Individual Progress Report

ECE 445, Senior Design

Pouya Akbarzadeh

April 8, 2021

1. Introduction

Our project's goal is to make electric longboard use safer by adding detection for wheel slip and correcting it. We are also trying to allow the user to know what is going on by giving user feedback through audio and visually as well.

I have been working on the User Interface, and Control subsystems for both the board and the remote. The Control Subsystem for the board and remote are in charge of sending and receiving data back and forth. The User Interface Subsystem is in charge of taking user input and displaying data for the user. The user uses the remote system to control the board system. The control is limited to the speed and direction of the board. Within the board system, the boards control subsystem is in charge of receiving data sent over from the remotes control subsystem and behave accordingly. Furthermore, I researched how we can build our own micro-controller and developed some modular testing for key components.

2. Design

2.1 Arduino/Micro-Controller Design

Since we had to make our own micro-controller, we decided to make one with the same architecture as the Arduino. This was due to the massive support of components, data sheets, forums, part availability, and cost. The first source of information for this was the Arduino website itself [1]. The first circuit made can be seen in figure 1, which was made by directly following instructions from the site. After some more research and testing, the circuit seen in figure 2 was built. This circuit was not only simpler, but was also specific to our microcontroller, ATMEGA328 [2]. However, the circuit for testing was even more reduced than the one found and seen in figure 3. Also, I should note that another team member created a programmer which is also a level shifter to allow us to program the micro controller at both 5V and 3.3V. That programmer can be seen in figure 12. The reason the circuit was simpler is the capacitor seen attached to the power source is there will be filtering from LDO. Furthermore, the resistor seen attached to pin 1 in figure 1 is implemented in the level shifter/programmer circuit.

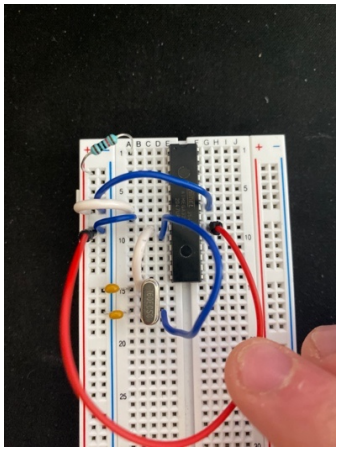

Figure 1

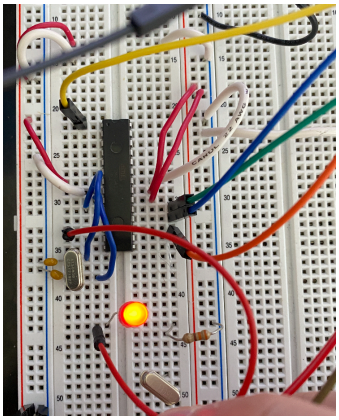

Figure 2

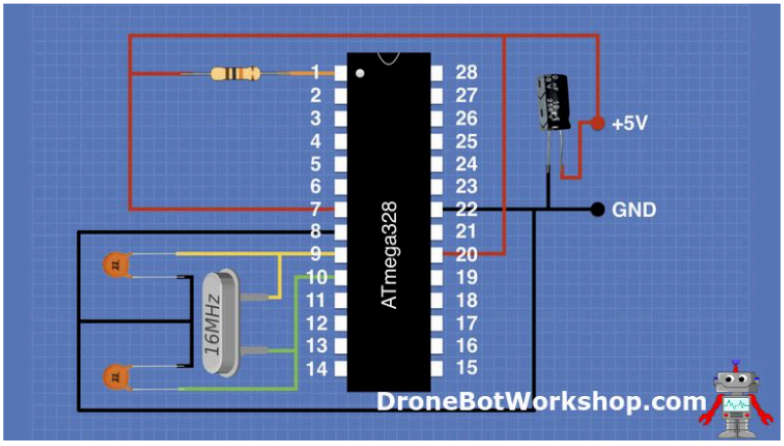

Figure 3

2.2 Wireless Communication

For wireless communications I decided to go with the NRF24L01 module which is a transceiver module allowing both systems to send and receive data. The very first circuit built was to ensure that we can have a stable 1-way communication [3]. The built circuit can be seen in figures 4 and 5. Here I built almost 2 identical circuits. What was different about them was the code that was uploaded to them. One of the circuits was always sending data and one was always receiving. This was simply a steppingstone toward building a 2-way communication for our project. To test 1-way communication working I decided to send a value based on the state of the button. If the button was pressed, the value sent would be 0.00, otherwise the value sent would be 1.00. The 1-way communication can be seen in figures 6 and 7.

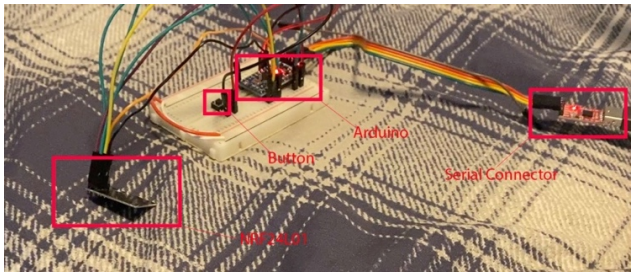

Figure 4

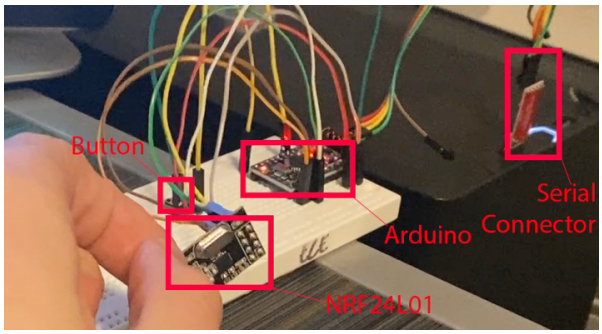

Figure 5

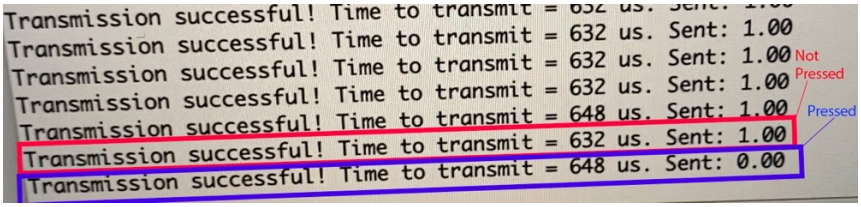

Figure 6

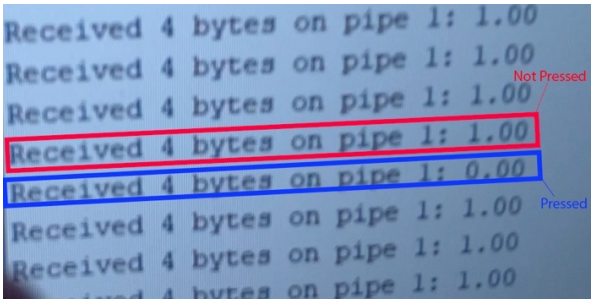

Figure 7

After getting 1 way communication working, I started with the first version of 2-way communication. The first approach was to constantly switch between sending and receiving modes. While this method worked, there was a slight amount of delay noticed during testing. Furthermore, it was an inefficient way. Thus, I switched so both systems are listening for packets. One board would send data every 50ms and the other would send every 55ms. Figure 8 would show a snippet of code that shows that. While this was working, and delay seemed gone, it was still not the best way.

```
54      if(timer % 50 == 0){
55        Serial.println("50 ms has passed");
```

Figure 8

Finally, the method I am using now is a bit more complex than the previous, however, it works much smoother and allows for a bit more versatility and functionality. Now, the remote sends data to the board, the board acknowledges that the packet was received and within that acknowledgment message sends some data back [4]. Not only this eliminates any issue by manually switching between sending and receiving data, but it also allows us to detect disconnection of either system. Furthermore, it simplifies the packet sent and packet received by decreasing the amount of data within. In figure 9 the code that is in charge of sending the acknowledgment packet can be seen. I still designed it so every 50ms a packet is sent, that means every second 20 packets are sent. That is more than enough, and communication being lost between them can be easily detected since after 5 fails in a row the speaker would play a high-pitched tone. This is explained more in the remote section of this report.

```
    myRadio.writeAckPayload(pipe, &data, sizeof(data));
    myRadio.read(&throttle, sizeof(throttle) );
```

Figure 9

Furthermore, to ensure the data sent is the exactly the data receive we use CRC, cyclic redundancy check. This method is used to check and detect accidental changes of data within communications channel [5]. This feature gets enabled after we enable acknowledgments [2]. Thus, not only are we ensuring data is sent and received, but we are also ensuring the integrity of the data as well.

2.3 Remote System

The remote system consists of power subsystem, user interface subsystem, and control subsystem. I worked on user interface and control subsystem.

Initially, I was planning on using the Wii Nunchuck as the remote for the user. The controller had a joystick and a few buttons which could be used for user input. Thus, the circuit seen in figure 10 was created on the bread board to read the inputs from the Nunchuck [7]. While this method was fully functional, as seen in Figure 11, a lot of the features such as the accelerometer, the second button, and the range of the joystick were not needed. Also, the amount of space within the controller would have made it difficult to fit in all the components. Also, taking into account that we would have been using another library to read the values of the Nunchuck we would have been adding a bit of complexity to our code and might have increased a bit of delay. Thus, another group member decided to find an alternative for the housing of the controller, while I worked on developing the communication for the remote.

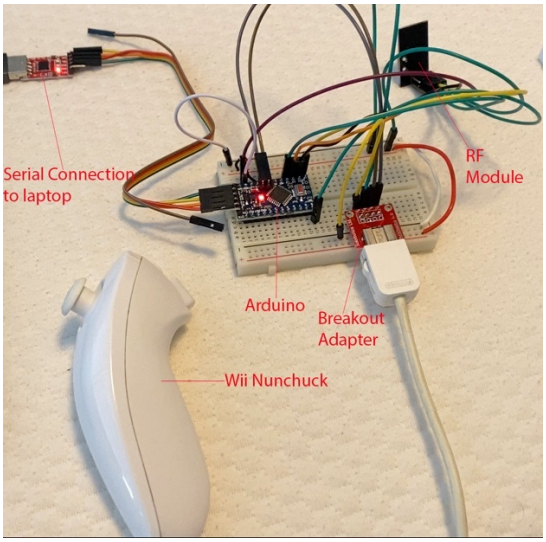

Figure 10

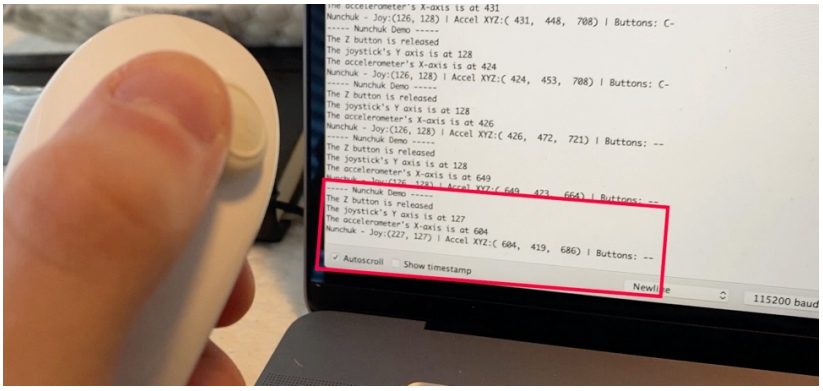

Figure 11

Therefore, the circuits showed in figures 12 and 13 were created. Circuit seen in figure 12 was for the remote and figure 13 circuit was the board. While the circuits look similar, the buttons, LEDs, and potentiometer served different purposes. The remote circuit is using the ATMEGA328 chip and basic circuit mentioned in section 2.1 and seen in figure 2. The potentiometer in this circuit is just used to mimic throttle for testing. It will be replaced with hall sensor, as seen later. The button is the dead-man switch. If dead-man switch is not pressed the board will receive the idle throttle value, while if pressed the board will read actual throttle value. The speaker is placed to place a sound if there is wheel slip detected on the board or connection is lost. During testing, the OLED screen displays the speed and time elapsed since board got power as seen in Figure 14. The LEDs are there as a visual to see level battery of the board. I will display the value on the screen during the final project, along side the battery value of the remote itself. In Figure 12 both user interface and control subsystem can be seen. The control subsystem includes the Arduino circuit and the NRF24L01 module while the rest, excluding the programmer, were part of the user interface subsystem.

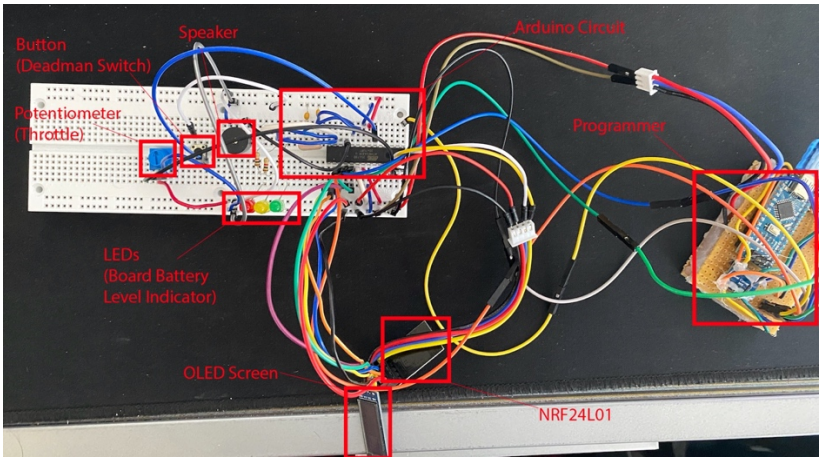

Figure 12

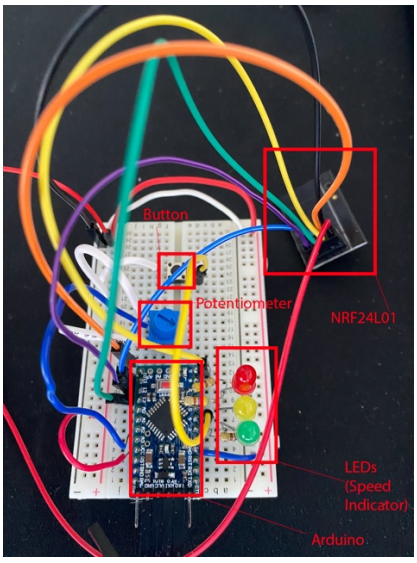

Figure 13

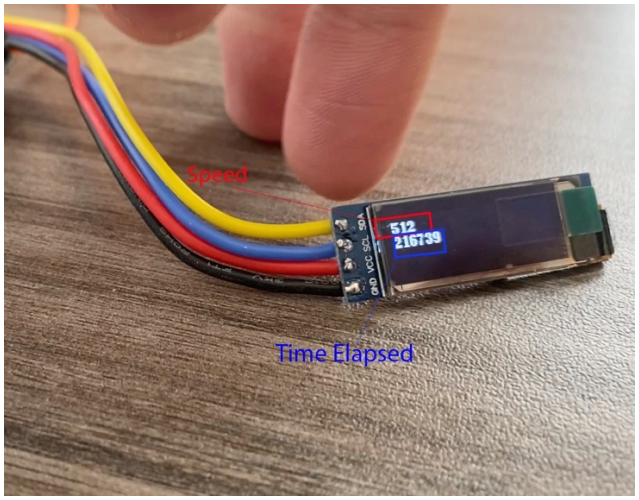

Figure 14

The use of hardware explained and seen above can be better explained using the following flow charts. Flow chart seen in figure 15 shows what the four basic things the code does.

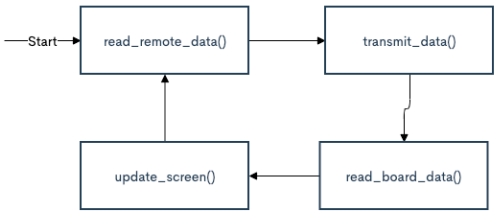

Figure 15

The four stages seen in figure 15 were all designed and tested separately. Designing each step as its own function allowed for modular design and easier ability to keep consistency. In figures 16, 17, and 18 the flow chart for reading remote data, transmitting data, and reading board data can be seen respectively.

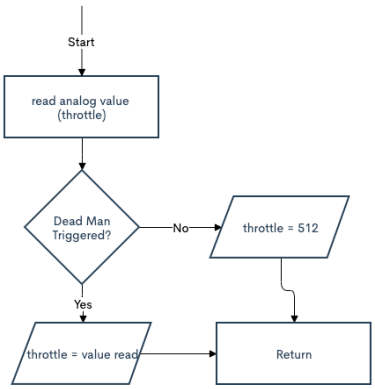

Figure 16

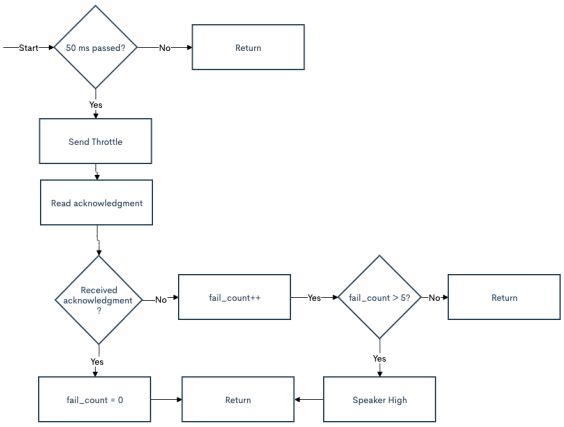

Figure 17

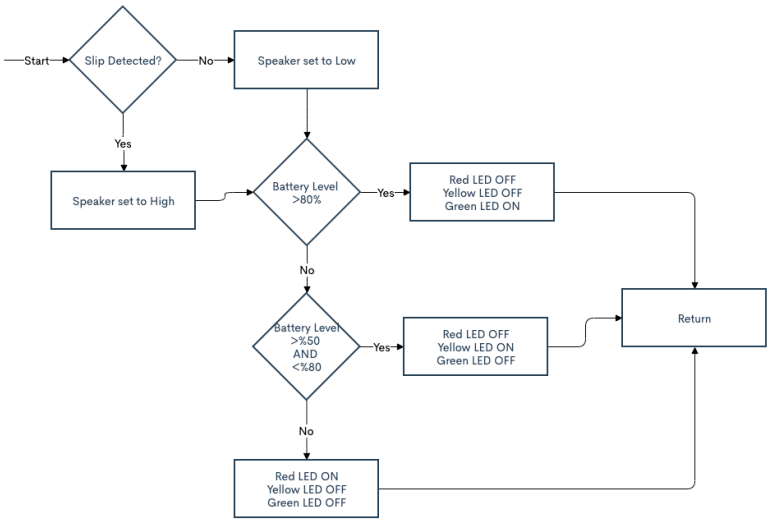

Figure 18

The test circuit is using a potentiometer due to the housing not being done. Thus, the remote has only been tested modularly. The remote can be seen in figure 24. The remote is using an analog input just like the mimic circuit in figure 12 was, thus it is not a huge change. The readings from the new throttle were tested and explained more in verification section. However, an issue was the scaling of the values read were not even. Meaning there were more data points from the 50%-100% throttle than there was from 0%-50%. After some research I found a function built into the Arduino library that would do the scaling [6].

Seen in figure 19 there are 2 boundaries, 230 and 250. If the value read is above 250, we are going forward and less than 230 is backward. Then using the map function, we set the range of values. 129-255 for upper range and 0-127 for lower range. The reason there is a 20-value gap between the two is due to the noise can be picked up by the hall sensor, thus this ensure we are reading the correct values.

```
if(throttle_read > 250){
  throttle  = map(throttle_read, 250, 1020, 129, 255);
}

else if (throttle_read < 230){
  throttle  = map(throttle_read, 30, 230, 0, 127);
}

else{ throttle = 128; }
```

Figure 19

2.4 Board System

The board subsystem design did not change as much, however, it's code did have significant changes. These changes were due to the remote code changing, and the way the remote and the board communicated. Just like the remote, the board had 3 versions. The first version received and sent data by switching back and forth between sending and receiving modes. The second version consisted of sending at intervals of 55ms, and listening the rest of the time. Finally, the final version, waits for a packet from the remote, and sends some data back along with the acknowledgement message. Figure 20 shows the final version, of the how the board receives data and sends data back. As seen in Figure 20 on line 59, we are sending "&data" with our acknowledgement. Our object "&data" consist of battery level and whether or not wheel slip was detected. This allows the correct data to be presented to the user through user interface subsystem. Furthermore, as another team member is working on wheel slip detection, I mimicked that being detected by a button as seen in figure 13. This allows consistency and allows me to see how responsive the system is. Furthermore, I added some LEDs to see if the correct speed was being detected at a first glance.

```
57  void get_data_remote(){
58      if (myRadio.available()){
59              myRadio.writeAckPayload(pipe, &data, sizeof(data));
60              myRadio.read(&throttle, sizeof(throttle) );
61              Serial.print("Speed: ");
62              Serial.println(throttle);
63      }
64  }
```

Figure 20

The flow of the board program can be better seen using the following flow charts. In figure 21 the overall flow of the program can be seen. While in figures 22 and 23 the specific functionality of the reading of the board data and updating LEDs can be seen. While reading remote data, the board data is sent as an acknowledgment as previously mentioned

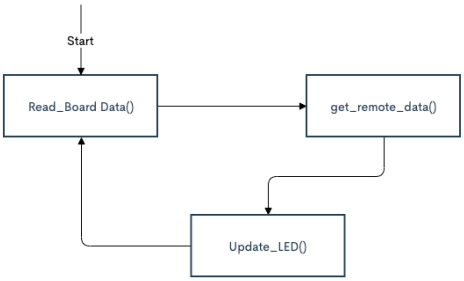

Figure 21

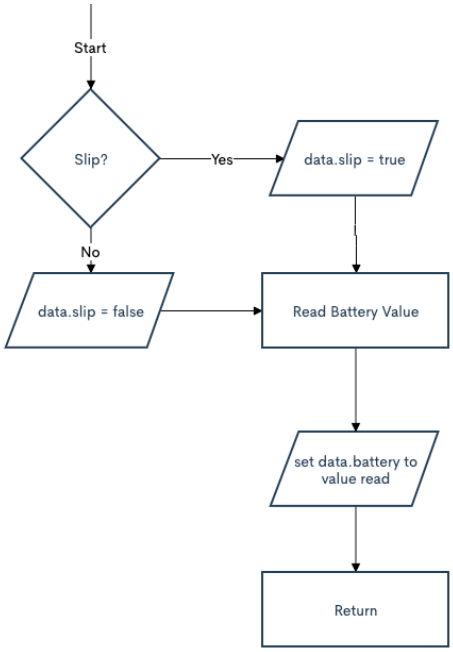

Figure 22

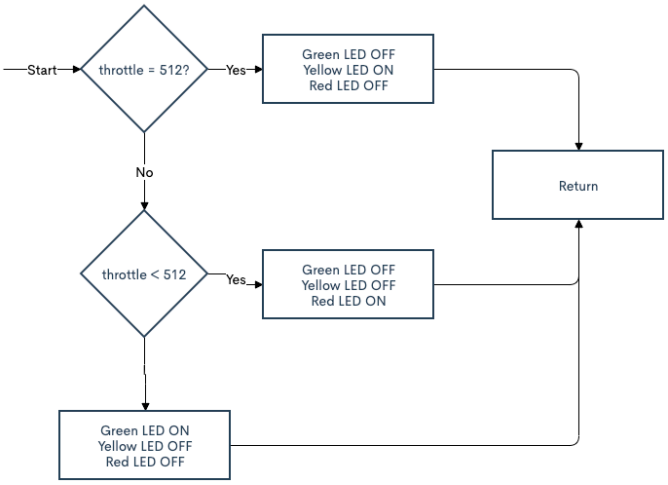

Figure 23

3. Verification

Verification has been very straight forward for the parts I have been designing. The first verification method I used is to see if the basic circuit worked. Each system was tested as it matured. The final system which I have explained below was tested from March 28th-April 3rd. However, each system was individually tested during each iteration.

3.1 Micro Controller

After building the micro controller circuit, I tested its basic functionality by running the blink code from the Basic Digital Examples of the Arduino program. By simply seeing the correct blinking I verified that the circuit was working. Furthermore, after building more complex circuits on top of the micro controller circuit, I did not face any issues.

3.2 Wireless Communication

I tested the wireless communication circuit by checking if the board can send an increasing float value and reading it. The wireless communication was later tested through testing other circuits that used its functionality. Another test written to test that NRF modules are functioning within the circuit is sending the elapsed time.

3.3 Remote System

The user interface subsystem was simply tested by reading the local circuit values and displaying it either through serial or on to the mini-OLED screen. The value of the potentiometer and button were tested by printing the value, while the LEDs were tested by reacting to those inputs. And mini-OLED was tested with printing known values on the circuit. The remote system also allowed testing for the wireless communication. When testing it with the board system.

The remote throttle was tested by reading its values and ensuring that the values read are scaled correctly. Figure 24 shows values of the hall sensor from the remote. The testing of the values and verifying their accuracy was simply done by continuously changing the throttle values and seeing the graph displayed. This test can be seen in the following video.

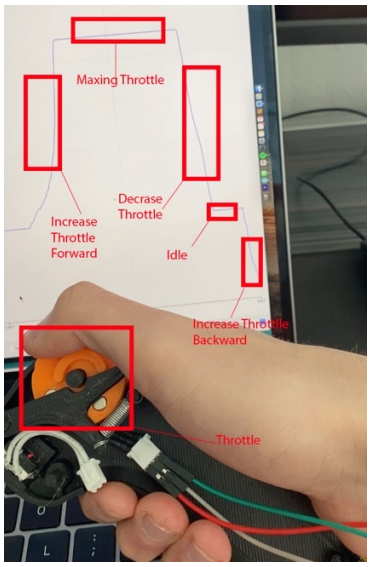

Figure 24

3.4 Board System

The board system was tested very similarly to the remote system. First it was tested to see local values were being read correctly by displaying its values through serial. Furthermore, I tested if they could communicate by sending their local info to one another.

3.5 Overall communication between board and system

The overall system was tested by mimicking what would be sent and received on a normal ride. A detailed video of me explaining the circuits again and showing functionality can be seen below.

4. Conclusion

In conclusion, I believe the workload has been equal for the most part. Not being able to meet as often or at all in person due to COVID has been making it harder to collaborate and help each other. However, we all have been dividing the work as much as we possibly can. I believe I have done my fair share, and whenever I was done or waiting for another component from another team member, I have reached out to help. I think we are pretty much on schedule, the PCB delay, did not help, however, now that they are delivered we are moving to put our parts together and start integrating and do some integration testing.

I believe by next week, April 12th, we should be moving on integrating and testing majority of the parts. And we should be set for mock demos the upcoming week after that. I believe that our development has been consistent through all members, thus making the integration part smoother and will make the date mentioned feasible.

Furthermore, all the code can be found on my GitHub and all the testing and mini demos can be found on my YouTube channel through the links below.

GitHub: https://github.com/OfficialPouya/senior_design

Demos on YouTube:

1. Basic NRF24L01 Comms: https://studio.youtube.com/video/J1pijmffAdQ/edit
2. Reading Nunchuck Values: https://studio.youtube.com/video/s_vrb_CYFLE/edit
3. Using Nunchuck as Remote: https://youtu.be/Pm6kk023FWY
4. 1st Version 2 Way Communication: https://youtu.be/bkiRsE8vne8
5. Delay Noticed in 1st Version of 2 Way: https://youtu.be/ULH5CsezqhU
6. 2nd Version of 2 way (Nunchuck): https://youtu.be/D7hF2wHwvaU
7. 3rd Version of 2 way: https://youtu.be/vbyY5nDkwJ0
8. Analog Reading Remote Using Hall Sensor: https://youtu.be/3Twb-AiJChY

5. Citations

[1] "Setting up an Arduino on a breadboard," Arduino. [Online]. Available: https://www.arduino.cc/en/main/standalone.

[2] DroneBot Workshop, "From Arduino to ATmega328," DroneBot Workshop, 24-Jun-2019. [Online]. Available: https://dronebotworkshop.com/arduino-uno-atmega328/.

[3] Dejan Nedelkovski February 20, "nRF24L01 - How It Works, Arduino Interface, Code, Schematic," HowToMechatronics, 05-Feb-2021. [Online]. Available: https://howtomechatronics.com/tutorials/arduino/arduino-wireless-communication-nrf24l01-tutorial/.

[4] Last Minute Engineers, "In-Depth: How nRF24L01 Wireless Module Works & Interface with Arduino," Last Minute Engineers, 18-Dec-2020. [Online]. Available: https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/.

[5] "Cyclic Redundancy Check and Modulo-2 Division," GeeksforGeeks, 22-Dec-2020. [Online]. Available: https://www.geeksforgeeks.org/modulo-2-binary-division/.

[6] "map()," map() - Arduino Reference. [Online]. Available: https://www.arduino.cc/reference/en/language/functions/math/map/.

[7] O. Liang, "Wii Nunchuck Arduino Tutorial," Oscar Liang, 18-Jun-2015. [Online]. Available: https://oscarliang.com/wii-nunchuck-arduino-tutorial/.