David Spalla

Ryan Adams

Brennan Powers

selfDrive.py

Once we had all of the resources installed, we needed a way to control everything from one easy to use python file. For this, we created a python script called selfDrive.py. The first thing we needed to set up was a way to talk with DeepGTA, for this we used VPilot. VPilot adds functions that allowed us to create a benchmark scenario and easily manipulate in-game mechanics. This was beneficial because the way that DeepGTA takes commands is via JSON message packets, we could have possibly worked with this however VPilot makes it more efficient. The first thing that selfDrive.py does it set up a model class which is used to control the throttle, brake and steering of the vehicle assigned in the scenario. It then establishes a connection with DeepGTA using port 8000 as a default. The next thing done is creating the scenario. The scenario creates a benchmark that is constant for all of our tests. The most important constant is the vehicle we used. We decided to use a motorcycle, more specifically one called the ‘Vader’, we decided to go with a motorcycle because of the small footprint it has on the LIDAR as well as its maneuverability. We quickly realized that there is another important variable in the scenario which controls the driving mode. Depending on what number is passed controls how the car behaves – whether it’s self-driving in GTA’s AI or manual driving. We found that the correct value for this is -1 however we made it easily changeable in the file so the effects it has can be seen. This file has a proof of concept code that is currently commented out, this code checks the LiDAR file for points which are red (red represents vehicles) and changes the command to apply the brakes. We ultimately commented this out because it is very dependent on the LiDAR config settings and it is not completely dialed in yet. The end of the file ends communication with DeepGTA and closes the client.

LiDAR Manipulation

For taking LiDAR images of the game we used a pre-existing system called ‘liDAR GTAV’. We used this because it was one of the most well documented LiDAR systems for GTAV and it did what we needed. The biggest issue was that it required a button press to activate and the key could not easily be rebound to another. To automatically press this key we used an extension of python called ‘pyautogui’ which simulates a keypress. selfDrive.py allows the user to run the program without taking LiDAR samples which can be beneficial for testing. Another big issue we ran into regarding LiDAR was finding the correct balance between runtime and having enough points for valid data. We found that if we had the distance set too high in the config, it ran too slow and the program would cut off the rest of the points, and in some cases – would crash our games. After the distance we then found the best vertical and horizontal steps that still allowed the program to run efficiently and still show enough data. If performance was not an issue, we would have left these as default.

Creating LiDAR Files

For generating LiDAR Files, we came to the conclusion that the fastest way to generate them would be to completely rewrite the LiDAR GTA V.cfg file in each iteration. This is possible because the configuration stays the same throughout the DeepGTA V session, the only thing that needed to be changed was the output file. Therefore, instead of reading and writing to edit it, which would have required opening the file twice, writing the file contents as a string, which is only a few lines, with a counter to change the file name is faster, only requiring the file to open once. This is all done in order to have the least amount of downtime acquiring the LIDAR’s during computation time. The sleep statements are added so that enough time is allowed to both rewrite the cfg file and gather all the point cloud data.

Using Matlab to animate Point Cloud files

First I made a simple plotter to show a single frame of the point cloud. This is what allowed us to see what the robot was seeing and adjust our lidar configuration appropriately. This is on the GitHub as ply\_visualizer.m and is definitely not the most efficient script but we needed something fast and I made it in an evening. The next step was to animate a series of frames. The code for this is in the file called animate\_test.m and is much more efficient as I took the time to properly use matrix manipulation. It is currently set to hold all of the frames so you end up with a large point cloud map, but the “hold on” and “hold off” commands can be put in different places to animate a single frame at a time more like a movie. This is a bit harder to understand when watching unless you move very little distance since the last frame was taken, which is why we have it the way we do.