Reading Glove

Winston Sun, Junzhe Tang, Zoltan Williamson
winston.sun, junzhetang, zoltantakahiro @berkeley.edu

GitHub Repository: https://github.com/winston-sun/embedded_system_design

ABSTRACT

Currently, visually-impaired individuals rely on alternatives like Braille to read text. Thus, for any poster or sign, unless there is Braille infrastructure attached, visually-impaired individuals are generally unable to read what is on them. In this project, we alter an alternative to these individuals that is much more generalizable by designing and prototyping "reading" gloves – essentially gloves with a camera attached on the palm that can be used to read text from the camera input and output the text to speech.

The glove setup and image capture is implemented on an Arduino attached to the glove as well as via BLE, and the image processing and output is done via a laptop. Overall, the reading gloves are mostly able to decipher text from the photos it takes, but occasionally can face a few issues. Sometimes, images can be tricky to capture as any movement during capture can lead to noisy images. Additionally, image stitching is tricky if the captured images do not have necessary overlap.

1 Design

1.1 Embedded System Design

The reading glove design is shown in Figure 1, where an Arduino and camera module are attached to the palm of the glove. When starting the application, the Arduino will periodically capture images of different sections of the text while the user moves the glove slowly, process the image to the right dimension and format (thresholding, downsample, etc), and send out the data to the PC through the UART interface. The default data size per image is 220(height pixels) x 100(width pixels) x 1(bytes/pixel) = 22000 bytes, but the user can read and write to the application configuration through BLE to change the image settings depending on the specific use case and environment.

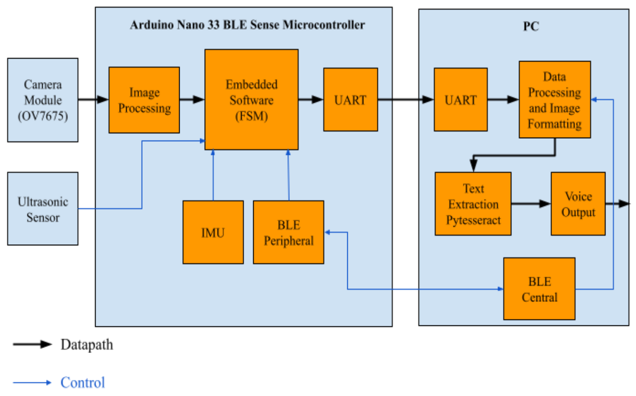


Figure 1: Block diagram of the reading glove application (datapath and control)

A finite state machine (Figure 2) is running on the Arduino to control the tasks that need to be performed. On startup, the Arduino enters the IDLE state and waits for the user to push the start button on the PCB. In the IDLE state, the user can also configure the image parameters through BLE to better fit the specific application and environment. It then enters the MEASURING state, where the Arduino reads all sensor data from the IMU and the ultrasonic sensor to decide if the conditions are satisfied to only take a clear and useful image (i.e. not taking blurred images caused by rapid motion). If it does not meet the conditions of image capturing, it will stay in the MEASURING state. The READING state is responsible for the image capture and goes to the SENDING state right after, where the image data will be sent through UART to the PC for further image processing to convert text to speech.

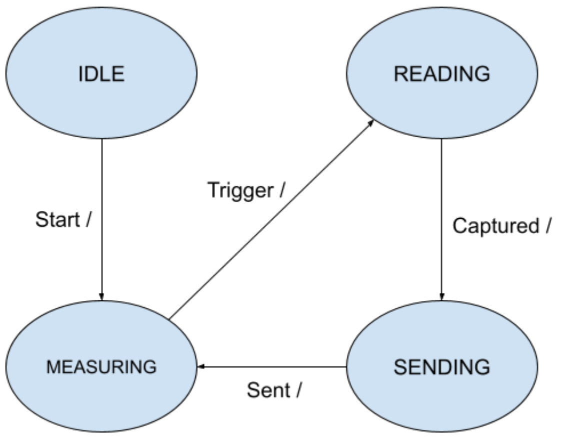


Figure 2: The finite state machine diagram

1.2 Software Design

On the laptop, a loop is set to run that will constantly read data from the UART. At any point when data is received, it is converted from its specified type to a numpy array for easier image processing. Then, it is kept in a list until a given set of images is completed. Once the image set is completed, these images are stitched together to form a single image with all the necessary text. Once stitched, this final large image including all text is converted to grayscale, gaussian blurred, and thresholded (for easier text extraction). Finally, we use the library pytesseract to get the text from the image, which is sent to the say function to be read aloud by the laptop.

2 Hardware and Software

The reading glove shown in Figure 3 has an Arduino Nano 33, a camera module, and an ultrasonic sensor attached to the palm side and the user can start the reading application when facing it downwards toward the text at a distance. A UART connection is used between the glove and the PC as both a power supply and data transfer mechanism.

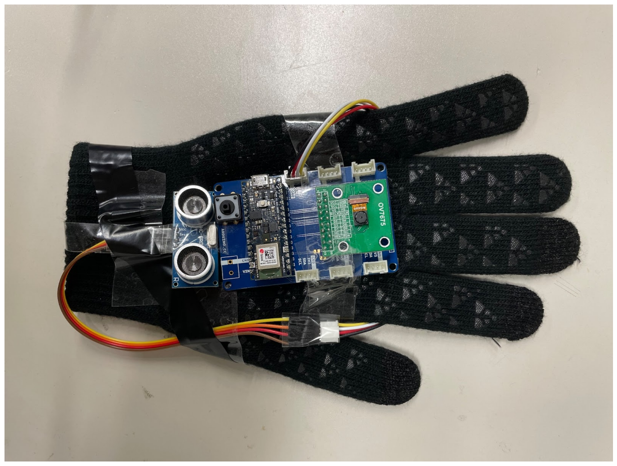


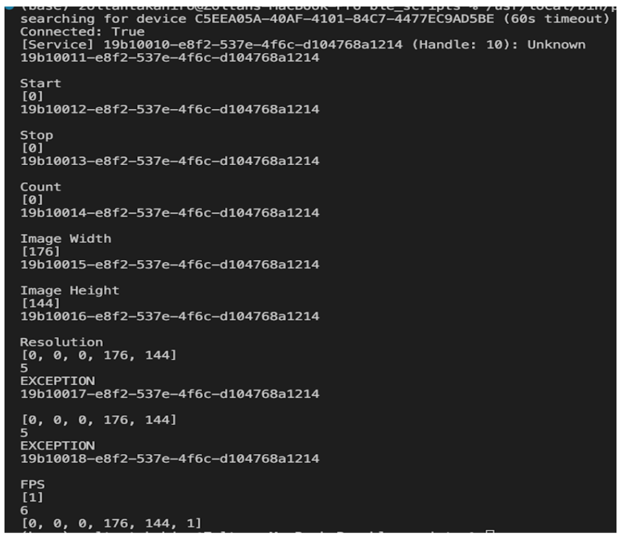
Figure 3: Reading glove design

2.1 Arduino Nano 33

Arduino Nano 33 is used to develop the embedded application. This microprocessor comes with a 9-axis inertial measurement unit (IMU), Bluetooth Low Energy (BLE) and universal asynchronous receiver-transmitter interface (UART). The microprocessor consumes ultra-low power and is suitable for designing embedded products [1].

2.1.1 IMU. The IMU is used to detect the movement and orientation of the glove. The embedded software uses the IMU data to determine if the glove is moving while capturing an image since motion will cause a blurred image capture. If fast acceleration (over a magnitude of 0.2 g in all axis) is detected, the software will halt the image capture process until the camera is stable. The IMU also detects the orientation of the glove and only captures images while the glove's camera is facing downward toward the text. The IMU decides when to capture images and it saves processing time and energy by not capturing and sending unusable images.

2.1.2 BLE. The BLE is used to transmit control and status data from the Arduino (peripheral device) to the PC (central device), including image capture controls (start, stop, and count), and image settings (width, height, resolution, format, and fps), as shown in Figure 4.. Depending on the settings and applications, the user could configure the image capture to ensure the optimal performance of the reading glove. For example, in a place with dim lighting, the user may select the image to be RGB565 to capture a clear image with higher resolution for more accurate image processing; whereas in a place with ambient lighting, a grayscale image can reduce the amount of data transfer, resulting in faster response time. See section 2.2 for more details on the camera configuration.



```
searching for device C5EEA05A-40AF-4101-84C7-4477EC9AD5BE (60s timeout)
Connected: True
[Service] 19b10010-e8f2-537e-4f6c-d104768a1214 (Handle: 10): Unknown
19b10011-e8f2-537e-4f6c-d104768a1214

Start
[0]
19b10012-e8f2-537e-4f6c-d104768a1214

Stop
[0]
19b10013-e8f2-537e-4f6c-d104768a1214

Count
[0]
19b10014-e8f2-537e-4f6c-d104768a1214

Image Width
[176]
19b10015-e8f2-537e-4f6c-d104768a1214

Image Height
[144]
19b10016-e8f2-537e-4f6c-d104768a1214

Resolution
[0, 0, 0, 176, 144]
5
EXCEPTION
19b10017-e8f2-537e-4f6c-d104768a1214

[0, 0, 0, 176, 144]
5
EXCEPTION
19b10018-e8f2-537e-4f6c-d104768a1214

FPS
[1]
6
[0, 0, 0, 176, 144, 1]
```

Figure 4: BLE readings for hardware configuration

2.1.3 UART. UART is used to transmit data from the Arduino to the PC for image processing. The baud rate of the transmission is set to a maximum of 115200 bps for the fastest data transfer speed. After each image capture, the UART will immediately transmit the data to the PC.

2.2 **Camera OV7675**

The OV7675 camera is a VGA sensor that connects to the Arduino using I2C [2]. It supports multiple modes that the users could select. The resolution of the images is VGA (640x480), CIF (352x240), QVGA (320x240), QCIF (176x144), and QQVGA (160x120). The formats are YUV422, RGB444, RGB565, and Grayscale. Note that RGB565 uses 2 bytes per pixel, whereas grayscale uses 1 byte per pixel.

2.3 **Ultrasonic Sensor**

The ultrasonic sensor is used to measure the distance that the reading glove has moved and decides the capture interval and stops capturing when the glove reaches the end of a page. The sensor passes the distance information to the FSM as input for logical decisions.

2.4 **Image processing**

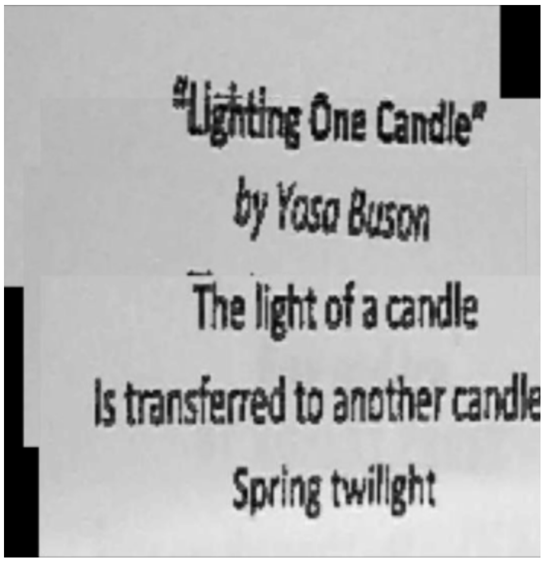
First the image must be reconstructed after it is received from the UART. Depending on the setup settings, the image can be in RGB or grayscale. Thus, the image reconstruction must be able to handle both cases. For grayscale images, this is pretty simple since the resolution is also known via setup, and becomes a simple resize. For RGB images, though, the hexadecimal bytes must be converted from RGB565 to RGB 24-bit (an easier format for future image processing). For most testing, grayscale was used as the additional information from RGB wasn't really necessary for text extraction, and just added unnecessary complexity.

After image reconstruction was completed, the reconstructed image was saved in a list of images for each image set. After the last image in a set was added, image stitching would commence.

2.5 **Image Stitching**

Initially, image stitching was attempted using OpenCV's stitcher class and stitch method. However, we quickly realized that OpenCV's stitch method was pretty reliant on finding a large number of keypoints that it could use for stitching, which made it great for something like panoramic images where there was a lot of overlap and many different possible keypoints, but pretty bad for images which only contained text (and therefore not enough keypoints). Thus, we pivoted to a different stitching method that we found online, and modified for our own purposes [3].

In essence, to circumvent the keypoint problem, images are stitched but with prior assumptions about how the images will be taken. By assuming that we will always take images by moving the camera down vertically along a wall of text, we can assume that the bottom of the first image will have some overlap with the second image. Thus, a set percentage of the bottom of the first image can be used to align with a set percentage of the top of the second image, and so on. Using this new algorithm, images can be pretty accurately stitched together, as long as images are taken according to the listed assumptions. Although this added restrictions to how we capture data, it also led to much more robust stitching. An example of four images stitched together can be seen in Figure 5.

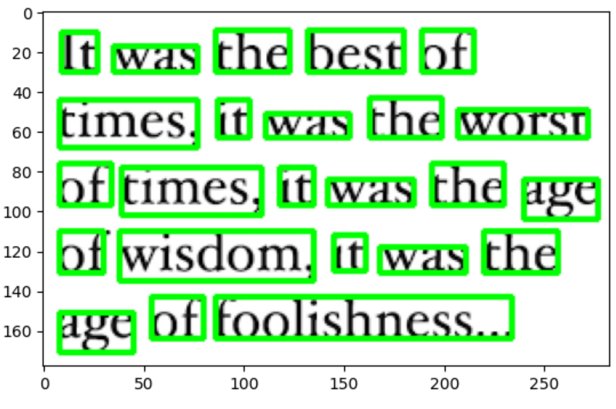


"Lighting One Candle"
by Yosa Buson
The light of a candle
Is transferred to another candle
Spring twilight

Figure 5: Example Image Stitch

2.6 Text Extraction and Output

After image processing and stitching, text extraction was performed (Figure 6). First, to make text extraction more accurate, the image is set to grayscale, gaussian blurred, and then thresholded. Then, pytesseract's image_to_string function was used, which essentially converts the text it can find within the image to a string. Finally, this string was parsed to ensure that it only contained alphanumeric characters, and was finally passed to the say function–which reads out the text to the laptop speakers (or headphones).



It was the best of times, it was the worst of times, it was the age of wisdom, it was the age of foolishness...

Figure 6: Text Extraction Bounding Boxes

3 Key Course Concepts

3.1 **Finite State Machine**

In this embedded system, we applied a finite state machine to control the reading glove. We have learned to code to perform practical tasks according to FSM diagrams.

3.2 **Sensors**

We used three kinds of sensors (i.e. IMU sensor, ultrasonic sensor, and Camera module). We have learned to read the documentation of the sensors to figure out their configuration and wire connection, and then acquire the outputs of the sensors.

3.3 **Wireless Network**

We used BLE to read the status and write the control of the Arduino hardware. The user could configure the camera settings for image processing through the BLE module.

4 Challenges

4.1 **BLE Data Overload**

Originally, we considered using BLE to transmit the image data; however, after some research on the BLE and hardware capability, it does not seem to be feasible. BLE has a low transmission speed and is not supposed to be used to transmit large data (one image is 22000 bytes) There are also packets lost during transmission. We instead use the serial link UART to transmit data, which is much faster and more reliable. We also considered using the WiFi transmission; however, the Arduino does not have a WiFi module embedded.

4.2 **Camera Resolution**

The camera captured blurred images (Figure 7) due to the reading glove's rapid movement, which is useless for image processing. The solution is to use the IMU data to decide when the glove is relatively stable and take the image.

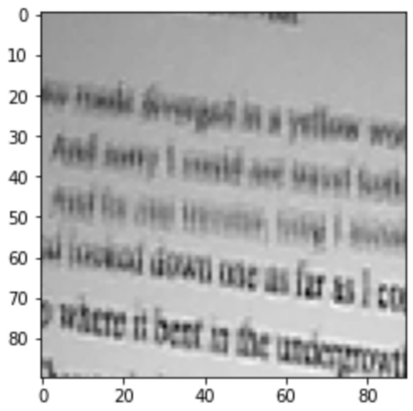

0
10
20
30
40
50
60
70
80
0
20
40
60
80
where it bent in the undergrowth
looked down one as far as I could

Figure 7: Blurred image captured due to fast glove movements

Also, the camera does not have a good resolution and is very sensitive to lighting (Figure 8). The workaround is to let the users have the flexibility to select different resolutions, sizes and formats of the image in different settings and environments.

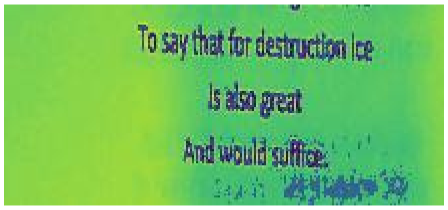


Figure 8: Odd discoloration (bottom right corner) due to errors when lighting changed

4.3 **Image Stitching**

The main issue we ran into with image stitching was a lack of keypoints in images containing predominantly text to use for OpenCV's stitcher class. Surprisingly, there are very few image stitching libraries besides OpenCV publicly available, which made solving this problem quite challenging. Additionally, stitching text images together seems to be a much more difficult problem that we initially anticipated. Eventually, we did find a solution [3] that we did have to slightly adapt to our own situation, but turned out to be quite well-suited to our own use-case.

5 Difficulties

The main difficulties caused by the strike is that we could not change hardware after realizing the limitations of what we had initially ordered. The Arduino Nano 33 BLE is not suitable for large data transmission since it does not have a WiFi module and limited memory. Also, the camera module does not have a good resolution and is very sensitive to lighting. Unfortunately, we couldn't order/change hardware after the strike. Additionally, it is possible that these issues we faced could have work-around solutions that might have been solved by TA support, or could have been foreseen earlier if we were able to consult with those familiar with the hardware.

6 Implication

The implications of our project for visually impaired individuals are quite exciting. As touched on prior, visually impaired individuals are quite reliant on additional infrastructure to be able to read any form of visual text around them. Oftentimes, Braille is only included on signs that require it–such as restroom signs, but are not included on posters or other forms of visual media. The introduction of something like the reading gloves offer an affordable and easy alternative, as it requires no additional infrastructure for others to adhere to. Although our implementation of the Reading Gloves is a pretty rudimentary prototype, it was still able to read poems printed out onto sheets of paper–and with some more sophisticated image processing and stitching (especially with Deep Neural Nets becoming more and more wide spread), the possibilities for reading gloves to be much more widely applicable are ever increasing.

Main areas of improvement moving forward would definitely be hardware that could take advantage of either WiFi or normal Bluetooth to send image data without having to be connected to a UART port. Additionally, better image processing such that image stitching is more reliable would also be helpful, as stitching together text images specifically proved to be a difficult problem that doesn't seem to have had much specialized work done to it as of yet.

ACKNOWLEDGMENTS

We used the example code of the BLE peripheral from Arduino [4] and the camera driver from TinyML [5]. Additionally, we adapted the solution for text image stitching from Stack Overflow [6].

REFERENCES

[1] "Arduino Nano 33 ble," Arduino Official Store. [Online]. Available: https://store.arduino.cc/products/arduino-nano-33-ble. [Accessed: 16-Dec-2022].

[2] "OV07675-A23A," OMNIVISION, 04-Jan-2022. [Online]. Available: https://www.ovt.com/products/ov07675-a23a/. [Accessed: 16-Dec-2022].

[3] Alaa M.Alaa M. 4, Elouarn LaineElouarn Laine 1, and Vikas GuptaVikas Gupta 56055 silver badges77 bronze badges, "Panorama stitching for text," Stack Overflow, 01-Sep-1964. [Online]. Available: https://stackoverflow.com/questions/45612933/panorama-stitching-for-text/45639406#45639406. [Accessed: 16-Dec-2022].

[4] K. Soderby, "Arduinoble/buttonled.ino at master · arduino-libraries/arduinoble," *GitHub*, 13-Nov-2018. [Online]. Available: https://github.com/arduino-libraries/ArduinoBLE/blob/master/examples/Peripheral/ButtonLED/ButtonLED.ino. [Accessed: 14-Dec-2022].

[5] S. Mistry, "Arduino_ov767x/ov767x.cpp at master · arduino-libraries/arduino_ov767x," *GitHub*, 12-Apr-2021. [Online]. Available: https://github.com/arduino-libraries/Arduino_OV767X/blob/master/src/OV767X.cpp. [Accessed: 14-Dec-2022].

[6] V. Alaa M.Alaa M. , Elouarn LaineElouarn Laine 1, and Vikas GuptaVikas Gupta 56055 silver badges77 bronze badges, "Panorama stitching for text," Stack Overflow, 01-Sep-1964. [Online]. Available: https://stackoverflow.com/questions/45612933/panorama-stitching-for-text/45639406#45639406. [Accessed: 16-Dec-2022].