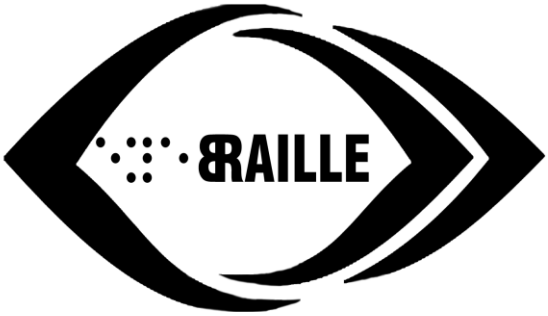
McMaster University
ENGINEERING



EYEBRAILLE

Reading and Learning Device for Visually Impaired.

Name	Student Number	MacID
Shajee Zuhair	001401116	zuhairs
Shubham Gupta	001416773	guptasr
Mrudang Kadakia	001422030	kadakmj
Gowthaman Vijayakumaran	001437500	vijayag

Group B12

Table of Contents

Introduction

According to the World Health Organization, there are an estimated 253 million people that live with vision impairment. Among the 253 million, 36 million are blind (WHO, 2017). Some of these individuals are deaf-blind and as a result rely heavily on learning braille alphabets. However, many of them have never been able to utilize the braille language in harmony with computers, tablets or smartphones to read electronic texts.

There are many factors that contribute to this phenomenon, however one of the major factor is price. The electronic “Refreshable Braille Displays can cost anywhere from $1,500 all the way up to almost $10,000” (Mineault, 2018). They are essentially a piece of hardware that provide braille output from a computer, smartphone or tablet (refer to Fig 1. & Fig 2. below). They work by raising and lowering plastic or metal pins which correspond to the letters and numbers in braille language.

Figure 1: Orbit Reader 20. Cost: $900

Figure 2: Humanware - Brailliant BI 40. Cost: $5,000

The goal of EyeBraille is to create a cost-effective electronic braille reader that takes an electronic text input from a PDF document, and converts it to a braille output. The project will use similar concept from refreshable braille displays of raising and lowering pins to represent letters and numbers for its respectable braille alphabets. Furthermore, the product’s target audience will be those who seek a cost-effective braille reading and learning device.

Project Summary

The main goal is to develop a braille reading and learning device for the visually impaired that is affordable, as well as easy to use. It will also be utilized to teach phonetic alphabets to children who are born visually impaired. The device will be fully capable of converting PDF documents into its equivalent braille translation, as well as output an audio using the on-board speaker. EyeBraille aims to provide an enjoyable, and hassle-free experience by ensuring a simple and minimalistic setup procedure. It comes with a user-interface that allows a second individual to easily setup the device for the visually impaired user. The device is also compact and perfectly captures portability, while keeping the overall cost to $200.

The Graphical User Interface (GUI) will also be utilized to allow users to upload PDF documents or text files onto the device. It also provides the capability to store the files which can be accessed later by the user anytime without having to upload the document again.

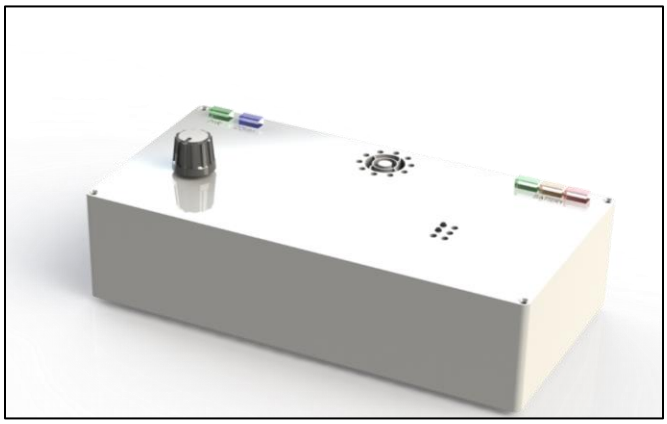
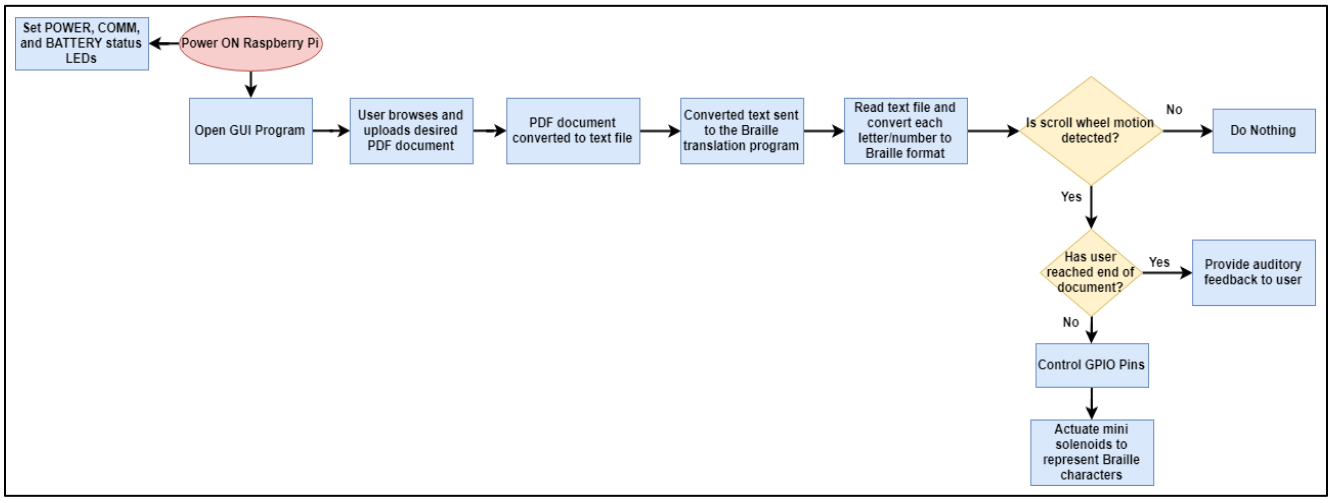


Figure 3: Preliminary Design

System Block Diagram



Design Requirements and Specifications

In order to keep the program easy to understand, the device will use Python as the primary programming language. Since Python is natively supported in Raspberry Pi, it will act as the single board computer. A key library that will be utilized is “pdfminer”, an open source library from GitHub [1]. This will convert the PDF document to a text document which can then be read letter by letter. Also, a conventional mouse scroll wheel will also be used to determine the speed and direction the user desires to read the document. The main function of the program will be to determine a method where the text document is being read with respect to the scroll wheel. Once the program determines the letter the user needs, micro solenoids referred in Figure 4 below will actuate the pins in the 3 by 2 grid. Thus, the resulting motion will represent the braille translation of the letter that is being read by the user in the program.

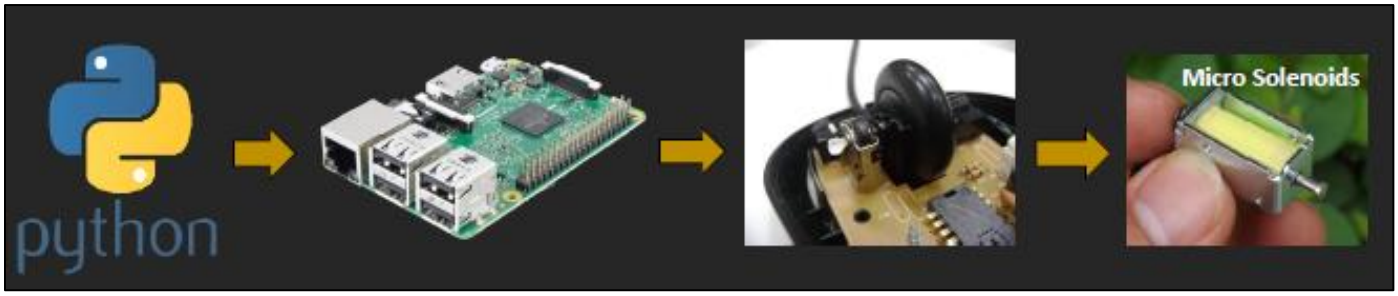


Figure 4: Design

System Design

Mechanical

The physical device sits inside a (200 x 100 x 55mm) enclosure that houses the internal circuitry and the micro solenoids. As seen in Figure. 5, the top panel of the enclosure has cut outs that are going to be used for speakers for providing auditory notices and LEDs to provide information regarding battery level, power, and communication to the user. A cut out is also placed for a scroll wheel on the bottom left which will be used to give the user the freedom of reading the text file in their preferred speed, as well as allow the user to go backwards to re-read missed text. Finally, six small holes are placed on the right side in a 3 by 2 grid configuration which will be used to represent each character in its braille format as seen in Figure 6.

Originally, the enclosure was 3D printed using ABS plastic, however earlier on during prototyping, it was found that the 3D printer did not accurately replicate the details on the top panel of the enclosure. Thus, the six holes used to represent a braille character would not print to the dimensions that were needed. Also, the other cutouts which were needed on the top enclosure were not as smooth as desired. Another problem which emerged using ABS plastic was the material would shrink once it was cooled, therefore it would not fit well with other parts. To solve this problem a different method and

material was used to create the enclosure. The new material was plywood, which would provide a strong and sturdy base for the enclosure and be able to protect all the electrical components that would be stored inside. The solution that yielded the best result was to use a laser cutter to precisely design the enclosure pieces to the required dimensions.

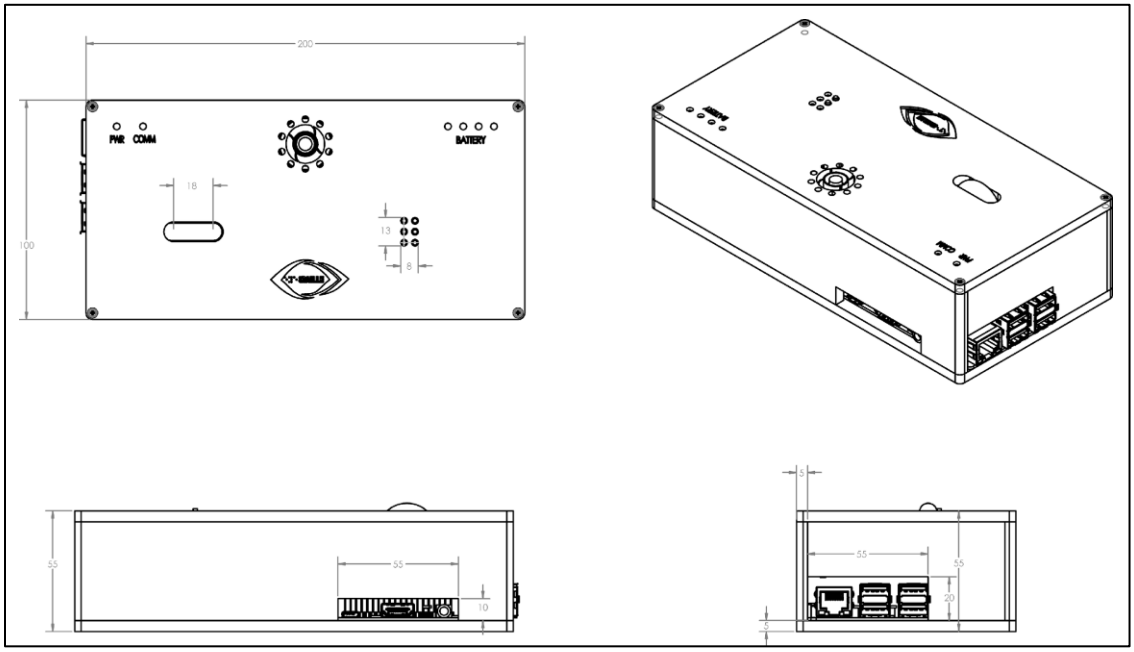


Figure 5: Engineering CAD Design

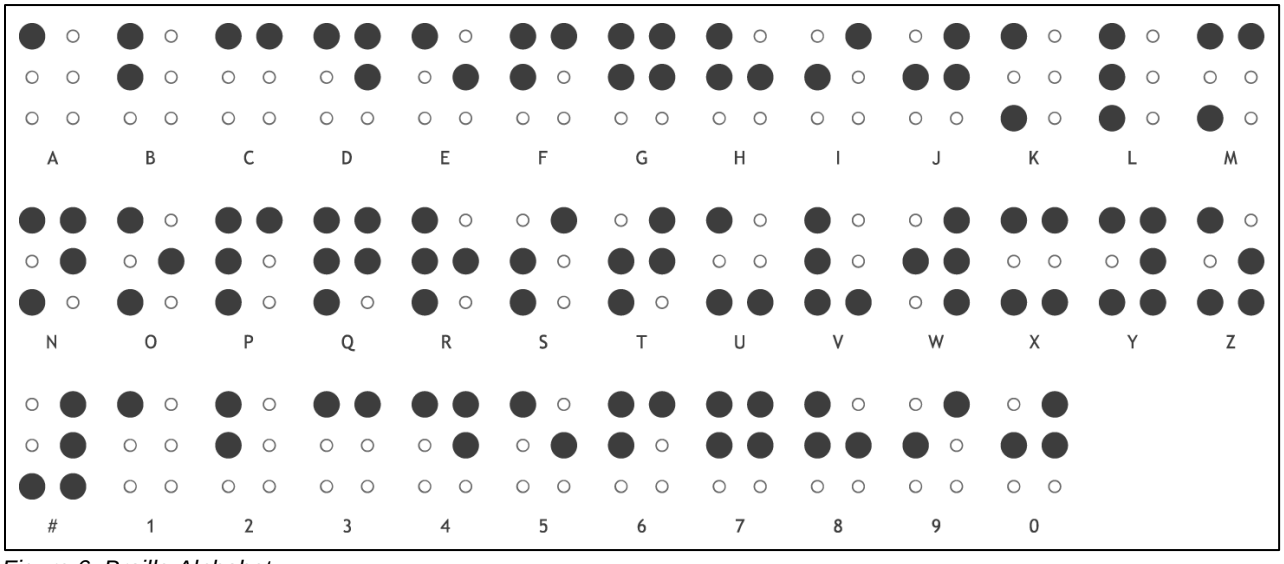


Figure 6: Braille Alphabet

The six holes located on the right side of the panel will be used to represent a braille character arranged in a 3 by 2 grid measured (8x13mm). This is so that a typical finger could read the entire character without having to move around. Each hole will have a pin in them which can actuate out of the hole, and depending on which pins are up and

which pins are down, it will represent a unique braille character as shown in Figure 6. Each pin will be connected individually to an actuator that is able to raise or lower. However, as shown in Figure 7 the actuators controlling the pins when arranged in a 3 by 2 grid were further apart than intended. Placing the solenoids together resulted in the pins being placed further apart. This was a major issue since it will not fit the (8x13mm) surface that is designed for the braille pad. To solve the problem custom pins were 3D printed for the head of the actuators in order to fit into the existing (8x13mm) braille pad as seen in Figure 8. The six actuators will then be connected to a microcontroller that will control which pins will actuate to produce braille output.

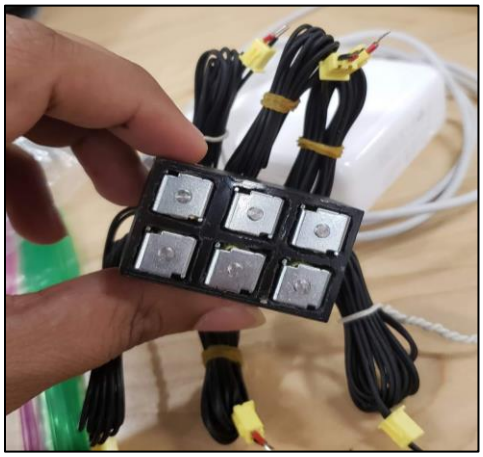


Figure 7: Solenoids housed to represent a braille character

Figure 8: 3D Printed Pins

One of the initial designs for the user to control the direction and speed at which the text was output was to use a potentiometer. The main principle with the design is that when the potentiometer is at the center, the direction is set to 0 and the speed is set to 0. When the potentiometer is at the right direction, the reading direction is set to right.

Similarly, when the potentiometer is set to the left, the direction is set to the left (allowing the user wants to read in the backward direction). The position at which the potentiometer is set between the center and the furthest point in the right direction determines the speed at which the user wants to read the text. The potentiometer would work by sending a voltage value to the Raspberry Pi which would determine the direction and speed at which the text is being read by the user. However, this approach does not provide the user with complete freedom of direction and speed. This is because, when the potentiometer is set to a specific direction at a specific speed, it is possible for the user to miss a character they would like to read. If the user wants to go back, they will have to re-adjust the potentiometer backwards. Since the potentiometer will not change direction until it has passed the center line, the user will miss more characters. Also, once the user has passed the center line and read the desired characters, the user will have to go forwards. Moreover, if the potentiometer is not returned to the center, the text will keep displaying at the specific speed and direction the potentiometer is left at. This will lead the user to be frustrated with finding the correct speed and direction to read the document at a leisurely pace.

In order to avoid this, a mouse scroll wheel is used. Since the Pi has dedicated USB inputs, a normal mouse scroll wheel PCB is directly connected to the Pi via USB. From this, the direction and speed of the scroll wheel is easily determined and sent to Python. This allows the text speed and direction to be controlled with more precision. Once the mouse scroll wheel is left untouched, the last text position is always maintained. This is similar to scrolling through a web-page or a word document on the computer. The user has complete control of the speed and direction at which they are reading the document.

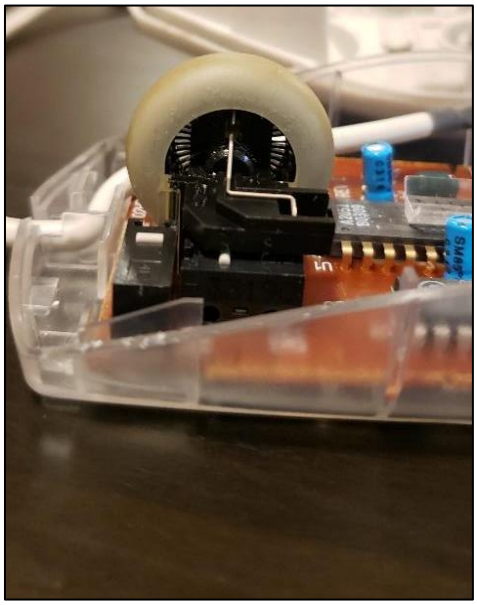


Figure 9: Mouse Scroll-Wheel

Electrical

The electrical components are all housed inside the device, these include the Raspberry Pi, the six solenoids, the scroll wheel, a voltage regulator, 5000 mAh power bank, and a circuit board. The scroll wheel is the only user required input which takes the scrolling motion as a signal into the Raspberry Pi and translates each digital text from the pdf to its equivalent braille character. Once the character has been translated to its braille equivalent the program will energize the required solenoids and actuate them to their correct positions.

The six solenoids consist of a coil of wire wrapped around a piston. When current passes through the coil, an electromagnetic field is generated. The device uses the magnetic field to create linear motion causing the piston to move up. The advantage of using an electromagnet is that the magnet can be energized by simply supplying a current through it. The solenoids operate on 12 volts (V), however the microcontroller can only supply a maximum of 3.3 volts (V) through its General-Purpose Input/output (GPIO) pins, therefore an external power supply was used. MOSFET transistors are also used as switches to control when the solenoids would either rise or fall. Referring to Figure 10 and Figure 11, the output pin from the microcontroller supplying the 3.3V is connected to a resistor which is then connected to the base of the transistor. The resistor is used to limit the current going into the transistor, and the ground is connected to the emitter of the transistor. The external 12V supply is connected to the positive end of the solenoid, while the negative end is connected to the collector of the transistor. A diode is placed in between the positive and negative end of the solenoid in order to eliminate fly back. A fly back is caused when there is as sudden voltage spike across an inductive load or when the current going through it is suddenly interrupted. This interruption can cause the inductive load to induce a very large voltage and cause a momentary electric arc. This arc can damage the transistor and ultimately become a hazard for our user. By placing a diode, the current will not be conducted in the solenoid due to reverse bias. As well as, when the current going through the solenoid is suddenly interrupted, the induced voltage across the inductive load would cause a forward bias in the diode and allow the current to go through. Ultimately, limiting the voltage across the inductor and prevent an arc from forming.

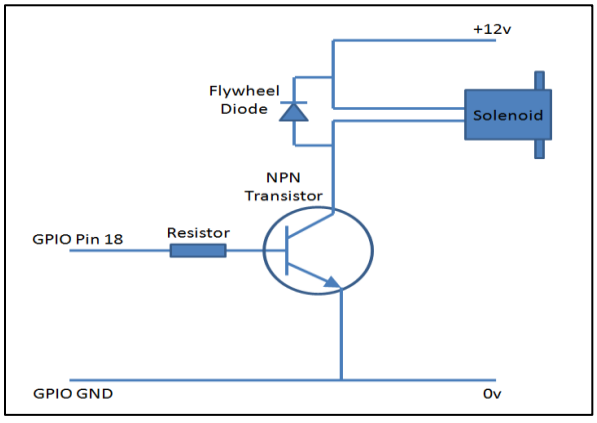


Figure 10: Voltage Controlled Switch

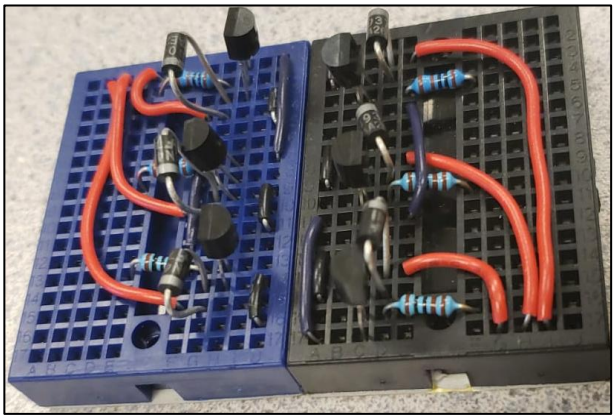


Figure 11: Voltage Controlled Switch

Furthermore, the transistors act as voltage-controlled switches that will control current to flow through the solenoids. When the output pin from the microcontroller outputs a digital high at the base, the transistor will allow current to flow from the collector to the emitter, thus completing the circuit and allowing the solenoid to actuate its piston. When the output pin from the microcontroller outputs a digital low at the base of the transistor, current will not be able to flow from the collector to the emitter thus no current will flow through the solenoid causing the piston to de-energize.

GUI/Program

The graphical user interface (GUI) is designed to be simple and straightforward to use. It has a single functionality; to allow the assistant to upload a PDF file to the device. The GUI is built using various open source libraries available in the python environment. The pygame library in python is used to create and display the main GUI, and use the Tkinter library to allow the aid to access the PDF file. The PDF file is processed and convert it to a text file so that it can be output as tangible braille letters using the solenoids. The process that is performed on the PDF file is as follows: Firstly, the PDF file is converted into a text file using the pdfminer library. This ignores any images that may reside in the PDF file and isolates the text. This text file is stored in a directory from where it is accessed by the text to braille conversion code which converts the initial character to its binary representation of braille before outputting to the device in braille. This code allows the user to use the scroll wheel to shift to the next character or previous character as they please. Since there are a limited number of characters represented in braille, and not all special characters can be represented (such as @), the special characters are ignored.

Figure 12: Graphical User Interface (GUI)

Results

The first major tasks that needed to be completed was to create the program to translate a given pdf character into its braille translation. This was done by creating a database of characters and cross referencing their braille translation on a database. Since the program was the most important component, time could not be wasted on building an actual prototype to test if the program was able to correctly translate text into the braille translation. Instead individual characters were translated and checked to verify if the program would output the correct high and low values that would be associated with the corresponding pins as needed.

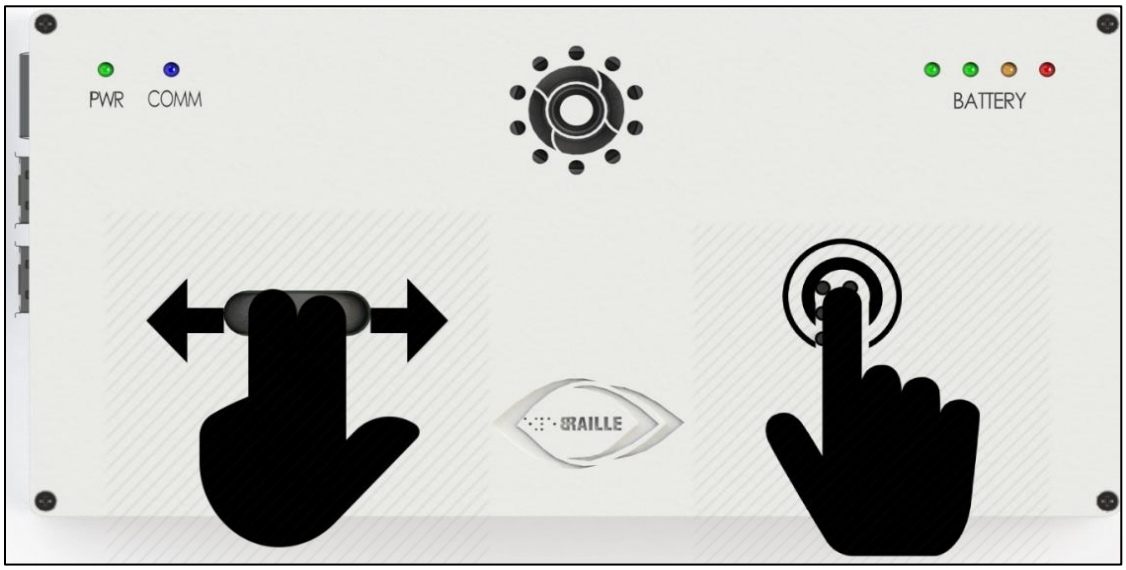


Figure 13: Final Design

Once it was verified that the program could output the correct high and low values to the corresponding pins, the next step was to see begin iterations on physically displaying the braille characters. The first prototype created was to represent a braille character using LEDs. The program was modified so that after a character was translated by the program, it would output a high or low digital value to the output pins on the single board computer to the corresponding LED's located in the breadboard as seen in Figure 14. This prototype was a big milestone for the group, as the next step would now be to replace the LED's with solenoids and incorporate the scroll wheel.

Still working with the first prototype, the scroll wheel was implemented into the program. The program was again modified so that when the user moves the scroll wheel the right, the program will translate each character as it moves right along the text. The scroll wheel was successfully integrated into the prototype as every time the scroll wheel was moved the correspond text it passed by was translated correctly and displayed correctly using the LEDs.

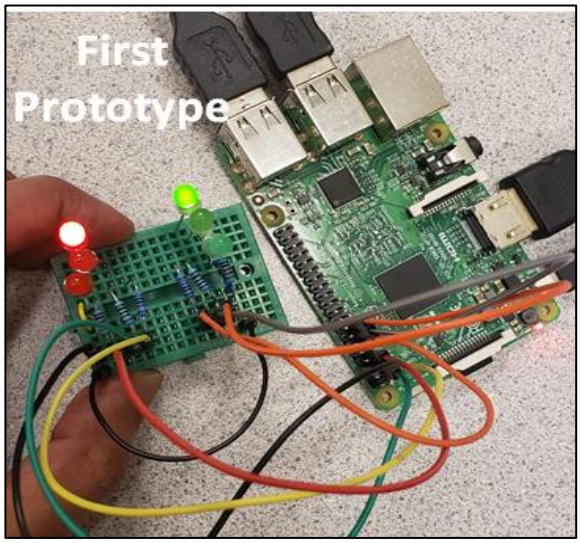


Figure 14: LED implementation

Once all the internal components were figured out, the next step was to create the enclosure. For the first prototype, the enclosure was 3D printed. However, there was a quick realization that the 3D printer was not accurate enough to create the details that were required at the top of the enclosure. It was decided then that the enclosure would be made of wood and laser cut.

Figure 15: Finished Product

Discussion

At the end of the capstone course our group was satisfied with what we were able to accomplish and were pleased with our final device. We were able to meet all of the goals we set and have already began planning the next steps to improve the device in the future.

Although we were satisfied with our final product, we explored many options in the way we would implement our idea. Our initial idea was to have a row of 36 by 3 arrangement of dots to represent 18 characters in the braille alphabet. That way the user could just slide their finger across the board and read a whole word or a small sentence just like how someone would read a normal braille text. The problem we quickly realized is that the microcontroller that we were using only had 40 GPIO pins that we could use as output pins to the solenoids, meaning that only 6 braille cells could be represented using one microcontroller. In order to represent all 18-braille cells, we would require 2 microcontrollers. Another problem with this design was the need for 108 solenoids. The space needed to house 108 solenoids would make our device a lot larger than what one of our requirements was, which was to make it small and portable. For that reason, our final design was to have 6 solenoids that would represent a single braille character and have those 6 solenoids change after each character is translated one at a time, this way our device would be small, and portable.

One of the biggest accomplishments of this design for us was being able to implement the scroll wheel which will give the user the freedom to control the speed at which they would want to read the text. Our original design had a potentiometer instead of a scroll wheel that could be set to a predetermined speed the braille text will be translated. The problem with this idea was we did not know what specific speeds different users would be comfortable with. An experienced reader might find it too slow, while an inexperienced user might find the pace to fast. The other problem we found is having the potentiometer being able to go backwards as well. By using a scroll wheel instead, the user could control the speed at which they are more comfortable with, a young or inexperienced with reading the braille language they could go at a slower rate, and for a more experienced user they can go at a faster speed.

Another idea that was proposed was to use piezoelectric material to act as the actuators instead of solenoids. The piezoelectric material takes up less space than the solenoids allowing us to achieve our goal of a small design. When an electric current is applied to the piezoelectric material, it would take form of a static shape. The drawback of using this material is that it is more expensive than the solenoids, which does not align with our goal of making it a cost-effective device and the other drawback was the amount of power that the piezoelectric consumes in order to operate. Another problem that we found was when the current was not running through the piezoelectric material, it took time for the material to relax, thereby taking more time between switching characters.

Even though we were pleased with the final product, we have already began looking into ways to improve our original design. One idea that we had was to have a battery

installed that could supply the device with power, allowing the user to use the device anywhere. This would make our device truly portable. Another idea that we wanted to explore was to have a mobile app that could connect to the device through Bluetooth. With the app, a helper or friend could wirelessly send PDF text to the device thereby not needing a computer every time to upload a file. One major improvement that we wanted was to use a PCB to house all the electronics instead of having a breadboard. This would make our device smaller and more compact, and there is less chance of wires or circuitry coming loose. Even though we did not use the potentiometer design to control the speed of the translation, we would still have to have that as an option along with the scroll wheel. Although the scroll wheel gives the user the freedom to read the text at their speed, scrolling through an entire book for example would be exhausting, so having a potentiometer that can be set to a continuous speed would benefit users reading large passages of text.

In conclusion, we were proud with what we accomplished at the end of the course. With this device, visually impaired users will be able to read online PDF and texts with a small compact device. For us, the goal of the project was met which was to build a cost effective, smaller and more portable device that could allow visually impaired individuals to read online documents.

Appendix

Python Text to Braille Conversion Code:

```
#import libraries
import RPi.GPIO as GPIO
import pygame
import time

#set up GPIO. This programm will use the BCM numbering
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

#def convert():
    #set up library for alphabet
dict = {}
dict[str('A')] = [1,0,0,0,0,0]#start of upper case letters
dict[str('B')] = [1,1,0,0,0,0]
dict[str('C')] = [1,0,0,1,0,0]
dict[str('D')] = [1,0,0,1,1,0]
dict[str('E')] = [1,0,0,0,1,0]
dict[str('F')] = [1,1,0,1,0,0]
dict[str('G')] = [1,1,0,1,1,0]
dict[str('H')] = [1,1,0,0,1,0]
dict[str('I')] = [0,1,0,1,0,0]
dict[str('J')] = [0,1,0,1,1,0]
dict[str('K')] = [1,0,1,0,0,0]
dict[str('L')] = [1,1,1,0,0,0]
dict[str('M')] = [1,0,1,1,0,0]
dict[str('N')] = [1,0,1,1,1,0]
dict[str('O')] = [1,0,1,0,1,0]
dict[str('P')] = [1,1,1,1,0,0]
dict[str('Q')] = [1,1,1,1,1,0]
dict[str('R')] = [1,1,1,0,1,0]
dict[str('S')] = [0,1,1,1,0,0]
dict[str('T')] = [0,1,1,1,1,0]
dict[str('U')] = [1,0,1,0,0,1]
dict[str('V')] = [1,1,1,0,0,1]
dict[str('W')] = [0,1,0,1,1,1]
dict[str('X')] = [1,0,1,1,0,1]
dict[str('Y')] = [1,0,1,1,1,1]
dict[str('Z')] = [1,0,1,0,1,1]
dict[str('a')] = [1,0,0,0,0,0]#start of lower case letters
dict[str('b')] = [1,1,0,0,0,0]
dict[str('c')] = [1,0,0,1,0,0]
dict[str('d')] = [1,0,0,1,1,0]
dict[str('e')] = [1,0,0,0,1,0]
dict[str('f')] = [1,1,0,1,0,0]
dict[str('g')] = [1,1,0,1,1,0]
dict[str('h')] = [1,1,0,0,1,0]
dict[str('i')] = [0,1,0,1,0,0]
dict[str('j')] = [0,1,0,1,1,0]
dict[str('k')] = [1,0,1,0,0,0]
dict[str('l')] = [1,1,1,0,0,0]
dict[str('m')] = [1,0,1,1,0,0]
```

```
dict[str('n')] = [1,0,1,1,1,0]
dict[str('o')] = [1,0,1,0,1,0]
dict[str('p')] = [1,1,1,1,0,0]
dict[str('q')] = [1,1,1,1,1,0]
dict[str('r')] = [1,1,1,0,1,0]
dict[str('s')] = [0,1,1,1,0,0]
dict[str('t')] = [0,1,1,1,1,0]
dict[str('u')] = [1,0,1,0,0,1]
dict[str('v')] = [1,1,1,0,0,1]
dict[str('w')] = [0,1,0,1,1,1]
dict[str('x')] = [1,0,1,1,0,1]
dict[str('y')] = [1,0,1,1,1,1]
dict[str('z')] = [1,0,1,0,1,1]
dict[str('numInd')] = [0,0,1,1,1,1] #this is used as the indicator when going
from letter to numbers
dict[str('1')] = [1,0,0,0,0,0]#start of numbers
dict[str('2')] = [1,1,0,0,0,0]
dict[str('3')] = [1,0,0,1,0,0]
dict[str('4')] = [1,0,0,1,1,0]
dict[str('5')] = [1,0,0,0,1,0]
dict[str('6')] = [1,1,0,1,0,0]
dict[str('7')] = [1,1,0,1,1,0]
dict[str('8')] = [1,1,0,0,1,0]
dict[str('9')] = [0,1,0,1,0,0]
dict[str('0')] = [0,1,0,1,1,0]
dict[str('puncInd')] = [0,0,0,1,1,1] #this is used as the indicator for
punctuations
dict[str(',')] = [0,0,1,0,0,0]#start of punctuation
dict[str(':')] = [0,1,0,0,1,0]
dict[str('!')] = [0,1,1,0,1,0]
dict[str('.')] = [0,1,0,0,1,1]
dict[str('?')] = [0,1,1,0,0,1]
dict[str(';')] = [0,1,1,0,0,0]


#set up the output pins
tl = 2
ml = 3
bl = 4
tr = 17
mr = 27
br = 22

GPIO.setup(tl, GPIO.OUT)
GPIO.setup(ml, GPIO.OUT)
GPIO.setup(bl, GPIO.OUT)
GPIO.setup(tr, GPIO.OUT)
GPIO.setup(mr, GPIO.OUT)
GPIO.setup(br, GPIO.OUT)

#GPIO Test
print("Confirming GPIO Setup...")
GPIO.output(tl, GPIO.HIGH)
GPIO.output(ml, GPIO.HIGH)
GPIO.output(bl, GPIO.HIGH)
GPIO.output(tr, GPIO.HIGH)
```

```
GPIO.output(mr, GPIO.HIGH)
GPIO.output(br, GPIO.HIGH)
time.sleep(1)
print("Confirming GPIO Setup......")
GPIO.output(tl, GPIO.LOW)
GPIO.output(ml, GPIO.LOW)
GPIO.output(bl, GPIO.LOW)
GPIO.output(tr, GPIO.LOW)
GPIO.output(mr, GPIO.LOW)
GPIO.output(br, GPIO.LOW)
time.sleep(1)
print("Confirming GPIO Setup.........")
print("GPIO Setup Complete!")

#opening target file
textFile = open('Text_File.txt', 'r')

#read all lines into stringFile array
stringFile = textFile.read().split('\n')

#determine number of lines in the file
numLines = len(stringFile)

#initialize array that will store each element in file
textFileArray = []
scrollDir = 0
textPos = -1
letterToPrint = ""

for i in range(numLines):
    for j in range (len(stringFile[i])):
        if (stringFile[i][j] == " " or stringFile[i][j] == "\x0c"):
            j=j+1
        else:
            textFileArray.append(stringFile[i][j])
            j=j+1
i=i+1

#initialize screen size for pygame
screen = pygame.display.set_mode((320,240))

while True:
    for e in pygame.event.get():
        if e.type == pygame.QUIT:
            pygame.quit()

        if e.type == pygame.MOUSEBUTTONDOWN:
            if((textPos==0 or textPos==-1) and e.button == 5):
                print("You are at the beginning of the document!")
                textPos = 0
                #PLAY AUDIO HERE
            if(textPos >= (len(textFileArray)-1) and e.button == 4):
                print("Reached end of document!")
                textPos = len(textFileArray) -1

            elif e.button == 4: #positive direction sequence
```

```
                if((ord(textFileArray[textPos])>=65 and
ord(textFileArray[textPos])<=122) and (ord(textFileArray[textPos+1])>=48 and
ord(textFileArray[textPos+1])<=58)):
                    print("numInd")
                    print('o'+"\t"+'x'+"\n"+'o'+"\t"+'x'+"\n"+'x'+"\t"+'x')
                    #GPIO.output(tl, GPIO.LOW)
                    #GPIO.output(ml, GPIO.LOW)
                    #GPIO.output(bl, GPIO.HIGH)
                    #GPIO.output(tr, GPIO.HIGH)
                    #GPIO.output(mr, GPIO.HIGH)
                    #GPIO.output(br, GPIO.HIGH)

                elif((ord(textFileArray[textPos+1])==44) or
(ord(textFileArray[textPos+1])==58) or (ord(textFileArray[textPos+1])==33) or
(ord(textFileArray[textPos+1])==46) or (ord(textFileArray[textPos+1])==63) or
(ord(textFileArray[textPos+1])==59)):
                    print("puncInd")
                    print('o'+"\t"+'x'+"\n"+'o'+"\t"+'x'+"\n"+'o'+"\t"+'x')
                    #GPIO.output(tl, GPIO.LOW)
                    #GPIO.output(ml, GPIO.LOW)
                    #GPIO.output(bl, GPIO.LOW)
                    #GPIO.output(tr, GPIO.HIGH)
                    #GPIO.output(mr, GPIO.HIGH)
                    #GPIO.output(br, GPIO.HIGH)




                letterToPrint = textFileArray[textPos+1]
                textPos = textPos + 1
            elif e.button == 5: #negative direction sequence
                if((ord(textFileArray[textPos])>=65 and
ord(textFileArray[textPos])<=122) and (ord(textFileArray[textPos-1])>=48 and
ord(textFileArray[textPos-1])<=58)):
                    print("numInd")
                    print('o'+"\t"+'x'+"\n"+'o'+"\t"+'x'+"\n"+'x'+"\t"+'x')
                    #GPIO.output(tl, GPIO.LOW)
                    #GPIO.output(ml, GPIO.LOW)
                    #GPIO.output(bl, GPIO.HIGH)
                    #GPIO.output(tr, GPIO.HIGH)
                    #GPIO.output(mr, GPIO.HIGH)
                    #GPIO.output(br, GPIO.HIGH)

                elif((ord(textFileArray[textPos-1])==44) or
(ord(textFileArray[textPos-1])==58) or (ord(textFileArray[textPos-1])==33) or
(ord(textFileArray[textPos-1])==46) or (ord(textFileArray[textPos-1])==63) or
(ord(textFileArray[textPos-1])==59)):
                    print("puncInd")
                    print('o'+"\t"+'x'+"\n"+'o'+"\t"+'x'+"\n"+'o'+"\t"+'x')
                    #GPIO.output(tl, GPIO.LOW)
                    #GPIO.output(ml, GPIO.LOW)
                    #GPIO.output(bl, GPIO.LOW)
                    #GPIO.output(tr, GPIO.HIGH)
                    #GPIO.output(mr, GPIO.HIGH)
                    #GPIO.output(br, GPIO.HIGH)
```

```
                letterToPrint = textFileArray[textPos-1]
                textPos = textPos - 1


            if letterToPrint in dict:
                if dict[letterToPrint][0] == 1:
                    #tl = 'x'
                    GPIO.output(tl, GPIO.HIGH)
                else:
                    #tl = 'o'
                    GPIO.output(tl, GPIO.LOW)
                if dict[letterToPrint][1] == 1:
                    #ml = 'x'
                    GPIO.output(ml, GPIO.HIGH)
                else:
                    #ml = 'o'
                    GPIO.output(ml, GPIO.LOW)
                if dict[letterToPrint][2] == 1:
                    #bl = 'x'
                    GPIO.output(bl, GPIO.HIGH)
                else:
                    #bl = 'o'
                    GPIO.output(bl, GPIO.LOW)
                if dict[letterToPrint][3] == 1:
                    #tr = 'x'
                    GPIO.output(tr, GPIO.HIGH)
                else:
                    #tr = 'o'
                    GPIO.output(tr, GPIO.LOW)
                if dict[letterToPrint][4] == 1:
                    #mr = 'x'
                    GPIO.output(mr, GPIO.HIGH)
                else:
                    #mr = 'o'
                    GPIO.output(mr, GPIO.LOW)
                if dict[letterToPrint][5] == 1:
                    #br = 'x'
                    GPIO.output(br, GPIO.HIGH)
                else:
                    #br = 'o'
                    GPIO.output(br, GPIO.LOW)
            #print(tl+"\t"+tr+"\n"+ml+"\t"+mr+"\n"+bl+"\t"+br)
            print(textPos)



    time.sleep(0.05)
    pygame.display.update()
```

GUI Code:

```
from textbox import *
#Output to EyeBraille
#from convertOutput import *
import subprocess
import threading

import pygame
import time
#File Dialog
import Tkinter
from Tkinter import *
from tkFileDialog import askopenfilename
#PDF
from cStringIO import StringIO
from pdfminer.pdfinterp import PDFResourceManager, PDFPageInterpreter
from pdfminer.converter import TextConverter
from pdfminer.layout import LAParams
from pdfminer.pdfpage import PDFPage
import os
import sys, getopt
#import os
#import pdfminer

pygame.init()

display_width = 800
display_height = 600

#Defined Colours
white = (255,255,255)
grey = (200,200,200)
light_grey = (125,125,125)
gold = (255,223,20)


GUI_Display = pygame.display.set_mode((display_width,display_height))
pygame.display.set_caption('Eye Braille')
clock = pygame.time.Clock()
exit_Condition = False



pygame.display.update()

##
def text_objects(text, font):
    textSurface = font.render(text, True, (0,0,0))
    return textSurface, textSurface.get_rect()
##

##
def message_display(text, font, font_size, rect):
    Text_Font = pygame.font.Font(font,font_size)
    TextSurf, TextRect = text_objects(text, Text_Font)
```

```
    TextRect.center = (rect[0]+rect[2]/2,rect[1]+rect[3]/2)
    GUI_Display.blit(TextSurf, TextRect)

    #time.sleep(2) #Display for 2 second
##

##Button
def button(msg, font, font_size, rect, ic, ac, action=None):
    mouse = pygame.mouse.get_pos()
    click = pygame.mouse.get_pressed()

    # Mouse Hover
    if rect[0]+rect[2] > mouse[0] > rect[0] and rect[1]+rect[3] > mouse[1] >
rect[1]:
        pygame.draw.rect(GUI_Display,ac,rect)
        if click[0] == 1 and action != None:
            action()
            action = None
    else:
        pygame.draw.rect(GUI_Display,ic,rect)

    message_display(msg,font,font_size,rect)
##

##Upload File
def upload_file():
    tk_root = Tk()
    tk_root.withdraw()

    result = askopenfilename(
        initialdir=tk_root,
        title="Select File",
        filetypes=[("PDF Documents", "*.pdf")],
    )

    if len(result)>0:
        print ("You chose %s" % result)
        #string = 'pdf2txt.py -o Text_File.txt -t tag %s' % result
        #os.system(string)
        text = convert(result,)
        textFile = open("Text_File.txt", "w") #make text file
        textFile.write(text) #write text to text file
##

##Convert PDF to Txt
def convert(fname, pages=None):
    if not pages:
        pagenums = set()
    else:
        pagenums = set(pages)

    output = StringIO()
    manager = PDFResourceManager()
    converter = TextConverter(manager, output, laparams=LAParams())
```

```
    interpreter = PDFPageInterpreter(manager, converter)

    infile = file(fname, 'rb')
    for page in PDFPage.get_pages(infile, pagenums):
        interpreter.process_page(page)
    infile.close()
    converter.close()
    text = output.getvalue()
    output.close
    return text
##

##Main Menu
def main_menu():

    global exit_Condition

    while not exit_Condition:
        pygame.display.update()
        clock.tick(15)
        GUI_Display.fill(gold)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit_Condition = True

        #message_display("Eye
Braille",'freesansbold.ttf',115,(0,0,display_width,display_height))
        logo = pygame.image.load('Eye_Braille.png')
        GUI_Display.blit(logo, (display_width/2-logo.get_width()/2,0))

        #Buttons/Menu Options
        button("Upload Content",'freesansbold.ttf',20,(display_width/2-
85,400,170,50),grey,light_grey, upload_content)
        button("Enter Text",'freesansbold.ttf',20,(display_width/2-
85,500,170,50),grey,light_grey, enter_text)
##

##Upload Content Menu
def upload_content():

    global exit_Condition

    while not exit_Condition:
        pygame.display.update()
        clock.tick(15)
        GUI_Display.fill(gold)

        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                exit_Condition = True

        #Instruction
```

```
        message_display("Select and upload a PDF
File:",'freesansbold.ttf',20,(0,0,display_width,display_height/8))

        button("Upload PDF",'freesansbold.ttf',20,(display_width/2-
75,100,150,50),grey,light_grey, upload_file)
        button("Back",'freesansbold.ttf',20,(50,500,100,50),grey,light_grey,
main_menu)
##

##Enter Text Menu
def enter_text():
    input_box = InputBox(GUI_Display, display_width/2-100, 100, 100, 32)
    input_boxes = [input_box]
    global exit_Condition



    while not exit_Condition:
        pygame.display.update()
        clock.tick(15)
        GUI_Display.fill(gold)

        for event in pygame.event.get():

            if event.type == pygame.QUIT:
                exit_Condition = True
            for box in input_boxes:
                box.handle_event(event)

        for box in input_boxes:
            box.update()

        for box in input_boxes:
            box.draw()


        #os.system("python2 convertOutput.py &")
        #subprocess.Popen(['./convertOutput.py'])
        #Instruction
        message_display("Enter text to be displayed on
EyeBraille",'freesansbold.ttf',20,(0,0,display_width,display_height/8))

        button("Back",'freesansbold.ttf',20,(50,500,100,50),grey,light_grey,
main_menu)

        #NOTE: Exclude some special characters
##

main_menu()
pygame.quit()
quit()
```