A Text-to-Braille and Text-to-Audio Translating Device

Sun Kwon, The Cooper Union for the Advancement of Science and Art
Sun Kwon is a senior majoring in Mechanical Engineering at the Cooper Union for the Advancement of Science and Art

Soyoung Moon, The Cooper Union for the Advancement of Science and Art
Soyoung Moon is a senior majoring in Mechanical Engineering at the Cooper Union for the Advancement of Science and Art

May 4, 2017
ME394: Senior Capstone Mechanical Engineering Design
Senior Advisors: Professor R. Dell, Professor S. Wei

Table of Contents

Abstract

The extent of technological advancements permits additional information to be assimilated beyond what is traditionally available for the visually impaired from just listening to audio or reading Braille. Recent progresses in video digitization and computer interfacing now provide the technology needed to scan printed text and then output it in Braille texture or audio [1]. Initial research and interviews conducted with optical research professionals clarified specific client needs to combine both formats of verbal communication in order to provide the end user with additional choices. Research has provided insight that there are no devices and prototypes currently out on the market that output both audio and Braille on a singular device. The design iterations created a working final product that produces one audio output capable of reading entire sentences and one reading output of Braille texture up to three letters long.

Introduction

Braille

Braille is a code to communicate through reading and writing for people who have total or partial visual impairment. It was first created by Louis Braille, a French innovator who lost his eyesight in a childhood accident. The first form of Braille was in French and included musical notation, and later when Braille was further developed, it was formally adopted as another form of writing.

The visually impaired can read by moving their fingers left-to-right on a system of raised dots. Each modern day Braille cell contains six dots arranged in a 3x2 array, which allows for 64 unique dot combinations. Figure 1 below shows a picture of the Braille forms of lower-case alphabets. Other combinations are possible in order to make upper-case letters, numbers, and punctuation marks with those six dots [6].

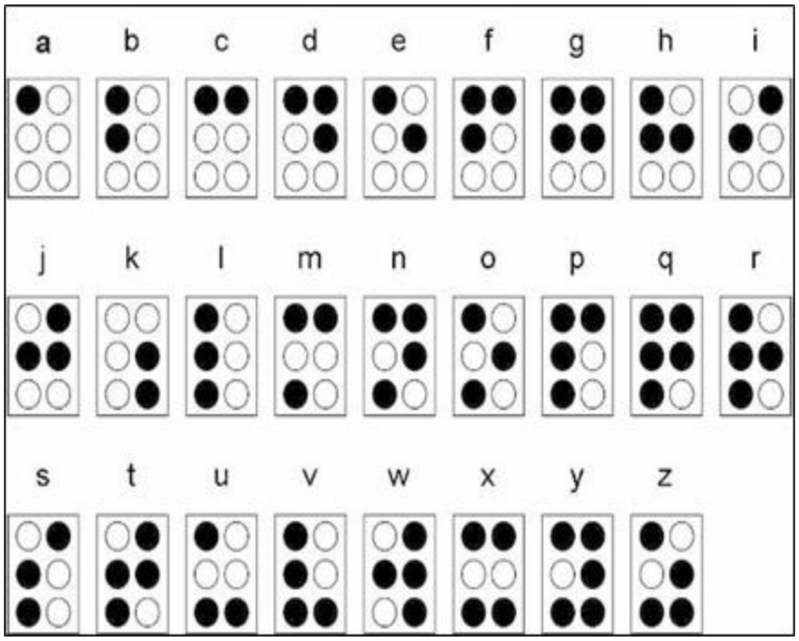


Figure 1. Braille for Lower-Case Alphabets

Motivation

Through research, the team has learned that the visually impaired are limited to two formats of reading: audio and Braille [4, 5]. Those who are very fluent in Braille prefer Braille; those who have never become fluent in the dot system prefer to use audio because many people experiencing vision loss are older and many of them do not want to learn a whole new language [2, 3]. Thus, the visually impaired use Braille and audio in equal measures to read and write. Further research has provided insight to the team that existing products out on the market provide the visually impaired with only one format of reading. Thus, the project's goal is to provide a device for them that is simple and easy to use.

Problem Statement

The proposed solution presents a device that translates printed text to a preferred format (Braille or audio) so that the users can choose within a single device instead of needing two separate devices. With the translating device, the visually impaired and specialized schools for the blind can have increased accessibility to reading resources. The device can also serve as a teaching aid for those interested in becoming conversant in Braille.

System Design

For the Fall of 2016, the team created a working prototype of a single enlarged Braille cell that consists of six solenoids and six pins. Figure 2 below shows a picture of a single Braille cell.

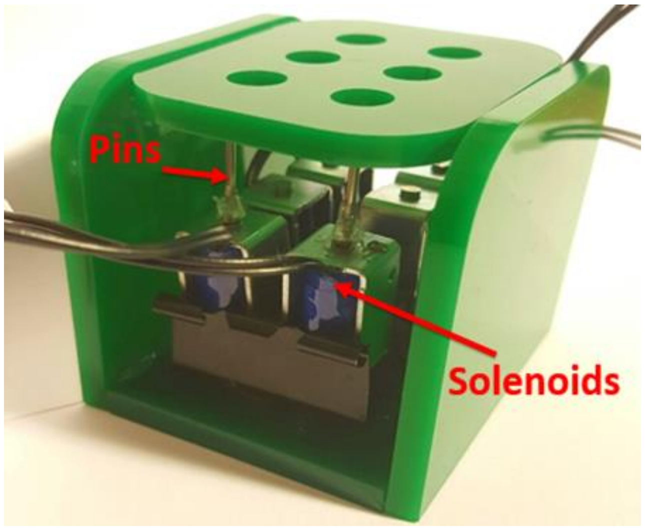


Figure 2. Prototype of a Single, Enlarged Braille Cell

For the Spring of 2017, the team has created a working prototype of the text-to-Braille and text-to-audio translator. Figure 3 below shows a picture of the final product of the spring semester.

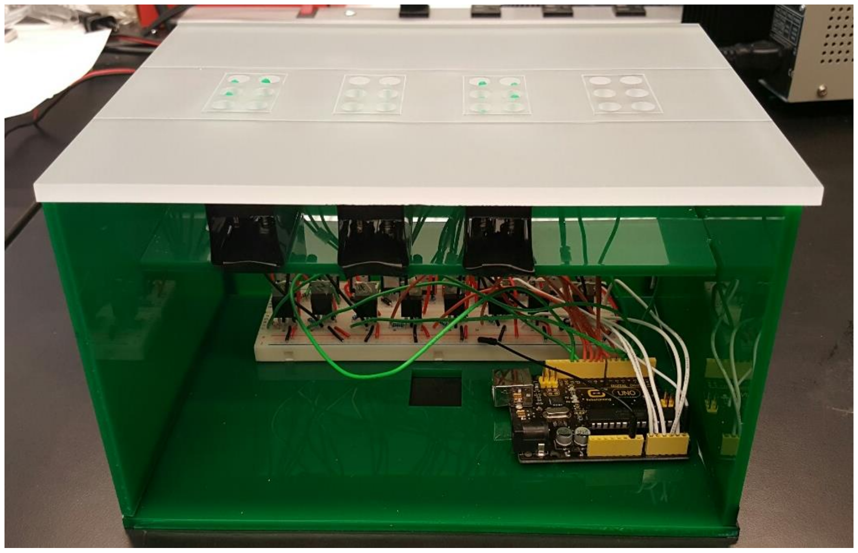


Figure 3. Final Product of the Translator

Mechanical Components

Figure 4 below shows the inside of the final product, which contains all the necessary components that permits the coding to run for the translation to Braille and audio.

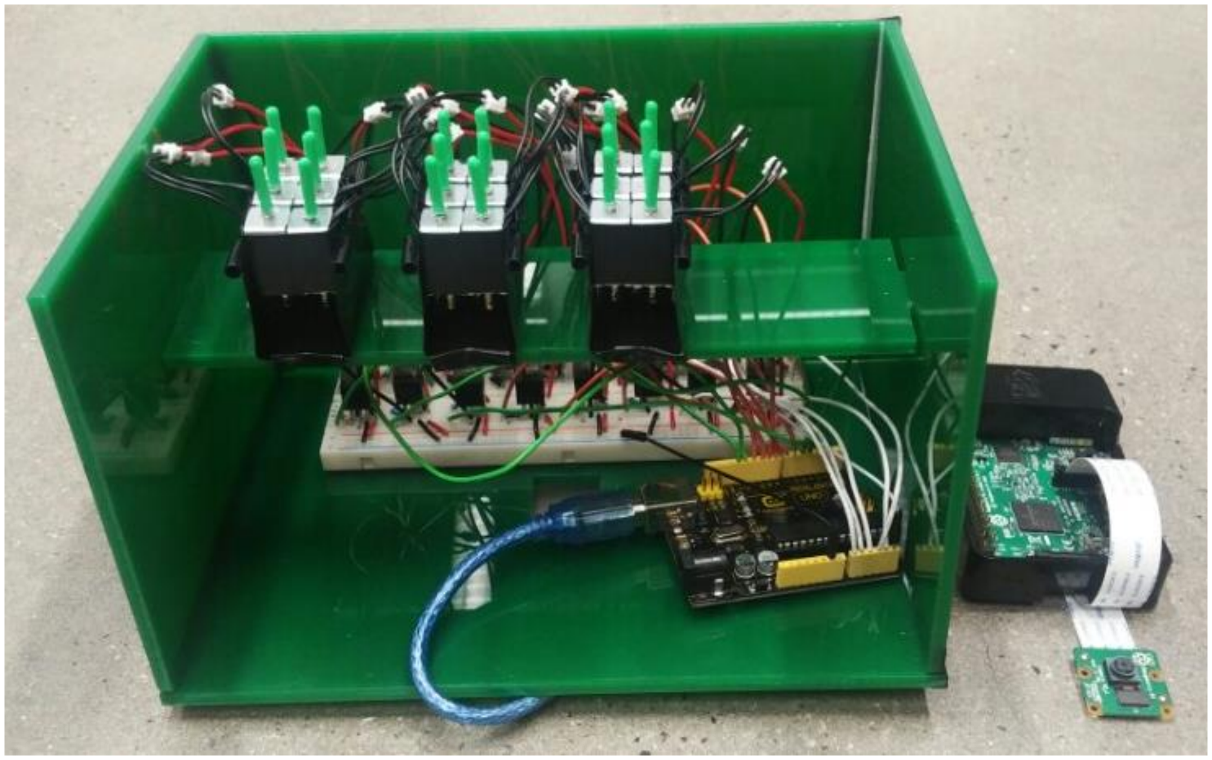


Figure 4. Inside of the Final Product

All of the coding has been done on the Arduino Uno and Raspberry Pi 3 Model B; a microcontroller and tiny computer have both been utilized in order to increase the number of output pins the team can work with so that more solenoids can be included. Just using a Raspberry Pi was not enough to connect enough solenoids to make three Braille cells; thus, an Arduino Uno was also integrated into the final product.

On the platform of the device are three Braille cells. Each cell is essentially three of the single enlarged prototypes shown in Figure 2. However, the six solenoids per cell are compacted in a way that sized the cell down a slight amount. The green pins that have been attached to each solenoid are 3D printed.

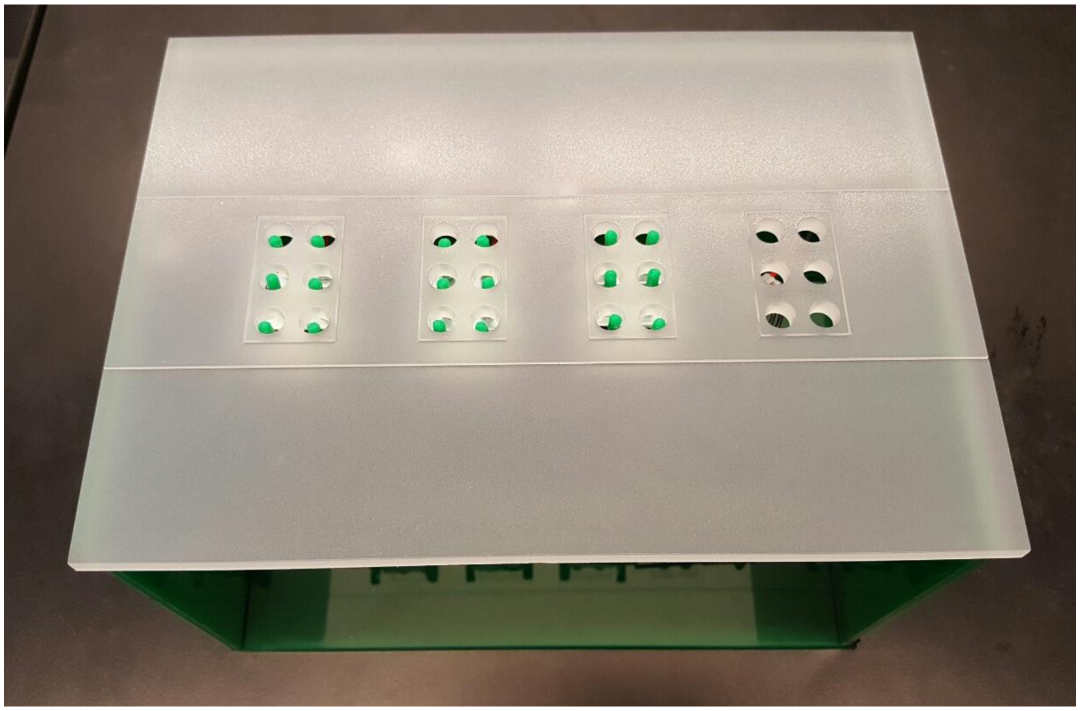


Figure 5. Device Surface that shows Holes for the Braille Pins

Figure 5 above shows the surface of the device. The exterior of the device is made with acrylic; however, white acrylic was used instead of green for the surface of the device in order to emphasize the outlines of the three Braille cells underneath the surface. A total of 24 holes were lasercut (six holes per cell), but a total of 18 holes were used for the testing (three cells).

Programming

Last semester, the team focused on the overall big picture of the device. Figure 6 shows the initial flow chart that the team came up during the previous semester before trying to connect different steps to each other.

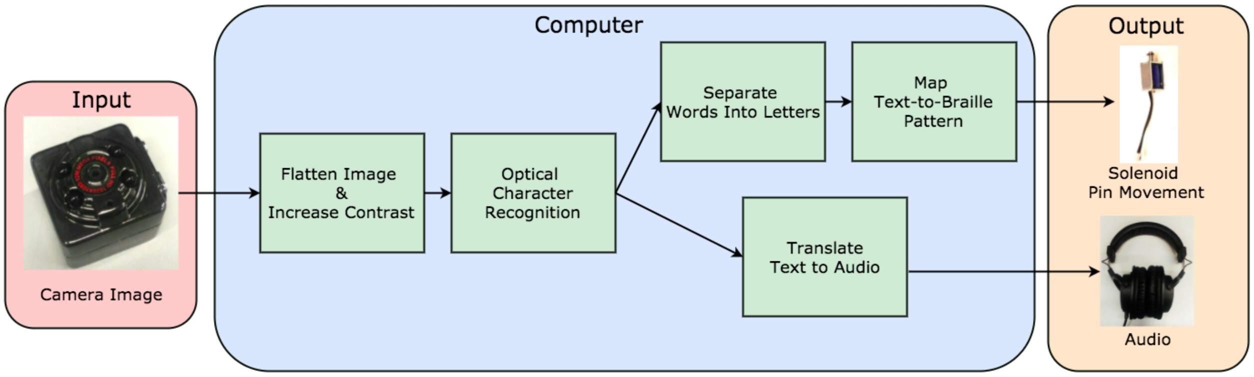

Input
Camera Image
Computer
Flatten Image & Increase Contrast
Optical Character Recognition
Separate Words Into Letters
Map Text-to-Braille Pattern
Translate Text to Audio
Output
Solenoid Pin Movement
Audio

Figure 6. Initial Flow Chart

The input of the device in the above flow chart is the image file from the SQ8 mini camera and the outputs are the audio from the speaker and the movement of the pins that represent Braille letters. At this point in time, the team was thinking of using just an Arduino Uno to implement the whole system. The team was using the open source software *tesseract* for image improvement and OCR and was using a C++ code written by the team to separate word strings into letters and map out the printed letters to Braille letters in binary form. In addition, to signal the solenoids, the team was using a simple Arduino code that turns the solenoids on and off.

At first, to connect the “printed text-to-binary number conversion” process to the Arduino that signals the solenoids, the team tried to connect the C++ code that contains text-to-Braille letter mapping data directly to the Arduino code. However, connecting a C++ code directly to an Arduino code through the serial port required both the Raspberry Pi and the Arduino to detect each other. After a few trials, the team figured out that it would be easier to make the C++ code create a text file that saves all the output data from the C++ code and then make the Arduino simply read off from the newly formed text file to get the data.

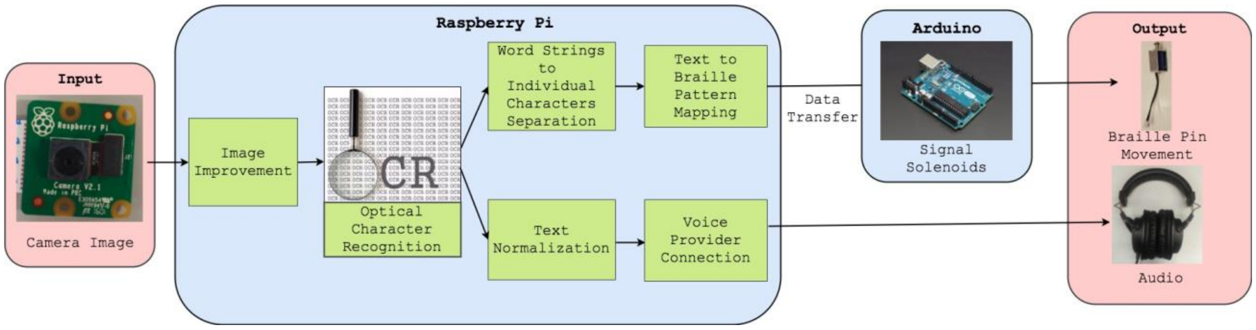

Input
Camera Image
Raspberry Pi
Image Improvement
Optical Character Recognition
Word Strings to Individual Characters Separation
Text to Braille Pattern Mapping
Text Normalization
Voice Provider Connection
Data Transfer
Arduino
Signal Solenoids
Output
Braille Pin Movement
Audio

Figure 7. Final Flow Chart

During the Spring of 2017, the team started to implement the above idea, which is to create a text file before connecting the Arduino to the overall code, and made a new flow chart as shown in Figure 7. Initially, in order to have enough space to save text files that contain data, the team used a Raspberry Pi instead of an Arduino as the device's main computer, because Raspberry Pi would be enough to implement all the steps for the device. However, the team figured out that the Raspberry Pi's General-Purpose Input/Output (GPIO) system only included seventeen output pins, which were even less than the pins needed to connect three Braille cells. Therefore, the team decided to use an Arduino in addition to a Raspberry Pi in order to increase the number of the output pins to connect to the solenoids.

As shown in the final flow chart, once the device is turned on, the Raspberry Pi Camera Module automatically takes a picture of the printed text and saves it as a TIF image file. This was possible through the use of an image taking software called *raspistill*. Then, the same image file goes through image improvement, such as increasing the contrast so that the computer can easily recognize where the text is located in the image. Next, the image goes through OCR which converts the TIF image file to a text file. Once there is a text file, depending on what the user chooses, the device either outputs Braille patterns or audio.

For the text-to-Braille conversion, first, the text word strings are separated into individual letters. Then, each character is translated into its corresponding Braille letter in binary form. The binary number data is first saved in the Raspberry Pi as a text file. For example, letter "n" will be converted to the binary 6-digit number, "110110", which means that at the end of the flow process, the first, second, fourth, and fifth pins attached to the solenoids will go up to represent the letter "n". Then, all the Raspberry Pi data saved in the text file is transferred to an Arduino Uno. For this data transfer, the Arduino reads off from the text file through the help of a Python code that connects the Arduino with the Raspberry Pi. Once the Arduino receives all the binary number data, it then signals the solenoids to push up for number “1” and stay down for number “0”.

For the next step, the team implemented the text-to-audio conversion. The team started out this process by directly typing in the sentences that the team wanted to translate onto the terminal and executing a voice provider program called *Flite*. Making the device read off from an image file without typing the text directly into the system was more difficult. Thus, the team used the text file that was already saved in the Raspberry Pi after OCR and made *Flite* read off from the text file.

Below is the step-by-step explanation of the whole process in more detail. First, the camera module takes a picture of the text. Figure 8 is an example image file before OCR.

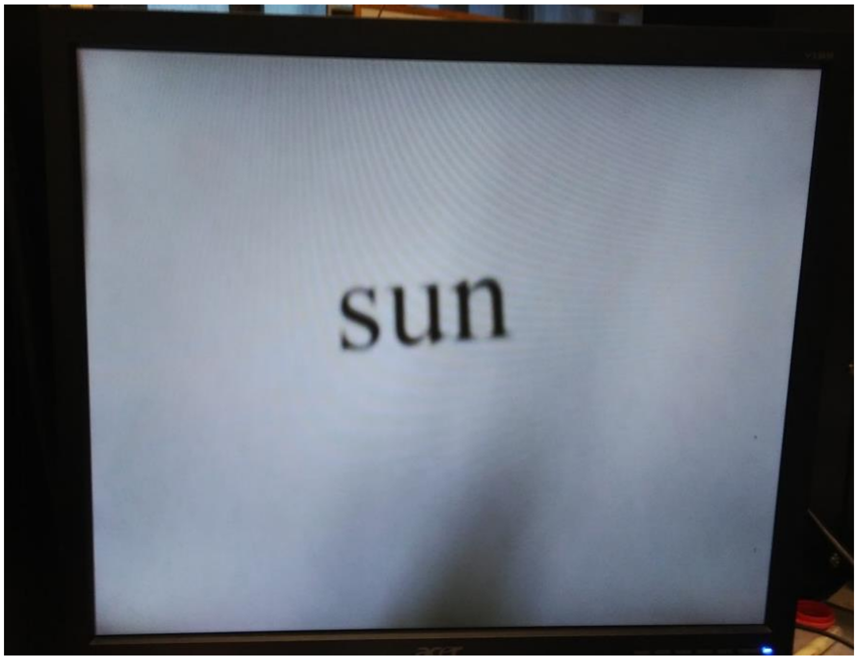
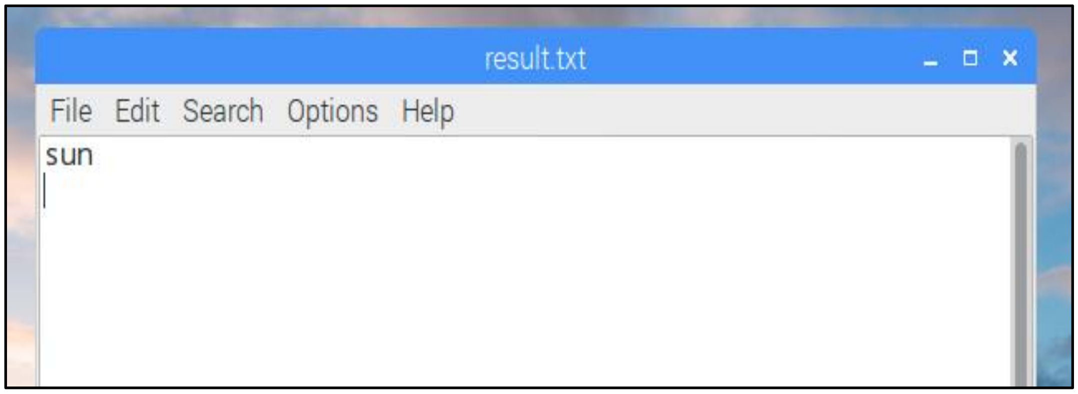


Figure 8. Camera Image before OCR

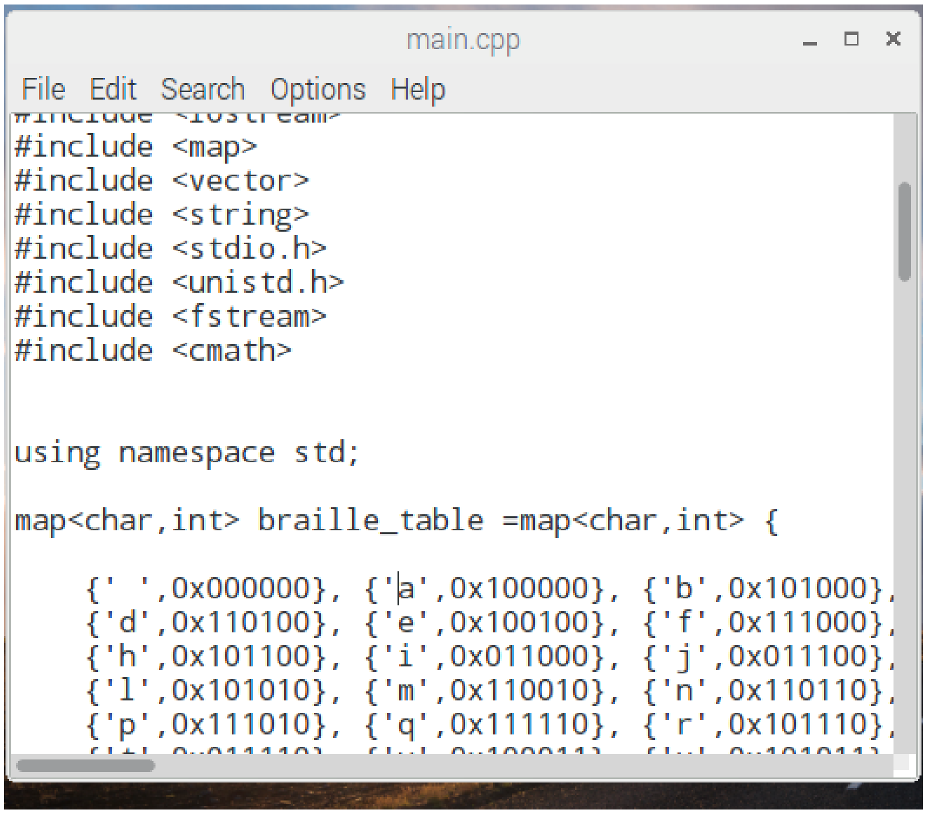
Then, the image file goes through OCR which converts the image to a text file and saves the text file to the Raspberry Pi. Figure 9 is an example text file saved in the Raspberry Pi after OCR.



sun

Figure 9. Text File after OCR

If the user chooses Braille as his or her preferred reading method, the C++ code shown below opens the text file from the above step and separates all the word strings into individual characters.

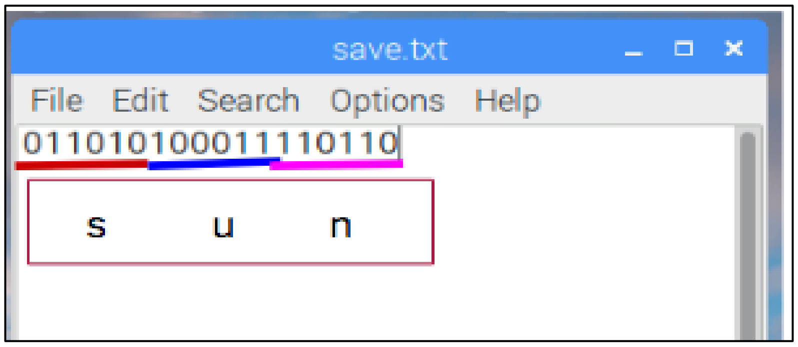


```
#include <map>
#include <vector>
#include <string>
#include <stdio.h>
#include <unistd.h>
#include <fstream>
#include <cmath>


using namespace std;

map<char,int> braille_table =map<char,int> {

    {' ',0x000000}, {'a',0x100000}, {'b',0x101000},
    {'d',0x110100}, {'e',0x100100}, {'f',0x111000},
    {'h',0x101100}, {'i',0x011000}, {'j',0x011100},
    {'l',0x101010}, {'m',0x110010}, {'n',0x110110},
    {'p',0x111010}, {'q',0x111110}, {'r',0x101110},
```

Figure 10. C++ Code for Printed Text-to-Braille Mapping

The mapping shown in the C++ code above converts each letter to its corresponding Braille letter in binary form. In the above image, it can be seen that all the numbers are represented as hexadecimals. Hexadecimal is a positional numeral system with a base of 16 and in C++ coding, hexadecimal is represented by putting “0x” in front of the number. The team used hexadecimals because they typically take up less memory space than a binary or digit system. However, for this project, since the team is only executing the numbers as strings, the above consideration becomes trivial. Therefore, the future teams can change this map to be based on the binary number system to make calculations straightforward.

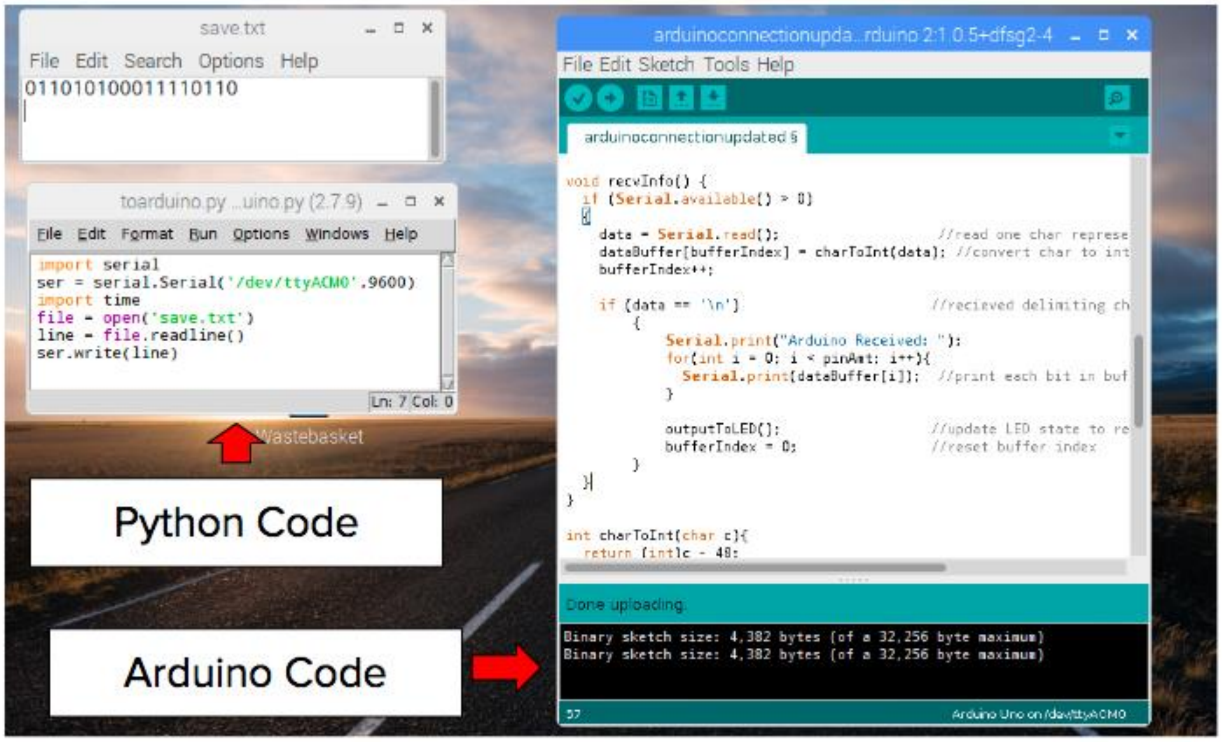
Once the C++ code converts all the characters to the corresponding Braille letters in binary form, the same C++ code saves the result as a text file in Raspberry Pi as shown in Figure 11.



```
011010100011110110
```

Figure 11. Saved Text File that shows Braille Letters in Binary Form

The next step is to transfer all the data saved in the saved text file to an Arduino. To do this, the team wrote a Python code that opens up the text file containing the binary numbers and sends the data line-by-line to the Arduino. Figure 12 below shows the Python code that sends the data and the Arduino code that receives the data for the project.


save.txt
File Edit Search Options Help
toarduino.py ...uino.py (2.7.9)
File Edit Format Run Options Windows Help
Ln: 7 Col: 0
Wastebasket
Python Code
Arduino Code
arduinoconnectionupda...rduino 2:1.0.5+dfsg2-4
File Edit Sketch Tools Help
arduinoconnectionupdated §
Done uploading.
57
Arduino Uno on /dev/ttyACM0

```
011010100011110110
```

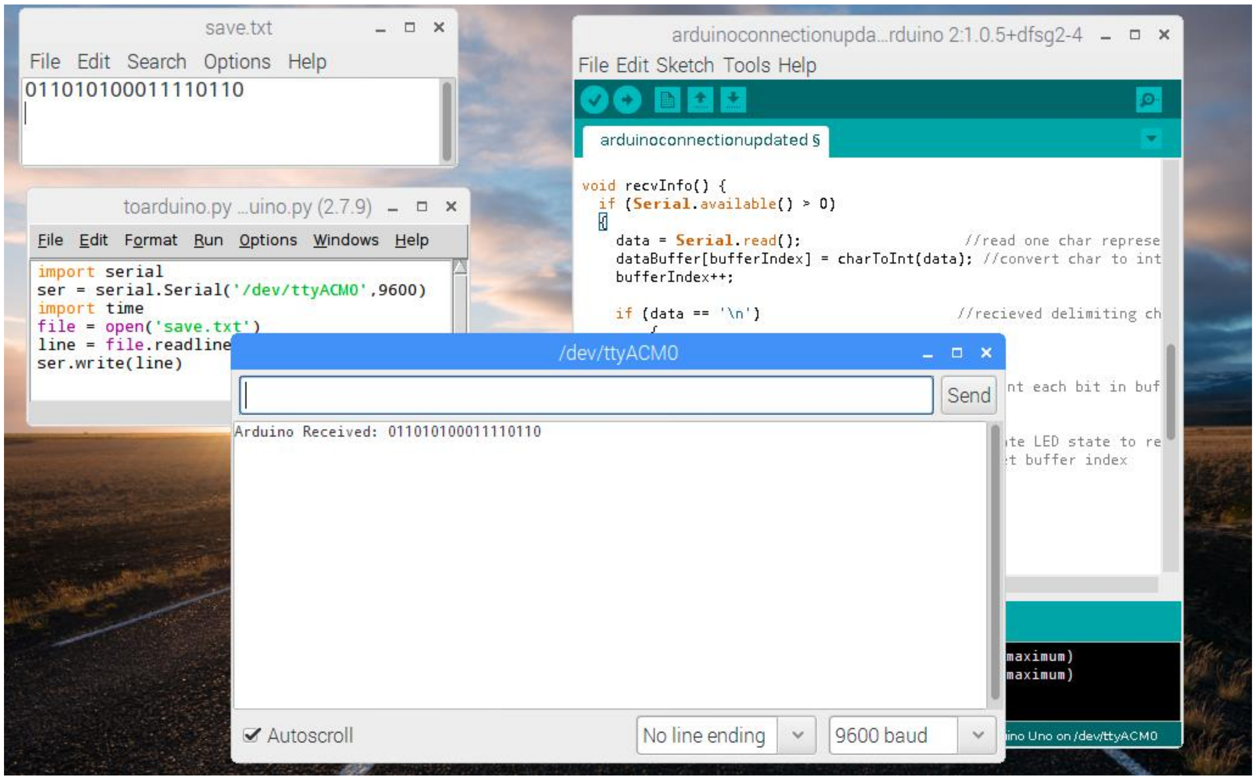
```python
import serial
ser = serial.Serial('/dev/ttyACM0'.9600)
import time
file = open('save.txt')
line = file.readline()
ser.write(line)
```

```cpp
void recvInfo() {
  if (Serial.available() > 0)
  {
    data = Serial.read();                            //read one char represe
    dataBuffer[bufferIndex] = charToInt(data); //convert char to int
    bufferIndex++;

    if (data == '\n')                               //recieved delimiting ch
      {
        Serial.print("Arduino Received: ");
        for(int i = 0; i < pinAmt; i++){
          Serial.print(dataBuffer[i]);  //print each bit in buf
        }

        outputToLED();                              //update LED state to re
        bufferIndex = 0;                            //reset buffer index
      }
  }
}

int charToInt(char c){
  return (int)c - 48;
```

```
Binary sketch size: 4,382 bytes (of a 32,256 byte maximum)
Binary sketch size: 4,382 bytes (of a 32,256 byte maximum)
```

Figure 12. Python and Arduino code for Data Transfer

Once the Arduino receives the data from the text file saved in the Raspberry Pi, the Arduino then prints out the received data on its serial monitor. The team created this step to confirm that all the data were transferred to the Arduino from the Raspberry Pi. Figure 13 below shows the serial monitor of the Arduino and the team can check that the number that the Arduino received exactly matches the number saved in the Raspberry Pi's text file. In addition, to make data transfer easier, the team converted the number which was originally in hexadecimal to a simple string before initiating the transfer.


save.txt
File Edit Search Options Help
toarduino.py ...uino.py (2.7.9)
File Edit Format Run Options Windows Help
arduinoconnectionupda...rduino 2:1.0.5+dfsg2-4
File Edit Sketch Tools Help
arduinoconnectionupdated §
/dev/ttyACM0
Send
Autoscroll
No line ending
9600 baud
ino Uno on /dev/ttyACM0

```
011010100011110110
```

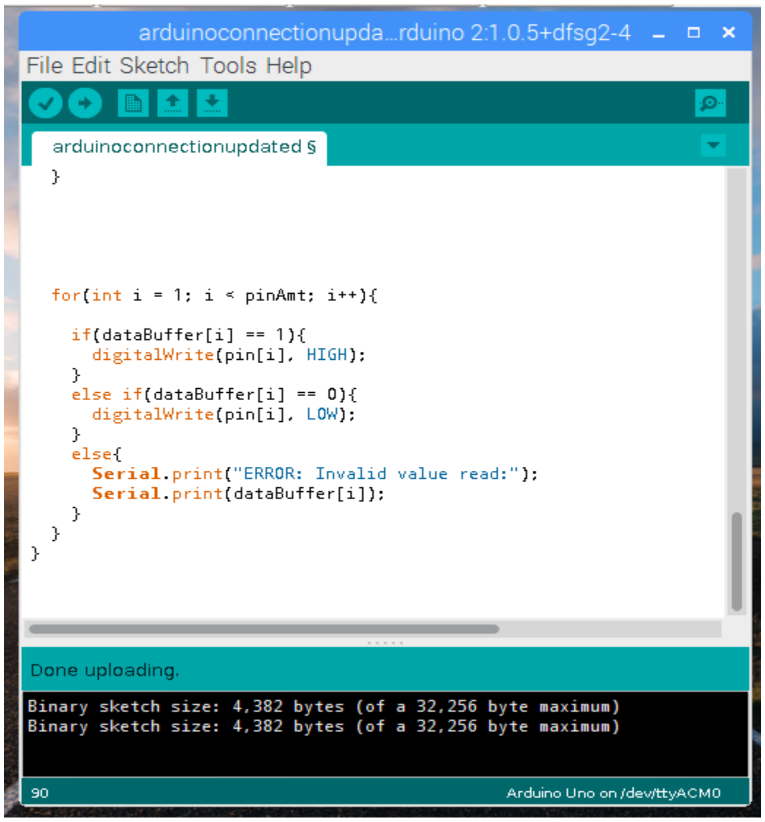
```
import serial
ser = serial.Serial('/dev/ttyACM0',9600)
import time
file = open('save.txt')
line = file.readline
ser.write(line)
```

```
void recvInfo() {
  if (Serial.available() > 0)
  {
    data = Serial.read();                           //read one char represe
    dataBuffer[bufferIndex] = charToInt(data); //convert char to int
    bufferIndex++;

    if (data == '\n')                              //recieved delimiting ch
```

```
Arduino Received: 011010100011110110
```

Figure 13. Python and Arduino Code with a Serial Monitor

After the Arduino receives all the data, through simple lines of code shown in Figure 14, the Arduino signals all the solenoids connected to the number "1" from the data to push up and the ones connected to the number "0" to stay down in order to form Braille patterns.


arduinoconnectionupda...rduino 2:1.0.5+dfsg2-4
File Edit Sketch Tools Help
arduinoconnectionupdated §

```
  }



  for(int i = 1; i < pinAmt; i++){

    if(dataBuffer[i] == 1){
      digitalWrite(pin[i], HIGH);
    }
    else if(dataBuffer[i] == 0){
      digitalWrite(pin[i], LOW);
    }
    else{
      Serial.print("ERROR: Invalid value read:");
      Serial.print(dataBuffer[i]);
    }
  }
}
```

```
Done uploading.
Binary sketch size: 4,382 bytes (of a 32,256 byte maximum)
Binary sketch size: 4,382 bytes (of a 32,256 byte maximum)
```

90
Arduino Uno on /dev/ttyACM0

Figure 14. Arduino Code for Braille Pin Signaling

The green rectangle below represents one Braille cell. A typical Braille cell contains six Braille dots, which are represented as yellow circles in Figure 15 below. The numbers inside the yellow circles represent which digit of the binary 6-digit number corresponds to the specific Braille dot.

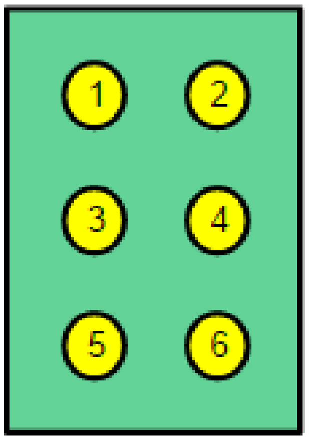


Figure 15. A Simplified Drawing of a Typical Braille Cell

Figure 16 below shows how the Braille patterns will be executed. Letter “s” corresponds to the Braille letter in binary number, “011010”; to represent “s” as Braille, the second, third, and fifth pins will be pushed up by the solenoids. Letter “u” corresponds to the Braille letter in binary number, “100011”; the first, fifth, and sixth pins will be pushed up. Letter “n” corresponds to the Braille letter in binary number, “110110”; the first, second, fourth, and fifth pins will be pushed up. The whole set of three letters will be executed all at once without any delay between the letters so that the user does not need to wait for the next letter.

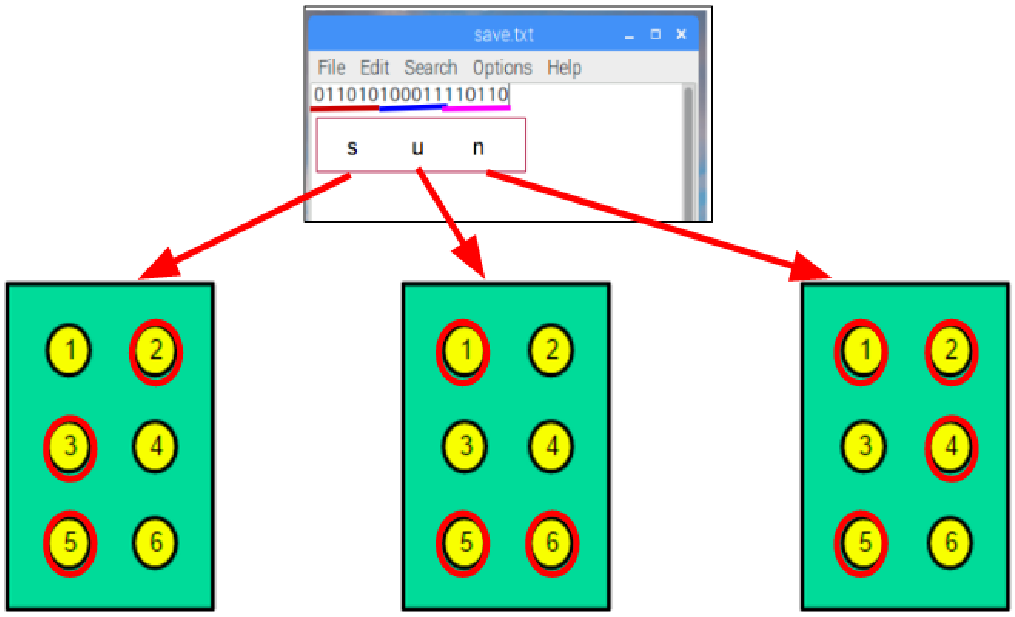
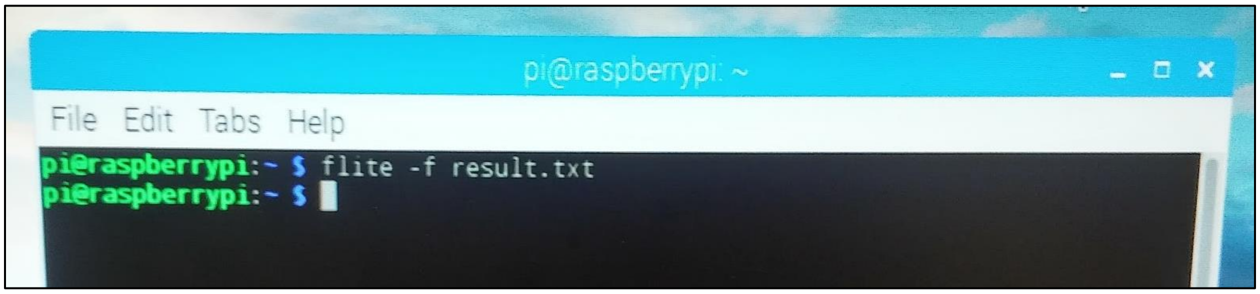


Figure 16. Braille Letters in Binary Form translated to Braille Patterns

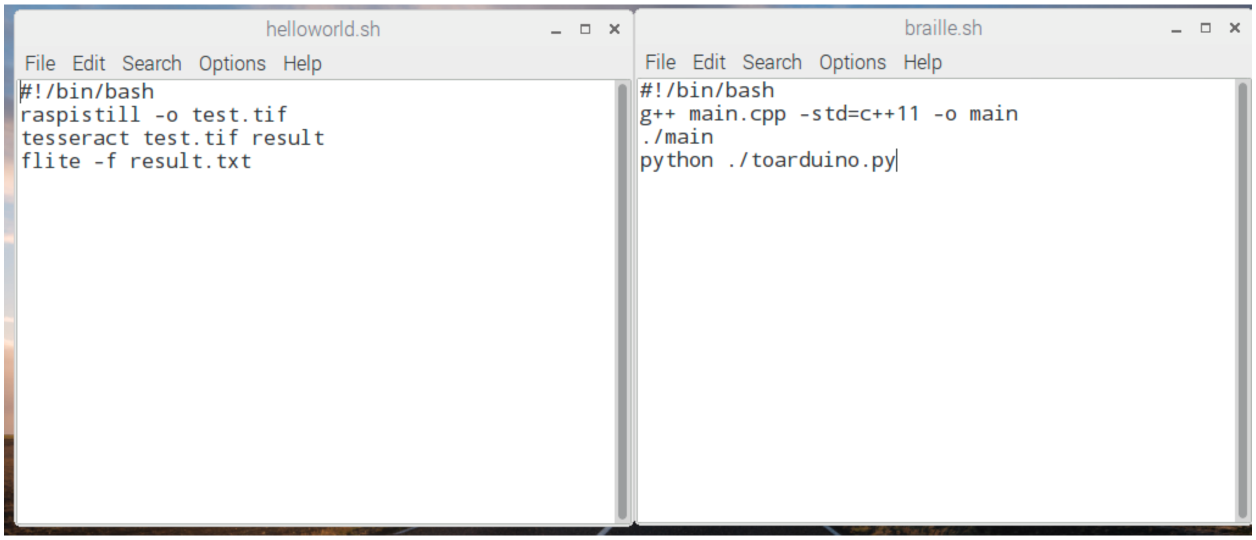
If the user uses audio as his or her preferred reading method, the process becomes much simpler. As shown in Figure 17, the team calls for the software *Flite* with a line of code, which normalizes text in the saved text file by converting raw numbers and abbreviations into equivalent written out words and provides a voice to the updated file after text normalization. For example, the phrase "1 solenoid" will be converted to "one solenoid" and the speaker will read out "one solenoid" out loud for the user.



```
pi@raspberrypi:~ $ flite -f result.txt
pi@raspberrypi:~ $
```

Figure 17. Printed Text to Audio Code

Connecting all the steps and making it into one flow process was another obstacle. To achieve the above, the team used bash scripts. Bash scripts are simple text files that contain different commands that would normally be typed directly on the terminal [7]. Figure 18 shows two different bash scripts that the team wrote for the project. The file on the left is a bash script for the OCR and the text-to-audio conversion. The file on the right is a bash script for the text-to-Braille conversion.



```
#!/bin/bash
raspistill -o test.tif
tesseract test.tif result
flite -f result.txt
```

```
#!/bin/bash
g++ main.cpp -std=c++11 -o main
./main
python ./toarduino.py
```

Figure 18. Bash Scripts for the Project

The script on the left contains the software *raspistill*, which signals the camera module to take a picture. It also contains the softwares *tesseract* and *flite*, which are directly used for the OCR and the text-to-audio conversion. In addition, the name of the specific files and file types that the team used for the project are shown. The first line from the script on the left shows that after the use of the software *raspistill*, the result is saved as “test.tif”. The second line shows that the TIF image file “test.tif” is then converted to a text file named “result” through OCR. The third line shows that the software *flite* provides a voice to the “result.txt” file.

The right column of the bash script shows that the C++ code file and Python code file are used for the text-to-Braille conversion. The first line shows that a C++ file called “main.cpp” is compiled and executed in C++ Version 11 by the use of the command, “g++”. The second line shows that a Python file named “toarduino.py” is executed.

Using the above bash scripts, all the processes explained in the above paragraph run step-by-step just by typing “./ helloworld.sh” (for the left script) and “./ braille.sh” (for the right script) on the terminal. Once the team implements an “on” button and a switch that allows the user to choose either Braille or audio, the button and the switch can be directly connected to these bash scripts. Therefore, the whole flow process will start and be executed just by pushing a button or changing the setting of a switch.

Camera Image Testing

To iterate what was accomplished during the Fall of 2016 for the camera image testing, the smartphone camera of a LG G Vista2 was used to take a picture of a portion of a book in order to test the OCR code. Figure 19 shows what the image looks like, shadows and bending of the page included.

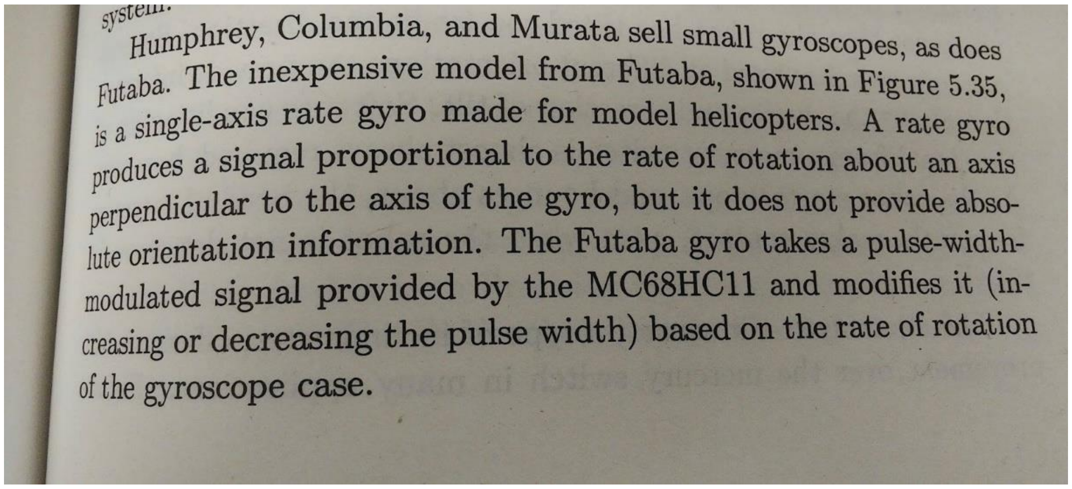

system.
Humphrey, Columbia, and Murata sell small gyroscopes, as does Futaba. The inexpensive model from Futaba, shown in Figure 5.35, is a single-axis rate gyro made for model helicopters. A rate gyro produces a signal proportional to the rate of rotation about an axis perpendicular to the axis of the gyro, but it does not provide absolute orientation information. The Futaba gyro takes a pulse-width-modulated signal provided by the MC68HC11 and modifies it (increasing or decreasing the pulse width) based on the rate of rotation of the gyroscope case.

Figure 19. Phone Camera Image File

The image file was automatically saved as a JPG file so it had to be converted into a TIF file through coding. Then, the image file went through OCR and was converted to the following text file shown in Figure 20.

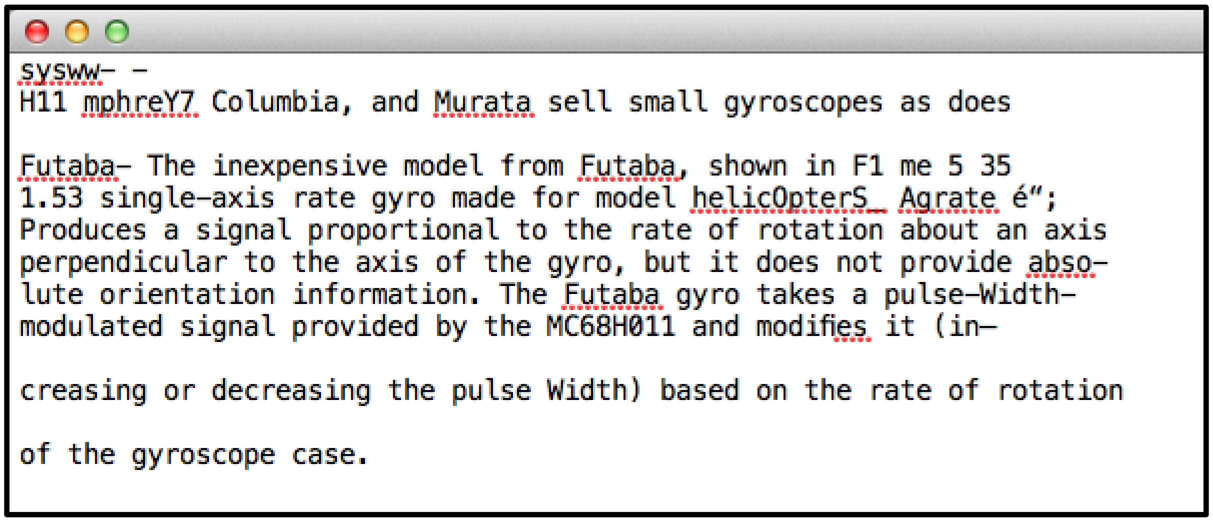
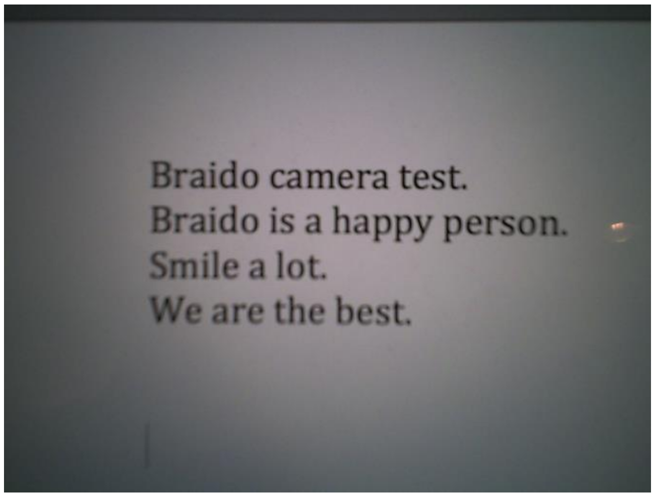

```
syswW- -
H11 mphreY7 Columbia, and Murata sell small gyroscopes as does

Futaba- The inexpensive model from Futaba, shown in F1 me 5 35
1.53 single-axis rate gyro made for model helicOpterS_ Agrate é";
Produces a signal proportional to the rate of rotation about an axis
perpendicular to the axis of the gyro, but it does not provide abso-
lute orientation information. The Futaba gyro takes a pulse-Width-
modulated signal provided by the MC68H011 and modifies it (in-

creasing or decreasing the pulse Width) based on the rate of rotation

of the gyroscope case.
```

Figure 20. Imaged File Converted into Text File through OCR

For the majority of the the text, it is quite accurately translated. However, it can be seen that the text is not translated perfectly where the book is not flat. For example, the part where it says “is a single axis” is not translated accurately. In addition, some parts of the sentences that are shadowed are not recognized nor translated accurately.

Afterwards, the team conducted a more realistic camera test using the SQ8 mini camera. Figure 21 shows what the image looks like, shadows and bending of the page included.



Braido camera test.
Braido is a happy person.
Smile a lot.
We are the best.

Figure 21. SQ8 Mini Camera Image File

Compared to the phone camera, the SQ8 mini camera has a lower resolution and therefore, the image is less accurate and has more shadowed parts.

During the Spring of 2017, the team decided to use a Raspberry Pi Camera Module V3 instead of the SQ8 since it was easier to code the camera module to automatically take pictures as a TIF image file and save them in certain locations. Under the section “Programming”, Figure 8 shows the word “sun” and is an image taken by the camera module. The camera quality is similar to that of the SQ8’s, and the module is highly sensitive to vibration and distance. As a solution, the team is in the process of building a stand at a certain height to fix the camera in one position and make it easier for the software *tesseract* to detect text in image files.

Conclusion

The final design of the team’s translating device is a rectangular-shaped box that fits three Braille cells. The user will be able to choose the preferred reading format by using a switch connected to a bash script for either printed text-to-Braille or printed text-to-audio conversion. This design is a result of several iterations. Originally, the team had thought of implementing only the text-to-audio function. However, after several interviews, the team was given insight that although the number of Braille users is decreasing, those who are fluent in Braille prefer to use Braille over audio. Therefore, the team decided to implement both text-to-Braille and text-to-audio functions in the device to ultimately improve the accessibility to reading materials for the visually impaired. Determining the size of the device also went through several iterative thinking processes. To make the operation of the device as simple as possible, the team had thought of making the device bigger and fit about half the length of regular paper so that there is no need for the user to move the device line-by-line; instead , the user would only have to move the device once. However, one of the design goals for the translating device was to make it portable and therefore, the team decided not to increase the size of the device. As of now, the prototype is bigger than the originally planned size of the device but when the team uses smaller solenoids, the device will get smaller as well as fit the width of a regular 8.5”x11” paper.

The team concluded the work with a laser cut acrylic prototype that fits three enlarged Braille cells and a completed flowchart that includes the OCR, the printed text-to-Braille conversion code, and the printed text-to audio-conversion code, which were simplified using bash scripts. The prototype works as proof of concept and will be further developed in the future.

Future Works

For the final product, the text-to-Braille and text-to-audio translations work through coding. One aspect to improve the device is to add a button to turn the device on and off and a switch that allows the user to go back and forth from audio and Braille. Another aspect to be worked on is to consider using smaller solenoids of good quality in order to make the device smaller and affordable. Moreover, in order to fully complete the flow process of the translation, the team will improve the OCR coding so that the conversion from the TIF file (camera module) to the text file occurs without any errors; it is expected that the device will better recognize text on the piece of paper. The final aspect is to create a stand for the device, as mentioned under "Camera Image Testing". The camera module and Raspberry Pi will be located inside the device, and as shown in Figure 22 below, a hole is lasercut on the bottom acrylic surface. That hole is where the camera module will be situated in the device, and since the camera is located at that bottom location, the device itself must be located at a certain height above ground level for the image of any text to come out clear and of good quality. The team has experimented and found out that the optimal height for the device to be above ground level is between 5 to 6 inches above the ground. Thus, a stand must be created for the device.

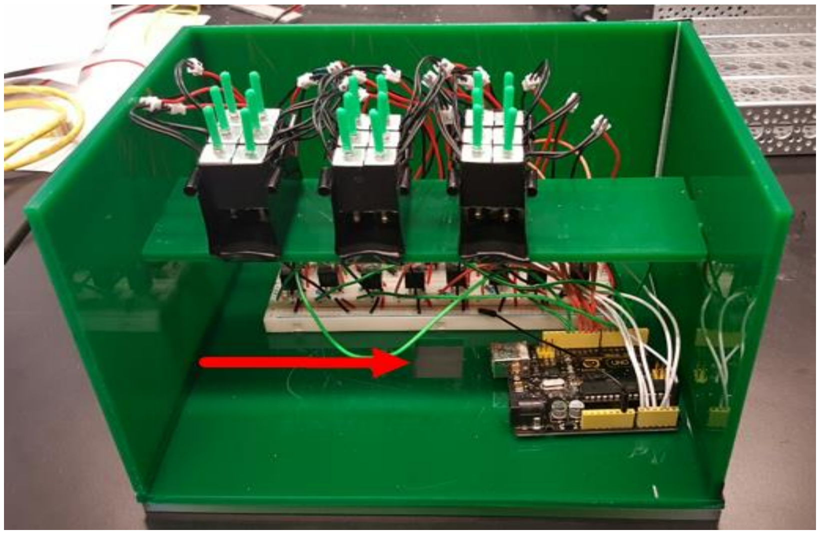


Figure 22. Inside of the Device with Arrow pointing to Hole on the Bottom

Acknowledgements

Valuable advice and recommendations were provided by Professor Robert Dell and Stan Wei. Instrumentation was provided by The Cooper Union and the professors, and special thanks to the Electrical Engineers at the Cooper Union who have helped with the coding and debugging process: Sebastien Charles, Susung Choi, and Darwin Huang.

References

1. R. Seth. “Reading Braille Aloud” Internet: http://www.yankodesign.com/2011/05/05/reading-braille-aloud/, May 5, 2011 [Sept. 26, 2016].

2. D. Rose. “Braille is spreading but who’s using it?” Internet: http://www.bbc.com/news/magazine-16984742, Feb. 13, 2012 [Sept. 30, 2016].

3. Associated Press. “Fewer blind Americans learning to use Braille” Internet: http://www.nbcnews.com/id/29882719/ns/us_news-life/t/fewer-blind-americans-learning-use-braille/, March 26, 2009 [Sept. 30, 2016]

4. C. Castellano. “Reading Issues: Getting to the Root of the Problem” Internet: https://nfb.org/images/nfb/publications/fr/fr31/4/fr310414.htm, [Sept. 30, 2016].

5. J. Weir, “What Problems Do People with Disabilities Have?” Internet: http://codi.tamucc.edu/archives/computing/.making/.part2.htm, [Sept. 29, 2016].

6. American Foundation for the Blind. “What Is Braille?” Internet: http://www.afb.org/info/living-with-vision-loss/braille/what-is-braille/123, [Feb. 9, 2017].

7. Ryans Tutorials. “Bash-Scripting-Tutorial” Internet: http://ryanstutorials.net/bash-scripting-tutorial/bash-script.php, [Mar. 14, 2017].

Appendices

Example Code for Text-to-Braille Conversion

```
//  main.cpp
//  C++ Practice
//
//  Created by Soyoung Moon on 9/17/16.
//  Copyright (c) 2016 Soyoung Moon. All rights reserved.
//

//https://www.reddit.com/r/dailyprogrammer/comments/1s061q/120313_challenge_143_easy_braille/

#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <stdio.h>
#include <unistd.h>
#include <fstream>
#include <cmath>

using namespace std;

map<char,int> braille_table =map<char,int> {

    {' ',0x000000}, {'a',0x100000}, {'b',0x101000}, {'c',0x110000},
    {'d',0x110100}, {'e',0x100100}, {'f',0x111000}, {'g',0x111100},
    {'h',0x101100}, {'i',0x011000}, {'j',0x011100}, {'k',0x100010},
    {'l',0x101010}, {'m',0x110010}, {'n',0x110110}, {'o',0x100110},
    {'p',0x111010}, {'q',0x111110}, {'r',0x101110}, {'s',0x011010},
    {'t',0x011110}, {'u',0x100011}, {'v',0x101011}, {'w',0x011101},
    {'x',0x110011}, {'y',0x110111}, {'z',0x100111}
};

vector<int> toBraille(string str){
    vector<int> braille;
    for(int i = 0; i < str.length(); i++){
        braille.push_back(braille_table[str[i]]);
    }
    return braille;
}

void printBraille(vector<int> braille){
    for(int i = 0; i < braille.size(); i++){
        printf("%06x\n", braille[i]);
    }
}

int getDigit(int input, int digit) {
            int individual = (int)(input/(pow(16,(digit-1))))%16;
            printf("%d\n", individual);
            return individual;
}

 int main(int argc, const char * argv[])
    {
       string line;
            std::ifstream myfile;
            myfile.open("result.txt");
            while (true)
                        {
                        //Get input
                        myfile >> line;

                        //Avoid repetition of last line:
                        if(myfile.eof() ) break;
                        }
        string s1 = line;
        vector<int> braille = toBraille(s1);

        int first = braille[0];
            int second = braille[1];
            int third = braille [2];
            int fourth = braille [3];
        int numberone = getDigit(first, 6);
        int numbertwo = getDigit(first, 5);

        printf("%06x\n",braille[0]);
```

```
            int numberthree = getDigit(first, 4);
            int numberfour = getDigit(first, 3);
            int numberfive = getDigit(first, 2);
            int numbersix = getDigit(first, 1);
            int numberseven = getDigit(second, 6);
            int numbereight = getDigit(second, 5);
            int numbernine = getDigit(second, 4);
            int numberten = getDigit(second, 3);
            int numbereleven = getDigit(second, 2);
            int numbertwelve = getDigit(second, 1);
            int numberthirteen = getDigit(third, 6);
            int numberfourteen = getDigit(third, 5);
            int numberfifteen = getDigit(third, 4);
            int numbersixteen = getDigit(third, 3);
            int numberseventeen = getDigit(third, 2);
            int numbereighteen = getDigit(third, 1);

        ofstream out("save.txt");

      out << numberone << numbertwo << numberthree << numberfour << numberfive <<
numbersix << numberseven << numbereight << numbernine << numberten << numbereleven <<
numbertwelve << numberthirteen << numberfourteen << numberfifteen << numbersixteen <<
numberseventeen << numbereighteen << endl;

   }
```

Example Arduino Code for Braille Pattern to Solenoid Movement Conversion

```
char data;
String datadata;

//Buffer Setup
const int pinAmt = 18;
int pin[pinAmt];
int dataBuffer[pinAmt];
int bufferIndex = 0;

void setup() {
  Serial.begin(9600);
  //Setup output pins
  for(int i = 2; i < 12; i++){
   pin[i] = i;     //pin numbers held in "pin" array
   pinMode(pin[i], OUTPUT);    //set pins 3, 4, 5, 6, 7, 8 to OUTPUT
  }
  pin[0] = 13;
  pin[1] = 12;
  pin[12] = A0;
  pin[13] = A1;
  pin[14] = A2;
  pin[15] = A3;
  pin[16] = A4;
  pin[17] = A5;
  pinMode(pin[0], OUTPUT);
   pinMode(pin[1], OUTPUT);
  pinMode(pin[12], OUTPUT);
  pinMode(pin[13], OUTPUT);
  pinMode(pin[14], OUTPUT);
  pinMode(pin[15], OUTPUT);
  pinMode(pin[16], OUTPUT);
  pinMode(pin[17], OUTPUT);
}

void loop() {
  recvInfo();
}

void recvInfo() {
  if (Serial.available() > 0)
  {
    data = Serial.read();                   //read one char representing 0 or 1
    dataBuffer[bufferIndex] = charToInt(data); //convert char to int, store in buffer
    bufferIndex++;

    if (data == '\n')                       //recieved delimiting character (end of character)
        {
            Serial.print("Arduino Received: ");
            for(int i = 0; i < pinAmt; i++){
              Serial.print(dataBuffer[i]);  //print each bit in buffer
            }
            Serial.print("First Digit: ");
            Serial.println(dataBuffer[0]);
            outputToLED();                  //update LED state to represent this character's 6 bits
            bufferIndex = 0;                //reset buffer index
        }
  }
}

int charToInt(char c){
  return (int)c - 48;
}

void outputToLED(){
  if(dataBuffer[0] == 1) {
    digitalWrite(13, HIGH);
  }
  else if(dataBuffer[0] == 0) {
    digitalWrite(13, LOW);
  }

  for(int i = 1; i < pinAmt; i++){

    if(dataBuffer[i] == 1){
      digitalWrite(pin[i], HIGH);
    }
    else if(dataBuffer[i] == 0){
      digitalWrite(pin[i], LOW);
    }
    else{
      Serial.print("ERROR: Invalid value read:");
      Serial.print(dataBuffer[i]);
    }
  }
}
```

Example Python Code for Raspberry Pi to Arduino Data Transfer

```
import serial
ser = serial.Serial('/dev/ttyACM0',9600)
```

```
import time
file = open('save.txt')
line = file.readline()
ser.write(line)
```

Bash Script for OCR and Text-to-Audio Conversion

```
#!/bin/bash
//raspistill -o test.tif
//tesseract test.tif result
flite -f result.txt
```

Bash Script for Text-to-Braille Conversion

```
#!/bin/bash
g++ main.cpp -std=c++11 -o main
./main
python ./toarduino.py
```