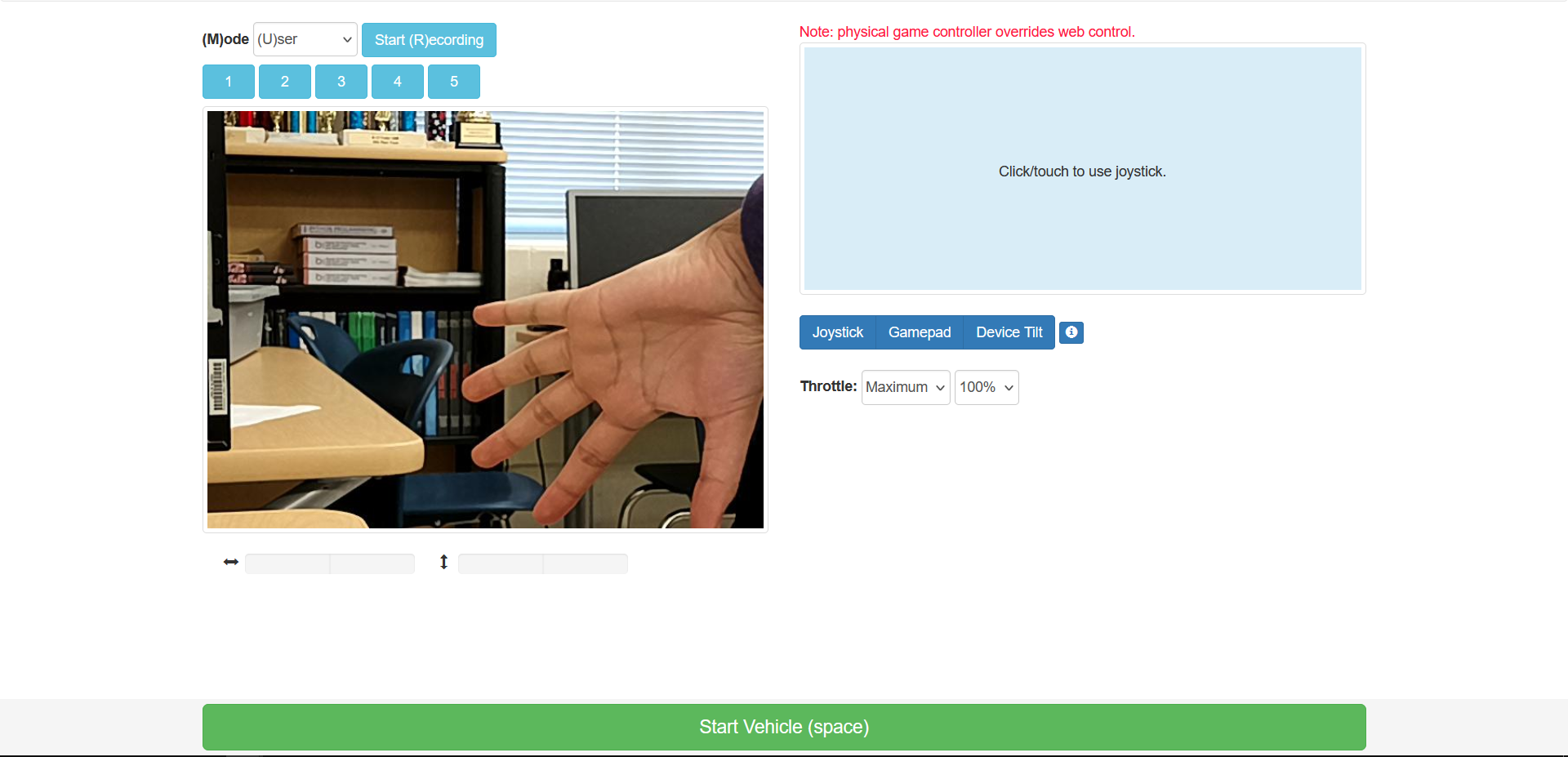
**Friday, February 2nd, 2024**

I started by testing the code I wrote to interface with the camera and the DK library. It worked, however, the RGB channels were flipped, which was easily fixed using OpenCV. Here is what the livestream looks like on the web interface:



Dr. Gabor and I talked about the plan for the project and how I plan to finish it. He identified two key considerations for implementing the reinforcement learning process, one of which I had previously not given much thought to:

1. Determining when an episode of training is over
   1. Both Dr. Gabor and I agreed we would have a track with two sets of parallel lines, most likely colored duct tape, on the ground. The goal of the reinforcement learning is to stay in between the lines. Thus, it’s important to know when the car has gone off the track since it’s no longer collecting useful information. The solution I will implement is to determine whether the boundary lines are visible on the ground. If they aren’t, we assume we have exited the track, and thus need to reset.
2. Resetting the car back to a good position so that training can re-start
   1. The solution which I already thought of was to maintain a “replay buffer” of previous throttle/actuation signals and then play them backwards so that the car would essentially reverse into a position it was previously at. I hope to use the motor controller’s RPM values in order to mitigate the problem of accumulated error. However, Dr. Gabor wanted a more robust solution. I have been doing some tests with the AprilTag, which is a fiducial marker. Given a calibrated camera (e.g. known focal length, image center, distortion coefficients), which I do have, there is readily available code to calculate the pose of an AprilTag relative to the camera. Thus, if I establish a set position for the AprilTag, I will know the pose of the camera. I can use an AprilTag, or multiple AprilTags, to re-localize in this way. By finding out the true position of the camera relative to the AprilTag, I can navigate back to a known position within the track boundaries.

Since I am waiting on the motor controller hardware, Dr. Gabor set a goal for me to get the RL working on the car in the next 3 weeks, assuming it takes 1 week for the parts to arrive, another week to verify control of the motors, and a third week to write the actual reinforcement learning.

During 8th period, we shifted back the camera mount because the cables coming from the camera were interfering with the spinning of the LiDAR sensor.

**Monday, February 5th, 2024**

I continued to consider the two problems above and how they will be solved. In particular, I was looking into how I could use AprilTags to give the car an estimate of its position.

The AprilTag library (<https://april.eecs.umich.edu/software/apriltag>) provides a camera to tag transformation matrix when it detects a tag. However, my goal is to know where the car is absolutely, instead of relative to a tag, since there would be multiple tags in strategic locations around the track.

To find the position of the car absolutely, I can construct a tag-to-world transformation, and then multiply the two transformations to get the camera-to-world transformation. I tried a basic example of this on paper and the math does check out.

I then started constructing the OpenAI gym environment for the real car, defining the action space and instantiating the parts I would need (motor controller, lidar, camera, web controller).

**Wednesday, February 7th, 2024**

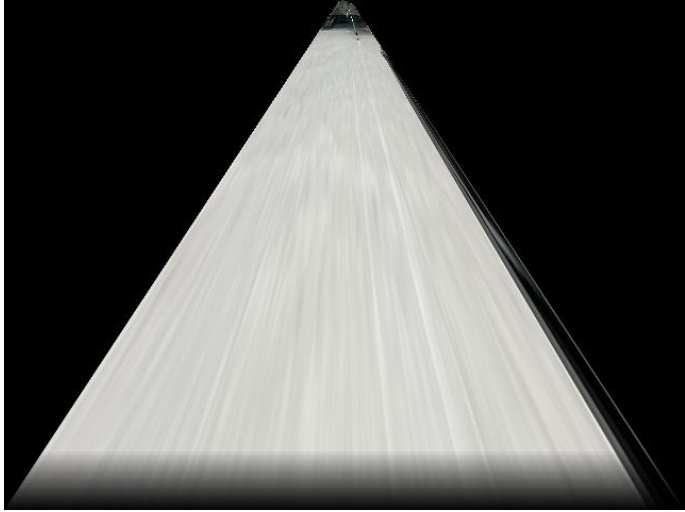
I finished a “first draft” of the Gym environment definition for the real car, with most of the methods un-implemented, to see if it would run without errors. After commenting out the motor control code, the code ran. Of course, it didn’t do much of anything since I haven’t defined the termination condition or reset functionality yet.

To implement the termination condition, I need to know if the car sees a line on the ground. To do this, I will first warp the image from the camera to a birds-eye view so that it’s easier to visualize the lines. The idea is essentially this:

A road with trees in the background

Description automatically generated

I tried implementing this on the real car using the OpenCV functions getPerspectiveTransform() and warpPerspective(). However, the problem is the camera is not actually seeing much of the ground in the first place. Thus, the transform look something like this:



Essentially, most of the pixels in the image are not pixels of the ground, and are thus useless for detecting lines on the ground. I will get a camera mount re-printed which angles the camera downwards to alleviate this issue.

That being said, I did find another approach which I think will be more robust than the camera. These cars are generally used with the Lidar sensor more than the camera. I found some work from UPenn researchers (<https://github.com/f1tenth/f1tenth_gym_onboard>) in which they use the lidar to determine if the episode is over (e.g. if something is too close). To define the track boundaries, they use hollow tubing with a height greater than that of the Lidar:

A person standing outside of a building

Description automatically generated (This is a picture from when I visited UPenn. A Ph.D student there gave me a small tour of their facilities including this test track).

As I was working on this during 8th period Dr. Torbert gave me the two adapters needed for the motor controller. I wired them up and successfully calibrated the new motor controller. On Friday I will test out Python-based control of the throttle and steering motors. Also, a new camera mount with the camera tilted downwards should hopefully be ready by then.