B. TECH. PROJECT REPORT

On

Developing an OEDR Module for Autonomous Vehicle Using Simulators

BY

T. Venkat Nikhil (160001058)

DISCIPLINE OF COMPUTER SCIENCE AND ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

December 2019

Developing an OEDR Module for Autonomous Vehicle Using Simulators

A PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degrees

of
BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE AND ENGINEERING

Submitted by:
T. Venkat Nikhil (160001058)
Discipline of Computer Science and Engineering
Indian Institute of Technology Indore

Guided by:
Dr. Gourinath Banda
Associate Professor
Discipline of Computer Science and Engineering
Indian Institute of Technology Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
December 2019

<u>CANDIDATE’S DECLARATION</u>

We hereby declare that the project entitled **“Developing an OEDR Module for Autonomous Vehicle Using Simulators”** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in ‘Computer Science and Engineering’ completed under the supervision of **Dr. Gourinath Banda, Associate Professor, Discipline of Computer Science and Engineering,** IIT Indore is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

T. Venkat Nikhil

<u>CERTIFICATE by BTP Guide(s)</u>

It is certified that the above statement made by the students is correct to the best of my knowledge.

Dr. Gourinath Banda

Associate Professor

Discipline of Computer Science and Engineering

Indian Institute of Technology Indore

Preface

This report on “Developing an OEDR Module for Autonomous Vehicle Using Simulators " is prepared under the guidance of Dr. Gourinath Banda.

Through this report we have tried to give a detailed design of an object and event detection and recognition utility for autonomous vehicle using simulators and try to cover every aspect of the new design, if the design is technically and economically sound and feasible.

We have tried to the best of our abilities and knowledge to explain the content in a lucid manner. We have also added figures and statistics to make it more illustrative.

T. Venkat Nikhil
160001058
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Acknowledgements

I want to express my gratitude to Dr. Gourinath Banda, Associate Professor, Indian Institute of Technology Indore for his kind support and valuable guidance. I offer my sincere thanks to IPG Automotive for providing us test license of CarMaker and their support. Also, I am grateful to my project partner, Mr. Souvik Mandal, Computer Science and Engineering, IIT Indore for his support throughout the project.

I am obliged to the institute for the opportunity to be exposed to systematic research especially Dr. Gourinath Banda's Lab for providing the necessary hardware utilities to complete the project.

Lastly, I offer my sincere thanks to everyone who helped me complete this project, whose name I might I have forgotten to mention.

It is their help and support, due to which we became able to complete the design and technical report. Without their support this report would not have been possible.

T. Venkat Nikhil
160001058
B.Tech. IV Year
Discipline of Computer Science and Engineering
IIT Indore

Abstract

Over 80% of car crashes in the US are caused by driver error. There would be less user errors and fewer mistakes on the roads if all vehicles became driverless. Autonomous vehicles may cut travel time by up to 40 percent, recover up to 80 billion hours lost to commuting and congestion, and reduce fuel consumption by up to 40 percent.

Simulators provide controlled environment and numerous environment perceiving models. The sensors and chips for autonomous cars are astronomically expensive. This is way beyond the price scope of 99% of drivers.

Self-driving simulations have advantages in mileage data collection efficiency, road condition dataset diversity, and sensor corresponding data accuracy. We have worked on developing the Object and Event Detection and Recognition (OEDR) module development using the IPG Carmaker simulation platform. During the OEDR's recognition phase, we are using the machine learning and computer vision concepts to provide supervised approach to the solution development. Our work makes use of the YOLOv3 model and several other custom built models. We have shown that with the help of simulator we can develop economical and efficient state of the art modules for self-driving cars. We have laid the foundation to enable research and development into Autonomous Cars at the university level without requiring the expensive true scale automotive/cars.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to autonomous vehicle

Autonomy means to function independently. Autonomous cars require a sensor module which is responsible for sensing objects and events ongoing in its environment. In this dissertation, we are proposing a simulator-based Object and Event Detection and Recognition(OEDR) module development using an advanced simulator from IPG[17]. This work contributes to the autonomous car development efforts.

The concept of autonomous car was first exhibited at the 1939 New York World's Fair. The exhibit was created by General Motors to display its vision of how the automated highway system of self-driving cars would be. At present, due to safety features like braking systems and assisted parking, and a few with autonomous driving, steering, brake, and parking capabilities, many vehicles on the road are considered to be semi-autonomous. Autonomous vehicle technology relies on GPS capabilities as well as advanced sensing systems that can detect lane boundaries, signs and signals, and unexpected obstacles. An autonomous vehicle offers:

- Improved safety: Around 94% of the accidents on a road are caused by human error and with the help of autonomous vehicles 90% of these accidents can be avoided. [2]

- Increased efficiency: On an average, in developed countries, people spend around 26 minutes in a car to commute to the work place [3]. Autonomous cars can let people use this time efficiently for other purposes.

- Greater mobility options: Autonomous vehicles can be used in public transport, vehicles for disabled and they can overcome congestion on roads.

According to the Society of Automotive Engineers(SAE) classification[1], autonomous vehicles are classified into 6 Levels: Level 0 to Level 5.

- Level 0 (No Driving Automation):
  Vehicles are manually controlled by the driver.

- Level 1 (Driver Assistance):
  It is the lowest level of autonomy. These vehicles include a single autonomous system for driver assistance, such as lateral or longitudinal control.

- Level 2 (Partial Driving Automation):
  This consists of advanced driver assistance systems or ADAS, which includes both lateral or longitudinal control to stay clear of other vehicles.

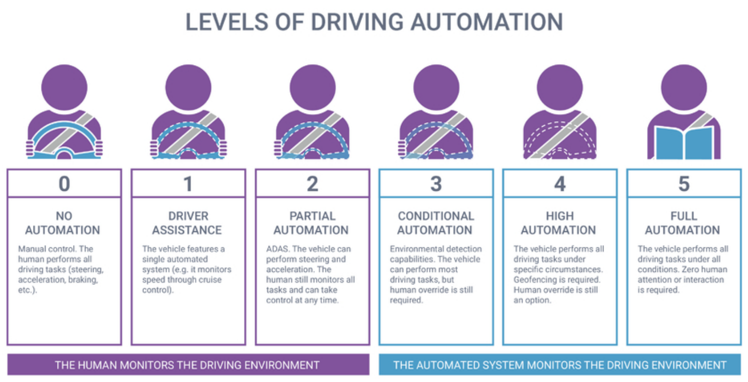

LEVELS OF DRIVING AUTOMATION
0
NO AUTOMATION
Manual control. The human performs all driving tasks (steering, acceleration, braking, etc.).
1
DRIVER ASSISTANCE
The vehicle features a single automated system (e.g. it monitors speed through cruise control).
2
PARTIAL AUTOMATION
ADAS. The vehicle can perform steering and acceleration. The human still monitors all tasks and can take control at any time.
3
CONDITIONAL AUTOMATION
Environmental detection capabilities. The vehicle can perform most driving tasks, but human override is still required.
4
HIGH AUTOMATION
The vehicle performs all driving tasks under specific circumstances. Geofencing is required. Human override is still an option.
5
FULL AUTOMATION
The vehicle performs all driving tasks under all conditions. Zero human attention or interaction is required.
THE HUMAN MONITORS THE DRIVING ENVIRONMENT
THE AUTOMATED SYSTEM MONITORS THE DRIVING ENVIRONMENT

Figure 1.1: Levels of Automation

- Level 3 (Conditional Driving Automation):

  The technological capabilities of an autonomous vehicle take a huge jump from level 2 to level 3. "Environment detection" capabilities are introduced in level 3 vehicles and they can make informed decisions for themselves. But they still require human override. The driver must remain alert and ready to take control if the system is unable to execute the task.

- Level 4 (High Driving Automation):

  The main difference between Level 3 and Level 4 is that Level 4 vehicles can intervene if things go wrong. In this sense, these cars do not require human interaction in most circumstances. However, a human still has the option to manual override.

- Level 5 (Full Driving Automation):

  These vehicles do not require human intervention and won't have steering wheel or acceleration/brake pedals. They can do what ever an experienced human driver would do and are not limited to the geofencing.

1.1.1 Generic block diagram of autonomous car systems

The following are the main components of an autonomous car system:

- Environment Mapping:

  It involves localisation of the vehicle with the help of an external aid (such as GPS) and also the information extracted from the environment perception module.

- Environment Perception:

  The OEDR module falls under this category. It is responsible for the decision taken by other modules present in an autonomous vehicle. It includes both detection and recognition of different objects and events.

- Motion planning:

This module is responsible for decision making based on the information provided by the environment perception module.

- Controller:

  It is responsible for implementing the decisions taken by the motion planner by taking the necessary lateral and longitudinal control.

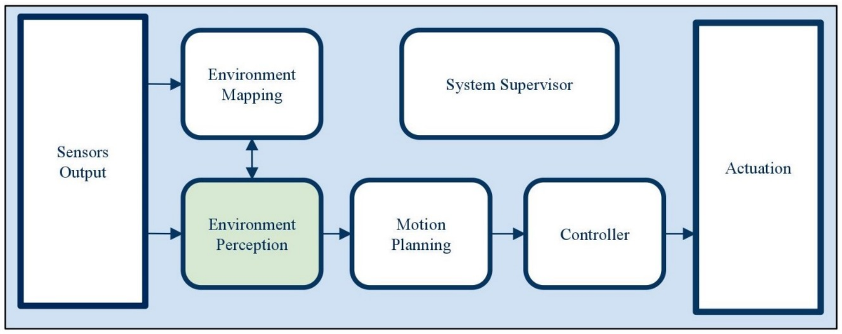


Figure 1.2: Generic Block Diagram of Autonomous Car Systems[4]

1.1.2 Importance of perception via OEDR

An autonomous vehicle should be able to both localize itself in an environment and identify and keep track of objects (both moving and stationary). The data can be accessed through sensors like camera, LIDAR and RADAR and also using other functionalities available like GPS. The information from these sensors can be used together and fused to localize the car and track objects in its environment, which will help it sustain the journey to its destination.

1.2 Contribution of this work

1.2.1 Developing the OEDR module of autonomous cars with state of the art simulators

According to a study, obtaining confidence in autonomous cars similar to that of human-level proficiency would require testing that takes about 400 years[5]. Considering the above fact and other key aspects of conventional autonomous driving such as cost of infrastructure and maintenance, the development of the autonomous vehicle will take a lot of time, effort and capital. By using simulators in the design and development of various modules of autonomous cars we can reduce this testing duration. In our project, we have employed the help of simulators to create a realistic imitation of the driving environment and developed an OEDR with stereo cameras.

- Achieving human level proficiency:
  Using simulators the time taken for the training of the autonomous vehicle can be reduced by employing the features of running a simulation. In a simulation the training speed can be greatly improved by running parallel instances of the same car and then collaborating the progress. In this way many of the cases a vehicle would encounter can be experienced by the simulated vehicle in a much faster and controlled environment.

- Controlled Environment:
  The environment of a simulator can be emulated accordingly to fit any scenario that one might encounter while driving as the simulator provides many options such as diverse environments and traffic conditions. The data retrieved by the vehicle in the simulator could be analysed in a very efficient way as the simulation can be rerun or paused at any time of the simulation, may it be while driving properly or during a crash scenario.

- Reduced development cost:

  The equipment and other infrastructure involved in the autonomous industry demands a substantial amount of capital which could set back the progress of the development. Whereas in simulators, only the computer in which simulation is conducted costs capital and other components such as vehicle, sensors can be avoided, thus reducing the development cost involved. Furthermore, the analysis results from simulation are very much valid in the field.

1.2.2 IPG CarMaker

The simulator with which we have completed our project is IPG CarMaker. CarMaker includes a complete model environment comprising an intelligent driver model, a detailed vehicle model and highly flexible models for roads and traffic. With the aid of this model environment, we can build complete and realistic test scenarios with ease, taking the test run off the road and directly to our computer. The event and maneuver-based testing method ensures that the necessary flexibility and realistic execution are available in the simulator, for example setting the environmental conditions like fog, sunlight glare, etc.

Chapter 2

OEDR and IPG CarMaker

2.1 Introduction to IPG CarMaker

IPG CarMaker can be used to accurately model real-world test scenarios, including the entire surrounding environment, in the virtual world. The powerful and capable models for vehicles, roads, drivers, and traffic make this possible.

- Open integration and test platform:

  CarMaker provides an open integration and test platform and can be used throughout the entire development process, from model to software to hardware to vehicle in the loop.

- Real time capable models:

  It provides many real time models capable such as intelligent driver model, a detailed vehicle model and highly flexible models for roads and traffic.

- Efficiency:

  CarMaker's performance guarantees flexibility, productivity and precision for all simulation tasks, thereby ensuring significant savings in cost and time for the development of your vehicle.

- Documentation:

  It provides extensive documentation which makes the process of navigating, accessing different models and testing much easier.

- Low hardware and Dependencies requirements:

  Aside from the low hardware requirements, the CarMaker software comes in with different modules which can installed only if required as there are less dependencies involved during a simulation.

2.2 Introduction to OEDR

OEDR module is a necessary component of the autonomous vehicle systems as it is responsible for environment perception. It is also known as the eye of the self-driving vehicle. It is responsible for the following features:

- Localisation:

  It helps place the vehicle at an appropriate position with respect to its environment.

- Safety:

  Many components of the autonomous vehicle such as collision detection, etc. are essential for ensuring not only the safety of the vehicle but also other drivers and pedestrians on the road.

- Lateral and Longitudinal control:

  The output of different models implemented in the OEDR module help the motion planning module to decide the necessary lateral and longitudinal variables needed for the movement of the vehicle.

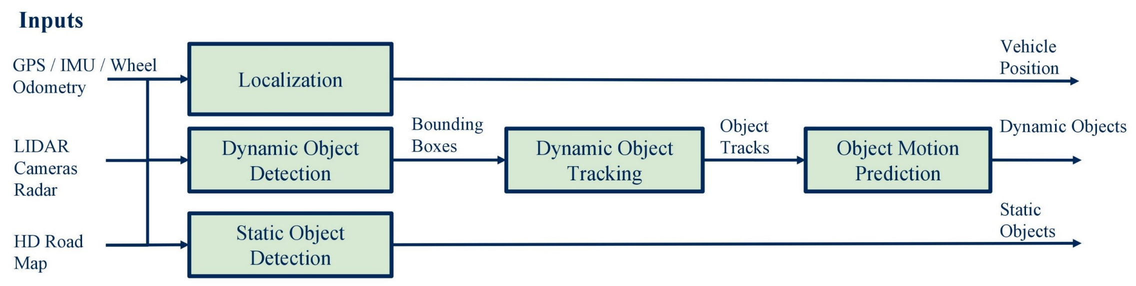


Figure 2.1: Components of OEDR Module[4]

The above features are achieved through help of the following components of the OEDR,

- Object Detection:

  It includes detection of obstacles and other objects(both static and dynamic) present in its vicinity.

- Object Tracking:

  Object tracking is an crucial task for the decision making process of the autonomous vehicle as it keeps track of all the objects which could help determine how these could affect the vehicle.

- Motion prediction:

  Using the information the object detection and tracking components the vehicle can now predict the motion of other objects to assess the decision that will be needed to be taken by the vehicle.

2.3 Developing OEDR in IPG CarMaker

Due to the availability of many sensors like camera, LIDAR and RADAR the development of the OEDR module is not restricted and they can be customized according to the necessities. The CarMaker also provides control over all the variables from climatic conditions to the friction of the tires which helps cover different real-life scenarios for productive development of the OEDR module.

Chapter 3

Sensor modules

3.1 Object Detection and Classification

Object detection involves detecting the instances of objects from a particular class in an image. The goal of object detection is to detect all instances of objects from a known class, such as people, cars, etc. The challenges involving object detection and classification are,

- Speed for real-time detection
- Multiple spatial scales and aspect ratios
- Limited data
- Class imbalance

A collection of instances is a dataset, in out object detection task we needed a dataset to train our deep learning model. Some of the features we were looking when we are searching for datasets are: the classes present in the dataset must be related to self-driving car related, the volume of the dataset is small to medium, and object annotation is present in the form of 2d annotation as we were interested in the creation of 2d boundary box. We considered Lyft[15], Waymo[16], and Rovit[11] dataset. Lyft dataset only contained 3d annotation of an object,

Object Detection Algorithm	**Mean Average Precision**	**Time(ms)**
Mask RCNN	36.2	172
SSD513	31.2	125
YOLOv3	33	51

Table 3.1: Object Detection Algorithm:Time-Accuracy Trade-off[10]

so we discarded it. Waymo dataset has a vast volume, and due to constrained computation power, we went with the Rovit dataset.

In our dissertation we have explored the following object detection models:

- Single Shot Detector(SSD) 513: A typical CNN network gradually shrinks the feature map size and increase the depth as it goes to the deeper layers. Taking advantage of the above feature, the shallow layers are used to predict small objects and deeper layers to predict big objects, as small objects don't need bigger receptive fields and bigger receptive fields can be confusing for small objects.[18]

- Mask RCNN: This approach efficiently detects objects in an image while simultaneously generating a high-quality segmentation mask for each instance. The method, called Mask R-CNN, extends Faster R-CNN by adding a branch for predicting an object mask in parallel with the existing branch for bounding box recognition[19].

- YOLOv3: YOLOv3 is a variant of a popular object detection algorithm YOLO – You Only Look Once[10].

Table 3.1 shows the comparision between the above mentioned models:

We can see that the Mask RCNN model offers a much better MAP of all but loses to others in the time taken for detection and the SSD model offers reduced time by trading off the MAP. However the YOLOv3 offers the best time for detection of the 3 available models while still maintaining a better MAP comparable to that of the others.

So, we have opted for YOLOv3, which uses a CNN based approach, for our object classification challenge.

3.1.1 CNN

In deep learning, a convolutional neural network (CNN, or ConvNet)[6] is a class of deep neural networks, most commonly applied to analyzing visual imagery. It has the following features:

- Local Receptive Fields

  With local receptive fields we can extract elementary visual features such as edges, corners, etc.

- Shared Weights

  Elementary feature detectors that are useful in one part of the image are likely to be across the entire image.

- Spatial/Temporal Sub-sampling

  Sub-sampling reduces the resolution to reduce the sensitivity to shifts and distortion

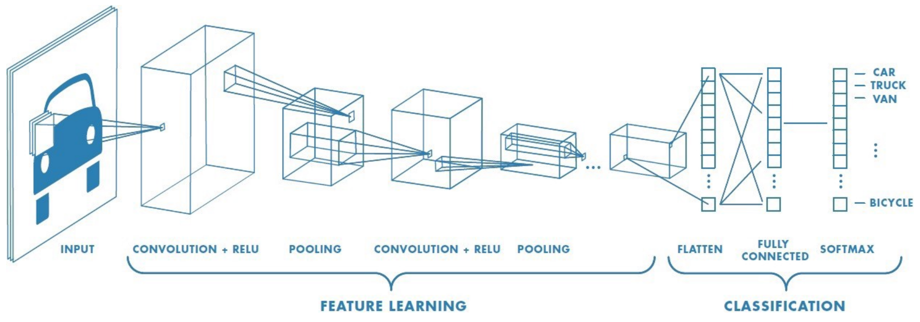


Figure 3.1: Convolutional Neural Network

3.1.2 YOLOv3

You only look once (YOLO) is an object detection system for real-time processing. This system is called this way because it traverses through the image only once to detect all the object instances. There are three versions of this system present as of now YOLO, YOLOv2, YOLOv3. For our implementation, we have chosen YOLOv3.

3.1.2.1 Understanding How YOLOv3 works

YOLOv3 is size invariant as it employs only the usage of convolutional layers making it a fully convoluted network. It consists of 75 convolutional layers, with skip connections and up-sampling layers. The lack of pooling layers in the model is supported as this helps in preventing loss of low-level features often attributed with pooling. The network down-samples the image by a factor called stride. Generally, a stride of any layer in the network is equal to the factor by which the output of the layer is smaller than the input image to the network.

The features learned by the convolutional layers are passed onto a classifier/regressor which makes the detection prediction. As YOLOv3 uses 1x1 convolutions during prediction, the size of the prediction map is exactly the size of the feature map before it. YOLOv3 contains (B x (5 + C)) entries in the feature map depth-wise. Where B is the number of bounding boxes each cell can predict. Each of the bounding boxes has 5 + C attributes, which describe the center coordinates, the dimensions, the objectness score and C class confidences for each bounding box. YOLOv3 predicts 3 bounding boxes for every cell and C is the number of classes included for classification.

In YOLOv3 only one cell of the formed grid[change this] is responsible for detecting an object. First, we must ascertain which of the cells this bounding box belongs to. To understand it better let us consider an example[7]. The input image is 416 x 416, and the stride of the network is 32 so the dimensions of the feature map will be 13 x 13. The cell containing the center of the ground truth box of an object is chosen to be the one responsible for predicting the object. Here, the cell is chosen to be the 7th cell in the 7th row of the grid and this cell can predict 3 bounding boxes for the detected object.
So now the model needs to choose one final bounding box to make the prediction. YOLOv3 uses log-space transforms to calculate x,y center co-ordinates, and the width and height of our prediction.

YOLOv3 doesn't predict the absolute coordinates of the bounding box's center but the

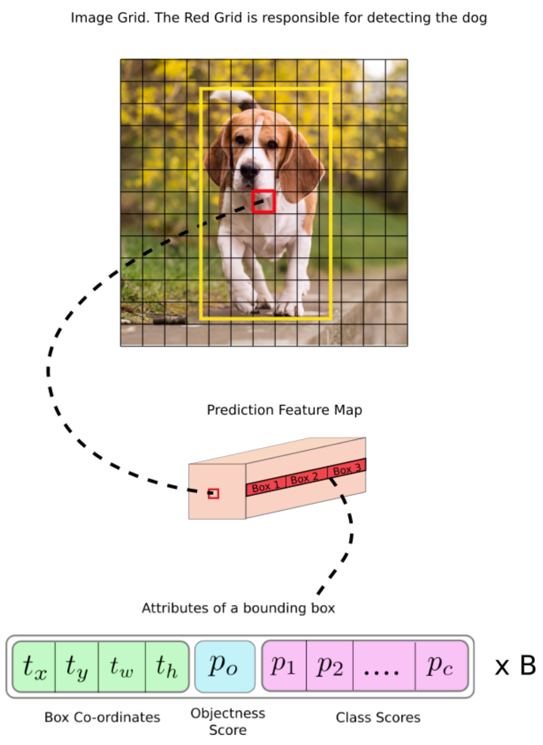


Image Grid. The Red Grid is responsible for detecting the dog
Prediction Feature Map
Box 1
Box 2
Box 3
Attributes of a bounding box
t_x t_y t_w t_h p_o p_1 p_2 .... p_c
x B
Box Co-ordinates
Objectness Score
Class Scores

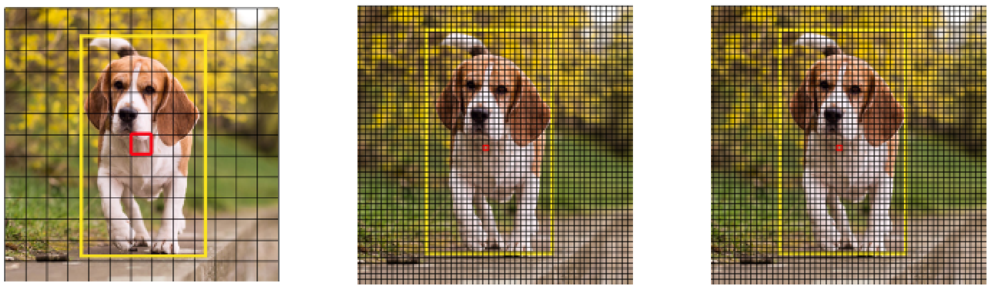
Figure 3.2: Example

normalized (by 1) dimensions of the cell relative to the top-left corner of the image. For example, if the prediction for the center is (0.4, 0.7) in the above given 13x13 feature grid that means that the center lies at (6.4,6,7) i.e the (7,7) cell of the grid.

But by using just this approach we break the theory behind YOLO. For example, consider the prediction to be (1.2,0.7), this leaves the prediction to be in the adjacent cell(8,7), whereas it is was initially predicted to be at cell (7,7). Therefore, to remedy this problem, the output is passed through a sigmoid function, which squashes the output in a range from 0 to 1, effectively keeping the center in the grid which is predicting.

Next comes interpreting the objectness score and class confidence scores. Objectness score denotes the probability that the object is contained in the bounding box predicted, whereas class confidence represents the probabilities of the detected object belonging to a particular class. YOLOv3 dropped the usage of softmax unlike its predecessors as softmaxing the class scores assumes that the classes are mutually exclusive.

YOLOv3 makes predictions across 3 different scales. The detection layer is used to make detection at feature maps of three different sizes, having strides 32, 16, 8 respectively. This means, with an input of 416 x 416, we make detections on scales 13 x 13, 26 x 26 and 52 x 52 as shown in the Figure 3.3. The feature is introduced in the third version of the model to help better the detect smaller objects.



(a) 13x13 (b) 26x26 (c) 52x52

Figure 3.3: Different Scales of Detection

For an image of size 416 x 416, YOLOv3 predicts ((52 x 52) + (26 x 26) + 13 x 13)) x 3 = 10647 bounding boxes. We use the following methods to reduce the from 10647 to 1.

- Thresholding by Object Confidence:

  First, we filter boxes based on their objectness score. Generally, boxes having scores below a threshold are ignored.

- Non-maximum Suppression

  NMS intends to cure the problem of multiple detections of the same image. For example, all the 3 bounding boxes of the red grid cell may detect a box or the adjacent cells may detect the same object. Using the IoU values, one of the 3 boxes is chosen. Intersection over Union(IoU) is the ratio of the area of intersection over the union area occupied by the ground truth bounding box and the predicted bounding box.

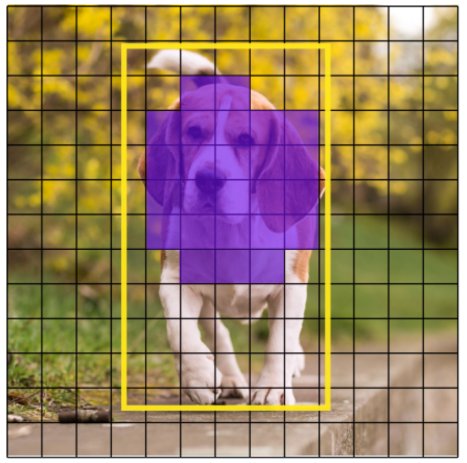

Multiple Grids may detect the same object
NMS is used to remove multiple detections

Figure 3.4: Non-maximum Suppression

3.1.2.2 Implementaion

Our implementation uses Rovit dataset which consists of 7 classes namely 'Person', 'Bicycle', 'Car', 'Motorbike', 'Bus' ,'Traffic sign' and 'Traffic light'. The filters at output layer can calculated using the formula (B x (5 + C))=(3 x (5 + 7))=36.

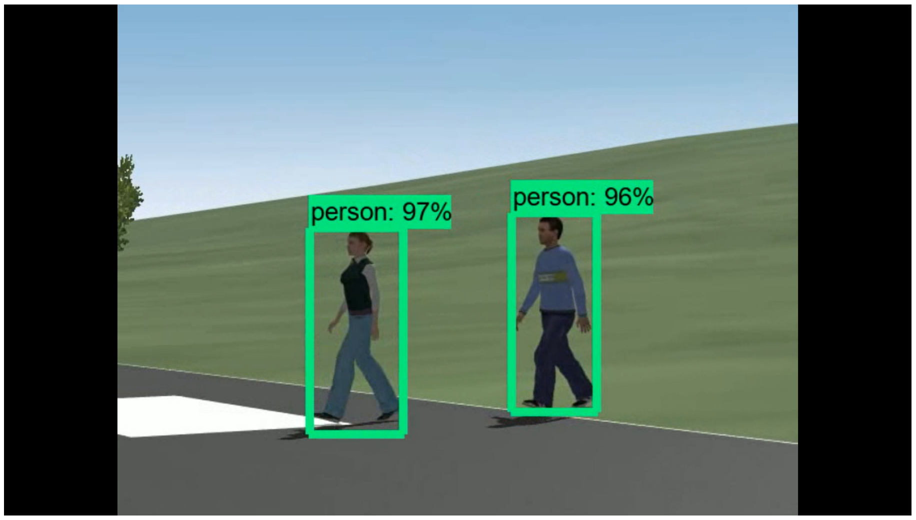


Figure 3.5: Object Detection in CarMaker

From the output image(Figure 3.5) we can see the classes of the detected objects along with their class confidence.

3.2 Traffic Sign Classification

Traffic signs, also known as road signs are signs placed at the side of or above roads to give instructions or provide information to road users. There are a large number of traffic signs that a driver needs to keep track of. We could place these signs as individual classes in the YOLOv3 module but it will increase the number of classes significantly as there are more than 40 traffic sign classes that we are considering which will increase the complexity of the model and time of execution. So, we have created a dedicated module for traffic sign detection.

3.2.1 Traffic Sign Detection Module Pipeline

Our traffic sign classifier take traffic sign images from the output of YOLOv3 module and each traffic sign image follow the pipeline given in Figure 3.6.

- Image Pre-processing
- Detection module

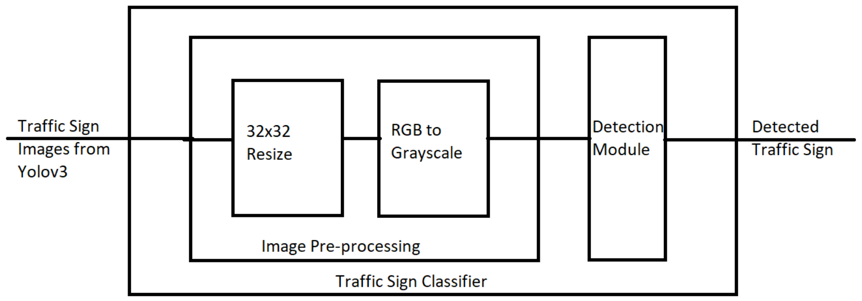


Figure 3.6: Traffic Sign Classifier Pipeline

3.2.1.1 Image Pre-processing

Each traffic sign image detected by the YOLOv3 module is in RGB representation. Although colors in the traffic signs are important for people to recognize them in the real world, traffic signs have different shapes and contents. So, we can ignore the color of traffic signs and only work with the shapes and contents. This will help us to reduce the size of the images significantly thus training and prediction time.

For example: If we take a 32x32 RGB image it will have 32x32x3 pixels. But if we consider the same image but in grayscale, it will have 32x32x1 pixels. Now, this may look relatively little change in pixel value considering the computational power of modern computers. But here we are considering only one image. If we consider all images in the dataset then there will be 153600000 (50,000x32x32x3) pixels in total wherein grayscale case we will only have 51200000 pixels. There is a difference of one hundred and two million, four hundred thousand pixels. This slows down our training significantly considering the performance it will give our model if we use RGB instead of grayscale images. It also slows predicting as we have to work almost three times more in the case of an RGB image.

Also, we are resizing each image to 32x32 pixels because the size of the boundary box from the YOLO module can change in each case. We are taking 32x32 as square image matrix can be rotated, and analyzed in smaller patches.

3.2.1.2 Detection Module

We have adopted a deep learning model to solve this problem. The first model that we have implemented is based on LeNet-5[6] architecture(Figure 3.7). The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully-connected layers and finally a softmax classifier.

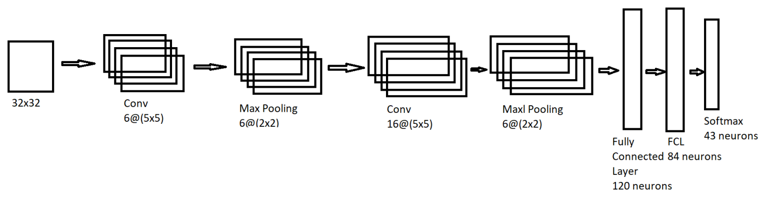


Figure 3.7: LeNet Architecture

We trained this architecture with the German Traffic Sign dataset[9] and the best accuracy we have achieved is 94%. Examining the misclassified images we found that most of the misclassification is caused when the pattern in the actual class and predicted class is similar. Also, classes with complicated patterns are predicted wrong almost all the time. This leads to the conclusion that this model is the facing bias problem.

To solve this issue we changed the architecture of our model(Figure 3.8). We introduced one more convolutional layer and changed the number of neurons in fully connected layers.

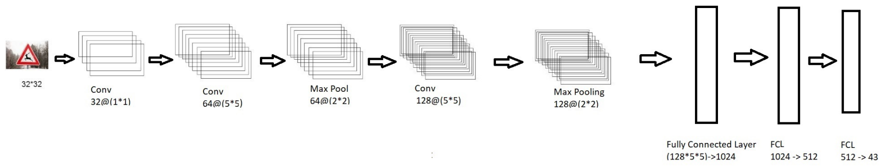


Figure 3.8: Detection Module Architecture

In the first convolutional layer, we are using 32 1x1 kernels. It will produce 32 32x32 layers because as input we have 1 32x32 layer. Here the idea is to distribute the features among 32 layers from 1 layer to get more patterns or features from the input image. Next, we are using a convolutional layer again but this time we are using 64 5x5 kernels. It will produce 64 28x28 layers. Here again, we are distributing features from 32 layers to 64 layers to get more features or pattern but at the same time, we are reducing each layer size(from 32x32 to 28x28) to improve computation. Then we are applying a 22 max-pooling layer. It will result in 64 14x14 layers. It is used to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network and also it extracts the sharpest features of each layer. So given an image, the sharpest features are the best lower-level representation of an image. Next, we are applying 128 5x5 convolutional layers again. It will produce 128 10x10 layers. Then we are applying a max-pooling layer of 2x2 size. It will produce 128 5x5 layers. The idea behind these last two layers is the same as before. Now we flatten the result of the last max-pooling layer and create a 128x5x5 1D tensor. We do this because the next layers are fully connected layers and they take 1D tensor as input. So, after this, there are three fully connected layers. The last layer being the output layer and as we have 43 classes in the dataset, the last layer has 43 neurons. The second last layer contains 512 neurons and the third last layer has 1024 neurons. The output from the convolutional layers represents high-level features in the data. Fully connected layers connect every neuron in one layer to every neuron in another layer. The flattened matrix goes through a fully connected layer to classify the images. Fully connected layers simply learn non-linear functions of high-level features (from convolution and pooling layers) and classify images based on High-level features. Here in each fully connected layer, we are using ReLU activation function and in each convolutional layer, there is no padding and stride is 1x1.

3.2.2 Model Statistics

We have tested this model with german traffic sign dataset.

3.2.2.1 Dataset Statistics

- This dataset contains 43 classes of different traffic sign.
- It has more than 50,000 images in total.
- Some examples of classes would be Stop, No vehicles, Speed Limit etc.

3.2.2.2 Training

- We have trained the model with 34799 images(Figure 3.9).

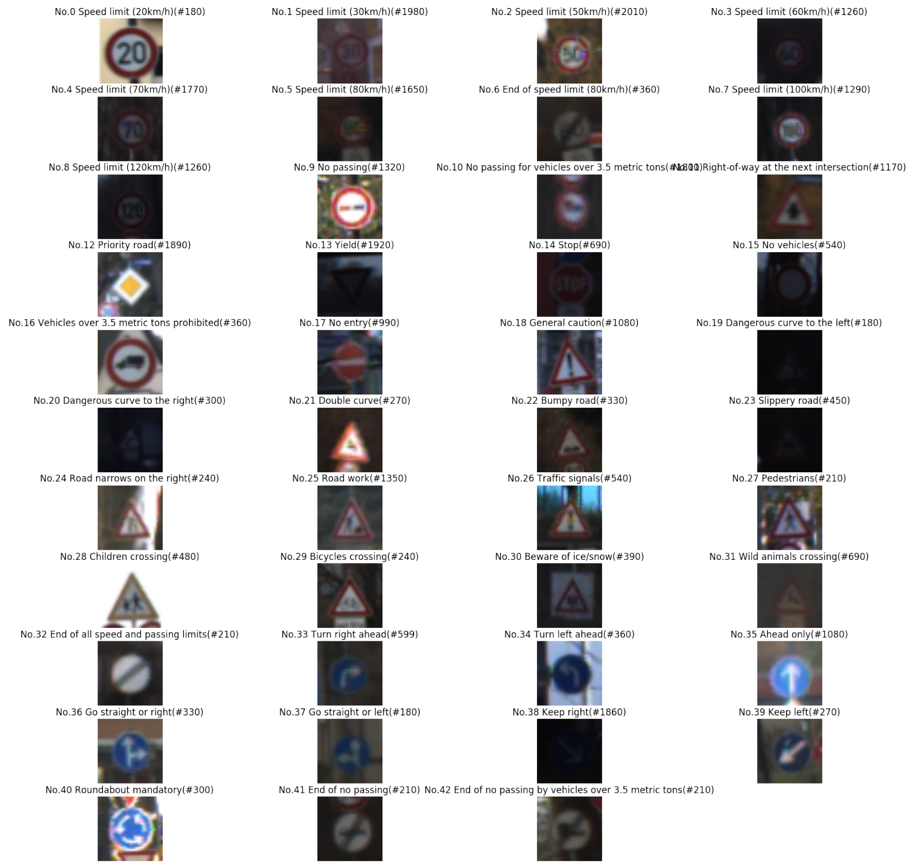

No.0 Speed limit (20km/h)(#180)
No.1 Speed limit (30km/h)(#1980)
No.2 Speed limit (50km/h)(#2010)
No.3 Speed limit (60km/h)(#1260)
No.4 Speed limit (70km/h)(#1770)
No.5 Speed limit (80km/h)(#1650)
No.6 End of speed limit (80km/h)(#360)
No.7 Speed limit (100km/h)(#1290)
No.8 Speed limit (120km/h)(#1260)
No.9 No passing(#1320)
No.10 No passing for vehicles over 3.5 metric tons(#1800)
No.11 Right-of-way at the next intersection(#1170)
No.12 Priority road(#1890)
No.13 Yield(#1920)
No.14 Stop(#690)
No.15 No vehicles(#540)
No.16 Vehicles over 3.5 metric tons prohibited(#360)
No.17 No entry(#990)
No.18 General caution(#1080)
No.19 Dangerous curve to the left(#180)
No.20 Dangerous curve to the right(#300)
No.21 Double curve(#270)
No.22 Bumpy road(#330)
No.23 Slippery road(#450)
No.24 Road narrows on the right(#240)
No.25 Road work(#1350)
No.26 Traffic signals(#540)
No.27 Pedestrians(#210)
No.28 Children crossing(#480)
No.29 Bicycles crossing(#240)
No.30 Beware of ice/snow(#390)
No.31 Wild animals crossing(#690)
No.32 End of all speed and passing limits(#210)
No.33 Turn right ahead(#599)
No.34 Turn left ahead(#360)
No.35 Ahead only(#1080)
No.36 Go straight or right(#330)
No.37 Go straight or left(#180)
No.38 Keep right(#1860)
No.39 Keep left(#270)
No.40 Roundabout mandatory(#300)
No.41 End of no passing(#210)
No.42 End of no passing by vehicles over 3.5 metric tons(#210)

Figure 3.9: Classes in German Traffic Sign Dataset

3.2.2.3 Test Results

- Accuracy of the Model is 97
- Least error in Class : Vehicles over 3.5 metric tons prohibited.
- Most error in Class : Speed limit(80 km/h)

3.2.2.4 Computation Time

Average Time for computation is 0.0033 s. Bar chart in Figure 3.10 showcase time of execution for different images.

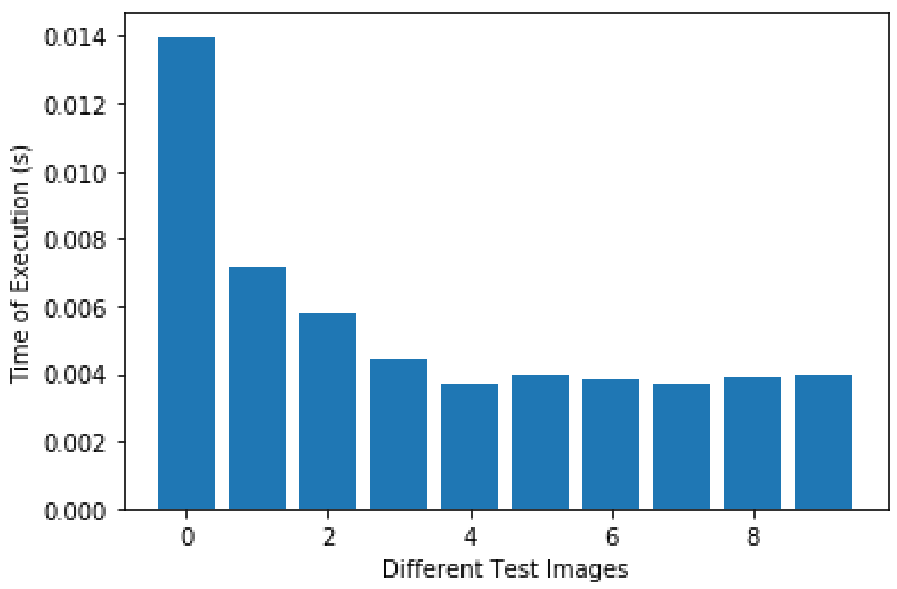


Figure 3.10: Computation Time of Traffic Sign Classifier

3.3 Traffic Light Classification

Traffic lights, also known as traffic signals, traffic lamps, stop lights, are used to control the movement of traffic. They are placed in roads at intersections and crossings. According to the Vienna Convention on road and signals, all traffic signals must be in some form of red, yellow, and green. When the traffic signal with three aspects is arranged horizontally or sideways, the arrangement depends on the rules of the road. In right-lane countries, the sequence (from left to right) is red-yellow-green. In left-lane countries, the sequence is green–yellow–red. In this project, we have adopted the right-lane rule of driving.

A report on the findings of an eight-year study conducted by the National Highway Traffic Safety Administration (NHTSA) says there were on average 1,578 fatalities each year resulting from two-vehicle traffic crashes at intersections controlled by traffic signals[14]. Approximately 51% of those fatal crashes were caused by drivers who ran red lights. So, we have implemented a dedicated module to classify traffic light. In this chapter we will be looking at our traffic light classifier in detail and why we choose it.

3.3.1 Traffic light Pipeline

Our traffic light classifier take traffic light images from the output of YOLO module and each traffic light image follow pipeline given in Figure 3.11.

- Image Pre-processing
- Feature Extraction
- Prediction

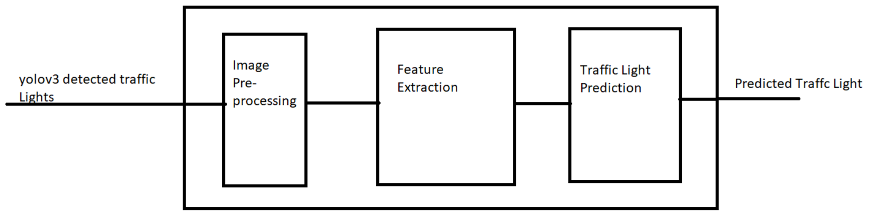


Figure 3.11: Traffic Light Classifier Pipeline

3.3.1.1 Image Pre-processing

The traffic light will be first detected through the YOLO model then it will be identified by this module. So, the size of the boundary box of a traffic light will not be fixed. But to follow our traffic light classifier we need a fixed height of the traffic light image. So, we have converted each traffic light image to a 32x32 image. We are taking 32x32 as a square image matrix which can be rotated and analyzed in smaller patches and as the size of the boundary box from the YOLO module can change on each case.

Now there may be unnecessary information and noise in the 32x32 RGB image which can lead to incorrect identification of traffic light. We are using Gaussian Filtering to achieve this. A Gaussian filter is a linear filter. The Gaussian filter reduces contrast and as increasing the contrast will increase the visible noise, especially in the shadow areas, the gaussian filter works great in this case to mask all the noises of the image. Figure 3.12 is an example of how this works.

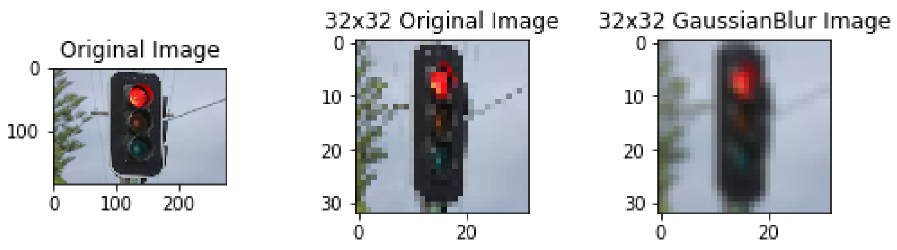


Figure 3.12: Image Prepossessing I

In the Figure 3.12 we can clearly see that there is less noise in the third image than there is noise in the first or second image.

Also, we are cropping 4 rows from both upper and lower end of image as we do not need whole traffic light board we only need the part of the board where these lights are present.

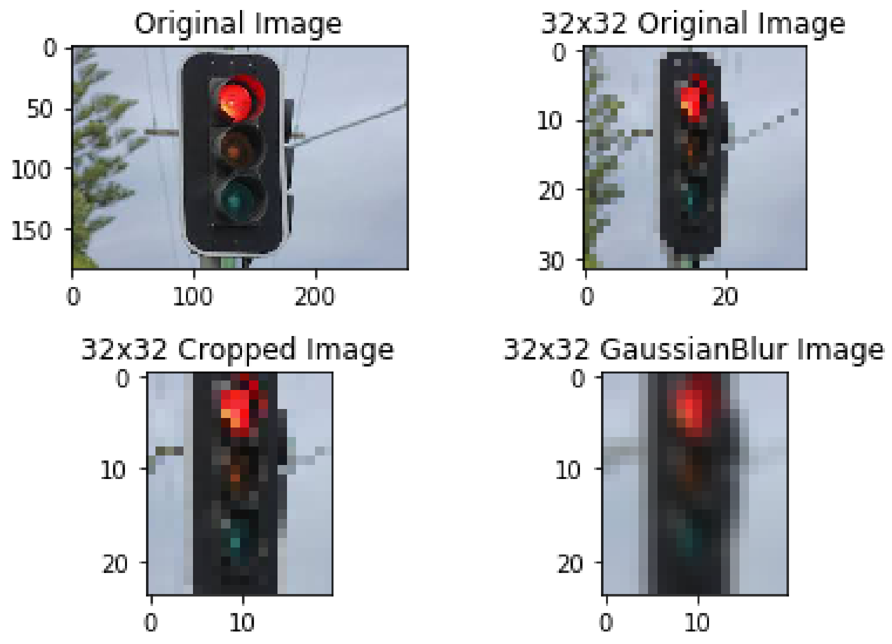


Figure 3.13: Image Prepossessing II

In Figure 3.13 we can see the final image (32x32 Gaussian Blur Image) only has the traffic sign than all other objects in the original image.

3.3.1.2 Feature Extraction

Traffic Light classifier module we are interested in a feature that can identify a light is illuminated. So, we are using brightness feature of an image.

Now to get the brightness feature from an image we will need to convert RGB image to HSV image. HSV is an alternative representation of the RGB color model, where H stands for hue, S stands for saturation, and V stands for Value. We are only interested in the value component only.

Value describes in brightness or intensity of the color, from 0-100 percent, where 0 is completely black, and 100 is the brightest and reveals the most color. Now, we only need to get three positions of images where red, green, and yellow light can be illuminated. Because then we only need to compare the average brightness value of these three regions and we can identify which one is illuminated at a moment. As we are using the right-lane driving convention we know red will be at the top, yellow at the middle, and green at the bottom. So, we have divided the preprocessed image into three segments horizontally. Now we are taking an average brightness value of the top segment to get the brightness of the red segment. Similarly, we are taking an average brightness of the middle and the bottom segment to get the brightness of the yellow and green segment respectively. The segment with the highest brightness is the one illuminated. So if we apply this logic to the preprocessed image we will get results like Figure 3.14.

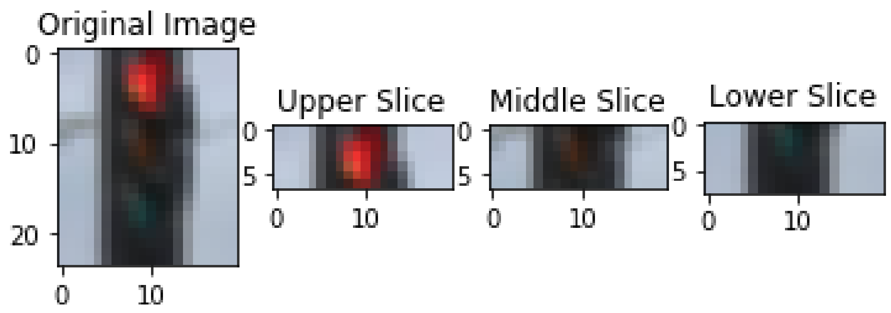


Figure 3.14: Divided Traffic Light Segments

3.3.1.3 Prediction

This step compares the average brightness from previous region and predicts the light corresponding to the highest brightness one as illuminated.

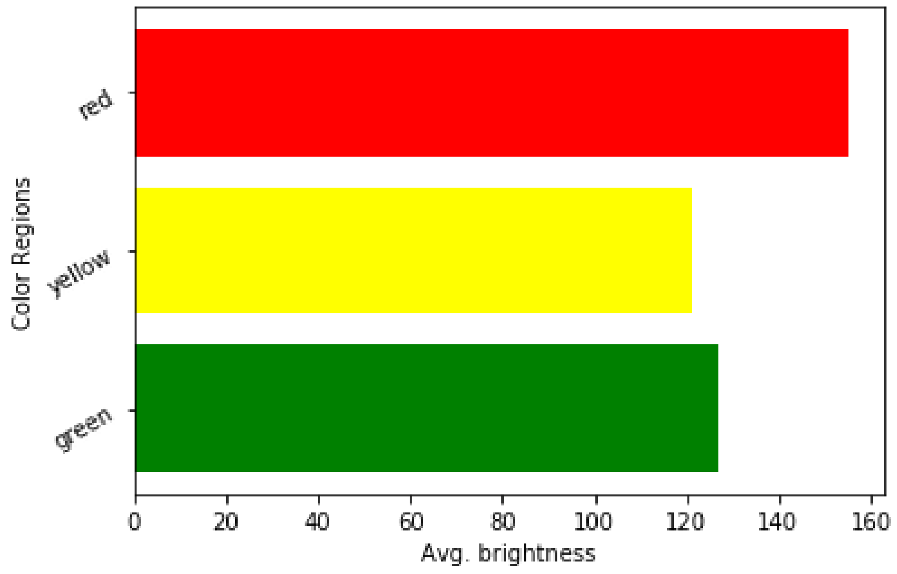


Figure 3.15: Average Region Brightness

If we plot the average brightness value for these three regions from feature extraction we get the chart shown in Figure 3.15.

It is clearly visible that the region corresponding to red is most bright. So this model will detect a red traffic light signal which is true if verified with the base image.

3.3.2 Model Statistics

3.3.2.1 Traffic Sign Testing Dataset Statistics

- No of Images with Red traffic light illuminated: 180
- No of Images with Yellow traffic light illuminated: 35
- No of Images with Green traffic light illuminated: 429

3.3.2.2 Test Result

- 100% accuracy for Red traffic light illuminated.
- 100% accuracy for Yellow traffic light illuminated.
- 99% accuracy for Green traffic light illuminated.

3.3.2.3 Computation Time

Average Time for computation is 0.0033s (Figure 3.16).

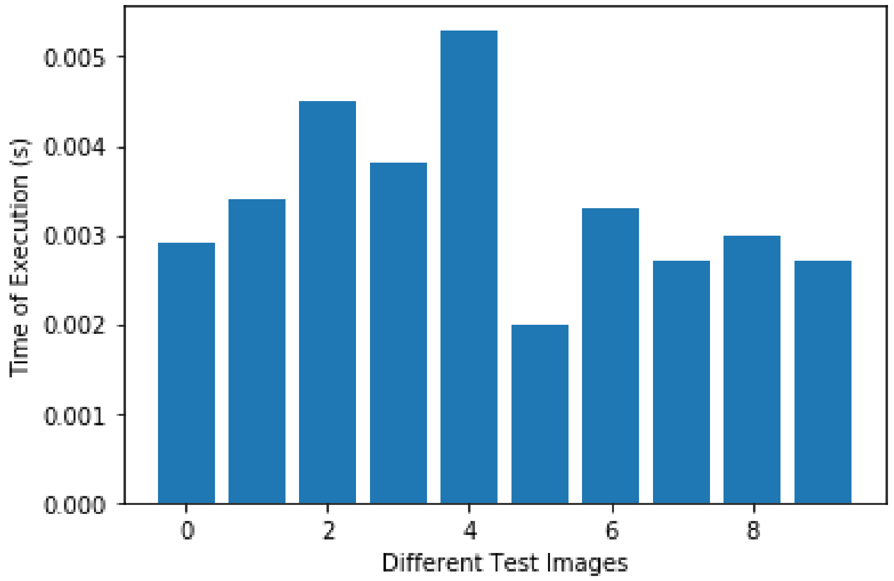


Figure 3.16: Computation Time:Traffic Light

3.4 Lane Detection and radius of road estimation

Lane detection is to detect lanes on the road and provide the accurate location and shape of each lane. It serves as one of the key techniques to enable modern assisted and autonomous driving systems.

Here we have proposed a lane detection pipeline with computer vision techniques.

3.4.1 Lane Detection Pipeline

- Image Pre-processing
- Binary Image
- Perspective Transform
- Lane Detection
- Reverse Perspective Transform
- Radius of Curvature

3.4.1.1 Image Pre-processing

First, we resize every image to 1280x720 pixels. This is to standardize the image size as well as ease of computation and parameter choice during the next phases.

3.4.1.2 Binary Image

In this step, we wanted to generate a binary image matrix. We create this binary matrix in such a way that the only contestant for lane pixel positions will only be one other will be zero. Here we are interested only in steep edges that are most likely to edges. To achieve this we are using the Sobel operator[12].

3.4.1.2.1 Sobel Operator

The Sobel operator performs a 2-D spatial gradient measurement on an image and so emphasizes regions of high spatial frequency that correspond to edges. Typically it is used to find the approximate absolute gradient magnitude at each point in a grayscale image. Sobel operator can be used to detect both vertical edge and horizontal edge.

-1	0	-1
-2	0	-2
-1	0	-1

Table 3.2: Vertical Sobel Operator

-1	-2	-1
0	0	0
-1	-2	-1

Table 3.3: Horizontal Sobel Operator

Vertical Edge detection

Sobel operator consists of a mask. When we apply this mask on the image it prominent vertical edges. It simply works like a first-order derivative and calculates the difference of pixel intensities in an edge region. For example, Table 3.2 is a 3x3 vertical mask for Sobel operation:

As the center column is of zero so it does not include the original values of an image but rather it calculates the difference of right and left pixel values around that edge. Also, the center values of both the first and third columns are 2 and -2 respectively. This gives more weight age to the pixel values around the edge region. This increases the edge intensity and it becomes enhanced comparatively to the original image. In the implementation 3x3 mask used for vertical edge detection.

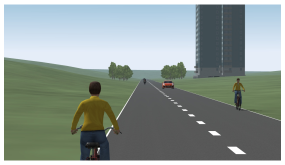
Horizontal Edge detection

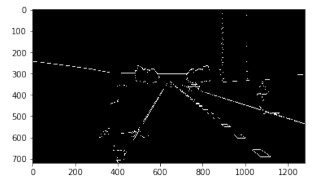
The Table 3.3 is an example of horizontal mask for sobel operation: This mask will prominent the horizontal edges in an image. It also works on the principle of the above vertical mask and calculates difference among the pixel intensities of a particular edge. As the center row of mask consists of zeros so it does not include the original values of edge in the image but rather it calculates the difference of above and below pixel intensities of the particular edge. Thus increasing the sudden change of intensities and making the edge more visible. In the implementation 3x3 mask used for horizontal edge detection.

Post-processing

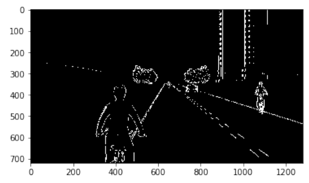
After each Sobel operation, we are using a minimum threshold and a maximum threshold. If the gradient output of a pixel after applying Sobel operation is inside that range then there will be one in the output matrix corresponding that pixel otherwise it will be zero. During implementation, we experimented with a wide range of threshold values and we found the best result in the range [15,150] for vertical edge detection and [30,150] for horizontal edge detection.

Figure 3.17 is an example of all the steps mentioned before.


(a) Original Image



(b) Horizontal Edge Detection



(c) Vertical Edge Detection

Figure 3.17: Sobel Operation

3.4.1.2.2 Magnitude of Gradient

In Sobel's operation, we can see that the *x* gradient (using the vertical mask) does a cleaner job of picking up the lane lines, but we can see the lines in the *y* gradient (using the horizontal mask) as well. Next, we will be applying a threshold to the overall magnitude of the gradient, in both *x* and *y*. The magnitude or absolute value of the gradient is just the square root of the squares of the individual *x* and *y* gradients. For a gradient in both the *x* and *y* directions, the magnitude is the square root of the sum of the squares.

$$absSobelX = \sqrt{(sobelx)^2}$$

$$absSobelY = \sqrt{(sobely)^2}$$

$$absSobelXY = \sqrt{(sobelx)^2 + (sobely)^2)}$$

Similarly, like Sobel operation, we are using minimum threshold and maximum threshold as the post-processing step. We found that [50,150] range gives us the best result. If we apply the magnitude of the gradient to the same image we tested during Sobel we get Figure 3.18.

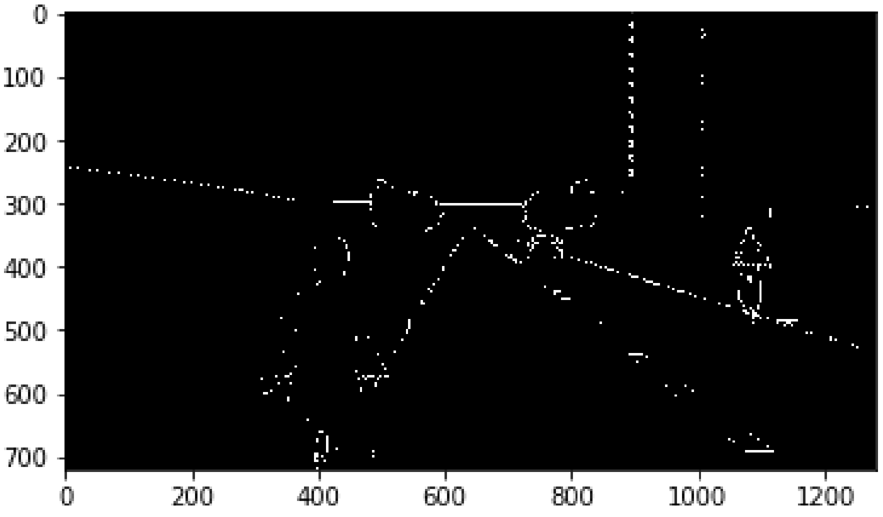


Figure 3.18: Magnitude of Gradient

3.4.1.2.3 Direction of Gradient

We can observe the last two process does great for any edge detection. But for lane detection we do not want to detect all the edges, we're interested only in edges of a particular orientation. So now we will explore the direction, or orientation, of the gradient. The direction of the gradient is simply the inverse tangent (arctangent) of the y gradient divided by the x gradient:

$$arctan(sobel_y/sobel_x)$$

Each pixel of the resulting image contains a value for the angle of the gradient away from horizontal in units of radians, covering a range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$. An orientation of 0 implies a vertical line and orientations of $+/-\frac{\pi}{2}$ imply horizontal lines.

Here we are also doing post-processing by using a threshold range. We got the best results for (0.7,1.3) range. If we apply direction of gradient to the same image we tested during Sobel we get the Figure 3.19.

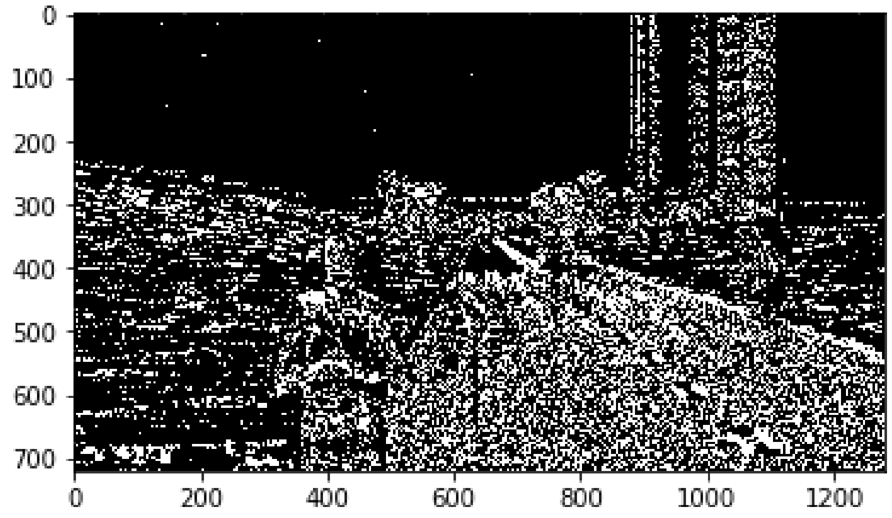


Figure 3.19: Direction of Gradient

3.4.1.2.4 HSL and Color Threshold

In the previous methods, we are considering grayscale images. Lane detection works fine for most of the time if we only create the output matrix by the above methods but with different color and contrast, there may be a problem with only considering gradient. Figure 3.20 is an example where previous gradient methods face problems.

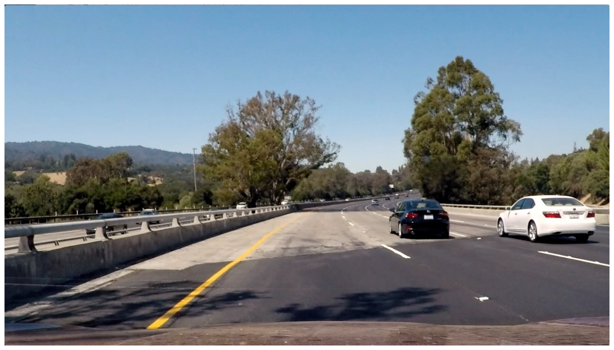


Figure 3.20: Corner Case

Figure 3.21 is how the combined binary looks like:

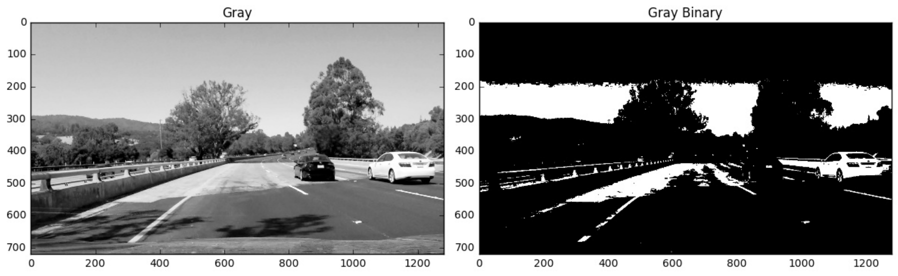


Figure 3.21: Corner Case:Combined Binary

We can clearly see lanes are not recognizable after some distance. To solve this problem we are using HSL and color threshold. HSL is an alternative representation of the RGB color model, where H stands for hue, S stands for saturation, and L stands for lightness. We have already discussed about hue and saturation in the traffic light classification module. Lightness is represented in percentage and 0% means black and 100% means white. Now after converting the Figure 3.20 to HSL representation and plotting the S matrix we get the Figure 3.22.

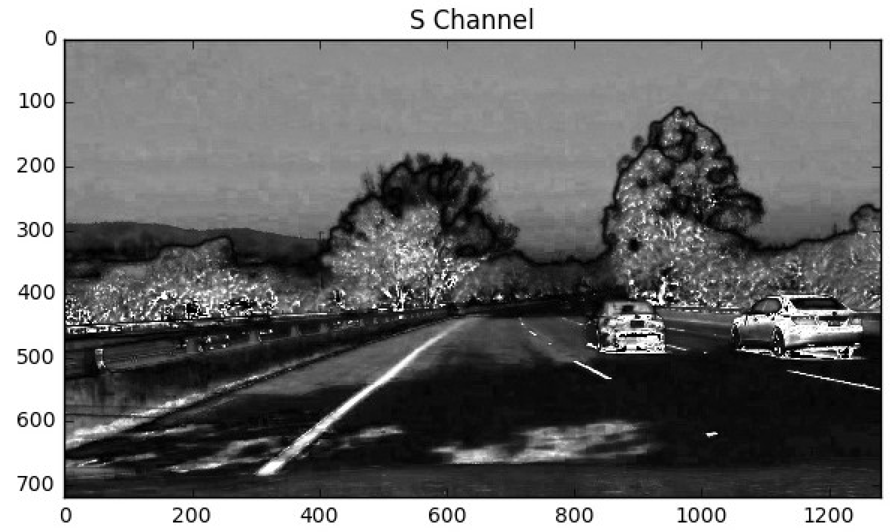


Figure 3.22: Corner Case:S Channel

Next we are applying thresholding and creating a binary matrix which contains one corresponding to the pixel positions which are in this threshold range. During implementation we found that (170,255) range gives best result. If we apply this threshold to the Figure 3.22 we get Figure 3.23.

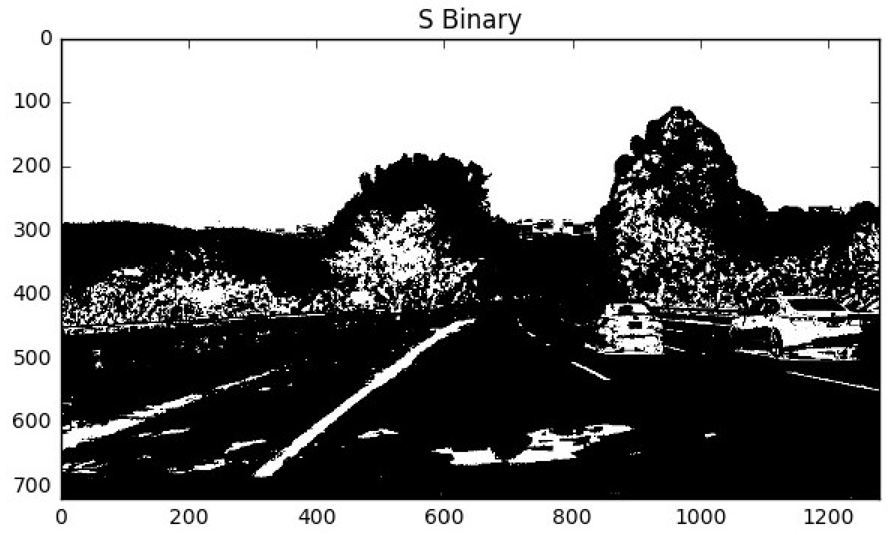


Figure 3.23: Corner Case:S Binary

The Figure 3.23 is much better than the binary image we got from gradient methods.

3.4.1.2.5 Combined Binary Matrix

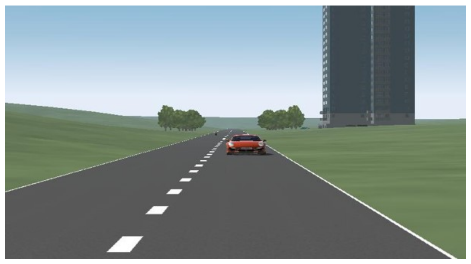
Finally we combine all the binary matrix from the previous discussion. Let's assume, $gradx_{bin}$ is the binary matrix resulted from vertical edge detection, $grady_{bin}$ is the binary matrix resulted from horizontal edge detection, mag_{bin} is the binary matrix resulted from magnitude of gradient, dir_{bin} is the binary matrix resulted from direction of gradient, hls_{bin} is the binary matrix resulted from HSL and thresholding the we say combined binary matrix as follows

$$Combined_{bin} = (gradx_{bin} AND\ grady_{bin})\ OR\ (mag_{bin}\ AND\ dir_{bin})\ OR\ hls_{bin}$$

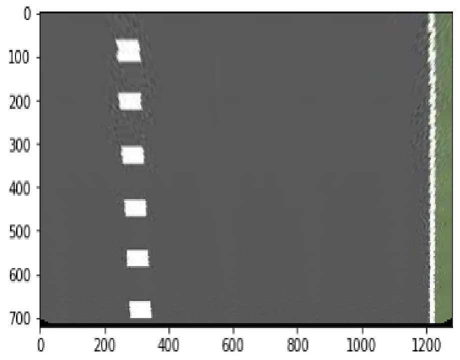
3.4.1.3 Perspective Transform

A perspective transform[13] maps the points in a given image to different, desired, image points with a new perspective. In the perspective transform, we are most interested in is a bird's-eye view transform that lets us view a lane from above; this will be useful for calculating the lane curvature later on. When we apply a perspective transform, we choose four points manually and from previous images, we decide the destination points such that mapping these points to the destination points we will be getting the top view of the lane. During the implementation we choose the following array as the source and destination points respectively :
source points = [[250,700], [1200, 700], [550, 450], [750, 450]]
destination points = [[250 , 700] , [1200, 700], [300, 50], [1000, 50]]

Using the above parameter if we apply the perspective transformation to a frame taken from IPG CarMaker we get the Figure 3.24.



(a) Original Image



(b) Perspective Transform Image

Figure 3.24: Perspective Transform Operation

It is obvious from the Figure 3.24 that if we choose the original image to find the radius of curvature of the road then we will be getting the wrong result but the perspective transformed image almost accurately shows the actual orientation of lane lines and radius of curvature.

3.4.1.4 Lane Detection

We will be detecting lane on perspective transformed the combined binary image. Here peaks of the histogram method are used to find the lanes. We create a histogram from the sum of columns of the bottom half of the combined binary image. In our combined binary image, pixels are either 0 or 1, so the two most prominent peaks in this histogram will be good indicators of the x-position of the base of the lane lines. We can use that as a starting point for where to search for the lines. From that point, we can use a sliding window, placed around the line centers, to find and follow the lines up to the top of the frame. There are a few parameters here that we have to consider:

- Number of windows: Number of sliding windows.
- Margin: Width of the window, in this area lane search is done.
- Minimum pixel: Minimum number of pixels found to recenter.

During implementation we fixed the image to 1280x720 pixels and took the following parameter values and got the best results:

- Number of windows: 9
- Margin: 110
- Minimum Pixel: 50

Figure 3.25 shows the above parameters and how the histogram is used to find lane.

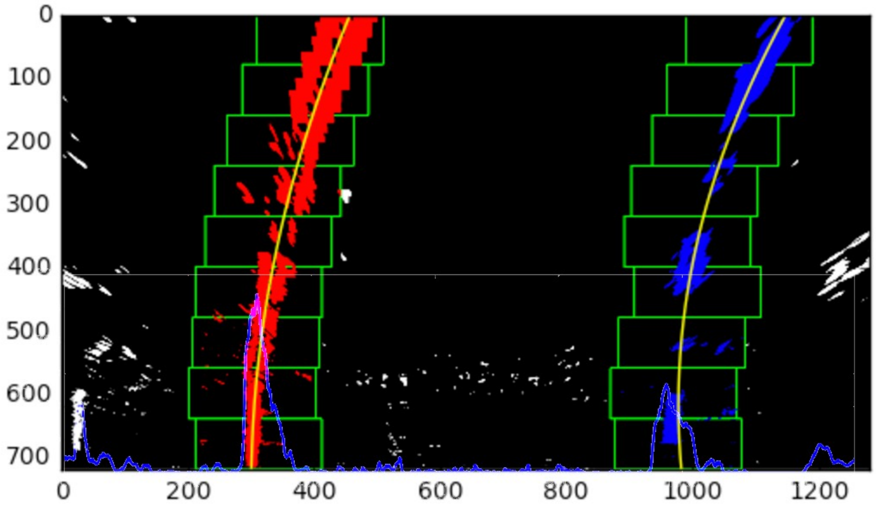


Figure 3.25: Lane Detection

3.4.1.5 Reverse Perspective Transform

After lanes are detected we transfer the image back to its original form by mapping destination points mentioned in perspective transform back to original points.

3.4.1.6 Radius of Curvature

Lane pixels detected during the lane detection phase used to calculate the radius of curvature. Lane pixel's x and y pixel positions fitted in a second order polynomial curve:

$f(y) = Ay^2 + By + C$

Here lane pixels are fitted for $f(y)$ rather than $f(x)$ because the lane lines in the warped image are near vertical and may have the same x value for more than one y value. Radius of curvature at any point x of function $x = f(y)$ given as follows:

$$R_{curve} = \frac{[1+(\frac{dy}{dx})^2]^{3/2}}{|\frac{d^2x}{dy^2}|}$$

In the case of the second order polynomial above, the radius of curvature becomes:

$$R_{curve} = \frac{[1+(2Ay+B)^2]^{3/2}}{|2A|}$$

3.5 Depth Estimation

Depth estimation or calculating the distance between the autonomous vehicle and the detected objects is a crucial step in maneuvering the vehicle in a safe and efficient manner. Depth estimation using cameras is done using either of the following ways.

- Monocular depth estimation:

  It involves processing of images from a single point of view i.e. using the data acquired using a single camera. The data is usually processed using machine learning models or by combining the image data with other sensors.

- Stereo depth estimation:

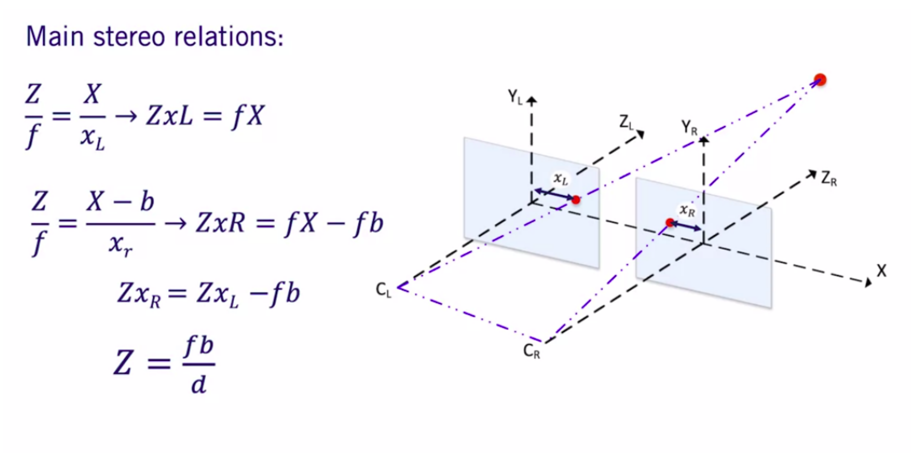
  It involves processing of images from multiple points of view i.e. using the data acquired using multiple cameras. The depth is estimated by using multiple cameras to triangulate the detected object.

The first model we researched was by using monocular depth estimation. This models uses the concept of triangular similarity. In this the depth(D) is estimated using the focal length(F) of the camera, width(W) of the object and width(P) of the object in pixels. Using the above information depth can be calculated as $D = (W * F)/P$. The accuracy of this method drops when the detected object is not aligned with the camera i.e. as the pixel width of the rotated object is inaccurate.

Despite the recent progress in monocular depth estimation, the accuracy of the results is still far behind the stereo depth estimation technique. This mainly due to the texture and structural variations, object occlusions, and rich geometric detailing which limits the development of monocular depth estimation. In our project, we have used Stereo depth estimation to triangulate and locate the detected objects.

3.5.1 Stereo Depth Estimation

Stereo depth estimation[4] employs the concept of triangulation i.e. the process of determining the location of a point by forming triangles to it from known points. Here the location of the object is determined using the location of the cameras as known points, so the distance of the object can be calculated as the cameras are fixed on the vehicle-no relative motion with respect to the vehicle.


Main stereo relations:

$$\frac{Z}{f} = \frac{X}{x_L} \rightarrow ZxL = fX$$

$$\frac{Z}{f} = \frac{X-b}{x_r} \rightarrow ZxR = fX - fb$$

$$Zx_R = Zx_L - fb$$

$$Z = \frac{fb}{d}$$

Figure 3.26: Triangulation Image and Equations[4]

The main challenge that arises in depth estimation is locating the same object in both camera images. Identifying the position of an object is done by selecting a part of the second image which most resembles the object selected in the first image as shown in Figure 3.27.

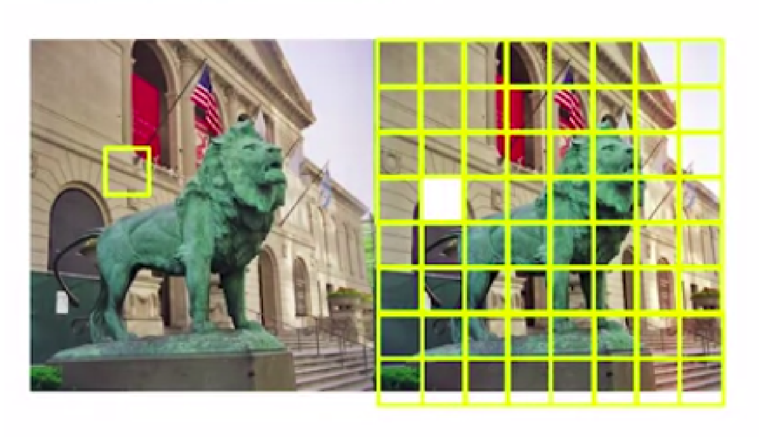


Figure 3.27: Detecting Most Similar Component of the Second Image[4]

The process of locating the object in the second image can be done by traversing through the second image systematically i.e choosing the left topmost corner and moving the chosen frame across the image such that it covers the entire image to find the most suitable frame. But this approach consumes a lot of computation time. The process of locating the appropriate frame can be shortened by leveraging the fact that upon placing the two cameras at the same height the process of traversing through the second image can be restricted to moving from only left to right in the same height range as that of the object chosen in the first image(Figure 3.28).

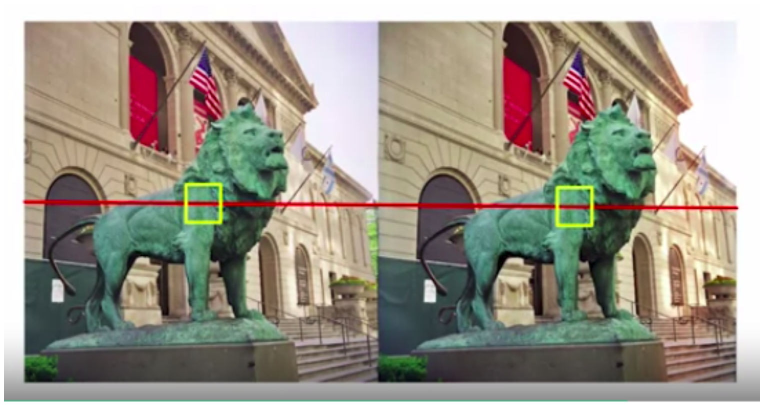


Figure 3.28: Optimised Search[4]

3.5.2 Implementation and examples

The method used for finding the best match with the two images is minimizing the euclidean distance calculated from the two images i.e finding the average of the distance between every pixel of the selected part of the images.
Upon finding the closest match using the above formula, depth/distance of the object can be calculated.

Consider the following set of images(Figure 3.29) that have the same object but at different positions of the image. As both the cameras are at the same height we can see that the object is present at the same height range in both the images.

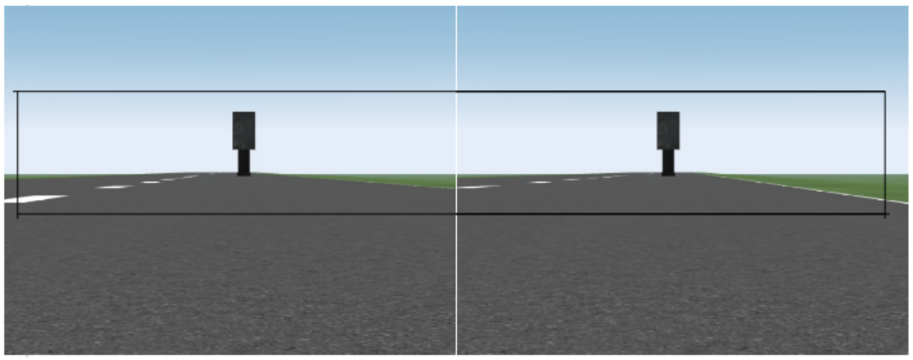


Figure 3.29: Input Left and Right Images

The disparity in the images can be calculated using the highlighted strips of the images. By using the above formula the depth of the object present can be calculated as we have disparity values.

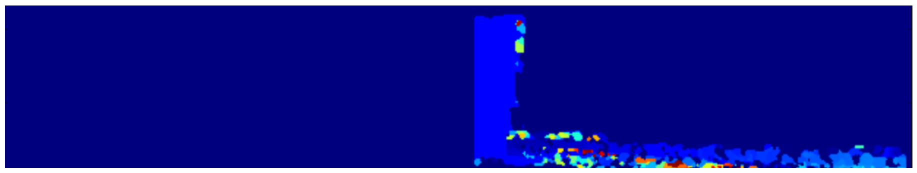


Figure 3.30: Processed Disparity Output

3.6 LIDAR

LIDAR—Light Detection and Ranging—is a remote sensing method used to examine the environment by projecting infrared lasers and studying the reflecting rays which are captured by sensors. Distance to the object is determined by recording the time between transmitted and reflected rays and by using the speed of light to calculate the distance traveled. The groups formed by the reflections can be analyzed to detect, classify and segment the objects present in the environment.

We have used the euclidean clustering algorithm[8] to group the point cloud data generated by LIDAR which in-turn is used for object detection.

3.6.1 Clustering Algorithms

Clustering is the task of dividing the population or data points into several groups such that data points in the same groups are more similar to other data points in the same group than those in other groups. In simple words, the aim is to segregate groups with similar traits and assign them into clusters. The clustering types can be divided into the following categories.

- Hard clustering:
  In this the data points are strictly classified into different groups i.e it either belongs to the group or not.

- Soft clustering:
  Unlike hard clustering, a data point can belong multiple groups with a set probability value for each group.

Whereas clustering algorithms fall into the below four categories.

- Connectivity models:
  These models are based on the notion that the data points closer in data space exhibit more similarity to each other than the data points lying farther away.

- Centroid models:

  These are iterative clustering algorithms in which the notion of similarity is derived by the closeness of a data point to the centroid of the clusters.

- Distribution models:

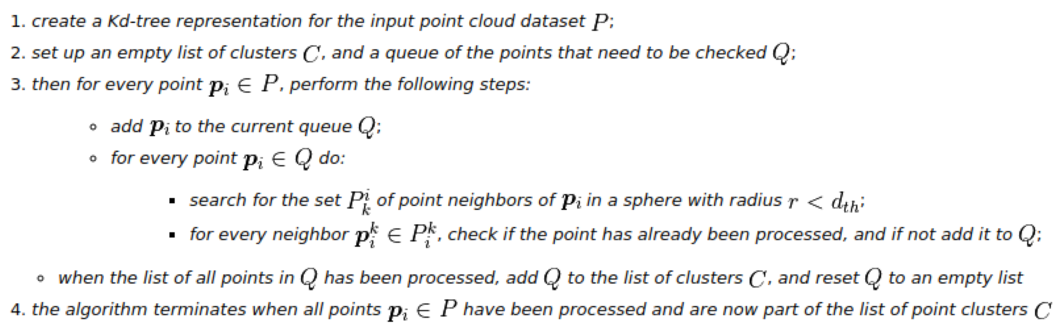
  These clustering models are based on the notion of how probable is it that all data points in the cluster belong to the same distribution.

- Density Models:

  These models search the data space for areas of varied density of data points in the data space.

3.6.2 Euclidean Clustering Algorithm

Euclidean clustering fall under hard clustering and centroid model as the data points classified fall under only one of the available groups and it is implemented in an incremental fashion.



```
1. create a Kd-tree representation for the input point cloud dataset $P$;
2. set up an empty list of clusters $C$, and a queue of the points that need to be checked $Q$;
3. then for every point $\boldsymbol{p}_i \in P$, perform the following steps:
        ◦ add $\boldsymbol{p}_i$ to the current queue $Q$;
        ◦ for every point $\boldsymbol{p}_i \in Q$ do:
                ▪ search for the set $P^i_k$ of point neighbors of $\boldsymbol{p}_i$ in a sphere with radius $r < d_{th}$;
                ▪ for every neighbor $\boldsymbol{p}^k_i \in P^k_i$, check if the point has already been processed, and if not add it to $Q$;
    ◦ when the list of all points in $Q$ has been processed, add $Q$ to the list of clusters $C$, and reset $Q$ to an empty list
4. the algorithm terminates when all points $\boldsymbol{p}_i \in P$ have been processed and are now part of the list of point clusters $C$
```

Figure 3.31: Clustering Algorithm[8]

The algorithm takes in a single point and adds all the points close enough into the point and contains this process till the no other points are inside the threshold, to form a group, this process continues till all the data points are considered to be in any group. During implementation, many variables are taken into consideration such as minimum cluster/group size, tolerance distance, etc.

3.6.3 Implementation and Examples

The implementation process can divided into the following steps.

- Extracting the point cloud data: The data from the PCD format of the LIDAR is extracted and the data is then down-sampled to achieve better performance.

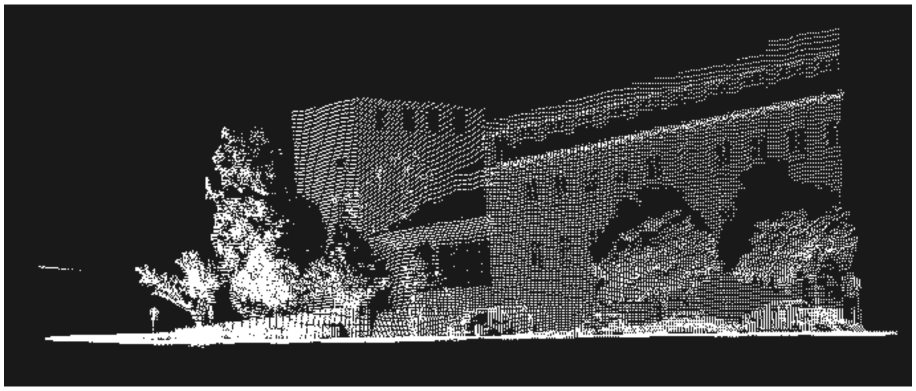


Figure 3.32: Input LIDAR Data

- Variable selection: In this step the variables such as minimum cluster size, maximum cluster size and cluster tolerance are set.

- Clustering: After setting up the variables, the above mentioned algorithm is used to extract the different clusters(Figure 3.33) from the given point cloud(Figure 3.32) and saved into different cluster files/variables for any further potential uses.

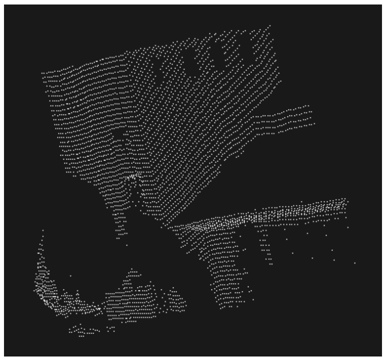


Figure 3.33: Detected Cluster Example

3.7 Camera Integration

All the individual modules are integrated with CarMaker. CarMaker streams data captures in bytes format to port 2210. Each image has a header file and in the header file camera number is mentioned. As we have used two cameras for this implementation we will be getting two images for each instance. Now right image will first be given input to YOLOv3 module and lane detection module. Left image will only be used in depth estimation. YOLOv3 module will return object that are present in the right frame. If there is any traffic light present in this frame then traffic light image is given as an input to traffic light classifier. Similarly, if traffic sign is detected then it is given as an input to traffic sign classifier. Every object is given as an input to depth estimation module. Now results from these modules are combined and shown in a frame with each detail detected by all modules shown. Figure 3.34 showcases the pipeline of different modules related to the camera sensor.

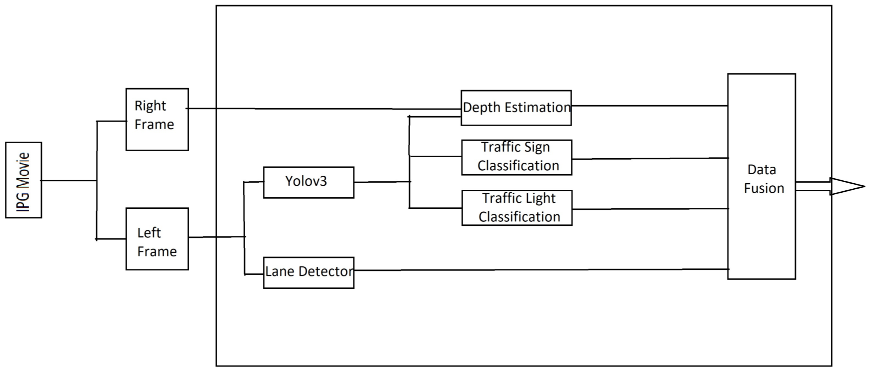


Figure 3.34: Camera Integration

Chapter 4

Conclusion and Future work

4.1 Conclusion

Driving is a complex phenomenon with lots of dependencies. Developing a level 5 self-driving car will take time. From the results of this thesis, we can create and test all modules of the self-driving car in simulators. We can run simulations in large numbers parallelly to train and find out any situation that causes the anomaly. Thus we can reduce development time significantly by using simulators and the fact that self-driving cars can share their experience in a matter of minutes to other self-driving cars, unlike humans. So, the state of the art development for self-driving cars can be done in simulators with significant efficiency and low development cost.

4.2 Future work

As per the current development state, we have only integrated the camera sensor modules to the IPG CarMaker, LIDAR sensor module is implemented but not integrated. So, as future work LIDAR sensor modules can be integrated to improve environment perception. Also, we can use other sensors like RADAR, ultrasonic sensors to improve environment perception.

4.3 Contributions

- YOLOv3 training
- Traffic Light Classification Research
- Depth Perception Research and Implementation
- Depth Perception module integration
- LIDAR data clustering Research and Implementation

I thank Souvik Mandal for his contribution in developing the CarMaker testing environment.

Bibliography

[1] Synopsys. www.synopsys.com/automotive/autonomous-driving-levels.html.12th November 2019.

[2] Bruce Brown. Evidence stacks up in favor of self-driving cars in 2016 NHTSA fatality report. Digitaltrends. 2017

[3] Sarah Berger. These are the states with the longest and shortest commutes — how does yours stack up? CNBC. 2018

[4] Steven Waslander, Jonathan Kelly. Self-Driving Cars Specialization. Coursera. July 25th 2019.

[5] Nidhi Kalra, Susan M. Paddock.Driving to Safety. RAND.

[6] Yann LeCun, Leon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-Based Learning Applied to Document Recognition. 1998

[7] Ayoosh Kathuria. How to implement a YOLO (v3) object detector from scratch in PyTorch. Paperspace. 2018.

[8] Pointclouds. Euclidean Cluster Extraction. pointclouds.org/documentation/tutorials/cluster_extraction.php. 5th November 2019.

[9] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In Proceedings of the IEEE International Joint Conference on Neural Networks, pages 1453–1460. 2011

[10] Redmon, Joseph and Farhadi, Ali.YOLOv3: An Incremental Improvement. ArXiv. 2018.

[11] A. Dominguez-Sanchez, M. Cazorla, and S. Orts-Escolano, “A new dataset and performance evaluation of a region-based cnn for urban object detection,” Electronics, vol. 7, iss. 11, 2018

[12] Tutorialspoint. Sobel Operator. www.tutorialspoint.com/dip/sobel_operator.htm. 10th October 2019.

[13] Tutorialspoint. Perspective Transformation. www.tutorialspoint.com/dip/perspective_transformation.htm. 2nd October 2019

[14] Marksandharrison. Causes of Car Accidents: Disobeying Traffic Signals in Virginia. www.marksandharrison.com/accident-attorney/car-accidents/causes-of-car-accidents/disobeying-traffic-signals/ 23rd September 2019.

[15] Kesten, R. and Usman, M. and Houston, J. and Pandya, T. and Nadhamuni, K. and Ferreira, A. and Yuan, M. and Low, B. and Jain, A. and Ondruska, P. and Omari, S. and Shah, S. and Kulkarni, A. and Kazakova, A. and Tao, C. and Platinsky, L. and Jiang, W. and Shet, V. Lyft Level 5 AV Dataset 2019. Level5 Lyft. 2019

[16] Waymo Open Dataset: An autonomous driving dataset. Waymo. 2019

[17] IPG Automotive. ipg-automotive.com. 25th July 2019.

[18] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg. SSD: Single Shot MultiBox Detector. ArXiv. 2015.

[19] Kaiming He, Georgia Gkioxari, Piotr Dollár, Ross Girshick. Mask R-CNN. ArXiv. 2017