Traffic Light Negotiation and Perception-Based Detection

Prepared by:
Meg Wilson

Prepared for:
EEE4022S
Department of Electrical Engineering
University of Cape Town

October 29, 2023

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

October 29, 2023

Name Surname

Date

Acknowledgements

This project has allowed me to expand my knowledge and explore new areas of interest in the field of engineering, empowered me to tackle real-world challenges beyond the University environment. This degree would not have been possible without the love and support from a few key individuals.

Firstly, to my supervisor, Boitumelo Dikoko. Thank you for your guidance and support throughout this project. You allowed me to navigate challenges independently, teaching me the necessary skills to achieve our goal. You emphasized the importance of seeking assistance when things aren't going your way. Your inclusive approach fostered a welcoming atmosphere in our meetings, encouraging mutual learning within our group. I am beyond grateful for the confidence you have given me and the skills I have learnt from working under your guidance.

To my family. Thank you for being my number one supporters and allowing me to spend my final year at home with you. Mom, to your endless patience listening to me complain about my software not working or the usual engineering complaints. You always made sure dinner was ready and dragged me out of my room making sure I took a break. Thank you for editing my work throughout my degree. Dad, thank you for the endless cups of tea and cornered kitchen chats. You always know how to take my mind off my work and have some debates to keep me sane. Moo, Thank you for always keeping me calm in the storm and guiding me through my university life. You always provided me with the calm energy to conquer anything I set my mind on.

To my boyfriend, Connor. Thank you for encouraging me throughout my entire journey and being my rock. Thank you for always believing in me, even when I didn't, and pushing me to achieve goals I have dreamt of.

Lastly, to the friends I've made along the way and have helped me get to the end. This would not have been possible without the constant teamwork.

Abstract

Automated vehicles have become a prominent topic of discussion, especially with the advancement in artificial intelligence and improved technologies. Designing an automated vehicle to navigate an intersection during load shedding conditions will increase road safety in South Africa. This project focuses on implementing perception based detection to navigate an automated vehicle through a South African Environment.

This project used a pre-existing Simulink model where an autonomous vehicle could navigate through an intersection during normal conditions. The primary aim of the project was to adapt this model to include the South African environment of load shedding. This was done by detecting the state of the traffic light using a You Only Look Once detector and detecting the stop line with image processing.

The results revealed a promising vehicle design for the automated vehicle to safely navigate through a South African intersection.

Contents

List of Figures

Abbreviations

ACF Aggregate Channel Features viii, ix, 20–25, 28, 37, 39, 40, 51

CNN convolutional Neural Network 27

CPS Cyber-Physical Systems 9

DSIP Distributed Synchronous Intersection Protocol 5

gTruth Table GroundTruth Table 21, 22, 24, 25, 27, 28, 35, 37, 40, 42, 45

HDS Hardware-Dependant Software 9

I)U Intersection Over Union 27

Lidar Light Detection and Ranging 8, 51

MMW Millimeter Waves 7

NMS Non maximum suppression 28

ROI Region of Interest 7, 21, 22, 24, 45

SA South Africa 3, 4, 16

TCP Transmission Control Protocol 10

UDK Unreal Development Kit 9, 10

YOLO v2 you only look once version 2 viii, ix, 15, 20, 21, 24, 25, 27–29, 32, 40–42, 45, 51, 52

Chapter 1

Introduction

1.1 Background

Autonomous vehicle sector is rapidly increasing and constantly adapting to be implemented in today's fast paced world. South Africa is currently experiencing increased power outages and continues to be a persisting problem for the country. The term load shedding is becoming the norm in South Africa referring to these extended periods with no power. Adapting the autonomous vehicle model to navigate these unique scenarios using perception based detection is becoming increasingly important.

The development of image processing dates back to 1970's, when edge detection and pattern recognition were experimented with, revolutionising image processing techniques. Computer vision began to rapidly grow as technology advanced, increasing computational power. Feature extraction and object recognition were explored in the 1990's, and in the early 2000's, with advanced sensors and cameras, the integration of vision-based systems in autonomous vehicles began. During the early stages of development, lane and pedestrian recognition were created [2].

In the mid 2000's image processing began to infiltrate into early driver-assistance vehicles. These systems focused on safety, providing automatic braking, and cruise control. Between 2000 and 2010 enormous strides were taken to improve computer vision and develop sophisticated algorithms and deep learning techniques. The convolutional neural networks advanced allowing this technique to solve complex tasks and create a breakthrough for autonomous vehicles.

Today we can see these various developments and integration in autonomous vehicles. These advanced systems integrated into autonomous vehicles utilize a combination of sensor fusion, image processing, simultaneous localization and mapping, deep learning based perception models allowing the system to make decisions in real-world driving environments.

1.2 Objectives

The primary aim of this project is to safely navigate an intersection, considering the traffic light state, and surrounding vehicles. The following list outlines the main objectives:

- Understand the existing Simulink model and any existing limitations.
- Design a system to identify the traffic light state using perception based detection.
- Safely navigate the intersection, considering the traffic light state, including load shedding

scenarios, and surrounding vehicles.

- Integrate the detection and navigation sub models into the existing Simulink model, and evaluate their combined performance.

1.3 Scope & Limitations

1.3.1 Project Scope

This project focuses on the research, design and implementation of a module aimed at safely navigating an intersection. It takes into account the traffic light state and surrounding vehicles by using a combination of sensors and perception algorithms.

1.3.2 Project Limitations

This project required working with an existing Simulink model, presenting various challenges due to the unfamiliarity and limitations of the code and model. As a result significant trial and error were involved, alongside the need for patience in comprehending each element of the module and its functionality.

The project scope involved configuring a simulated South African environment, enabling an automated vehicle to detect the traffic light condition, including scenarios involving load shedding, with perception based detection. Due to time constraints the load shedding scenario was simplified and treated as a stop street.

Several limitations were encountered when attempting to implement the South African road environment from Road Runner, as it didn't align with the characteristics of the program. The project was adjusted to incorporate the US City block environment, which was customized to include specific South African features.

Another major setback was attempting to implement the model on an Apple Mac laptop. RoadRunner isn't compatible with iOS, so the project had to be done on a laptop with the Windows operating system.

1.4 Report Outline

The report is divided into seven chapters and begins with reviewing the relevant literature to provide an overall understanding in Chapter 2. Chapter 3 gives the theoretical background for the existing model the project is built from. This provides insight into the functions of the existing model and the modifications to this model. Chapter 4 and 5 discuss the methodology and design choices in producing a perception based system to navigate the South African intersections. Following the design is Chapter 6, which explains how the model was setup to produce the results, and Chapter 7, which evaluates the choices made and their performance. The final chapter 8 reflects on the results obtained and provides future recommendations to enhance the project.

Chapter 2

Literature Review

> Traffic should not always be a problem in your way.
>
> —*Unknown*

Last year, South Africa (SA) experienced more than 12, 000 fatalities resulting from motor vehicle-related accidents [3]. According to a study conducted by the South African Medical Journal, 71% of fatal accidents in SA were a result of driver errors [4]. From this study, it is clear most vehicle-related incidents occur due to human error therefore implementing automated vehicles into SA roads could rapidly improve road safety.

Although the Covid-19 pandemic has shifted the working world with more employees working from home, decreasing road congestion, SA still faces the challenges of reckless drivers and poor road conditions. This literature review focuses specifically on the struggles SA faces in everyday road congestion, and traffic light negotiation and how Automated vehicles can be implemented in this unpredictable road environment. This project only focuses on the simulation environment.

2.1 South African Traffic and Road Conditions

2.1.1 Vehicles and obstacles on South African roads

SA is a third-world country and still feeling the lasting ramifications of Apartheid where the challenges of ensuring efficient public transport persist. Due to this separation, many people live far from where they work and there is a lack of infrastructure in these areas [5]. This gap has made the taxi industry the most used form of public transport in SA. A study conducted by the Automobile Association of South Africa revealed there were approximately 70,000 minibus taxi accidents reported annually [6]. This reveals that minibus taxi crashes amount to more than double that of all other passenger vehicles combined [6]. One of the major issues drivers face is taxi drivers don't adhere to the rules of the road.

Another study was conducted comparing Sweden and South African drivers where 60% of South Africans believed fellow drivers didn't respect or obey traffic laws [7]. Another factor to consider is pedestrians as most pedestrians don't follow the rules of the road, and jaywalking is very common. This can be challenging when trying to incorporate automated vehicles into this uncertain environment.

The biggest obstacle SA is currently facing is load shedding. This makes traffic negotiation a lot more challenging as traffic lights do not work and human intuition comes into play. As mentioned above the rules of the road aren't followed as they should be. There is often a traffic officer who conducts

the traffic when traffic lights are out. This is important to consider when implementing automated vehicles in SA as they don't have human intuition and need to be programmed for these unexpected situations. This is a challenging task as automated cars can only act for conditions they have been programmed to use or figure out meaning load shedding conditions are unknown to most autonomous vehicles and careful planning needs to be implemented for the vehicle to safely cross the intersection.

2.2 Automated Vehicles

Automated Vehicles, also known as self-driving cars, are vehicles able to detect objects in their surroundings and act on these detection's without the need for human assistance. This is becoming more popular and affordable making them now accessible to a larger market. Having more automated vehicles on the roads reduces the risk of driving incidences due to human error creating safer road conditions. Autonomous vehicles need high accuracy and efficiency to ensure they are built to every condition experienced in the real environment. Training these vehicles is one of the most important parts, as the training needs to incorporate a variety of different conditions the vehicle needs to be able to handle. These include different weather and lighting conditions [8].

Automated cars have many different capabilities depending on the model and brand. These have evolved over many years and are becoming more common in multiple car brands. The difference between autonomous vehicles and human driven vehicles lies in the software which will be discussed below. Below is an image of the inside of the Tesla Model S self-driving car Figure 2.1.

Figure 2.1: Inside View of a Tesla Model S Self-Driving vehicle

2.3 Navigation of Automated cars with non-Automated cars

2.3.1 Solutions to Deadlocking

Automated and non-Automated cars need to be able to coexist as there will be a long transition period where the cars co-exist [9]. Deadlocking is a common concern when automating cars. This is when cars come to a standstill and cannot proceed due to the systems in place failing. A-DRIVE was created to mitigate these situations [10]. In this project, we will only be looking at local perception-based A-DRIVE as the vehicles cannot communicate with each other. A-Drive is designed to detect and recover from deadlock by calculating the maximum time the car can wait in a specific state, the threshold waiting time, and the negative impact associated with giving up the current state of the car, yielding cost in a decentralized manner. Once the vehicle's waiting time is greater than its calculated threshold it initiates an action to resolve the deadlock [10]. The A-Drive uses a Local perception-based approach which always checks the movements of surrounding cars by using three elements, vehicle state, localization state, and yielding cost [10]. The vehicle's state constantly changes. A Boolean function represents the localization state, displaying the number 1 when the vehicle enters the intersection range or a shared road segment and a 0 when the vehicle is not within the intersection range. To recover from a deadlock A-DRIVE uses 6 states NOT AROUND INTERSECTION, APPROACH, WAIT, CROSSING, IN DEADLOCK, and YIELDING [10]. The figure below 2.2 shows the decision logic using a Stateflow chart to ensure the vehicles can exit a deadlock situation.

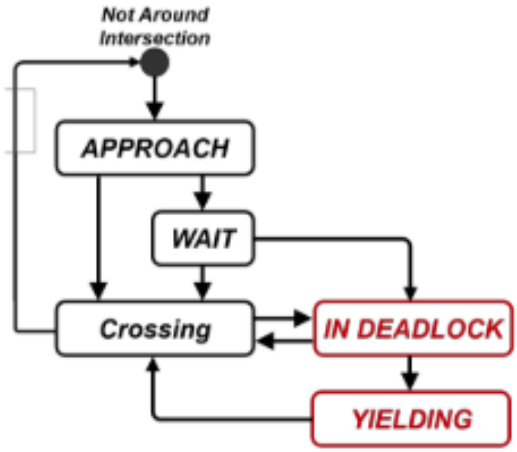


Figure 2.2: Stateflow chart showing the logic to ensure vehicles can escape deadlocking [1]

2.4 Automated and Human Driven cars co-existing

In Aoki and Rajkumar's (2022) paper, the use of Distributed Synchronous Intersection Protocol (DSIP) aims to prevent deadlocking and vehicle collisions while ensuring optimal traffic flow. A key feature in DSIP is the dynamic decision-making mechanism that enables each Connected Autonomous Vehicle (CAV) to intelligently adjust its decision-making policy in response to the surrounding environment. This is achieved by using the vehicle's current state and its operational mode. The vehicle's state is constantly changing and is instrumental in determining the decision of the vehicle at the intersection.

The vehicle also needs to consider the traffic congestion, other vehicles on the road, and the environment. To factor in human-driven vehicles, autonomous cars need to predict future maneuvers considering traffic light state and Synchronous mode. Depending on the mode the vehicle makes the necessary decisions and navigates through the intersection [11].

2.5 Perception-Based Detection

Perception-based detection is a process where sensory data is used to detect objects or events in the vehicle's environment. There are several different types of sensors used for perception-based detection. The three main sensors used for autonomous vehicles are cameras, lidar, and radar. Cameras are small and low cost, radar can detect bad weather conditions, and lidar is excellent at 3D modeling. Models have been created which utilize all three detection methods to ensure accuracy however these methods come at an expense. Figure 2.3 illustrates the Tesla model detecting objects and vehicles using advanced sensors. These three sensory options will be explored below.

Figure 2.3: Perception Based Detection of objects and pedestrians

2.5.1 Detection using Cameras

Cameras are often used to detect the lines of the road which enables the vehicle to stay in the lane [12]. There are two setups used, monocular or binocular cameras. Monocular cameras consist of a single camera where as binocular cameras mimic the human binocular vision by having two cameras side by side. This enables more advanced capabilities like simulating a 3D environment at the level of human perception [13]. Camera technology allows for the identification of traffic light colours, and the distance of objects using natural light. This is also at a higher resolution and lower in cost than lidar. This is only viable with clear weather conditions.

Zhou, et al. [12] used the Vislab Lane detector to detect the lane by gathering multiple sets of coordinates which represent points along the road line. To ensure smooth and steady vehicle trajectory while following the lane, the local path planner requires comprehensive and accurate information about the road lines. An example of lane detection is shown in Figure 2.4

Since cameras are relatively cheap and have advanced software capabilities allowing them to detect both moving and stationary objects. This provides a solution for autonomous vehicles to identify and track lane markings, road signs and traffic lights. Although camera sensory systems allow for a variety of detection's it is still limited to perfect conditions therefore can be utilized with the help of other sensors [14].

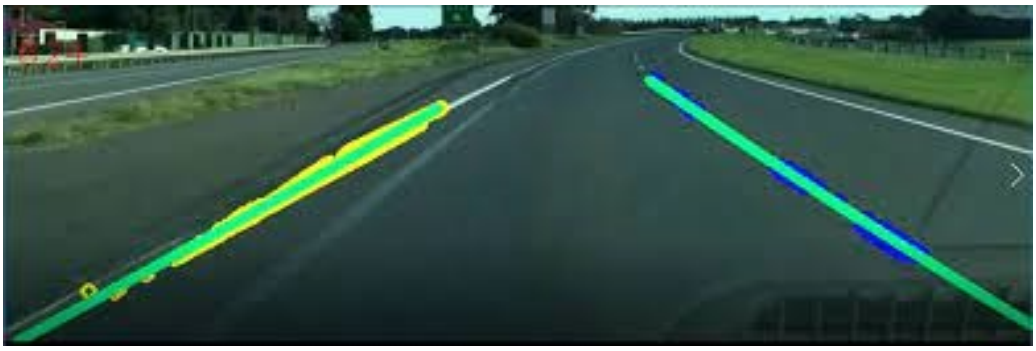


Figure 2.4: Lane detection using a camera positioned on the rear view mirror of the vehicle

2.5.2 Detection using Radar

Radar detection uses Millimeter Waves (MMW) which allows the detection of objects to be accurate to within a millimeter. In the paper by Zhou, et al.(2009) [12], the LMS291 Laser Measurement Systems is used. This sensor uses polar coordinates to determine the objects location which is then converted into a user-friendly form, XY coordinates. By configuring the maximum scanning field to span 180 degrees and setting the resolution to 0.5 degrees, the sensor can produce three hundred and sixty-one measured values.

Han et al.(2016) [1] introduces an integrated method using MMW radar and monocular camera for frontal object detection. This was done by firstly mapping the positions of objects detected by MMW radar onto the image plane, creating corresponding Region of Interest (ROI). Four excellently trained DPM classifiers are used to identify different object classes within these ROI. A mixer module is used to combine the classification results. This model makes use of a probabilistic inference framework to calculate the final detection results. Han, et al.(2016) [1] also used the MMW radar and dynamic Gaussian background mode to create a generating procedure to sample negative outputs which will improve the performance of the system.

Han, et al.(2016) [1] makes use of the Delphi ESR bi-mode radar which is positioned on the front bumper and a PointGray FMUV03MTC colour camera positioned behind the front windshield.

2.5.3 Detection using Lidar

Light Detection and Ranging (Lidar) is used to determine how far away an object is. This is done by sending laser pulses toward an object and calculating the time the refracted light returns. The Lidar data also known as cloud data consists of many X, Y, and Z coordinates to depict the surroundings. This data is difficult to process in real-time as it is large. Anand, et al (2020) [15] proposed a 6-step robust framework for selecting the area of interest and using the point cloud to identify the vehicle. This framework can identify and calculate the number of vehicles in the selected frame [16].

Lidar has limitations when detecting the surrounding environment. The cloud data is unable to categorize the detected objects making it difficult to identify the type of object, humans, and buildings. Lidar is also expensive to implement on large scales therefore many companies use lidars for specific functions, or implement solid-state lidar [13].

2.5.4 Sensor Fusion

After evaluating these three methods of detection and tracking, it can be concluded that sensor fusion is the most practical solution for accurately tracking objects and obstacles in the autonomous vehicles surroundings. Figure 2.5 demonstrates the optimal positioning of the three sensors , providing the best detection. Figure 2.6 shows the physical placement of each sensor, enabling the maximum range of detection [16].

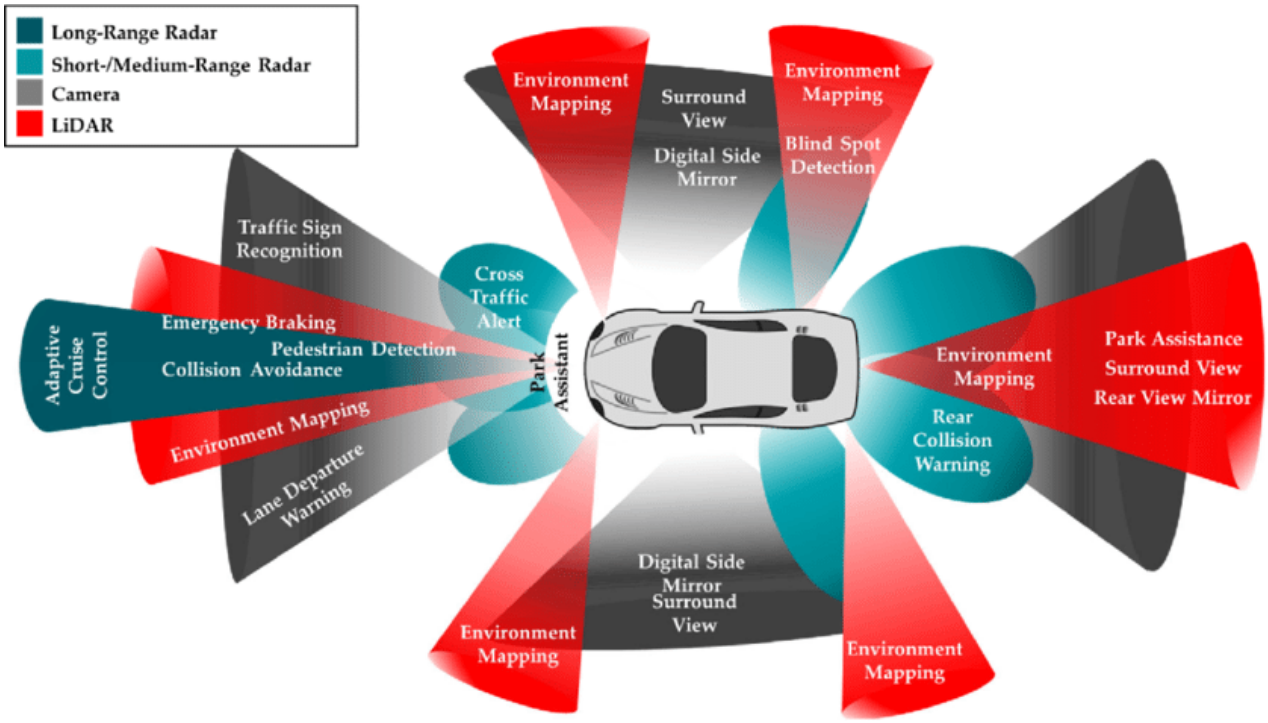


Figure 2.5: Optimal sensor placement on autonomous vehicles to achieve 360 degree environment perception.

Figure 2.6: Sensor placement on autonomous vehicle to enhance detection's

2.6 Modeling the system using Unreal Engine and Simulink

2.6.1 Why use Simulink?

Simulink is a software tool used by engineers for simulation and graphical modeling of Cyber-Physical Systems (CPS) [17]. Subsystem blocks are predefined and are connected using virtual wires, and signals, to create a model. These blocks vary from complicated algorithms such as Kalman filters to basic model elements such as constant blocks. It is essential to understand that the block models in Simulink are fully executable entities meaning when inputs are applied to these blocks the necessary calculations or algorithms are performed producing actual outputs [17]. Engineers can use the 'abstraction' feature to manage large complex cyber-physical systems models. This is done by nesting subsystem models into user-defined blocks, which can represent more significant functionalities. This allows engineers to simplify the overall model by encapsulating complex subsystems into reusable components [17]. There is a wide range of toolboxes in Simulink which appeals to Model-based design because there is no need in developing basic domain elements for example the predefined PID controller. Engineers also have the flexibility to develop a customized block or function using MATLAB language or C. Simulink can synthesize Hardware Description Language or C-Language Hardware-Dependant Software (HDS) implementations from the model. This feature enables the translation of the model into executable code that can be deployed on hardware platforms or used for hardware-in-the-loop simulations. This is useful when no predefined block exists.

2.6.2 Interfacing Simulink with Unreal engine

Unreal Engine is a comprehensive integrated software package developed by Epic Games which is used to create real-time interactive 3D simulation making it popular for game developers. The built-in networking support of Unreal Development Kit (UDK) allows seamless interaction between multiple players within the virtual world. UDK offers a user-friendly method of engaging with multiple Simulink models [17]. The advanced features of UDK have been used across a diverse range of applications including first-person games and even for non-entertainment, military.

Haley, et al.(2012) [17] developed a toolkit to utilize the strengths of both Simulink and UDK virtual worlds allowing two-way interactions between these two platforms. This is done by implementing a Transmission Control Protocol (TCP) connection to allow two-way communication. The program has the capability of expanding by adding multiple Simulink models demonstrated in Figure 2.7. This combination allows Simulink to be used to model high-level algorithms, physics, and hardware that dictate the actions of actors (objects) within the virtual environment, UDK makes use of its GPU processing capabilities to dynamically change the position of UDK objects, identify collisions between actors, and implement suitable lighting.

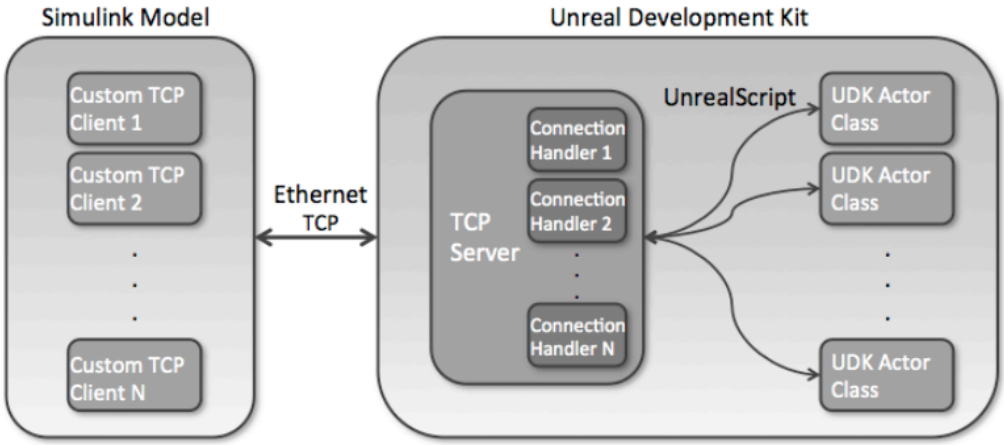


Figure 2.7: Block Diagram of Simulink and Unreal Development Kit toolkit

From the figure above 2.7, the Simulink model establishes a TCP socket connection allowing more than one Simulink model to interact with the virtual world, multiplayer game [17]. The UDK can manage the virtual world by performing fundamental tasks [17]. The UDK only informs Simulink when critical events happen allowing Simulink to only have to implement its computations and simulations.

2.7 Simulating the Road Environment

Matlab Driving Scenario Designer tool can be used to create the environment the vehicle will be tested in [18]. To create the traffic intersection Matlab has a built-in feature, RoadRunner [**?**]. It is an interactive tool where users can design and construct virtual environments enabling evaluation and assessment of the performance of automated vehicles.

Figure 2.8: RoadRunner software used to create unique road for specific scenarios

2.8 Interacting with the Environment

Stateflow charts are a graphical programming language used in Simulink to model dynamic systems. It consists of states, transitions, events, actions, and conditions where decision logic can be easily understood and executed. This is where the developer will design a stateflow chart for the vehicle to execute ensuring no collisions. Matlab allows Stateflow charts to be embedded in Simulink models allowing a smooth transition between the two [19].

2.9 Literature Critique

This literature review has shown the challenges faced by drivers in South African road environments. This includes reckless driving behavior, load shedding, and poor road infrastructure. By integrating autonomous vehicles into this environment it provides a promising solution to enhance road safety and efficiency. However, the mixed environment of automated and human-driven vehicles, as well as navigating challenging intersections, remains a complex task [11].

Due to improvements in perception-based detection, there has been an improvement in autonomous vehicles' ability to navigate these challenges. The three sensors explored each have unique advantages, Cameras are cost-effective, radar can accurately detect in all weather conditions, and lidar offers precise 3D modeling. These sensors can be used together to form the foundation for intelligent and safe autonomous navigation [16].

The integration of Simulink and Unreal Engine creates a robust framework for simulating and testing autonomous vehicles in multiple scenarios. The challenges of mixed traffic environments, deadlocking prevention, and maintaining traffic flow are critical aspects that have been explored through different approaches. This report aims to investigate a solution that integrates some of the aspects researched above, to develop an autonomous vehicle that can navigate through the South African Road environment.

Chapter 3

Theoretical Background

3.1 Simulink Model Setup

This project uses the prebuilt Simulink model, 'Traffic Light Negotiation with Unreal Engine Visualization'. This has been adapted to a South African environment where the vehicles drive on the left hand side of the road, and are able to use perception based detection to navigate through the traffic intersection. It consists of four subsystems specifically designed to simulate a vehicle's navigation through a traffic intersection. These subsystems include Sensors and Environment, Traffic Light Decision Logic, Lane Following Controller and Vehicle Dynamics.

Some terms to be familiar with are:

- Ego vehicle: Automated car present in the scenario.
- groundTruth table: An object in Matlab that holds data source information, labels and marked label annotations.
- Unreal Engine: A software program used by Simulink to visualise the scene and scenario in a 3D environment.
- Load Shedding: Strategy used by power supply companies to restrict electricity supply during high demands. This typically involves scheduled power outages lasting two to four hours, during which the affected area experiences no power from the supplier [20].

3.1.1 Overview of MATLAB setup

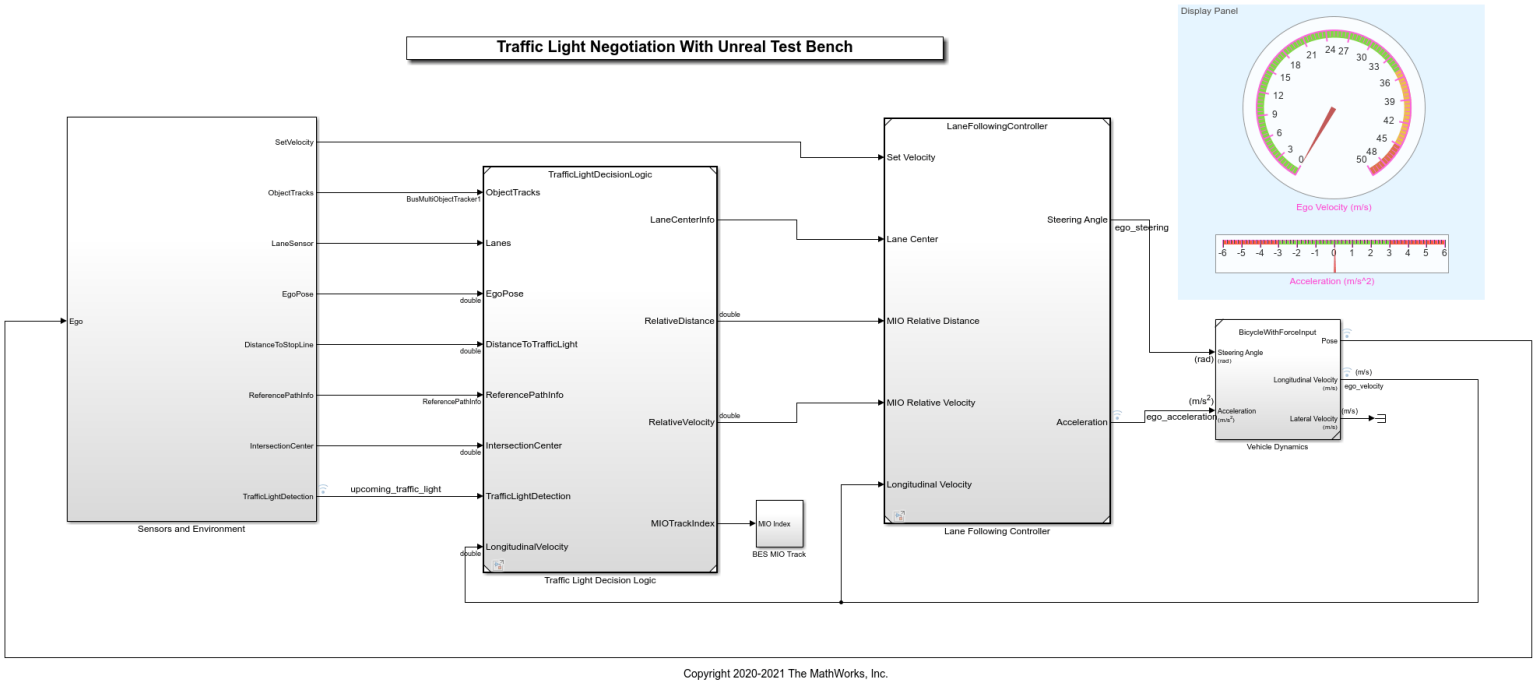


Figure 3.1: Overview of Simulink Model, Traffic Light Negotiation with Unreal Engine Visualization

Refer to Appendix A.1 to view a larger image.

The designing and implementation of the model was done using MATLAB toolbox add-ons: Automated Driving Toolbox, Model Predictive Control Toolbox, Deep Learning Toolbox, Computer Vision Toolbox, Image Processing Toolbox, Parallel Computing Toolbox, Statistics and Machine Learning Toolbox, Matlab Coder and Simulink.

3.1.2 Sensors and Environment Overview

This subsystem models all the sensors used for simulation, including the vehicles, road network, camera,and radar sensors.

The 3D Scene is configured to a prebuilt 3D simulation environment in Unreal Engine, US City Block and for this project only one traffic light intersection is used for testing and validating. New scenes can be built using RoadRunner or the current scenarios can be edited using the Driving Scenario Designer App. The road network is read from the Scenario Reader block which takes in the ego vehicle as input and performs closed loop simulation. It outputs a groudTruth table of lanes boundaries and vehicles in vehicle coordinates. The Driving Radar Data Generator and Vision Detection Generator blocks process this groundTruth data which is then fused and tracked using a Multi-Object Tracker block to detect objects around the vehicle. The Vision Detection also detects vehicles driving in the ego vehicles lane. The Driving Scenario information is then simulated in 3D using the Unreal Engine interface. The ego Vehicle also uses the groundTruth to access the traffic light ID and obtain the coordinates of the traffic light. This tells the ego vehicle where to stop at the intersection.

The Simulation 3D traffic Light Controller helper block controls the traffic light state in an Unreal

Scene.

The traffic light state is determined by the Traffic Light Switching Logic state flow chart which uses the vehicle's location to activate the state change.

3.1.3 Traffic Light Decision Logic and Lane Following Controller Overview

This subsystem determines if there is a lead vehicle within the same lane as the ego vehicle and calculates the relative distance and velocity between these two vehicles. This subsystem also makes use of the ego vehicles reference path if no lanes are detected at an intersection. The Lane Following Controller subsystem makes use of an Model Predictive Control and watchdog controller subsystems to ensure the vehicle stays within the lane. This subsystem outputs the steering angle and acceleration needed for the Vehicle Dynamics subsystem. This subsystem can be seen in Figure 3.2.

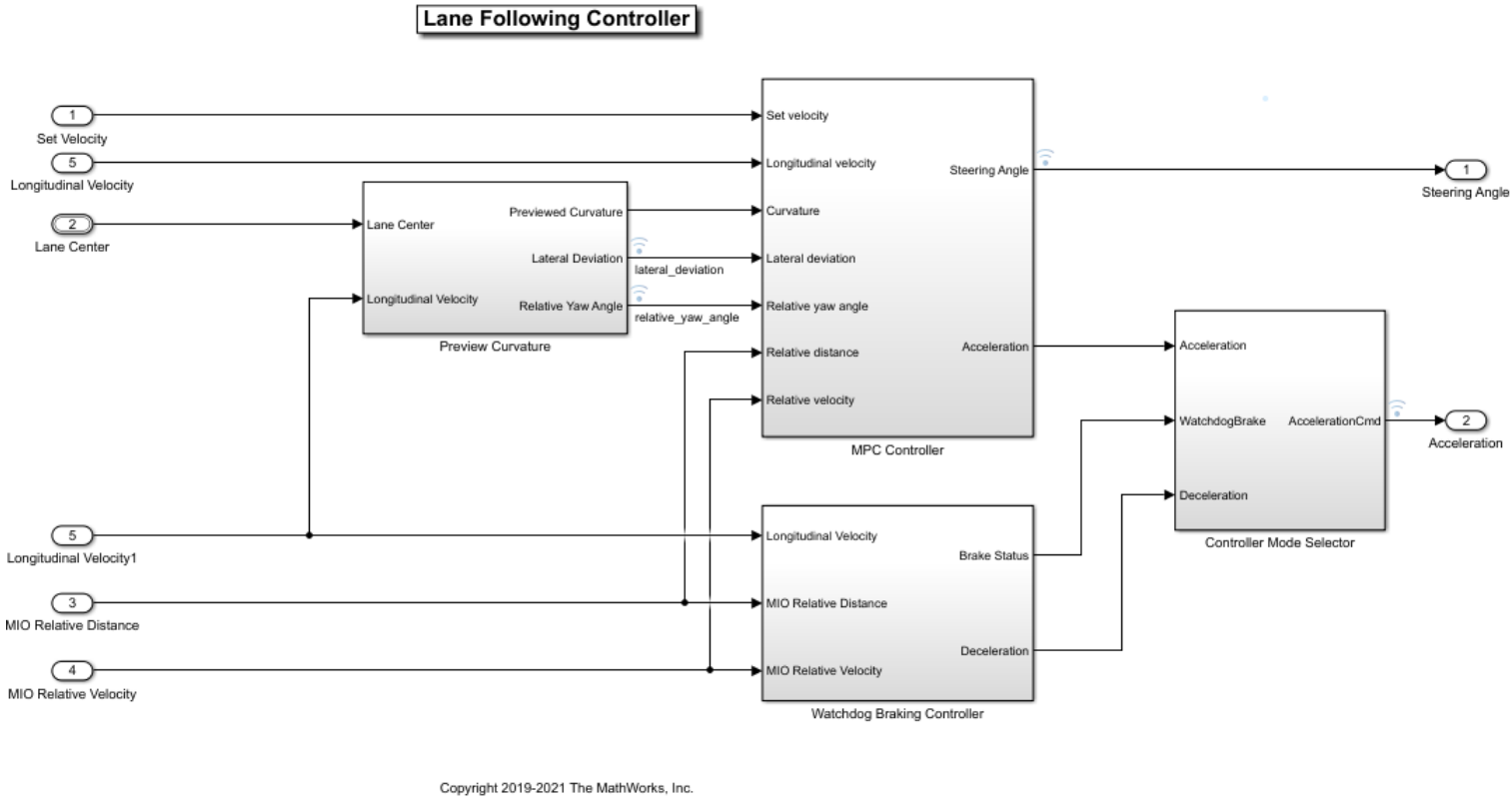


Figure 3.2: Simulink subsystem Lane Following Controller

Refer to Appendix A.2 to view a larger image.

This subsystem is where the braking force is determined and passed to the vehicle dynamics subsystem. This is a critical aspect of the model as the correct braking force needs to be applied for the vehicle to stop at the traffic light and brake when there are obstacles in its trajectory.

3.1.4 Vehicle Dynamics Overview

The Vehicle Dynamics subsystem uses the steering angle and acceleration as inputs from Lane Following subsystem and makes use of Simulinks Bicycle Model to compute the vehicle.

3.1.5 Modifications to the Original Model

The traffic light state flow makes use of the ego vehicle position and changes the state of the traffic light from green to red when the vehicle is closer than 10m. This was modified to a time-based cycle and each scenario has a unique time for each state to be on for and is explored in Chapter 7. The simulation time was reduced to 10 minutes as Simulink and Unreal Engine are resource heavy applications and running simulations for extended periods of time causes the system to significantly lag.

The vehicle's start position is now set at 40 meters away from the traffic light stop line to reduce simulation time.

This project focuses on the implementation of perception based detection using camera detectors mounted onto the roof center of the ego vehicle. This position was chosen as the traffic light is full in view of the camera and the camera has a greater distance to detect the stop line. The vehicle uses a trained YOLO v2 detector to detect the traffic light state and applies the logic for the given detected state. The model originally used predefined logic that the ego vehicle interpreted. To calculate the distance to the traffic light, the camera frames are used to identify the stop line, and calculate a distance from this line detection.

The radar sensor also only had an small angle which was adjusted and moved from the front bumper to the rear view mirror spanning a larger area for defections.

These modifications will be explained in detail throughout this report.

3.1.6 Github Code

All Simulink models and code used for this project can be found at this link: GitHub Repository

3.2 Environment

3.2.1 Creating an Environment

Unreal Engine is a real-time 3D creation tool used to visualize scenarios. The MATLAB Traffic Light Negotiation with Unreal Engine Visualization Simulink model uses the prebuilt Unreal Engine scene, US city block. This block contains 15 intersections and 30 traffic lights. For this project we will only be focusing on one intersection and adapt the scene to incorporate a four traffic light stage representing load shedding conditions. This is when all traffic lights are off and the interaction becomes a four way stop.

3.2.2 US City Block in Unreal Engine

In order to train the detector we need to simulate all conditions with multiple data sets in order to acquire enough data to accurately train the detector.

Figure 3.3 shows the Unreal Engine pre-built scene. The scene was edited to enable the load shedding state, no traffic lights on.

Figure 3.3: US city block layout showing 15 intersections

3.2.3 Designing for No State

For this project the main objective was to simulate an environment resembling load shedding conditions in SA. This was done inside the Unreal Engine application and then implemented in the Simulink model. The traffic light colours were altered in Unreal Engine to display no colour on the light. This was then used in Simulink to train the detectors for a load shedding scenario. A summary of the steps taken are shown in Figure 3.4

The load shedding traffic light scenario can be seen in Figure 3.5. The traffic light displays no colours. This picture is taken from the camera on the ego vehicle's rear-view mirror.

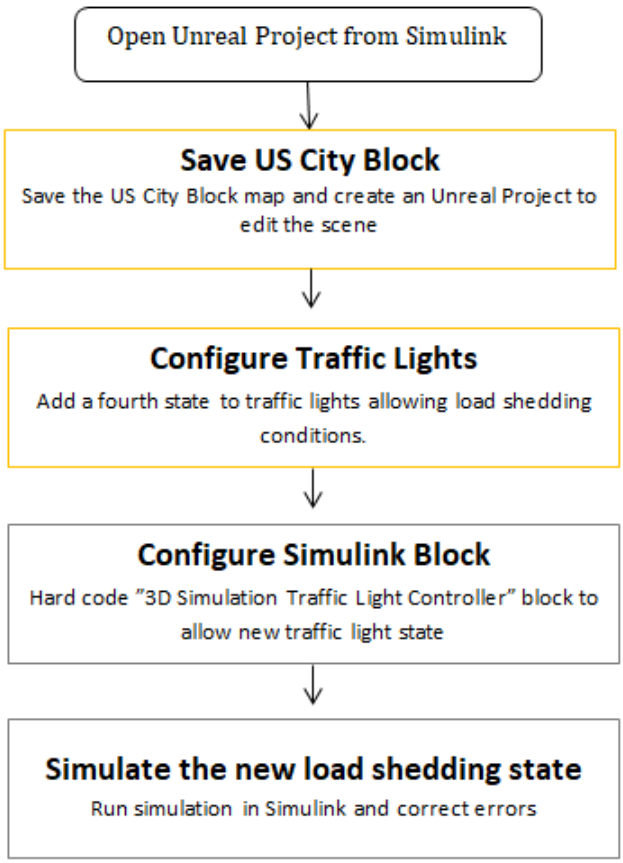


Figure 3.4: Flow chart showing the steps taken to create the load shedding scenario in Unreal Engine

Figure 3.5: Load shedding condition where traffic light is off, No state condition

3.3 Tracking and Detecting Objects

The Simulink model includes radar and camera data generator blocks. This block reads the data from the scenario reader block and outputs the actors and objects detected from radar and camera vision. These detection's are combined using the Multi Object Tracker block to track the objects around the vehicle. This information is then passed to the braking system, which applies the necessary acceleration or deceleration for the given information.

Figure 3.6 shows the Scenario reader block connected to the Camera and Radar Data Generator blocks as input. The output from these two blocks is then used in the Sensor Fusion subsystem shown in Figure 3.7.

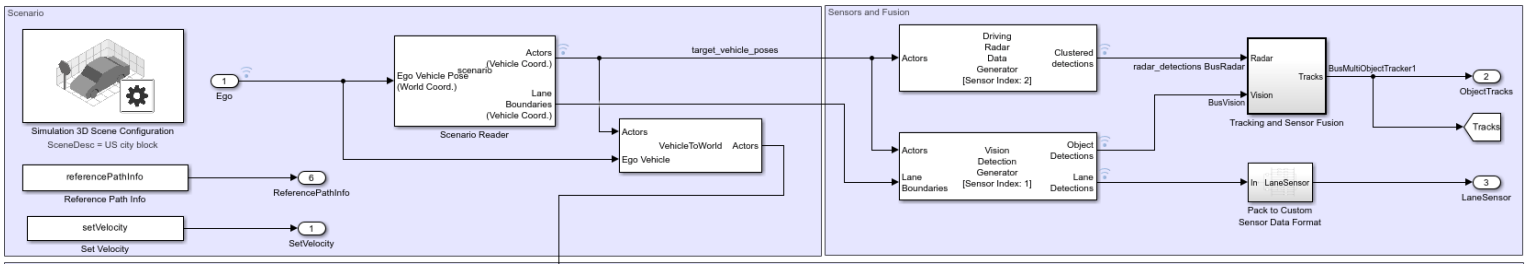


Figure 3.6: Simulink Subsystem showing Radar and Camera Sensors

Refer to A.3 to view a larger image.

Figure 3.7 shows the radar and camera data combining to detect and track object's and obstacles.

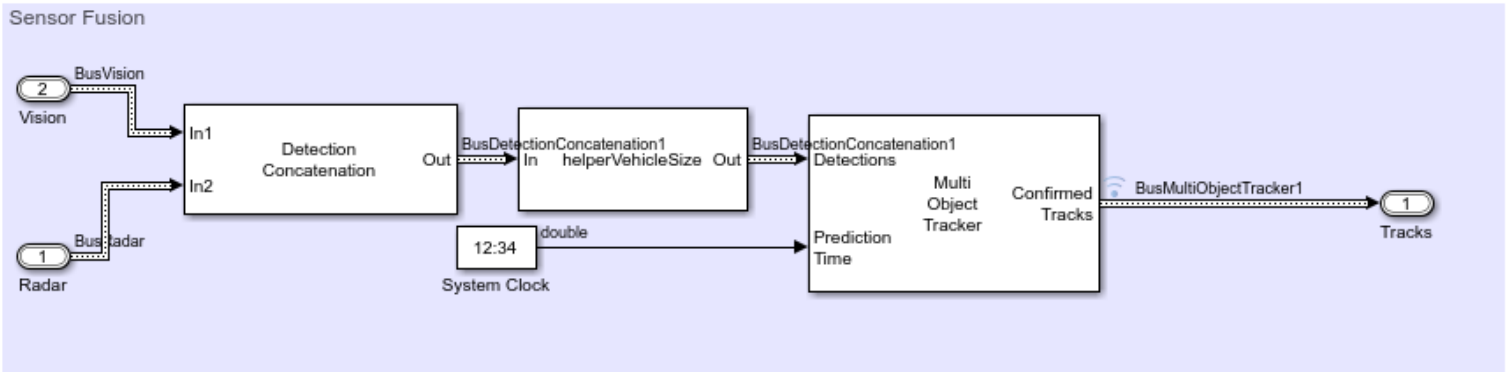


Figure 3.7: Simulink Subsystem showing Fusing of Camera and Radar Data

Figure 3.8 shows a birds eye-view of the radar and camera detection coverage. The radar coverage is shown in red and the camera coverage is shown in blue. This module only has one radar sensor and one camera sensor for detecting the surroundings.

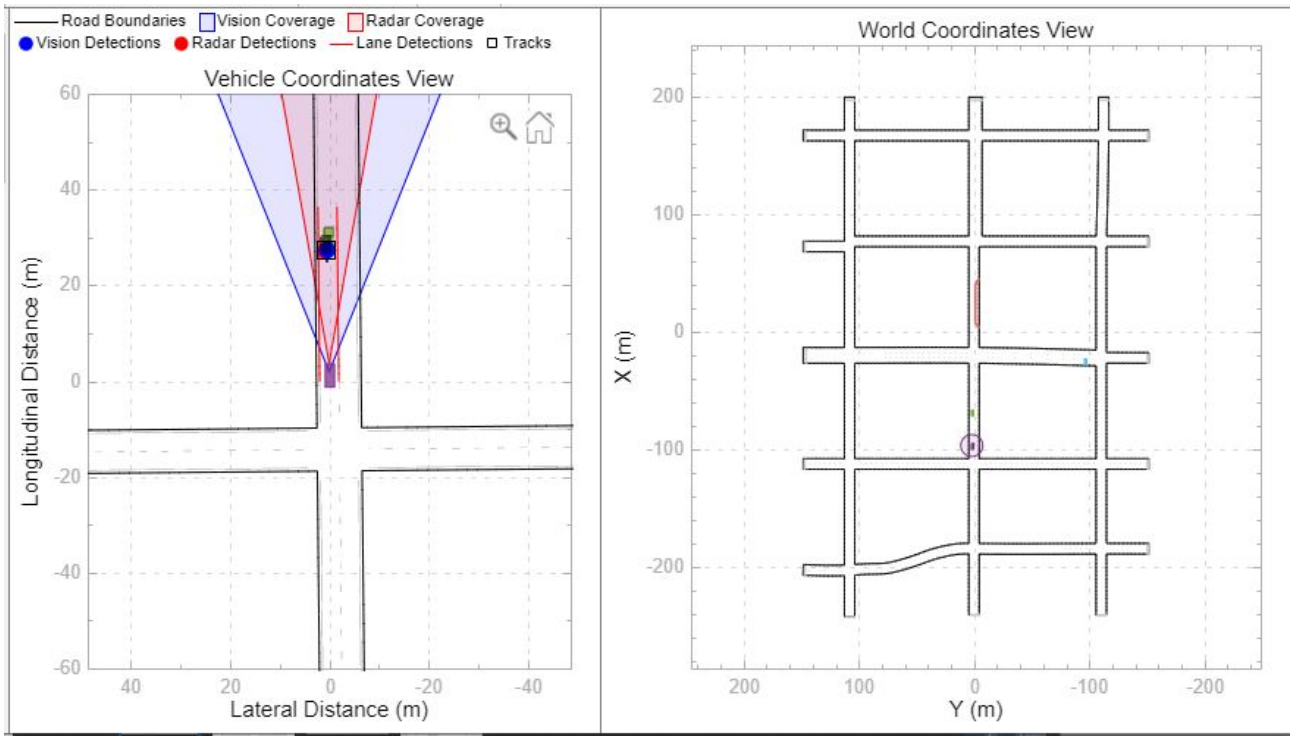


Figure 3.8: Bird eye-view of Radar and Camera range of detection

Chapter 4

Perception Based Detection

4.1 Overview

Identifying objects within images or videos is a simple task for humans but can become exponentially complex for computer applications. Object detection is a method commonly used by Computer Vision to pinpoint an accurate location of the specific object being detected. These algorithms use deep learning and machine learning to replicate this [21]. This project explored and evaluated two methods of detection training. These include ACF object detection and YOLO v2 object detection. Both detection methods follow a similar training procedure which will be explored in this chapter.

4.2 Method

Gathering sufficient data to train the ACF and YOLO v2 detectors followed the same procedure. However, both detectors had a unique way of training, which will be discussed in this chapter.

Figure 4.1 shows the general flow diagram of the process used to train,test and validate ACF and YOLO v2 detector.

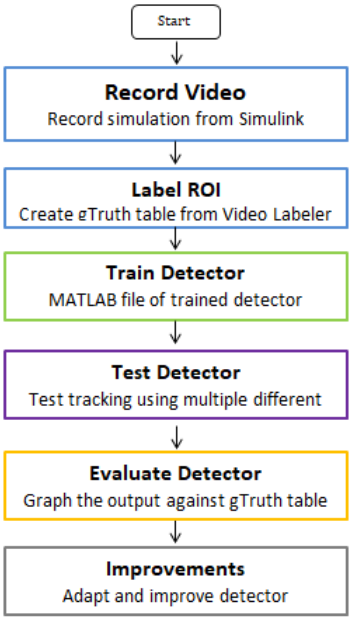


Figure 4.1: Flow Diagram showing the general procedure for creating an ACF and YOLO v2 detector in MATLAB

For this project, the traffic lights will have four states Green, Yellow, Red, and No State. Typically, the green and red states are on for thirty to sixty seconds. However, for the purpose of this simulation, the traffic lights will initially be set to ten seconds for green and red, and yellow for two seconds. These times will be adjusted and modified to understand the data required for the detector. Additionally, a separate scenario will be simulated where all traffic lights are off, replicating conditions during load shedding.

4.3 Region of Interest Video Labeler

The Video Labeler App is an add-on that comes with MATLAB's Image Processing and Computer Vision Toolboxes. It provides a user-friendly method to isolate regions of interest in videos or images and generate a GroundTruth Table (gTruth Table), which is used to train the detectors. The exported gTruth Table object from the Video Labeler app consists of three arrays: 'LabelData' represents the bounding boxes for the ROI, 'DataDefinitions' contains the names associated with the labeled data, and 'DataSource' includes the images or videos used to create ROI. The format of the gTruth Table object takes the general form gTruth = groundTruth(DataSource,DataDefintions,LabelData).

To capture the video data required for training the detectors, a simulation video from the Simulink model was recorded using a camera positioned on the rearview mirror of the ego vehicles.The Simulink to Multimedia block was used to capture the camera's view during the simulation, which served as the dataSource for the gTruth Table. This video was then used to manually label the ROI for the ACF detector and the traffic light states, Green, Red, Yellow and NoState for the YOLO v2 detector. A point algorithm was used to annotate the labels across all frames of the video, representing the labeldefs parameter in the gTruth Table object. Each state in the video was labeled in a frame and annotated to create labels for instances where there is a traffic light in the video. These ROI were used to create the gTruth Table, which is then saved and exported to the MATLAB workspace. This is the gTruth Table used to train the detector.

Figure 4.2: Labeling traffic light in Video Labeler

This image shows a frame of a video being labeled creating the traffic light ROI in Video Labeler. This is then used to create the gTruth Table.

The challenges involved recording a video with multiple instances of different states. The traffic light states were adjusted in the Simulink model to remain active for extended periods of time, allowing more data to be recorded for each state. This step was crucial for the training process, as the detector requires sufficient data to ensure accuracy and reliability. To gather significant training data for the detector, the states were maintained throughout the entire simulation, and the videos corresponding to each state were concatenated to create one single, extended training video. This approach enabled training the detector with various angles and distances at which the traffic lights could appear.

The starting position of the ego vehicle was adjusted to capture videos from a greater distance away from the traffic light, providing the detector with a more diverse dataset during training. Multiple different videos were saved to optimize the performance of the detector.

4.4 Aggregate Channel Features (ACF) Detector

4.4.1 Overview

To design the ACF detector, the gTruth Table created from the Video Laberler app was implemented in MATLAB using the *trainacfObjectDetector* object. This is a feature in the Computer Vision Toolbox™ allows the training of a detector to accurately identify objects in a video or image. The trained detector together with machine learning and deep learning can accurately recognize the objects quickly. This was the initial type of object detection explored to identify the traffic light.

4.4.2 Aggregate Channel Features (ACF) Object Detector Training

All the training was done using MATLAB code and MATLAB built in features. All code used for training ACF can be found at this link GitHub Repository.

Figure 4.3 illustrates the complete ACF detector process, which includes data capturing, training, testing, and validating.

The ACF detector is capable of detecting only a single class object. It was trained to identify the presence of a traffic light in any state. The steps are shown in Figure 4.3 and are as follows: As explained in Video Labeler section 4.3, the gTruth Table table is created. To train the ACF detector, the traffic light states don't influence the data, as the detector identifies the traffic light structure rather than the state.

From this gTruth Table dataset, only images with the traffic light label are extracted, and a new gTruth Table is created for training data, along with a folder to save the training data images into. Using MATLAB's *objectDetectorTrainingData* function, a table is created containing the training data. This is done to streamline the training data by organising and formatting the dataset, making it easier to train the detector with [21].

Creating training data is one of the most important steps, as the accuracy of the detector highly depends on the availability of a sufficient amount of data. This training data is then used in MATLAB's

trainACFObjectDetector function, and the trained detector is saved.

This detection algorithm uses a binary classifier that assigns the data into two categories. Images containing the traffic light are labeled 1, while images not containing the traffic light are labeled 0. These labels are used to learn the patterns necessary for identifying the traffic light.

The algorithm establishes a decision boundary that distinguishes the two classes and minimizes identification errors. Once this decision boundary is established, the algorithm can recognise classes in unseen data and make accurate predictions. This stage marks the completion of the trained ACF detector.

This training process was repeated three times. Three videos were recorded where the ego vehicle had different starting locations. The detector was trained with each of these different videos and the results were interpreted.

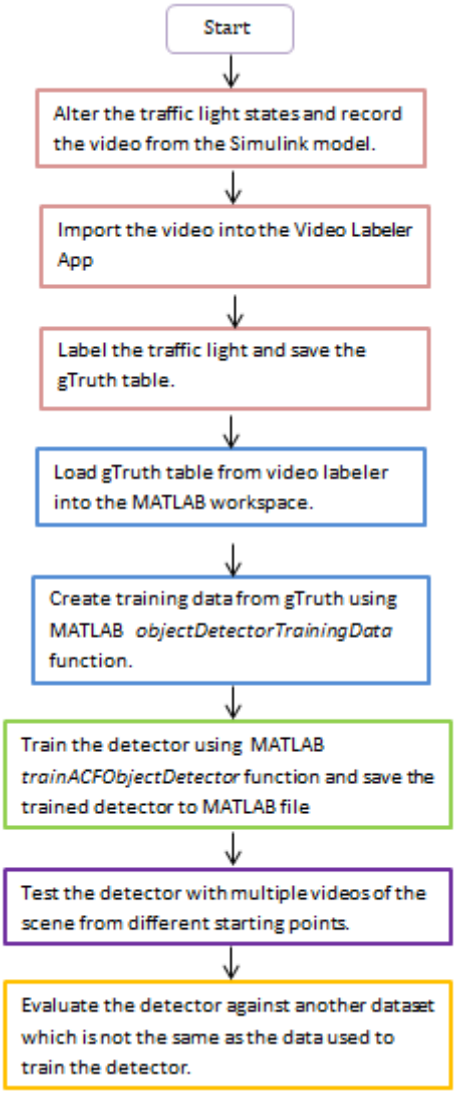


Figure 4.3: Flow Diagram showing the procedure for training, testing and evaluating the ACF detector in MATLAB

4.4.3 Aggregate Channel Features (ACF) Detector Testing

Once the detector is trained it is tested in various scenarios, including situations where the vehicle is at different distances from the traffic light and the traffic light is changed more frequently. To visualise the results, the bounding boxes and labels were annotated onto the video frames and displayed on the video. The detector's outcomes were compared to the gTruth Table, which contains accurate detection data. This can be seen in the results chapter 7.

4.5 You Only Look Once Version 2 (YOLOv2) Detector

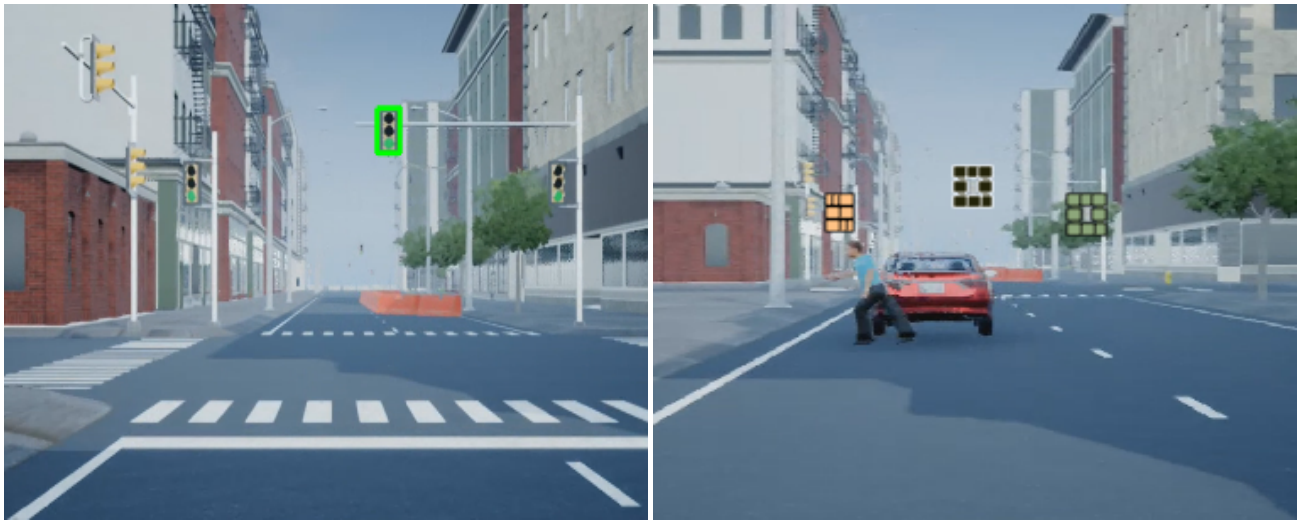
4.5.1 Overview

You-only-look-once object detector, YOLO v2, employs a single stage network for object detection which enables it to achieve higher speeds compared to two-stage deep learning object detector, ACF detectors.This is done by processing an input image using deep learning to establish predictions and create bounding boxes. These bounding boxes can be seen in Figure 4.4 . Deep learning neural networks are used to rapidly and accurately predict the position and dimension of an object in an image. Incorporating anchor boxes improves the reliability and efficiency during detection. YOLO v2 identifies object classes using anchor boxes [22]. Figure 4.6 shows the complex YOLO v2 detector architecture.

Two different YOLO v2 detectors were trained and tested to achieve the most reliable output. These include training the detector to:

- Identify the middle traffic light state
- Identify the three physical traffic lights and the current state of each traffic light.

The ROI can be seen for both detectors in Figure 4.4



(a) Bounding box for middle traffic light detector (b) Bounding box for all three traffic lights detector

Figure 4.4: Bounding boxes for the two trained YOLO v2 detectors

4.5.2 You Only Look Once Version 2 (YOLOv2) Detector Designing and Training

All the designing and training was done using MATLAB code and MATLAB built in features. All code used for training ACF can be found at this link GitHub Repository.

YOLO v2 detector is capable of detecting multiple object classes, enabling it to identify each state of the traffic light, including Green, Yellow, Red and No State. The training process involved teaching the detector to recognize each of the four states, as depicted in Figure 4.5. The following sections provide more detailed discussion of the steps taken to train the detector. Initially, the YOLO v2 detector was trained to only identify the middle traffic light state. Subsequently, the same steps were repeated, utilizing a larger gTruth Table table to train a detector capable of identifying all three physical traffic lights.

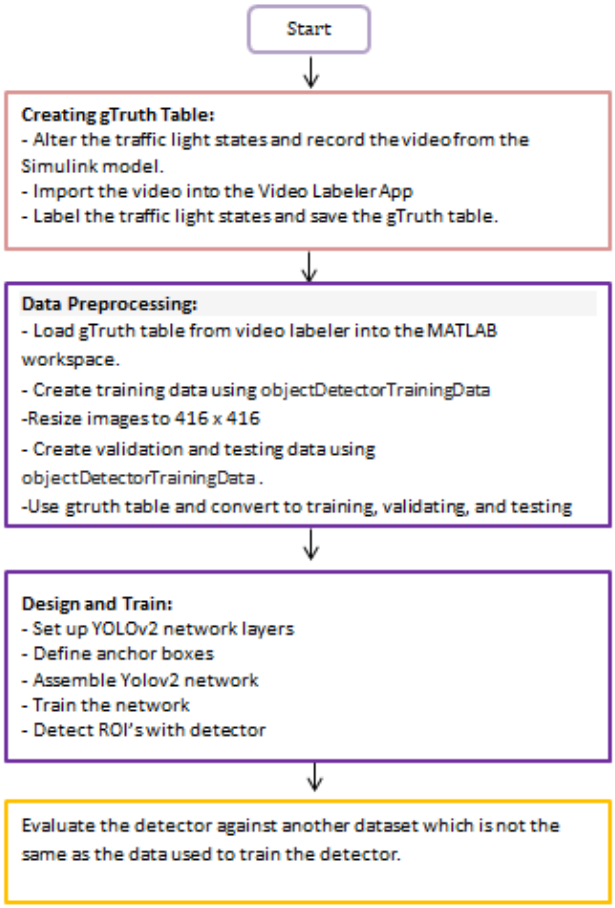


Figure 4.5: Flow Diagram showing the procedure for designing, training, testing and evaluating the YOLO v2 detector in MATLAB

4.5.3 Design and Training

The design process involves the creation of various custom network layers, all working together to interpret the input image and calculate the probability of bounding boxes and class detection. Combined with the loss function and anchor box method this creates a well rounded accurate detector.

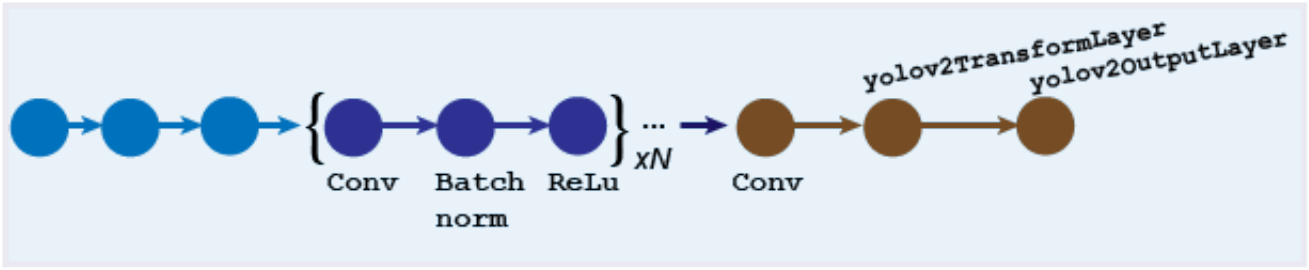


Figure 4.6: YOLOv2 architecture

The figure above illustrates the YOLOv2 architecture, which is built layer by layer to create a unique detector specific to this project:

The design phase includes several key elements:

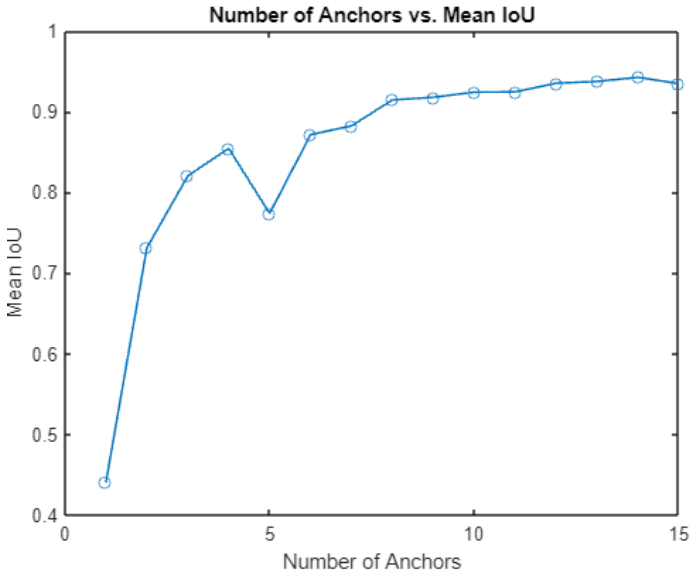
- Input Layer: The input layer is sectioned into cells, where each cell is accountable for identifying an object within its own cell.
- Convolutional Layers: Filters are used to identify patterns and characteristics at various spatial scales in order to extract important data from the input images.
- Batch Normalisation Layers: Incorporated to speedup and create a stable training process.
- Rectified Linear Unit (ReLU) layers: This layer handles non-linear relationships within the data, which is crucial, especially in datasets containing complex relationships. It enables the network to learn and capture these intricate patterns effectively.
- Max Pooling Layer: The outputs from the convolutional layers each highlight different important aspects which are referred to as feature maps. These maps are then reduced by decreasing the spatial dimensions. This enables the network to concentrate on the essential information while minimizing computational load.
- Anchors boxes: Allow detection at the same time giving the detector the ability to detect all the objects in the frame simultaneously.

The process for creating the layers is as follows:

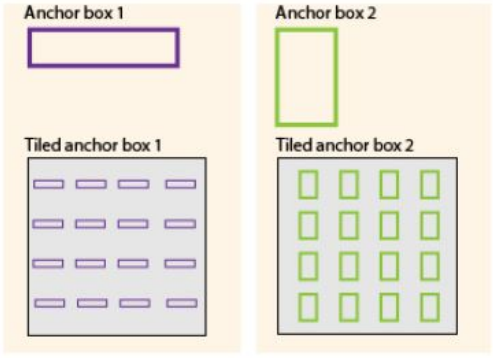
The initial layer in the neural network is the input layer, which receives the initial data and forwards it to the appropriate layers for processing. The next layer is the middle layer which consists of Convolution2dLayer,Batch Normalization Layer,RelU Layer and Max Pooling Layer, following the YOLO900 papers approach [23]. The network progressively learns by increasing the number of channels after every pooling stage. As the data passes through deeper layers, the network learns more complex patterns, enhancing its ability to differentiate between classes. Incorporating batch normalization during training enhances the stability and speed of convergence. It achieves this by establishing a standard mean and variance for input images, simplifying the optimization process. The input and middle layers and then combined to form a layers graph object.

Anchor Boxes, as depicted in Figure 4.7b below, are rectangular bounding boxes that play a critical role in training the detector. They reduce processing time and improve efficiency by aiding in object

detection simultaneously. Each dataset has a unique set of anchor boxes, with *estimateAnchorBoxes* used to calculate the anchor boxes for each dataset and detector. Figure 4.7a displays a plot of the Intersection Over Union (I)U) compared to the number of anchor boxes, with the I)U displaying the degree of overlap between the boundary boxes and the gTruth Table, within the range 0 to 1.



(a) Graph showing the Number of Anchor Boxes vs Mean IOU



(b) Anchor Boxes

Figure 4.7b displays two rectangular anchor boxes, one orientated vertically and the other horizontally. The tiling of these boxes across the image to detect objects is clearly depicted. Although gaps exist between the boxes, which may potentially lead to localization errors, YOLO v2 detectors are designed to account for this error.

The graph below illustrates the influence of the number of anchors on mean I)U.

The number of anchors required is determined by ensuring that the mean I)U is above 0.65. As depicted in Figure 4.7a, the I)U exceeds 0.65 after 1.8 anchors boxes, but begins to plateau from 8 anchors boxes onwards. For the YOLO v2 detector. Four anchor boxes were used for this project's detectors.

The placement of the anchor boxes is calculated by translating the networks,convolutional Neural Network (CNN), output back to the input image. This process is carried out for all network outputs, creating anchor boxes across the entire image, as shown in Figure 4.8. The number of network output corresponds to the number of tiled anchor boxes. These tiled anchor boxes have gaps between them, and the distance is a function of the CNN downsampling, which can produce localization errors, Figure 4.7b. YOLO v2 detector has the ability to predict the offset and incorporate this into the tiled anchor boxes location and size.

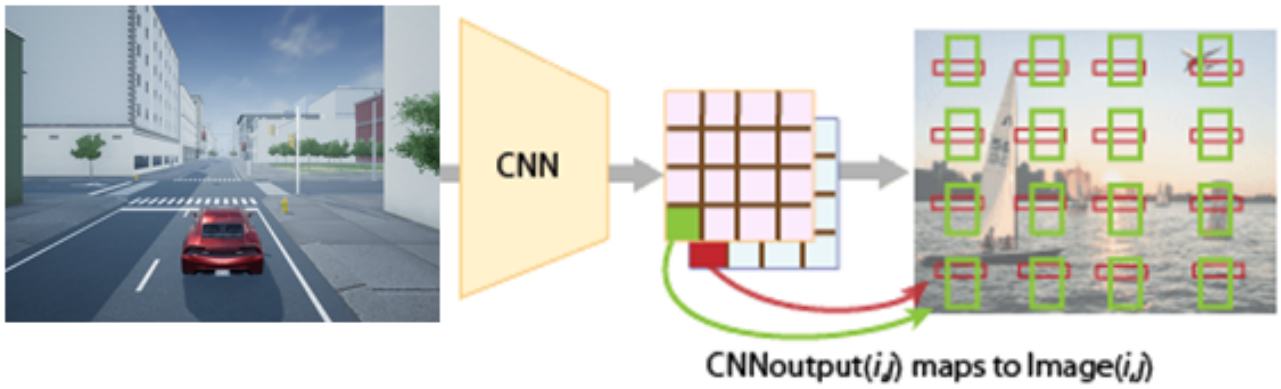


Figure 4.8: Anchor box used in YOLO v2 detection

Finally, anchor boxes associated with the background class are discarded, while the other boxes are sorted according to their confidence score. The highest confidence score anchor boxes are retrieved using Non maximum suppression (NMS). This Computer vision method isolates an individual object from a large data set, selecting the most confident score associated with the anchor box.

The final layer of the architecture, the *yolov2Layers* function, incorporates all the design steps to create the YOLO v2 network architecture for detection. Subsequently, this network is trained based on the size and complexity of the data set using MATLAB's *trainYOLOv2ObjectDetector* function. The training is conducted using a stochastic gradient descent solver for 80 epochs and an initial learning rate of 0.001. The stochastic gradient solver is a technique often used in machine learning for optimization. The solver interactively updates parameters in the model to reduce the loss function. This is done by minimizing the difference between the predicted bounding box and the gTruth Table throughout the training phase. Choosing a low gradient allows for a smoother training approach. The both detectors took three hours to train.

The detectors are then evaluated and tested and the results are discussed in Chapter 7.

4.6 Limitations

The ACF Detector has one major limitation which hinders its performance in this project. The detector can only detect one object class therefore cannot detect each different state of the traffic light. This detection is best for identifying a single class object like the traffic light body. ACF detectors perform well when detecting objects but lack speed and accuracy when compared to YOLO v2

The YOLO v2 detector is more accurate than theACF detector as is uses deep learning to detect multiple object classes. The detector can detect the three traffic light states, red, green and yellow. The detector can also detect load shedding traffic lights. It also performs faster identification as it has GPU support. This comes at a cost as this requires powerful hardware and GPU setup. This results in real time simulation from the Simulink model to lag.

Due to the single class limitation of the ACF detector, the YOLO v2 detector was used in the Simulink model to detect the traffic light state.

Chapter 5

Navigating the Intersection

Once the detector can accurately identify the traffic light state, the automated vehicle needs to navigate through the intersection without relying on information from the driving scenario. The system currently includes a radar data generator block and a vision data generator block, which detects objects and lanes from the driving scenario. Additionally, a 3D simulation camera block is utilized by the YOLO v2 detector to identify the traffic light state.

5.1 Driving Scenario Designer

The Driving Scenario Designer is a MATLAB app that facilitates the creation of different scenarios compatible with the US City Block 3D environment. The pre-existing scenarios consist of three vehicles: the ego vehicle, and two other vehicles in the Simulink. Each scenario includes one radar and one camera sensor on the ego vehicle. These scenarios were adapted for training the detector by adjusting the ego-vehicle's start position and removing the lead car to capture the stop line from large distances away from the traffic light. Once the training was complete, the simulation time was reduced by moving the ego vehicle's starting position closer to the traffic light.

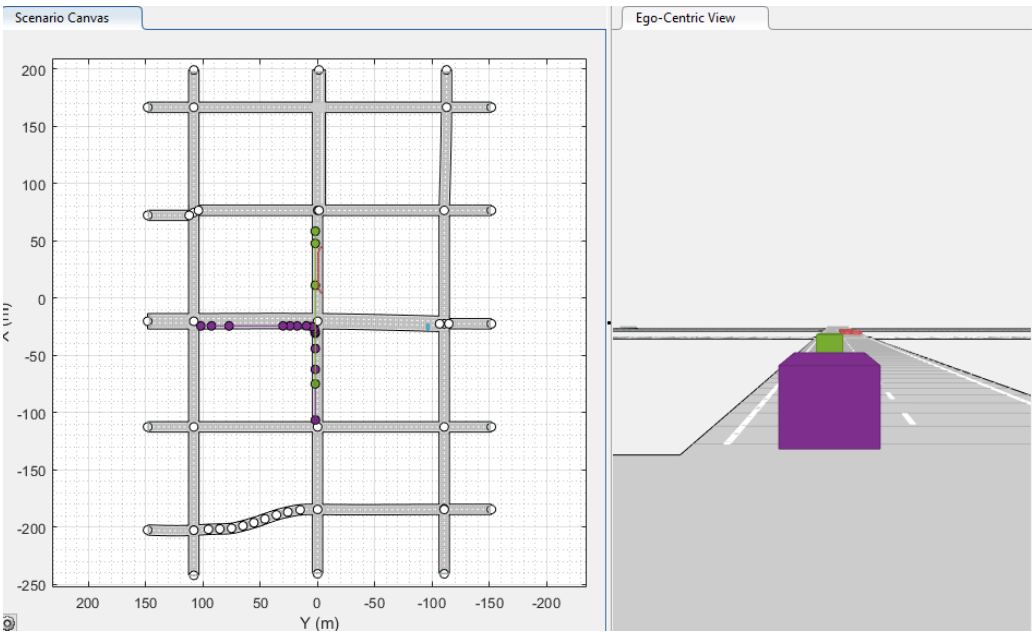


Figure 5.1: Driving Scenario Designer App Scenario Canvas and Ego-centric view

The scenario canvas displays the actors and roads in the scenario. This is where the vehicle's trajectory

is created, actors are placed, and sensors can be mounted on the vehicle. Scenarios are available in the pre-build Simulink model. Additionally, there is an Ego-Centric view that simulates what the vehicle experiences in front of it as it moves along its trajectory.

5.2 Detecting Traffic Light Intersection

5.2.1 Overview

The automated vehicle does not receive any data from the driving scenario related to the intersection . As a result, the vehicle must detect the traffic light state, determine its distance from the intersection, apply the necessary logic based on this distance and state,and navigate through the intersection safely. Several steps were taken to detect how far the intersection is from the ego vehicle. The reason for choosing to detect the road lines is as follows:

- The project's scope incorporated the use of radar and camera sensors as much as possible keeping the design cost-effective.
- The traffic light structure could be damaged, making it difficult to identify the distance from the vehicle.
- The traffic light intersection could have multiple lanes, resulting in the traffic light structure being further away, leading to incorrect distance measurements.

For these reasons the image processing techniques below were used and the steps are as follows:

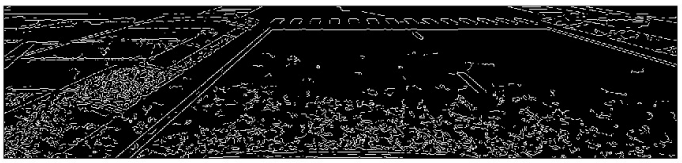
1. Crop the image to only display the bottom half of the image.
2. Convert the image to grayscale, Figure 5.3a.
3. Reduce the noise of the image by using Gaussian filtering, Figure 5.3b and 5.3c.
4. Preform Hough Transform, Figure 5.3d
5. Filter lines detected by angle and length, Figure 5.4.
6. Calculate the distance to the stop line.

5.2.2 Method

The approach is illustrated in Figure 5.3. Images from the simulation camera are fed into the MATLAB function block. Here, the image is processed, the stop line is identified, and the calculated distance to the traffic light stop line is outputted.

Figure 5.2: Original image captured by the camera mounted on the ego vehicle's rear view mirror

(a) Gray scale Cropped Image



(b) Unfiltered image after canny edge detection

(c) Filtered image after canny edge detection

(d) Lines detected after applying Hough transform

Figure 5.3: Image process showing how the stop line is detected before filtering the detected lines

Figure 5.4: Line detected after filtering data

The image is first converted to grayscale and then cropped to display only the bottom half of the full image. This is to ensure that only the road lines can be detected and minimize inaccurate detection's.

Shadows and over exposed areas in the image contribute to image noise. To reduce the noise, a Gaussian filter is applied to smooth the image. MATLAB has a built-in Gaussian function that applies the filter to the image. The filter works by calculating a weighted average of the image pixels based on Gaussian distribution, which means that the center pixels carry more weight than the outer pixels. For these images, a Gaussian distribution of three was used, and this choice is explained in Chapter 6. This gave sufficient noise filtering without distorting the stop line [24].

Figure 5.3b shows the noise in the edge detected image if no Gaussian filtering is applied. This demonstrates the effect of the Gaussian filter.

Next, edge detection is applied to the smoothed image to identify the boundaries of objects in an image, Figure 5.3c. This process involves detecting areas of abrupt change in intensity, which represents a curve in the image. These areas indicate the detected lines. MATLAB's houghlines function is used to identify any lines in the image. This is accomplished by extracting line segments in the image that match the Hough transform matrix.These lines are then filtered by length and angles to isolate the stop line. Since the stop line is always at a -90° angle, any angles outside of the range -85° to -95° are discarded.

Once the image is processed and a single stop line is detected, the distance from the stop line is calculated and used to adjust the vehicle's velocity and acceleration. The image is also cropped to ensure that the line detection's are strictly identifying the road markings and not objects in the camera view. The image processes used to extract the stop line are shown below.

If there is a stop line detected in the image, the distance is calculated as follows:

- Focal Length = 1190 m
- Actual width of the stop line = 6m

The pixel line length is calculated from the detected line points using the following code. The end points of the stop line are extracted and the pixel length is calculated using the formula below.

$\text{PixelLineLength} = \sqrt{(x2^2 - x1^2) + (y2^2 - y1^2)}$

The distance is then calculated using the following formula: $\frac{\text{FocalLength} \times \text{Width}}{\text{PixelLineLength}}$

This formula uses the concept of similar triangles. Similar triangles have the same angles but different side lengths, and they are in proportion, meaning they are related by a ratio. In the context of the images, the width of the line in the image is compared to its actual length, establishing a ratio between these measurements. This ratio also relates the distance to the camera and the focal length. Since the focal length, pixel length and width of the line are all known, the distance can be calculated using the established ratios.

5.3 No State - Load Shedding Conditions

Due to time constraints, the traffic light intersection was treated as a stop street during load shedding conditions. The automated vehicle makes use of the trained YOLO v2 detector explained in Chapter 4 and identifies the traffic light state, load shedding. The vehicle then comes to a stop at the intersection

and proceeds straight over the intersection. The vehicle detects the stop line using the line detection logic and this is integrated into the Simulink model.

The Simulink model already incorporates a sub-model for the vehicle's braking logic, Figure 5.5. The watchdog brake determines the braking status, no braking, partial braking, for full braking. The switch seen in Figure 5.5 is determined by the boolean expression and either acceleration or deceleration is applied to stop the vehicle at the intersection.

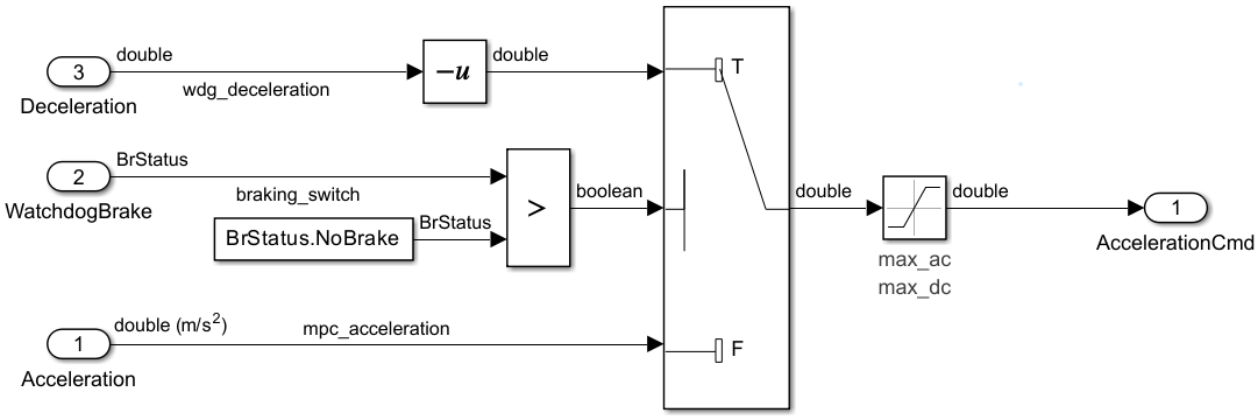


Figure 5.5: Braking logic in the Simulink model

The results can be seen in Results Section 7.

Chapter 6

Experimental Setup

This section explains how the model was set up to create the results in Chapter 7.

6.1 Gaussian Filtering

The Gaussian filter was used to reduce noise in order to accurately detect the stop line.

Figure 6.1: No Gaussian Distribution

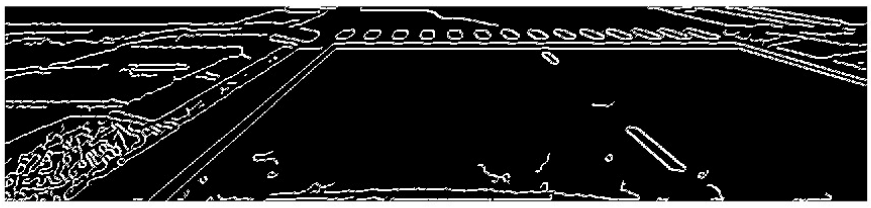


Figure 6.2: Gaussian Distribution of 3

Figure 6.3: Gaussian Distribution of 5

There is a significant amount of noise in Figure 6.1, particularly the bottom left corner. A Gaussian distribution of 5 effectively filters out the noise, distorts the stop line. Although Figure 6.2 still contains some noise, the stop line is perfectly shown with minimal interference. The Gaussian distribution of three was therefore chosen as it minimizes the noise without compensating the line that needs to be detected.

It was discovered that the noise in the bottom of the image was from the car's bumper. The image was cropped by 20cm mitigating the unwanted noise which could be confused with the lines needed to be detected.

6.2 Scenario Setup

The scenario used in the line detection results didn't incorporate a vehicle in front of the ego vehicle, as it hinders the camera's ability to detect the line, with the vehicle blocking the line.

The scenario used for detecting the traffic light state incorporated one lead vehicle in front of the ego vehicle and a vehicle crossing the intersection. Any scenario could have been used as the detector is not affect by vehicles or pedestrians.

The gTruth Table used to evaluate the detector was modified to accommodate larger bounding boxes for the traffic light.

6.3 Intersection in normal conditions

The Simulink pre-built model navigates through the intersection during normal conditions. This project adapts the model to navigate through the intersection during load shedding conditions. The normal condition navigation can be seen below.

There are four graphs in Figure 6.4: Traffic light state, Distance to the traffic light stop line, Ego acceleration, and Ego velocity. These graphs show the ego vehicle maintaining a constant velocity of 10 m/s with no acceleration. A negative acceleration of -10m/s^2 occurs at 6.8 seconds causing the vehicle's velocity to decrease until it reaches 0m/s at 7.8 seconds. Between 7.8 seconds and 13 seconds, the vehicle remains stationary while the traffic light is red.

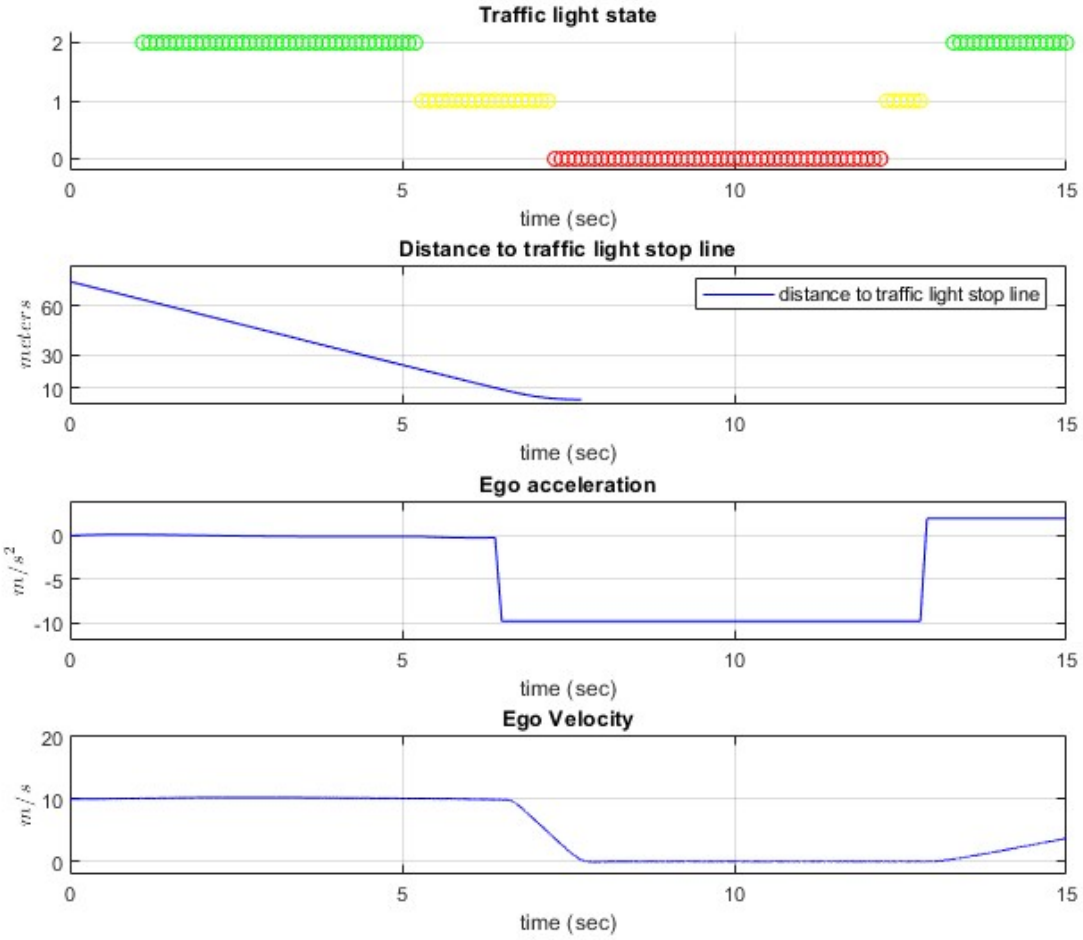


Figure 6.4: Intersection navigation in normal conditions

Chapter 7

Results and Discussion

The Results and Discussion section consists of two parts: Evaluating the two different types of trained detectors and evaluating the automated vehicles ability to navigate the intersection during load shedding conditions. At the end of each section, the results for the respective topic will be discussed.

7.1 Aggregate Channel Features (ACF) Traffic Light Detection Results

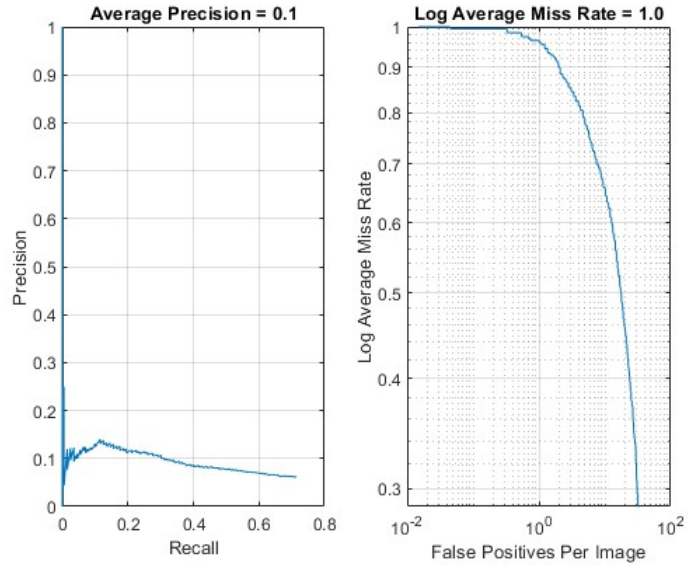
The following results evaluate the trained detectors ability to accurately detect the physical traffic light. The objective is to assess the detector's accuracy to select the most suitable detector for simulation. The ACF detector was used as a tool to gain an understand of how detector training works.

To evaluate the detector, the results of the trained detector are compared to a different gTruth Table dataset than the one used for training. Below are the graphs depicting the average precision and miss rates, with varying threshold and overlap values. The threshold sets the minimum value required for a detection to be considered positive. A higher threshold implies a stricter criterion for acceptance. Overlap or Intersection over union is the degree of agreement allowed between the bounding boxes of the detector and gTruth Table.

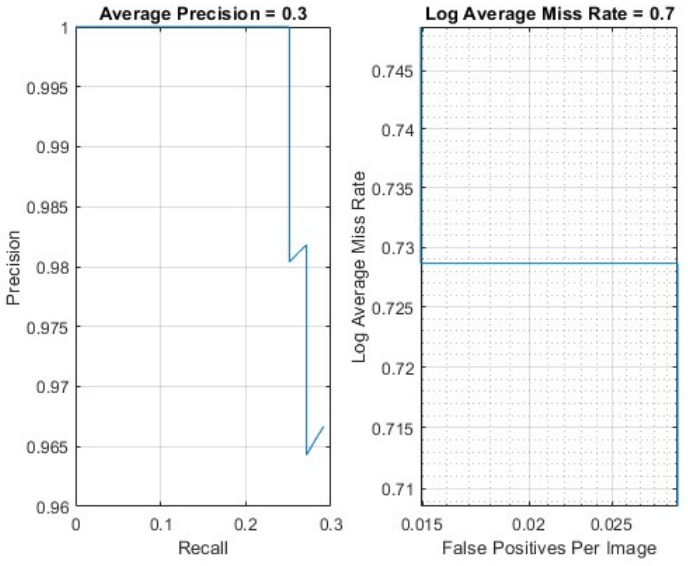
7.1.1 Results Analysis

The first detector designed was trained from a distance of 20 meters from the traffic light. The testing of the detector was done using a dataset captured from a distance of 50 meters. The results are presented below.

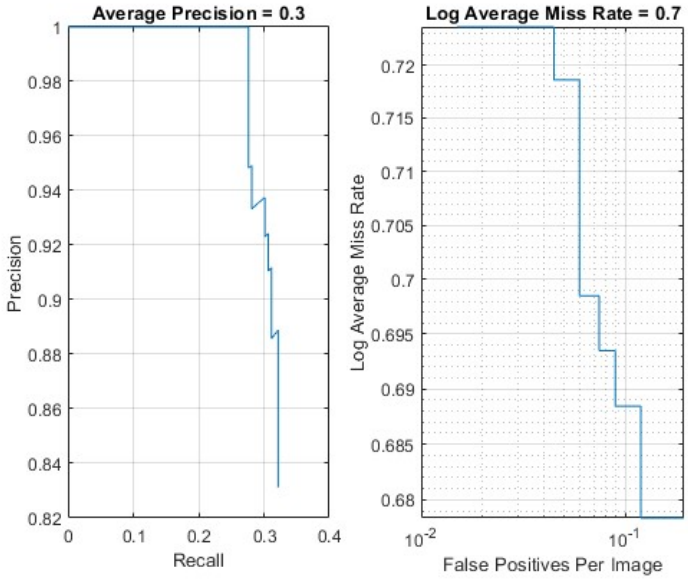
Figure's 7.1a and 7.1b have a fixed threshold of 1. This limits the detection as the most confident scores will only be considered. In contrast, Figure 7.1a demonstrates a high overlap of 0.8, and Figure 7.1b demonstrates a moderate overlap of 0.5. Figure 7.1c and 7.1d have a constant overlap of 0.5 and a varying threshold. These four graphs along with the graphs 7.3 were used to determine the acceptable threshold and overlap values used throughout the results section.



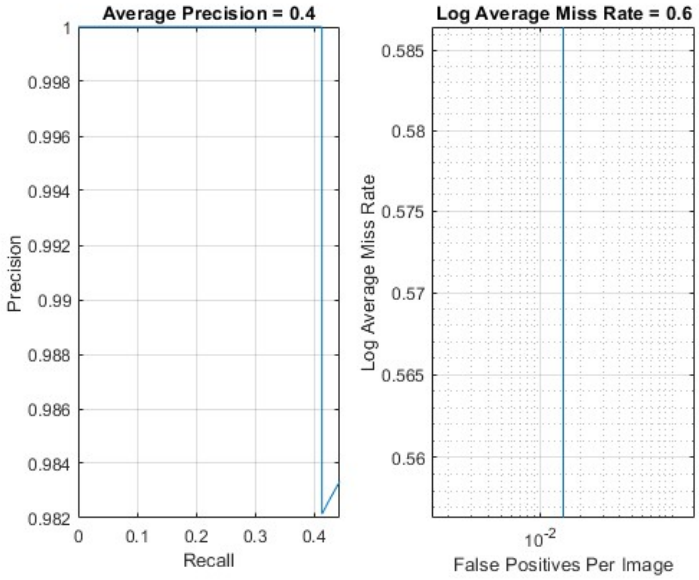
(a) Threshold of 1 and overlap of 0.8



(b) Threshold of 1 and overlap 0.5



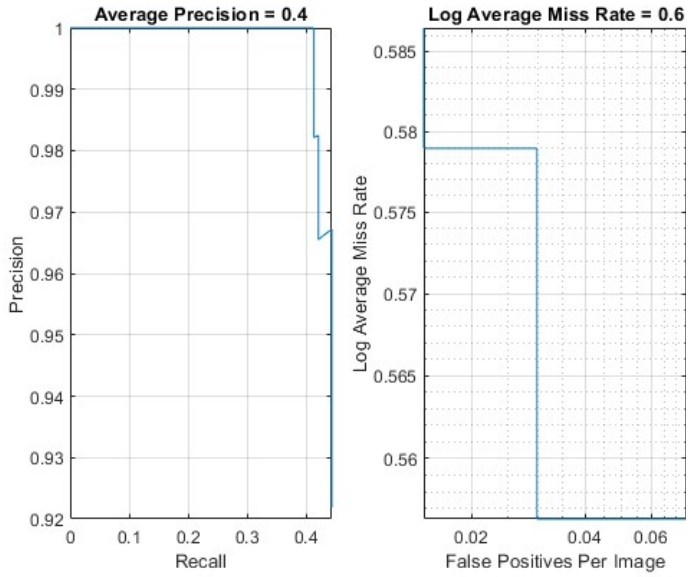
(c) Threshold of 0.1 and overlap 0.5



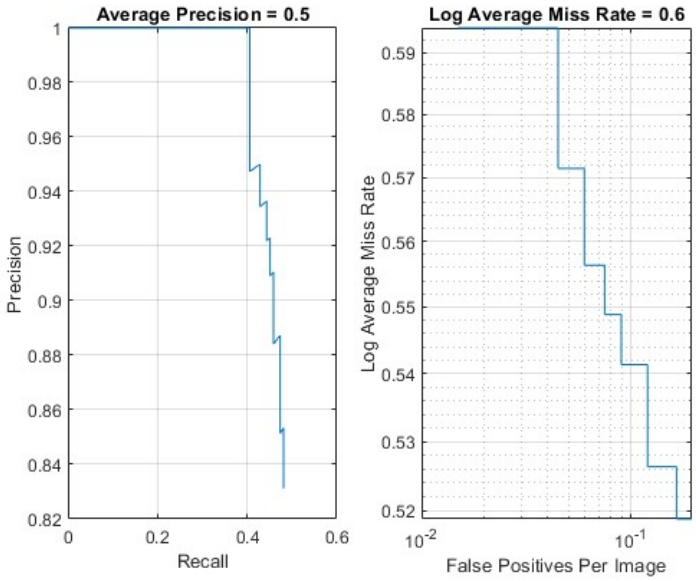
(d) Threshold of 0.5 overlap of 0.5

Figure 7.1: First ACF Detector Precision and Miss Rate with an overlap of 0.5

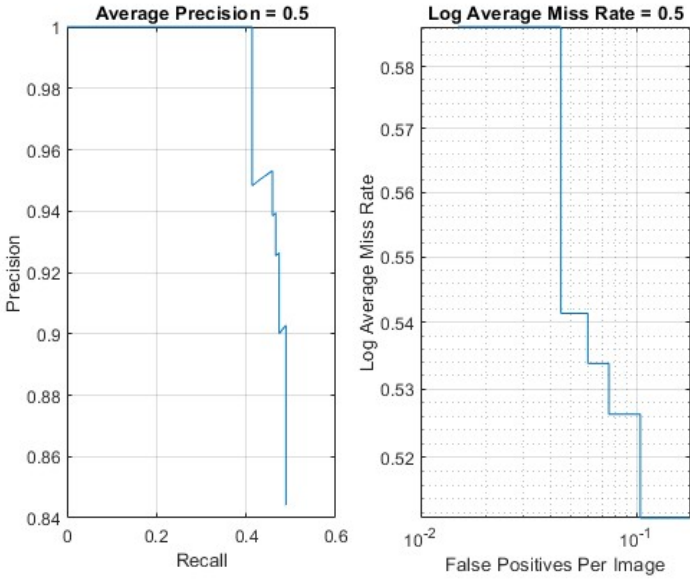
The next attempt to improve the detector involved training the detector with more data. This was achieved by moving the ego vehicle starting position to 50m away from the traffic light and recording the simulation. This setup resulted in more data for the training process to obtain better detection results. Figure 7.3 shows the new trained detectors results with varying threshold and overlap values.



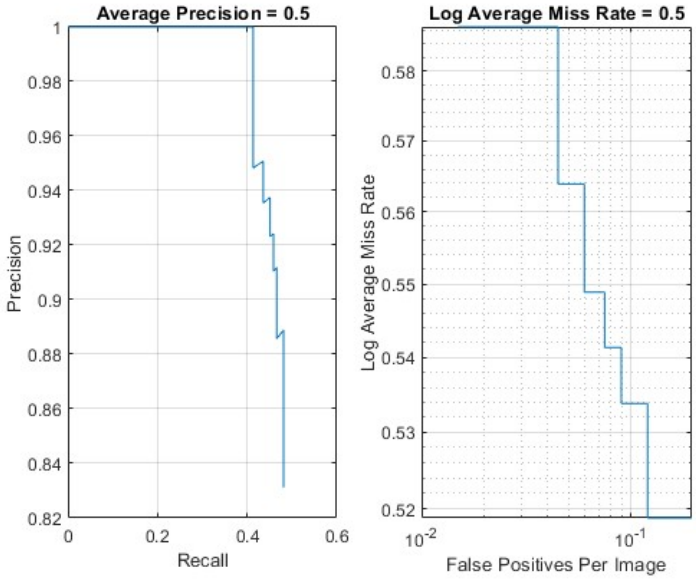
(a) Threshold of 0.8 overlap of 0.5



(b) Threshold 0.1 and Overlap 0.5



(a) Threshold of 0.5 overlap of 0.1



(b) Threshold of 0.5 overlap of 0.5

Figure 7.3: Second ACF Detector Average Precision and Miss Rate

The results from Figure 7.1 and 7.3 are discussed below.

7.1.2 Discussion

The initial ACF detector was very inaccurate due to the limited data from each state provided during training. The video used to train the detector only captured each state for 3 seconds, leading to inaccurate detection's of the initial traffic lights, which were further away and thus harder to identify. As the vehicle approached the traffic light, the detection's become more accurate. Decreasing overlap allowed for the detection's that were not full matching to be included. However, decreasing the

threshold and overlap further than 0.5 was not advisable, as it could lead to a larger miss rate by allowing more detection boxes that partially match the gTruth Table to be counted. Figure 7.2b and 7.3b illustrate this effect. The miss rate in Figure 7.2b is higher, allowing for more incorrect detection's as false positives are not filtered as strictly.

The best result for the first trained ACF detector was a threshold of 0.5 and an overlap of 0.5. Values below 0.5 are considered negative and values above 0.5 are considered positive. The overlap represents the amount the detection box overlaps with the gTruth Table bounding box. These values are commonly used as they provide a balance between precision and miss rate.

As observed in Figure 7.3, the average precision and miss rate improved compared to those in Figure 7.1. A higher threshold allows only highly confident detection scores. The decrease in threshold can be observed in Figure 7.4b. As the threshold decreased to 0.5, the precision increased since it allowed lower scores to be considered. For this detector's purpose, it only needs to detect the presence of the traffic light therefor, the overlap can be low.

Although these graphs show only 50% precision, the detector when applied to a video, exhibits a high accuracy rate in detecting the traffic light, with slightly off center bounding boxes. This can be seen by the graph results as decreasing the overlap decreases the miss rate.

7.2 You Only Look Once Version 2 (YOLOv2) Detector Results

To produce the final YOLO v2 detector, many different trained detectors were evaluated. The first YOLO v2 detector was trained in normal conditions, considering only three traffic light states: green, yellow and red. The detector was then trained to detect the load shedding states. The traffic light training data was continuously manipulated to improve the detector's results throughout the whole project. For reasons discussed in the section above, all threshold and overlap values are set to 0.5.

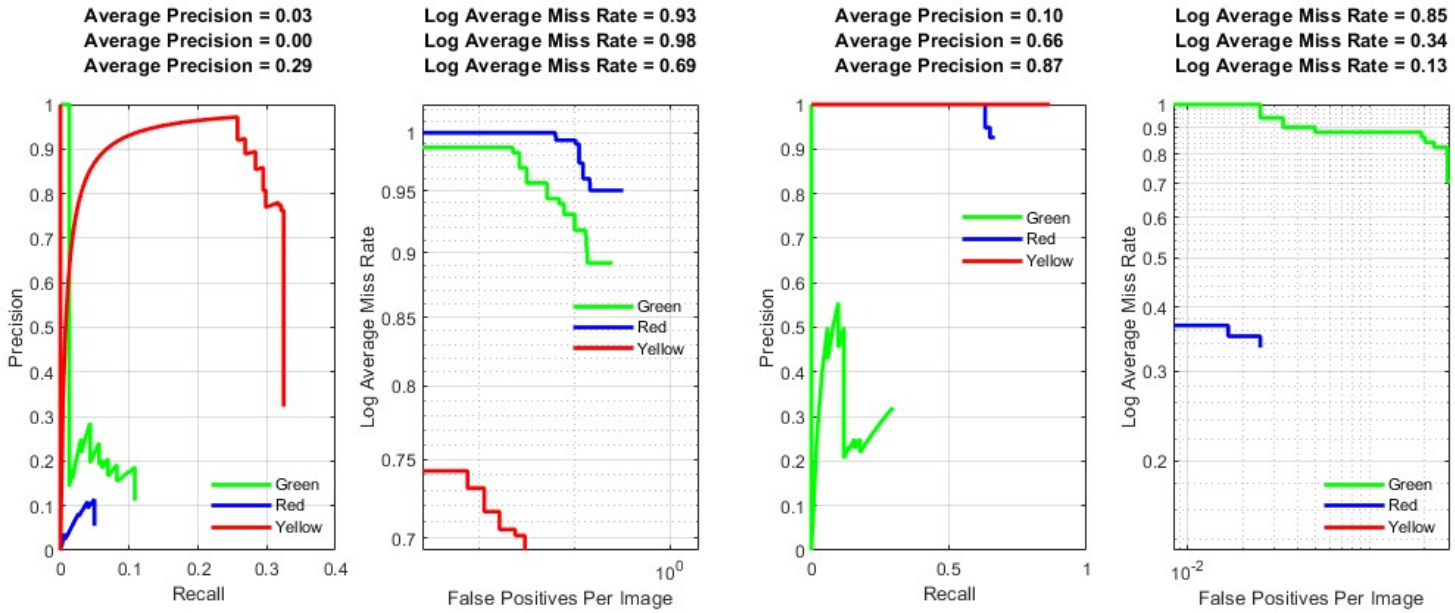
To determine the best detector to be implemented in the Simulink model, three different variations of detection were explored.

1. Detecting the middle traffic light state in normal conditions.
2. Detecting the middle traffic light state under both normal and load shedding conditions.
3. Detecting all three traffic lights states.

7.2.1 Detecting the Traffic Light State in Normal conditions

The figures presented below depict average precision and average miss rate of the detector, in comparison with a gTruth Table table. The gTruth Table table is generated from a distinct dataset, ensuring the detectors accuracy for various scenarios.

The detector was trained using data where the red and green states were displayed for five seconds, and the yellow for two seconds, Figure 7.4a. In Figure 7.4b, the yellow time was extended to five seconds, all states were displayed for five seconds.



(a) Displaying Green and Red for 5s, Yellow for 2s

(b) Holding each state for 5s

Figure 7.4: First YOLO v2 Detector Average Precision and Miss Rate

The increase in the duration of the traffic light state, as observed in Figure 7.4a and 7.4b, indicates a slight improvement in precision and a decrease in miss rate. However, both of the trained detectors illustrated in Figure 7.4 remain highly inaccurate. Figure 7.4a has no state precision above 29%. The yellow state has the highest average precision with 29%, with the lowest miss rate at 69%. The miss rate of green is 93% and red is 69% ,respectively.

Figure 7.4b demonstrates a better result, with the highest average precision being 87% and a miss rate of 13%. The green and yellow states still show low precision, 10% for green 10% and 66% for yellow. The miss rate for the green state is exceptionally high at 85%, and yellow state at 34%.

The detector was then trained with a larger dataset, Figure 7.5. Each state was displayed for 10 seconds, providing the detector with more diverse data from various angles and distances to learn from.

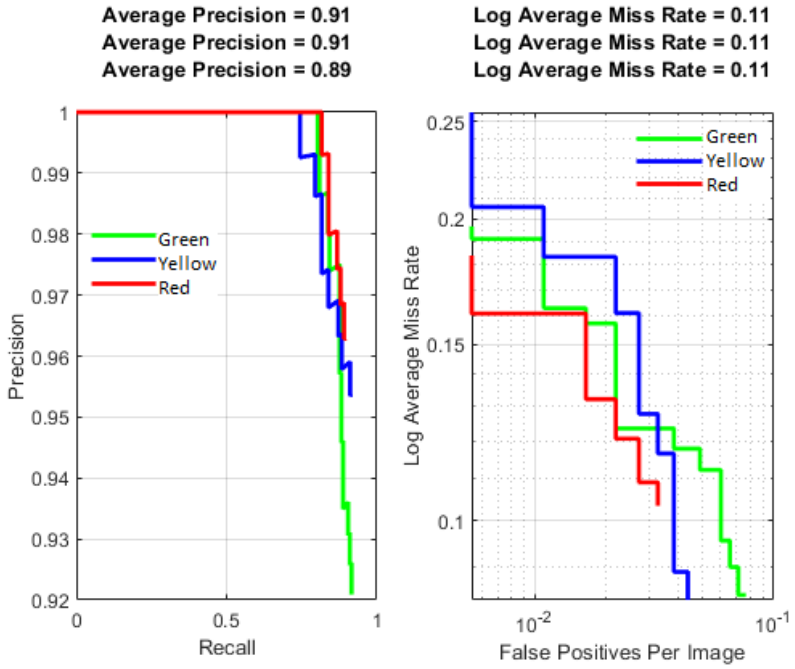


Figure 7.5: Final three state YOLO v2 detector results implemented in Simulink model

Figure 7.5 displays the results of the detector after being trained with a large dataset, indicating promising outcomes for all colour detecting. The average precision for green increased by 81%, for red by 25%, and for yellow by 2%, while the average miss rate decreased by 74% for green, 23% for red, and 3% for yellow.

7.2.2 Detecting Traffic Light States in Load Shedding Conditions

Having trained and tested multiple detectors, refining the detectors based on the results significantly reduced the time required for training several detectors. The detector was trained using a long series of frames, displaying each state for ten seconds and at various angles. The results of the load shedding detector are presented below.

In Figure 7.6, the green and no state have the lowest precision and highest miss rates. This is due to the bounding box not overlapping the gTruth Table by more than 50%. Although the detector accurately detects the traffic light state during runtime, the bounding box is incorrect, resulting in a negative detection.

An example of the detector's bounding boxes is shown in Figure 7.7 for load shedding conditions.

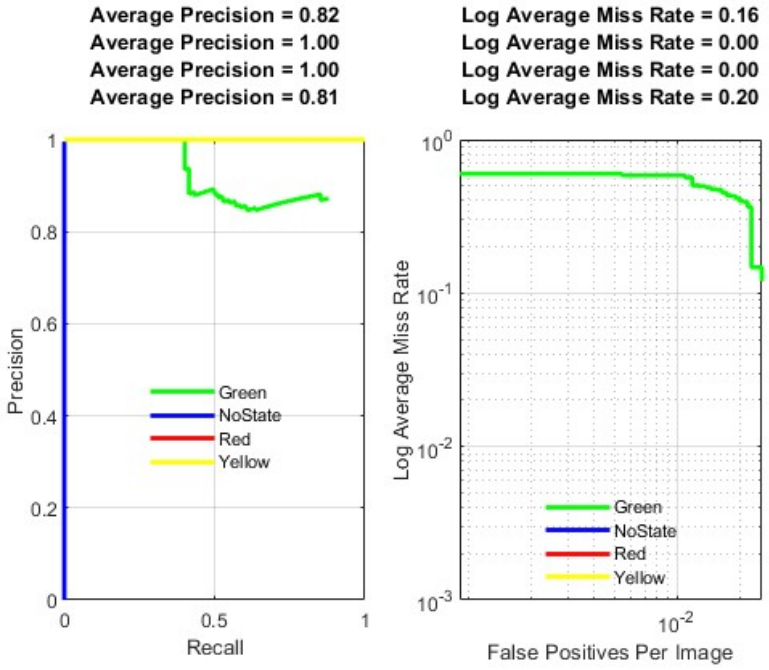
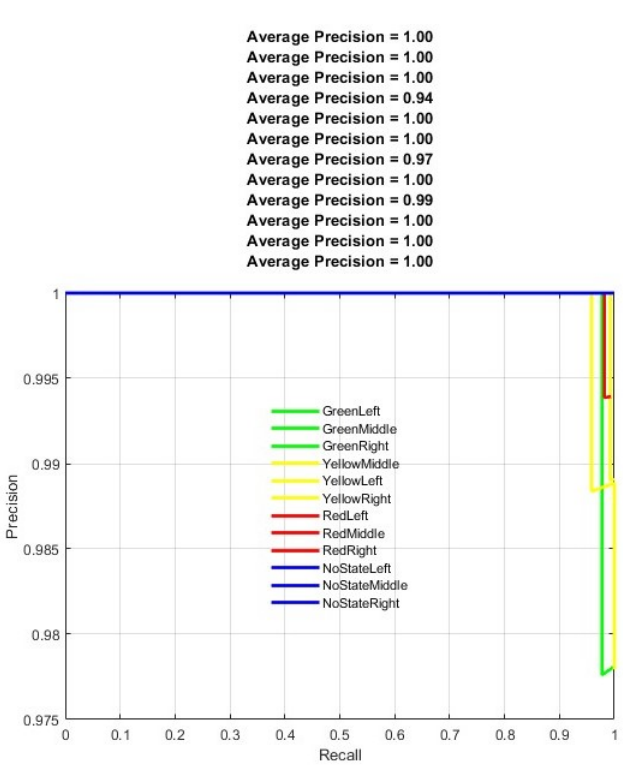


Figure 7.6: Average Precision and Miss Rate for all states

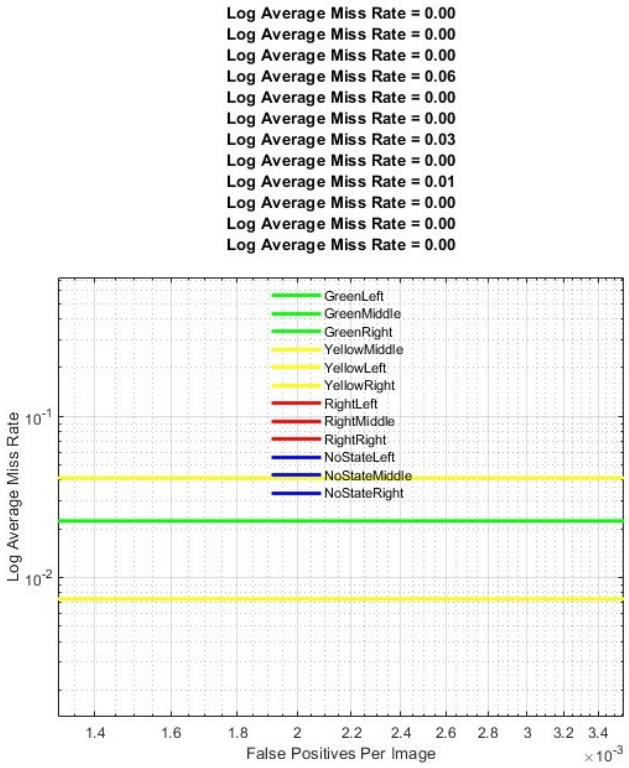
Figure 7.7: Bounding Boxes annotated during load shedding conditions

7.2.3 Detecting All Three Traffic Lights in All States

The final detector was trained to identify all the traffic lights and their states. The results are presented in Figure 7.8a.



(a) Average Precision for all three traffic lights and all four states



(b) Average Miss rate for all three traffic lights and all four states

Figure 7.8: Average Precision and Miss Rate for detecting each physical traffic light state

Detecting all three traffic lights and their states proved to be successful. Only three detection's didn't have 100% precision with the lowest being 94%. Only three detection's had an average miss rate above 0%, with the highest being 6%. These inaccuracies were due to the bounding boxes being off-center.

Figure 7.9 illustrates how all three traffic lights are detected.

Figure 7.9: Detection of all three traffic lights

7.2.4 Discussion

The first detector, as depicted in Figure 7.4, was trained with limited data, Only five seconds of each state, creating a restricted angle for each state. During the 3D Simulation, the camera approaches the traffic intersection with the first state being green, then yellow then red. This sequencing leads to the red state having the highest accuracy, as the vehicle is closest to the light during the red state, allowing for more accurate red detections. However, this scenario specificity creates implications when the detector is presented with a different scenario to what it was trained with. This can be seen by the inaccuracy in Figure 7.4. The yellow average precision is significantly higher than all the other state detection in both Figure 7.4a and 7.4b. This is because the red light is the last traffic light state, placing the vehicle closest to the light at this point.

After training the detector with a wider data set the results improved significantly for the YOLO v2 detectors. The detector was given a dataset where each state was held for 10 seconds, the full duration of simulation. Each state was recorded and then the video was concatenated to form one long dataset. Comparing Figure 7.4a and 7.5 there is a significant increase. Average precision for green increased by 88%, red by 91%, and yellow by 60%. The average miss rate decreased for green by 82%, red by 87%, and yellow by 58%. This proves that the data used to train the detector directly affects the detectors ability. Therefore manipulating the data to contain enough data for each state produces the best detector.

Since multiple detectors had been trained and tested there was no need to train the detector with minimal data as the results shown above don't prove positive. The load shedding detector, Figure 7.6, had an accuracy of 100% for NoState, representing load shedding conditions, and red. Green had 82% precision and yellow 81%. The miss rate for no state and red are 0%, green 16% and yellow 20%. The green and yellow states had the lowest accuracy due to the bounding boxes seen in Figure 7.7. Although the detector always accurately detects the colour of the traffic light, the ROI is off center producing a negative result when compared to the gTruth Table.

The final detector demonstrates a high level of accuracy, with only three states below 100% precision and miss rate, the lowest being 94% and 6%, respectively. Despite these minor inaccuracies, the detector effectively identifies the correct state. This detector's capability to rely on multiple traffic light detections is especially valuable in South Africa, where many traffic lights are often damaged, ensuring reliable state detection based on the other two traffic lights.

7.3 Navigating the Intersection for Load Shedding

The objective is to understand the system's accuracy of how the system behaves during simulation.

Once the traffic light can detect the state, the appropriate logic is applied to ensure the vehicle comes to a stop at the intersection and then continues through it. These experiments aim to evaluate the Simulink model's response during load shedding conditions. The graphs below show the distance to traffic light, ego vehicles acceleration and velocity. The simulation time was reduced to 15 seconds.

The results shown below are using the scenario's intersection distance which is used to compared with the line detection distance.

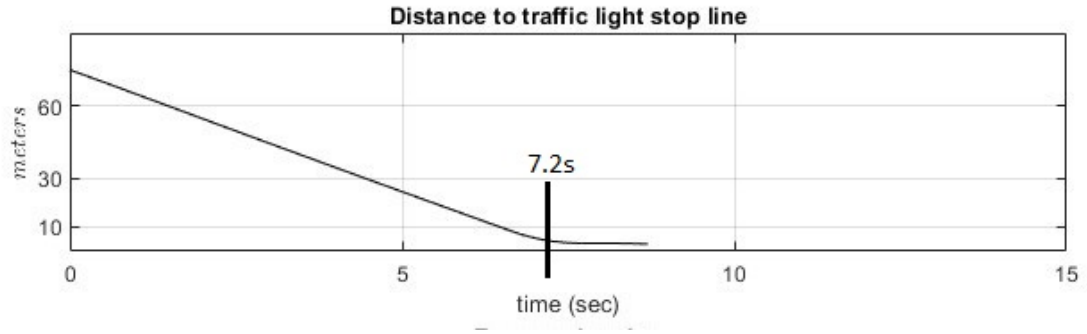


Figure 7.10: Distance to traffic light from ego vehicle

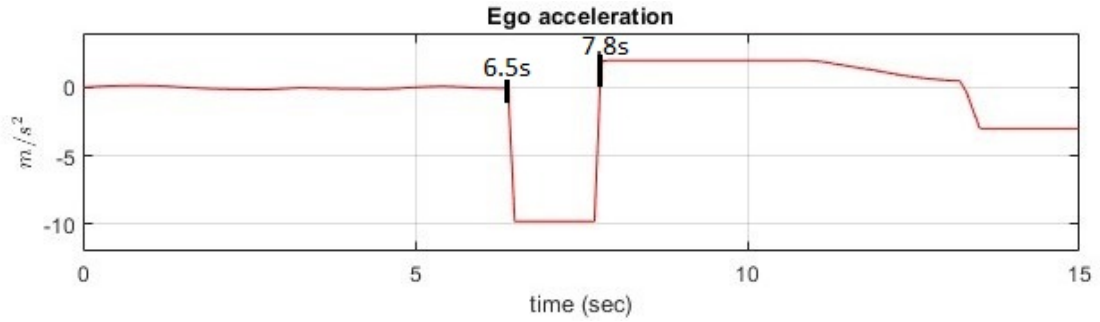


Figure 7.11: Ego vehicles acceleration

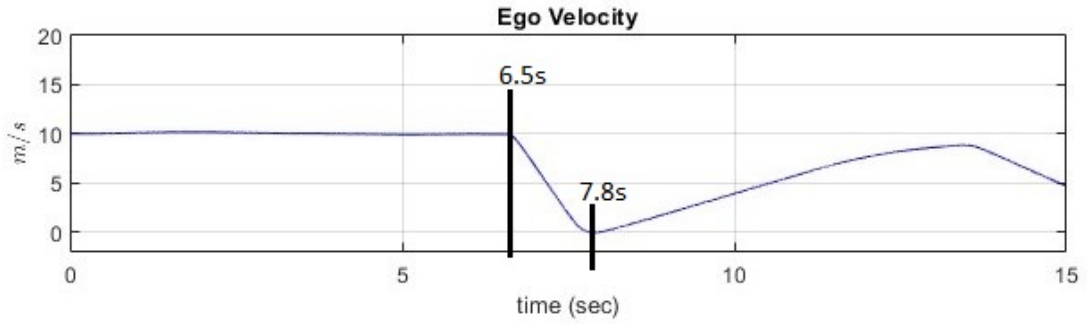


Figure 7.12: Ego vehicles velocity

The graph shown in Figure 7.10 illustrates the decreasing distance to the traffic light overtime. The

vehicle reaches the traffic light within 7.2 seconds, and the graph stops at 8.5 seconds as the vehicle moves past the stop line and proceeds through the intersection.

Figure 7.11 displays a sudden decrease in acceleration from 0m/s^2 to -9.8 m/s^2 at 6.5 seconds, remaining constant for 1.2 seconds before quickly spiking to 0m/s^2 at 7.8 seconds.

In Figure 7.12, we observe a rapid decrease in velocity from 10 m/s to 0m/s in 1.2 seconds. The velocity gradually increases until 14 seconds. Subsequently, there is a decrease in velocity accompanied by a negative acceleration at 6.5 seconds, followed by an increase at 7.8 seconds. Fluctuations in acceleration directly impact the velocity.

As the distance to the traffic light decreases, the velocity decreases due to negative acceleration which quickly slows down the ego vehicle. The ego vehicle maintains a velocity of 0 m/s for 0.7 second upon reaching the stop line. The acceleration then spikes at 7.8 seconds to 0 m/s, allowing for a gradual increase in velocity over time. The time frame has been minimized for simulation purposes,but in real world experiments, these duration's would be extended.

The results using the line detection established in Chapter 5 are shown below.

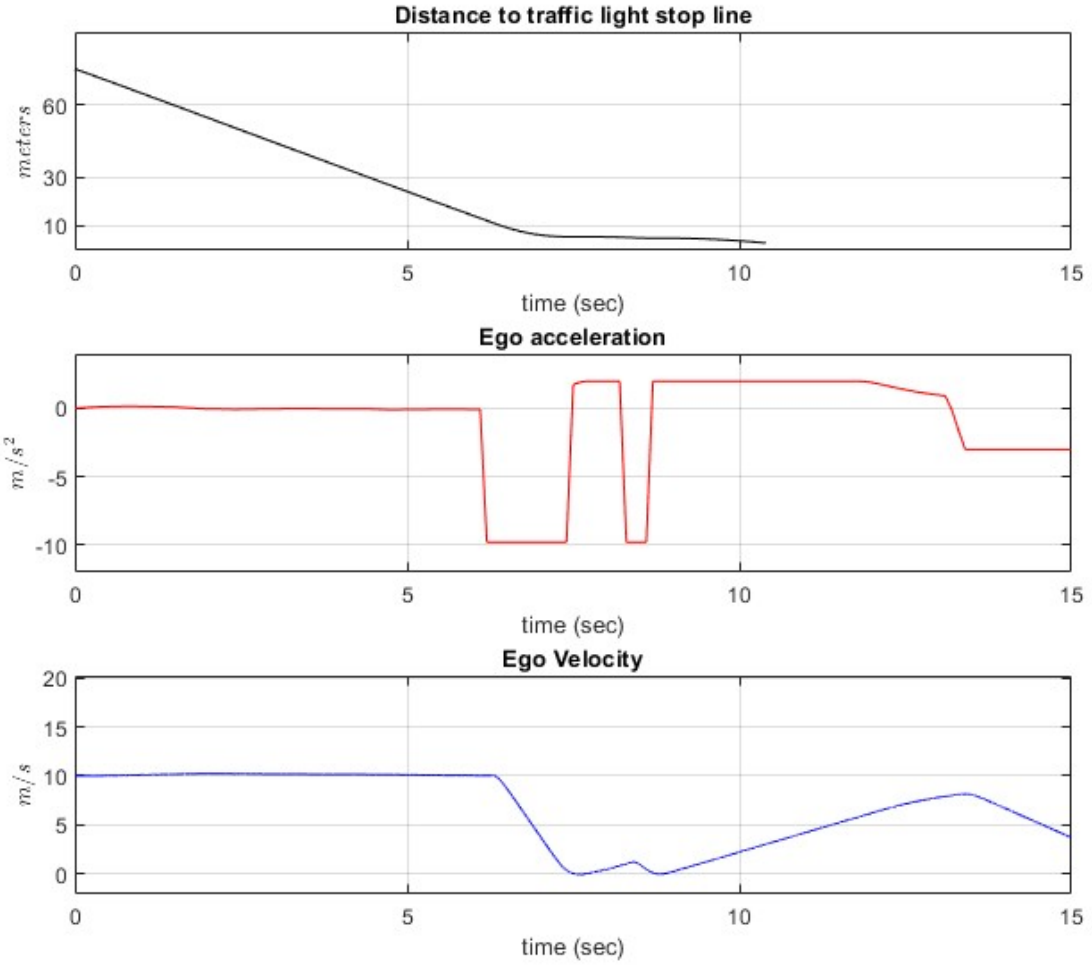


Figure 7.13: Navigating the Intersection using line detection

At first, the ego velocity decreases between 6.5 seconds and 7.8 seconds. Between 7.8 seconds and 8.3 seconds, the velocity gradually increases until 13.5 seconds, with abnormal decrease occurring between 8.3 seconds and 8.8 seconds. These fluctuations in velocity correspond to the changing in the acceleration graph. During deceleration, the vehicle slows down until the vehicle reaches the intersection. Acceleration is then applied once the vehicle has stopped at the intersection. The distance to the traffic light line stops graphing as the vehicle moves across the intersection.

7.3.1 Discussion

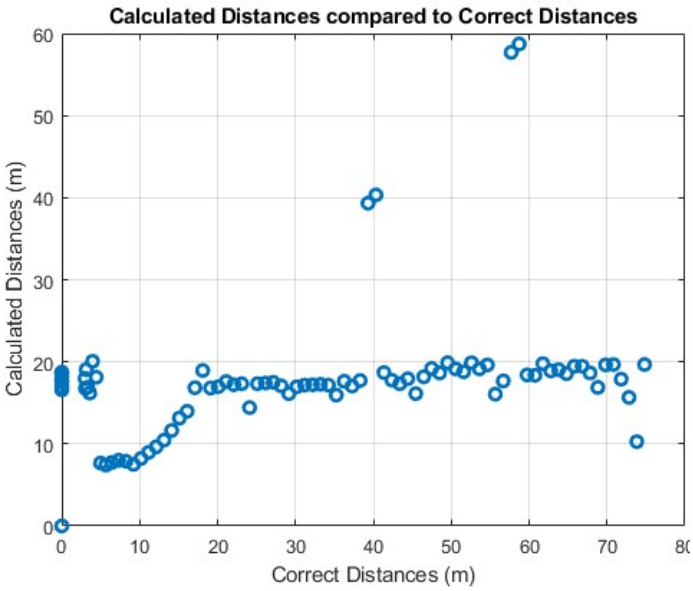
The results obtained above validate the ego vehicle's capability to navigate through the intersection during load shedding conditions, although some irregularities are observed in the graphs. Initially, the ego vehicle gradually reduces its speed when it approaches within 11 meters, followed by rapid deceleration, bringing the vehicle to a complete stop at 7.8 seconds. The vehicle then resumes motion, moving away from the stop line with an increasing velocity, demonstrating its successful passage over the intersection.

Due to the limitations of the line detection, as discussed in Chapter 5, detection's at the stop line are slightly inaccurate. Consequently, the model experiences decelerate between 8.3 seconds and 8.8 seconds.

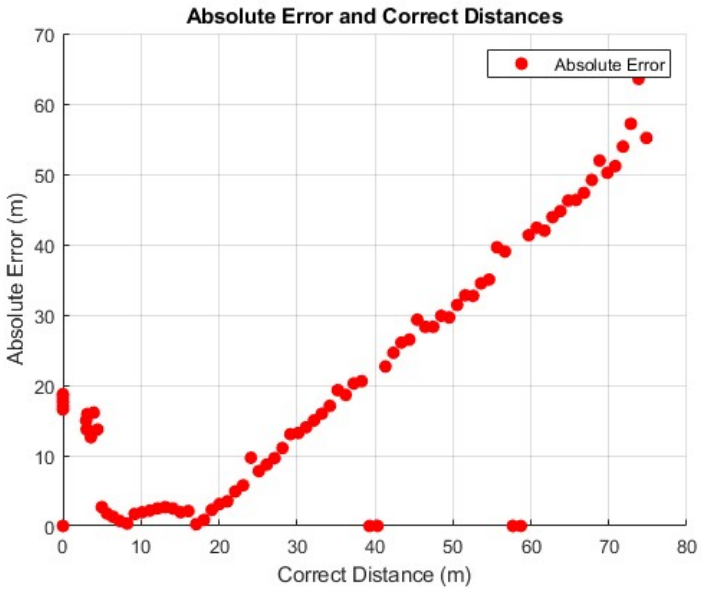
7.4 Detecting Traffic Light Intersection

Detecting the stop line at the traffic intersection can be a complex process since camera frames can be positioned incorrectly and the road view is limited. The image was cropped to only process the bottom half, therefore reducing processing and incorrect line detection. The calculated distance is directly affected by the line detected after image processing. This can be seen by Figure 7.14a.

To evaluate the calculated distance to the stop line, the simulation was run and error between the calculated and actual distances was plotted below. For distances where there was no line detected the distance was set to the correct distance in order to graph the response. This can be seen in the error graphs, Figure 7.14b and 7.15b, from 80m to 22m with some outliers where an incorrect detection was made. The lines were first detected without applying the Gaussian filter, Figure 7.14. A Gaussian filter was applied and graphed in Figure 7.15.

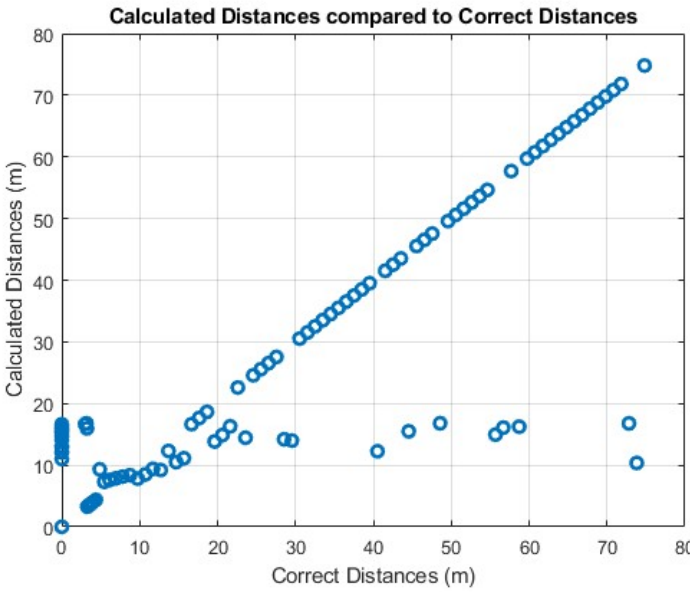


(a) Error between calculated distance and the actual distance to the stop line before Gaussian filter

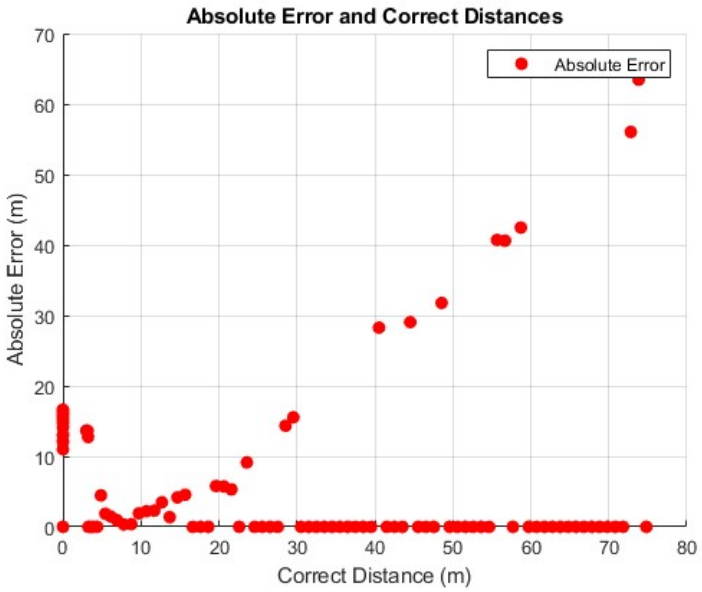


(b) Error between calculated distance and the actual distance to the stop line after Gaussian filter

Figure 7.14: Graphs showing line detection results before Gaussian filtering



(a) Error between calculated distance and the actual distance to the stop line before Gaussian filter



(b) Graphs showing line detection results after Gaussian filtering

Figure 7.15: Error between calculated distances and the actual distances to the stop line

Since the stop line is only visible from 22m away due to the cropping of the image, We will only consider these detection's and the outliers will be discussed in the limitations section.

7.4.1 Discussion

Applying the Gaussian filter proves to be successful in Figure 7.15 showing a significant improvement. The line detection is accurate to within 7m. As the vehicle gets closer and reaches the stop line, the

detections are inaccurate. This is due to the stop line being too large in the image or the stop line getting cut off in the image. These limitations are discussed below.

7.4.2 Limitations

One of the biggest limitations is the shadows along the road. The edge detection picks up the shadows as lines creating false positives. This is illustrated in Figure 7.16.

Figure 7.16: Shadows from buildings image

Another limitation is the full line not being detected. This results in the length of the line being incorrect giving an incorrect distance. Figure 7.17 also illustrates the stop line being cut off the image. This is due to the camera positioning.

Figure 7.17: Only half the line detected

Chapter 8

Conclusions and Future Recommendations

> The same rule holds for us now, of course: we choose our next world through what we learn in this one. Learn nothing, and the next world is the same as this one.
>
> —*Richard Bach, Jonathan Livingston Seagull*

8.1 Report Summary

The purpose of this project was to create a South African environment for an autonomous vehicle to navigate through an intersection using perception to identify traffic lights.

This report began with Chapter 2 reviewing various literature related to perception based detection and autonomous vehicles. The review started with an overview of South African road conditions and autonomous vehicles, setting the scene for the environment and the type of vehicle used in this project. It then continues to review three types of perception based detection, camera, radar and Lidar and explains further the software used for similar projects. The last part of the review looked at the integration of the automated vehicle and non-automated vehicles. This review found that camera and radar are the better option for this project due to costs and complexity of the system.

Chapter 3 then provided a summary of the theory behind the existing Simulink model, which was then modified for the project. The chapter began with Simulink model setup, summarising all the subsystems which work together to create the working Simulink model. The chapter then explains the environment for the model and visualization and ends off with tracking and detecting objects.

The bulk of the work for this project followed next, in Chapter 4 and Chapter 5. This is where the project design and simulation was established. These chapters work together to achieve the overall perception based detection Simulink model. Chapter 4 creates the traffic light detection system and Chapter 5 designs an image processing model to detect how far the intersection is from the vehicle. Combining these two Chapters creates the overall Simulink model solution for the project.

In Chapter 4 two types of detectors are designed, YOLO v2 and ACF. The YOLO v2 detector is trained to detect all traffic light states including load shedding conditions. Chapter 5 then creates a MATLAB function which uses image processing techniques to calculate how far the stop line is from the vehicle in order to apply the necessary breaking force to safely stop at the intersection.

Finally, Chapter 7 evaluates each section's results and discusses the outcome. It was found that identifying all three physical traffic light states is the best option, as traffic lights can be damaged allowing the detector to still detect the state using another one of the three traffic lights. It was also noted that there is large amounts of noise in the images as the shadows and lighting of the simulation creates large contrasts in the image. The line detection can be very inaccurate due to this. The distance to the traffic light is also influenced by the detection and hence the noise creating errors with the calculation. Further image processing can be implemented to achieve more accurate results.

8.2 Conclusion

In summary, the project achieved the goals that were set out. A Simulink model that can detect the traffic light state and navigate through the intersection safely using perception. There is room for improvement in detecting the stop line and investigating other methods. In conclusion, the project was successful, achieving all goals set out in the scope.

8.3 Future Recommendations

8.3.1 Exploring Lidar Detection

All designed sub-models used the 3D simulation camera module in Simulink to minimize costs and complexity. There is also radar detection in the pre-built Simulink model to track objects. Lidar data could be used to track the distance of the traffic light or enhance object detection.

8.3.2 Training a new version of You Only Look Once Detector

The YOLO v2 detector has a new version with improved performance. This means better existing features of the YOLO v2. These include improved precision to detect the traffic light from further away,increase in processing speed, and handle a wider range of objects to detect. The YOLOv4 offers increased flexibility allowing training flexibility and network configuration adapting the detector to the necessary needs.

8.3.3 Radar and Camera Sensors

Currently, the Simulink model only has one front facing camera and radar sensor. This limits detection to objects in front of the vehicle. Implementing radar and camera sensors all around the vehicle will enhance the detection and collision of objects.

Bibliography

[1] S. Han, X. Wang, L. Xu, H. Sun, and N. Zheng, "Frontal object perception for intelligent vehicles based on radar and camera fusion," in *2016 35th Chinese Control Conference (CCC)*. IEEE, 2016, pp. 4003–4008.

[2] Varsha, "Computer vision: History amp; how it works," Apr 2023. [Online]. Available: https://www.sama.com/blog/computer-vision-history-how-it-works/#:~:text=The%20History%20of%20Computer%20Vision,image%20processing%20and%20pattern%20recognition.

[3] Brendan, "4 common causes of road accidents: Understanding the risks," Mar 2023. [Online]. Available: https://www.gertnelincattorneys.co.za/blog/raf/4-common-causes-of-road-accidents-understanding-the-risks/#:~:text=According%20to%20Gert%20Nel%2C%20%E2%80%9CReckless,Africa%20were%20caused%20by%20driver

[4] Anon., "The cities with the worst traffic in South Africa and where the most dangerous roads are," *BusinessTech*, 2023. [Online]. Available: https://businesstech.co.za/news/motoring/545018/the-cities-with-the-worst-traffic-in-south-africa-and-where-the-most-dangerous-roads-are/

[5] D. P. Thomas, "Public transportation in south africa: Challenges and opportunities," *World*, vol. 3, no. 3, 2016.

[6] Jul 2023. [Online]. Available: https://www.arrivealive.mobi/minibus-taxis-and-road-safety#:~:text=A%20study%20done%20by%20the,than%20all%20other%20passenger%20vehicles.

[7] M. Sinclair, "Attitudes, norms and driving behaviour: A comparison of young drivers in south africa and sweden," *Transportation research part F: traffic psychology and behaviour*, vol. 20, pp. 170–181, 2013.

[8] H.-H. Jebamikyous and R. Kashef, "Autonomous vehicles perception (avp) using deep learning: Modeling, assessment, and challenges," *IEEE Access*, vol. 10, pp. 10 523–10 535, 2022.

[9] P. a. K. Bansal, "Forecasting americans' long-term adoption of connected and a," Jan 1970. [Online]. Available: https://ideas.repec.org/a/eee/transa/v95y2017icp49-63.html

[10] S. Aoki and R. Raj Rajkumar, "A-drive: Autonomous deadlock detection and recovery at road intersections for connected and automated vehicles," in *2022 IEEE Intelligent Vehicles Symposium (IV)*, 2022, pp. 29–36.

[11] S. Aoki and R. Rajkumar, "Safe intersection management with cooperative perception for mixed traffic of human-driven and autonomous vehicles," *IEEE Open Journal of Vehicular Technology*, vol. 3, pp. 251–265, 2022.

[12] J. Zhou, H. Chen, and C. Xiu, "A simulation model to evaluate and verify functions of autonomous vehicle based on simulink®," in *Intelligent Robotics and Applications: Second International Conference, ICIRA 2009, Singapore, December 16-18, 2009. Proceedings 2*. Springer, 2009, pp. 645–656.

[13] P. Wang, "Research on comparison of lidar and camera in autonomous driving," *Journal of Physics: Conference Series*, vol. 2093, p. 012032, 11 2021.

[14] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, 2021.

[15] B. Anand, V. Barsaiyan, M. Senapati, and P. Rajalakshmi, "Region of interest and car detection using lidar data for advanced traffic management system," in *2020 IEEE 6th World Forum on Internet of Things (WF-IoT)*. IEEE, 2020, pp. 1–5.

[16] D. J. Yeong, G. Velasco-Hernandez, J. Barry, and J. Walsh, "Sensor and sensor fusion technology in autonomous vehicles: A review," *Sensors*, vol. 21, no. 6, p. 2140, Mar 2021. [Online]. Available: http://dx.doi.org/10.3390/s21062140

[17] E. R. Haley, D. J. Coe, and J. H. Kulick, "3-d visualization of simulink physics models using unreal engine," in *Proceedings of the International Conference on Modeling, Simulation and Visualization Methods (MSV)*. The Steering Committee of The World Congress in Computer Science, Computer . . . , 2012, p. 1.

[18] Z. Xinxin, L. Fei, and W. Xiangbin, "Csg: Critical scenario generation from real traffic accidents," in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, pp. 1330–1336.

[19] G. Yanhui, X. Qiangui, H. Shousong, and J. Xiao, "Flight control system simulation platform for uav based on integrating simulink with stateflow," *TELKOMNIKA Indonesian Journal of Electrical Engineering*, vol. 10, no. 5, pp. 985–991, 2012.

[20] N. Barney and M. Courtemanche, "What is load shedding?: Definition from techtarget," May 2023. [Online]. Available: https://www.techtarget.com/searchdatacenter/definition/load-shedding#:~:text=Load%20shedding%20(loadshedding)%20is%20a,primary%20power%20source%20can%20supply.

[21] [Online]. Available: https://www.mathworks.com/discovery/object-detection.html

[22] [Online]. Available: https://www.mathworks.com/help/vision/ug/getting-started-with-yolo-v2.html

[23] J. Redmon and A. Farhadi, "Yolo9000: Better, faster, stronger," 07 2017, pp. 6517–6525.

[24] [Online]. Available: https://www.southampton.ac.uk/~msn/book/new_demo/gaussian/#:~:text=The%20Gaussian%20Smoothing%20Operator%20performs,how%20large%20the%20template%20is.

Appendix A

Appendix

A.1 Simulink Model

Simulink Model GitHub Repository

A.1.1 Line detection Code

Line Detection GitHub Repository

A.1.2 Detectors

ACF Detector GitHub Repository

YOLOv2 Detector GitHub Repository

A.2 Simulink Model Diagrams

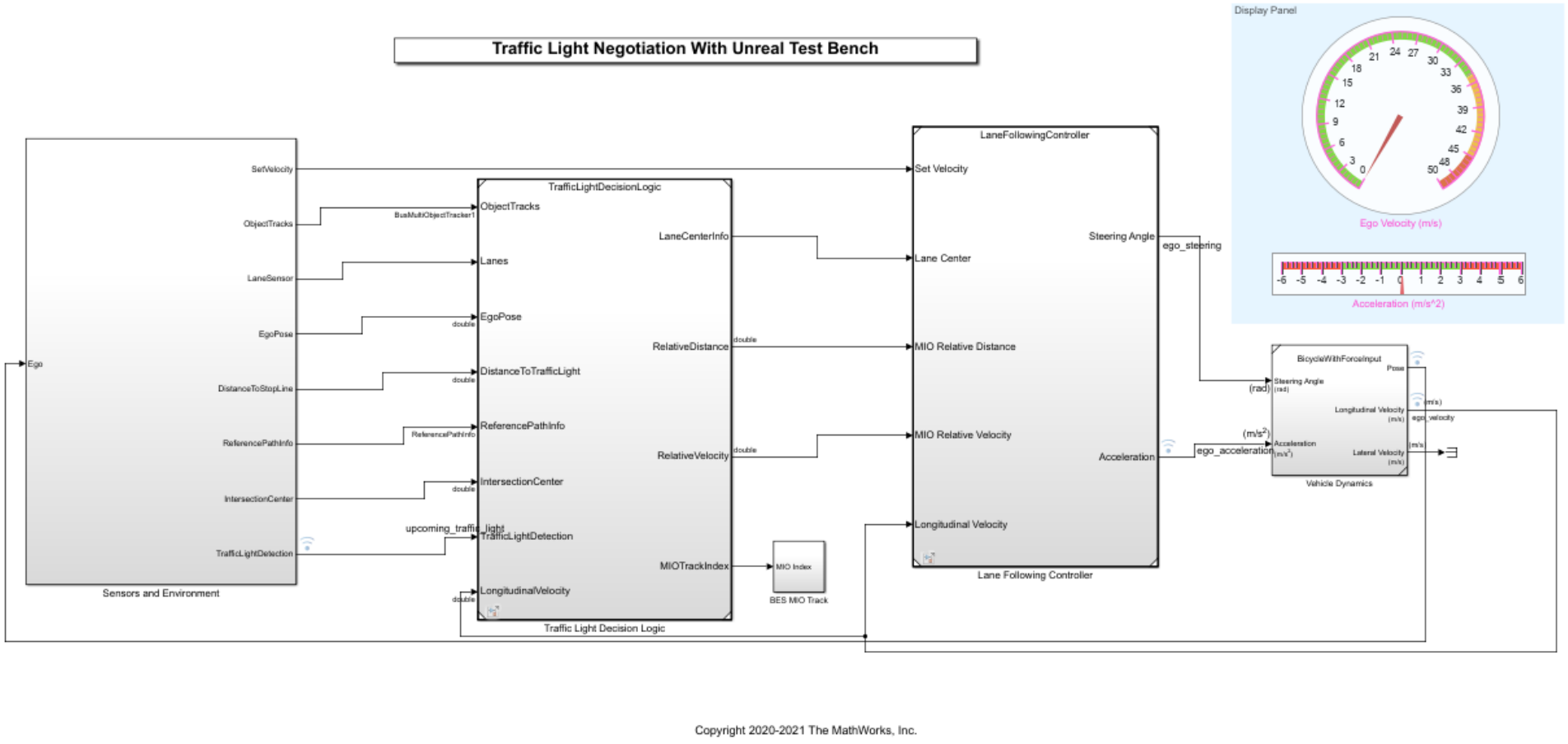


Figure A.1: Simulink Model

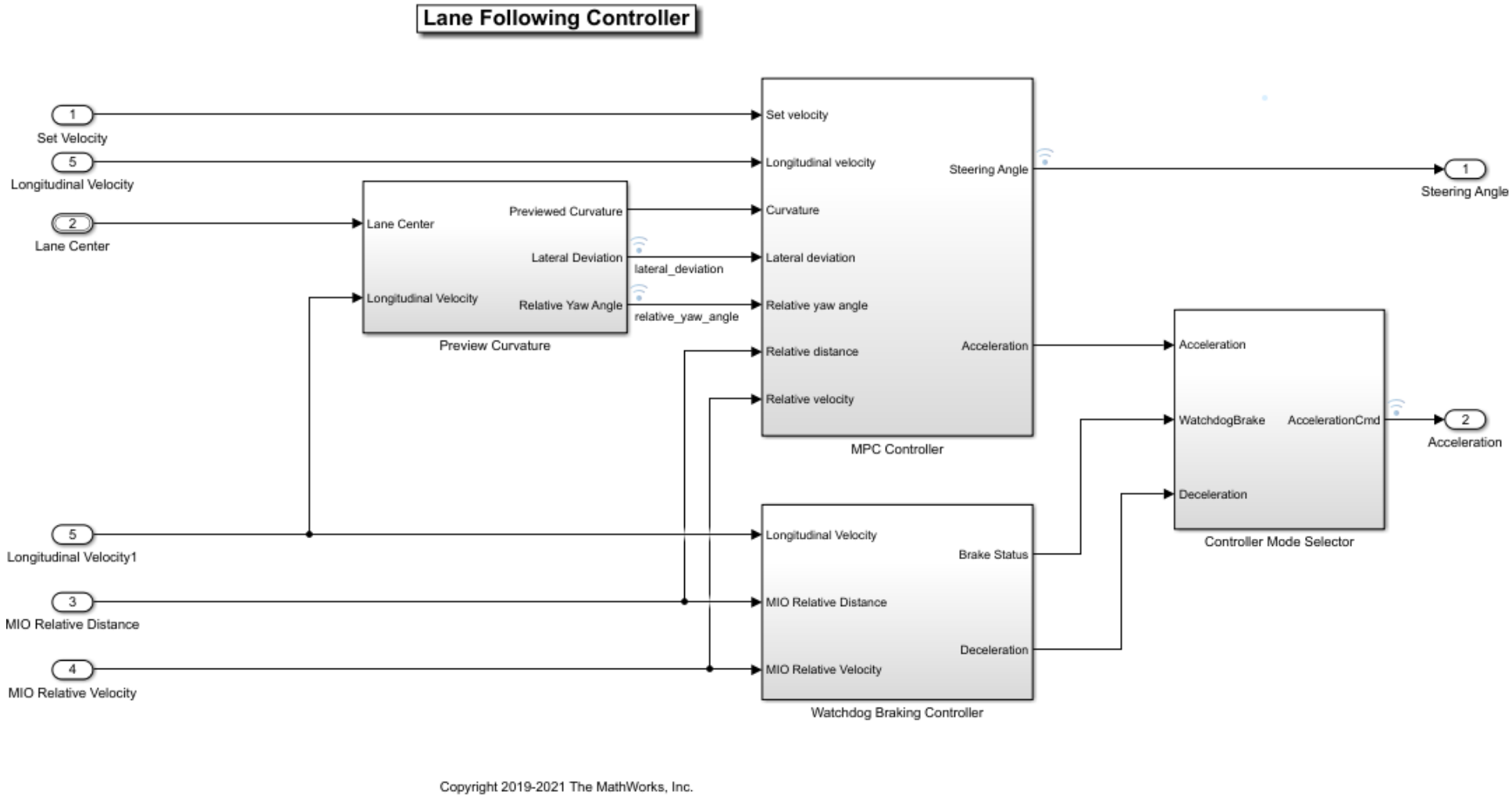


Figure A.2: Lane following Controller in prebuilt Simulink model

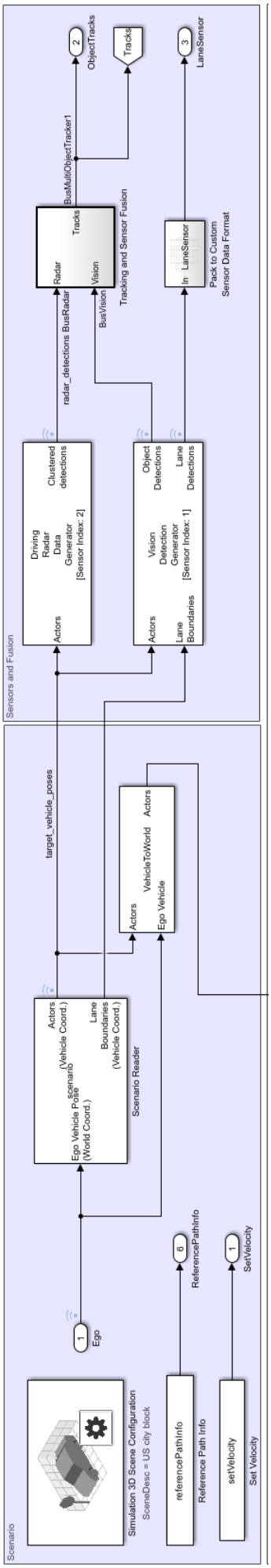


Figure A.3: Lane following Controller in prebuilt Simulink model