**Friday, February 9th, 2024**

Shreyan designed and printed a new camera mount which is adjustable. We will mount it on the car during 8th period.

A group of white plastic objects on a table

Description automatically generated

Note that the camera can now be rotated around since there’s a curved slot instead of just holes. This will let me find the best angle to capture the bird’s eye view of the track.

On Wednesday, I established motor control with my laptop, so today I focused on getting it working while wired up to the car. In Linux, all USB devices appear under the /dev/ folder as files, with names such as /dev/ttyUSB0 or /dev/ttyACM0. However, this depends on the order they were plugged in/initialized, and are thus not statically assigned to specific devices such as the motor controller or camera. Thus, my first step was to assign aliases to each so that on boot, they each get a correct, human-readable name.

Notes for my own use:

* /dev/ttyACM0 🡪 VESC motor controller 🡪 /dev/sensors/vesc
  + idVendor = 0483
  + idProduct = 5740
* /dev/ttyUSB0 🡪 RPLidar 🡪 /dev/sensors/rplidar
  + idVendor = 10c4
  + idProduct = ea60

I then tried to use the library’s code to control the motor controller. However, when it tried to initialize, it would hang forever and I had to forcefully exit the program. I found a simple example code using the same underlying library and installed it on my computer, and successfully spun the motors. Thus, the DK library’s wrapping of this code is faulty, so I debugged it.

I started by running the same basic script I ran on my laptop on the Jetson. It resulted in the same problem as I was having before. It’s not immediately clear to me why this is happening, so I filed an issue on GitHub to resolve it. Since I might be waiting for a while for a response, I started looking into alternative methods for interfacing with the motor controller. It implements a custom protocol which is quite complex, so I don’t think I will be able to implement it myself, correctly, in a reasonable amount of time. There is a ROS (Robot Operating System) which would allow me to ctrol the VESC motor controller through C++ instead of Python. I tried installing the ROS library, however, I ran into some issues with package version conflicts I wasn’t able to resolve.

**Monday, February 12th, 2024**

I got a response from the person who created the original Python VESC code. They said that on the Jetson, my user had to be added to the dialout group in order to have the proper permissions to access the USB port. I checked, and my user was already added to the dialout group. Past that, they had no further suggestions.

I moved on to controlling the VESC with an Arduino, and then controlling the Arduino through the Jetson. This is because the VESC natively has C bindings which I can run on the Arduino, and then run my own protocol through the Jetson so that I can control it through Python.

**Wednesday, February 14th, 2024**

I started working on the Arduino/Jetson communication code. I found the pySerialTransfer library (<https://github.com/PowerBroker2/SerialTransfer>) which creates an interface between a computer running Python and the Arduino. I first started with a basic mirror example that would transfer data to the Arduino, and then the Arduino would send that data back.

I then wrote a simple script to control the (old) VESC through the motor. It would ramp up to 50% throttle, and then back down to 0% throttle. This allowed the car to move untethered for this first time! I tested it outside in the commons where the drone group works.

I then bumped up the maximum throttle to 100% and tested it again. 100% throttle is way too fast: the car crashed into the entrance of the Robotics room and this caused part of the steering linkage to break on the car. We will fix it on Friday during 8th period. I limited the code to 80% throttle in the meantime.

I then tried to control the servo motor directly through the Arduino.