**Monday, January 22nd, 2024**

I watched and wrote feedback for the following people’s presentations:

* Ben
* Tristan
* The pendulum group (Michael, Eric, Johnny)
* Lucas

**Wednesday, January 24th, 2024**

I watched and wrote feedback for everyone else’s presentations except Akshith.

Also, Krishnan has set up the router to work with the TJ network. He sent me the IP address, uplink gateway, and DNS servers I needed to input into the router’s configuration dashboard. I followed his instructions, but it is still not working. I will follow up with him next week.

**Monday, January 29th, 2024**

Krishnan came to help me figure out the router setup. Currently, the error the router outputs is that the uplink gateway is unreachable. The uplink gateway is the DHCP server which assigns IP addresses to computers on the network. However, Krishnan logged into the DHCP server and looked at the log files to see if there were any connection errors, however, there was nothing in the log files regarding my router specifically. Setting the parameters through static IP instead of DHCP did not fix this issue. Krishnan believes that the router is simply trying to reach the wrong DHCP server and thus failing, but we aren’t sure how to proceed from here. It may be necessary to try an alternative solution such as using 2 Wi-Fi adapters to connect to a hotspot network and the FCPS network simultaneously in order to maintain an Internet connection. We also discussed using access point or internet repeater modes on the router, but determined that those won’t let you give devices on the network static IP addresses, which is important since we don’t have mouse/keyboard access to the Nvidia Jetson board.

**Tuesday, January 30th, 2024**

Last Friday during 8th period, we tried using the new motor controller. It has a larger power plug which is rated for a higher current. It is incompatible with the adapter previously being used in the car. I found the adapter used by the universities who originally created this project, so I am confident there will be no wiring issues.

**Wednesday, January 31st, 2024**

Although the Wi-Fi situation is still unresolved, I started working on combining the different hardware elements (camera, Nvidia Jetson board, lidar) in software, leaving the motor control functionality as a “Java interface” for now.

The car’s battery was discharged, so I plugged it into the charger. I then verified the LiDAR sensor was detectable through software on the Jetson. I used the Adafruit CircuitPython library for the lidar (<https://github.com/adafruit/Adafruit_CircuitPython_RPLIDAR>) and printed out the distance readings using a provided example script:

A computer monitor with a robot on it

Description automatically generated

Note that in the picture the sensor is activated and spinning.

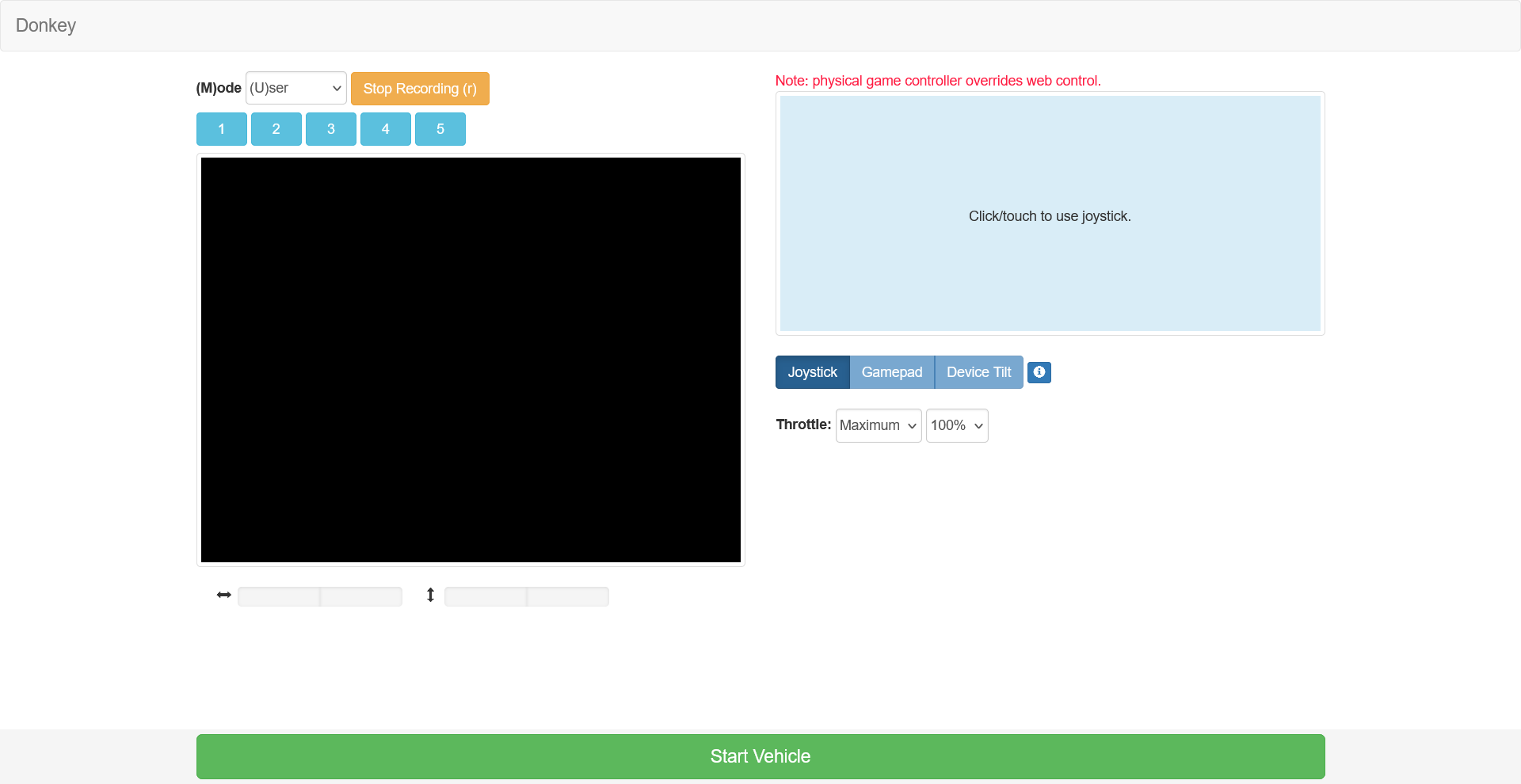
Next, I moved onto the overall vehicle pipeline. I have already verified that the camera works. Thus, the next step is to integrate the camera into the overall vehicle loop: observe, plan, act. More granularly, this is what the car will be executing in a tight loop:

1. **Observe**: Gather sensor readings (camera, lidar, IMU, etc.)
2. **Plan**: Use reinforcement learning or any other algorithm to process the data and generate motor commands
3. **Act**: Send commands to the motors, and ensure they are doing what we expect them to do.

I plan to do this through the DK framework (<https://github.com/autorope/donkeycar>) which is a “is a minimalist and modular self driving library for Python. It is developed for hobbyists and students with a focus on allowing fast experimentation and easy community contributions.” The library contains utilities to work with the common sensors/actuators and pull them together into the overall execution loop, which will save me a lot of time.

Installing DK went smoothly, however, one of its dependencies is TensorFlow. Naturally, I want the version of TensorFlow with GPU acceleration to be installed rather than the CPU-only version which will be much slower and less power efficient. Oddly, the Jetson wasn’t able to connect to the NVIDIA servers to download the wheel needed for installation when I connected it to the FCPSGuest network. Thus, I had to download the wheels on my computer and transfer them using the *scp* command-line utility to the Jetson. After doing this, I verified that TensorFlow was able to use the dynamically linked TensorRT libraries in order to use the CUDA cores for accelerating matrix operations.

Here is the web interface DK provides for controlling the car. It contains useful elements such as a joystick, throttle limiter, emergency stop, and the ability to start/stop recording data.



The black box is where the camera live feed would normally go. Although the DK library provides interfaces for common camera types such as the Raspberry Pi Camera and USB webcams, the OAK camera has its own Python library, *depthai*. Thus, I will need to write some code to provide a standard interface to the camera. I’ve already accessed the camera through Python, I will just need to determine the correct format so that it is interoperable with DK’s conventions and standards for passing data around.

I watched Akshith’s presentation and wrote feedback for him at the end of class.

During 8th period, I worked on writing the interface code for the camera. A key function of the DK library is that it maintains a global state and links the inputs and outputs of various parts, so that information can be re-used across them. To better illustrate what I mean, here is a snippet of my code:

elif cfg.CAMERA\_TYPE == "OAKD":

        from donkeycar.parts.oak\_d import OakD

        cam = OakD(

            enable\_rgb=cfg.OAKD\_RGB,

            enable\_depth=cfg.OAKD\_DEPTH,

            device\_id=cfg.OAKD\_ID)

        V.add(cam, inputs=[],

              outputs=['cam/image\_array', 'cam/depth\_array'],

              threaded=True)

In this code, V is a “Vehicle” object which maintains all information about every sensor, actuator, and piece of logic driving the car. The camera part does not need any further inputs, since it is just returning a stream of RGB and depth images. I also specify that this class can run in its own thread so that it doesn’t block the main loop.

An object that interfaces with the motor controller would take inputs for steering and throttle instead. Also, any code which wants access to the camera images would simply need to take ‘cam/image\_array’ as an input.

On Friday I will test the OakD class to make sure it conforms to the specification needed by the DK library and that I can see the camera feed over the web interface.