response to a request from Tietronix, they were changed to more of a progress report format, describing what was accomplished each week, what the goal for the next week was, and what roadblocks were encountered each week. These reports were used to keep Tietronix up to date on the team's progress as well as to help the team stay organized and set goals. They are reproduced here in their original format, ordered by date.

Mentor Meeting 1/21/16

Meeting Goals

- Obtain a detailed description of the project from Tietronix
- Determine specifications for the project. What does Tietronix want to have delivered by the end of the semester?
- Check specifications with Dr. Scheets. Does he feel that they meet the requirements of the course?
- Determine a weekly meeting time.

Meeting Notes

Project Description:

Primary objective: text recognition.
Read a text file, read a string from it, find string in the tablet screen. 9 different languages need to be supported. Need to verify whether a string exists on the screen.

We will be provided a working baseline. We can choose to use last semester's code or start from tietronix's baseline. Last year's code will be available to us. Design team's code has some glitches, but tietronix's is mostly glitch-free right now.

Specs will be provided hopefully by close of business Friday (tomorrow).

Step-by-step example: text file is opened. String is picked out. Text is "program patient". Camera moves to center of tablet and takes screenshot. Analyze screen, look for string. If not found, test case fails. If so, case passes.

Second example: same string. Plotter presses a button saying "interrogate patient." Data is retrieved and displayed. Select option to change output current. This activates program patient button. Identify string and pass or fail.

Goals Before Next Meeting

- Read last year's proposal and feature document.
- Find test system and start experimenting.
- Create proposal presentation.
- Meet Dr. Scheets between 10:30am – 3:30pm on Monday to show work completed over the weekend.

Mentor Meeting 1/25/16

Meeting Goals

- Show Dr. Scheets our draft of the proposal presentation; discuss potential changes.
- Discuss Tietronix's specifications and potential changes with Dr. Scheets.

Meeting Notes

- Proposal presentation looks good overall
- Some of Tietronix's specifications do not fit the nature of the class very well
    - Remove Tietronix specs 9 and 10 due to difficulty of testing them
    - Change Tietronix spec 11 to specify simply a general type of room lighting rather than a specific percent variation.
- Some specs should be added to narrow our scope a little
    - Add spec 11 detailing types of strings that should be recognizable
    - Add spec 13 to detail supported operating system/s
- For clarification, add the phrase "at the location of the string" to spec 4.

Goals Before Next Meeting

- Send Dr. Scheets a copy of presentation for him to read on his own time
- Give proposal presentation

Mentor Meeting 2/4/16

Meeting Goals

- Report on current progress:
    - Ran Makeblock's firmware on ATS. Able to use it to move plotter around.
    - Currently setting up OpenCV environment. Has proven difficult in Windows, so now using Linux.
- Ask about pulse generator interrogation. It hasn't worked for us so far; wand battery is probably dead.
- Discuss Cyberonics app backlighting bug.

Meeting Notes

- Backlight bug should have been fixed in Cyberonics app version 11.0.4.3.
- If we cannot get the backlight setting to work in the app, there is an "Easter egg" to allow setting it outside the app.
- Programming wand should have a fresh battery in it already.
- Power light on programming wand is burnt out.
- Data light on programming wand should blink if wand is operating.
- One of the pulse generators is known to have a dead battery.

Goals Before Next Meeting

- Email Cody to set up file transfer for ATS baseline
- Get Tietronix's baseline running on the ATS
- Verify Cyberonics app software version is 11.0.4.3 and configuration is 2.0-OUS

Mentor Meeting 2/11/16

Meeting Goals

- Progress report for Tietronix and Cyberonics.
- Baseline is running successfully?
- Programming wand works now after replacing the battery. We have successfully communicated with the two working pulse generators.
- Can now compile with OpenCV and have begun experimenting with image enhancement.
- Still cannot change backlight, despite verifying app version is 11.0.4.3 with configuration 2.0-OUS. Can we be told how to get outside the app to change brightness that way?

Meeting Notes

- Baseline is not running successfully after all. It appears we misunderstood what exactly Tietronix wanted us to get running.
- Tietronix baseline is the netbeans project from bitbucket.
- To change the backlight on the tablet, one must first exit out of the app.
    - There are two invisible buttons on either side of the battery status bar on the User Preferences page.
    - To exit the app, press the buttons in this order: left 3 times, right 1 time, left 2 times, right 2 times, left 1 time, right 3 times.
    - If you make a mistake, you must leave the User Preferences page, then come back and try again.

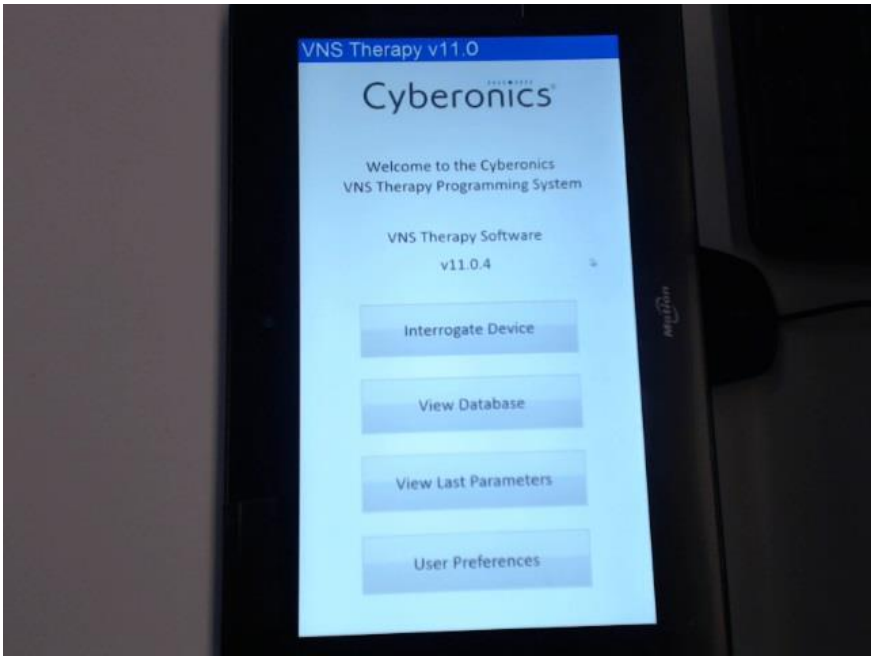
Goals Before Next Meeting

- Get the Tietronix baseline running (netbeans project)

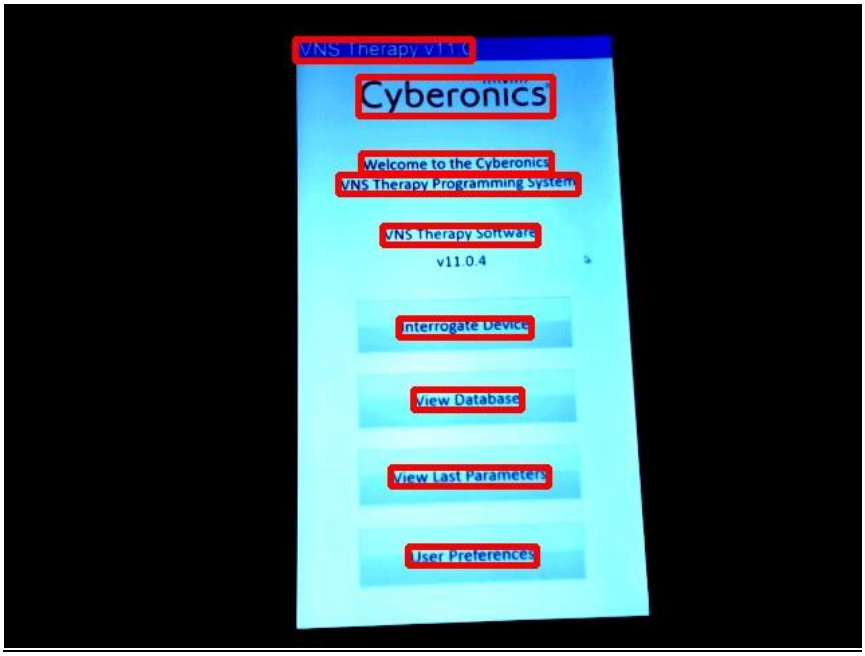
Progress Report 2/18/16

Accomplishments Since Last Report

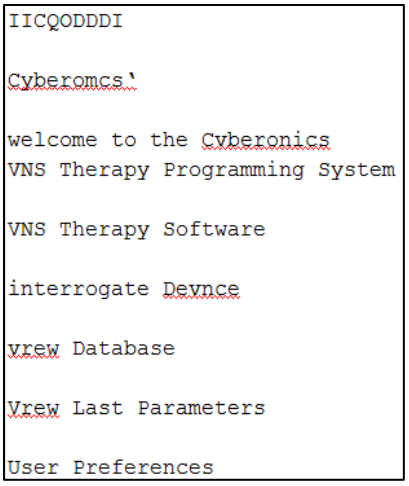
- The team has successfully built and run the NetBeans project containing the Tietronix ATS baseline software. We were able to use the baseline to communicate with the ATS plotter and create and run a simple test case.
- We have made progress with OpenCV image enhancement. Text can currently be recognized from the app's home screen with approximately 81.7% accuracy. There are two main reasons which drive this number down. The first is that some blocks of text are not recognized at all by the algorithm, so they are completely deleted in the process of enhancing the image. This occurs before the Optical Character Recognition (OCR) phase even begins. The second is that the algorithm currently only works well with dark text on a light background, but the app's home screen has a line of light text on a dark background. If only the blocks of dark text which were successfully recognized by the algorithm which determines where text appears are considered, then the accuracy is approximately 96.6%. The greatest improvement, then, can be achieved by improving the way in which the algorithm determines the regions of the screen which contain text and by adding support for light text on a dark background. Note that the accuracy numbers are very rough estimates, calculated using the simple formula accuracy = (mistaken characters) / (total correct characters).



App home screen test image.



Result of the algorithm which determines the regions of the screen to enhance for the OCR phase.



Result of the OCR phase.

Goals for Next Report

- Examine the software baselines from both Tietronix and the Fall 2015 Capstone Design team to determine which we would like to ultimately integrate our software into. This will affect which language we need to use, so although it is not necessary to decide before our prototyping presentation, we would like to decide as soon as possible to avoid redoing work later on.
- Continue to improve the accuracy of the image enhancement algorithm. The focus will be on improving the way the algorithm determines which regions of the screen to enhance prior to the OCR phase.

Roadblocks

- There are no roadblocks that we are aware of at the moment.

OSU Capstone Design ATS Project
Progress Report
February 25, 2016

Accomplishments Since Last Report

- The team has decided we will integrate our code into the Java baseline from Tietronix. Because we had to start prototyping before this decision was reached, our prototypes are currently coded in C++. We will therefore continue using C++ for the next week or two until our prototyping demonstration is complete, then as part of the integration phase we will convert our code to Java for compatibility with the baseline.
- The image enhancement algorithm has been improved. The algorithm now produces much higher quality isolated text images than it did last week. Because of this improvement, the camera resolution is much less likely to be a problem than we were afraid it might be last week.

Goals for Next Week

- We will make a presentation for our prototype presentation. The presentation will be given on March 3. Note that, because this presentation is scheduled for 3:30 to 6:30, we will not be able to hold our weekly Skype meeting next week. We will resume the following week.

Roadblocks

- The tablet is not recognizing taps from the stylus currently mounted on the ATS plotter. We believe this can

be solved by replacing the stylus with a different one.
- We are still unable to change the brightness of the tablet. As a result, we have decided to remove the specification concerning tablet brightness from our spec sheet.

OSU Capstone Design ATS Project
Progress Report
March 10, 2016

Accomplishments Since Last Report

- We gave a presentation in our Capstone Design course about where each team member stands currently on our individual prototypes. This presentation will be included in the email along with this report. This marks the end of the prototyping phase of the project and the beginning of the integration phase. We will be combining our subsystems together into a final product over the next several weeks.
- We have decided not to order a higher resolution camera. After doing some testing, we believe we will be able to meet all of our project specifications with the current camera.

Goals for Next Meeting

- We will not be holding a meeting next week (March 17) because this is during OSU's spring break. We will resume meeting in two weeks, on March 24.
- We will begin integrating our subsystems together between now and the next meeting. Because we will be going separate ways over spring break and the integration phase requires close collaboration, we do not intend to have two weeks' worth of work done by the next meeting. We do, however, plan to have at least some of the subsystems working together.

Roadblocks

- There are no roadblocks that we are aware of at the moment.

OSU Capstone Design ATS Project
Progress Report
March 24, 2016

Accomplishments Since Last Report

- We have begun making the necessary modifications to the code for our individual subsystems in preparation for combining the subsystems together. This primarily consists of removing extra logic used to allow the subsystems to function as stand-alone programs for prototype testing.

Goals for Next Meeting

- We plan to have Matt's subsystems integrated with Ben's subsystems. These systems include user input, image acquisition, ATS/OCR interface, string similarity calculation, image enhancement, and optical character recognition. These systems can fully function together without adding Chengjie's systems, so this is where we will begin. We intend to have them fully working together by the next meeting.

Roadblocks

- There are no roadblocks that we are aware of at the moment.

OSU Capstone Design ATS Project
Progress Report
March 31, 2016

Accomplishments Since Last Report

- We have integrated the following components of the program:
    - Image Acquisition
    - ATS/OCR Interface
    - Image Enhancement
    - Optical Character Recognition
    - Similarity Calculation

  These components can now be run together as a single stand-alone program. Please see the next page for an example test case demonstrating the operation of the program as it exists now.

Goals for Next Meeting

- We will continue integrating this week. The components which still have yet to be integrated are:
    - User Input
    - String Position Calculation
    - Log File Generation
    - Stylus Control
    - Overall Test Logic

  We are not certain yet how many of these components we will be able to integrate by next week. Our goal is to complete as many of them as possible.

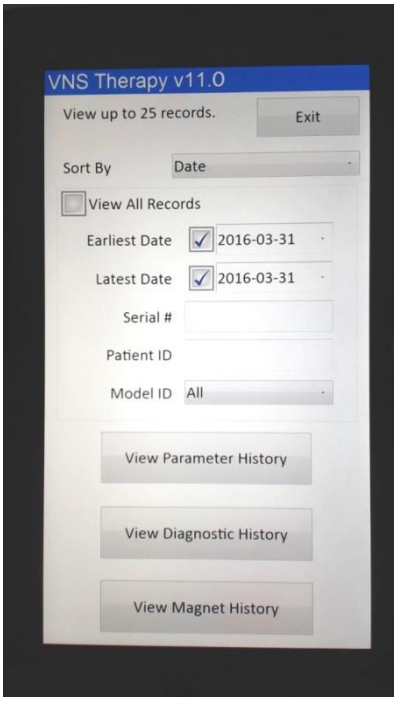
Roadblocks

- There are no roadblocks that we are aware of at the moment.

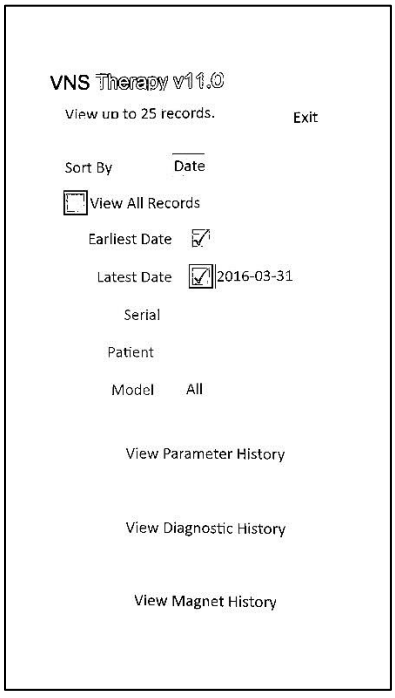
Example Test Case

When the program as it stands now is run, the following sequence of steps is executed. This test case was run on the "View Database" screen in the Cyberonics app as an example case.

1. Capture image from the webcam.



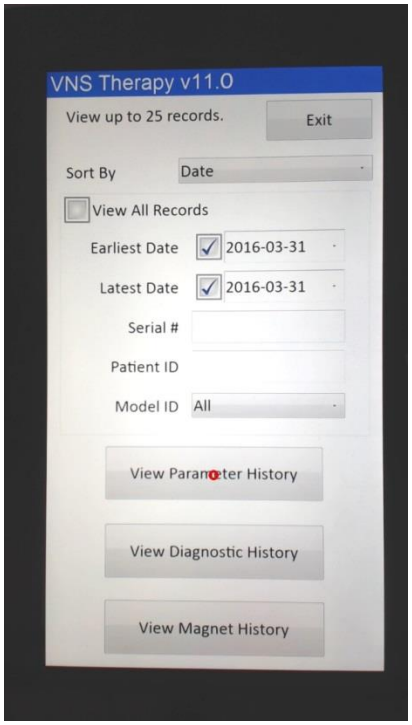
2. Enhance the image by isolating the text. This stage may still be improved if time permits, but it is functioning well as it stands now.



3. Run Tesseract on the image to produce a plain text file. The following image shows the text output from Tesseract. It is not 100% perfect, nor do we expect it to be completely flawless this semester. But it is successfully recognizing the great majority of text.

```
1  VNS  Vflfio@
2
3  View uo to 25 'records. Exit
4
5  Sort By Dfi
6
7  View All Records
8
9  Earliest Date
10
11 Latest Date [2016-03-31
12 Serial
13 Pafient
14
15 Model All
16
17 View Parameter History
18 View Diagnostic History
19
20 View Magnet History
21
22
```

4. Locate a string which satisfies the desired confidence value. For this test, we were searching for the string "View Parameter History" with 85% confidence. The program was able to find this entire string (it can be seen in line 17 of the image above).
5. Use Tesseract's output file to determine where in the image the string was located and draw a circle on the original image in the location where the stylus will eventually tap the image. The image below shows that the program successfully found the string's location.



OSU Capstone Design ATS Project
Progress Report
April 7, 2016

Accomplishments Since Last Report

- We have integrated the following components of the project:
    - Overall Test Logic
    - String Position Calculation
    - Stylus Control

  These components have been added to the integrated components we listed last week.
- We have successfully tested navigating through multiple screens with the string recognition module.

Goals for Next Meeting

- We will add the final components that have yet to be integrated:
    - Log File Generation
    - User Input
- We will be meeting with Chris in person next Thursday, April 14. We will demonstrate the project's operation here at OSU. Note that this means we will not need to have our usual Skype meeting next week.

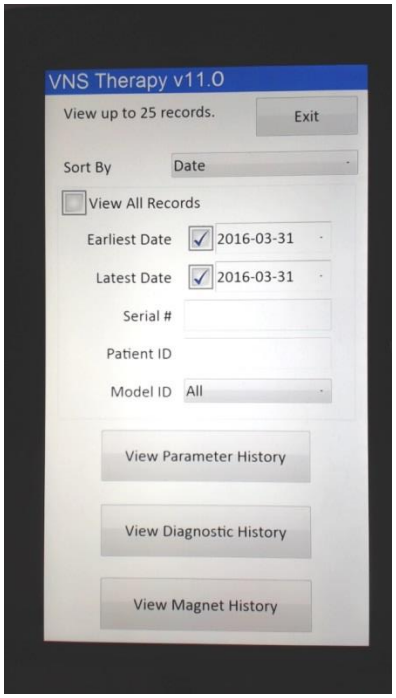
Roadblocks

- We have come across an issue that really is not a roadblock for us, but it is something you should be aware of. Occasionally the blue highlighting of a button can flicker continuously after the tablet is turned on. Navigating to a different screen and coming back does not stop the button flickering. If this occurs, the string recognition will have trouble recognizing the text on the flickering button. The flickering can, however, be stopped by restarting the tablet. So this does not impede our ability to test using the tablet, but it may cause future problems when someone tries to use the string recognition module for an automated test and is not aware the flickering is occurring.

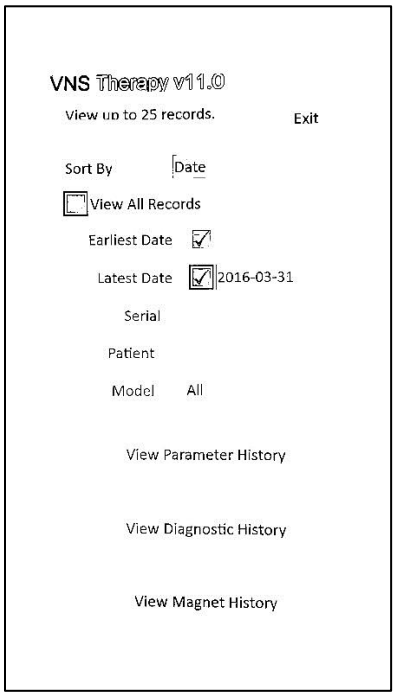
Example Test Case

When the program as it stands now is run, the following sequence of steps is executed. This test case was run on the "View Database" screen in the Cyberonics app as an example case.

6. Capture image from the webcam.



7. Enhance the image by isolating the text. This stage may still be improved if time permits, but it is functioning well as it stands now.



8. Run Tesseract on the image to produce a plain text file. The following image shows the text output from Tesseract. It is not 100% perfect, nor do we expect it to be completely flawless this semester. But it is successfully recognizing the great majority of text.

```
1  VNS  Vflfio@
2
3  View uo to 25 'records. Exit
4
5  Sort By Dfi
6
7  View All Records
8
9  Earliest Date
10
11 Latest Date [2016-03-31
12 Serial
13 Pafient
14
15 Model All
16
17 View Parameter History
18 View Diagnostic History
19
20 View Magnet History
21
22
```

9. Locate a string which satisfies the desired confidence value. For this test, we were searching for the string "View Parameter History" with 85% confidence. The program was able to find this entire string (it can be seen in line 17 of the image above).
10. Use Tesseract's output file to determine where in the image the string was located and draw a circle on the original image in the location where the stylus will eventually tap the image. The image below shows that the program successfully found the string's location.