Autonomous Lane Keeping and Road Sign Detection for a scaled down RC vehicle

Aditya Parameshwaran (Group 4)
PhD in Mechanical Engineering
Clemson University
aparame@clemson.edu

***Abstract*—The aim of this group project is to implement the autonomous driving skills learnt in class to develop an autonomous lane keeping and road sign detecting RC car in a known environment. The tasks of the project were broken down into sections based on the requirements for successfully navigating the RC bot around the track. We began by identifying the layout of the track and determining the best camera locations for the bot. The image data was pre-processed to be used by the lane keeping and sign detection algorithm. A CNN based on the ResNet architecture was trained for sign detection using images captured by the bot's secondary camera. The result was a autonomous RC bot that was successfully able to identify and navigate through the track, while detecting the STOP and SCHOOL signs accurately and reacting. appropriately.**

I. Introduction

Autonomous driving has become one of the fastest growing and rapidly engaging technological advancements of the 21st century. Autonomous vehicles (AV) are developed with a strong combination of technical knowledge and run time testing. AV's are only as robust as the developer designs them to be and this results in the need of a rigorous hands on testing for the algorithms running the car.

The technical advancements of AV's indulge in a combination of engineering fields focused on the image processing, data learning, model prediction and vehicular controls. The Autonomous Driving Technologies coursework project particularly aims at bridging the gap between the plethora of information regarding AV technologies taught in class and its implementation on a real life RC car. The course is a good hands on course that allows the student to enter the field of AV with a solid base setup for further specialization.

II. Problem Statement

The goal of the project is to develop an autonomous RC car that is capable of driving on a indoor track and reacting to road signs along the track autonomously. The project was divided in the class and groups of 4 to 5 members were made for the project. The objectives of the project were to:

1) Develop and Lane Keeping algorithm for the RC bot
2) Develop a Road sign detection algorithm
3) Enable Wifi communication between the 2 algorithms
4) Successfully test the bot around the track for 3 laps

We segregated the tasks to each of the team members based on the project objectives and we were able to navigate the bot on the track autonomously for 3 laps as tasked.

A. Track Setup

The track is made of 2 blue tape lines that run parallel in an oval shape with a stop sign and a school sign placed approximately at the start and midway through the track. This can be better visualised in Figure 1

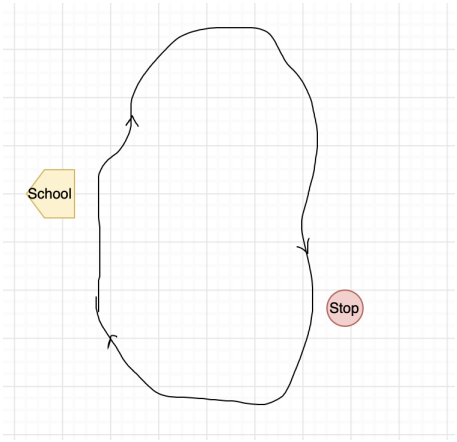

Fig. 1. Track Setup with locations of Stop and School Signs

As seen in Figure 1, the track has 2 road signs. The objective of the bot is to slow down to half speed once the SCHOOL sign is identified till the STOP sign, where it will stop for 4 seconds and then continue back to normal speed. The blue taped lines are almost exactly the width of the bot. Hence this calls for accuracy in determining the correct longitudinal velocity and a responsive lane keeping controller to avoid steering off track quite often.

B. Autonomous Vehicle Setup

Setting up the vehicle was an important task as it is instrumental in determining the parameters for the lane keeping and sign detection algorithms. The vehicle is a rear wheel electric motor powered remote control bot. Along with the bot, we were provided with 2 cameras, 1 Arduino board for serial I/O with the ESC and servo motors, and a bunch of jumper cables. Figure 2 shows that both cameras and the Arduino board are securely mounted to the strut, which resists vibration due to the higher tolerance of the metal struts used. Therefore not only are our cameras adjustable, but we they were also resistant to distortion of image feedback due

to vibration. The front camera was mounted to a height of a 387.5mm for lane keeping, and a 151.5mm height with a 50 $\deg$ angle on the side camera for sign recognition.

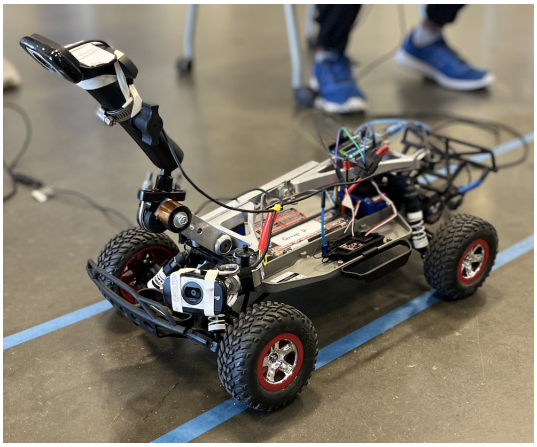

Fig. 2. RC Bot Setup

C. Camera Calibration and Parameter Identification

As mentioned in Section II-B, the vehicle has 2 cameras for road sign detection and lane tracking. The main camera, which is located at a height of 380mm, passing over the roll axis of the bot, is used for lane detection and tracking. The secondary camera is located facing at approximately $50\,\deg$ from the roll axis, pointing primarily towards the road signs. The cameras need to be calibrated for the intrinsic parameters only as we are mounting the camera's directly above the front axle of the vehicle. As the cameras are both of the same model, the intrinsic parameters were calibrated using the "Camera Calibrator" app on MATLAB. To calibrate the camera, we click images of a checkerboard and upload them to the app. Along with the image, we also need to upload the length of the side of a square in the board. The app then takes care of the rest and sends out the intrinsic parameters of the camera to a *.mat* file, that can be further used for controller design.

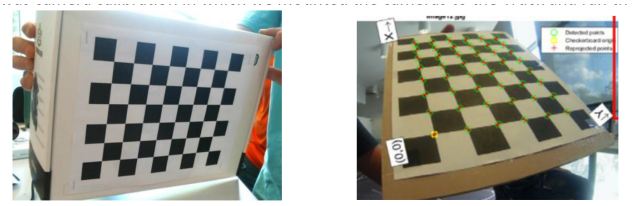

Fig. 3. Camera Calibration

III. Lane Keeping Controller

The Autonomous Lane Keeping module comprises of extracting useful data from raw camera images and an algorithm that controls the steering input of the vehicle based on the extracted data. The only form of data we have is images from the main camera for the lane keeping action. Hence we need to pre-process the images for useful data and after that we can extract the data. The information from the data is then interpreted as error for Proportional-Integral (PI) steering controller and the steering inputs are sent to the servo motor.

A. Image Processing

The front facing camera for the bot acts as the main and only source of data extraction sensor for the bot. The camera is angled at about $45\,\deg$ from the horizontal facing down towards the lane as discussed in section II-B The images from the main camera are received in natively in 1920 x 1080 format. The raw image size is quite high for image processing on run time hence we resize the image 0.25 times for a higher data handling rate. To work with a more compact data, we use the region of interest concept to crop the image to the lower half along the horizontal. This allows us to save up on data and provide a faster response for the controller.

1) RGB Mask: During some of our initial tests, we realised that for best possible accuracy in lane detection, it would be best if we were to use an RGB mask that would mask out everything but the blue tapped lines. Without the mask, we were facing errors in determining straight lines due to the glare produced by the lights and the every other straight line on the floor apart from the tapped line. After a few trial and error runs, we found the perfect range for the RBG values to be $R = [90,\ 160]$; $G = [120,\ 230]$; $B = [190,\ 255]$ .The mask allows our edge detector to work perfectly with the blue tapped lines only.

2) Canny Edge Detector: Canny edge detection is a technique to extract useful structural information from different vision objects and dramatically reduce the amount of data to be processed. The edge detector allows us to detect the edge lines for the track. The edges are bounded between a threshold of $[0.2,\ 0.4]$ for the MATLAB *edge()* function. After the edge detection, we convert to image to gray-scale and then use the MATLAB *imfill()* function to fill any holes within the lines. Figure 2 shows the comparison between the original image and the image after edge detection and RGB mask

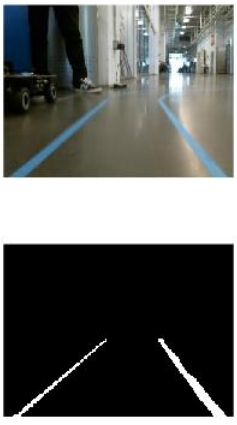

Fig. 4. Normal vs Processed image

3) Hough Lines: The Hough Transform is a popular technique to detect any shape, if one can represent that shape in a mathematical form. It can detect the shape even if it is broken or distorted a little bit. The Hough transform is used to determine the properties of straight lines from the edges shown in the final image processed from the Canny Edge detection section. The *houghlines* function allows a set of minimum length to be considered as a line and the max gap allowed between sets of edges to be determined as a line. The function provides 2 set of lines on the image for the left and right track edges. The set of end coordinates for the lines are then used to determine the track coordinates in image frame, from which we can calculate the dimensions of a line passing right through the centre of those lines. That line would be the reference line for our bot to follow. In some instances, the camera might lose track of one of the tracks. In such cases, the reference line is considered to be at a specific distance from the other track in the image frame.

B. Controller

The reference line is basically necessary to calculate the deviation of the bot's orientation and the distance of the centre of the bot from the middle of the track. As mentioned earlier, the camera is mounted such that it lies exactly on the roll axis of the bot. We can estimate that for the bot to be headed on the track, it has to have the reference line close to parallel to the tracks. With this estimate in mind, we can consider the distance of the 2 end points of the reference line from the central axis as the error in our heading orientation. This error can be visualised in Figure 3

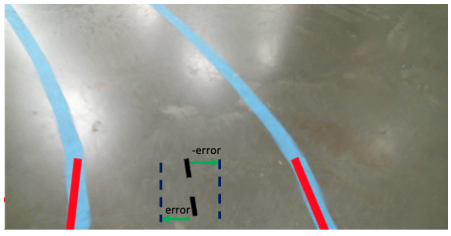

Fig. 5. Reference line and Lane error

The error shown in the above figure what is then interpreted as the required steering angle and eventually the servo motor input value. The error values are used in a PI controller to calculate the required steering input at the time step. The Proportional gain k_p = 0.007 and the Integral gain k_i = 0.01. We first tried running the bot with only a proportional gain, but that resulted in multiple deviations from the set point, that led to unstable fluctuations about the reference line. The integral controller was included to reduce said fluctuations. A set of transformations are required to convert the steering angle input to a [0 - 0.5 - 1] normalised system as the servo motor that controls the vehicle steering takes only such values as input. steering (δ) = Proportional control + Integral control, where:

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

IV. Road Sign Detection, Control and Communication

A. Data Collection and Model Training

For the road sign detection, we were tasked to train a Neural Network to be capable of detecting the STOP and SCHOOL signs. We first trained a cascade object detector to detect the signs. Unfortunately, the object detector was not really accurate for our use case. We also had to provide the model with a null use class that would be contain images of everything but the STOP and SCHOOL signs. In order to train the object detector, we used the images given to us as well as a few more images we clicked ourselves. Unfortunately, for the object detector, we needed to label the entire data set and also provide a null data set

To avoid that process, we worked with a Deep Learning

Fig. 6. STOP and SCHOOL Sign on track

model for the detection. The model was trained on the "Deep Network Designer" app on MATLAB where we chose a pre-trained model to perform transfer learning on the same. First we chose GoogLeNet CNN, trained it with images of STOP and SCHOOL sign. The GoogLeNet model is 22 layer deep Convolution Neural Network that usually used for adversarial training and takes 224 x 224 sized images for inputs. For the training we had three classes, STOP , SCHOOL and NONE. The images in the NONE class have images from the track without the STOP and SCHOOL sign. The training results showed that the model overfit, this was due to small size of the dataset. Moreover, the model was also inaccurate, so we had to look for accurate models. Then we tried the "Resnet-18", this model has around 11 million parameters, 71 layers and 77 connections. We trained the model similarly as the GoogLeNet and tested it. This model was faster and was more accurate compared to the previous models. For higher accuracy on track, we ran the bot along the track manually and recorded images from the secondary camera as they would look like on run-time. This resulted in more scenario accurate images with consistent robustness.

B. Longitudinal Speed Controller

The first step was to calibrate the ESC and Arduino to understand the base digital values for which the motor starts and stops rotating. The longitudinal speed control was designed as an extended *if-else* loop based on the location of the bot on the

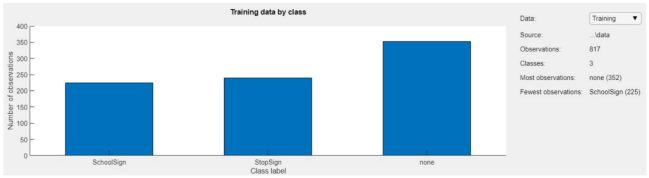

Fig. 7. Classes and the number of observations required for each

track. The values for the ESC Start-Stop were then built in to the *if-else* loop with a binary counter that decided whether we had passed the STOP sign or not. This counter was necessary as we needed to have full speed on the bot once it crosses the STOP sign and half speed once it crosses the SCHOOL sign.

```
if mod(ii,3) == 0
    if data == 5 && count == 1
        writePosition(v,0.486)
        pause(0.2)
        writePosition(v,0.5040);
        sign = "Full Speed";
```

Fig. 8. If-else loop for the Longitudinal Speed Controller

C. WiFi Communication

To run the entire setup as discussed above, the team uses 2 separate laptops that communicate among each other with the MATLAB *udp* block. User Datagram Protocol (UDP) is a communications protocol that is primarily used to establish low-latency and loss-tolerating connections between applications on the internet. UDP allows the primary computer that runs the Lane Keep Controller to receive data from the secondary computer that runs the Pre-Trained CNN. The secondary computer is connected to the secondary camera, and has the job of identifying the class of the detected road sign and send it over to the primary computer via a 1-to-1 direct communication portal. The primary computer interprets this data and decides the control action for the vehicle to react to. To enable a faster data transfer rate, we used a local area network that was private access only to the 2 laptops and had data being sent in 3 integer values only.

V. Results

The RC bot was successfully capable of travelling 3 laps around the track while following all of the road signs and reacting accordingly. The time taken to travel around the track once was about *2mins 34 secs*. The following plot shows the variation in the servo motor inputs all around the track.

Figure 9 shows that the turns are heavily favoured towards the right as the we had to traverse the almost circular track anti-clockwise. Also the fluctuations in the steering input value was particularly due to the fact that the bot had a favourable gain towards correcting its path. Without the Integral controller, the bot would have even higher fluctuations and cause

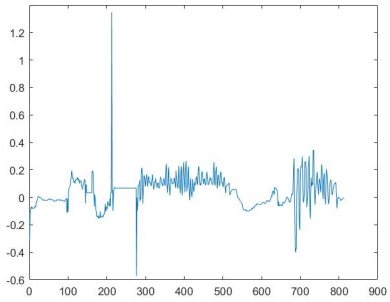

Fig. 9. Servo Motor inputs for steering the bot (where 1 = Max Turn Right, 0 = Max turn Left, 0.5 = Neutral)

the bot to move off track. Apart from the steering angle, the bot successfully identified all of the road signs and varied the longitudinal speed based on the controller described in Section IV-B. The plot shown in the Figure 10 shows the different locations the bot is in based on the external environment. The text displayed on top of the the lane markings allowed us to easily interpret if the secondary camera has successfully seen the sign and that we are receiving data accurately from the secondary computer via the communication block.

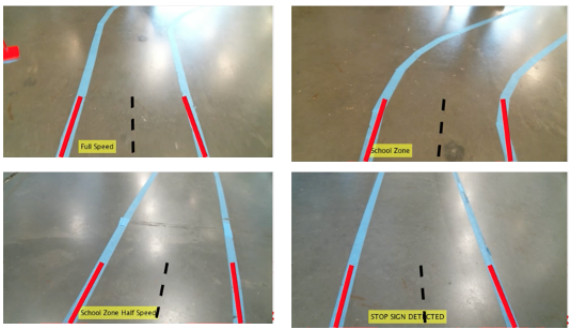

Fig. 10. Visualization of Signs based on the location of the bot on the track

A. Challenges

- *Sign Detection False Positives*: We faced an issue where the secondary camera was not quite accurate in detecting the difference between SCHOOL signs and NULL spaces. 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 distinguish better between the 3 classes and now with 100% accuracy.
- *Vehicle Speed Control*: One of the main issues was that the vehicle speed control was heavily affected by the State of Charge of the battery. This meant that the controller would be tuned for a specific SoC range and over time with the battery draining we would have erratic behavior. This caused us to decide a range for the SoC and always keep the batteries charged to that range.

- *Camera Mount and Image Pre-Processing*: The main camera aimed to detect the tracks was initially not sturdy enough to vehicle pitch forces. Also in the case of testing the bot, we have hit the camera a couple of times that changed its orientation. This caused us to create a sturdier mount and record the location of the camera mount this time. The blue tapped lines were first more difficult to segregate due to changing light conditions throughout the day. Hence we used a RGB mask to remove objects from the image that were not part of the track. This helped us identify the tracks with even more accuracy.
- *Error in Reference Line*: Sometimes when the track had a steeper curvature, one of the tracks would move out of the image frame. This would cause the bot to lose track of the required steering input to stay on track. To tackle this problem, we changed the lane keeping controller to remember the last known error and repeat to decrease that error proportionally till both the tracks become visible.

VI. Conclusion

In conclusion, we were successfully able to complete all of the tasks for the project. The autonomous lane keeping task was completed accurately with the vehicle traversing the track thrice with perfect repeatability. The data communication between the 2 laptops was successful without major loss in the data rate, and the secondary computer was able to identify the road signs with 100% accuracy. All of the required servo and ESC control inputs were calculated by the computers online and we were able to write them to the digital pins of the Arduino successfully.