AuE 8240: Autonomous Driving Technologies

Final Project Report

Autonomous Navigation on the Road

Course Instructor:

Dr. Yunyi Jia

Course Teaching Assistants:

Haotian Su

Yuchen Yan

Project Report by:

Mayuresh Bhosale

1 Problem Statement:

Autonomous Navigation on Road: The primary objective of this project is to apply and demonstrate the concepts of Autonomous navigation of a vehicle on the road by using Deep Learning models and Image processing using cameras.

Two tasks mimic the autonomous navigation using a small-scaled Remote-Controlled Car, two onboard cameras, and an Arduino Board. These two tasks are:

1) Lane detection and lane-keeping
2) Sign detection and velocity control
    a. Stop sign detection and stop for 4 seconds
    b. School sign recognition and slowing down at the school zone

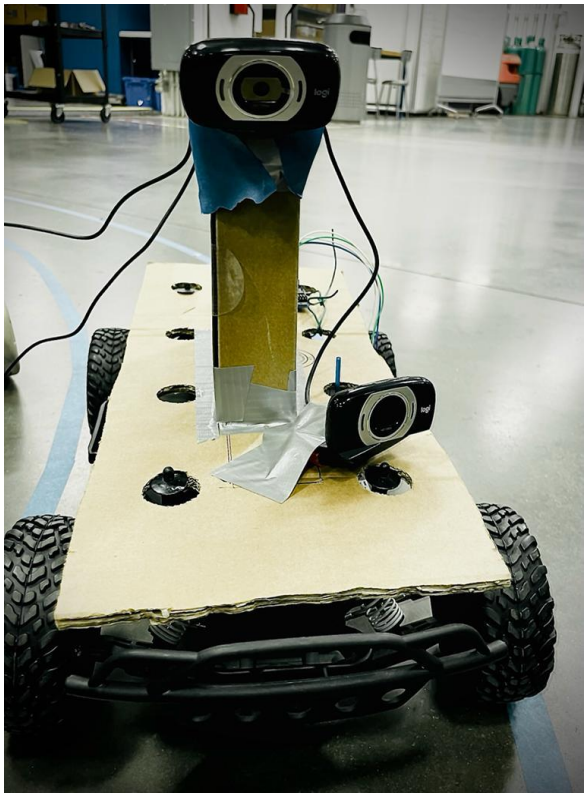

Figure 1: RC Car with two cameras and Arduino board

2 Introduction:

Advanced Driver Assistance Systems (ADAS) in current vehicle applications such as Autonomous Lane Keeping and Automatic Cruise Control have been highly effective in alerting humans and reducing accidents that cause injuries or fatalities. ADAS implementation is considered Level 2 of Autonomy (Partial Autonomy). Sign recognition and vehicle control further reduce human

interaction with the vehicle and can be an essential tool for developing fully autonomous vehicles.

This project applies two autonomous vehicle technologies in smaller-scaled RC cars using robust fail-safe algorithms. This is achieved by using computer vision and deep learning to recognize and apply vehicle controls accordingly.

The two Autonomous Technologies implemented on the RC car are:

1) Autonomous Lane keeping
2) Traffic Sign Recognition and Vehicle Control

The following sections are necessary tools and setup to apply lane keeping, sign recognition, and vehicle controls.

2.1 Hardware Setup

1) Arduino Uno
   Arduino Uno transfers the commands from the Computer to ESC and Steering Servo.

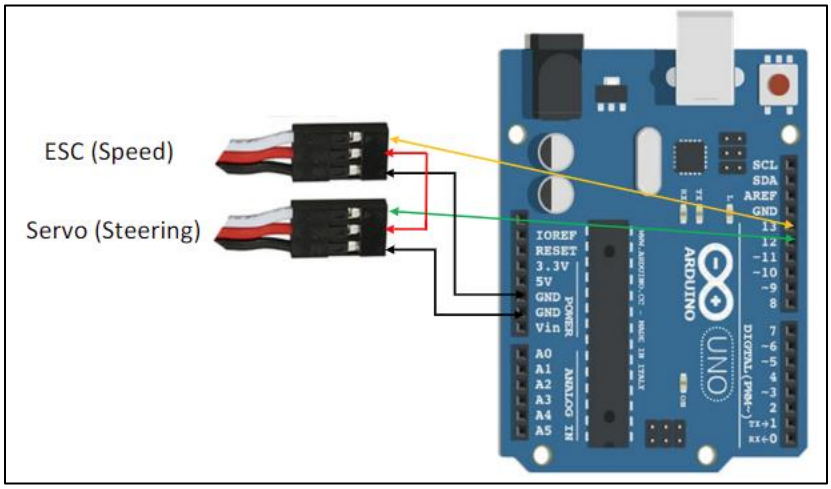

Figure 2: Arduino Uno board setup

2) Logitech C 615 Camera
   Two Cameras are used, one for lane-keeping and another for sign recognition.

Figure 3: Logitech C 615 Camera

Source: https://www.logitech.com/en-us/products/webcams/c615-webcam.960-000733.html

3) Traxxas RC vehicle

The RC car consists of a Steering Servo Motor and an Electronic Speed Control unit and motor for Steering and speed control.

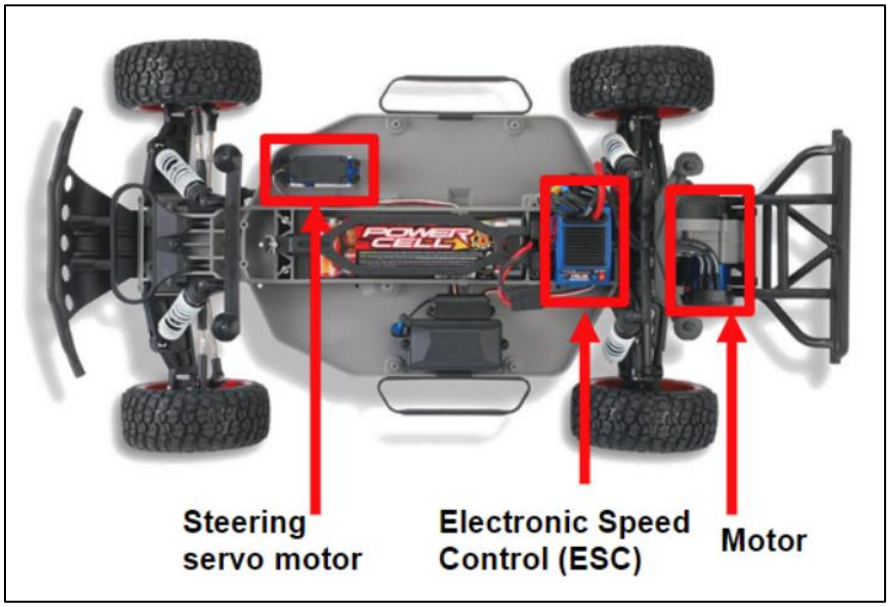

Figure 4: Traxxas RC vehicle

Steering servo motor requires 5-volt input for commands, and the circuit between the Arduino Board and RC car was connected using the Jumper Cables.

2.2 Software Implementation:

MATLAB R2021a version processes the image data received from onboard cameras and sends commands to the Arduino for steering and speed control. The computer vision toolbox in MATLAB is used for traffic sign recognition using deep learning models. Arduino Hardware Package in MATLAB enables it to control the input and output commands from Arduino Uno Board.

2.3 Camera Calibration:

Camera calibration is necessary to measure the object details accurately in an environment. The radial and tangential distortions can produce bad image/object detection, and Camera calibration can remove these errors.

The Camera calibration is done by using the Camera calibrator application from MATLAB. Known-sized checkerboard paper is printed and is used for camera calibration to set camera parameters such as focal length, exposure, and intrinsic properties. These camera parameters are further used for lane detection and traffic sign recognition.

Following is the flow chart explaining the camera calibration process:

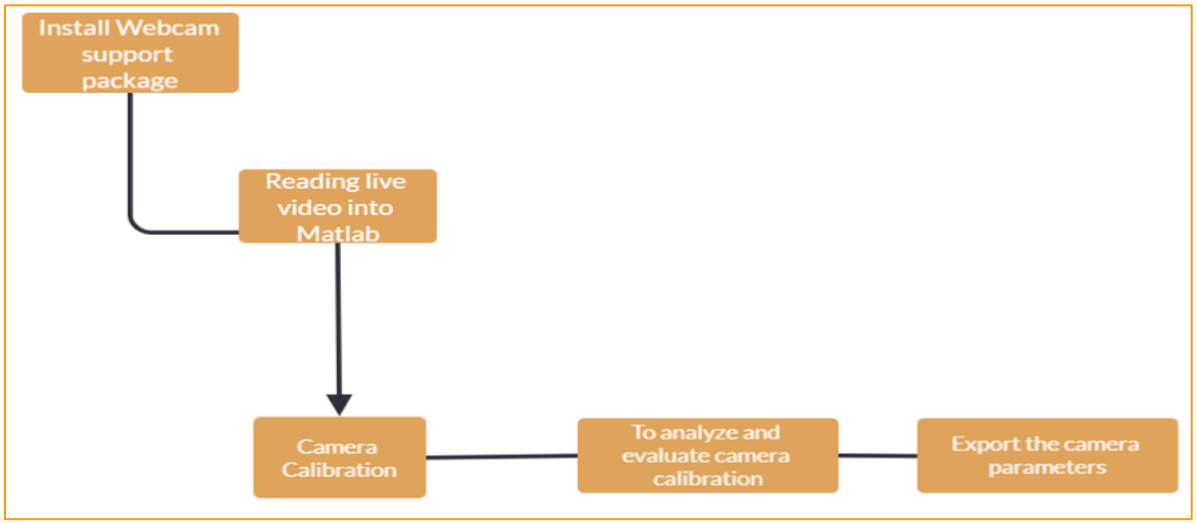

Figure 5: Camera calibration flowchart

3. Autonomous Lane Keeping

3.1 Methodology

Autonomous Lane Keeping is achieved by using multiple edge detection methods and applying different types of controllers.

The below flow chart explains the generic approach for Autonomous Lane-keeping:

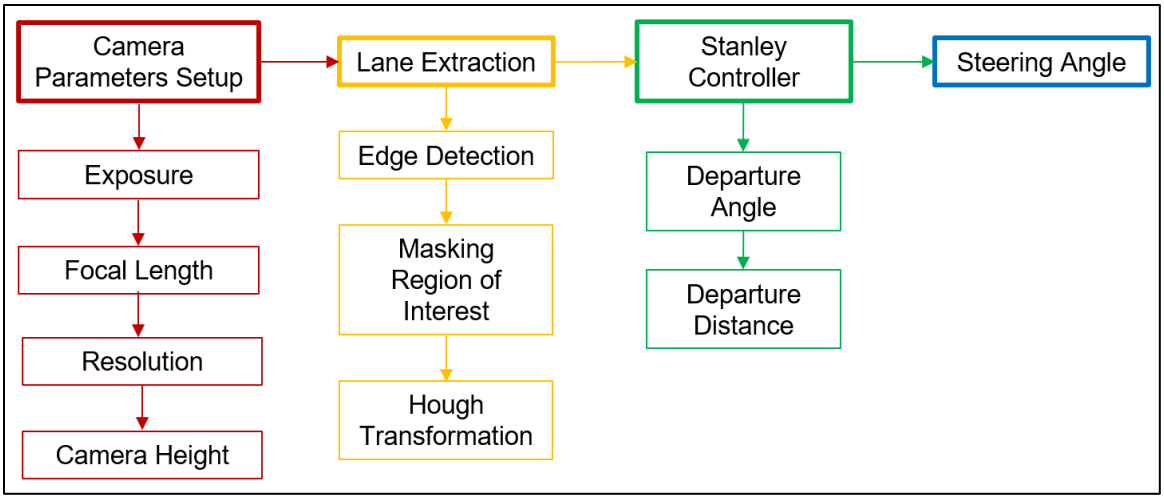

Figure 6: Autonomous Lane Keeping Approach

A more detailed version of the different configurations tried, tested and used, or disregarded is mentioned below sections.

3.1.1 Camera Parameters Setup

As much as Camera calibration is important to obtain the camera parameters used for the accurate edge detection, setting the camera to generate good live results is equally important.

Exposure Time: It was noticeable that different exposure times using the same camera resulted in more or less accurate edge detection. The test to set optimum exposure time was done based on the bright glare detected by the camera on the testing track. The objective of setting a specific exposure time was to reduce the bright glare without compromising the edge detection. An exposure time of 80 in Linux MATLAB or -7 in Windows MATLAB was considered the optimum to provide good results.

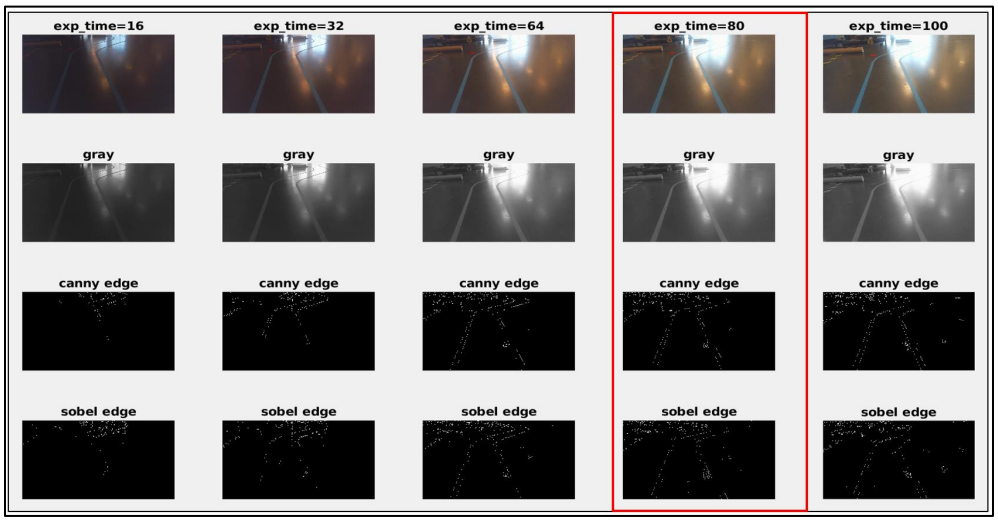

Figure 7: Edge detection concerning the exposure time

Camera Height:

While testing the camera, its height was one of the significant factors that were responsible for edge detection. Based on the analysis, it was observed that the Camera mounted High on the RC vehicle provided good edge detection compared to the low camera.

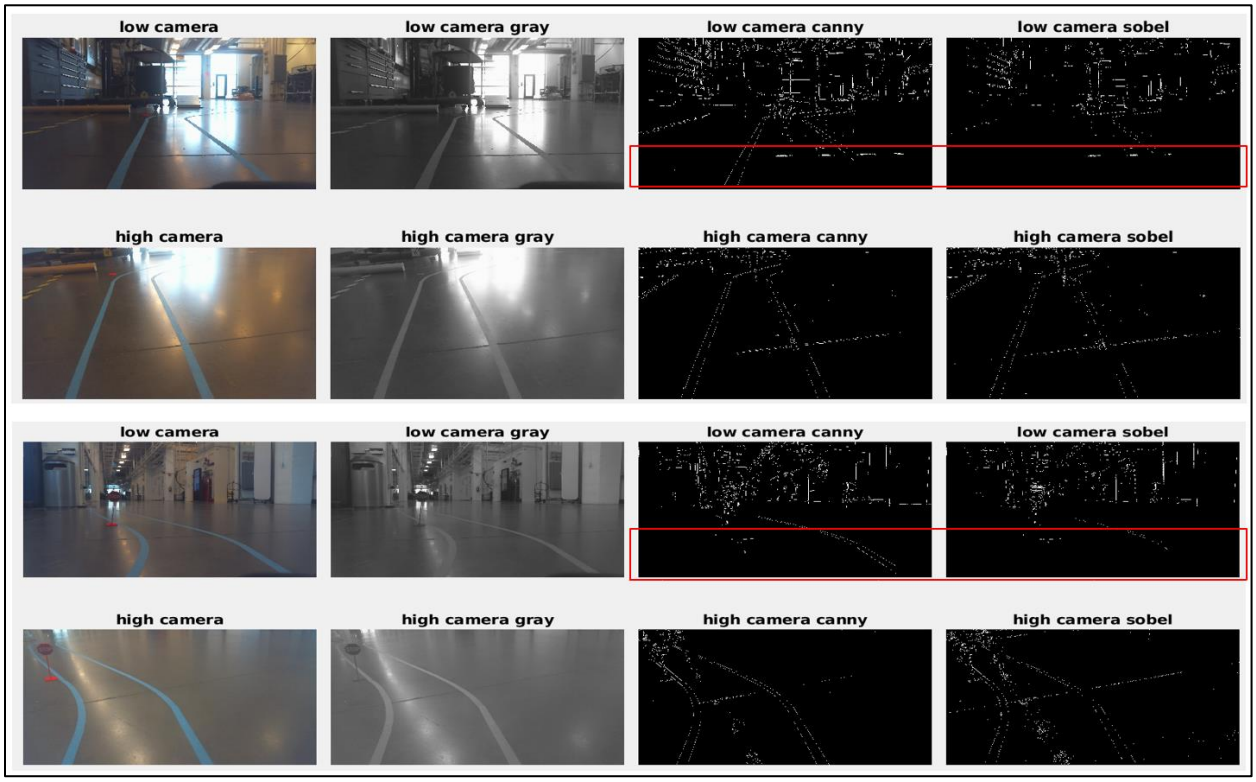

Figure 8: Edge detection concerning camera height

Focal Length and Resolution:

Multiple testing iterations were made to set one focal length of 51 and a resolution of 640x360. This was done to reduce the computation power while still maintaining edge detection's consistency and accuracy.

3.1.2 Lane Extraction:

The Lane is extracted based on edge detection and Hough Transform. Multiple techniques are established and analyzed for edge detection, and an optimum solution is selected based on accuracy and computation/time cost.

For Edge detection, the following methods were analyzed:

1) Canny vs. Sobel Filter based detection
2) RGB Edge Merge vs. Gray Scale
3) Image Rescale

4) Denoise
5) Gamma Correction
6) Histogram Equalization

1) Canny Edge Detection vs. Sobel Filter:

Canny edge detection is a more accurate approach to detecting edges than the Sobel Filter. In Canny edge detection, the Gaussian filter is applied to remove the noise present in the image. After removing the noise, there are several further steps based on Gradient intensity, thresholding, and noise/thick edge suppression by which an accurate result for edge detection can be achieved.
On the other hand, Sobel Filter works on the principle of Gradient Intensity classification for edge detection, where an edge defines the intensity difference in the image. As mentioned in the below image, the tested Sobel Filter applied to the image was computationally less expensive by giving similar results as Canny edge detection.

Canny vs. Sobel

Result on: AMD® Ryzen 5 5600h with radeon graphics × 12, 32GB memory
process 106 imgs, repeat 100 times, total 10600 run for each method:

Type	Time [s]	FPS
Sobel	0.001264	791
Canny [0.05,0.1]	0.007790	128
Canny [0.1,0.2]	0.007525	132

Figure 9: Canny edge detection vs. Sobel filter performance analysis

As seen in Figure 10, Lane edges are detected clearly in canny and using the Sobel filter. Similar results can be observed in Figure 11, with bright glare edges still detected using the Sobel filter.

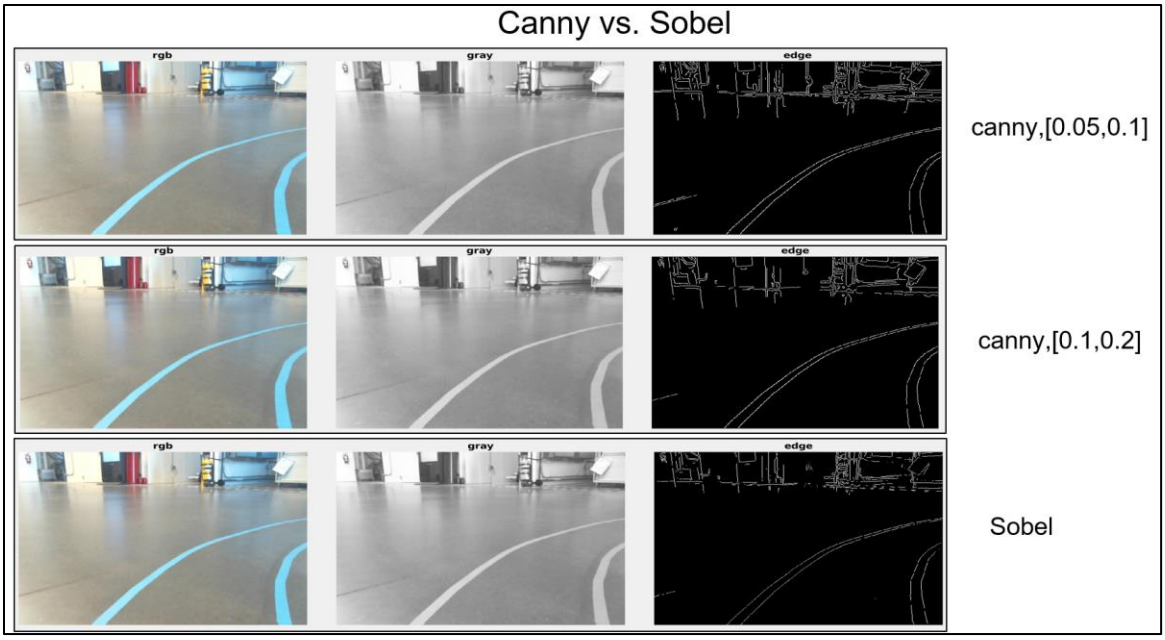

Figure 10: Canny vs. Sobel Edge detection 1 (No Glare)

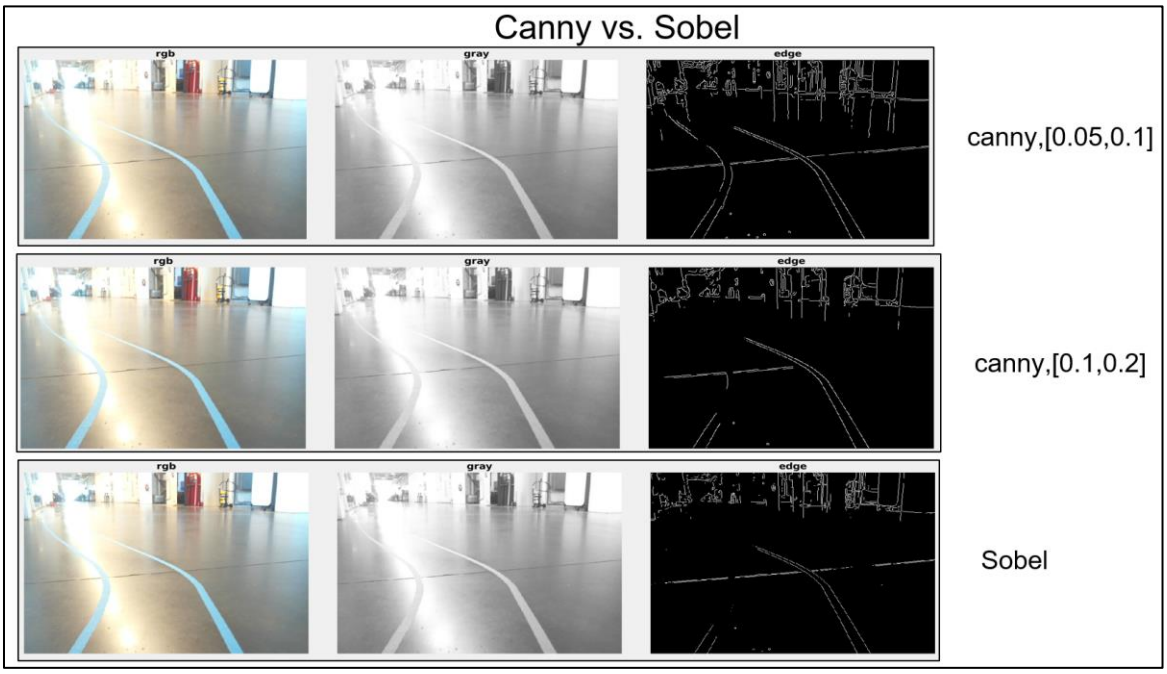

Figure 11: Canny vs. Sobel Edge detection 2 (with glare)

2) RGB Edge vs. Gray Scale:
Based on the analysis, it is observed that the RGB edge merge image, where red, green, and blue edges detected in individual images are merged provides better results than grayscale image edge detection (refer to Figure 12)

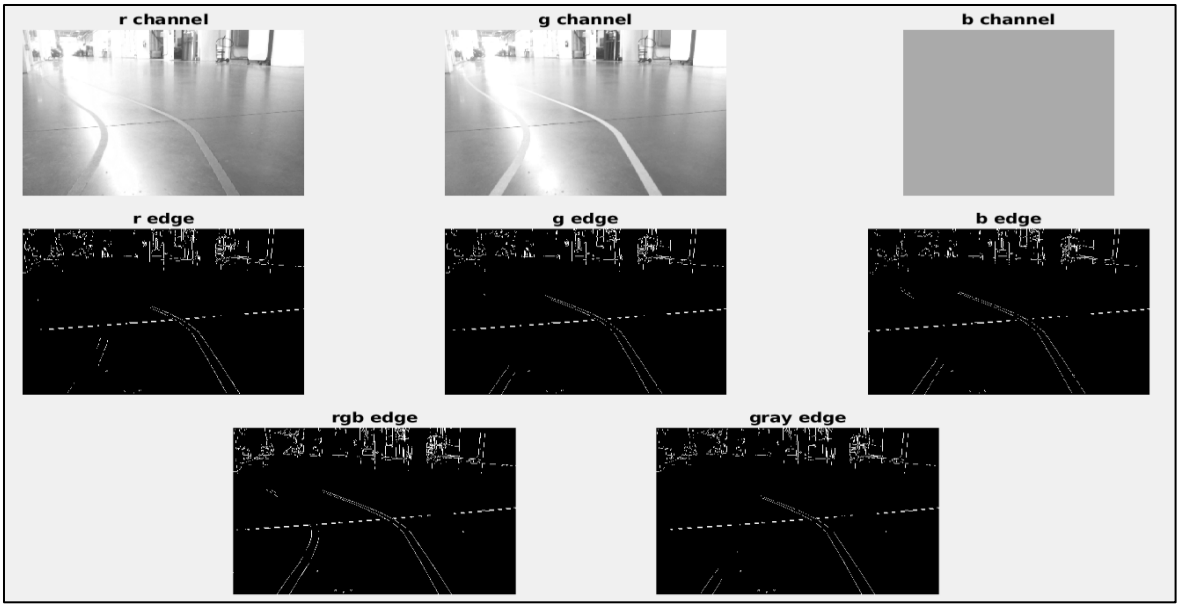

Figure 12: RGB Merged edge detection vs. Grayscale edge detection

3) Image Rescale:
To reduce the computation power and time taken to process the image, rescaling of the image for different image sizes was analyzed. An optimum solution to increase the computation speed without removing edge detection accuracy was observed when the image was rescaled to 160x120 size (refer to Figure 13).

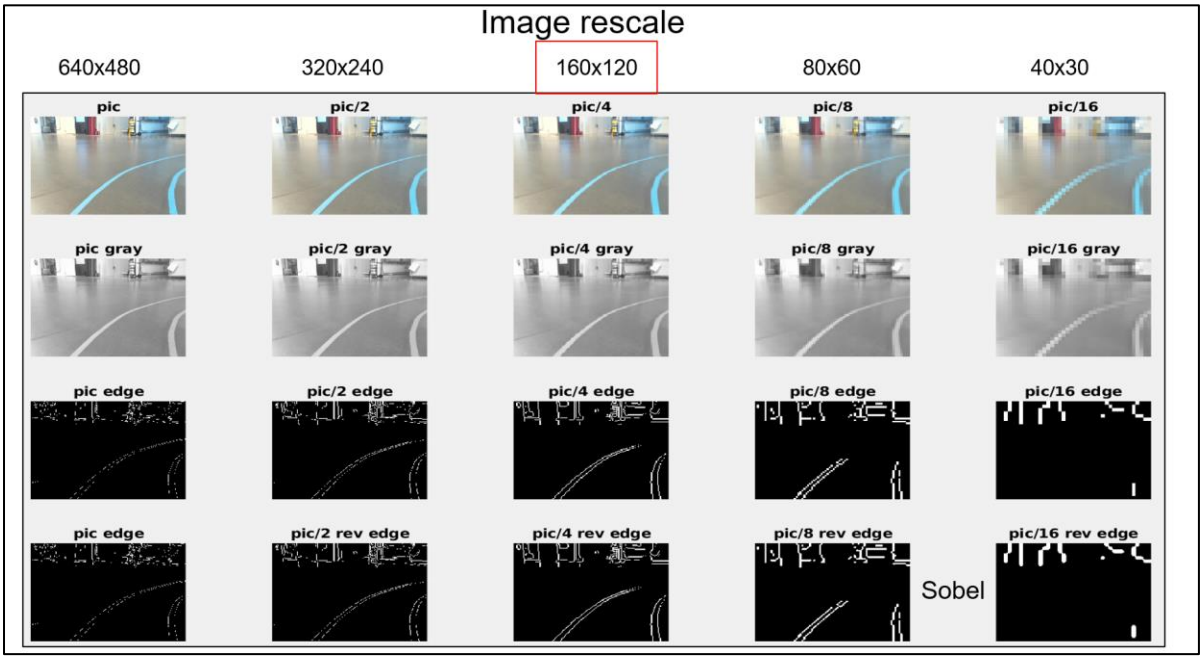

Figure 13: Edge detection with different image resizing

4) Denoise:
Denoise mean and median filters were applied to the image, and a significant difference was observed in lane edge detection. Some of the outliers in terms of perpendicular edges to the lane were removed with the help of a median filter later applied in the actual code and test (refer to Figure14)

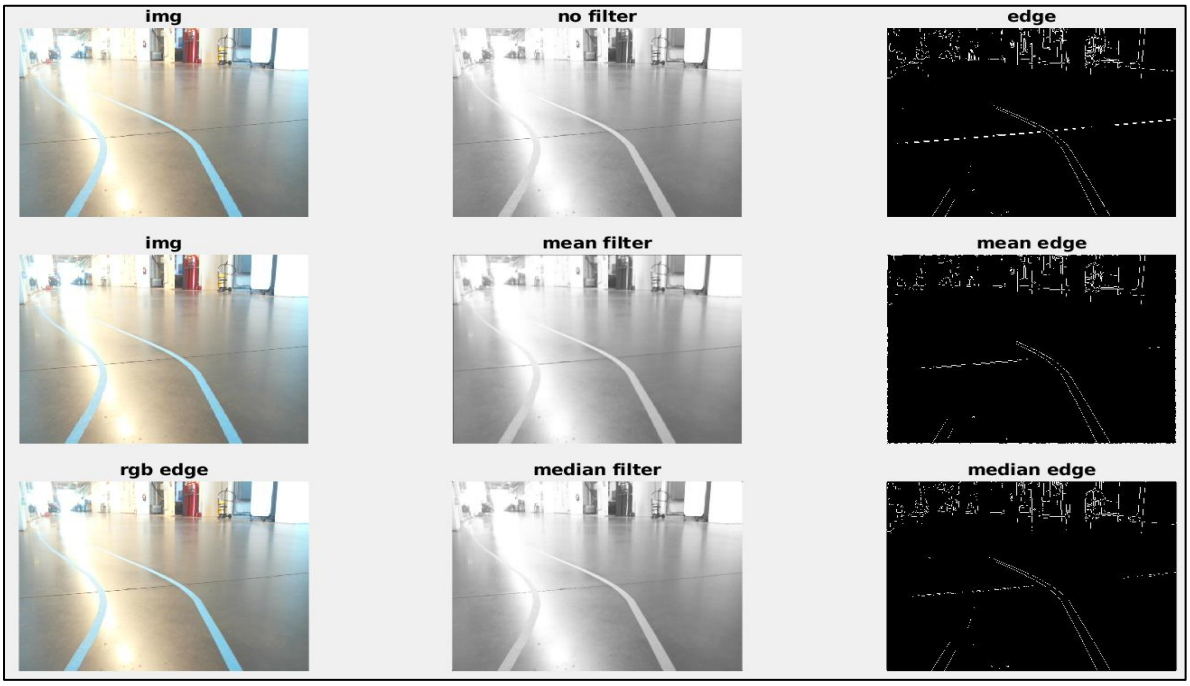

Figure 14: Denoise by median and mean filter

5) Gamma Correction:
Similar to the Denoise, there was no significant difference made by applying gamma correction for edge detection (refer to Figure15)

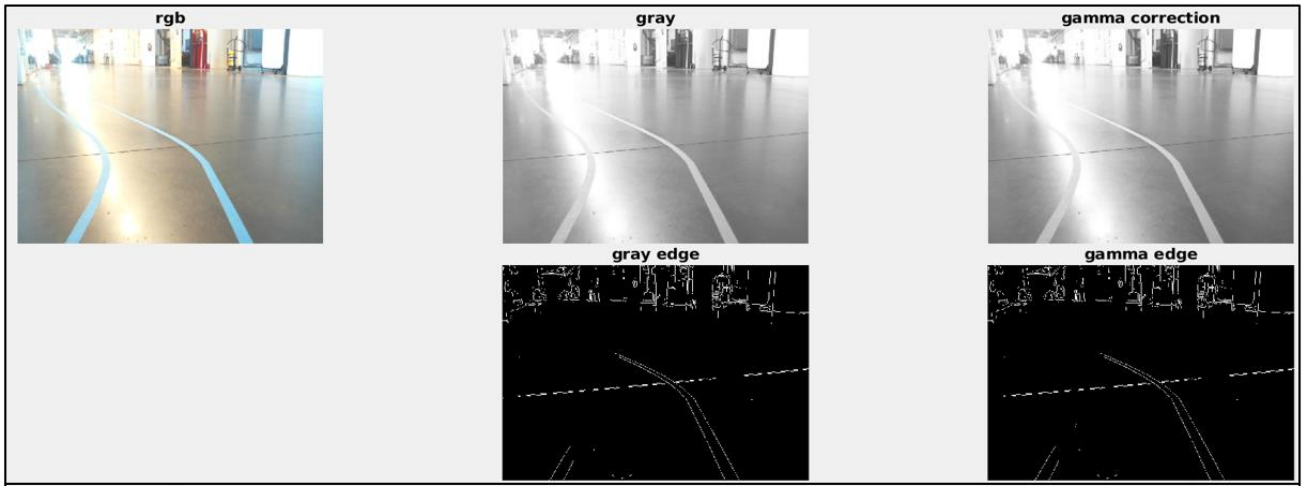

Figure 15: Edge detection after gamma correction

6) Histogram Equalization:
Contrast can be increased by using histogram equalization by spreading the most frequently occurring intensity values, thus increasing the intensity range. The histogram equalization, when applied to the image, did not show significant improvement in edge detection (refer to Figure 16)

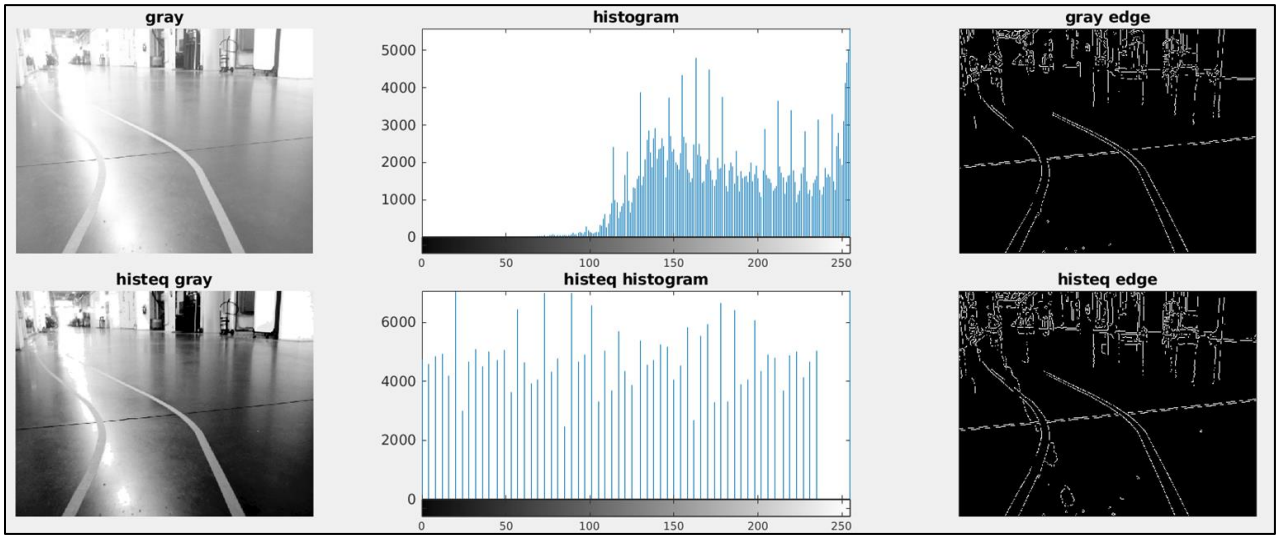

Figure 16: Edge detection after histogram equalization

Based on the analysis, the following methods for edge detection were considered, and some of them were disregarded.

Camera Position	Exposure time	Edge detection	Rgb merge	Image rescale	denoise	Gamma correction	Histogram equalization
Top-back	80	sobel	✓	160x120	Median filter	?	✗

Hough Transform:

Hough transform is a feature extraction technique to detect particular shapes. It is an efficient tool to recognize the lanes on the road in the form of straight lines, and many autonomous driving vehicles use this technology for lane extraction.

Based on the previous edge detection, Figure 17 represents the detection of Hough lines based on RGB merged image and Gray-scale image.

The region of interest (ROI) is selected for the Hough lines to determine the length of the Hough lines.

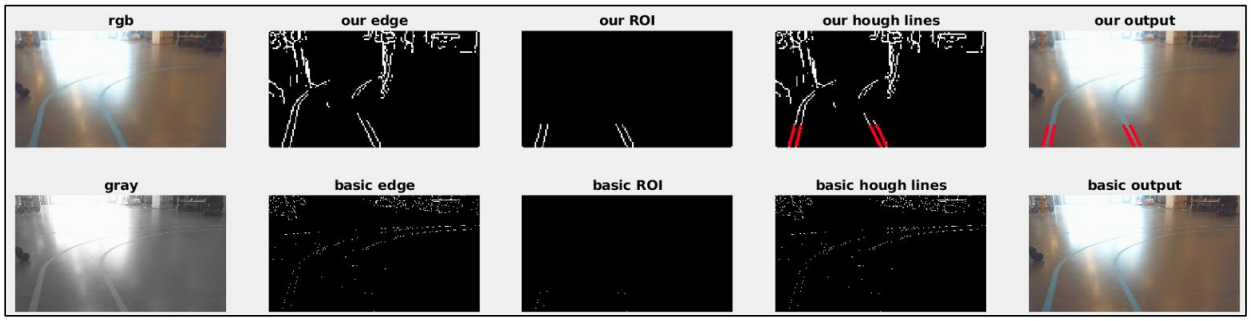

Figure 17: Hough line detection based on RGB edge and Grayscale edge

Further, for the merged case, two Hough lines formed at a lane are transformed into one Hough line, and a departure line (Yellow Colored) is plotted using the poly-fit function. The departure line is plotted considering the slope of two Hough lines on the left and the right side of the lane. Figure 18 shows a case in which if one lane is not visible, the edge of the image is considered a Hough line until the Lane edge appears. The edge is selected based on the slope of one available Hough line from Lane detection. This departure lane is further used for steering and velocity control commands.

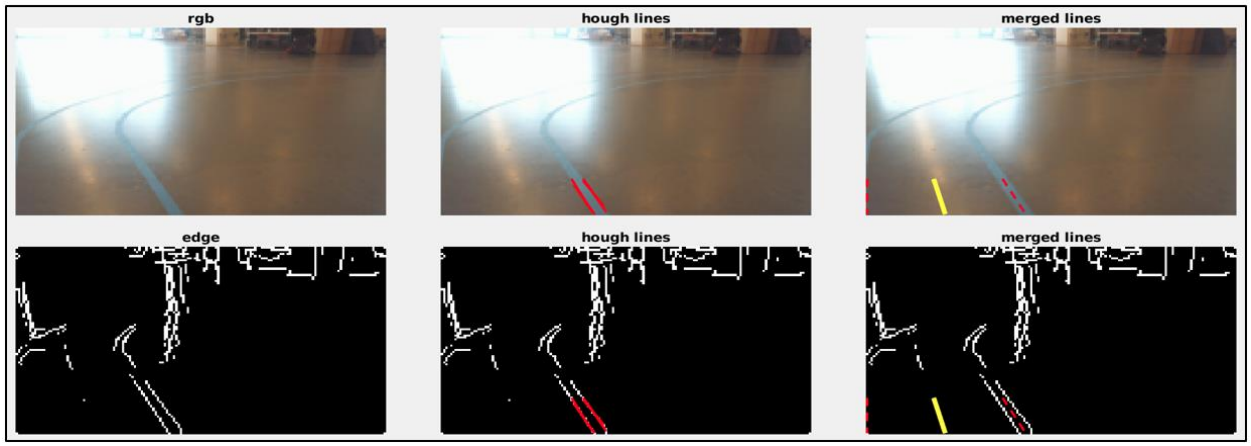

Figure 18: Hough line consideration with one lane missing from the camera frame

3.1.3 Vehicle Control (Autonomous Lane Keeping)

For Autonomous Lane Keeping, correct steering angles should be calculated for the RC car to stay in the lanes. The RC car available is an Ackerman steered car, and two main types of controllers can be applied for steering control, Pure Pursuit and Stanley Control. In this project, we chose to implement Stanley Control for Autonomous Lane-keeping.

Stanley Control

Stanley Controller used for steering controls calculates steering angle δ based on the departure angle θe and departure distance efa from the centerline.

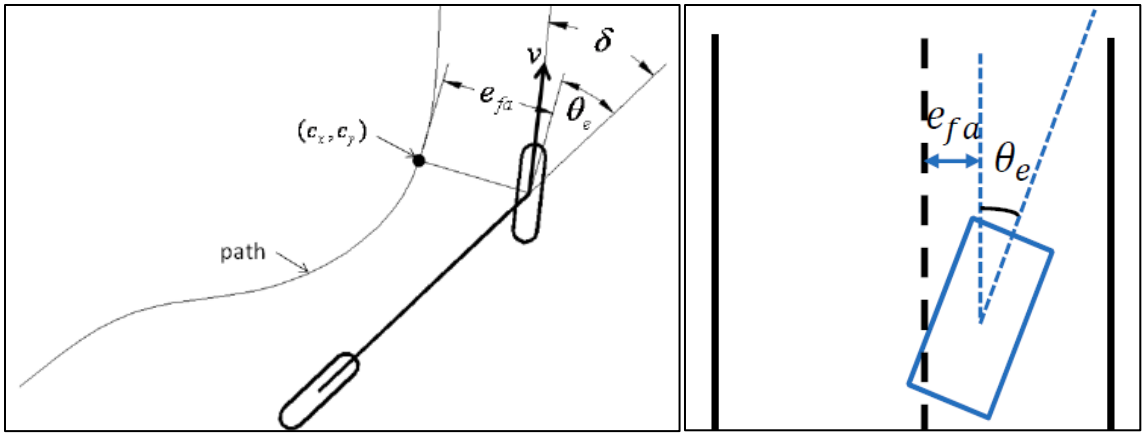

Figure 19: Stanley Controller

(b) Departure angle

(a) Departure distance

Steering angle: $\varphi(t) = \theta_e(t) + \tan^{-1}\left(\frac{ke_{fa}(t)}{v(t)}\right)$

where, *v(t)* is the driving speed.

Figure 20: Steering angle formula by using Stanley controller

A simplified version of the Stanley controller is used that considers departure angle and departure distance to calculate the steering angles (refer to Figure 21)

$$\varphi(t) = -k_1 \cdot \theta_e - k_2 \cdot e_{fa}$$

a)

```
kscale=0.5;
k1=0.70*kscale; k2=0.3*kscale;
angleScaleVal=1.6;
```

```
phi = k1*dirTheta + k2*distTheta;
phi = phi * angleScale;
phi = max(min(phi,45),-45);
```

b)

Figure 21: Simplified Stanley controller a) Equation b) Code equation and implementation.

K1 and K2 values are set based on the entire setup and iterations to achieve accurate autonomous lane-keeping.

Velocity Control

We also defined the velocity control based on the loop count concerning the Steering angles. The velocity control is set so that if the steering angle is more significant than a certain threshold, the vehicle will accelerate slowly by increasing the loop count to turn smoothly. Once the steering angle is less than 4 degrees, the loop count will reduce, and the vehicle will begin to move quickly (refer to Figure 22)

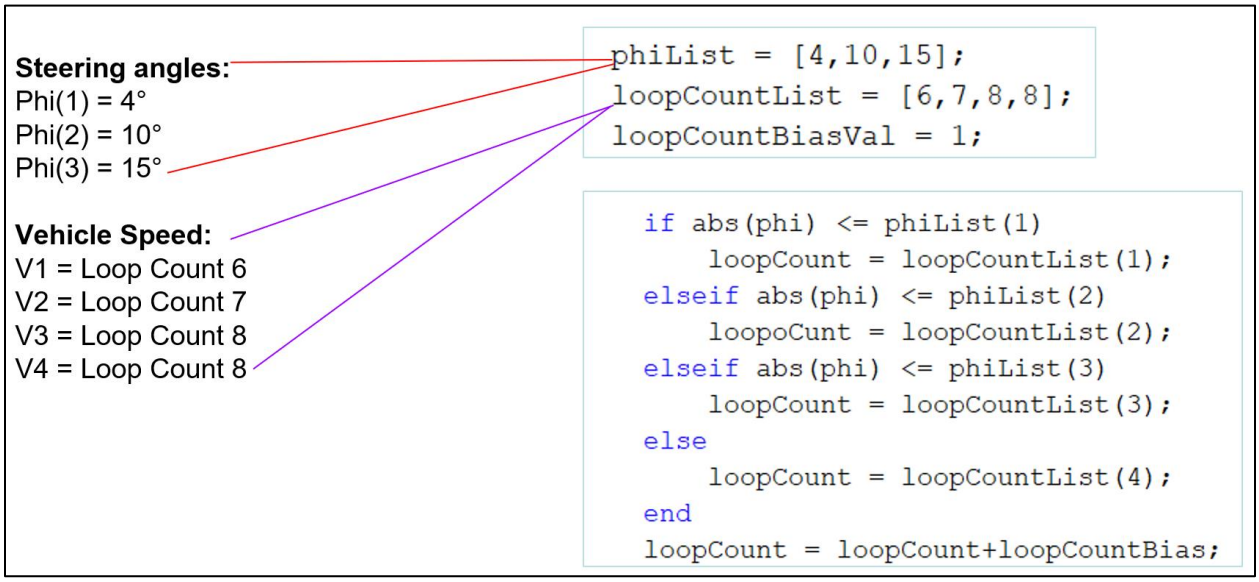

Figure 22: Velocity control concerning the steering angle

3.2 Test Results – Autonomous Lane Keeping

The demonstration of results obtained for autonomous lane keeping can be viewed in the link https://drive.google.com/file/d/12xWEvnSn9i8Q-DK_wOCP_l1uQGZn9pQF/view?usp=sharing.

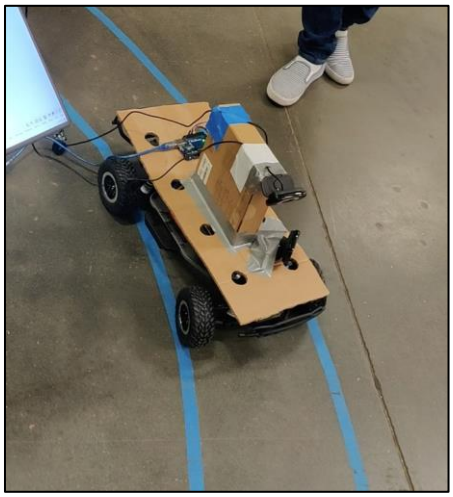

Figure 23: RC car demonstrating Autonomous Lane keeping

For Autonomous Navigation, from Figure 24, the steering angles can be determined as an input to the controller ranging from 0 to 0.5 for left steering and 0.5 to 1 for right steering (0.5 being neutral). We can observe right after the school section track has multiple turns where the RC car has to turn right first, left for a short time, and then right turn for a longer time to get through a section of a track. This can be observed in Figure 24, where right after the school sign, the steering angle changes with respect to the following turn the RC vehicle has to maneuver.

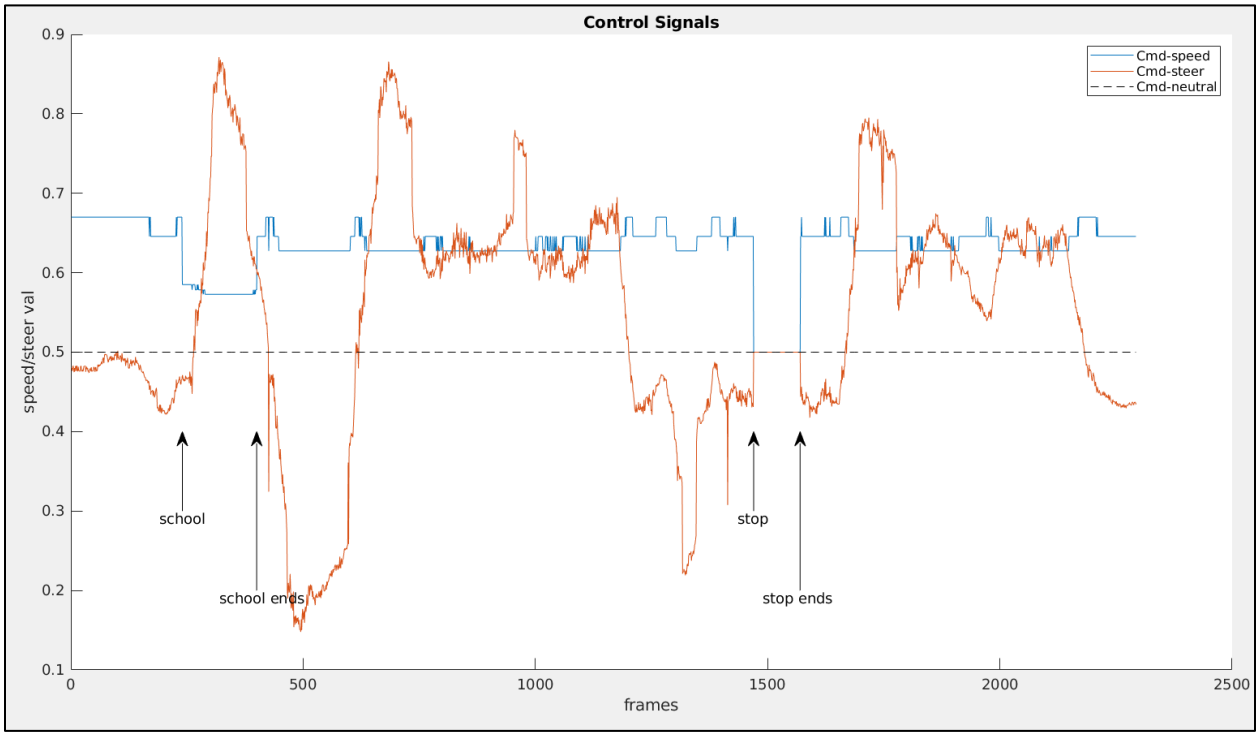

Figure 24: Test Results demonstrating steering angle and velocity inputs vs. frames captured around the testing track one lap

4. Traffic Sign Recognition and Vehicle Control

Traffic sign detection and control are necessary to prevent accidents on the road and as an aid to vehicle autonomy. Modern cars have loads of equipment, including an infotainment system by which a driver might get distracted, leading to an accidental situation. Traffic sign recognition and vehicle control also act as ADAS in a vehicle; in a scenario, if the driver misses a traffic sign, ADAS can work by recognizing a traffic sign and react accordingly to prevent the accident.

On-board cameras are used to detect the traffic signs, and Deep Learning algorithms aid in recognizing the type of traffic sign and sending control commands for vehicle navigation.

4.1 Methodology

4.1.1 Traffic sign Recognition

For the project, two main approaches were considered for traffic sign recognition:

1) Traditional Machine Learning by using Cascade Object Detector
2) Deep Learning approach using Convolutional Neural Networks

We tried implementing Cascade Object Detector but faced several issues. Later we considered CIFAR 10 Deep Learning CNN approach, which was more accurate and quicker.

Cascade Object Detector:

Cascade classifier consists of stages consisting of weak learners. At each stage, the classifier is labeled as a region positive or negative based on if the object is detected or not.

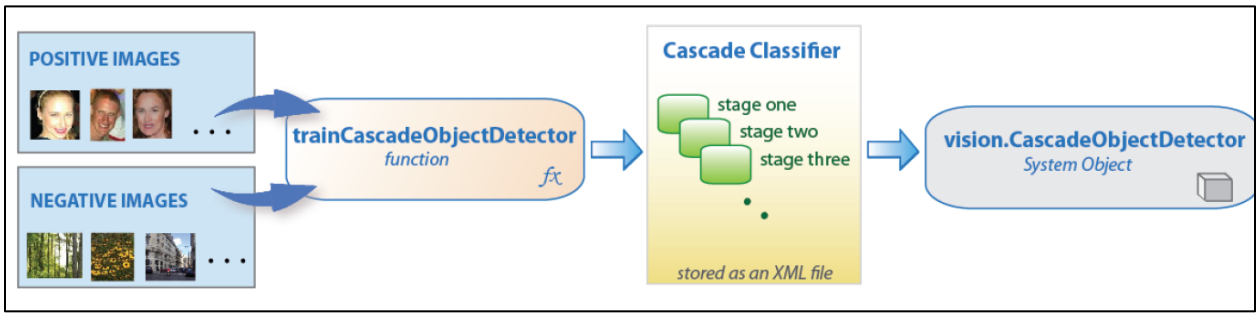

Figure 25: Cascade Object Detector Flowchart

The Cascade classifier consists of the following stages

1) True Positive Case – where the object is detected correctly
2) False Positive Case – where the object is detected but incorrectly or without an object
3) False Negative Case – where a false sample is detected as positive

As long as the classification is not False Negative, the False-positive cases can be removed stage after stage. Cascade classifier works on various methods based on feature-based extraction for object recognition.

Deep Learning – Convolutional Neural Network (CNN) model:

A CNN takes input images and assigns weights and biases to each image for segregating from each other. CNN comprises node layers containing input layers, hidden or middle layers, and output layers.

CNN has two main steps

1) Feature detection using multiple convolutional layers
2) Classification using a fully connected neural network

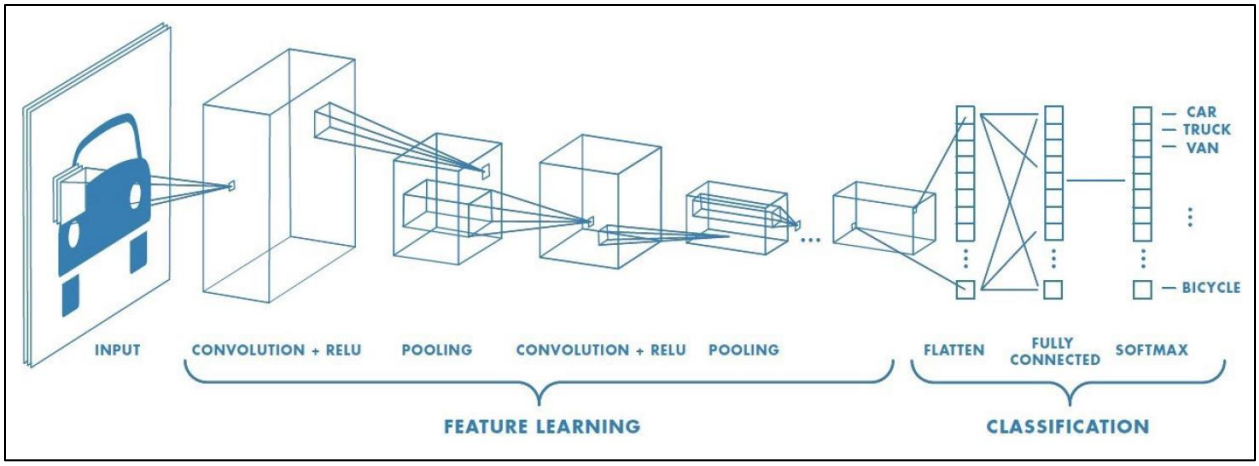

Figure 26: CNN classification

Currently, the implemented approach uses CIFAR10Net deep learning model. This model operates on CIFAR 10 image data sets and has two fully connected networks and two convolutional layers. Each layer is followed by a ReLU layer for feature-based extraction and a max-pooling layer of size 2x2.

Creating Dataset for training

MATLAB Image Labeler toolbox generates images with labels for training the CNN model. Three classes were created, a stop sign, a school sign, and a background class, and labeled on the images. Background class was input as negatives to consider if stop and school signs are not present.

Figure 27: Labelling the image for stop sign detection

Data Augmentation and training

The dataset created used several images in different orientations. This was solely done to improve the detection accuracy of the CNN model and as experience from previously trained models.

Some of the dataset examples are mentioned in Figure:

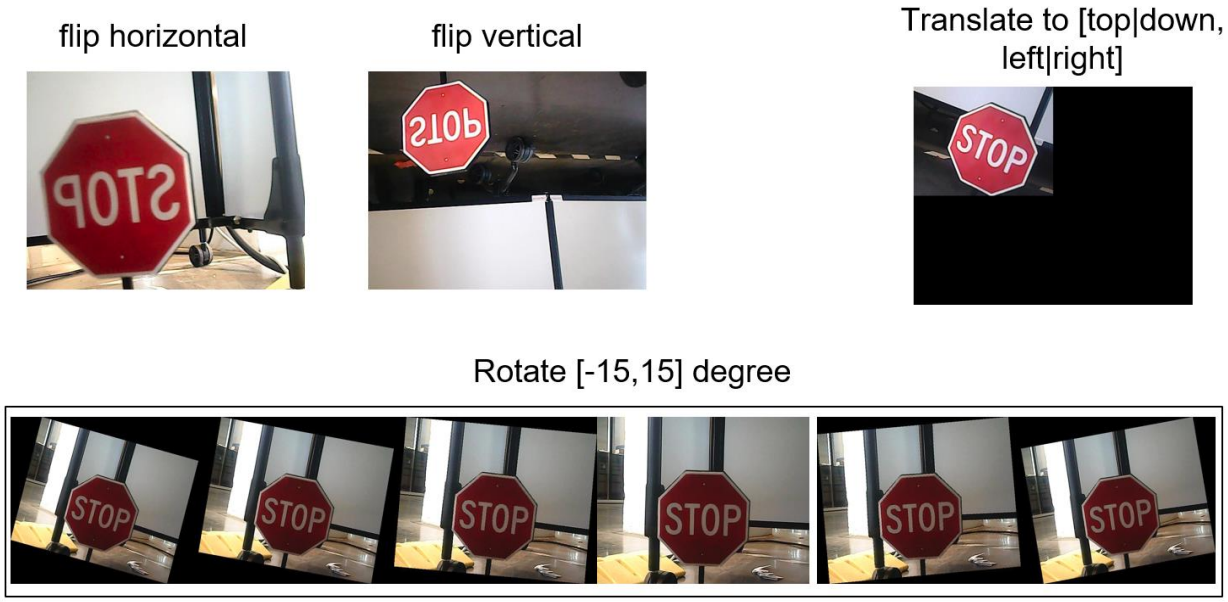

Figure 28: Example dataset images for training

4.2 Traffic sign recognition results

1) Cascade Object Detector
   The results obtained by cascade object detector for a stop sign and school sign detection were inaccurate, and cascade also recognized few false-positive recognition results.

Cascade Detector - Stop Sign

1. Stop: 0.3, 10, accuracy = 72%

neg: 943, pos: 311, time: 0.049687, FPS: 20

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	191	231
	Negative	120	712

2. Stop: 0.1, 10, accuracy = 81.89%

neg: 943, pos: 311, time: 0.049569, FPS: 20

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	176	92
	Negative	135	851

3. Stop: 0.3, 20, stop at stage 12, accuracy = 81.89%

neg: 943, pos: 311, time: 0.048755, FPS: 20

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	176	92
	Negative	135	851

4. Stop-0.1, 20, stop at stage 7, accuracy = 80.54

neg: 943, pos: 311, time: 0.049497, FPS: 20

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	175	108
	Negative	136	835

Cascade Detector - School Sign

1. School-0.3-10, accuracy = 7.33%

neg: 358, pos: 896, time: 0.054726, FPS: 18

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	93	269
	Negative	803	89

1. School-0.1-10, accuracy =

neg: 358, pos: 896, time: 0.055506, FPS: 18

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	114	224
	Negative	782	134

3. School-0.3-20, stop at stage 9, accuracy =
neg: 358, pos: 896, time: 0.055255, FPS: 18

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	78	221
	Negative	818	137

4. School-0.1-20, stop at stage 5
neg: 358, pos: 896, time: 0.055072, FPS: 18

	Confusion Matrix		
		Actual	
		Positive	Negative
Predicted	Positive	97	225
	Negative	799	133

Figure 29: Accuracy and confusion matrix for Cascade model

2) CIFAR10Net

A more effective and accurate recognition was found by using Deep Learning-based CIDAR10Net CNN. The stop sign and School Sign recognition accuracy were about 99.98% accurate, and it was pretty quick compared to other CNN-based models.

Figure 30 shows the comparison between different network models used and the time taken to run each model. Although CIFAR 10 shows many false-positive results, the detection is effective and accurate if the accuracy threshold is set high.

CNN models	Layer #	performance	Time per second [s]	FPS
cifar10	15	Many false positive	0.15	6.7
alexnet	25	Many false positive	0.6	1.7
resnet18	71	accurate	0.68	1.47
googlenet	144	accurate	1.2	0.83

Figure 30: Rate performance comparison of different CNN models

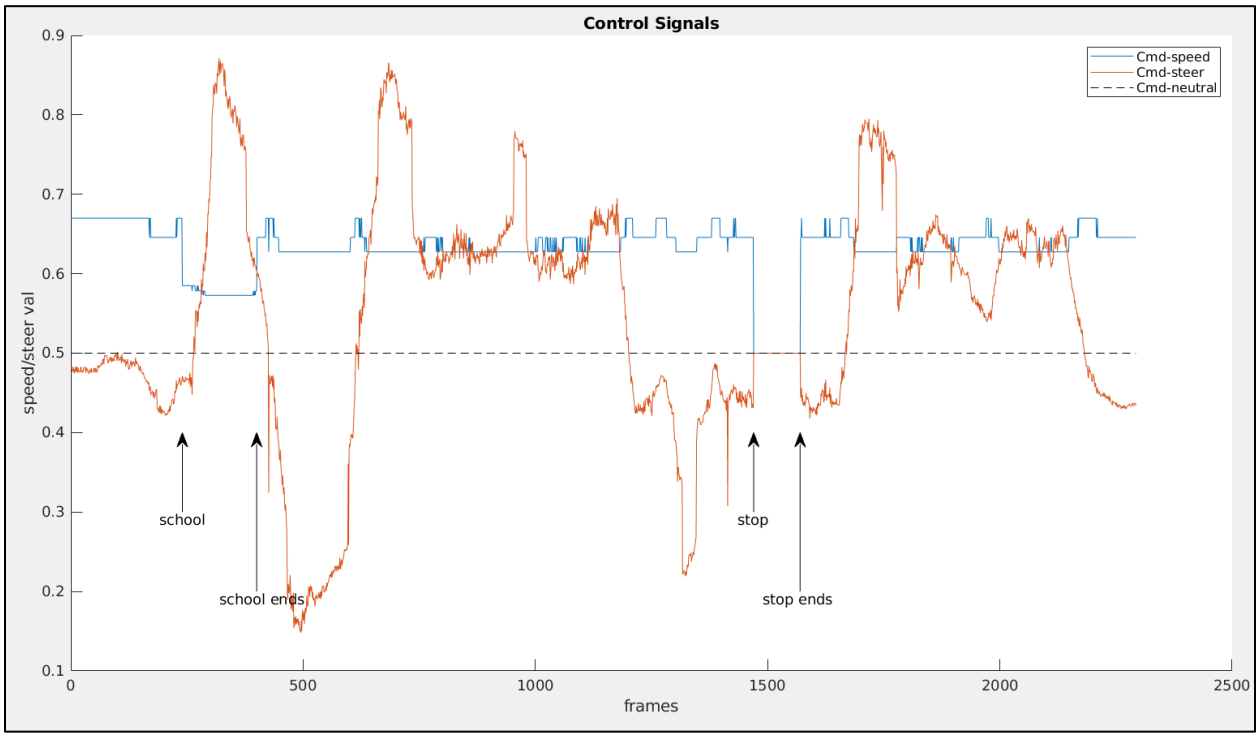

Figure 31: Test Results demonstrating stop sign and school sign recognition and control by the testing on track for one lap

Results on an actual RC car can be derived from Figure 31; it can be observed that at the school sign zone, based on the commands or velocity input, the loop counter has increased, and the velocity has gradually decreased. Similarly, at the stop sign detection zone, the vehicle stopped for a few seconds with 0.5 or neutral velocity inputs and then proceeded. This indicates that the physical implementation of traffic sign recognition is successful.

4.3 Vehicle to Vehicle (V2V) communication:

Connected mobility is an essential factor responsible for successfully implementing autonomous driving in the world. Connected mobility includes V2V vehicle to vehicle and V2I Vehicle to infrastructure communication that sends and receives valuable information for autonomous navigation of a vehicle. Moreover, the connected car can also reduce the number of vehicles on the road, regulate the traffic and effectively implement shared mobility.

V2V communication in this project is attempted by connecting two computers. One computer controls the vehicle by an autonomous lane-keeping algorithm running on it. Another computer recognizes the traffic signs and gives commands to the main computer if detected for vehicle control accordingly.

Communication between two computers is established by using the User Datagram Protocol UDP.

As mentioned in Figure 32, Once stop and school signs are detected, appropriate commands will be given for vehicle control to slow down or stop.

```
loopCountBias=0;
speedScale=1;
% for detection of school sign and stop sign
if udpB.BytesAvailable > 0
    msg = fscanf(udpB);
      fprintf('%s',msg);
    signs=split(msg);sign=signs{1,1};
    if strcmp(sign,'StopSign') && ~StopCount
        StopCount=StopLoop;
        speedScale=0;
        disp('1st receive StopSign');
    elseif strcmp(sign, 'SchoolSign') && ~SchoolCunt
        StopCount=StopLoop;
        loopCountBias=loopCountBiasVal;
        disp('1st receive SchoolSign');
    end
    flushinput(udpB);
end

if SchoolCount
    SchoolCount = SchoolCount-1;
    loopCountBias = loopCountBiasVal;
end
if StopCount
    StopCount = StopCount-1;
    speedScale=0;
end
```

Figure 32: Traffic sign vehicle control in MATLAB Script

5. Challenges and Achievement Discussions

Most of the challenges incurred in the project were solved in the building stages of the project. Some of the challenges are mentioned below.

1) Bright glare interfering with edge detection

One major issue encountered during edge detection was the bright light around the school sign area that hindered the edge detection. This problem was resolved by iterating between different exposure values and setting up an optimal exposure value that gave the best results for lane extraction all over the track (refer to Figure 33)

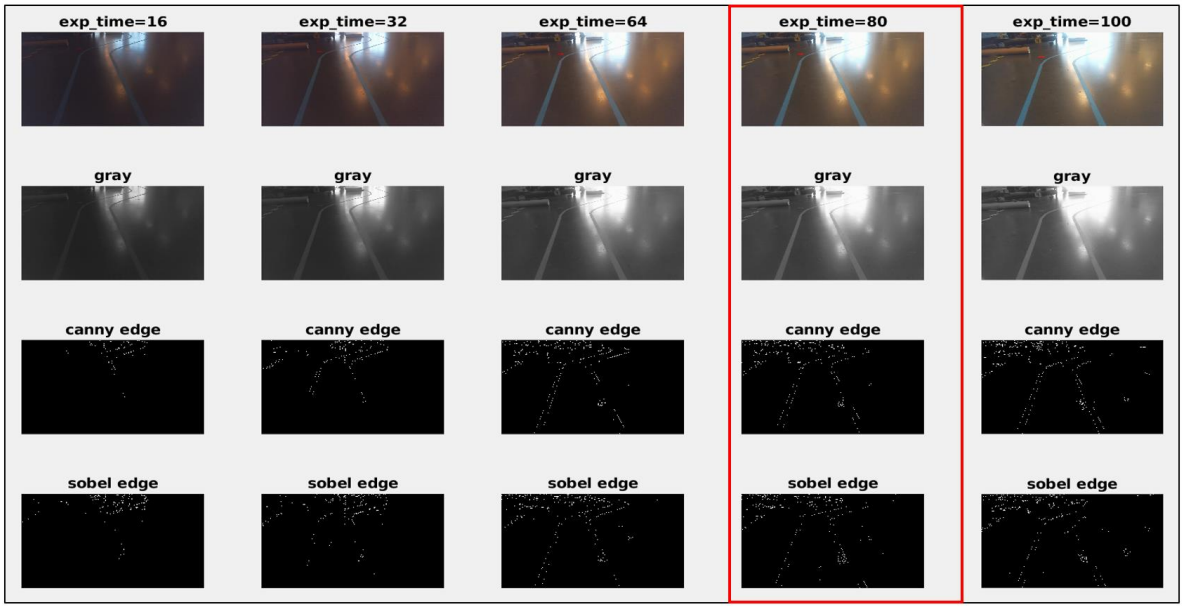

Figure 33: Edge detection concerning the exposure time

2) Corner cases while edge detection

While doing line extraction, even the edges are detected, and the region of interest is selected for Hough lines; there was some noise present due to which corner cases were detected. Referring to Figure 34, it can be observed that unwanted edges were detected. This issue was resolved by setting up threshold values for minimum and maximum distances between the lines (Lane width), theta angle or level of parallelism between the two lines, and lane color consistency. After implementation, the edges on the lane were ideally detected.

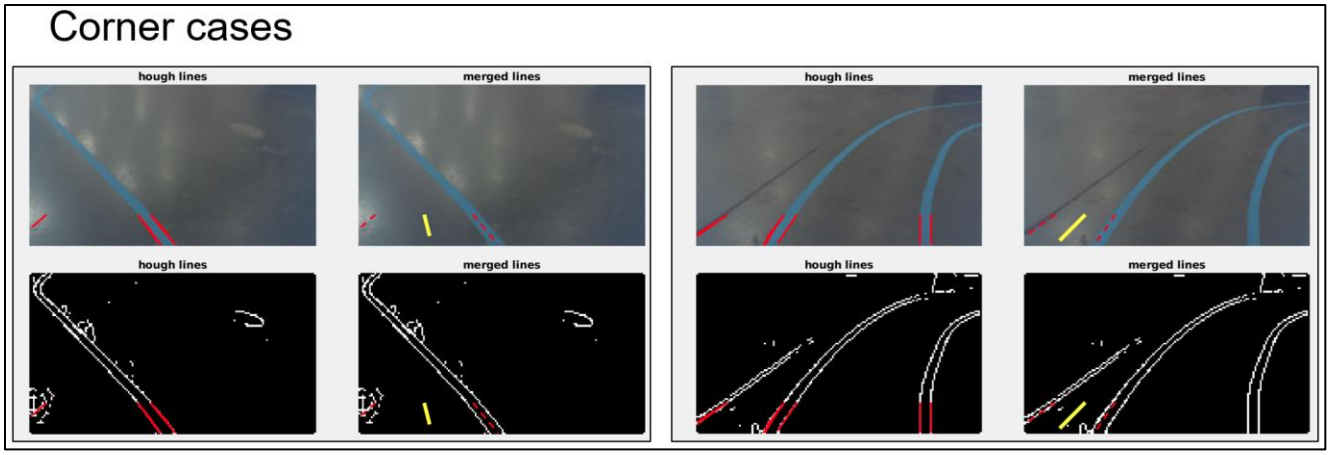

Figure 34: Corner cases for edge detections

3) Camera hardware issues

A significant issue faced during the final demonstration day, testing and recording the video, was a camera hardware issue. The cable connected to the camera gave abrupt readings by freezing at times. It was noticeable that slight movement in the camera wire leads to freezing of the frame, and the vehicle at that time either stops or accelerates based on the velocity command for that particular frame. We were unable to resolve this issue but were able to record a video of one successful lap of the course later.

Figure 35: Camera Hardware

4) Traffic Sign recognition issues

A Major challenge was the inconsistent behavior of the models used for traffic sign recognition. Cascade object detector required a large dataset to train the stop sign and school sign, and although it was fast, it was inaccurate.

Resnet18 provided an accurate result but was slow and could not be implemented in the dynamic running motion of the RC car as it failed to detect signs.

CIFAR10Net was quick but gave false-positive results. This model was used for the final test. The issue was resolved by using a high accuracy threshold to eliminate false-positive sign detection.

6. Future Scope or alternate implementation:

1) Implementation of shape-based traffic sign recognition

Instead of the stop sign and school sign detection, another approach would be to detect the traffic sign based on shape detection. A code can be written to detect octagonal-shaped stop signs of a particular dimension and triangular for school sign

2) Implementing Pure pursuit controller or Blob tracking and PID controller

Stanley controller used can be replaced by implementing pure pursuit control or PID controller based on blob tracking to generate the differences between these three vehicle controllers. Statistical analysis can be developed based on the live feed data used to run three different types of controllers.

References:

1) *https://www.ibm.com/cloud/learn/convolutional-neural-networks*
2) *https://en.wikibooks.org/wiki/Communication_Networks/TCP_and_UDP_Protocols*
3) *https://en.wikipedia.org/wiki/Canny_edge_detector*
4) *https://www.mathworks.com/products/computer-vision.html*
5) *https://www.mathworks.com/hardware-support/arduino-matlab.html*
6) *Lecture Slides AuE 8240 & Project Presentation slides*
7) *https://www.bloomberg.com/news/articles/2017-02-10/how-autonomous-vehicles-recognize-traffic-signs*
8) *A. Sivasangari, S. Nivetha, Pavithra, P. Ajitha and R. M. Gomathi, "Indian Traffic Sign Board Recognition and Driver Alert System Using CNN,"* 2020 4th International Conference on Computer, Communication and Signal Processing (ICCCSP)*, 2020, pp. 1-4, doi: 10.1109/ICCCSP49186.2020.9315260.*
9) *https://www.mathworks.com/help/vision/ug/train-acf-based-stop-sign-detector.html*