EEE174 –CpE185 Introduction to Microprocessors

LAB 6 – Tesseract Library

Lab Session: Wednesday 6:30PM - 9:10PM

Section 32385

Lab Instructor: Sean Kennedy

Student Name: Andrew Robertson

TABLE OF CONTENTS

Part 1

Overview

This lab is part 1 of 5 for the series of do it yourself labs geared towards helping with the final project. Since my group involves OCR (Optical Character Recognition), the Tesseract library for Linux / Windows/ Mac is a good place to start. This library is built entirely around processing a saved image into text.

Lab Discussion

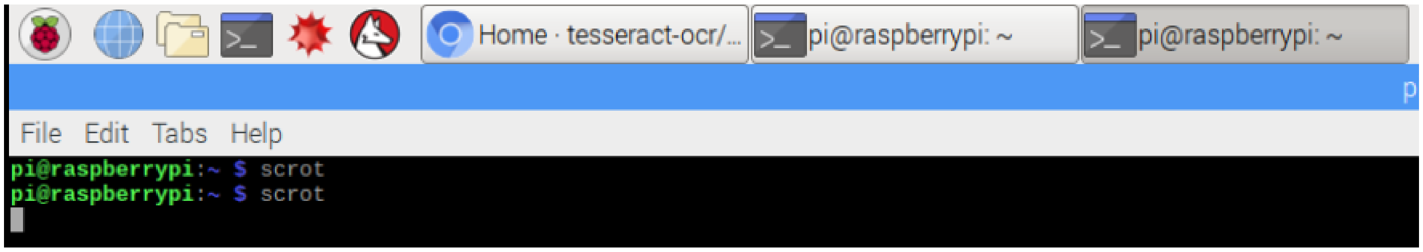
Work Performed / Solution:

The first thing I needed to do before I could start using Tesseracts terminal commands was to install the library. Linux make this very easy and provides a few ways to do so. The three ways I know of to install a package or library is: 1st – using your distributions app store; 2nd – terminal command; 3rd – downloading from a source and using the package manager. I chose option 2 because both this option and the first take care of all dependency and file version tasks for you. I would normally use option 1 but since I could not find this feature in Raspbian I could not do so. This is all I needed to type to have all aspects of installation handled for me:

```
pi@raspberrypi:~ $ sudo apt-get install tesseract-ocr
```

Test 1

After installation, nothing else needed to be done to start using this library's functions. Not knowing where to start, I decided to take a screen shot of the desktop to feed to Tesseract.

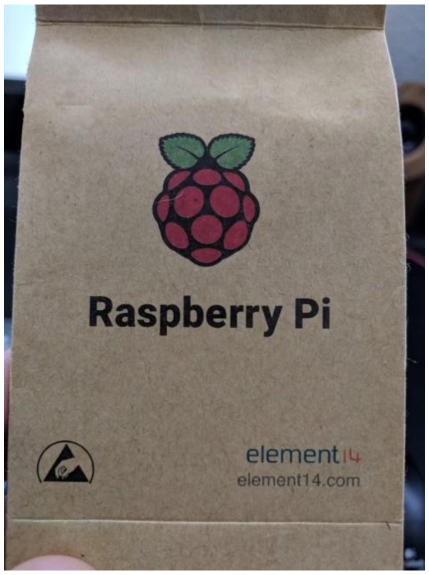


```
pi@raspberrypi:~ $ scrot
pi@raspberrypi:~ $ scrot
```

I've cropped a decent amount of this so it is legible, the rest of the display was essentially just black space. As can be seen in figure 1, the output was abysmal. The letter "i" was interpreted as a parenthesis twice, the entire top section was ignored, $ became an s, and scrot is scrul. I thought some of this would be due to small font, perhaps style of font, contrast between background and foreground and so on so I began a series of tests.

Test 2

I wondered how much of some plain packaging would be able to be processed and used a photo of the Raspberry Pi lunch bag packaging as can be seen to the right. Noting figure 2, the result from this was also terrible. I decided to step away from images that have symbols and figures in them to something like we would be working with in our project, plain text.

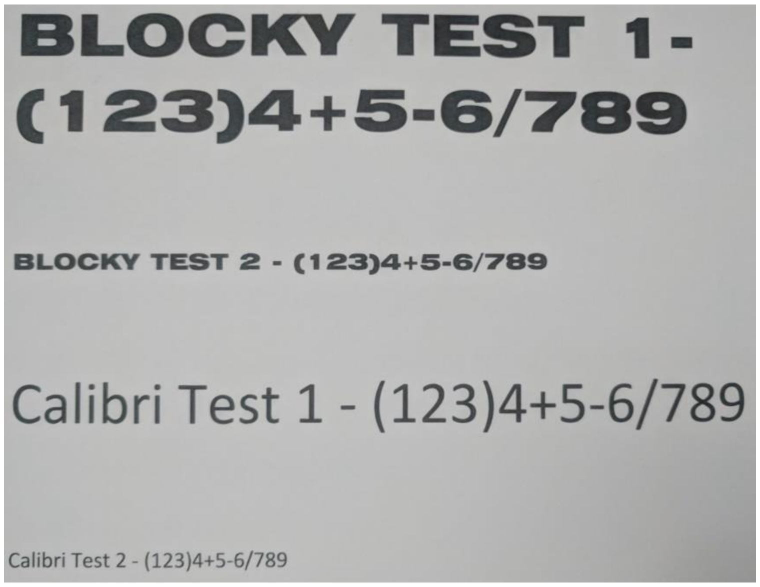


Test 3

I again used the packaging as seen to the right but cropped the image down to only the Raspberry Pi letters in the center of the image and finally received output that can be considered usable, as seen in figure 3.

Test 4

At this point it was pretty obvious that the quality of the input greatly affected the quality of the output so I began testing different fonts and sizes to see what results I would get. When I first researched a good OCR library for Linux I saw that a notable setback for Tesseract is blocky fonts like those you would find on a credit card so here I tried both a blocky font and a very round font, both of large and medium size fonts.



BLOCKY TEST 1-
(123)4+5-6/789

BLOCKY TEST 2 - (123)4+5-6/789

Calibri Test 1 - (123)4+5-6/789

Calibri Test 2 - (123)4+5-6/789

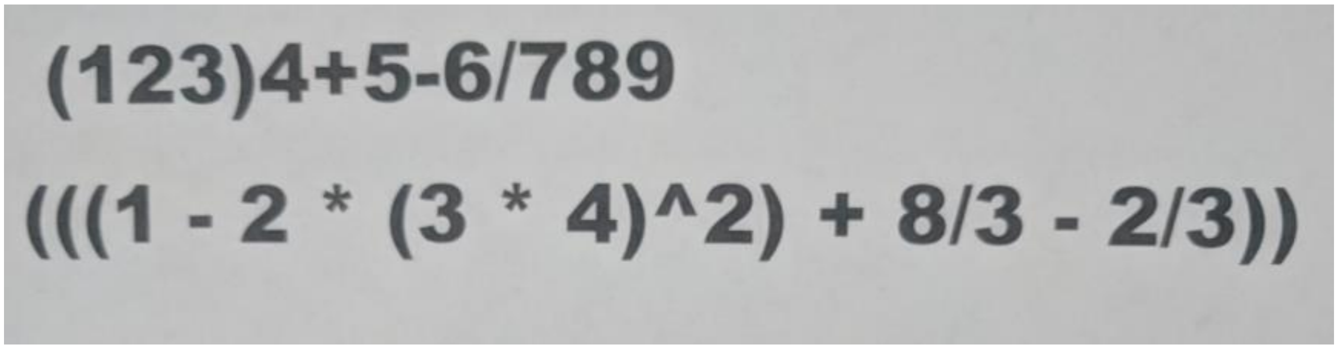
I created the document to the right in Microsoft Word, printed it out, took a photo of it, and used that photo for input. Figure 4 shows the OCR result.

From this I gathered:

- The abnormal space around the blocky fonts 1 is interpreted as a space key
- Dashes in the larger Calibri font are interpreted as long dashes
- The Blocky Y’s look like V’s so not really a fault of Tesseract
- The Blocky numbers and symbols were flawlessly interpreted

Test 5

Knowing we were going to be interpreting arithmetic, I decided to no longer use text, and only use numbers / symbols relevant to this cause in a font sort of in between the blocky one and Calibri called Arial black. I also wanted to see if white space made any significant differences.



(123)4+5-6/789

(((1 - 2 * (3 * 4)^2) + 8/3 - 2/3))

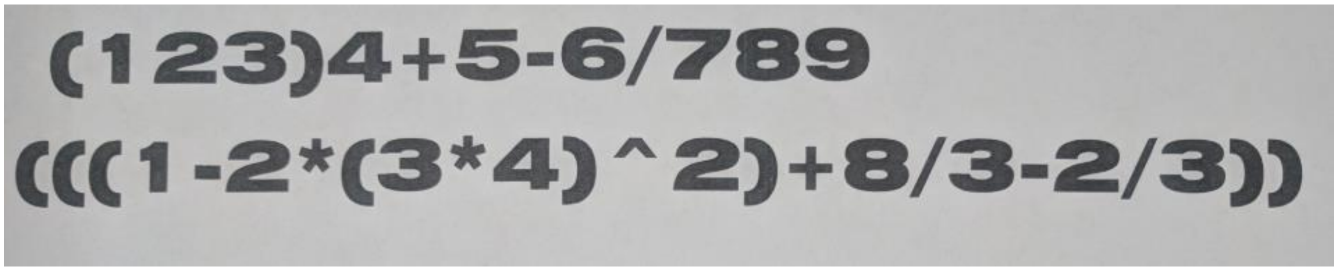
Again, this was printed and then a photo was taken of the print. The results could be seen in figure 5. At this point I wasn’t sure what happened to the exponential sign but did notice minus signs disappear with the added whitespace.

Test 6

Given the previous results I then removed the whitespace and tried again, this time twice. Once under normal lighting, and once under very dark lighting. Figure 6 shows the results for normal lighting and 7 the dark lighting. First thing I noticed is performance detecting the minus symbols is much improved without the added whitespace. The second difference was a surprise to me, the darker image was more accurate. In the lighter image, a slash was misinterpreted as a 1, but in the darker image this did not happen.

Test 7

Last, I wanted to test two more scenarios. Given the blocky fonts perfect record for number and symbol recognition (ignoring white space) I wanted to return to this font and try again. I also wanted to see if .jpg images worked too. The library information I used under listing files only mentioned .png but .jpg is used more frequently when taking photos. I fed Tesseract the same image as both a .jpg and .png using blocky font and was again surprised.


(123)4+5-6/789
(((1-2*(3*4)^2)+8/3-2/3))

I expected the .jpg not to work since it wasn't mentioned but the results from both files were identical (figure 8). Unfortunately, many of the left parenthesis became letter c.

Listing Files(s):

Tesseract resource: https://github.com/tesseract-ocr/tesseract/wiki

Figure 1

```
p)@raspherryp - s scrul
p)@raspherryp s scrul
```

Figure 2

```
{
```

Figure 3

```
Raspberry Pi
```

Figure 4

```
BLOCKV TEST 1 -
( 1 23)4+5-6/789

BLOCKV TEST 2 - (1 23)4+5-6/789

Calibri Test 1 â€” (123)4+5â€”6/789

Calibri Test 2 - (123)4+S-6/789
```

Figure 5

```
(123)4+5-6/789
(((1 2 (3 * 4)"2) + 8/3 2/3))
```

Figure 6

```
(123)4+5-6l789
(((1-2*(3*4)"2)+8/3-2/3))
```

Figure 7

```
(123)4+5-6/789
(((1-2*(3*4)"2)+8/3-2/3))
```

Figure 8

```
c 1 2334+5-6/739
(cc 1 -2*(3*4) A 2) + 8/3-2/3))
```

Conclusion

The font forewarning was not ill-spoken. I am certainly on the right track for finding a consistent font, but I fear it will be mostly a trial and error process. Font size did not seem to have a tremendous effect once reasonably large. This library is insanely easy to use and isn’t too slow, but we do need to somehow give it about a 20 second or so buffer just to be sure