**Friday, March 8th, 2024**

I continued to work on the serial communication in order to control the motors, trying out the various fixes I found on Wednesday.

Flushing the buffer did not do anything. I printed out the number of bytes in the buffer and found that there were zero in both the read and write buffers. Thus, there was no real need to flush the buffer since there was nothing left in it. To further trace the issue, I timed the execution of each of 3 commands:

1. Reading the acknowledgement from the Arduino off of the read buffer
2. Writing the control command to the Arduino
3. Flushing the buffers

It turns out that 1) and 3) took a negligible amount of time, while the write command would take much longer. I created a separate script to test this and found that if one serial command was sent, it took a very short time (0.0001 s) to send, but after around 2-3 messages it would start taking over 5 seconds to execute the exact same write() command. This behaviour persisted even when I tried two different software-emulated serial port implementations (<https://pyserial.readthedocs.io/en/latest/url_handlers.html#alt>). I did some more testing and found that the time it took to write would go from 0.0001 s, to around 0.4-0.6 s, to over 5 s when I tried to send three consecutive messages. If I let the serial port “cooldown” for a few seconds then this would reset for another three messages. This is not ideal since I need to be able to send updates at a high rate, ideally at least 20 Hz, without interruptions or high latency.

8th Period

We attempted to fix the problem with the broken steering on the car. The car would need to be almost completely disassembled to get at the broken plastic piece which will take at least an hour or two of work. Our 8th period block is only 40 minutes so I decided to take one of my club’s functional cars and use that instead. This car had a Raspberry Pi connected to a PCA9685 I2C multiplexer, which directly controlled the servo motor and the motor controller. I mention this because this car worked perfectly without any communication issues between the PCA9685 and the Raspberry Pi.

**Monday, March 11th, 2024**

I think there is reason to believe that there would also be less communication issues between the Jetson and the PCA9685 compared to the issues that have been frustrating me with Jetson/Arduino communication through serial over USB. I think it would be useful to switch to the PCA9685 as the low-level hardware driver instead of the Arduino given that I know it’s worked in the past. I did some research today to determine if the Jetson plays nice with the PCA9685 board and if there is already publicly available, tested, and reliable code to do this. I found some tutorials and guides online which do explain how to wire up and use the PCA9685 board. I will try to use it on Wednesday, replacing the Arduino.

**Wednesday, March 11th, 2024**

I started to wire up the PCA9685 I2C multiplexer to the Nvidia board. First I went to the Robotics lab to get the required type of screwdriver and also to charge one of the discharged batteries I had (their charger is very nice and apparently faster than the one we have in the CS lab).

The wiring was quite simple although it did take some time as I wanted to verify the corrections were correct. I then tested out the control of the servo motor. It didn’t work, but only because a wire came loose. After that, it worked flawlessly. I then worked on getting the throttle motor working with an ESC that was conveniently already in the chassis I acquired. This did not work. I am not sure why. However, using the ESC from the previous iteration of the car made it work perfectly. The wiring is still very messy, but I will try to clean it up so that I can run a stand-alone, untethered (not running top of a box) test next class.