**Friday, February 23rd, 2023**

I brought in a battery and charger which supports the plug for the backup motor controller I discussed in the previous journal report. I will attempt to use this instead of the VESC, and I have previously interfaced this type of motor controller with a Raspberry Pi. However, the battery was discharged, so I started class by charging it.

I wrote my own serial protocol to transmit the data and successfully confirmed that I was getting *both* the steering and throttle values through. This link was very helpful: <https://forum.arduino.cc/t/sending-multiple-values-to-arduino-through-serial/40454/8>

I sent the values back over the same serial connection and verified they were correct. Thus, I eliminated the serial connection as a potential source of failure. The servo control worked perfectly fine, however, the throttle control was still not working.

I got throttle control to work, it turns out that the VESC doesn’t follow the convention of a 1000 microsecond pulse being full backwards, 1500 being stop, and 2000 being full forwards. That is probably why it crashed last week – I was giving it a value I thought was reasonably fast but it interpreted it as full throttle.

Here is a screenshot of my testing setup. On the left is the serial monitor where I manually enter commands, and the right is the VESC configuration tool where I check to see what the value the VESC think it’s getting is.

A screenshot of a computer

Description automatically generated

For example, here I gave it a pulse of 1220, or 1.220 microseconds, and it thinks it has a pulse of 1.412 microseconds (see the two horizontal blue bars).   
These are just notes from my testing:

1220 min speed to run forwards. ~1250 should be max speed

1400 min speed to run backwards. ~1720 should be max speed

So what I found was that instead of mapping the full throttle range across 1000-2000, there are very specific zones which the VESC seems to like. Also, the range is backwards. In between 1220 and 1400, the motors simply won’t move, and past the extreme values is way too fast to be practical for what I’m trying to do. It already badly crashed with an input of 1800 (and 2000 should be max throttle, theoretically. But I think the VESC is shifting values forward by around 0.2 microseconds, or 200).

**Monday, February 26th, 2024**

I wired up the non-sensored motor controller to the car. I verified the motor controller could be controlled through the Arduino by running a script I have which cycles through a set of throttle values to ensure the wheels are spinning in the right direction.

**Wednesday, February 28th, 2024**

I attempted to verify the serial communication between the Arduino and the Jetson. However, the serial communication was extremely unreliable. I would sometimes have to send a command twice in order to get the motor to spin up. Furthermore, the serial.write() function seems to be a blocking command – it took over 1 second, on average, to run in the main loop (over two minutes, the longest was actually 2.7 seconds), thus preventing the camera frames or anything else from updating. On the other hand, it is effectively instantaneous running on my laptop.

To fix this, I looked at the documentation for the serial library to determine why the serial communication is so inconsistent and slow on the Jetson. I first found a few online sources which suggested that Linux ARM platforms use a smaller serial buffer than Windows x86. Thus, the buffer could be getting filled, causing a delay in sending any new messages, since old messages in the buffer have to be cleared first. To combat this, I tried clearing the output buffer after sending each command. This did not make it any faster or increase the reliability of the serial link. I also tried the serial.flush() method, which clears both the input and output buffer, and reading anything off of the input buffer just in case that was somehow slowing down the serial connection. None of these approaches fixed the problem.

I also added a delay after instantiating the serial port as I found other people with similar code who noted that opening the serial port was not an instantaneous operation, and trying to access the serial port too soon could cause problems. However, sending an individual message still takes about 1 second, which is far too slow to be usable.