Raspberry Pi Mathematician: A Mobile Optical Character Recognition System

John Knowles

A Senior Project

Department of Physics
California Polytechnic State University - San Luis Obispo
10/30/16

CONTENTS

Raspberry Pi Mathematician: A Mobile Optical Character Recognition System

John Knowles

***Abstract*—Recorded Notes of the process, findings, and development of making a simple device that reads handwritten addition problems via photoelectric devices. While Tesseract is an OCR platform designed for printed texts, this project focuses on reading handwritten text and symbols. Results using Tesseract were found to be acceptable in comparison to translating printed texts. Results were found to have at least a 90% rate of accuracy in comparison to the true character values.**

This paper also discusses the technology and development of augmented reality and virtual reality in head-mounted displays. The conception of this project is based on creating an assistive device capable of allowing deaf or hard of hearing individuals the opportunity to communicate in otherwise difficult situations. This includes group conversations, noisy environments, etc.

***Index Terms*—Virtual and augmented reality, smart clothing, microdisplay, optical head-mounted display, wearable technology, OpenCV, Tesseract, ImageMagick, Python, Raspberry Pi**

Introduction

Optical character recognition (OCR) is a method of automatic character identification that is readable by both humans and machines. It deals with the problem of recognizing optically processed characters produced by typeface or handwriting. The quality of the input document, number of constraints, and whether or not the input characters are printed or handwritten affects the translation success rate. OCR is unique in comparison to other automatic recognition processes such as bar code, magnetic strip, RF, vision system, or even speech recognition translations in the respect that it does not need to have control of the process that creates the input information. OCR consists of the following stages: optical scanning, location segmentation, preprocessing, character extraction, and a post-processing stage. [1], [2], [3]

I. Project Scope

There are a few solutions for OCR on the market, Tesseract being the most popular as it leads commercial engines in terms of accuracy [3]. While it is designed more for printed texts, it is feasible to obtain accurate results with handwritten texts. In order to do so, certain requirements must be met (see Requirements). Other OCR solutions include: ABBYY FineReader, OmniPage, TypeReader, and many other derivatives of those listed. Tesseract was chosen due to its popularity, well-documented support, and portability. The goal of this project is to translate basic, handwritten math problems (addition, subtraction, multiplication, and division) using a white-on-black writing medium such as a 3x5 flash card centered on a black paper. The writing utensil can be any sized point, felt or ink, although better results can be achieved with a tip that yields a fine trace.

Advisors: Professor Tom Bensky tbensky@calpoly.edu, Professor Matt Moelter mmoelter@calpoly.edu

II. Technical Details

A. Hardware

This project utilizes a laptop, spare sunglasses with the lens popped out, a glue gun, Raspberry Pi 2 Model B, the Adafruit Picamera, a 32GB microSD card, a wireless mouse and keyboard, and the necessary cables for functionality. A black permanent marker with a felt tip and 3x5 cards were used to produce simple addition problems. Training data was obtained using a 600 dpi scanning bed.

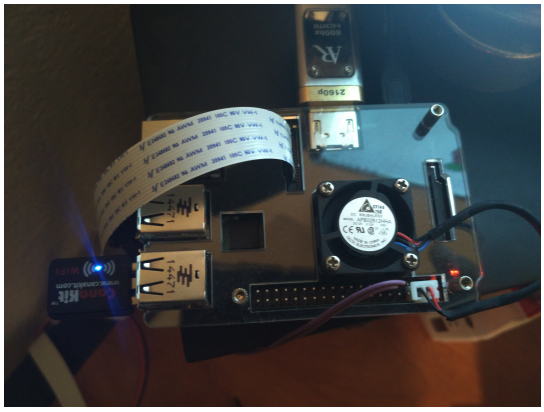

Fig. 1: Raspberry Pi 2 Model B with an acrylic case, 32GB microSD card, wireless keyboard receiver, wireless dongle, and on-board fan.

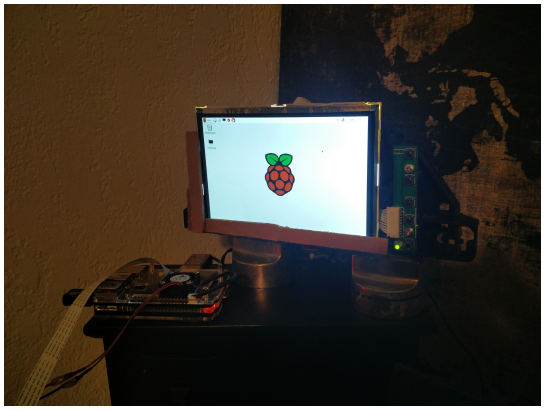

Fig. 2: Adafruit Monitor on a custom stand built from recycled TV parts and cardboard.

B. Software

Windows 10, Tesseract, jTessBoxEditor, OpenCV, ImageMagick, Mathematica, MobaXTerm, Raspbian, Python,

Fig. 3: Salvaged sunglasses.The Picamera was easily set in place with a glue gun.

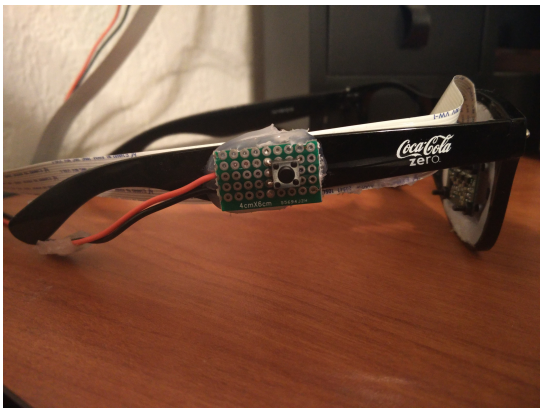

Fig. 4: Button functionality on the side of the frame. Three modes of usage: Single click takes a picture, Three second hold kills the application, and Five+ seconds will shut the Raspberry Pi down altogether.

bash scripts, iPython Notebook/Jupyter Notebook, and C++ were all utilized in the completion of this project. The full program python code can be found in Appendix A.

III. Image Processing

As previously mentioned, image processing was needed in order to improve overall results. Training data (see Training Data) needed to be scanned as a 600 dpi monochrome tiff image in order to preserve the quality of the image. From there, each element was associated with its respective character using jTessBoxEditor and then compiled all together as a library for Tesseract. In addition, each image capture needed to be converted into a monochromatic color scale (black and white), cropped, cleaned up of all residual artifacts, and rotated before Tesseract could interpret the experimental data [4], [5], [6], [7], [8], [9] (see Appendix D and Appendix F for respective image preprocessing and processing code.)

IV. Initial Trial

The project apparatus consisted of a Raspberry Pi 2 Model B loaded with raspbian and connected to the picamera via the dedicated on-board connection. This stage of the project was not yet utilizing a camera mounted on a pair of sunglasses; rather, it was situated on a small tripod aimed at 3x5 flash cards which were positioned at a fixed distance from the camera. The capture quality would degenerate if the camera was too close or too far from the handwritten cards. If the capture was too close, the image would blur, introducing a multitude of leftover artifacts during the image-cleaning process. Results for whole equations using each of these arithmetic functions were found to be horribly inaccurate, frequently producing results that did not agree with the true data at all.

V. Training Data

In order to obtain better readings, training data was created using jTessBoxEditor, a java applet that boxes individual characters with its true symbol and then compiles everything into a data file Tesseract can read [10]. jTessBoxEditor accepts scanned monochromatic TIFs at 600 dpi as shown in Figure 5.

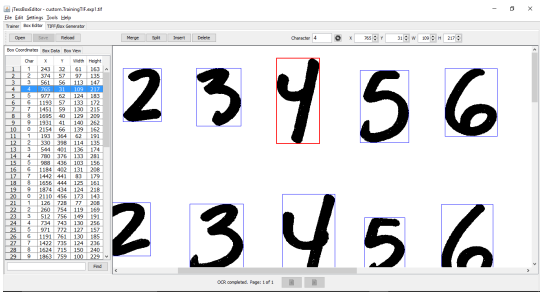

Fig. 5: jTessBoxEditor's character boxing functionality. Each symbol is enclosed as a unitary character associated with its true value.

Numerical symbols were drawn onto a clean 8x11 sheet of paper with a permanent felt marker. Characters were drawn with very little variation in order to encourage better character identification. This was purely an educated guess about how Tesseract's learning algorithm may read data. Attempts to somewhat-to-moderately vary characters during a future training process may occur just to see how it affects the results.

Examples of handwritten numerical digits are shown below in Figure 6.

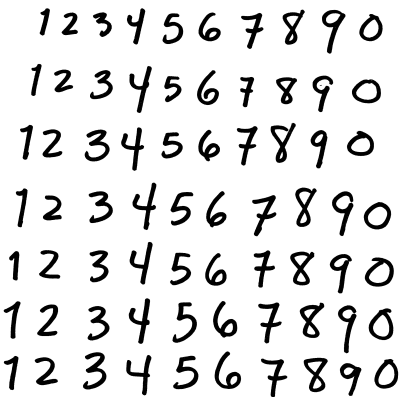

Fig. 6: Custom numerical training data. Note the lack of extreme variation in handwriting with the intent of preventing Tesseract from getting confused.

Initially, this project was meant to translate all forms of math (which was quickly reduced to basic arithmetic functions: addition, subtraction, division, and multiplication.) Granted, the initial scope of this project was ambitious; nevertheless, little by little, new mathematical operations will be introduced once the general process is established.

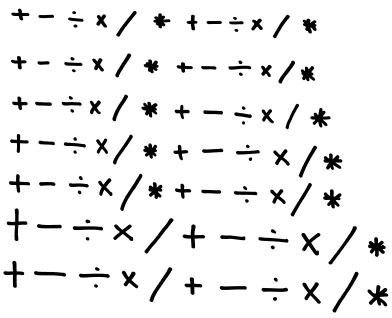

Fig. 7: Custom arithmetic training data.

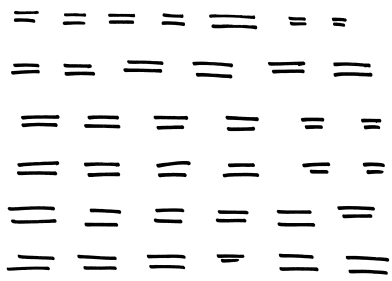

Fig. 8: Custom equality training data.

OCR results from the first generation training data (FGTD) improved overall accuracy; however, the second generation training data (SGTD) yielded more inaccuracies compared to the first. This was due to expanding the possible arithmetic functionality of the program. Regardless of the negative change, there was experimental agreement with this particular phenomena observed by third-party results. The phenomena is attributed to the idiosyncrasies of Tesseract. [4] It quickly became clear that Tesseract struggled with differentiating the subtraction symbol from the division symbol, 2, 5, 7, and equality.

Dictionary rules were subsequently created to help translate the custom handwriting data (See Appendix I). By providing feasibility rules for handwritten symbols, results improved somewhat respectably, but not within the desired accuracy of 95%+. Tesseract was consistent in reading all digits except for 2, 5, and 7 with at least 98% verity at this point.
Example selection rules are listed (See Appendix I for all of the translation rules):

Examples of preferred output rules

```
v2
22 =2 1
27 =7 1
25 =5 1
72 7= 1
52 5= 1
```

TABLE I: Preferred Output Rule Format

Selection Characters	Target Characters	Type Indicator

The above syntax is described in Table 1.

v2 corresponds to version 2 of the Tesseract selection rule source code. Version 2 was created to better work with python version 3 and to reduce the amount of syntax to create custom selection rules. Type indicator values can either be 0 or 1. 0 is a non-mandatory substitution. This informs Tesseract to consider the ambiguity as a hint to the segmentation search that it should continue working if replacement of 'source' with 'target' creates a dictionary word from a non-dictionary word. Dictionary words that can be turned to another dictionary word via the ambiguity will not be used to train the adaptive classifier. 1 is a mandatory substitution. This informs Tesseract to always replace the matched 'source' with the 'target' strings. [4]

A. *Successes*

If avoiding "danger characters," otherwise known as the following: $2, 5, 7, \div, -$, and $=$, Tesseract would return highly acceptable results.

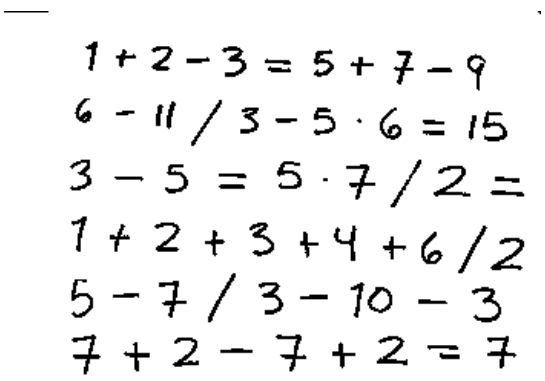

Fig. 9: Six lines of illogical mathematical statements are shown here. This image was evaluated from within jTessBoxEditor.

Results from within jTessBoxEditor:

```
1-2-3=5+7-9
6-11/3-5-6=15
3-5--5-7/2-2
1+2+3+4+6/2
5-7/3-10-3
7+2-7+2-7
```

The above results show minor discrepancies at the training level. These results are used as a method of validating training data. Since each image capture undergoes image processing that is quite different from a clean scan, Tesseract's final output will differ from the training validation results.

Lines 2, 3, and 6 show errors with the "danger characters," as expected. This won't matter as the project is limited to

addition problems in the project's next phase (see section Secondary Trial Results.)

B. Inaccuracies

Certain problems yielded higher inaccuracies. Given an example problem of $6 - 2 =$, Tesseract returned similarly inaccurate translations such as $6 - -- =$ or $6 - 2- =$. However, $6 + 3 =$ would likely return $6 + 3 - -$ as the error statement, but more likely achieve 100% verity. Creating selection rules for these inaccurate translations cleaned up a lot of these mistakes.

VI. Secondary Trial Results

At this point, the camera was mounted onto a pair of sunglasses with the use of a glue gun as shown above. Because of the inaccuracies regarding the "danger characters," arithmetic problems were reduced to simple arithmetic problems without an equality symbol. Any input would imply an equation to be solved. New focusing challenges arose from using the sunglasses as the mounting mechanism. OpenCV's ROI cropping functionality proved useful in eliminating random artifacts outside the ROI.

A. Requirements

OpenCV was introduced in order to improve overall results by utilizing its region of interest (ROI) cropping algorithm (See Appendix E for OpenCV code). This was crucial in eliminating a multitude of new issues that arose with switching the camera mounting mechanism from a fixed stand to mobile glasses. OpenCV reads the image file, converts it into grayscale, finds the threshold values and applies contours based on those values. Sections of the original image are removed until all that remains are the contours with the lightest shades. This assumes that any input math is done on 3x5 flash cards or printer paper and that the background surrounding the paper is darker in color. To facilitate this effect with good frequency, I lowered the brightness of the camera to an optimized value of 50. This proved very effective in eliminating unwanted sections in a capture.

OpenCV returned coordinates for the ROI in x_{max}, x_{min}, y_{max}, y_{min} format, but needed to be in $Crop_{Area} + x_{offset} + y_{offset}$ form so that ImageMagick could work with it (See Appendix F), which was an easy fix. $Crop_{Area} = (x_{max} - x_{min}) * (y_{max} - y_{min})$ and the offsets were just the minimal axis values. At this point, I had a functional program. Minor code tweaking and cleanup occurred at this point.

B. Processing

Very little code change was implemented in response to OpenCV, so the image capture processing stage remained the same. OpenCV provided the mechanism for determining the region of interest coordinates, which were used for the purposes of cropping out unnecessary background noise. Commented lines of the image processing function code were image processing algorithms that were no longer needed to clean up the image. Turns out that less is better, sometimes.

A basic calculator function was implemented to demonstrate return feedback. The calculator does not factor in order of operations in the input strings, nor does it account for non-arithmetic problems.

The use of regular expression manipulation (See Appendix G) replaced a large portion of code dealing with the filtering and sorting of OCR results. This was used in order to clean up any unnecessary artifacts or illogical statements in Tesseract's return strings. A very well constructed online regex tool can be found at [11] with complementary information in the form of a cheatsheet located at [12].

C. Successes

The overall tweaks to the main code resulted in better accuracy in the derived outputs. Overall complexities to the code were increased with the development of the program, and more functionality was realized, and in some cases, better performance.

D. Inaccuracies

Inaccuracies in the Tesseract output string were reduced significantly, albeit with external help. Each output was made to adhere to a desired form such that the string equations were usable for calculations.

Calculation of the input strings are very rudimentary; there are no accounting for complexities, order of operations, and the regular expression manipulation may in fact change the desired input when dealing with negative numbers.

E. Limitations

Limitations were found in terms of the speed of RPi hardware, the accuracy of Tesseract's OCR algorithms in handwriting script, time constraints, and in novice programming. Ideally, better threading techniques would be employed in a faster platform. A Shunting Yard algorithm to handle input characters was considered briefly; however, order of operation handling was figured to be better left to future implementation.

Tesseract's usefulness primarily resides in typeface scripts, so handwritten scripts proved to be much more challenging to work with. [3] Tesseract's OCR limitations were identified in several ways. Of these, characters must be isolated from adjacent characters such that it can be encapsulated by a "box", characters must be free of flourishes and unconnected lines, relative character sizes must be similar, scanned images must be black and white 300+ DPI with artifacts removed during preprocessing (or Image Processing) [13], [3], [9].

VII. Results

Results were obtained using a training set from a single handwriting source. To understand the performance of the techniques used in this project, the following expression was utilized.

$$\text{OCR Accuracy} = \frac{\text{C}_{\text{Exact}}}{\text{C}_{\text{Inexact}} + \text{C}_{\text{Unsegmented}}}$$

TABLE II: Classification Rules

	Total	Exact	Inexact	Unsegmented	Success Rate
Trial # 1	7	7	0	0	100
Trial # 2	21	20	1	0	95.2
Trial # 3	25	24	1	0	96
Totals	53	51	2	0	97.1

The classification results are derived only from the secondary trial results. Errors due to Tesseract's inability to segment digit samples into boxes are ignored since they do not contribute to the recognition efficiency [6], [7], [8].

VIII. Analysis

Despite the successes of this project, this project would still fair better in developing a better OCR platform that utilized a handwriting script database such as IAM or CEDAR in conjunction with OpenCV. A typical read would take about 10 seconds to process, which would not be ideal for mobile applications. This is largely due to the RPi hardware limitations. Mobile phone platforms may perform on a better capacity.

Future Implementations - The "What Next?"

IX. Near-To-Eye Display

A. Introduction

A head-mounted display usually comes in one or two lenses, depending on the application, using liquid crystal displays, liquid crystal on silicon, or organic light emitting diodes as the chief, miniaturized display technology. Aviation and tactical, engineering, medical, training, simulation, gaming, video and research applications can be realized from the use of this technology.

B. Virtual Reality

Virtual Reality (VR) is a virtual environment modeled by computer simulations to recreate the effects of physical reality. It covers a wide variety of applications, such as immersive computer simulations and 3D environments and the development of CAD software, head-mounted displays and so on.

C. Augmented Reality

Augmented Reality (AR) is associated with live views of physical realities that have been altered or augmented in terms of sensory inputs such as sound, video, graphics, or even GPS. AR serves to enhance user perceptions of reality. This is in contrast to VR, which replaces real-world input.

D. Project Forerunner

Project Forerunner is a research and development project aimed at developing an augmented reality head-mounted display capable of translating spoken word into text projected in virtual space. This project's intended purpose is to provide the deaf or hard of hearing with an affordable solution to real-time communication difficulties.

X. Project Scope

The goal of this project is to ultimately design and create a working prototype micro-display and lens system; however, the scope of the current project lies in becoming familiar with the different types of technologies available on the market today. Namely, the goal is to construct a system similar to Google Glass, which uses a near-to-eye display [14], [15], [16].

The principle elements of Google glasses are listed: LCOS display and control board, polarizing beamsplitter cube with a half-silvered mirror/beam splitter housed inside, a polarization conversion system (PCS) composed of a crossed polarization grating and a polymer diffuser, a wedge fly-eye with a cover reflector adjacent to an LED array, and finally a concave reflector.

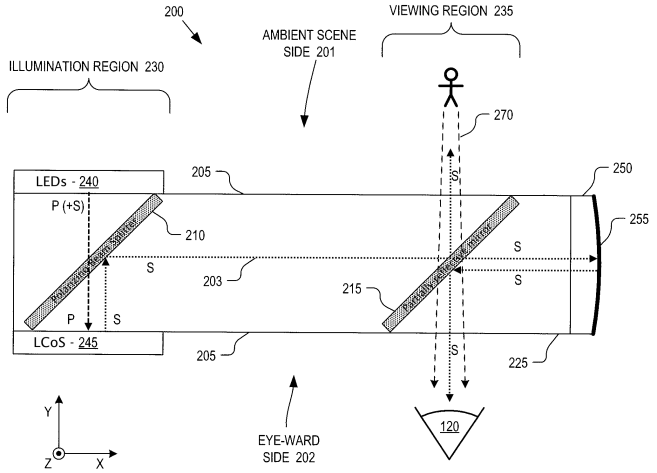

Fig. 10: Google Glass display schematic that also serves as to show general functionality of a micro-display.

A. How Google Glasses Work

Figure 10 shows a general setup for a micro-display. Near eye display applications are lower power systems that use a small RGB LED side-emitting array that feeds light into a wedge-shaped fly-eye homogenizer lens. The wedge fly-eye has a reflective surface (most likely vapor deposited aluminum) that reflects all light back to the fly-eye. Incident illumination from the LED is homogenized by the fly-eye and then pushed through the PCS. The PCS is composed of a polymer reflective polarizing diffuser and a reflective crossed-wire grid based polarizing grating.

Unpolarized light from from the homogenizer/fly-eye reaches the polarizing diffuser, which rotates a small part of the S-input to P randomly. The S-input is the plane of incidence; which includes the propagation direction of the light and the surface normal of the component and is arbitrary for normal incidence. P polarization is defined as the condition where the plane of incidence includes the plane of polarization. The remaining S and P inputs are incident on the reflective polarizing grating which accepts all P inputs but reflects S-inputs back through the polarizing diffuser. The rotating process is repeated as the light passes through the polarizing diffuser and converts

remaining S-inputs to P. The light continues through the fly-eye and hits the reflector to once again repeat the process until the PCS maximally converts LED light to P-polarized.
The PCS output beam has a uniform polarization that matches the LCOS array requirements.

B. Future Plans

XI. Real-time Projected Subtitles

The method of creating real-time subtitles for spoken dialog is a challenging one. As of right now, there are no viable, light-weight solutions in the area of speech recognition that can be utilized straight out of the box. In order to make an effective solution for the deaf or hearing impaired, speech recognition accuracy must be near perfect.

If such a solution were to exist, it would work beautifully with a pair mounted glasses using a near-to-eye projection display. Ideally, subtitles would be uniquely color-coded for individual speakers and spatially projected near the associated speaker.

XII. Bone Transducer

A bone transducer affixed to the frame of the glasses could provide a moderately effective solution for those that are hearing impaired; however, this addition would seem to be more useful for the non-impaired users that find such a piece of technology appealing to use.

XIII. Non-Invasive Data Collection

The biggest concern with Google Glass was that of privacy. A large number of privacy issues arose after the release of Glass, most of which were focused on the video capturing capabilities. As such, a large number of public establishments banned the device from use.

Since this project can be realized without the use of video-saving functions, or perhaps even without the use of cameras altogether, the privacy concerns are largely mitigated. Moreover, privacy concerns could be fully eliminated when realizing that collected data is only used real-time and not saved beyond the scope of that time span.

XIV. Conclusion

This project was not without its challenges in programming; however, it was an idea that needed some exploring since the nature of the project resembled Forerunner. OCR results were determined to be fairly good, the mobility of the project was a success, and the use of Raspberry Pi as the computational unit found reasonable processing times.

References

[1] L. Eikvil, "Optical character recognition," *citeseer. ist. psu. edu/142042. html*, 1993. [Online]. Available: http://www.academia.edu/download/33085443/OCR.pdf

[2] S. Mori, H. Nishida, and H. Yamada, *Optical Character Recognition*, 1st ed. New York, NY, USA: John Wiley & Sons, Inc., 1999.

[3] R. Smith, "An overview of the Tesseract OCR engine," 2007. [Online]. Available: https://research.google.com/pubs/pub33418.html

[4] Google Inc., "tesseract-ocr/tesseract." [Online]. Available: https://github.com/tesseract-ocr/tesseract

[5] C. Patel, A. Patel, and D. Patel, "Optical character recognition by open source OCR tool tesseract: A case study," *International Journal of Computer Applications*, vol. 55, no. 10, 2012. [Online]. Available: http://search.proquest.com/openview/cce8cb91514de142cb85302d5fc1280f/1?pq-origsite=gscholar

[6] S. Rakshit and S. Basu, "Development of a multi-user handwriting recognition system using Tesseract open source OCR engine," *arXiv preprint arXiv:1003.5886*, 2010. [Online]. Available: http://arxiv.org/abs/1003.5886

[7] S. Rakshit, A. Kundu, M. Maity, S. Mandal, S. Sarkar, and S. Basu, "Recognition of handwritten Roman Numerals using Tesseract open source OCR engine," *arXiv preprint arXiv:1003.5898*, 2010. [Online]. Available: http://arxiv.org/abs/1003.5898

[8] S. Rakshit, S. Basu, and H. Ikeda, "Recognition of Handwritten Textual Annotations using Tesseract Open Source OCR Engine for information Just In Time (iJIT)," *arXiv preprint arXiv:1003.5893*, 2010. [Online]. Available: http://arxiv.org/abs/1003.5893

[9] B. Zareba, "How to prepare training files for Tesseract OCR and improve characters recognition?" Jun. 2016. [Online]. Available: https://goo.gl/3wFuUG

[10] Q. Nguyen, "jTessBoxEditor - Tesseract box editor & trainer." [Online]. Available: http://vietocr.sourceforge.net/training.html

[11] F. Dib, "Online regex tester and debugger: PHP, PCRE, Python, Golang and JavaScript." [Online]. Available: https://regex101.com/

[12] J. Hartley, "tartley/python-regex-cheatsheet." [Online]. Available: https://github.com/tartley/python-regex-cheatsheet

[13] A. Mordvintsev and A. K., "Welcome to OpenCV-Python Tutorialss documentation! OpenCV-Python Tutorials 1 documentation." [Online]. Available: https://opencv-python-tutroals.readthedocs.io/en/latest/index.html

[14] S. Dent, "Google Glass patent application shows detailed diagrams." [Online]. Available: https://www.engadget.com/2013/02/21/google-glass-patent-application-diagramsi/

[15] R. Furlan, "Build Your Own Google Glass," Dec. 2012. [Online]. Available: http://spectrum.ieee.org/geek-life/hands-on/build-your-own-google-glass

[16] Google Inc., "Google Glass," Oct. 2016, page Version ID: 744048276. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Google_Glass&oldid=744048276

[17] J. M. White and G. D. Rohrer, "Image thresholding for optical character recognition and other applications requiring character image extraction," *IBM Journal of research and development*, vol. 27, no. 4, pp. 400–411, 1983. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5390437

[18] A. News, "What Went Wrong With Google Glass," Jan. 2015. [Online]. Available: http://abcnews.go.com/Technology/glassholes-privacy-issues-troubled-run-edition-google-glass/story?id=28269049

[19] Python Software Foundation, "Overview Python 2.7.12 documentation." [Online]. Available: https://docs.python.org/2.7/

[20] OpenCV Dev. Team, "Welcome to opencv documentation! OpenCV 2.4.13.1 documentation." [Online]. Available: http://docs.opencv.org/2.4/index.html

[21] F. Weinhaus, "Fred's ImageMagick Scripts: AUTOTRIM." [Online]. Available: http://www.fmwconcepts.com/imagemagick/autotrim/index.php

[22] ——, "ImageMagick: Convert, Edit, Or Compose Bitmap Images." [Online]. Available: http://www.imagemagick.org/script/index.php

[23] Frank Mittelbach and et all., "LaTeX Documentation." [Online]. Available: http://www.latex-project.org/help/documentation/

[24] Raspberry Pi Foundation, "Raspberry Pi Documentation." [Online]. Available: https://www.raspberrypi.org/documentation/

XV. Appendix A

Appendix A details the entire code used for the current project and it represents a singular file with only a few rudimentary attempts to optimize the code for speed and efficiency. Subsequent appendices isolate functional sections of the code in sequential fashion, so that the reader may "step" into the code, much like a break-point does in debugging.

A. Python Source Code for OCR

```
from pkg_resources import require
from tabulate import tabulate          # Nice console output tables.
import multiprocessing                  # For intercepting button commands.
import datetime as dt                  # For unique file names
import numpy as np                      # Array slicing
import subprocess                      # Terminal stuff
# import itertools                      # Itertools: CrackCocaine for Iterations.
import operator                        # String characters to operators
# import warnings                      # Debugging helper
# import argparse                      # Terminal commands
import os.path                         # Working with system paths
import signal                          # HIGH/LOW signal
# import atexit                        # Clean exits
import time                            # Sleep
# import pudb                          # Debugging helper
import sys                             # Stout terminal messages
import cv2                             # Opencv
import re                              # Regex. CrackCocaine for strings.

# Open channels, processes, and imports
try:
    print("Imported GPIO")
    import RPi.GPIO as GPIO
except ImportError:
    print("Can't find RPi.GPIO")
    GPIO = None
try:
    import picamera as pc
    print("Imported Camera")
except ImportError:
    print("Can't import the camera libraries. Try CTRL-D")
    pc = None
# try:
    # cpProc = 'sudo pkill -f takeSnap.py'
    # subprocess.call(cpProc, shell=True)

# Assigning GPIO channel for button presses
input = 17
GPIO.setmode(GPIO.BCM)
GPIO.setup(input, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Globals for path aliases, naming standards, etc.
global scripts
global debugTess
global save_path
global save_path
global tess
global TessParameters
global start
global timestamp
global initImage
```

```
52 global tierOne
53 global tierTwo
54 global tierTemp
55 global expDict
56
57 # Save image files with time stamps
58 start = dt.datetime.now()
59 timestamp = str(start.strftime("%d-%m-%y_%H:%M:%S") + ".png")
60 # Multicore threading process id
61 p = multiprocessing.current_process()
62
63 # Create path aliases
64 scripts = '/home/pi/projects/Senior_Project/MathOCR/'
65 debugTess = 'debug_file=/home/pi/projects/Senior_Project/Tess_Out/tesseract.log'
66 save_path = '/home/pi/projects/Senior_Project/'
67 save_path2 = '/home/pi/projects/Senior_Project/Images/'
68 tess = '/home/pi/projects/Senior_Project/Tess_Out/output'
69 # Parameters for OCR
70 TessParameters =
    ↪ '-c load_system_dawg=0 -c load_freq_dawg=0 -c {0} -c tessedit_write_images=true -c ambigs_
71     debugTess)
72
73 # Create seperate image files for each image in the clean up process
74 initImage = os.path.join(save_path, timestamp)
75 tierOne = os.path.join(save_path2, timestamp)
76 tierTwo = os.path.join(save_path2, timestamp)
77 tierTemp = os.path.join(save_path2, timestamp)
78
79 # Remove old files:
80 [os.remove(os.path.join("/home/pi/projects/Senior_Project/Images/", f))
81  for f in os.listdir("/home/pi/projects/Senior_Project/Images/") if
     ↪ f.endswith(".png")]
82 [os.remove(os.path.join("/home/pi/projects/Senior_Project/", f))
83  for f in os.listdir("/home/pi/projects/Senior_Project/") if f.endswith(".png")]
84
85 # Version control information (will most likely comment out at final)
86 print("Picamera Version: ", require('picamera')[0].version)
87 print("Install Directory: ", require('picamera'))
88 print("GPIO.RPI_REVISION:", GPIO.RPI_REVISION)
89 print("GPIO.VERSION: ", GPIO.VERSION)
90
91
92 def system_action(input):
93     '''Thread for listening in on button presses.
94     A single click captures an image, holding the button down for
95     at least two seconds quits the program, and anything over three
96     seconds will shut down the RPi completely.'''
97
98     button_press_timer = 0
99     # If input signal is a falling edge, send feedback.
100    if GPIO.input(input) == GPIO.LOW:
101        print('Button press = negative edge detected on channel %s' % input)
102        print("[PID:{}] acquired resource".format(p.pid))
103    # Wait for the button press to finish
104    while True:
105        # While Button is Pressed Down
106        if (GPIO.input(input) == GPIO.LOW):
107            # Keep on Counting
```

```
108             button_press_timer += 1
109             sys.stdout.write("Press time: {0}\r".format(button_press_timer))
110             sys.stdout.flush()
111             # print("Press time: {0}\r".format(button_press_timer, end='')),
112         else:
113             # Differentiating between single clicks and holding the button down
114             if(button_press_timer > .5 and button_press_timer < 3):
115                 # Capture image
116                 buttonProc(button_press_timer)
117                 button_press_timer = 0
118                 break
119             # press for > 2 seconds
120             elif (button_press_timer > 3 and button_press_timer < 10):
121                 os.kill(p.pid, signal.SIGINT)
122             elif (button_press_timer > 10):
123                 # Shutting down cleanly
124                 print("Closing Threads. Exiting.")
125                 print("Shutting RPi Down", button_press_timer)
126                 time.sleep(3)
127                 subprocess.call(
128                     ['shutdown -h now "System Halted by GPIO Action" &'],
129                     shell=True)
130             else:
131                 # reset the button timer.
132                 button_press_timer = 0
133                 break
134         time.sleep(.2)
135
136
137 def buttonProc(button_press_timer):
138     '''Image capture, retrieve contour bounding boxes,
139     image preproccessing, & call for OCR'''
140     global tierOne
141     global tierTwo
142     global tierTemp
143     cam.capture(initImage)
144     time.sleep(.5)
145     # Retrieve contour boundary dimensions from OpenCV
146     max_x, max_y, min_x, min_y = findROI()
147     # Might be useless to include this conditional
148     if(button_press_timer < 2):
149         # Protect original by making a new copy
150         workable = 'cp {0} {1}'.format(initImage, tierOne)
151         subprocess.call(workable, shell=True)
152         # Loop clean-up process until each desired preprocessing is complete
153         for i in range(3):
154             cmd = updatevals(i, tierTemp, tierOne, tierTwo,
155                              TessParameters, initImage, tess, max_x, max_y, min_x,
                                 ↪ min_y)
156             subprocess.call(cmd[i + 2], shell=True)
157             tierTemp = cmd[0]
158             tierOne = cmd[1]
159             # Debugging file names for each preprocessing argument:
160             # Output new file names for each clean-up stage
161             # if i < 4:
162             #     print(ctp(i) + "\n Old Save Name: " +
163             #           tierOne + "\n New Save Name: " + tierTemp)
164             # elif i == 4:
```

```
            #     print(ctp(i))
    time.sleep(.2)
    tessRead()

def findROI():
    '''Read image capture, apply grayscale, threshold,
       contours, and then determine ROI boundaries.'''
    # Create a multiprocessing.current_process() to kill program if neccessary
    global p
    image_src = cv2.imread(initImage)                                  # Opencv read image
    gray = cv2.cvtColor(image_src, cv2.COLOR_BGR2GRAY)                  # Grayscale
    ret, thresh = cv2.threshold(gray, 127, 255, 0)                     # Threshold
    contours = cv2.findContours(
        thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]  # Get contours
    if len(contours) > 0:  # Processing here.
        # Sort contour areas
        largest_area = sorted(contours, key=cv2.contourArea, reverse=True)
        # Empty mask array the shape of original image with unsigned integers
        # (255)
        mask = np.zeros(image_src.shape, np.uint8)
        # Draw contour regions using largest area list. -1 fills the shape.
        cv2.drawContours(
            mask, np.array(list(largest_area)), 0, (255, 255, 255, 255), -1)
        # Calculates the per-element bit-wise conjunction of two arrays.
        dst = cv2.bitwise_and(image_src, mask)
        # Remove unneccessary values from mask
        mask = 255 - mask
        # Sum of dst and mask = ROI
        roi = cv2.add(dst, mask)
        # cv2.imshow("Before", roi) ---- hardware limitations
        roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        ret, gray = cv2.threshold(roi_gray, 127, 255, 0)
        contours = cv2.findContours(
            gray, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
        max_x = 0
        max_y = 0
        min_x = image_src.shape[1]
        min_y = image_src.shape[0]
        for c in contours:
            # Determine boundary points
            if 150 < cv2.contourArea(c) < 100000:
                x, y, w, h = cv2.boundingRect(c)
                min_x = min(x, min_x)
                min_y = min(y, min_y)
                max_x = max(x + w, max_x)
                max_y = max(y + h, max_y)
        roi = roi[min_y:max_y, min_x:max_x]
        # cv2.imshow("After",roi) ---- hardware limitations
        # print(max_x, max_y, min_x, min_y) ---- Debugging purposes
        # Saves roi image for crop comparisons
        cv2.imwrite('/home/pi/projects/Senior_Project/Images/roi.png', roi)
    else:
        print("No image contours were found. Try again.")
        # Kill program.
        os.kill(p.pid, signal.SIGINT)
    return (max_x, max_y, min_x, min_y)
```

```
223 def updatevals(count, tierTemp, nextImage, tierTwo, TessParameters, initImage,
224                tess, max_x, max_y, min_x, min_y):
225     '''Image preprocessing. LTHRESH and CROP are the
226     only ones used in final version.'''
227     scripts = '/home/pi/projects/Senior_Project/MathOCR/'
228     tierOne = tierTemp
229     # Image file name structuring.
230     tierTemp = '{0}{1}_{2}'.format(tierTwo[:40], count, tierTwo[-21:])
231     # Convert OpenCV ROI bounds for ImageMagick's CROP
232     W = max_x - min_x
233     H = max_y - min_y
234     X = min_x
235     Y = min_y
236     # Might be useless to account for count here...
237     if count == 0:
238         # ImageMagick LTHRESH applies a threshold to the image
239         LTHRESH = '{0}localthresh -m 3 -r 40 -b 20 -n yes {1} {2}'.format(
240             scripts, initImage, tierTemp)
241     else:
242         LTHRESH = '{0}localthresh -m 3 -r 40 -b 20 -n yes {1} {2}'.format(
243             scripts, tierOne, tierTemp)
244     # ImageMagick CROP uses dimensions from OpenCV ROI
245     CROP = 'convert {0} -crop {1}x{2}+{3}+{4} -fuzz 1% -trim +repage {5}'.format(
246         tierOne, W, H, X, Y, tierTemp)
247     # Terminal argument to execute Tesseract after preprocessing
248     TESS = 'tesseract {0} {1} {2} -l custom -psm 6'.format(
249         tierOne, tess, TessParameters)
250     # Unused image processors:
251     # NEGATE = 'convert {0} -negate {1}'.format(tierOne, tierTemp)
252     # TCLEAN = '{0}textcleaner -g -e stretch -f 25 -o 10 -u -s 1 -T -p 10 {1}
        ↪ {2}'.format(scripts, tierOne, tierTemp)
253     # AUTOTRIM = '{0}autotrim -m o {1} {2}'.format(scripts, tierOne, tierTemp)
254     # UNROTATE = '{0}unrotate -f 30 {1} {2}'.format(scripts, tierOne, tierTemp)
255     # MONO = 'mogrify -monochrome {0}'.format(tierOne)
256     # GRAY = 'convert {0} -grayscale Rec709Luminance {1}'.format(initImage, tierOne)
257     # OTSU = '/home/pi/projects/Senior_Project/MathOCR/otsuthresh {0}
        ↪ {1}'.format(initImage,tierOne)
258     # DENOISE = '{0}denoise -f 10 -n 12 {1} {2}'.format(scripts, tierOne, tierTemp)
259     return(tierTemp, tierOne, LTHRESH, CROP, TESS)
260
261
262 def ctp(count):
263     '''Keep track of program process.'''
264     switch = {
265         0: "Cropping...",
266         1: "Local Threshold...",
267         2: "Running Tesseract-OCR...",
268     }
269     return switch.get(count, "nothing")
270
271
272 def tessRead():
273     '''tessRead() reads the text file generated by Tesseract
274     and then cleans up the line-separated strings corresponding
275     to singular equations with regular expression.The variable
276     tar handles adjacently-placed characters that yield illogical
277     problems, subst handles the substitution of those misplaced
278     characters, and regex cleans up the rest.'''
```

```
279     dataList = []
280     partList = []
281     subst = ['\1',
         ↪  '//','-','-','-','+','+','-','+','+','x','x','x','x','x','//','//','//','']
282     regex =
         ↪  re.compile(ur'([^\Da-w+$]+|[^\w=\\\'\;\/\.](?=[^\D])|[^\w=\\\'\;\/\.](?=\=[^\D])|[x]+(
283                       re.MULTILINE | re.IGNORECASE | re.VERBOSE | re.DOTALL |
                           ↪  re.UNICODE)
284     tar = [u'^[-+x*\/]+|[-+*x\/]+$|([-+*x\/])[-+*x\/]+',
285            u'(\xc3\xb7)',
286            u'=',
287            u'(\-\/)',
288            u'(\-x)',
289            u'(\-\-)',
290            u'(\+\+)',
291            u'(\+\-)',
292            u'(\+x)',
293            u'(\+\/)',
294            u'(xx)',
295            u'(x\+)',
296            u'(x\-)',
297            u'(x\/)',
298            u'(\/\/)',
299            u'(\/\+)',
300            u'(\/\-)',
301            u'(\/x)',
302            u'\x01']
303     # Read tess data and strip whitespaces
304     with open('{0}.txt'.format(tess), 'rw+') as f:
305         for line in f:
306             # Remove white spaces
307             subs = re.sub(r'\s+', '', line)
308             for j in range(0,2):
309                 for i in range(0,19):
310                     # Unicode conversions (for division symbols)
311                     subs = re.sub(tar[i], subst[i], subs)
312             # if current line is not empty
313             if subs:
314                 dataList.append(subs)
315             print subs
316         # Return to beginning of text file
317         f.seek(0)
318         # Overwrite original Tesseract output with the cleaned lines.
319         f.writelines(dataList)
320     # Feedback
321     print("OCR Results:", dataList)
322     # Final regular expression filter
323     for sm in dataList:
324         mPart = re.findall(regex, sm)
325         # print(type(mPart), mPart)
326         # mPart2 = re.findall(regex2, mPart)
327         partList.append(mPart)
328     # print("Datalist after Regex:", dataList)
329     # Send to crude digit/operator sorting function.
330     pickToSolve(partList)
331
332
333 def pickToSolve(partList):
```

```
334     finalEQ = []
335     finalCalc = []
336     ops = ['*','x','//','-','+']
337     # Obtain index and key of all the elements in list
338     for ind1, top in enumerate(partList):
339         # Reinitialize to null/zero to distinguish between separate equations
340         count = 0
341         runVal = 0
342         sumCalc = []
343         # Copy original lines for equation results
344         finalEQ.append(''.join(partList[ind1][:]))
345         # Enumerate for index and key in nested list
346         for ind2, item in enumerate(top):
347             # Account for the number of sublist elements >= 3
348             if len(partList[ind1][:]) >= 3:
349                 # count == 1 when index resides on the first operator
350                 if count == 1 and item in ops:
351                     # print("in count == 0")
352                     intOne = partList[ind1][ind2-1]
353                     intTwo = partList[ind1][ind2+1]
354                     # Evaluate this expression
355                     runVal = evalExp(intOne, item, intTwo)
356                     partList[ind1][ind2+1] = str(runVal)
357                     # Debugging
358                     # print(item)
359                     # print("runval 0", runVal)
360                     # print(intOne,item,intTwo,'*=',runVal)
361                     # print("{0}:{1}; {2}:{3}; runVal
                        ↪  {4}".format(intOne,ind2-1,intTwo,ind2+1,runVal))
362                     # Handle lookahead and lookbehind digits
363                 if count > 1 and item in ops:
364                     # print("in count > 0")
365                     intOne = partList[ind1][ind2-1]
366                     intTwo = partList[ind1][ind2+1]
367                     # Evaluate this expression
368                     runVal = evalExp(intOne, item, intTwo)
369                     partList[ind1][ind2+1] = str(runVal)
370                     # print(item)
371                     # print("runval 1", runVal)
372                     # print(intOne,item,intTwo,'=',runVal)
373                     # print("{0}:{1}; {2}:{3}; runVal
                         ↪  {4}".format(intOne,ind2-1,intTwo,ind2+1,runVal))
374                 # Running summations
375                 sumCalc.append(runVal)
376                 count+=1
377             # If singular line is composed only of a single digit
378             else:
379                 if partList[ind1][ind2-1].isdigit():
380                     runVal = partList[ind1][ind2-1]
381                 else:
382                     runVal = partList[ind1][ind2+1]
383                 sumCalc.append(runVal)
384                 count+=1
385         # Append the last index value of the running sums
386         finalCalc.append(sumCalc[-1])
387     # print("{0} = {1}".format(finalEQ, finalCalc))
388     # Create a table of equations and solutions
389     eq = ['=']*len(finalEQ)
```

```
390         zipped = zip(finalEQ, eq, finalCalc)
391         # print zipped
392         print tabulate(zipped, headers=['Input Eqn', '=', 'Result'])
393 
394 def evalExp(preOp, op, postOp):
395         '''Input digits and operators are evaluated.'''
396         return opSymb(op)(int(preOp), int(postOp))
397 
398 
399 def opSymb(op):
400         '''Parse the input variable op to determine which
401         mathematical function must be returned.'''
402         # try:
403         return{
404             '+':operator.add,
405             '-':operator.sub,
406             'x':operator.mul,
407             '*':operator.mul,
408             '//':operator.div,
409             '/':operator.div,
410             }[op]
411         # Misplaced undefined exceptions
412         # except ValueError:
413         #     print("Division by zero is undefined.")
414         #     print("Try again.")
415         #     # time.sleep(1.5)
416         #     # os.kill(p.pid, signal.SIGINT)
417 
418 # Setup the thread, detect a falling edge on channel 17 and debounce it
419 # with 200mSec
420 GPIO.add_event_detect(
421         input, GPIO.FALLING, callback=system_action, bouncetime=200)
422 # Initialize camera, establish properties and start preview:
423 cam = pc.PiCamera()
424 cam.exif_tags['IFD0.Copyright'] = 'Copyright (c) Mahdie Industries'
425 cam.resolution = (1920, 1080)
426 # Brightness value greatly impacts OpenCV contour bounding
427 # Good values are around 50 +/- 3
428 cam.brightness = 50
429 cam.led = False
430 # cam.sharpness = 100;
431 # Lower ISO value yields clearer images
432 cam.iso = 400
433 cam.vflip = True
434 cam.hflip = True
435 # cam.preview_alpha = 80
436 cam.start_preview(fullscreen=False, window = (600, 300, 640, 480))
437 # Loop until program is quit via button:
438 try:
439         while True:
440             time.sleep(.1)
441             pass
442 except KeyboardInterrupt:
443         print(" Application Manual Interrupt.")
444 finally:
445         cam.close()
446         GPIO.cleanup()
447         raise SystemExit
```

XVI. Appendix B

A. Signal Interrupt and Camera Initializations

```
# Setup the thread, detect a falling edge on channel 17 and debounce it
# with 200mSec
GPIO.add_event_detect(
    input, GPIO.FALLING, callback=system_action, bouncetime=200)
# Initialize camera, establish properties and start preview:
cam = pc.PiCamera()
cam.exif_tags['IFD0.Copyright'] = 'Copyright (c) Mahdie Industries'
cam.resolution = (1920, 1080)
# Brightness value greatly impacts OpenCV contour bounding
# Good values are around 50 +/- 3
cam.brightness = 50
cam.led = False
# cam.sharpness = 100;
# Lower ISO value yields clearer images
cam.iso = 400
cam.vflip = True
cam.hflip = True
# cam.preview_alpha = 80
cam.start_preview(fullscreen=False, window = (600, 300, 640, 480))
# Loop until program is quit via button:
try:
    while True:
        time.sleep(.1)
        pass
except KeyboardInterrupt:
    print(" Application Manual Interrupt.")
finally:
    cam.close()
    GPIO.cleanup()
    raise SystemExit
```

A system thread is created with a bouncetime of 200 ms to reduce the number of system callbacks due to switch bouncing. This could be done using a 0.1µF capacitor, but instead was done using the bouncetime parameter. The system callback is activated with the press of the button, which activates a falling edge on channel 17 of the RPi GPIO interface. From that callback, the button presses are considered and return the appropriate functionality.

Next, the PiCamera is initialized and given parameters for usage. The camera brightness was chosen to be 50 for the purposes of working with OpenCV's threshold algorithms. It was found that by reducing the brightness, contours were more easily found. Camera ISO value was set to the 400 without being subjected to optimized tweaking. A value of 400 is used for environments that are in bright daylight. Clearly, the project was done indoors, so the value of 400 was used to keep the noise level low.

Finally, a `while` loop embedded in a `try/except` conditional parses for system interrupts while allowing enough time to consider other CPU processes. If triggered, the program cleans up remaining processes and exits.

XVII. Appendix C

A. Asynchronous Button-Clicks

```
def system_action(input):
    '''Thread for listening in on button presses.
    A single click captures an image, holding the button down for
    at least two seconds quits the program, and anything over three
    seconds will shut down the RPi completely.'''

    button_press_timer = 0
    # If input signal is a falling edge, send feedback.
    if GPIO.input(input) == GPIO.LOW:
        print('Button press = negative edge detected on channel %s' % input)
        print("[PID:{}] acquired resource".format(p.pid))
    # Wait for the button press to finish
    while True:
        # While Button is Pressed Down
        if (GPIO.input(input) == GPIO.LOW):
            # Keep on Counting
            button_press_timer += 1
            sys.stdout.write("Press time: {0}\r".format(button_press_timer))
            sys.stdout.flush()
            # print("Press time: {0}\r".format(button_press_timer, end='')),
        else:
            # Differentiating between single clicks and holding the button down
            if(button_press_timer > .5 and button_press_timer < 3):
                # Capture image
                buttonProc(button_press_timer)
                button_press_timer = 0
                break
            # press for > 2 seconds
            elif (button_press_timer > 3 and button_press_timer < 10):
                os.kill(p.pid, signal.SIGINT)
            elif (button_press_timer > 10):
                # Shutting down cleanly
                print("Closing Threads. Exiting.")
                print("Shutting RPi Down", button_press_timer)
                time.sleep(3)
                subprocess.call(
                    ['shutdown -h now "System Halted by GPIO Action" &'],
                    shell=True)
            else:
                # reset the button timer.
                button_press_timer = 0
                break
        time.sleep(.2)
```

To parse asynchronous button-clicks, a system callback function, `system_action()`, was created to monitor falling edges on channel 17 of the RPi GPIO.

The button timer is initialized to zero and the function checks the state of the input signal passed to the callback. If it is low (which in this project it will always be), then the process ID, time pressed, and channel information is displayed in the terminal. Conditionals are utilized to achieve different modes of interaction for the button press length of time. Due to system latencies, the button press times are estimated and do not accurately measure in seconds, which explains the discrepancies between the times stated in the function description and in the `while` block. A single click takes a picture, a long press of about 1-2 seconds will exit the program, and anything over 3 seconds will terminate the program and shutdown the RPi. Each cycle of the callback function will reset the button-press timer.

XVIII. Appendix D

A. General Preparations and Image Processing

```
1  def buttonProc(button_press_timer):
2    '''Image capture, contour bounding boxes & outputs'''
3    global tierOne
4    global tierTwo
5    global tierTemp
6    cam.capture(initImage)
7    time.sleep(.5)
8    max_x, max_y, min_x, min_y = findROI()
9    if(button_press_timer < 2):
10     # Protect original by making a new copy
11     workable = 'cp {0} {1}'.format(initImage, tierOne)
12     subprocess.call(workable, shell=True)
13     # Loop clean-up process until each desired function is complete
14     for i in range(3):
15       cmd = updatevals(i, tierTemp, tierOne, tierTwo,
16                TessParameters, initImage, tess, max_x, max_y, min_x, min_y)
17       subprocess.call(cmd[i + 2], shell=True)
18       tierTemp = cmd[0]
19       tierOne = cmd[1]
20       # Output new file names for each clean-up stage
21       # if i < 4:
22       #   print(ctp(i) + "\n Old Save Name: " +
23       #       tierOne + "\n New Save Name: " + tierTemp)
24       # elif i == 4:
25       #   print(ctp(i))
26   time.sleep(.2)
27   tessRead()
```

The function `buttonProc()`, executes one of three options when triggered. Asynchronous signal events are produced by the button on the side of the camera apparatus when not in an "atomic" state, or more succinctly, when the program is not in the middle of an operation. To watch for the asynchronous signal, a thread was created (See Appendix C) to detect a falling edge on pin 17 of the RPi. This was useful to implement since the program needed to idle until some input was given.

Beginning with line 6, the function obtains the initial image to be preprocessed and makes a copy of it in another directory by passing an argument to the terminal environment. The input image is processed by OpenCV in `findROI()`, such that the relevant data is bounded and cropped for cleaner OCR results. Each successive manipulation of the initial capture is renamed and saved for the purposes of debugging the preprocessing stage in `updatevals()`. Finally, `tessRead()` is called, where the OCR functionality takes place.

XIX. Appendix E

A. OpenCV Bounding Contours and Thresholds

```
def findROI():
    '''Read image capture, apply grayscale, threshold,
        contours, and then determine ROI boundaries.'''
    # Create a multiprocessing.current_process() to kill program if neccessary
    global p
    image_src = cv2.imread(initImage)                                   # Opencv read image
    gray = cv2.cvtColor(image_src, cv2.COLOR_BGR2GRAY)                   # Grayscale
    ret, thresh = cv2.threshold(gray, 127, 255, 0)                      # Threshold
    contours = cv2.findContours(
        thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]  # Get contours
    if len(contours) > 0:  # Processing here.
        # Sort contour areas
        largest_area = sorted(contours, key=cv2.contourArea, reverse=True)
        # Empty mask array the shape of original image with unsigned integers
        # (255)
        mask = np.zeros(image_src.shape, np.uint8)
        # Draw contour regions using largest area list. -1 fills the shape.
        cv2.drawContours(
            mask, np.array(list(largest_area)), 0, (255, 255, 255, 255), -1)
        # Calculates the per-element bit-wise conjunction of two arrays.
        dst = cv2.bitwise_and(image_src, mask)
        # Remove unneccessary values from mask
        mask = 255 - mask
        # Sum of dst and mask = ROI
        roi = cv2.add(dst, mask)
        # cv2.imshow("Before", roi) ---- hardware limitations
        roi_gray = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
        ret, gray = cv2.threshold(roi_gray, 127, 255, 0)
        contours = cv2.findContours(
            gray, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)[0]
        max_x = 0
        max_y = 0
        min_x = image_src.shape[1]
        min_y = image_src.shape[0]
        for c in contours:
            # Determine boundary points
            if 150 < cv2.contourArea(c) < 100000:
                x, y, w, h = cv2.boundingRect(c)
                min_x = min(x, min_x)
                min_y = min(y, min_y)
                max_x = max(x + w, max_x)
                max_y = max(y + h, max_y)
        roi = roi[min_y:max_y, min_x:max_x]
        # cv2.imshow("After",roi) ---- hardware limitations
        # print(max_x, max_y, min_x, min_y) ---- Debugging purposes
        # Saves roi image for crop comparisons
        cv2.imwrite('/home/pi/projects/Senior_Project/Images/roi.png', roi)
    else:
        print("No image contours were found. Try again.")
        # Kill program.
        os.kill(p.pid, signal.SIGINT)
    return (max_x, max_y, min_x, min_y)
```

OpenCV was incredibly useful to the development of this program. In hindsight, time would have been better spent learning how to fully take advantage of OpenCV since it proved to be a powerful asset. In this project, it was solely utilized for the identification and extraction of handwriting script among various leftover artifacts and graphical noise surrounding the text after the initial cleanup.

To begin, the initial image capture is read into memory and then converted into grayscale for the discovery of threshold boundaries. Using the threshold values, the associated contours are found and sorted into area bins of largest to smallest order. A mask array with the shape of the original image is created and the discovered contours are filled.

Using the original image and the mask, a conjunction of the two arrays is performed and the the region of interest is determined from the additive area of the per-element, bit-wise conjunction and the effective mask. The new image is then saved for debugging purposes.

Region of Interest boundary values are collected and passed back to buttonProc to be used in cropping in the image cleanup process.

XX. Appendix F

A. Image Processing

```
1  def updatevals(count, tierTemp, nextImage, tierTwo, TessParameters, initImage,
2                 tess, max_x, max_y, min_x, min_y):
3      '''Image processing'''
4      scripts = '/home/pi/projects/Senior_Project/MathOCR/'
5      tierOne = tierTemp
6      tierTemp = '{0}{1}_{2}'.format(tierTwo[:40], count, tierTwo[-21:])
7      W = max_x - min_x
8      H = max_y - min_y
9      X = min_x
10     Y = min_y
11     if count == 0:
12         LTHRESH = '{0}localthresh -m 3 -r 40 -b 20 -n yes {1} {2}'.format(
13             scripts, initImage, tierTemp)
14     else:
15         LTHRESH = '{0}localthresh -m 3 -r 40 -b 20 -n yes {1} {2}'.format(
16             scripts, tierOne, tierTemp)
17     CROP = 'convert {0} -crop {1}x{2}+{3}+{4} -fuzz 1% -trim +repage {5}'.format(
18         tierOne, W, H, X, Y, tierTemp)
19     TESS = 'tesseract {0} {1} {2} -l custom -psm 6'.format(
20         tierOne, tess, TessParameters)
21     # Unused image processors:
22     # NEGATE = 'convert {0} -negate {1}'.format(tierOne, tierTemp)
23     # TCLEAN = '{0}textcleaner -g -e stretch -f 25 -o 10 -u -s 1 -T -p 10 {1}
       ↪ {2}'.format(scripts, tierOne, tierTemp)
24     # AUTOTRIM = '{0}autotrim -m o {1} {2}'.format(scripts, tierOne, tierTemp)
25     # UNROTATE = '{0}unrotate -f 30 {1} {2}'.format(scripts, tierOne, tierTemp)
26     # MONO = 'mogrify -monochrome {0}'.format(tierOne)
27     # GRAY = 'convert {0} -grayscale Rec709Luminance {1}'.format(initImage, tierOne)
28     # OTSU = '/home/pi/projects/Senior_Project/MathOCR/otsuthresh {0}
       ↪ {1}'.format(initImage,tierOne)
29     # DENOISE = '{0}denoise -f 10 -n 12 {1} {2}'.format(scripts, tierOne, tierTemp)
30     return(tierTemp, tierOne, LTHRESH, CROP, TESS)
```

The function `updatevals()` handles the program's progress, naming associations, and image preprocessing. This is done by inputting the current stage count, the initial and temporary images to be altered, the lookahead image, the parameters of the algorithm to be used by Tesseract, and the dimensions of the local equation as determined by OpenCV. The interesting part of this function can be found starting on line 11, which states a conditional for passing arguments to the RPi terminal. `LTHRESH` and `CROP` were the only two preprocessing filters from ImageMagick that were determined to be critical in the clean-up stage. All of the commented preprocessing arguments were found to be excessive in cleaning up the input image.

In the initial passthrough case, the first argument is to perform a threshold on the initial image and remove a large number of artifacts. Following this, every pass through this function where count is greater than zero (or the image is not the initial image), a threshold is applied to a copy of the original image. The `CROP` argument armed with the relevant region of interest dimensions from OpenCV worked well for removing unnecessary portions of the image. Finally, when `TESS` is passed to the terminal, it invokes Tesseract to process the clean image.

XXI. Appendix G

A. Tesseract Output and Regular Expressions

```
def tessRead():
  dataList = []
  partList = []
  subst = ['\1',
   ↪  '//','-','-','-','+','+','-','+','+','x','x','x','x','x','//','//','//','']
  regex =
   ↪  re.compile(ur'([^\Da-w+$]+|[^\w=\\\'\;\/\.](?=[^\D])|[^\w=\\\'\;\/\.](?=\=[^\D])|[x]+(?=
            re.MULTILINE | re.IGNORECASE | re.VERBOSE | re.DOTALL | re.UNICODE)
  tar = [u'^[-+x*\/]+|[-+*x\/]+$|([-+*x\/])[-+*x\/]+',
       u'(\xc3\xb7)',
       u'=',
       u'(\-\/)',
       u'(\-x)',
       u'(\-\-)',
       u'(\+\+)',
       u'(\+\-)',
       u'(\+x)',
       u'(\+\/)',
       u'(xx)',
       u'(x\+)',
       u'(x\-)',
       u'(x\/)',
       u'(\/\/)',
       u'(\/\+)',
       u'(\/\-)',
       u'(\/x)',
       u'\x01']
  # Read tess data and strip whitespaces
  with open('{0}.txt'.format(tess), 'rw+') as f:
    for line in f:
      subs = re.sub(r'\s+', '', line)    # White spaces
      for j in range(0,2):
        for i in range(0,19):
          subs = re.sub(tar[i], subst[i], subs)  # Unicode conversions
      if subs:
        dataList.append(subs)
      print subs
      # if cleanedLine: # is not empty
      #   dataList.append(cleanedLine)
    f.seek(0)
    f.writelines(dataList)
  print("OCR Results:", dataList)
  for sm in dataList:
    mPart = re.findall(regex, sm)
    # print(type(mPart), mPart)
    # mPart2 = re.findall(regex2, mPart)
    partList.append(mPart)
  # print("Datalist after Regex:", dataList)
  pickToSolve(partList)
```

The function `tessRead()` parses the text file generated by Tesseract for any unwanted characters. It completes this by utilizing regular expressions. Regex is largely uninteresting to explain, so I will limit the discussion here to the overall method of cleaning–which was to remove white spaces, empty lines, alien characters, and to eliminate illogical operators.

XXII. Appendix H

A. List Segmentations

```
def pickToSolve(partList):
    finalEQ = []
    finalCalc = []
    ops = ['*','x','//','-','+']
    # Obtain index and key of all the elements in list
    for ind1, top in enumerate(partList):
        # Reinitialize to null/zero to distinguish between separate equations
        count = 0
        runVal = 0
        sumCalc = []
        # Copy original lines for equation results
        finalEQ.append(''.join(partList[ind1][:]))
        # Enumerate for index and key in nested list
        for ind2, item in enumerate(top):
            # Account for the number of sublist elements >= 3
            if len(partList[ind1][:]) >= 3:
                # count == 1 when index resides on the first operator
                if count == 1 and item in ops:
                    # print("in count == 0")
                    intOne = partList[ind1][ind2-1]
                    intTwo = partList[ind1][ind2+1]
                    # Evaluate this expression
                    runVal = evalExp(intOne, item, intTwo)
                    partList[ind1][ind2+1] = str(runVal)
                    # Debugging
                    # print(item)
                    # print("runval 0", runVal)
                    # print(intOne,item,intTwo,'*=',runVal)
                    # print("{0}:{1}; {2}:{3}; runVal
                     ↪  {4}".format(intOne,ind2-1,intTwo,ind2+1,runVal))
                    # Handle lookahead and lookbehind digits
                if count > 1 and item in ops:
                    # print("in count > 0")
                    intOne = partList[ind1][ind2-1]
                    intTwo = partList[ind1][ind2+1]
                    # Evaluate this expression
                    runVal = evalExp(intOne, item, intTwo)
                    partList[ind1][ind2+1] = str(runVal)
                    # print(item)
                    # print("runval 1", runVal)
                    # print(intOne,item,intTwo,'=',runVal)
                    # print("{0}:{1}; {2}:{3}; runVal
                     ↪  {4}".format(intOne,ind2-1,intTwo,ind2+1,runVal))
                # Running summations
                sumCalc.append(runVal)
                count+=1
            # If singular line is composed only of a single digit
            else:
                if partList[ind1][ind2-1].isdigit():
                    runVal = partList[ind1][ind2-1]
                else:
                    runVal = partList[ind1][ind2+1]
                sumCalc.append(runVal)
                count+=1
        # Append the last index value of the running sums
```

```
54          finalCalc.append(sumCalc[-1])
55      # print("{0} = {1}".format(finalEQ, finalCalc))
56      # Create a table of equations and solutions
57      eq = ['=']*len(finalEQ)
58      zipped = zip(finalEQ, eq, finalCalc)
59      # print zipped
60      print tabulate(zipped, headers=['Input Eqn', '=', 'Result'])
61
62  def evalExp(preOp, op, postOp):
63      '''Input digits and operators are evaluated.'''
64      return opSymb(op)(int(preOp), int(postOp))
65
66
67  def opSymb(op):
68      '''Parse the input variable op to determine which
69      mathematical function must be returned.'''
70      # try:
71      return{
72          '+':operator.add,
73          '-':operator.sub,
74          'x':operator.mul,
75          '*':operator.mul,
76          '//':operator.div,
77          '/':operator.div,
78          }[op]
```

The three functions, `pickToSolve()`, `evalExp()`, and `opSymb()` are grouped together here by relation.

`pickToSolve()` handles the line-by-line parsing of Tesseract output data. Expressions are separated and evaluated from one another if they're separated by a new line. Each expression is collected into an array and the resulting evaluation is stored into another array. This is for the purposes of producing a tabular representation of data in the terminal.

`evalExp()` sends the adjacent input integers separated by some string operator to the function `opSymb()`, where it promptly returns the associated operator function of the operator string and evaluates the three groups in linear fashion. As such, no order of operations is performed on the overall expression.

XXIII. Appendix I

A. Tesseract Selection Rules

```
v1
4    - - 2 - -    1    =    1
4    - - 2 --     1    =    1
4    -- 2 - -     1    =    1
4    --2--        1    =    1
4    -- 2 --      1    =    1
4    - -2--       1    =    1
4    -- 2--       1    =    1
4    --2 --       1    =    1
4    --2- -       1    =    1
4    - -2- -      1    =    1
4    --=-         1    =    1
4    ----         1    =    1
3    -- -         1    =    1
3    - --         1    =    1
3    - - -        1    =    1
3    ---          1    =    1
3    -=-          1    =    1
3    - =-         1    =    1
3    =--          1    =    1
3    =- -         1    =    1
3    -= -         1    =    1
3    --=          1    =    1
3    - = -        1    =    1
3    =2-          1    =    1
3    2-=          1    =    1
2    22   2    =2   1
2    1|   2    11   1
2    |1   2    11   1
2    +/   2    +1   1
2    -/   2    -1   1
2    */   2    *1   1
2    ||   2    11   1
2    +-   1    +    1
2    -+   1    +    1
2    - - 1     =    1
2    --   1    =    1
2    -=   1    =    1
2    =-   1    =    1
1     2   1    =    1
```

```
v2
22 =2 1
27 =7 1
25 =5 1
72 7= 1
52 5= 1
-=- -2 1
2-- 2= 1
7-- 7= 1
5-- 5= 1
--2-- -2= 1
--7-- -7= 1
--5-- -5= 1
-2-- -2= 1
-7-- -7= 1
-5-- -5= 1
--2 =2 1
--7 =7 1
--5 =5 1
-- = 1
--- =- 1
=-- =-
1| 1/ 1
|1 /1 1
+/ +1 1
-/ -1 1
| 1 1
*/ *1 1
+- - 1
-+ - 1
```

Initially, an older version of Tesseract was utilized before updating to the current version `3.0.5` (This was a forked Windows executable version). Because of this, the syntax for creating selection rules were slightly different. In v1, the rules required the number of characters to be passed to Tesseract for consideration, but that was not the case in v2.

See Table 1 for the syntax form of v2.

There are considerable differences between both v1 and v2 due to the evolution of the coding process. The majority of v1 selection rules were needed for the sake of testing, later of which was redundant due to regular expression handling in the final version of code. It is was determined that lines 23-28 in v2 were unnecessary to include due to the fact that another file required by Tesseract (unicharset) governed all acceptable characters in the program. By populating the unicharset file, there was no need for these conditional selection rules in lines 23-28.