AUTONOMOUS NAVIGATION ON ROAD

AuE 8240 Autonomous Driving Technology

Abstract

This report is a final report of the capstone project for the subject Autonomous driving technology in Spring 22. This project aimed at implementing various controller strategies learned during the course of the semester.

Harshal B Varpe

Introduction:

Autonomous vehicles are no longer a subject of fantasy. We are yet to make a fully autonomous vehicle; however, the current autonomous vehicles can keep track of the lane and following it. An autonomous vehicle is one with multiple sensors working together to navigate the known or unknown environment.

However, advanced the technology may be, it has its roots in the rudimentary techniques. The project introduced us to the simple concept of implementing autonomous navigation by using simple tools and techniques.

Problem Statement:

The problem statement of the project was to make a controller for a scaled RC car that follows track and reacts to the signs upon recognition. The problem was divided into multiple tasks. Task 1 was to autonomously track the lane clockwise using a camera and a laptop as fast as possible. Second task was to recognize a stop sign and a school zone sign using a second camera and second laptop. Third task was to establish a communication channel between two laptops and pass the information from the second laptop to the first laptop through Wi-Fi. The fourth and final task was to control the vehicle speed depending on the sign. The vehicle should stop for 2 seconds and continue at normal/high speeds. The vehicle should reduce the speed by the half at school zone sign.

Figure 1. Project Track

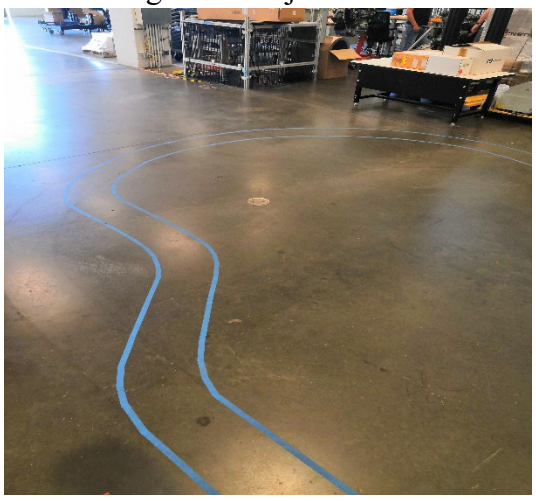

Figure 2. Image of Actual Track

Hardware:

For this project we were provided with the following equipment.

1) 1 RC car with battery
2) 1 Arduino kit
3) 2 Logitech HD C615 Camera
4) USB extension cable

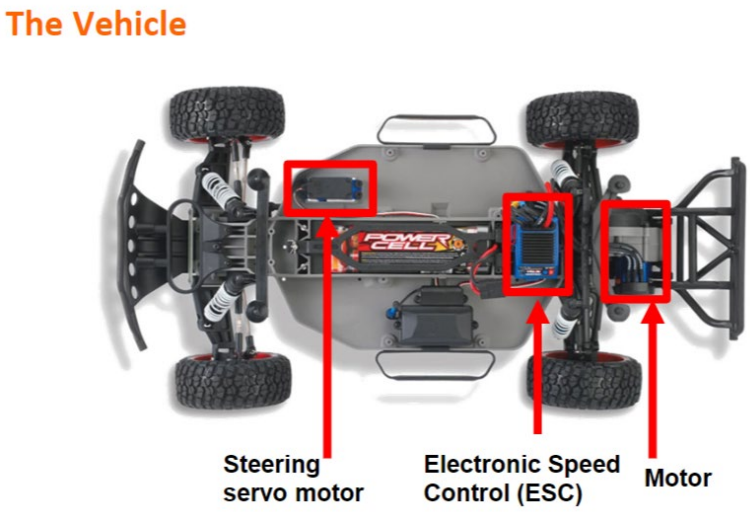

Figure 3 RC car

The RC car originally is bare bone chassis with no mounts on it. Hence, to be able to mount the camera and Arduino board on the vehicle, we had to some modifications. We added aluminum chassis as shown in the picture below. The aluminum frame also gave us a mounting point for our camera.

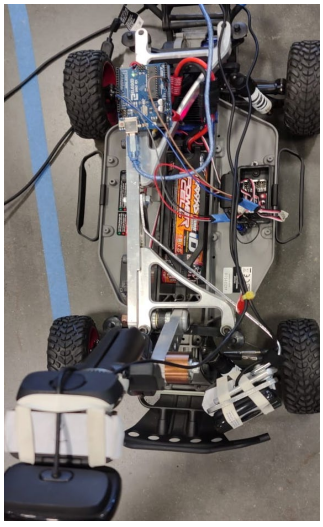

Figure 4 Our RC Car

Figure 5 Camera Positions

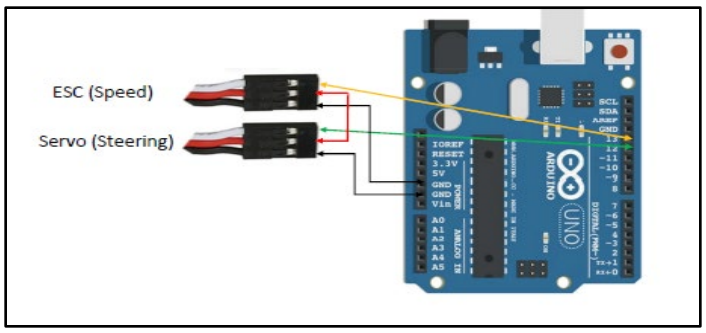

Figure 6 Arduino Connections

This concluded the hardware setup for this project.

Software Implementation :

1. Camera Calibration

We had divided the solution approach into different subtasks. The first subtask was to calibrate the camera and check the servo minimum activation value and stopping value. Before we begin with the camera calibration, let us talk about how to interface MATLAB and Camera.

First check if the camera is connected. To do that use the “webcamlist” command. If the camera is connected, then initialize a variable that stores the camera information. For example, I have initialized the integrated camera of a laptop in the following image.

```
>> webcamlist

ans =

  1×1 cell array

    {'Integrated Camera'}

>> cam = webcam("Integrated Camera")

cam =

  webcam with properties:

                       Name: 'Integrated Camera'
    AvailableResolutions: {1×9 cell}
              Resolution: '1920x1080'
               Sharpness: 50
           WhiteBalance: 4500
                    Hue: 50
           ExposureMode: 'auto'
                  Gamma: 50
               Exposure: -6
   BacklightCompensation: 0
             Brightness: 50
        WhiteBalanceMode: 'auto'
              Saturation: 50
                Contrast: 50
```

Figure 7 webcam initialization

Now, let us begin with camera calibration.

We had two cameras for two different purposes. One camera was used for autonomous lane detection whereas the other camera was used for sign detections. One of the reasons as to why we had to rely on two camera is that for proper lane detection one camera had to be positioned higher and looking down. This was done so that lanes are extracted properly. This meant that this camera could not “see” the signs. The secondary camera was mounted on the front left of the

vehicle, since the signs were going to be in the left. {may be put this in hardware} Now that we have talked about the camera locations, we move on to camera calibration. We only did the camera calibration for the primary camera i.e., the camera used for lane detection. The camera calibration was done to extract only the intrinsic parameters. Since the camera was mounted front and center of the vehicle, there was no need for extrinsic calibration. This is because if camera were to hit something the vehicle would it too or in other words there was no need for depth information for this setup of the camera.

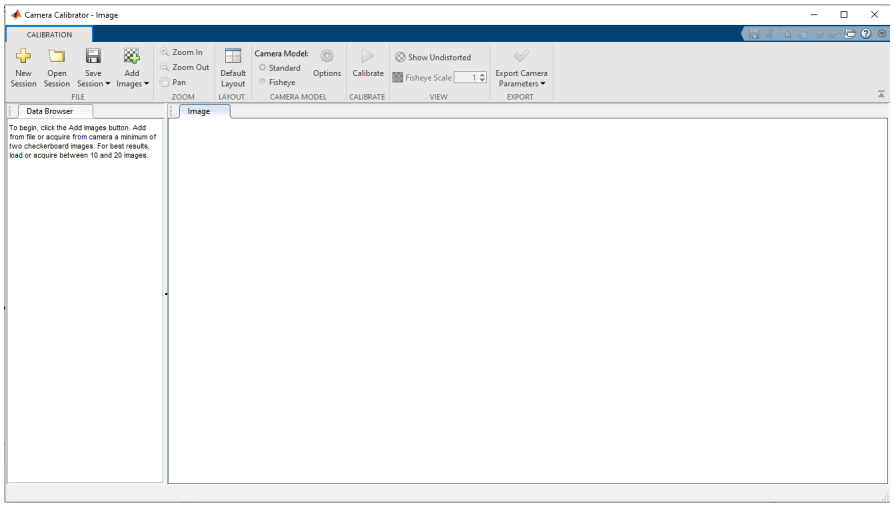

Figure 8 Camera Calibrator

The image above shows the MATLAB camera calibrator interface. To calibrate the camera, click on add images and the choose from camera. Choose the appropriate camera and images. Generally, the greater number of images mean better calibration results. However, we chose 20 images which was set by default. After the images are added, enter the parameters of the checkerboard i.e., the length of the side. Camera calibrator app will detect the corners of each square in each of the images and then calculate the camera parameters. If everything is done right, now we can export the camera parameters as mat file or directly to the workspace.

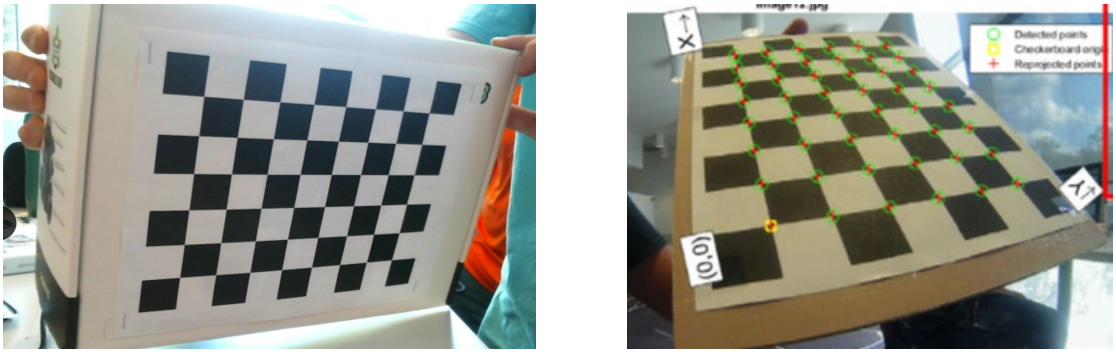

Figure 9 Camera Calibration process

Now we can use the camera for the lane extraction and following.

2. Controller

The controller had two aspects to it. One was to steer the vehicle after detecting lane turns. The second aspect was the velocity inputs to the vehicle.

Let us first talk about image processing for lane extraction. As mentioned previously, the primary camera was mounted front and center and slight downward tilt. The resolution of the images captured from this camera is 1920 x 1080. Since the raw image size was high for image processing, we had to resize the image to 0.25 times for higher data handling rate. We used region of interest to mask the certain part of image. We will explain the reason in subsequent paragraphs. We created a RGB mask that would mask out everything apart from the blue lane lines. Without ROI and RGB mask, the glare from the lighting above would cause errors in lane detection. We also tried to use the HSV filter, however, we found out that the RGB filter yielded in better results than HSV filter. The correct RGB values we chose are – R [90,160], G [120,230] and B [190,255].

We then used this color filtered image for edge detection. We canny edge detection method to extract edges from the images / live video. After the edge detection, we used the *imfill()* function to fill the holes in withing the detected edges.

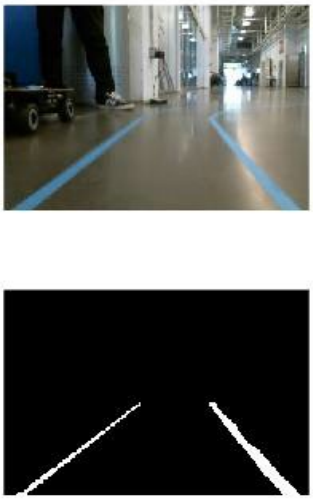

Figure 10 Edge Extraction with canny edge detector

Since the edges extracted by the canny edge detector are not always straight, we use the houghlines function to get straight lines corresponding to each lane marking. Houghlines function allowed us to minimum length of one a single continuous edge to be considered along with the distance between two edges. With this function we calculated the coordinates of the two lines for the left and right edges. These coordinates are then used to calculate the centerline of the track. There were few corners along the track where the vehicle would lose the track. In such

cases, we assumed the missing line to be away from certain distance from the first line and at angle of the first line.

Once the centerline of the track was obtained, we used the slope and the distance between the edge points and vertical line (called as error) to steer the vehicle. The image below shows the lane detection and centerline calculation.

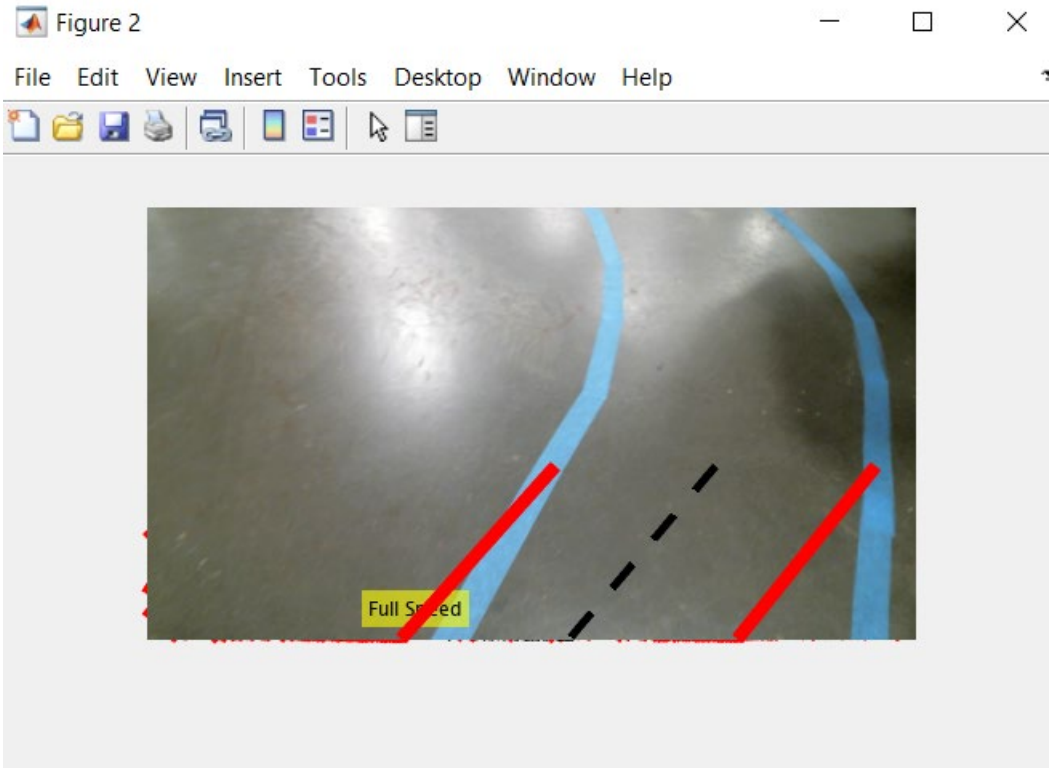

Figure 11 Lane detection

After we fine tuned our image processing part, we moved on to the creating a controller for steering only, since the velocity was constant for the entire track. (Half of this constant in the school zone.)

Our first approach to the problem was using a Stanley controller. However, when accounting for the departure distance and departure angle, the steering angle (delta) was a bit jittery. Then we moved on to the simple PI controller with proportional gain of (k_p)= 0.007 and integral gain of (k_i) = 0.01. During our initial runs, we only used the proportional gain. However, this also led to erroneous results. Moreover, we also had normalized the steering angle we received between [0 – 0.5 - 1]. The equation for calculating the steering angle is given below.

$$\text{steering } (\delta) = kp * \text{error} + ki * (\text{error} - \text{error}_{prev})$$

```
if (steering > -0.1) && (steering <= 0.1)
    s_input = 0.5;
else
    s_input = 0.5 + steering/6;
    if s_input > 1
        s_input = 1;
    elseif s_input < 0
        s_input = 0;
```

Figure 12 actual steering inputs to the vehicle

Since the steering angle calculation depends upon the line detected, steering inputs were to sensitive and vehicle would keep correcting constantly. To make this steering smoother, we mapped it like in the image shown above.

For motor controller, we tried several approaches. First approach was by using the pause function between publishing each velocity commands. However, along with steering inputs the vehicle would not behave as expected. After that, we moved on to approach where we would pulsate only after certain number of loops of the main controller. In this approach we would accelerate the vehicle each time it almost came to stop.

```
elseif data == 5 && count == 1
    writePosition(v,0.485)
    pause(0.2)
    writePosition(v,0.504);
    sign = "Full Speed";
```

Figure 13 Code snippet od velocity control

3. Sign detection and communication

Once we managed to get our vehicle follow the track perfectly. We moved on to the next task of sign detection and communication. For the sign detection, we first tried Cascade Object Detector with MATLAB. For the cascade object detector, we captured the images of the stop sign and school zone sign form various angles and distance. We then labeled these images manually and trained the model. However, the results of this model were not perfect.

We then had to move to deep learning model for sign detection. We used the Deep Network Designer and chose a pre-trained model and transfer learning. For this purpose, we tried to use GoogleNet. For this model we had three classes – Stop sign, school zone and none. However, we found that the model was overfit, due to small size of the dataset.

In the end we settled on the resnet-18 model. The model had around 71 layers with 77 connections and 11 million parameters. This model was much faster and accurate than any other methods we tried.

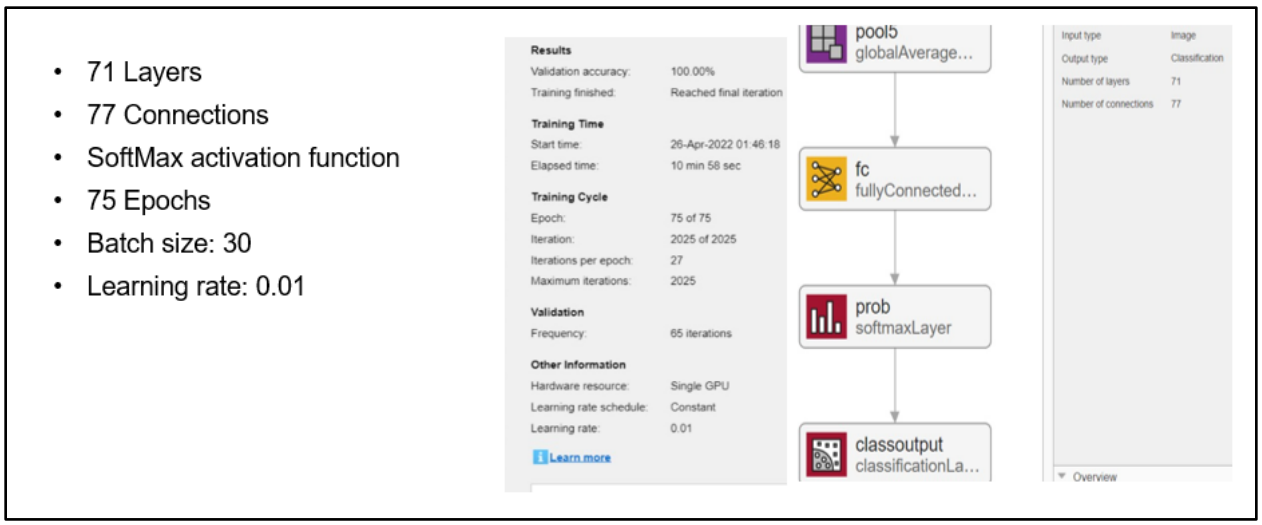

Figure 13 Resnet training model and parameters

After all these task, the task of establishing a communication channel between the two laptops such that one laptop would detect signs and then pass the information onto the first laptop with controller was easy. For this purpose, we would simply pass a integer value of 0 and 1, where o means no detection and 1 means positive detection. We also passed the values only after the possibility score was above 90 percent. This made the vehicle stop at proper distance away from the stop sign and reduce speed at the school zone sign.

Challenges:

One of the major challenges, my group faced was the fact that the camera would detect the signs a bit early. We realized it was the issue with the confidence rating we had set. For testing purposes and dealing with various lighting conditions, we had set the confidence rating on the lower side. However, in the end we changed the confidence rating from 80 percent or 0.8 to 90 percent or 0.9. This helped us achieve the required results.

Another challenge was speed control of the vehicle. As explained in the previous section, we had tried multiple approaches to control the speed of the vehicle. However, later we found out that the vehicle speed was greatly affected by the State of Charge of the battery. This meant controller tuned for one SoC would behave erratically, if the SoC was lowered or if the battery was charged.

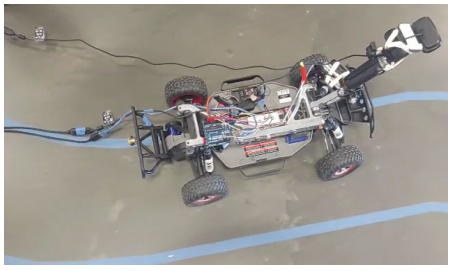

Figure 14 Vehicle Out of Track

Environment, especially the lighting conditions greatly affected the performance the lane detection algorithm. Our team had tried both the HSV and RGB approach for proper lane detection. To deal with this problem, we changed the mounting position of the camera. We increased the camera height and tilted it downward so that the glare would be minimum. Moreover, we also changed the blue values for the RGB in lane detection algorithm to accept wide range of blue colors.

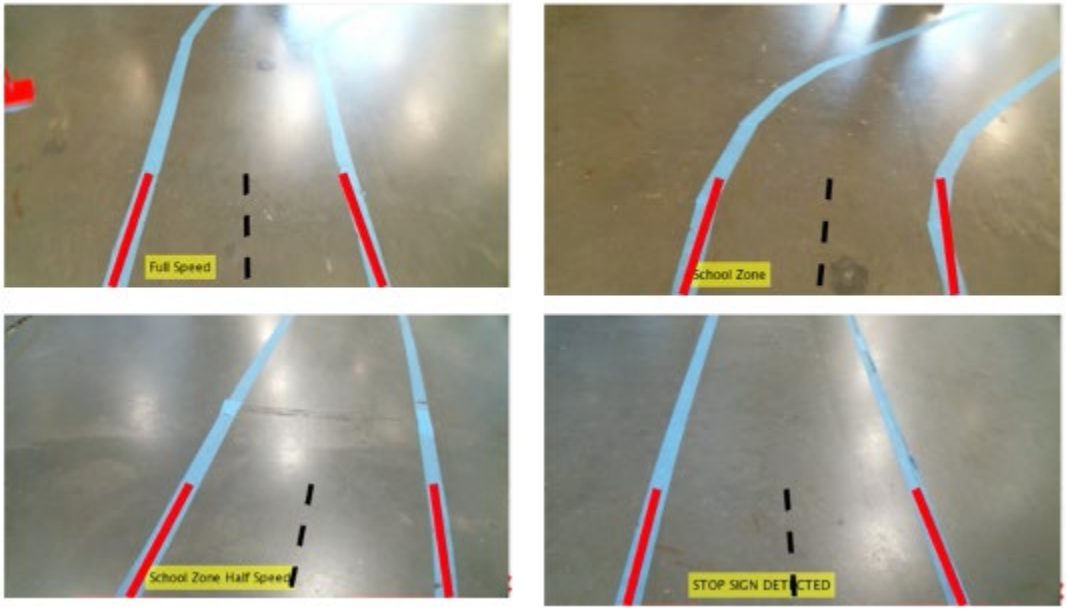

Figure 17 Lane Extraction and Sign detection output

Result:

To test the steering behavior of the vehicle, we also plotted the graph of steering angles against time. The x-axis represents time, and the y-axis represents the steering input given.

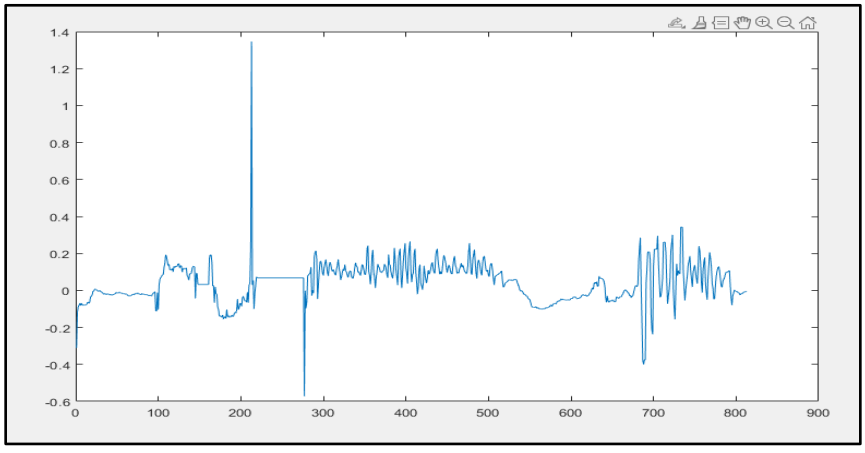

Figure 16 Steering Input Graph

The sharp spikes denote the sharp corners along the track. The area of the graph with spike of the same amplitude represents the minor correction the vehicle tried to make along the track.

All the task for this project were successfully done. For the autonomous lane keeping task, we were able to create a controller robust enough extract lanes properly and follow them throughout the track without much deviation.

For the tasks of sign recognition and communication, we were successful in recognizing the signs and passing these detections to another laptop which would pass these to Arduino and stop the vehicle at stop sign and reduce the speed at school sign.