**Friday, March 15th, 2024**

I secured the wiring on the car using zip ties so that when the car moves, everything wouldn’t come apart and stop working. To do this, I tied down the I2C multiplexer board to the wooden mounting plate and also tied down some extra long wires which were getting tangled up in the wheels. Then, I turned on the car while it was on the floor. The Nvidia Jetson board automatically connected to the router that I set up and I was able to access it by connecting my laptop to the router as well. Once I did that, I could remote-control the car through the web interface hosted on the Jetson.

I quickly found that the steering and throttle limits I had put in while the car was still on the stand were way off. I had to run at full throttle in order to get the car to crawl forward. While this is a safe behavior, it’s not good for RL where we want a smooth response for throttle 🡪 velocity. Thus, I bumped up the maximum forward and reverse throttle so that the car would start moving (very) slowly when a throttle of +-0.2 was applied. Next, I tuned the steering. The left steering limit and right steering limit I had put in resulted in the car turning slightly to the right when the steering value was 0. Through some trial and error, I found a left/right limit which let the car turn symmetrically both left and right and allowed the car to go in a straight line when steering = 0.

Once I had got the basic setup, I re-connected the other components which I had removed when I had disassembled the car. First, I re-connected the camera, and verified the camera livestream appeared in the web interface. Next, I re-connected the lidar, and ensured that the data was still streaming.

**Monday, March 18th, 2024**

I worked on setting up the reinforcement learning code to test. I first want to test the reset() method, which will drive the car back to a set, known position. My first attempt at doing this will simply replay back the last N steering/throttle commands, where N is the number of timesteps since the last reset. To do this in python, I created a deque which stores the commands, and then popped them off from the right, inverting the throttle value so that it will go backwards.

**Wednesday, March 20th, 2024**

All seniors were on a field trip in DC.

**Friday, March 22nd, 2024**

I continued working on the reinforcement learning code. I realized that I could potentially use lidar data as an input: this would eliminate the need to use an autoencoder to reduce the dimensionality of the input. Thus, I worked on creating an interface to the lidar in a way that would provide useful data to a reinforcement learning algorithm. The lidar provides data in individual time-stamped packets, each with an angle and distance measurement. I wrote code to compile this data into an array of values, where each index from 0-360 represents an angle. As the lidar spins, the array is filled with data. I tested this and it works well.

**Spring Break**

I tested the reset() method for the reinforcement learning which “plays back” the past steering and throttle inputs to return the car to a previously known state. I found this to be extremely inconsistent. When the replay buffer became longer, the car would attempt to re-trace a very long series of steps, introducing a compounding error both with steering and throttle. The end result was quite a few bad crashes. Thankfully, nothing broke, especially because I tied everything down with new zip ties and fixed the messy wiring.

Given the time-intensive nature of reinforcement learning, I realized that I wouldn’t be able to fully train the car during spring break in my house (and it would take much more time to train it again at school, especially since I would need to stop it so that other people could use the commons when class was not in session). Furthermore, the car’s secondary battery used to power all the electronics lasts about 2-3 hours, which means I would need to periodically recharge it.

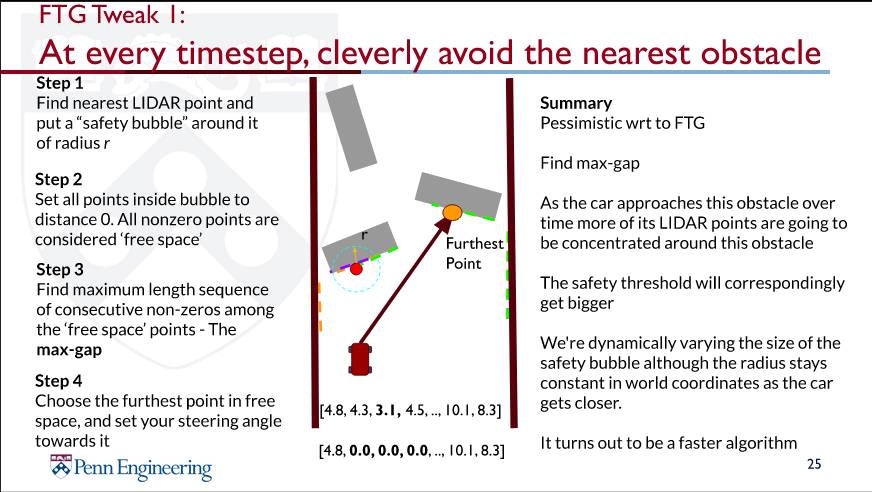
Thus, I pivoted to working on some possibly interesting demonstrations I could do with the car. My intention is to do a live demo during tjSTAR, so I worked on creating a reliable demo which I could repeat at school. My first idea was person following. I have discussed this in a previous journal report, but I experimented with detecting AprilTags, which are fiducial markers. With a calibrated camera (known focal length and distortion coefficients), the exact position, in meters, of the tag relative to the camera can be computed. My goal with the person following was to stick a tag on a person’s back, and have the car follow the tag’s position. This could be potentially be extended to a YOLO model coupled with stereo depth to eliminate the need for any tags.

I implemented this through two PID (proportional, integral, derivative) controllers, one controlling steering, and one controlling throttle. The steering PID aims to drive the X “error”, or camera’s offset from the tag, to zero, ensuring that the car is directly behind the tag. The throttle PID aims to drive the Z (depth) distance to a fixed number, which I set to 1 meter. Thus, the throttle would be modulated to keep the car 1 meter behind the person. I tested this out, and did some tuning to determine the optimal PID gains. I found that driving on a smooth surface such as wood is significantly different than a rough surface like carpet and requires an entirely new set of coefficients. Regardless, I was able to successfully make the car hold a constant distance from a person walking straight, even if the person stopped or walked backwards. However, the steering was more difficult. When the person with the tag turned, the camera would sometimes lose track of the tag momentarily, resetting the PID controller. (It would be a bad idea to let the integral term continuously accumulate forever, this is known as “integral windup” in PID control and it is generally thought to not be helpful). Furthermore, if the person moved sideways quickly, the tag would exit the camera’s field of view entirely. This is not good, especially for a demonstration where I would ideally let someone from the audience be followed and expect it to work reliably.

My second idea was to implement control strategies that used the Lidar data, instead. I remembered one instance in class when the car nearly crashed, and you asked me, “Shouldn’t it avoid crashes?”. Lidar data is definitely much more useful for collision detection and avoiding obstacles than the camera.

To start with, I tried to implement the strategy of “drive towards the farthest point you see”. To do this, I used the lidar to populate the angle 🡪 distance array. I then found the index corresponding to the largest value of the array, which was the angle at which that distance was measured. Finally, I linearly mapped the angle to a steering value from -1 to 1 and controlled the steering servo motor. This did not work well, particularly because there would sometimes be occasional outlier values even though the surrounding points were much closer. To fix this, I tried creating a quantized array with 30 elements, each representing the distance of a 12-degree slice. This was slightly better but it was still not able to complete a small circuit in my basement without crashing.

I decided to incorporate some of the more advanced lidar-based control algorithms which I had previous experience with using in a simulation. One of these algorithms is known as “Gap Follower”, which is an intelligent version of the “drive to the farthest point” algorithm I implemented earlier. Here is a lecture slide about the algorithm that I found useful:



I tried to directly import my previous Gap Follower code to work with the real car. I had to do some tweaking, as the coordinate systems with the simulated car and real car are different, and the distance measurements in the array are counted clockwise instead of counterclockwise. The bigger problem I ran into was the actual output of the algorithm. The algorithm output angle values as its output for the car to steer. I was linearly mapping these angles to steering values, but was completely guessing what the scaling factor should be. -1 could correspond to 45 degrees, or 60, or 30, and I had no clue which one was right. Thus, I had to do some calibration to determine the real turning angle of the car. I also suspected that the car’s turning was nonlinear.

A small vehicle with wheels and wires

Description automatically generatedTo measure this, I attached a laser range-finder to the car’s wheel and pointed the car to be facing a flat wall. Pictures of my testing setup are A small vehicle with wires and a stick

Description automatically generated with medium confidenceabove. The piece of wood keeps the wheel from turning while the rangefinder is tied to it, so that the rangefinder is always parallel to the ground. I planned to use trigonometry with the distance measurements from the rangefinder attached to the wheel to determine the true angle the wheel was turned. Here are my results:

Note the quadratic line of best fit. Also, positive steering inputs result in a negative angle because the coordinate system increments angles positively going counterclockwise. This best-fit line takes steering value as an input and returns the true angle as an output. However, I need the inverse of that, which looks like this:

A screen shot of a graph

Description automatically generated

The inverse function essentially told me that the maximum angle the car could turn was 30 degrees. However, the Gap Follower algorithm would frequently command the car to turn more than 40 degrees even though this was not physically possible. I knew the algorithm itself was correct, since it works amazingly well in simulation, but I needed a method which would respect the physical limitations of the car.

Since I had just tried out the Gap Follower algorithm, I remembered another technique also explained in the same UPenn course – wall following. The idea behind wall following is to maintain a constant distance from a wall, or maintain an equal distance between the left and right walls. I implemented this through a PID controller (again). The setpoint was 0, representing the delta between the left wall and right wall. These distances were calculated as the average distance of a 90 degree cone to the left and right of the car. After a small amount of tuning (much less than I did with the AprilTag follower), the car successfully navigated around a small track in my basement.

[](https://www.youtube.com/embed/phSQ4hcjYlg?feature=oembed)I added an emergency stop feature that would back up the car if the front distance became too small.

I think this is a good demonstration for tjSTAR since it is transferable to any environment provided there are solid, opaque obstacles that serve as walls.