Computer Vision Techniques for Automotive Perception Systems

THESIS

Presented in Partial Fulfillment of the Requirements for the Degree Master of Science in the Graduate school of The Ohio State University

By

Evan Michael Stoddart, B.S.

Graduate Program in Electrical and Computer Engineering

The Ohio State University

2019

Thesis Committee:

Shawn Midlam-Mohler, Ph.D., Advisor

Giorgio Rizzoni, Ph.D.

Abstract

Computer vision is widely used in modern robotics and automated machinery due to its highly diverse ability to recognize patterns, track objects, and map environments. Over the last decade, computer vision has also become an integral part in advanced driver assistance systems (ADAS) which assist drivers in operating motor vehicles. Modern ADAS use sensors, such as cameras, to characterize the environment and either alert the driver or take control to lessen the severity of, and even prevent, accidents. This work discusses a vision-based, SAE Level 0 ADAS system designed for a collegiate automotive design competition. The experimental design and setup will introduce the custom, hybrid-electric ego-vehicle, used as a prototype deployment vehicle for the system, as well as the components and software practices. Next, the computer vision techniques used to detect and track target vehicles, will be discussed. This is structured chronologically in the stages followed to make a camera-based system. This included component modeling and calibration, image processing and computer vision operations, and application-focused feature extraction techniques used in lane line identification, vehicle classification and localization, and vehicle tracking. All software developed for this system used the OpenCV library for Python and C++. This library is easy to use and well-documented, making it a convenient resource for a plethora of vision-based algorithms. Validation techniques will also be discussed in terms of software-in-the-loop, hardware-in-the-loop, and in-vehicle testing. The system placed 4th of 16 schools in the ADAS Evaluation Event at the end of year competition. The performance of the system is discussed. The performance results were sufficient for the needs of the competition; however, additional reliability would be desired for commercial use. The concepts discussed in this work, however, show promise towards implementing a similar system by following the systems-level development and validation practices discussed. This work serves as a template for the design and implementation of a vision-based perception system with relevance to not only driver assistance, but autonomous vehicles and robotics as well.

Dedication

To my father, David Stoddart, who inspired me from a young age to pursue engineering. You always led by example; showing me the value of hard work and commitment. A dream fifteen years in the making and I am finally ready to start an engineering career of my own.

Acknowledgements

I would like to thank my teammates: Simon Trask, Shlok Goel, and Akshra Ramakrishnan for their support on this document and in this research. My former teammates: Brandon Bishop, Greg Jankord, and Andrew Huster for their leadership and example they set as team leaders, graduate students, and engineers. I would like to thank my team: Subash Chebolu, Joey Chui, George Li, Andrew Fu, Ryan Williamson, and Daniel Dillon for their willingness to learn and eagerness to contribute to EcoCAR projects. I would also like to thank my faculty advisor, Shawn Midlam-Mohler, for his endless support and encouragement. Lastly, I would like to thank Ohio State, General Motors, and Argonne National Lab for providing me with the opportunity and resources to enhance my engineering and leadership skillsets on EcoCAR.

Vita

2014………………………………………………………………… North Allegheny Senior High School

2018………………………… B.S. Electrical and Computer Engineering, The Ohio State University

2019………………………… M.S. Electrical and Computer Engineering, The Ohio State University

Publications

19AE-0071 - Systems Engineering of an Advanced Driver Assistance System, Evan Stoddart, Subash Chebolu, Shawn Midlam-Mohler, SAE World Congress, 2019

Field of Study

Major Field: Electrical and Computer Engineering

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Modern driver assistance systems are complex, utilizing vehicle parameters, environmental observations, and traffic patterns to assist the driver to improve safety and efficiency. These systems are added cost-to-ownership due to the added expense of sensors and computing hardware needed to perceive the environment. Perception sensors are costly, and so are the associated calibration and repair costs. Thus, further development in this area is needed to improve reliability, performance, and decrease costs. This work describes image processing, computer vision, and sensor fusion techniques used for the development of a production advanced driver assistance system (ADAS) for a student engineering competition. The ego-vehicle, a custom hybrid-electric Chevrolet Camaro, will be introduced for context for the project methodology and the competition. Project goals, scope, and timeline will be provided as well as an overview of the system before software techniques are discussed.

1.1 EcoCAR 3 Challenge

The Center for Automotive Research at the Ohio State University hosts several collegiate motorsports teams including a hybrid vehicle research team competing in US Department of Energy (DoE) hosted Advanced Vehicle Technology Competitions (AVTCs). From 2014 to 2018, the DoE hosted the EcoCAR 3 Competition in collaboration with General Motors (GM), and several other supporting sponsors [1]. The competition selected 16 universities in the US and Canada to redesign a classic American muscle car; a 2016 Chevrolet Camaro. The EcoCAR 3 Challenge served as a hands-on opportunity for students to implement hybrid technologies into a GM donated Camaro with their own designs, components, and controls development. Students design their powertrain architecture in Year 1 and perform simulation to defend the feasibility of their design. During Year 2, the teams benchmark their Camaro, then disassemble much of the stock vehicle to implement their own version of the powertrain. Years 2 through 4 featured annual competitions at GM proving grounds for dynamic test events, using industry-grade evaluation methods, to evaluate performance criteria such as 0-60-time, handling, emissions, and fuel economy. The AVTC series is a highly recognized program for both its research and as a gateway for students into the automotive sector. The Ohio State team has participated in these competitions for over 20 years and has a track record of success including 4 straight 1st place finishes in the EcoCAR 3 Challenge.

1.1.1 Ego – Vehicle Architecture

For EcoCAR 3, OSU created a plugin hybrid electric vehicle using a P0-P3 parallel/series architecture with a longitudinal 5 speed automated manual transmission and a four-cylinder E85 direct-injected internal combustion engine. The P0 motor, a belted alternator starter, provided a solution to start and stop the engine quickly as well as charge the vehicle's custom 18.9 kWh battery pack. Furthermore, a P3 motor provided 40 miles of all electric range as well as the option for parallel operation.

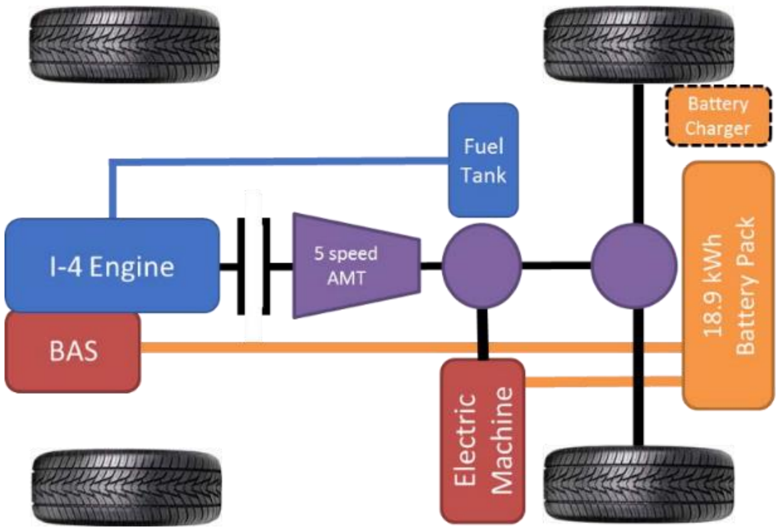


Figure 1.1: EcoCAR 3 Architecture

Historically, AVTC projects focused on energy efficiency and emissions using alternative fuels and electrification. EcoCAR 3 added additional emphasis on consumer acceptability and driver assistance features, due to the recent trends in the industry. In Year 4, the final year of the competition from Autumn 2017 through Spring 2018, the team was tasked with incorporating an ADAS system into their Camaro. The system was to utilize a camera, competition supplied hardware, and additional sensors to improve safety and fuel economy. The vehicle architecture discussed above served as the development, testing, and deployment platform for the ADAS projects discussed in this work.

This work will focus on the perception system design, implementation, and validation for an ADAS for a student automotive engineering competition. This section outlines the goals of the project and the timeline in which major milestones of the project were completed.

1.2 Project Scope and Goals

The project was outlined by competition rules as an ADAS system capable of providing feedback to the driver through a team-designed human machine interface (HMI). An example of an HMI is shown in Figure 1.2, which illustrates two icons that might appear to the driver when a lane centering feature is active on a vehicle.

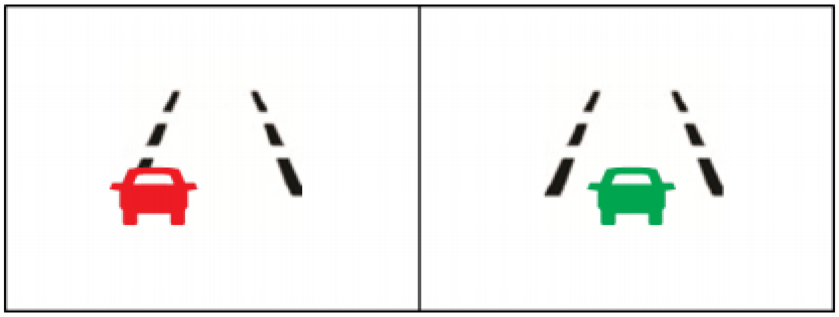


Figure 1.2: Example of a Lane Centering HMI [2]

The objective of the system was to implement features capable of driver feedback only, meaning that the driver must maintain sole ownership of powertrain control. Feature design was left up to teams to determine which feature and feedback method worked best for their vehicle. The system was also expected to identify and track target vehicles in the forward-facing region of the ego-vehicle and log their trajectories as part of the technical evaluation of the system. Logs were taken directly from the on-board diagnostics port (OBD-II) of the ego-vehicle which contained target vehicle data including a unique identifier, position, velocity, and lane. An illustration of what the ego-vehicle and a target vehicle might look like in an example scenario is shown in Figure 1.3. In this example, the log would report the longitudinal distance to the target vehicle measured directly along the longitudinal (x) axis directly boresight with the ego-vehicle. It would also report the azimuth (angle from boresight) to the target as well as it’s longitudinal velocity and lane position. The log format was specified by the competition and kept standard across all teams. To implement the ADAS system, computer vision and sensor fusion software would need to be developed to not only provide the data needed for the ADAS features, but to also match the log format.

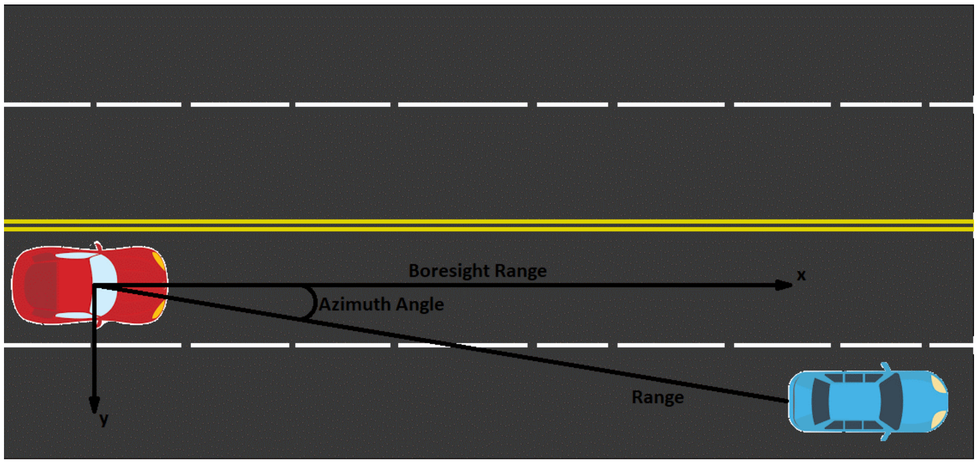


Figure 1.3: Illustration of Ego-Vehicle and a Target Vehicle in a Highway Scenario

Technical constraints were also outlined. The system was to use at least one camera to place emphasis on in-house computer vision development, but teams were permitted to use additional sensors. Additionally, the system was prohibited from interacting directly with the ego-vehicle powertrain. The system was to behave within the parameters of an SAE Level 0 system, only providing suggestions to the driver on how to operate the vehicle [3].

1.2.1 Project Timeline

The work for this project was split into three sections, characterized by intermediate checkpoints in the competition. This is shown in the Gantt Chart in Figure 1.4. All aspects of the project were completed in 1 academic year in preparation for the technical evaluation at Year 4 competition.

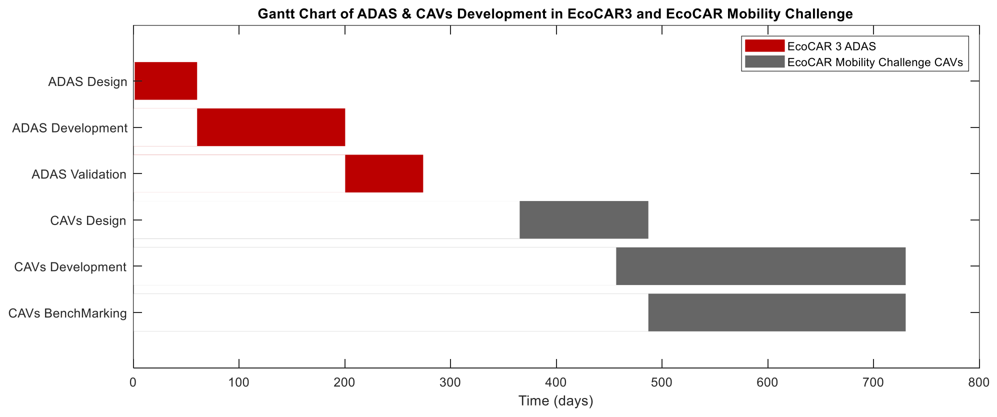


Figure 1.4: Project Timeline

1.3 Summary

A high-level introduction has been outlined with project scope and goals to provide insight into the purpose and background of this project. Additionally, a timeline illustrates the time spent designing, developing, and validating the system discussed in this work. This will serve as a reference for the methodology that will be explored and described in the following chapters.

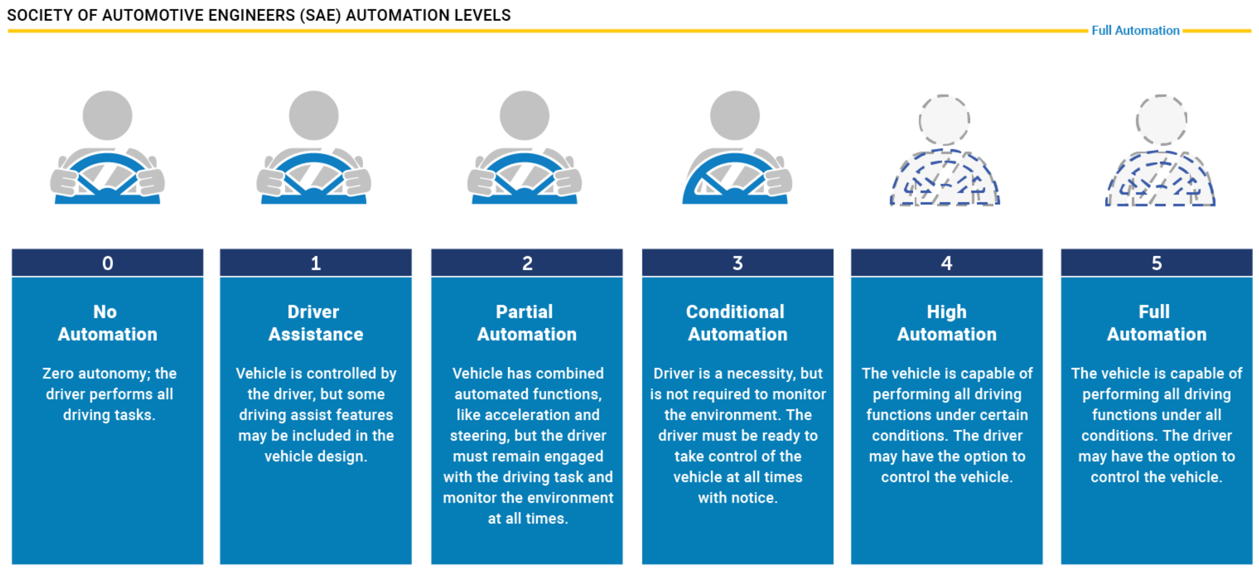
Chapter 2: Literature Review

Driver assistance systems have become a leading technology for modern automobiles. It is a market which is expected to exceed 67 billion USD by the year 2025 [4]. These systems have become increasingly more complex since their debut, placing more emphasis on the perception techniques discussed in this work.

2.1 Introduction to Driver Assistance

The first wave of driver assistance was systems like anti-lock braking systems (ABS) which provided the driver with additional control during dynamic maneuvers. Traction control, electronic stability control, and cruise control have also utilized electronic controls to automate tasks usually performed by the driver. These systems use sensors to monitor vehicle parameters and gain insight into how the vehicle is behaving to conditions in the environment. Now, modern ADAS use sensors to directly monitor their environment to provide additional levels of assistance through warnings and automation.

The industry categories safety systems into either passive safety (PS) or active safety (AS). PS systems increase safety during and after crashes. Airbags are a primary example of such systems. AS systems are preemptive, attempting to prevent crashes from occurring in the first place. These systems are the focus for ADAS from a safety. AS and ADAS are generally synonymous, however ADAS is a more encompassing term because it includes systems geared toward safety as well as convenience and energy usage. ADAS can be divided into several additional categories: Driver Aids, Collision Avoidance, and Semi-Autonomous features. All three of these categories offer varying levels of automation which would typically performed by the driver. The Society of Automotive Engineers (SAE) Levels of Automation, shown in Figure 2.1, outlines the behavioral expectations of a driver when operating a vehicle with ADAS through autonomous capability [5].


SOCIETY OF AUTOMOTIVE ENGINEERS (SAE) AUTOMATION LEVELS

0	1	2	3	4	5
No Automation	**Driver Assistance**	**Partial Automation**	**Conditional Automation**	**High Automation**	**Full Automation**
Zero autonomy; the driver performs all driving tasks.	Vehicle is controlled by the driver, but some driving assist features may be included in the vehicle design.	Vehicle has combined automated functions, like acceleration and steering, but the driver must remain engaged with the driving task and monitor the environment at all times.	Driver is a necessity, but is not required to monitor the environment. The driver must be ready to take control of the vehicle at all times with notice.	The vehicle is capable of performing all driving functions under certain conditions. The driver may have the option to control the vehicle.	The vehicle is capable of performing all driving functions under all conditions. The driver may have the option to control the vehicle.

Figure 2.1: SAE Levels of Automation [3]

2.1.1 Driver Aids

Driver aiding systems are typically SAE Level 0-1 [5]. They offer the driver “another set of eyes” when performing normal driving functions. For instance, blind spot monitoring systems use sensors to monitor obstructed road regions to warn the driver of obstacles they may not be able to see. Additionally, backup cameras with a dashboard display assist the driver when in reverse. However, driver aids can utilize automation for specific tasks, such as park assist, which uses sensors and ECUs to guide the vehicle into parallel and perpendicular parking spaces when enabled. This level of automation is categorized as Level 1 autonomy.

2.1.2 Collision Avoidance

Collision Avoidance systems actively keep the vehicle out of accidents, when possible. This can be through warnings to the driver or direct intervention. Examples of warning systems are forward collision warning (FCW), lane departure warning (LDW), and rear cross traffic alert (RCT). However, most collision avoidance systems now include an automated second-stage which will activate if the driver does not respond to the warning. These are systems like automatic emergency braking (AEB), lane keep assist (LKA), and rear-collision avoidance.

2.1.3 Semi-Autonomous Features

Semi-Autonomous features are characterized by SAE Levels 2 and 3 of automation in which the driver's role has been heavily augmented or even replaced completely for an extended duration [5]. An example of this is park assist. All the driver must do is stop adjacent to an open spot and select the park assist function and the vehicle will control the brake, accelerator, and steering wheel to perform the maneuver. This feature is available from nearly every major automaker as of 2018 [6]. Another such example is highway assist. Highway assist features combines adaptive cruise control (ACC) and LKA to produce a hands-off highway assistance system which uses lane centering as well as throttle and brake control to maintain a set speed. This is useful for long stretches of highway driving, such as interstate travel, but can also be used in stop-and-go traffic. Currently, there are three such systems in the United States market: Tesla's Autopilot, Nissan's ProPilot Assist, and GM's Super Cruise.

Autopilot was the first semi-autonomous system on the market. However, it is not completely hands-free. It is classified as SAE Level 2 automation with lane centering and ACC operations, however, due to Level 2 requirements, the driver must remain in the loop. Autopilot requires that the driver place their hands on the steering wheel to ensure they are prepared to take control if needed. Tesla verifies this by detecting torque applied to the steering column. If the driver does not cooperate with the warning to place their hands on the wheel, the system will disengage and slow the vehicle to a stop [7]. The system can perform and suggest lane changes, provided the driver initiates the maneuver by engaging the turn signal. If the lane of interest is occupied, the system will wait for traffic to clear before transitioning. The system can be engaged on any road but cannot monitor traffic lights or stop signs. This presents a problem in usability since the system can be engaged anywhere but cannot perform the necessary tasks to operate anywhere. Tesla embraces this approach, allowing customers the freedom to make decisions about how they use their product.

Nissan ProPilot Assist is an Autopilot competitor aiming specifically at stop-and-go traffic applications. While Nissan’s system can only be enabled at speeds above 32 km/h, it will bring the vehicle to a complete stop to match traffic. Like Autopilot, it also requires that the driver place their hands on the steering wheel. However, ProPilot Assist requires this interaction always while it is enabled. Currently, the system only performs single lane steering and cannot automatically perform lane changes [8].

GM claims Super Cruise is the first “truly hands-free highway system” [9]. It monitors the driver using an infrared camera above the steering wheel. This means the driver is not required to interact physically with the vehicle unless takeover is needed; however, their focus must remain on the road. Like systems from other manufacturers, this ensures that the driver remains attentive and can override the system when necessary. Super Cruise is newer than Autopilot but has fewer features. For instance, Super Cruise does not have an automatic lane change feature and cannot be activated on every road. However, Super Cruise is much more robust. It utilizes calibrated maps of almost all United States limited-access freeways to localize the vehicle’s coordinated to 5 cm precision [10]. The service is currently only on the Cadillac CT6 platform, with further expansion expected.

Semi-Autonomous features are the latest form of driver assistance systems on the market. They provide robust vehicle control in high-level driving operations. All ADAS systems, including semi-autonomous systems, prioritize driver input. They are designed to disable if the driver uses the brake or takes control of the steering wheel. Currently, semi-autonomous systems are considered a luxury and won't be standard on most vehicles until the cost of development can be reduced.

2.2 Consumer Cost of ADAS

ADAS packages will add additional cost to the sticker price for consumer vehicles. This cost will vary by make, manufacturer, and feature due to the differences in sensors, calibration techniques, and software. Some systems, like the semi-autonomous highway assistant feature, cost as much as $7,000 dollars. Moreover, these systems are expensive to repair. Research has concluded that even minor incidents to vehicles with ADAS technology can add up to $3,000 in additional repair costs due to both replacement parts as well as meticulous recalibration [11].

Due to the success of ADAS systems, legislation will soon mandate some features on all consumer vehicles. On March 31, 2014, the US Department of Transportation's National Highway Traffic Safety Administration (NHTSA) required all new vehicles under 10,000 pounds (4,500 kg) to have rear view cameras by May 2018. According to the Institute for Highway Safety (IIHS), AEB reduces the rate of rear-end collisions by 39 percent [12]. In 2016, NHTSA and IIHS announced that 20 automakers had agreed to include AEB standard by September 1st, 2022. This accounts for more than 99 percent of the US automotive consumer market [13].

2.3 Complexity of Perception

Automated driving research followed soon after the invention of the automobile. In the 1920s, researchers experimented with radio-controlled cars [14]. In the 1960s, cable-controlled cars were introduced using magnetic cables embedded in the roadway [15] [16]. By the 1980s, experiments were being conducted with computers to automate driving, similar to ADAS systems of today.

Automation was solved soon after automobiles were introduced, but more was needed to reach true autonomy. Figure 2.2 outlines a high-level flowchart of an autonomous, robotic system. Research quickly developed the technology needed for motion control and primitive path planning but underestimated the complexity of perception and localization.

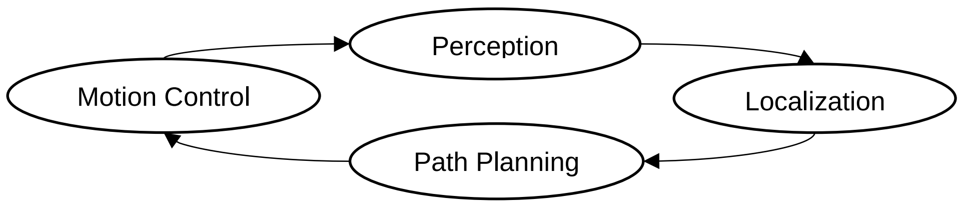


Figure 2.2: Autonomous System Flowchart

Perception and localization allow a robotic system to *perceive* the environment around it and *localize* itself within that environment. Perception and localization are critical for truly autonomous and “smart” driver assistance.

The perception system aims to develop a model of the ego-vehicle’s environment. To do this, the system uses a perception architecture which consists of the sensors, computers, and software. Architectures differ based on the function(s) the system performs. Modern ADAS use production-ready sensors that can be easily fitted and calibrated on the vehicle. Autonomous vehicles (AVs), on the other hand, may use more expensive, state-of-the-art sensors mounted on the vehicle body. Perception accounts for 94% of autonomous vehicle computing [17]. This comes as no surprise since researchers estimate that 30% of the human brain is used to solve this same problem [18].

2.3.1 Imaging and Ranging Sensors

The sensors used for perception can be classified into two groups: imaging and ranging sensors. Imaging sensors provide a high-resolution image of a portion of the environment. This is useful for classification or detection of objects within the image. Ranging sensors provide highly accurate point estimates of objects in their field of view (FoV). An overview comparison between sensor types is shown in Table 2.1 below.

Table 2.1: Comparison of Sensor Types

Sensor Type	Classification		Dx (Longitudinal)		Dy (Lateral)		Dynamics		Cost	
Camera	🟩	Pattern Recognition	🟨	Perspective Mapping	🟨	Perspective Mapping	🟨	Optical Flow Odometry	🟩	< $500
Radar	🟨	Radar Cross Section	🟩	Low Signal Attenuation	🟥	Low Resolution	🟩	Doppler Frequency Shift	🟨	< $5k
LiDAR	🟨	Point Cloud	🟩	Time of flight	🟩	High Resolution	🟨	Discrete Derivative	🟥	<$20k
Ultrasonic	🟥	Road Surface	🟨	Limited range	🟨	Limited range	🟥	none	🟩	< $500
*Green-best, Yellow-average, and Red-worst										

Cameras are the primary imaging sensor used in automotive and robotic perception systems. By themselves, cameras only record images and videos based on light exposure on an image sensor, but with image processing software the data becomes useful for automation. There is a wide range of software including neural networks, mappings, and optical flow to classify and localize objects within the camera frame. This will be discussed further in this chapter and this work.

Radar is a ranging method which uses radio wave reflectivity to measure distances between the sensor’s antenna and a metallic object in the emission path. There have been two main categories of automotive radars: short-range and long-range. Short-range radar (SRR) used a 24GHz signal and was primarily used for short range ADAS applications like blind-spot detection and collision avoidance. Modern radar sensors are based on long-range radar (LRR) technology, utilizing a 77GHz band. These sensors are compact due to a smaller antenna, which scales linearly with frequency. Radar uses several properties that allow it to detect, classify, and estimate states of objects. Radars can estimate longitudinal, lateral, and vertical position as well as well as classify the object using the signals returned radar cross section (RCS). Additionally, velocities can be measured using the Doppler Shift from the returned signal [19].

Light Distance and Ranging (LiDAR) is a detection method which uses visible light beams produced by lasers. The beams are reflected off objects and are returned to the sensor. This generates a point cloud of the surroundings, useful for surveying and mapping. Similar to radar, LiDAR calculates the time of flight of each light beam to determine the distance to the object that caused the reflection. LiDAR sensors have been adopted for autonomous vehicles because of the high-resolution localized map they create.

Ultrasonic sensors use a single element for both signal emission and reception based on the same time-of-flight principle used in radar and LiDAR, however, they use high frequency sound instead of radio or light. The speed of sound is orders of magnitude slower than the speed of light, at which radar and LiDAR signals travel [20]. This speed difference causes lower resolution and higher attenuation during transmission. For these reasons, ultrasonics are commonly used in close range applications such as backup assistance and blind spot monitoring. The amplitude of the returned sound, known as echo, can be used to filter invalid objects at these ranges. If more than one sensor is used, they must be spaced appropriately due to interference caused by multiple ultrasonic sensors in the same FoV. Using just one sensor can be appropriate for implementing some ADAS features, such as rear collision warning or lane departure warning, but oftentimes sensors are used together to increase the fidelity and redundancy of the environmental perception model. When two or more sensors are used together, the detections from each sensor must be combined into one data set using a process known as sensor fusion.

Perception software can be categorized into three categories: the detection layer, object layer, and target layer as shown in Figure 2.3. The detection layer consists of the raw detections from each sensor combined into one dataset with no additional processing. This creates point cloud data describing points of reflection from each sensor. The reflections can come from all objects within the FoV of the sensors, and there is likely no classification performed to differentiate the objects. In the object layer, a fusion algorithm is used to cluster point estimates from the dataset formed in the detection layer into objects. Objects have information like a unique tracking ID, tracking existence time, class, and trajectory. These fields are calculated using the combined information from all sensors in the set which can be filtered and processed to characterize objects more cohesively.

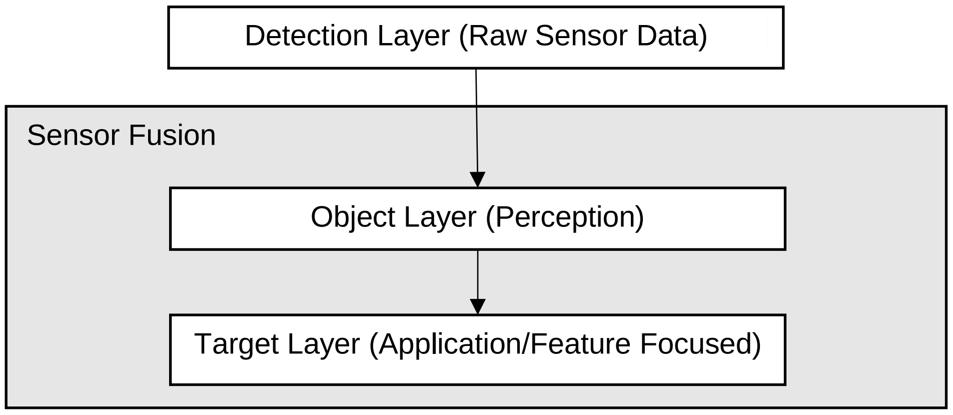

Detection Layer (Raw Sensor Data)
Sensor Fusion
Object Layer (Perception)
Target Layer (Application/Feature Focused)

Figure 2.3: Perception System Layers

The target layer filters objects further, using parameters specific to the assumptions made for a specific ADAS feature.

2.3.2 Classification and Object Detection

Classification is needed to organize the environmental model cohesively. This is useful in applications which can filter information based on what is relevant for that feature. For example, ACC behaves differently when a lead vehicle (the vehicle directly in front of the ego-vehicle) is present and when there is open road. As ADAS become more complex, classification adds additional reliability since an object is both located and classified. Classification is performed most commonly with camera sensors but can be done with radar and LiDAR as well.

Classification is an area of pattern recognition, a sector computer science which deals with algorithms that recognize and interpret patterns found in signals. At its simplest form, a pattern recognition algorithm uses a series of thresholds to filter data. In computer vision, a straightforward example would be characterizing images based on color. The algorithm could read in an image, sum up the red, green, and blue channels of each pixel in the image, and take the average for each channel. It could then compare this to a database of red, green, and blue values for different objects and see which object best matches the three average channel values. This would be a classification algorithm, developed entirely by an engineer hard-coding to identify three features, or dimensions, of classification: red, green, and blue color. This would form a relatively weak classifier, incapable of distinguishing objects of the same average color. However, the engineer could improve this classification by examining shape. This would be a tedious process in which morphological operations would be hardcoded to understand shapes within parts of the image. This is a more sophisticated algorithm that just average color, but the concept is still the same: hardcoding thresholds based on intuition of the size, shape, and color of an object.

Machine Learning is the concept of using large datasets to choose the thresholds instead of hard-coding them based on intuition.

With machine learning techniques, an engineer can develop an algorithm to perform classification. The algorithm would evaluate inputs based on a set of features, but the threshold values would be assigned based on iterative exposure to data. Machine learning has advanced the industry by optimizing classification parameters automatically. It has become a common practice in autonomous systems over the last decade. Modern machine learning uses more advanced techniques coined as *Deep Learning*. Deep learning builds on principles found in its basis model, the convolutional neural network. With this model, parameters can be tuned to work in any number of applications. Today, there are free, open-source deep learning frameworks from companies such as Google and Facebook for developing state-of-the-art machine learning algorithms. Figure 2.4 shows the mentions of deep learning frameworks in relevant machine learning literature over the years of 2014 to 2018.

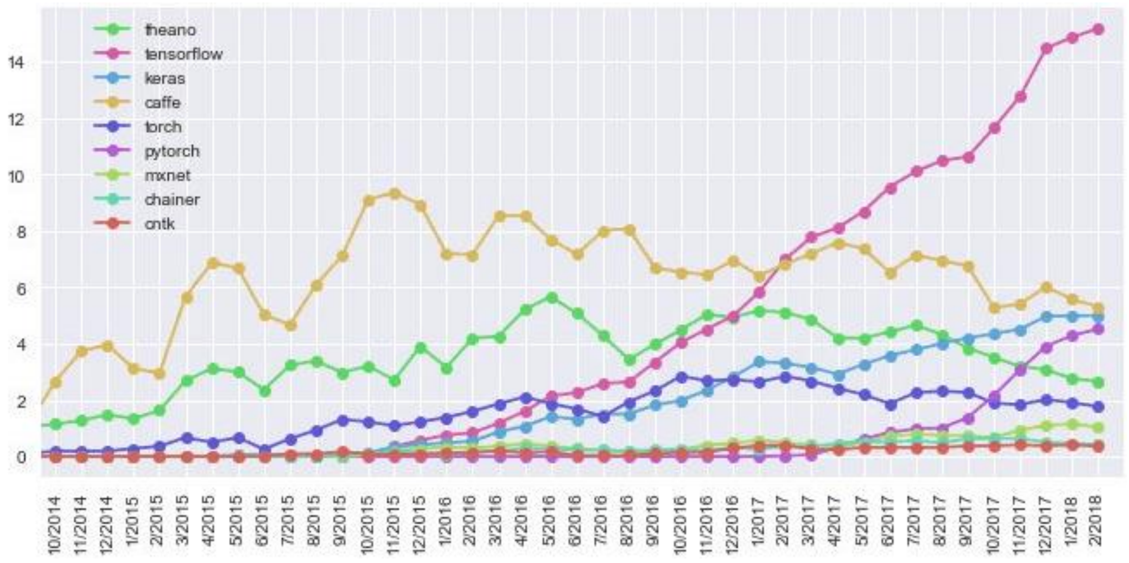


Figure 2.4: Growth of Deep Learning Frameworks in Terms of Thousands of Mentions in Literature [21]

Deep learning has issued a new wave in classification software. However, in computer vision, the object-of-interest is rarely the only object in the image frame. So, the object must first be localized within the image frame before it can be classified. This localization and classification problem is often combined in the computer vision technique known as *object detection*.

Object detection algorithms receive an image matrix as input and output a bounding box coordinate set that describes the four vertices of a rectangle which bounds the perimeter of an object in the 2D image plane. This output has one of four possibilities for a single object-of-interest, as illustrated in Figure 2.5.

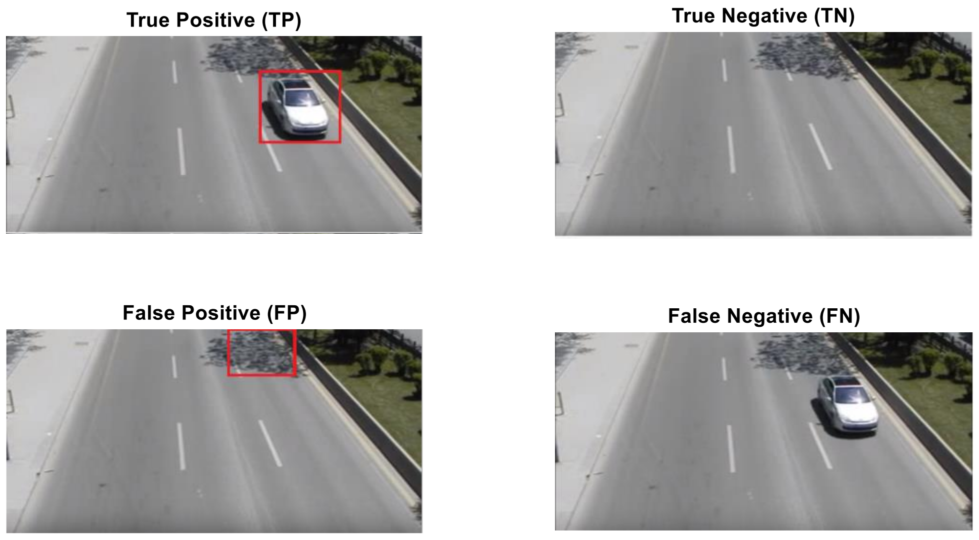


Figure 2.5: Object Detection Definition [22]

If an object-of-interest is present in the camera FoV, the algorithm has one of two options. The algorithm, with output bounding box shown in red, can detect and localize the object correctly, known as a true positive (TP), or it can miss the object altogether, known as a False Negative (FN). However, object detection differs from pure image classification in that there may not always be an object present. In this case, the object detector is expected to ignore these scenarios, without producing an output. This case is known as a True Negative (TN). Unfortunately, poorly designed object detectors can produce outputs even when none are present. This is known as a False Positive (FP). In the example in Figure 2.5, the detector accidently classifies the shadow from a tree as a vehicle. Computer vision engineers often design object classifiers to maintain as many true positives as possible while minimizing the number of false positives observed. This is known as the sensitivity of the detector.

2.3.3 Real-time

Perception systems must be able to maintain real-time processing for the environmental model. Real-time in this application is defined based on the time a human would take to react to a driving scenario and behave accordingly. According to research, the average driver reacts in about 100ms [17]. For an automated driving system, 100ms would include perception, localization, path planning, and motor control. Thus, the perception system should update its environment model at around 40Hz to allow time for other processes to use the data to make decisions. Innovations in parallel computing machinery have allowed perception systems to offload computation to maintain real-time performance.

2.3.4 Emphasis on Simulation

To meet reliability expectations, a fleet of autonomous cars would need to demonstrate several billion miles of on-road testing to ensure, with confidence, that the autonomous system is safe [23]. This would take most companies hundreds of years to accomplish, due to the timely cost of driving so many miles in real-time. Currently, Google's Waymo leads the industry with over 10 million miles driven on road [24]. Simulation plays a role in testing, as it can be performed more rapidly, safely, and cheaply then driving on roads. Google also boasts 5 billion miles driven in simulation, which increases the reliability of their system.

2.4 Summary

The industry is moving closer towards autonomy with several automakers claiming to reach zero accidents in the next decade using these technologies [25]. Autonomous vehicles has been a prominent research area since nearly the invention of automobiles but is now nearing production use. Industry and research are using the computer vision technologies discussed in this work as part of the sensing software needed to create perception systems for ADAS and AV applications.

Chapter 3: Experimental Design and Process

The perception system techniques discussed in this work were experimentally implemented with production-level intention. Thus, much of the design process was focused on vehicle fitment and integration. The experimental design and setup of the sensors, embedded computers, and supporting hardware is provided in this section as context to the software techniques to be discussed. This was done through system requirements derived from the project scope and goals, a component selection process, and vehicle fitment. An overview of the software development workflow will also be provided.

3.1 System Requirements

Based on the project goals outlined in Chapter 1, the team developed a set of ADAS consumer features that could be implemented within the constraints of a SAE Level 0 automated system. It was decided that lane departure warning (LDA) and Forward Collision Warning (FCW) would be effective safety-oriented features. Additionally, an open-loop implementation of an adaptive cruise control (ACC) was to be implemented. This feature would display a difference between the current headway distance between the ego-vehicle and immediate forward-facing target and the ideal headway distance based on the current speed. This feature, coined Recommended Following Distance (RFD), provided the driver with a live measurement feed to which the driver could alter their throttling behavior to minimize the delta in headway distance as shown on the HMI display. These ADAS features were outlined in terms of component types: sensors and HMI styles. Using product breakdown structures (PBS), as shown below in Figure 3.1, it was determined that all three features could be implemented using only a camera and radar sensor and a graphical interface for visual driver feedback.

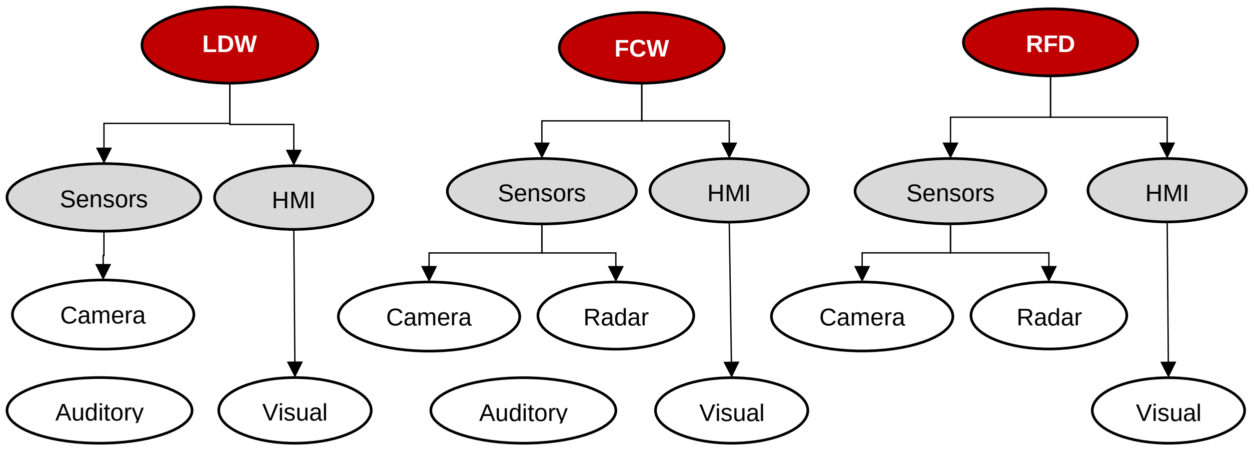


Figure 3.1: Product Breakdown Structures of Desired ADAS Features

LDW, FCW, and RFD formed a simple yet sufficient set of ADAS features that worked within the constraints of the project while providing useful technology for vehicle operation. Next, components were selected that provided sensing, computing, and HMI functionality needed to implement the selected ADAS features.

3.2 Component Selection

Camera and radar sensors were surveyed for selection. Cameras were evaluated based on interfacing capability, frame-rate, image resolution, and distortion characteristics. Radars were evaluated based on interfacing capability, longitudinal range, and FoV. Interfacing capability was limited to I/O interfaces available on computational hardware that the team planned to use. Although computational hardware was selected after sensing components were finalized, restrictions on the sensor I/O was limited to CAN, USB, and ethernet. The system was required by logging specifications to identify target vehicles at least 100m away. Furthermore, the entire frontward environment was within testing limits, so a large FoV was desired for both sensors. The selected sensor set consisted of a Delphi Electronically Scanning Radar (ESR) and a Spinel USB mono-camera. A Bird’s Eye Plot illustrating the system’s FoV and range is shown in Figure 3.2.

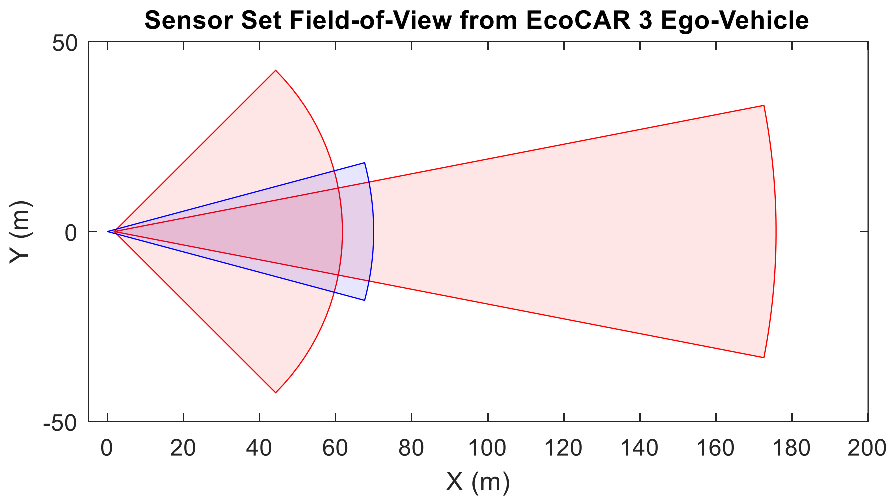


Figure 3.2: Bird's Eye Plot of System FoV with Camera (blue) and ESR (red)

As shown in the bird's eye plot, the sensor set has three vision cones, formed by the two individual antenna bands embedded in the ESR module and the single lens of the mono-camera. The antenna bands on the ESR provided a large longitudinal and lateral FoV at up to 174m at 22° (+/- 11° from boresight) and 60m at 90° (+/- 45° from boresight). An image of the ESR is shown in Figure 3.3. This sensor provided enough lateral visibility for overtaking vehicles in adjacent lanes and ample longitudinal visibility for vehicles in the host-lane.

Figure 3.3: Delphi ESR 2.5 [26]

The camera chosen for this project used a 2 mega-pixel (MP) CMOS imaging sensor designed for full high definition (HD) streaming at 30 frames per second (fps). The lens included an IR-cut filter at 850nm, designed for day and night use. Per experimental measurements with team-developed software and calibration equipment, it was determined that the camera was capable of 30° FoV and 70m range. The range was highly dependent on image compression needed for real-time object detection. For instance, at full HD resolution, the object detection algorithm performed better, as more pixels were used to represent the object at far distances. However, with a compressed, 720 by 480 resolution, the range of the object detection algorithm diminished to 70m but was real-time implementable due to the decrease in data throughput.

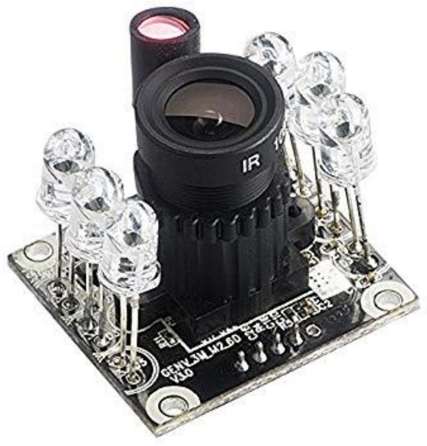


Figure 3.4: Spinel 2MP Full HD USB Camera [27]

The ESR and camera were evaluated, and a database of radar data and camera data was collected for real-time playback. Computational requirements were outlined based on throughput metrics of each sensor. Software development toolchains were evaluated using virtual machines (VMs) to model hardware options before purchasing. Due to the high data-throughput of imaging sensors, and the parallel, graphics optimization used in computer vision, it was determined that the computer vision pipeline should be separated, physically, from the rest of the ADAS software.

Figure 3.5: NVIDIA Jetson TX2 [28]

An NVIDIA Jetson TX2, as shown in Figure 3.5, was used for computer vision. This was a low-power, mobile Linux device designed for AI and robotics. It uses an NVIDIA Tegra Parker system-on-a-chip (SoC) with two central processing units (CPU), a graphics processing unit (GPU), and image signal processor (ISP) integrated onto one piece of silicon. This allows the development kit to be minimal in size with centralized cooling for the quad-core ARM Cortex A57 CPU, dual-core NVIDIA Denver 2 CPU, 256-core NVIDIA Pascal GPU, and on-chip 1400 MP/sec ISP.

As mentioned, the TX2 was used to process the computer vision pipeline of the mono-camera, imaging sensor. Images were processed off USB using the Tegra ISP and queued in memory. The computer vision pipeline used a collection of algorithms to pre-process the image, perform object detections, update tracking information, and output the data to the ADAS supervisory controller. This limited the scope of the TX2 to just computer vision, which allowed the ADAS supervisory controller, an NXP S32 Evaluation Board (EVB), to perform in real-time without latency issues from camera processing.

The EVB, shown in Figure 3.6, served as the ADAS supervisory controller, tasked with radar processing, ego-vehicle status updates, and sensor fusion. It included two CAN interfaces as well as ADAS specific processors. This was useful for vehicle interfacing and radar processing as CAN was needed for both reading and writing to the CAN bus. Similarly, the ADAS processors were used for processing ESR data and converting it from supplier object-level detections to application-specific targets, as expected for the ADAS features the system was to perform.

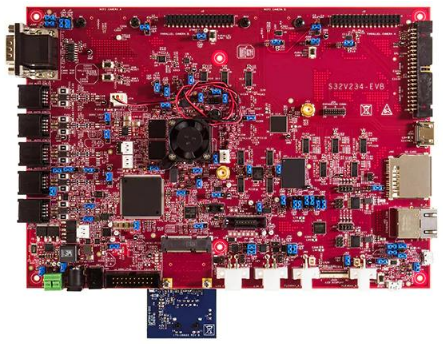


Figure 3.6: NXP S32V234 EVB [29]

THE EVB communicated with the ego-vehicle through the hybrid supervisory controller. This controller oversaw all vehicle controllers and determined how to convert driver inputs into torque requests for individual powertrain components. A dSPACE MicroAutoBoxII (MABx) served as the hardware for the hybrid supervisory controller which interfaced with the EVB for startup, shutdown, diagnostics, and HMI interaction. Additionally, a human machine interface (HMI) was developed using hobbyist electronic components consisting of an Arduino Mega as the HMI controller and an LCD matrix and LED light strip as the display. Characteristics of the display such as colors, frequencies, and patterns followed NHTSA recommendations for driver interaction devices [2]. The display was controlled through the MABx, which relayed messages from the EVB to the HMI controller during ADAS operation.

3.3 Vehicle Integration

Each component was fitted into the ego-vehicle to meet production-ready expectations. Components, especially sensing components, were well hidden from view, which was done through calibration and testing.

3.3.1 Sensing Components

Sensor placement was based not only on characteristics like FoV, vibration sensitivity, and signal attenuation, but on consumer expectations of ADAS systems. The ESR was placed on the front bumper behind the fascia, as shown on the right in Figure 3.7. Fitment testing was performed with and without the front fascia obstructing the sensor antenna. Comparing the data from each test, it was not apparent that the fascia attenuated signals enough to affect the internal filtering software on the ESRs embedded controller.

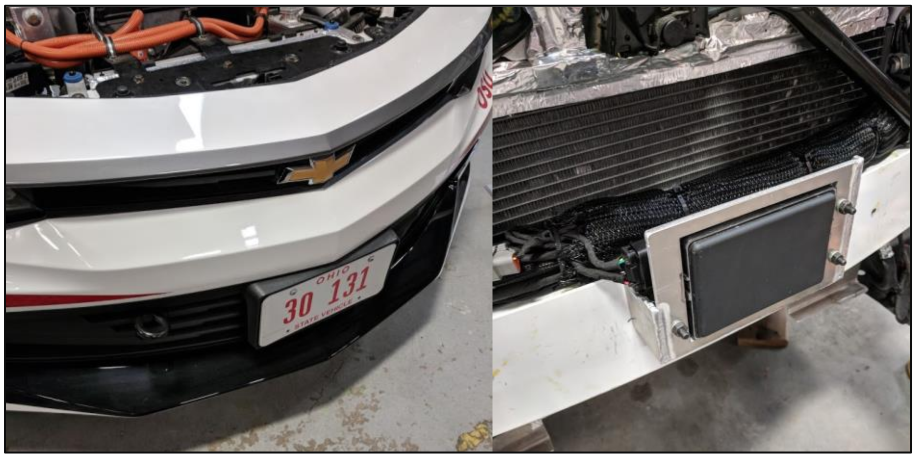


Figure 3.7: ESR fastened to the front bumper, behind the fascia [22]

The mono-camera was mounted behind the rear-view mirror, a commonplace for imaging sensors in industry, because it provided protection by the cabin as well as a transparent viewing window from the windshield. A custom, 3D printed mount with a 3D printed lens hood was used to decrease glare coming through the windshield. This mount was made by modifying the stock vehicle computer-aided design (CAD) part for the rear-view mirror cover and re-printing it with mounting standoffs. The apparatus is shown in Figure 3.8.

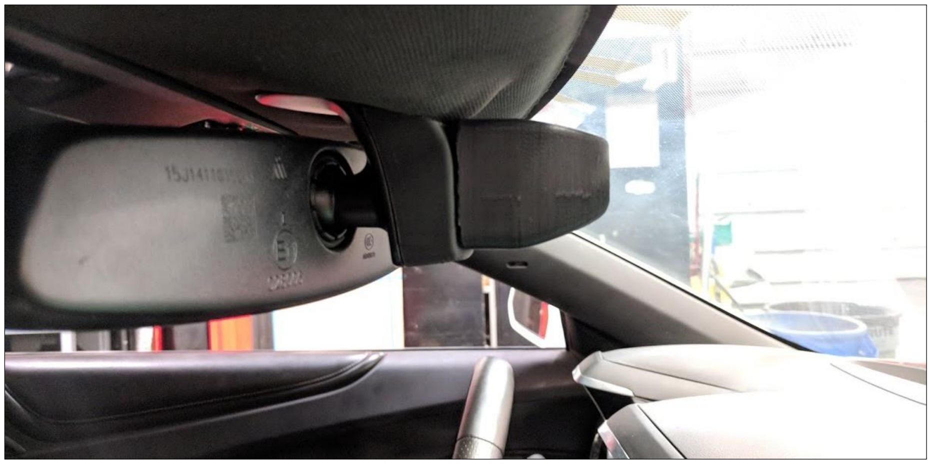


Figure 3.8: Camera Mounting Placement [22]

This location optimized the height as much as possible without exceeding the vertical dimension of the cabin. However, there was still loss in FoV due to the hood. This was measured to be about 3.8m of longitudinal distance from the tip of the fascia but was determined acceptable since the low height of the Camaro and large front hood would ultimately cause visibility issues for any camera mounting location. With this limitation, the sensor range was restricted from 3.8m to 70m with a 30° FoV. However, the blind spot produced at less than 3.8m was covered by the radar.

3.3.2 Controllers

ECU mounting locations were determined with space claim checks, including air-flow needs for cooling. The TX2 and EVB were placed behind the rear bench seats, fastened to the top of the shelf. The MABx was in the left side of the trunk behind a taillight. The TX2 and EVB includes fans for active integrated circuit (IC) cooling. The ECUs were housed in aluminum enclosures which included orifices for the fans. The enclosures were oriented such that air ducts were present at the openings of the enclosure orifice to provide airflow.

3.3.3 HMI

The HMI device was a custom “driver display” located above the stock infotainment system. The display used an LCD matrix and individually-accessible RGB LED strip housed in a 3D-printed case. The display doubled as a vehicle status bar as well as an ADAS feedback alert system. When the ADAS system was turned off, the driver could use the stock paddle shifters to select between driver menus such as mpg, remaining all-electric range, and full range. However, flipping the overhead ADAS switch converted the display into ADAS mode. The paddle shifters could similarly be used to select ADAS features such as RFD and LDW. This is shown in Figure 3.9.

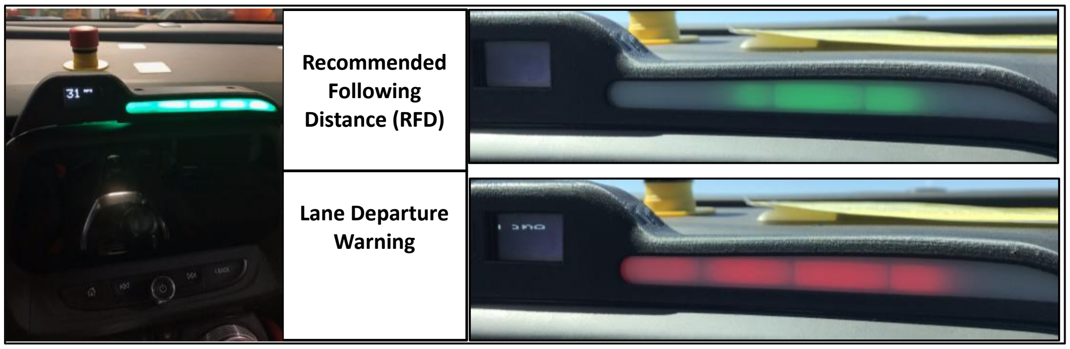


Figure 3.9: Driver Display [22]

FCW was not set as a selectable mode; Instead, it preemptively overrode the display during safety-critical scenarios.

3.4 Software Development Process

Software was developed by a team of ten students. To organize code and projects, the web-based project management software, Asana, was used to assign tasks using agile development methodology. Additionally, Git version control software and a Github-hosted repository was used to combine code from team members and perform unit and integration testing. Regular version releases were performed once vehicle testing became frequent.

Code was written in C++ for radar processing, CAN communication, and sensor fusion. This was the language of choice since the S32 EVB came preloaded with a custom C++ compiler included out of the box. Code could also be written on another machine and cross-compiled for embedded use. For computer vision, Python 2.7 was used with the OpenCV 2.4.13 library due to the wide availability of machine learning modules compatible with the Python platform.

3.5 Summary

Experimental design and process have been discussed in terms of requirements, component selection, vehicle fitment and integration, and the software development process. The sensors and computing hardware were used to evaluate the software, discussed in Chapter 4:, experimentally on both a test bench and, later, in a prototype vehicle. The software development process details the libraries and programs used to create the computer vision software deployed to the embedded hardware.

Chapter 4: Computer Vision Techniques

Computer vision techniques were used to process the digital images returned by the camera module discussed in Chapter 3:. Without computer vision and image processing techniques, a digital image contains no useful information to a computer. Most of the software development for this project went towards implementing a set of algorithms to transform raw pixel data, such as the example below in Figure 4.1, into a set of 3D coordinates approximating the real-world location of objects within the image.

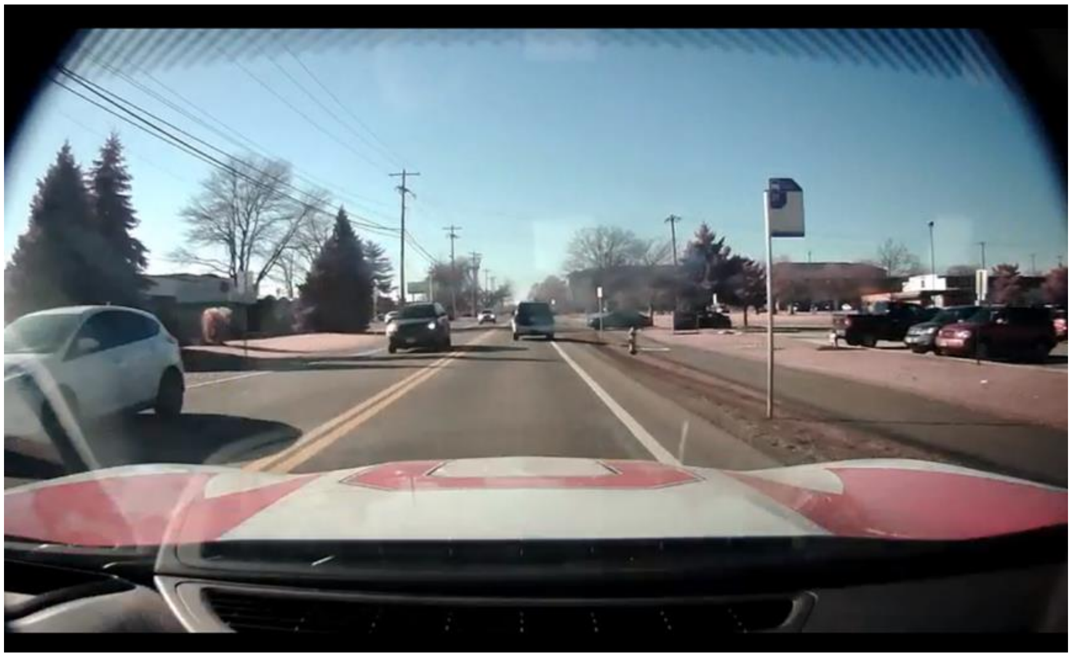


Figure 4.1: Sample Image from Ego-Vehicle Camera

The computer vision pipeline accomplished this by classifying target vehicles, tracking their position over time, and estimating their ego-vehicle coordinate position. Additional functionality was implemented to provide sensor fault diagnostics. This chapter will discuss the computer vision techniques used to develop these algorithms.

4.1 Camera Modeling

Camera modeling was used to define a relationship between objects in 3D space and the 2D image captured by the camera. This was useful because it defines the physical barrier between it and the real world; namely, the sensor face. Cameras are modeled using both intrinsic and extrinsic parameters. Intrinsic parameters define the topography of the image sensor which is fixed for each module as well as the lens. The topography includes the retina, the face of the imaging sensor. In this project, the retina was a grid of complementary metal-oxide-semiconductor (CMOS) image sensors in a 1/2.7" optical format. Intrinsic parameters include *l* and *k*, the height and width of the pixel in 1/m, which are used as a meters-to-pixels conversion. An angle, ϴ, represents the skew of each pixel. This is normally used to compensate for manufacturing defects in which the pixels may not be perfectly square. For simplicity of this model, the constraints of square pixels and zero skewness are placed in Equation 1 and Equation 2, respectively.

$$l = k = \frac{1920 pixels}{0.00537 m} = 357541.9 \frac{pixels}{m} \approx \frac{1}{3.0\ \mu m} \tag{1}$$

$$\theta = 90° \tag{2}$$

A visual representation of one pixel in the retina is shown in Figure 4.2. The mono-camera used in this work used a 2MP CMOS imaging sensor, thus a two-million-pixel grid of the square pixel below was used to model the entire surface of the retina.

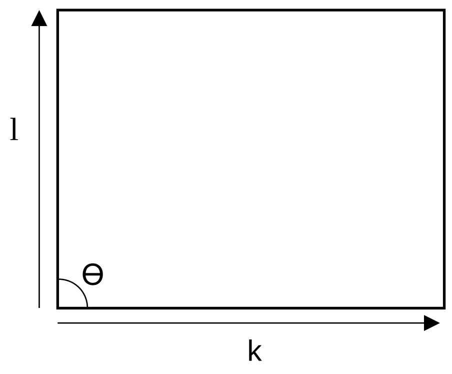


Figure 4.2: Pixel Definition

Additional intrinsic parameters, such as the focal length, can be varied using a manually adjustable lens. However, in this work, the lens was kept constant after it was calibrated for optimal focus. Extrinsic parameters include three-dimensional rotation and translation of the camera module. It is not dependent on the camera’s internal build, but on the placement and orientation of the camera. There are several camera models that use intrinsic and extrinsic parameters to define the behavior of a camera, but the most popular for computer vision applications is the camera pinhole model.

The pinhole model characterizes perspective projections and affine projections, only, as transformations from $\mathbb{R}3$ to $\mathbb{R}2$. A perspective projection is observed when two parallel lines in 3D space converge towards a “vanishing point” in 2D space. For example, when a road converges into a single point toward the middle of a camera image. Conversely, an affine projection preserves parallel lines by only scaling, rotating, translating, mirroring, and shearing the 2D plane of an image. Examples of perspective and affine projections will be provided in a later section.

The pinhole model does not consider lenses, which only complicate the model and aren’t very useful in computer vision. Instead, an infinitesimally small “pinhole” located at the origin of the 3D space is used to represent the opening of the camera. This hole controls exposure to the imaging sensor, allowing light to enter the cavity and illuminate the retina. In practice, we can't assume that the aperture for the camera is as small as this pinhole, but this is rarely a concern in computer vision applications.

For a point P, in real world 3D coordinates of a Euclidean vector space, the model describes a point P' in the 2D image frame of reference define by the basis vectors $\hat{u}$ and $\hat{v}$. This is shown in Equation 3. Note, the equations provided in this section do not follow the vehicle coordinate system outlined in Figure 1.3.

$$P := \begin{bmatrix} x \\ y \\ z \end{bmatrix} \rightarrow P' := \begin{bmatrix} \mathrm{u} \\ v \end{bmatrix} \tag{3}$$

This is known as a projective transformation since P is projected onto the 2D plane created by u and v. In this model, the focal length, *f*, is defined as the orthogonal distance between the retina and the pinhole, measured along the *z* direction, known as the *optical axis*. The simplest approximation for P' is given by Equation 4, which uses only the aspect ratios (k and l) to transform P into P' on the image.

$$P' = \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \frac{kfx}{z} & \frac{lfy}{z} \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \frac{\alpha x}{z} & \frac{\beta y}{z} \end{bmatrix}^{\mathrm{T}} \tag{4}$$

This uses a fundamental perspective projection based on the ratio between the focal length and z, the distance between P and the pinhole along the optical axis. This is what scales objects, making distant objects appear smaller than closer ones in an image. Most digital image frames define the top-left corner of the image as (0, 0) for indexing. Thus, we can compensate for this offset in the model by adding the digital origin (u_0, v_0). Likewise, we can account for skewness using Θ. These compensations are provided in Equation 5.

$$P' = \begin{bmatrix} \frac{\alpha x}{z} - \alpha \cot(\theta) + u_0 & \frac{\beta y}{z sin(\theta)} + v_0 \end{bmatrix}^{\mathrm{T}} \tag{5}$$

Due to the division of the input parameter *z*, Equations 4 and 5 are nonlinear. However, a homogenous coordinate system transformation will linearize the relationship between the output P' and the input P. Equation 6 redefines P and P' as follows:

$$P := \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \to P' := \begin{bmatrix} u \\ v \\ z \end{bmatrix} \tag{6}$$

Where x is the same in both P and P' since the *x* dimension cannot be mapped directly to a 2D plane. Using the newly defined P and P', a linear relationship between P and P' is outlined, as shown in Equation 7.

$$P' = \begin{bmatrix} u \\ v \\ z \end{bmatrix} = \begin{bmatrix} \alpha & -\alpha\cot(\theta) & u_0 & 0 \\ 0 & \frac{\beta}{\sin(\theta)} & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \mathrm{MP} = \mathrm{K}[I \quad 0]P \tag{7}$$

Lastly, P'=MP can be expanded to account for real world coordinates, P, that are in a different frame of reference, such as a point in the world reference frame denoted as P_W. A point P_W would need an additional transformation matrix to account for the coordinate transformation into the camera reference frame and then into the image reference frame. This transformation matrix couples as the extrinsic parameters of the camera which takes place of the homogeneous identity matrix [I 0]. This is shown in Equation 8.

$$P' = MP = K \begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} P_W \qquad [8]$$

This is the derivation of the camera pinhole model. It provides a linear transformation between any real-world coordinate and its 2D representation on the image plane. Furthermore, the intrinsic and extrinsic parameters used to model the camera behavior are presented, compactly, in one coefficient matrix, M. This is a powerful model with 11 degrees of freedom formed by 5 intrinsic measurements, 3 extrinsic rotations, and 3 extrinsic translations [30]. It will be discussed in this chapter as a solution for removing distortion, perspective mapping, and object depth estimation.

4.2 Camera Calibration

All intrinsic and extrinsic parameters captured in *M* can be recovered, stochastically, using calibration techniques. One such technique used in the *MATLAB Computer Vision Toolbox* uses a calibration pattern, a checkerboard of black and white squares, to determine the parameters of the camera [31] [32]. The technique does this by determining the relationship between the camera pixels in the camera frame of reference and the width of the calibration pattern squares in world coordinates. It is based on the pinhole camera model with an addition model for lens distortion. In this work, an 8x8 calibration pattern of 2cm x 2cm black and white was used. A digital image of the pattern is shown in Figure 4.3.

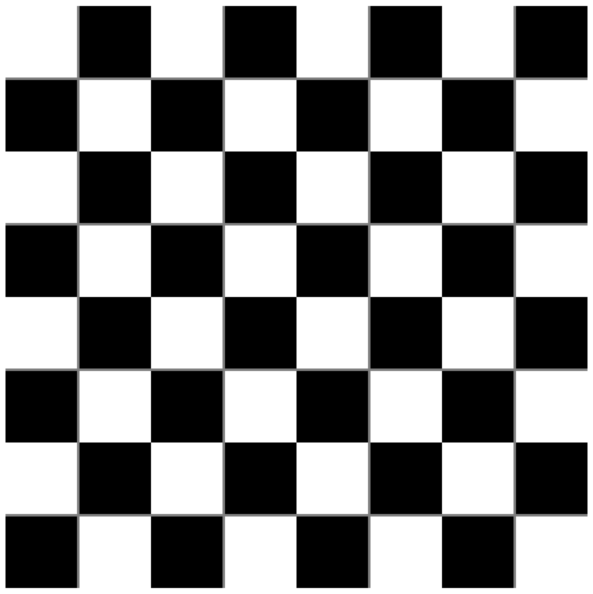


Figure 4.3: Camera Calibration Pattern [32]

A set of 40 images were taken of the face of the calibration pattern in the camera's native resolution. The images were then reduced to the size used for the computer vision pipeline: 480x640. The Camera Calibrator application interface in MATLAB was then used to estimate the camera parameters based on the world coordinate estimations of the calibration pattern in the images of the calibration set. This is performed in two stages. The first estimates the rigid transformation needed to map a pattern point in world coordinates to a point in camera coordinates.

$$P_w \coloneqq \begin{bmatrix} P_{w_x} \\ P_{w_y} \\ P_{w_z} \end{bmatrix} \rightarrow P \coloneqq \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \quad [9]$$

The second stage estimates the projective transformation from the point, P, in the camera coordinate space to a pixel, P', in the 2D image coordinate system.

$$P \coloneqq \begin{bmatrix} P_x \\ P_y \\ P_z \end{bmatrix} \rightarrow P' \coloneqq \begin{bmatrix} u \\ v \end{bmatrix} \quad [10]$$

Using the calibrator, this operation is performed iteratively over all internal, or shared, corners in the squares of the calibration pattern. For the 8x8 pattern used in this work, this would amount to 49 world coordinate estimates per calibration image. Each corner point, Pw, is estimated from P' in the image. Then, with the estimated camera matrix, M, the point, Pw, is reprojected back into the image using Equation 8 of the camera pinhole model. The reprojection error between the estimate of P' and the known pixel location of P' can be compared and the pixel error can be calculated based on the Euclidian planar distance. Of the 40 images in the calibration set, 29 were determined to be low in average pixel error to use in the calibration. A bar graph of the mean reprojection error per calibration image is shown in Figure 4.4.

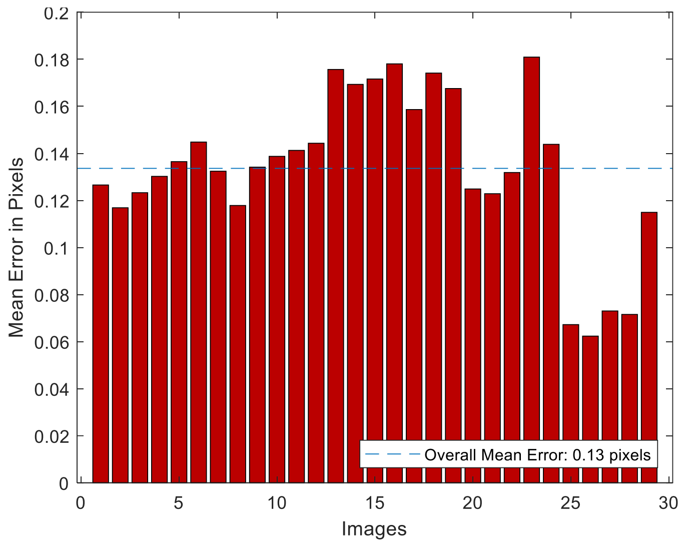


Figure 4.4: Mean Reprojection Error per Calibration Image

The overall mean error was 0.1336 pixels, which is in the recommended range of error when using this calibration technique [32]. For verification, we can recreate a model of the calibration data set using the estimations of the calibration pattern defined by the intrinsic and extrinsic parameters obtained by the calibration. This is shown in Figure 4.5.

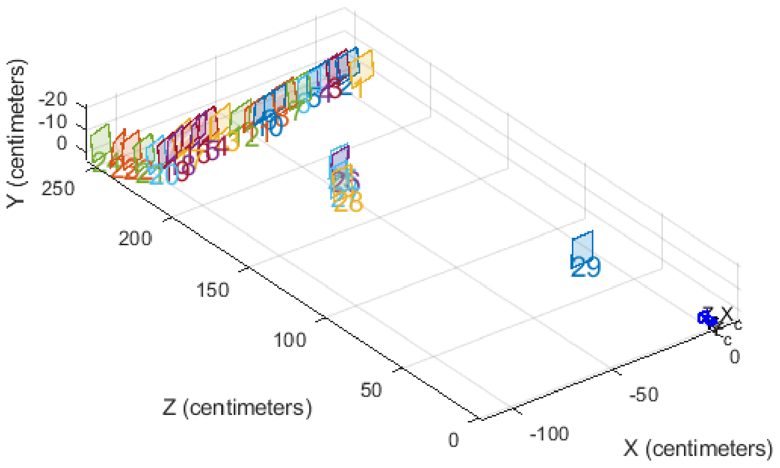


Figure 4.5: Camera-Centric Calibration Pattern Visualization

This is then compared to the image set to ensure that the calibration pattern location in the images used in the calibration matches the visualization. This visualization also demonstrates how the camera pinhole model can be used for depth estimation, which will be discussed further in this chapter.

$$Focal\ length\ (pixels):\ \ [\ 536.0757\ +/-\ 31.4792\quad 715.1305\ +/-\ 42.1131\ ]$$

$$Principal\ point\ (pixels):[\ 361.9890\ +/-\ 15.0390\quad 223.7204\ +/-\ 6.4365\ ]$$

$$Radial\ distortion:\quad [\ -0.3776\ +/-\ 0.0263\quad 0.1422\ +/-\ 0.0654\ ]$$

After validation, the calibration data output by MATLAB can be placed in the camera matrix, M. Recall from Equations # and #. In this project, an additional coordinate system was not used to define objects. The origin was defined as the location of the camera in 3D space, thus Pw=P.

$$M = K\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 536.0757 & 0 & u_0 & 0 \\ 0 & 715.1305 & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} I & 0 \\ 0 & 1 \end{bmatrix} \qquad [11]$$

4.2.1 Removing Distortion

The calibration also produced lens distortion parameters to combat radial distortion caused by the lens. By design, the optical center of the lens has the least amount of distortion. However, depending on the lens, distortion can occur near the edges of the lens due to the optical characteristics of the lens. An example of undistorting an image is shown in Figure 4.6. Image correction was not used in the deployed version of the project software because the correction limited the FoV.

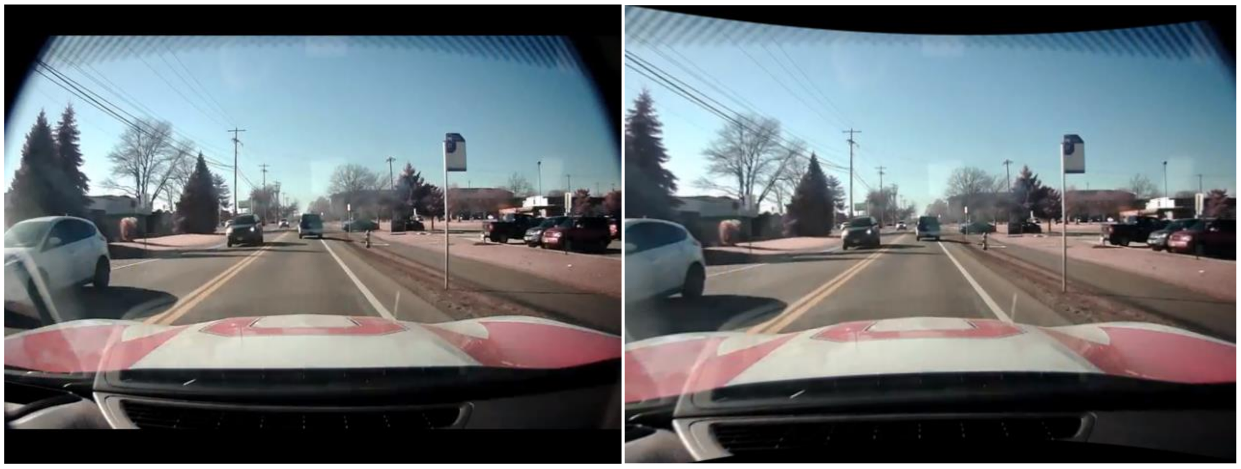


Figure 4.6: Distorted Original Image (Left), Undistorted Image (Right)

In this example, the white, oncoming SUV becomes cropped in the image once distortion is corrected. The distorted image proved more useful because it maintained a larger FoV.

4.2.2 Depth Estimation

The calibrated camera matrix, M, was also applied to monocular depth estimation to predict the real-world coordinates of objects in the image plane. This was done using a derivation of the camera pinhole model which uses the relationship between the expected width and observed width of an object to proportionally estimate its longitudinal depth through the image plane. For example, Figure 4.7 shows an image in which the region occupied by an object-of-interest has been bounded by a “bounding box”, a common localization technique which will be discussed later in this chapter. This bounding box technique provides an estimated width and height of the object in pixels.

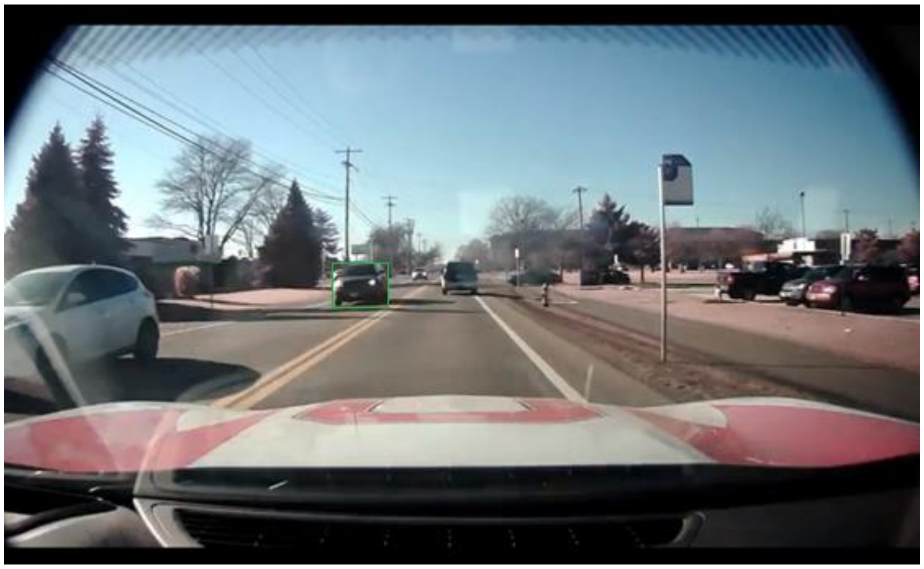


Figure 4.7: Depth Estimation Using Calibrated Camera Model

A linear range estimate is determined based on the proportion of an objects expected width to its observed width within the image. The expected width of a vehicle was set at 1.8m, thus the estimated distance of the object in Figure 4.7 is derived in Equation 12 [33].

$$\hat{T}_{Range} = \left(\frac{T_{width_{expected}}}{\hat{T}_{width_{observed}}}\right) f = \left(\frac{1.8}{35}\right) 536.0757 = 27.6m \qquad [12]$$

There is an effective range associated with this estimate, due to the nonlinear distortions caused as objects approach the vanishing point. For instance, a small error in the estimated width of an object as it nears the vanishing point will cause as much as 50% to 100% in error. This was not problematic for this project as the object detection methods used were generally inoperable outside of 60m. However, vision systems with higher resolution and detection range would need a more robust depth estimation algorithm to avoid estimate error at farther distances. Bounding error did cause fluctuation in depth estimate by as much as 20%. Semantic segmentation offers the best approach to reducing this error since the bound uses color and shape information to outline the object precisely, without generalizing it to just a rectangle. Due to time constraints, segmentation was not used in this work.

4.3 Color Spaces

The data stored in a digital image file can be used to represent the image, but only within a specific color space. The most common color space is Red Green Blue (RBG). Which uses linear combinations of red, green, and blue to represent each pixel in the image as a color found in the RGB color space. Figure 4.8 illustrates the 2D image stored as a 3D matrix with channels for the red, green, and blue components that make up each pixel value.

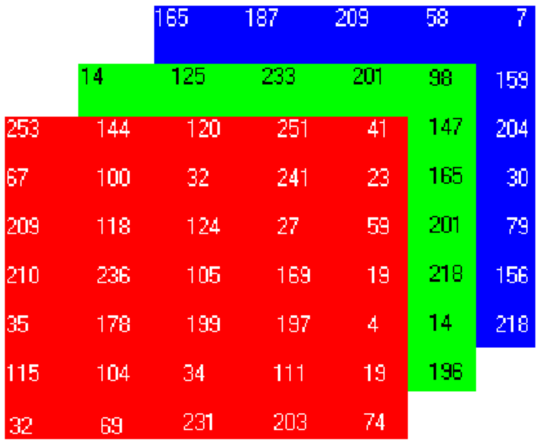


Figure 4.8: Image as an RGB matrix

The RGB color space is commonly used in computer vision because the channels can be separated and recombined as linear combinations. The top row of Figure 4.10 illustrates how a sample RBG image can be separated into its three channels while the bottom row demonstrates how the channels can be combined to recreate the original image.

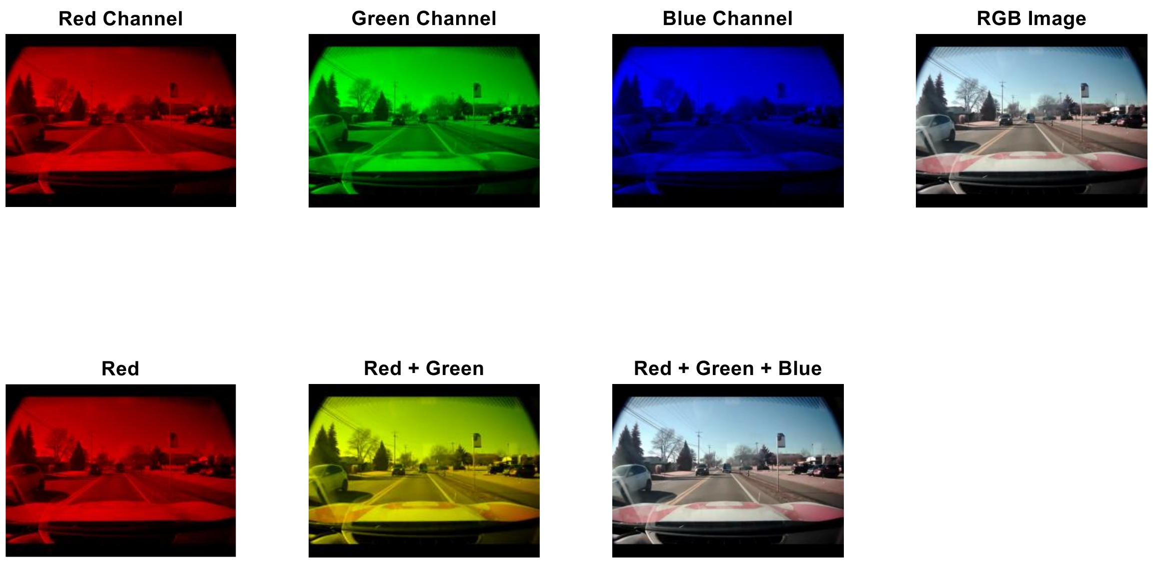


Figure 4.9: Red, Green, and Blue Channels of an RGB Image

RGB is useful for this reason, but it is not the only color space. For instance, Figure 4.10 shows the sample image from Figure 4.1 represented in RGB, HSV, YCbCr, and CIELAB (L* a* b*) color spaces.

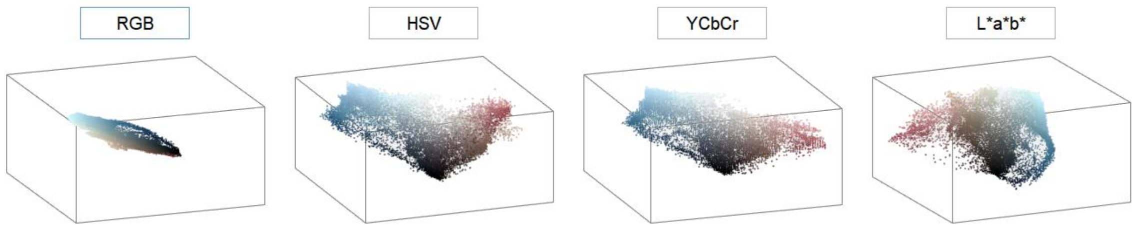


*Figure 4.10: Visualization of Sample Image in RBG, HSV, YCbCr, and L*a*b Color Spaces*

Notice the separability of the HSV, YCbCr, and CIELAB spaces compared to the RGB. Classification, based on color, is far more successful in the other three spaces as compared to

RGB since the object-of-interest can be better isolated in the three-dimensional space. For instance, red can be isolated using thresholds placed in the HSV space more so than thresholds in the RGB space. This is because, in RGB, nearly every object has red, green, and blue components. In the sample image in Figure 4.10, the objects are nearly all recognizable even when only looking at one of the three channels. In a later section, this concept will be demonstrated using yellow and white lane lines.

From the color space, we also derive the size of the feature space. A feature space represents the dimensionality of the expected input variable, in this case an image, for an algorithm. For RGB images, the feature space contains any image that can be created within the parameters of the image size, and color space used. In this project, RGB images were used at a resolution of 480 by 640. At this size, there are 480x640=307,200 pixels. Each pixel is represented as three 8-bit unsigned integers, one for each channel. Thus, each pixel can take on 255*255*255=16,581,375 values. This produces a feature space of 307,200*16,581,375 = 5.0938e+12 possible images that can be produced at this resolution and color space.

Most images in the space are noise, but theoretically, all 640x480 color images of every person, place, or thing exist within this feature space. To understand the content of the images more clearly, feature extraction techniques were used to reduce the dimensionality of the feature space and isolate areas of the images that were relevant to the application.

4.4 Image Pre-Processing Algorithms

Image pre-processing algorithms were used to transform raw camera frames into uniform image sequences that could be fed into the computer vision pipeline. In general, image processing and computer vision are highly related terms, but are not usually synonymous. Image processing algorithms are used to transform images but make no attempt to understand the image contents or perform inference. Computer vision, on the other hand, may use image processing techniques but will introduce logic to draw conclusions based on the data presented.

4.4.1 Affine Transformation

Affine transformations are used to rotate, scale, shear, and translate images. These transformations preserve points, straight lines, and planes in terms of shape but not magnitude (distance). They do this by keeping parallel lines parallel, however, the length of the line can still change. Figure 4.11 shows the output of a checkerboard image and a sample image using an identity affine mapping for reference against other mapping functions in this section.

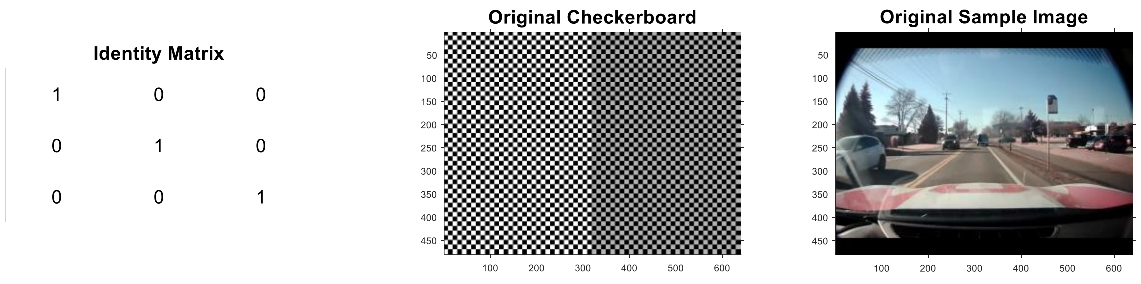


Figure 4.11: Baseline Checkerboard and Sample Image

Rigid transformations allow for six degrees of freedom in three-dimensional Euclidean Space. A rigid body transformation cannot alter the Euclidean distance, or 2-norm, between any two points in the space. In 2D, similarly, a rigid transformation has three degrees of freedom: two translational and one rotational. Rigid transformations, translation and rotation, are a subset of Affine transformations, shown in Figure 4.12 and Figure 4.13, respectively.

Translation Matrix

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 75 & 100 & 1 \end{bmatrix}$$

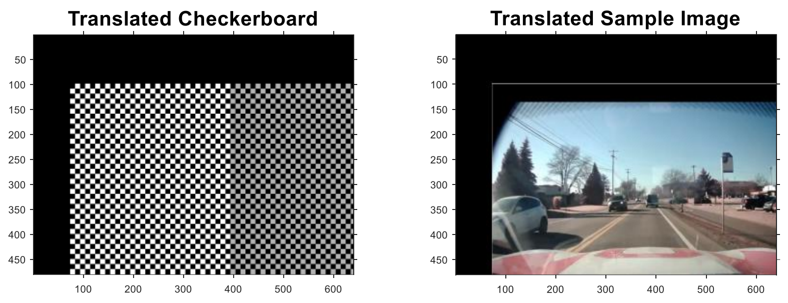


Figure 4.12: Checkerboard and Sample Image after Translational Affine Transformation

Rotation Matrix

$$\begin{bmatrix} 0.98079 & 0.19509 & 0 \\ -0.19509 & 0.98079 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

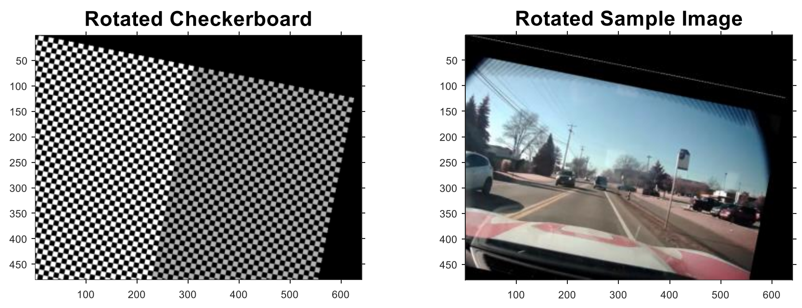


Figure 4.13: Checkerboard and Sample Image after Rotational Affine Transformation

As mentioned, affine transformations only promise to maintain parallelism between lines in the image. Thus, all rigid transformations are affine but not all affine transformations are rigid. For example, scaling and shearing are affine transformations which compress images vertically, horizontally, or any linear combination of the two. Figure 4.14 and Figure 4.15 demonstrate scaling and shearing transformations, respectively.

Scale Matrix

0.66667	0	0
0	1.2	0
0	0	1

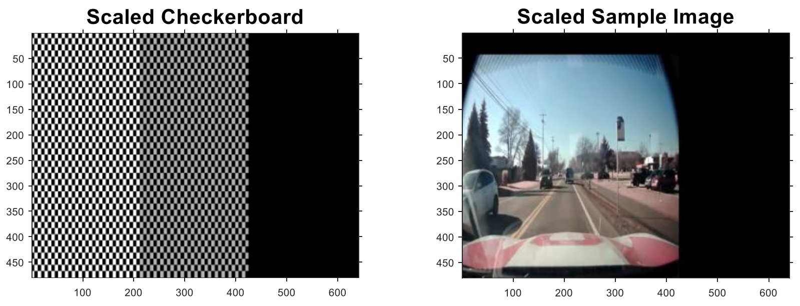


Figure 4.14: Checkerboard and Sample Image after Scalar Affine Transformation

Shear Matrix

1	0.04	0
0.08	1	0
0	0	1

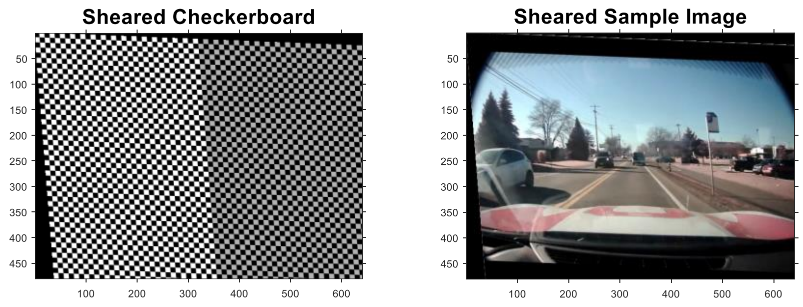


Figure 4.15: Checkerboard and Sample Image after Shear Affine Transformation

4.4.2 Perspective Transformation

Another useful type of transformation is one of perspective. If the parameters of the camera are known, then the image taken by the camera can be transformed into an image taken at a different angle or location. In automotive applications, it is common to create a birds eye view of the environment as shown in Figure 4.16, to translate image coordinates u,v into the real world point coordinates x,y,z.

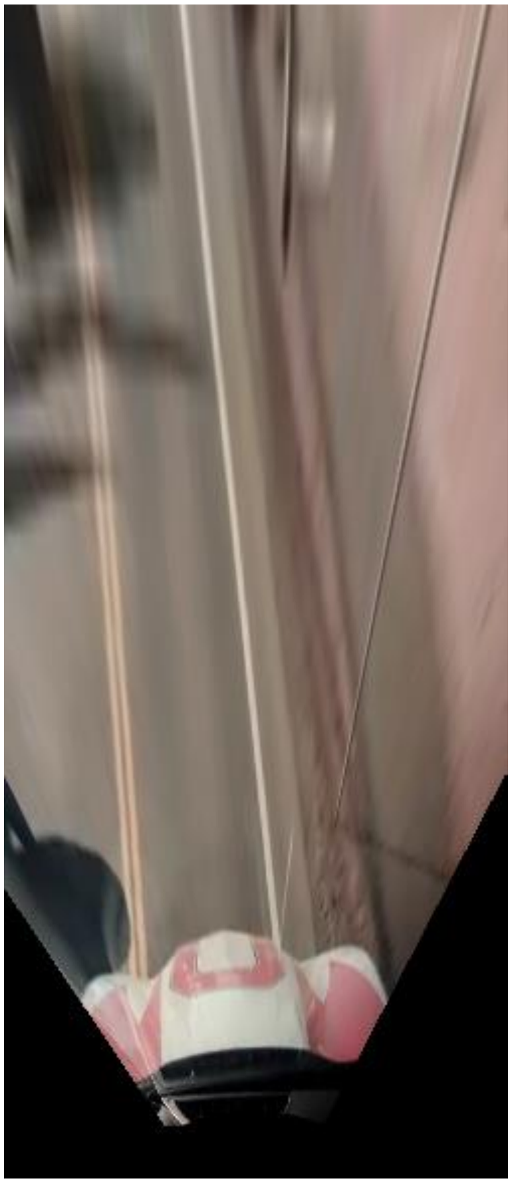


Figure 4.16: Inverse Perspective Mapping

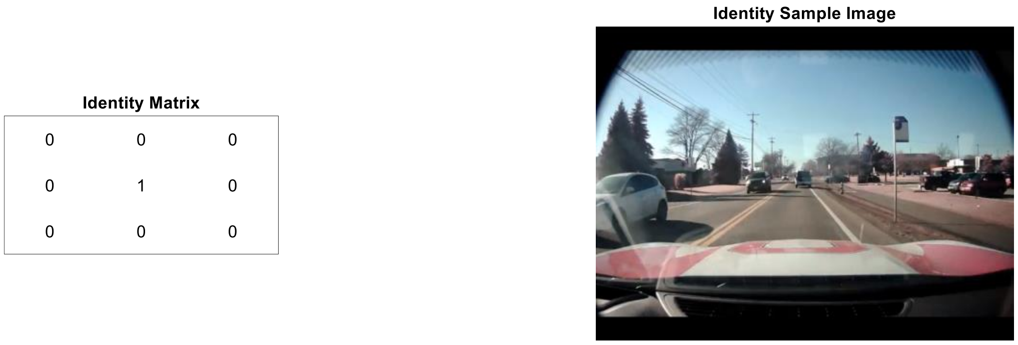
This is known as an inverse perspective mapping. One important feature of this transform is that converging lane lines, toward a vanishing point, in the image space become parallel in this perspective. This is often used as a prerequisite to lane detection algorithms for this reason.

4.4.3 Digital Image Filtering

Most image processing algorithms are performed using variations of the convolutional sliding window method using a weight matrix, known as a kernel.

$$g(u,v) = (w * I)(u,v) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) I(u-s, v-t) \qquad [13]$$

In Equation 13, *g* is an output image formed by the convolution of the input image *I* with a convolutional kernel operator, *w*. This is normally expressed as a weighted average between each pixel and its nearest neighbors performed through a raster scan. The magnitude and arrangements of the weights, expressed in the kernel operator, determine the output of the filter as well as whether it is high pass or low pass. For reference, the output of an identity kernel matrix is shown in Figure 4.17.



0	0	0
0	1	0
0	0	0

Figure 4.17: Identity Kernel Matrix and Sample Image

A common kernel is the averaging kernel known as a normalized box blur. All kernels produce a weighted average, but the box blur uses normalized weights such that each element in the kernel is identical. For a 3x3 kernel with 9 elements, this results in 1/9 ~ 0.11111 as each element. This acts as a spatial lowpass filter as each pixel's output value is determined by the average of itself and all its nearest neighbors. An example in shown in Figure 4.18.

Average (Normalized Box Blur) Matrix

0.11111	0.11111	0.11111
0.11111	0.11111	0.11111
0.11111	0.11111	0.11111

Average (Normalized Box Blur) Sample Image

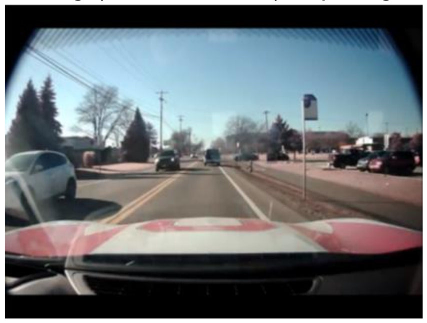


Figure 4.18: Average Kernel Matrix and Sample Image

Additionally, kernels can enhance features in the image. A common filter for this is sharpening. A sharpening kernel heavily weights the input pixels and removes the noise caused by its nearest neighbors by subtracting their value from the average. This results in an image with sharper, more prominent edges. This is shown in the sample image in Figure 4.19.

Sharpening Matrix

0	-1	0
-1	5	-1
0	-1	0

Sharpening Sample Image

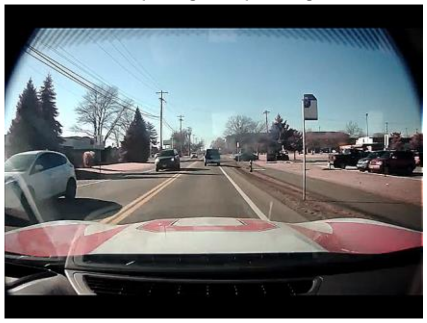
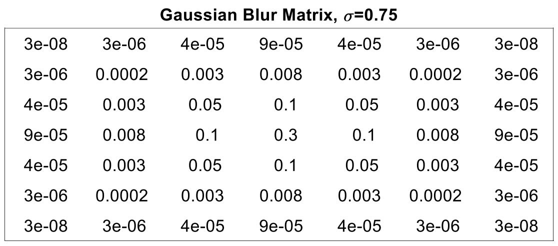


Figure 4.19: Sharpening Kernel Matrix and Sample Image

The box blur and sharpening filter exemplify two variations of kernel filters: low pass and high pass filters. For any pair of pixels in an image, the difference in value between the two can be considered a frequency. Blurs, generically, are low pass filters. They smoothen the image by averaging the frequencies. Thus, low frequencies present in the input image will remain low, but high frequencies will have their values attenuated during averaging. Conversely, edge detection filters, commonly known as derivative filters, target spatial regions with large differences in intensity. These contain large frequencies in the frequency domain, and thus pass through the filter unattenuated.

The most famous kernel operator is the Gaussian Blur. This filter uses a discretized two-dimensional Gaussian distribution as weights for a weighted average of the pixels within the kernel-sized neighborhood. This emphasizes the pixels centered around the mean of the distribution, which can be tuned based on the standard deviation, σ.


Gaussian Blur Matrix, σ=0.75

3e-08	3e-06	4e-05	9e-05	4e-05	3e-06	3e-08
3e-06	0.0002	0.003	0.008	0.003	0.0002	3e-06
4e-05	0.003	0.05	0.1	0.05	0.003	4e-05
9e-05	0.008	0.1	0.3	0.1	0.008	9e-05
4e-05	0.003	0.05	0.1	0.05	0.003	4e-05
3e-06	0.0002	0.003	0.008	0.003	0.0002	3e-06
3e-08	3e-06	4e-05	9e-05	4e-05	3e-06	3e-08

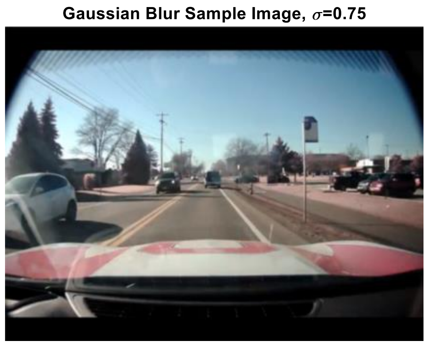


Figure 4.20: Gaussian Blur using σ=0.75

Figure 4.20 and Figure 4.21 show examples of two gaussian blur matrixes with σ=0.75 and σ=3, respectively. This function was used in ADAS software to blur the image before lane line identification. This is useful for reducing noise in color thresholding since the various shades of white and yellow will average with one another to become more uniform with less outliers.

Gaussian Blur Matrix, σ=3

0.01	0.01	0.02	0.02	0.02	0.01	0.01
0.01	0.02	0.02	0.02	0.02	0.02	0.01
0.02	0.02	0.03	0.03	0.03	0.02	0.02
0.02	0.02	0.03	0.03	0.03	0.02	0.02
0.02	0.02	0.03	0.03	0.03	0.02	0.02
0.01	0.02	0.02	0.02	0.02	0.02	0.01
0.01	0.01	0.02	0.02	0.02	0.01	0.01

Gaussian Blur Sample Image, σ=3

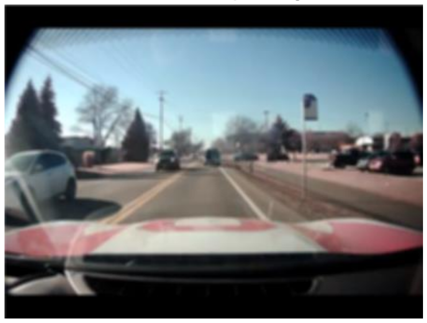


Figure 4.21: Gaussian Blur using σ=3

With the modeling, calibration, and pre-processing techniques discussed so far in this chapter, applications could be built to find and track lane lines and vehicle.

4.5 Lane Line Identification

Lane line recognition was needed to classify vehicles based on lane position. This was done by projecting the lane line coordinates toward the vanishing point of the image and determining whether target vehicle centroids were localized to the left, right, or in-between the lane line projections. This approach used a simplified version of several well-known algorithms to approximate the straight lines of the host lane.

4.5.1 Color Threshold

There are many lines present in a typical road scenario, and images in general. The algorithm can isolate lines of interest using colors and other features. In this application, white and yellow thresholds were set to isolate lane lines to mathematically model the coordinates. Binary masks can be made by filtering images for a specific color threshold. This is commonly done using the Hue Saturation Value (HSV) or Hue Saturation Lightness (HSL) color spaces, as they are usually better at isolating specific colors. Masks were also created in the RGB color space for both yellow and white separately for added robustness.

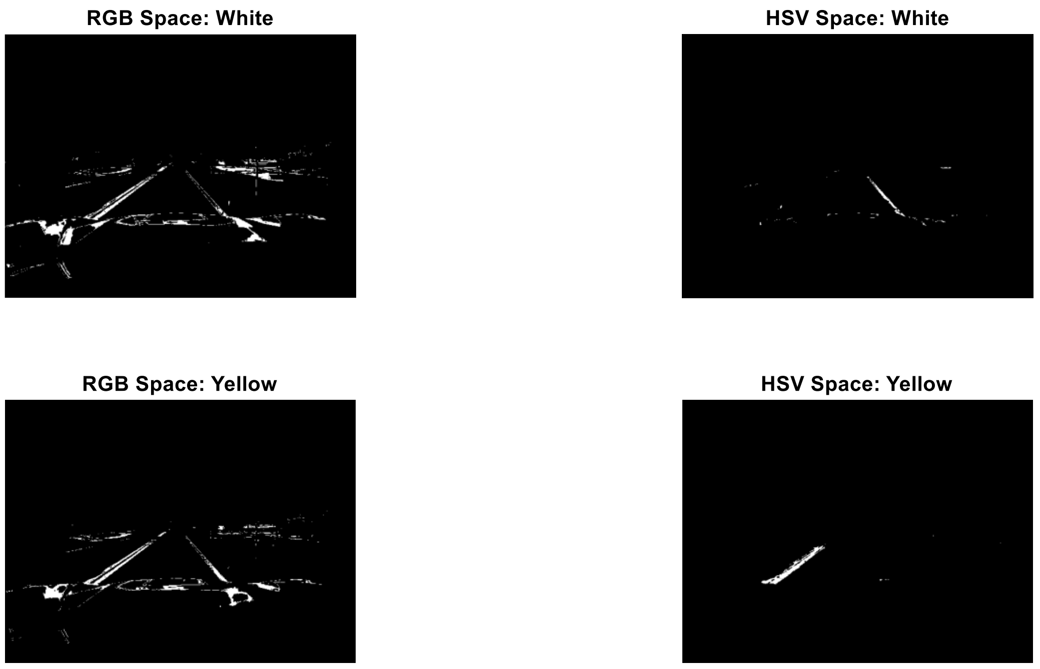


Figure 4.22: Binary Masks created by Color Thresholding in RGB and HSV Color Spaces

As shown in Figure 4.22, masks created in the RGB space contain more noise than those from the HSV space. Masks were then combined using a bitwise or operation. This created a black and white image isolating blobs of white and yellow objects within the upper and lower thresholds of each RBG and HSL channel used. This is shown in Figure 4.23.

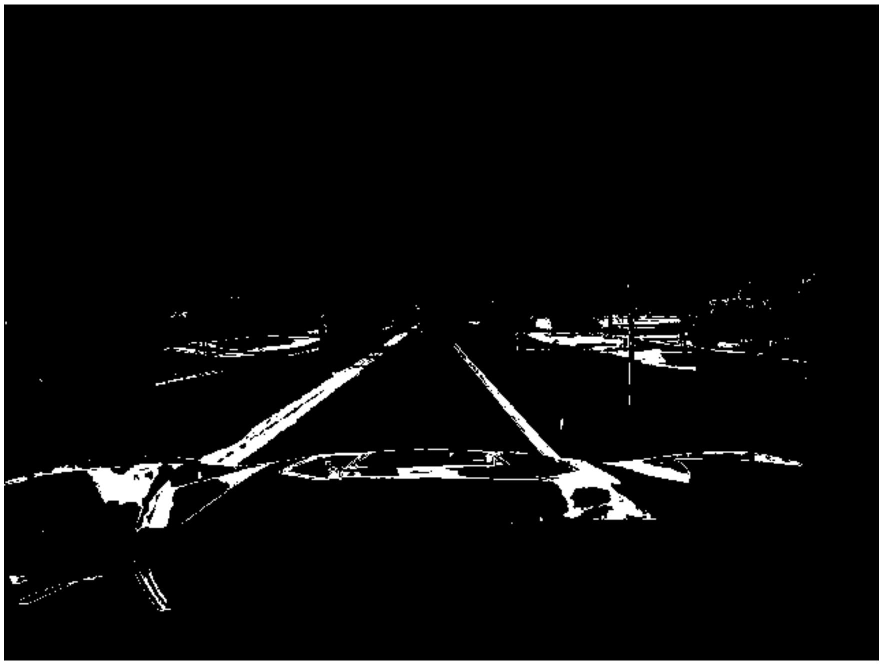


Figure 4.23: Bitwise OR Operation used to Combine Masks

The blobs contain lane information as well as other objects and noise of similar color. This noise must be eliminated to more accurately project the direction of the lane lines. This is done by restricting a region of interest mask based on the expected location of the desired lines.

4.5.2 Region of Interest

Once the camera was calibrated in its mounting location on the ego-vehicle, the position of the road was consistent with respect to the camera. Thus, lane lines can only appear in roughly the bottom, one-third region of the camera frame. Using this assumption, a region of interest (ROI) can be constructed that masks much of the noise in the image. The region of interest chosen for this project is shown in Figure 4.24.

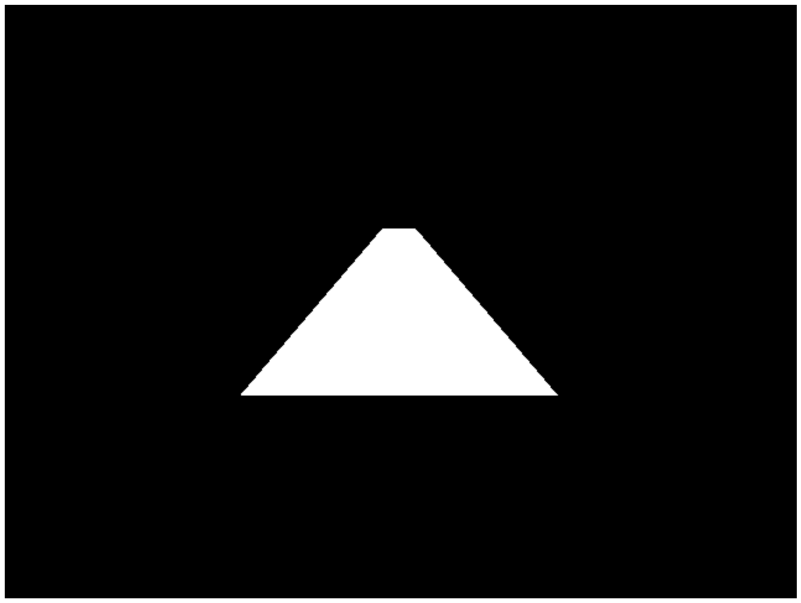


Figure 4.24: Lane Line Region of Interest

The ROI can be restricted in the image frame by using a bitwise and operation. Thus, all pixels in the binary mask that are 1 and fall within the ROI will remain at 1. Additional noise can be filtered using morphological blob analysis techniques. These can be used to morphologically close or open blobs of certain shapes and sizes to eliminate distortion and noise. Here, a morphological close operator is used to close blobs of disk shape with less than 3 pixels to help smoothen the lane lines. Additionally, a morphological open operator is used to open blobs with area less than 20 pixels which removed noise from undesired white and yellow areas of the image. Using color masks, ROI constraints, and morphological operators, a clean binary image of isolated lane lines can be produced as shown in Figure 4.25.

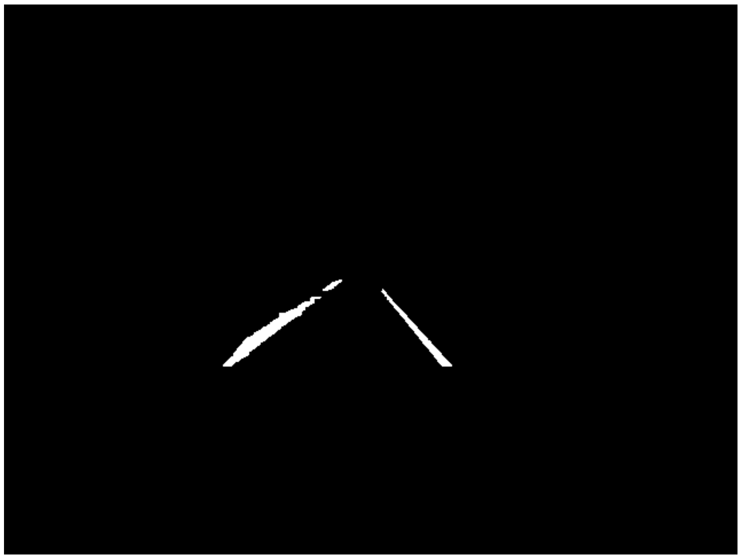


Figure 4.25: Lane Line Binary Mask Output

A filtered lane line mask is now present in the system, but alone, it is not useful to a control algorithm because it has no mathematical representation in the vehicle coordinate space. The binary image must now be passed through a series of algorithms, Canny Edge Detection and Hough Transforms, to obtain the coordinates of the lane lines.

4.5.3 Canny Edge Detection

Edge detection is a feature extraction technique used to filter blobs into just their edges. This is a useful technique in computer vision as it reduces object data and complexity while still preserving shape, size, and orientation. There are numerous implementations of this technique which generally use gradients to uncover rapid changes in pixel intensity, implying an edge. This is the idea behind the Sobel operator, which is a convolutional kernel used to identify horizontal and/or vertical edges based on gradients. If the magnitude of the gradient is high, this implies that an edge is present since the intensity is changing rapidly. A grayscale image is used because color is not needed for this algorithm, as intensity is sufficient to compute gradients.

$$m = \sqrt{G_U^2 + G_V^2} \qquad [14]$$

Equation 14 calculates the magnitude, *m*, from the gradients, G_U and G_Y, in the *u* and *v* direction, respectively. Edge direction can also be computed. The direction can be determined by the arctangent of the y-direction gradient over the x-direction gradient as shown in Equation 15 [34].

$$\theta = \arctan\left(\frac{G_V}{G_U}\right) \qquad [15]$$

In Canny edge detection, gradients are calculated using the Sobel operator, but with additional processing to filter edges captured by noise. First, large intensity frequencies are attenuated using a 5x5 Gaussian filter to smooth intensities with their neighbors. Sharp edges will be maintained, but softer edges causing noise will be dampened further. Canny outlines three criteria used to evaluate the performance of the detector during operation: detection, localization, and single response. First, detection must be consistent with a low probability of missing edges and a low probability of false alarm [35]. Additionally, the algorithm should localize the position of the edge accurately against the true center of each edge. Lastly, the algorithm should explicitly detect each edge only once. Canny Edge detection is desirable because it uses an iterative approach to maximize detection while minimizing localization error using the criteria we have just defined.

Detection criteria is described as the maximization of the signal-to-noise ratio (SNR). This is defined as the ratio of a chosen filter's response to the signal, an edge, to its response to noise only. Localization criteria is similarly defined as a maximum of the inverse of the root-mean-squared error between an edge detection and its true center [35]. For localization, the Canny algorithm will evaluate each pixel with its neighbors to determine local maxima in the direction of its gradient. The local maxima are maintained and neighbors on the gradient axis are suppressed.

This process thins edges to just one pixel in width. This is useful because it removes noise but maintains boundaries.

Lastly, hysteresis thresholding is used to more accurately remove false alarm edges. In two dimensions, edges are evaluated at each continuous gradient magnitude in relation to the upper and lower hysteresis threshold. If a gradient on the edge exceeds the upper threshold, then it is considered a valid edge. However, if an edge has a gradient which drops below the lower threshold the edge is removed. This maintains weak edges as long as they are continuously connected to a strong edge and maintain the gradient orientation of the strong edge. This compensates for lighting discrepancies and other external factors within the image because shape can be maintained even when intensity cannot. The OpenCV Canny implementation allows the developer to tune the hysteresis thresholds, while the MATLAB implementation can also automatically select thresholds based on amount of noise in the image using a signal-to-noise ratio [35]. Now, the input image has been processed to contain just the edges of each lane line.

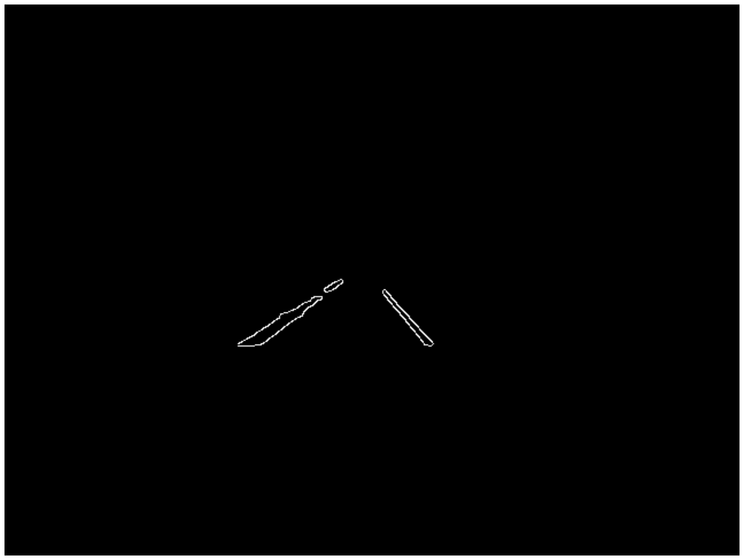


Figure 4.26: Canny Edge Detection on Binary Mask Lane Line Image

4.5.4 Hough Transforms

Canny Edge Detection was used to isolate the edges of the host lane lines, but further processing is needed to recover the coordinates of the edges in both the camera and ego-vehicle coordinate spaces for developing applications. Hough transforms were used to find lines formed by the white pixels in the binary Canny image.

Lines can be defined in the feature space in slope-intercept notation, for a single-channel binary image, as shown in Equation 16.

$$v = au + b \tag{16}$$

Where (u,v) indicates a 2D image point on the line defined by au + b. Similarly, we can define lines, not in the cartesian space of orthogonal basis vectors $\hat{u}$ and $\hat{v}$, but in polar coordinates defined by an angle, ϴ, and a radius, ρ. This is shown in Equations 17, 18, and 19.

$$u = \rho \cos(\theta) \tag{17}$$

$$v = \rho \sin(\theta) \tag{18}$$

$$ucos(\theta) + vsin(\theta) = \rho \tag{19}$$

The Hough transform finds (ϴ, ρ) by iterating over the image until an edge pixel is identified. Once such a pixel is found, the algorithm iterates over angular values of theta until another neighboring edge pixel is found. The previous theta value is then used to check for another edge point on the line formed by the previous two edge points. Using this method, a Hough space can be formed using the strength of theta and rho values based on the number of edge points in the image that are found from each (ϴ, ρ). The Hough space is visualized in Figure 4.27 for the Canny input image shown in Figure 4.26.

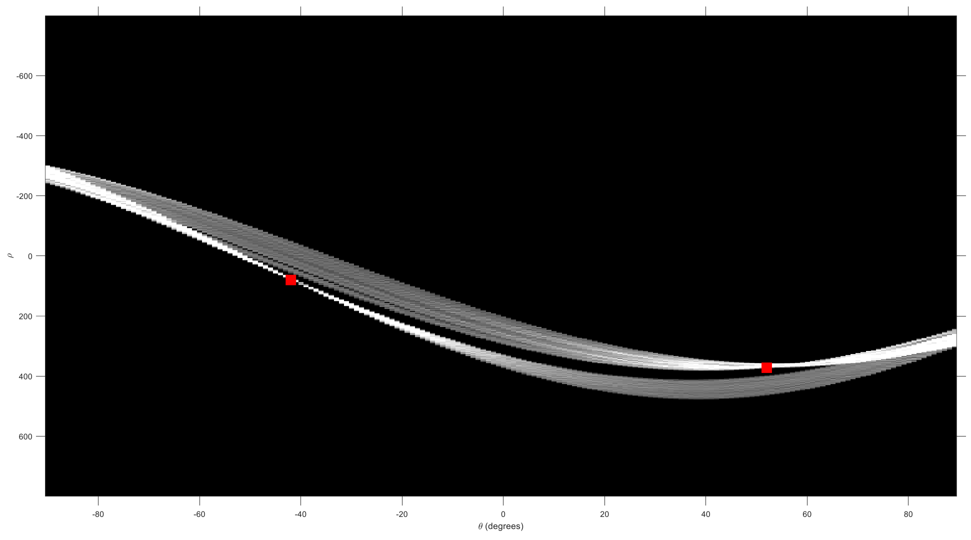


Figure 4.27: Visualization of Hough Space Generated from Camera View Image

In this visualization, we see two surfaces which illustrate the two lane lines in the input image. Each edge of the lane lines is detected by the Hough transform, those there is a wide variance in the two Hough surfaces, accounting for the slight dissymmetry in the converging lane lines as they approach the image vanishing point. The peaks of the Hough Space, shown as red squares in Figure 4.27, indicate Hough parameters of highest certainty, since the most edge points can be represented on each lane line using this parameterization. It is not surprising that both peaks are nearly symmetrical about 0 degrees theta, as both lane lines in the camera view should be oppositely signed angles converging toward the principle point of the camera view.

We can also transform our binary masked image into a Bird-s-Eye-View equivalent to detect edges and Hough lines as well. Figure 4.28 shows the binary masked image and canny edge image in a Bird's-Eye-View after performing an inverse perspective mapping.

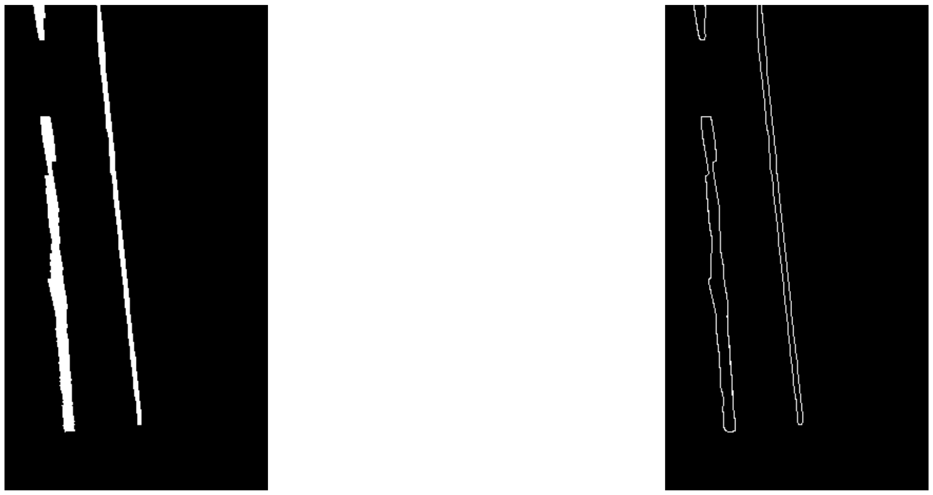


Figure 4.28: Inverse Perspective Mapping Performed on Camera View Binary Mask and Canny Edge

Hough Transforms were also computed on the Canny image in a Bird's-Eye-View perspective. Figure 4.29 shows the Hough space formed with peak points located nearly at the same theta, -4° and -6°. This is expected since lines in the bird's-eye-view remain parallel and don't converge near a vanishing point.

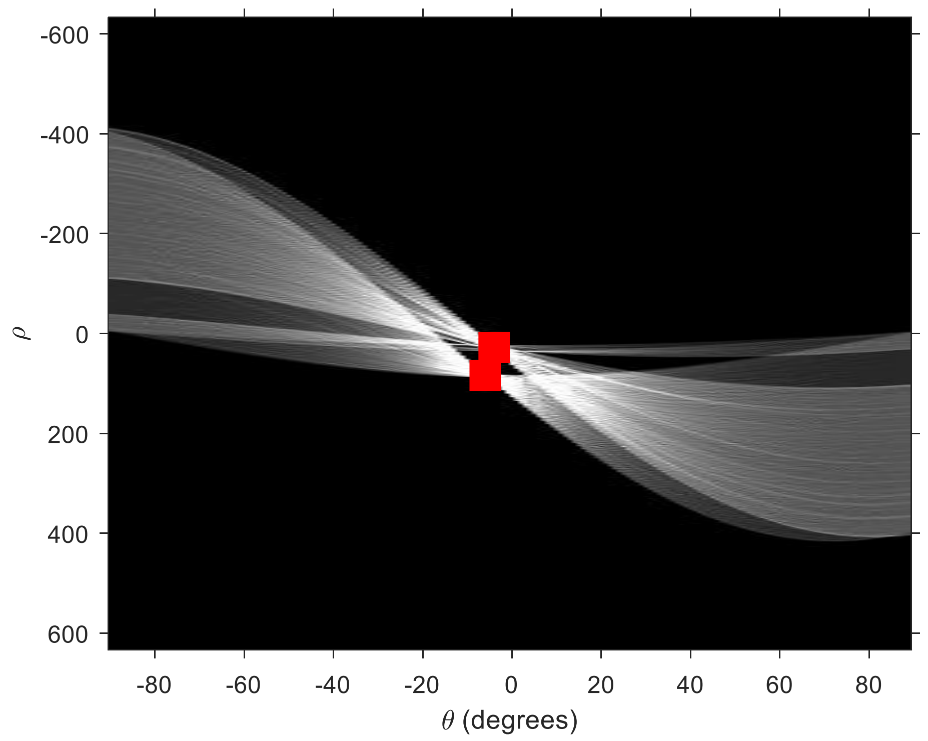


Figure 4.29: Visualization of Hough Space Generated from Bird's-Eye-View Image

We then map the peak Hough parameters to lines in the camera coordinate space to mathematically represent lane lines for applications like lane departure warning, as shown in Figure 4.30. Similarly, we can map the peak Hough parameters from the inverse perspective into a birds-eye-image representation to see how accurately the linear approximation aligns against the ground truth. This is shown in Figure 4.31.

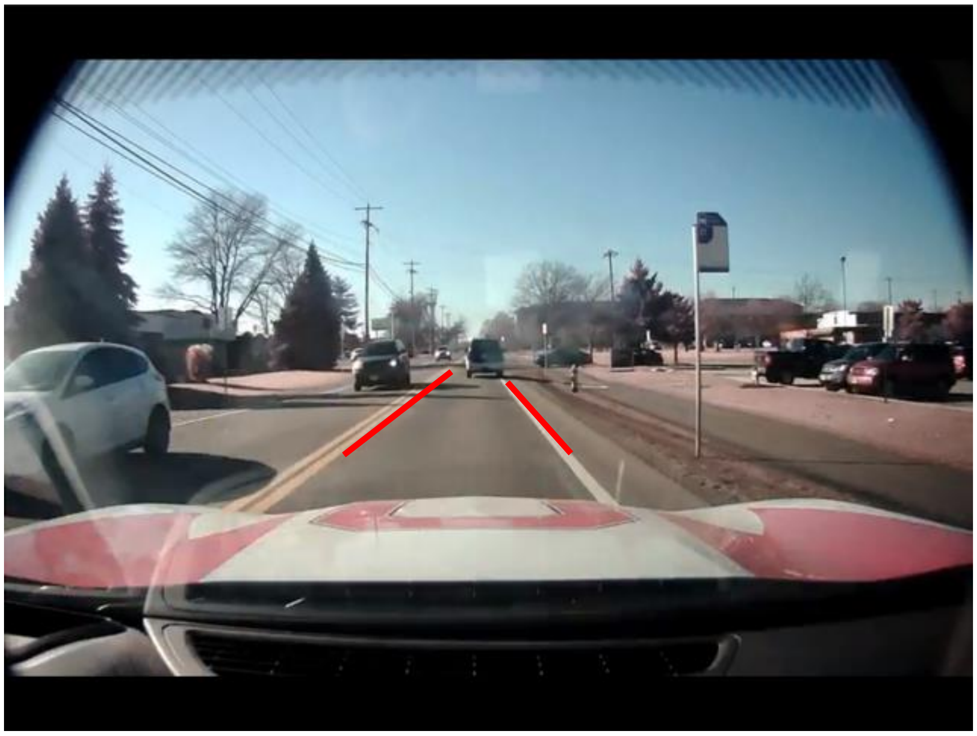


Figure 4.30: Hough Generated Lane Lines Shown in Camera View Perspective

The lane lines are shown only within the spatial constraints set by the ROI; however, the projection of the lines allow us to accurately categorize target vehicles based on their relative position to these lane lines.

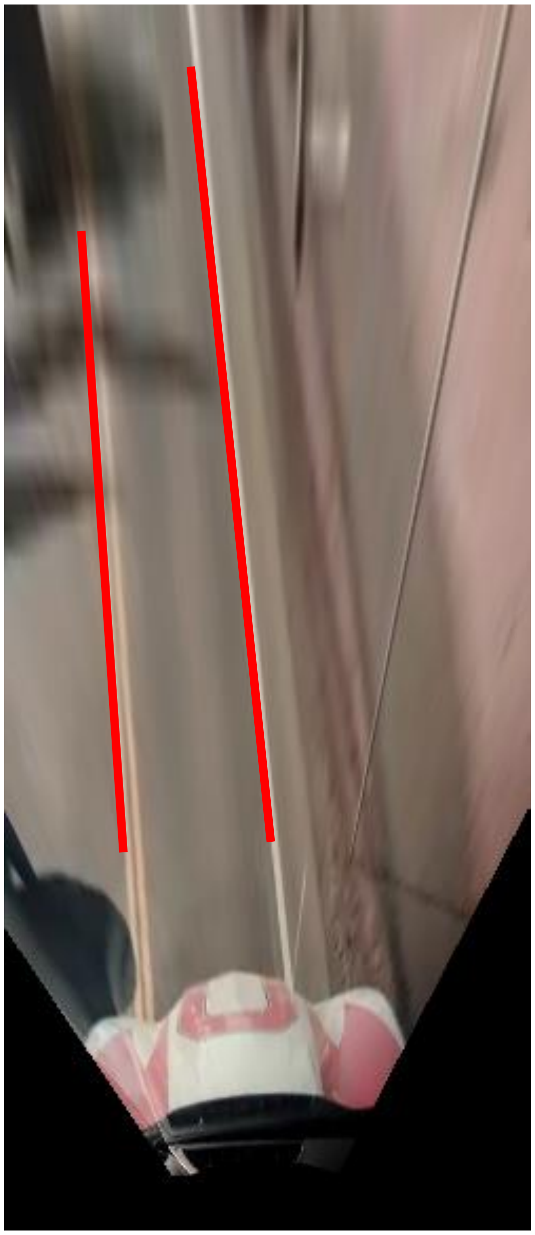


Figure 4.31: Hough Generated Lane Lines Shown in Bird's-Eye-View Perspective

4.6 Vehicle Detection

Object detection is the process of Object classification and localization methods were used to localize target vehicles for tracking as well as applications such as FCW. Object detection was the most computationally rigorous section of the computer vision software, accounting for nearly all latency in the system. High object detection rates, measured in terms of high hit-rates and low miss-rates, were usually inversely proportional to computation time. Thus, design decisions were made to improve hit-rate and suppress false positives while maintaining real-time. Real-time must be defined for the system and application. Real-time can mean, and usually does, that the software is able to perform computations in time for the next sample to arrive. For a camera-based system, this would be around 20-30Hz since most modern cameras deliver this frequency in terms of frame-rate. For this system, less than 100ms of total latency was desired, such that the system operated faster than a driver could react [17]. Of course, the faster the performance, the better in terms of accident prevention. Since this was an open loop system, faster feedback would only improve the changes of accident mitigation since it would leave more time for the driver to intervene.

To improve detection performance and minimize latency, two popular algorithms were implemented based on the Viola Jones algorithm and convolutional neural network (CNN) with filtering and fusion techniques to fuse the detectors. Each algorithm will be discussed in terms of theory, training, and test results. Each algorithm was tested on an annotated video from the Laboratory for Intelligent and Safe Automobiles (LISA) Vehicle Detection Dataset [36].

4.6.1 Viola Jones Algorithm

The Viola Jones Algorithm was used to train and deploy several machine-learning based object detectors. It is based off the principle found in most Haar classifiers, namely classifying objects based on weights which define the linear combination of Haar features, shown in Figure 4.30, which define shaped defined by image intensities.

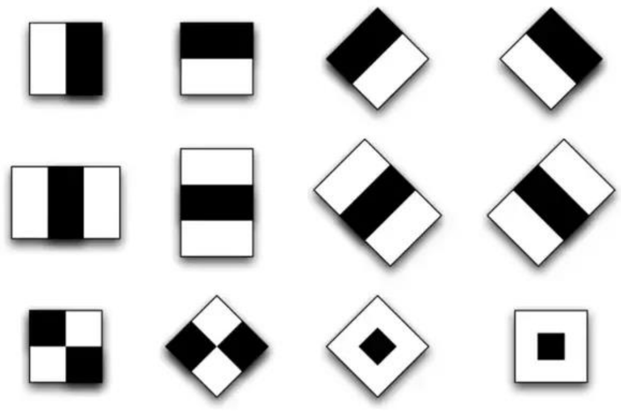


Figure 4.32: Two-Dimensional Rectangular Haar Features [37]

However, Viola-Jones can produce quicker results using 3 distinguishing differences.

(1) The input image is transformed into an *integral image* which can be processed quicker because fewer operations are performed per pixel. According to the work of Viola and Jones, rectangular features can be computed timelier if the image is preprocessed for row-wise convolution. Here, an integral image is computed so that each image coordinate contains the sum of the pixels above and to the left such that the image contains a monotonically increasing set of pixel information from the upper left corner origin toward the bottom right corner ending location [38]. This is summarized in Equation 20.

$$ii(u,v) = \sum_{u' \leq u,\ v' \leq v} i(u',v') \qquad [20]$$

(2) *Weak learners* are created using a modification in the AdaBoost training algorithm. In this implementation, the single Haar feature which produces the largest separation in positives and negative training samples is used, while all other

features are suppressed. Thus, the produced weak learner is constrained to only a single feature [38].

(3) The AdaBoost method is a method of weak classifier production which identifies objects based on only one feature. This is faster when used on a test image because the detector is attempting to classify objects using just one feature at a time instead of an entire set. However, just one feature results in high error rates due to the number of classes that may exhibit that feature. Viola-Jones solves this issue by cascading weak classifiers in series. This reduces error by filtering object classes against several models each examining a different feature. This increases classification speed as simple false positives can be filtered early in the cascade, while more complex classifications are performed later in the cascade [38]. This means the cascade classifier can prioritize simple features without having to check each sub-window in the image for every feature which defines the class.

In this work, two Haar classifiers were trained using the Viola-Jones framework. The first used approximately 700 close-range positive images. The second classifier was trained on approximately 700 far-range positive images. Both training sets were augmented with roughly 1400 negative images of roadways with zero automobiles. All data was collected using the system camera captured in native resolution and annotated using MATLABs Ground Truth Labeler application. Training was performed using the following function in MATLABs Computer Vision System Toolbox.

trainCascadeObjectDetector(outputXMLFilename, positiveInstances, negativeImages)

This generated OpenCV compatible XML files which describe the feature weights used to define a *vehicle* class. These were then exported into the Python OpenCV implementation of a cascade classifier.

cv.CascadeClassifier('Haar_Training_rearVehicleDetectorHaar_700sample.xml')

For analysis, a test video from the LISA dataset was evaluated using the MATLAB script in Code Snippet 1. Note, a ROI was constructed to reduce unnecessary false positives, however, no additional filtering or parameterization was performed to ensure an apples-to-apples comparison between classification techniques. Figure 4.33 shows a still frame with close-range classifier detections (red) and ground truth (blue) annotations.

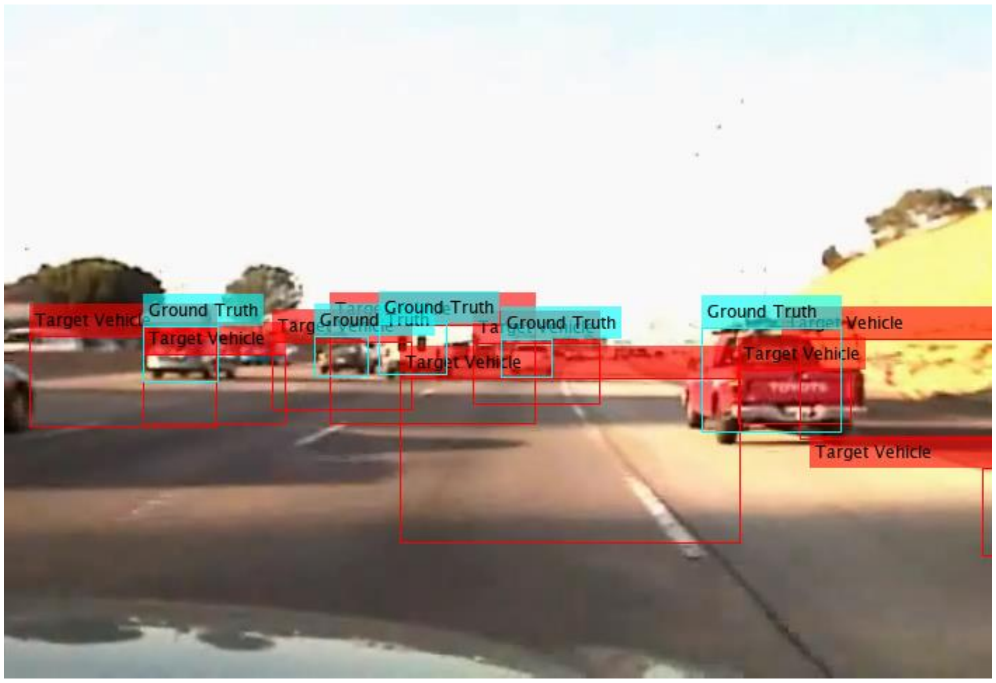


Figure 4.33: Output of Close-Range Classifier and Ground Truth on a Test Frame

This classifier was overly sensitive which usually led to more false positives that true positives per frame. However, this algorithm was still useful for detecting vehicles at close ranges.

```
%% Params
hitThreshold = 0.4;
avgVehicleSizeInPixels =2.4883e+03;
%% Run Detector and Compare to Groundtruth
trainingSets = dir('*.xml');
trainingSetLineSpec = {'-r', '-b'};
for ii = 1:length(trainingSets)
    detector = vision.CascadeObjectDetector(trainingSets(ii).name);
    line_num = 0;
    TParray = []; TNarray = []; FParray = []; FNarray = [];
    totalTargetsinGT = [];
    plotInd = 0;
    %% Run Video
    fileId=fopen(‘pos_annot.dat','r');
    v = VideoReader(‘jan28.avi');
    figure;
    while hasFrame(v)
        videoFrame = readFrame(v);
        videoFrameROI = videoFrame(160:400, 1:end);
        detectorOutput = step(detector,videoFrameROI);
        detectorOutput = [detectorOutput(:, 1) detectorOutput(:, 2) + 160 detectorOutput(:,
3:4)];
        line_num=line_num+1;
        lineData=fgetl(fileId); % read a line
        lineDataDelim = strsplit(lineData, '\t');
        %% Evaluate Detections again Ground Truth
        groundTruth = [];
        numDetectionsInGT = str2num(lineDataDelim{2});
        for boxInd=1:numDetectionsInGT
            groundTruth = [groundTruth; str2num(lineDataDelim{boxInd+2})];
        end
        if(all(groundTruth, 'all') && all(detectorOutput, 'all'))
            detectionVsGT = bboxOverlapRatio(detectorOutput, groundTruth) > hitThreshold;
            areaOfImageOccupiedByVehicles = sum(groundTruth(:, 3) .* groundTruth(:,4));
            TParray = [TParray sum(any(detectionVsGT, 1))];
            FNarray = [FNarray numDetectionsInGT - sum(any(detectionVsGT, 1))];
            FPthisFrame = length(detectionVsGT(:, 1)) - sum(any(detectionVsGT, 1));
            FParray = [FParray FPthisFrame];
            TNthisFrame = max(((double(videoFrameROI(3))*double(videoFrameROI(4)) -
areaOfImageOccupiedByVehicles)/avgVehicleSizeInPixels) - FPthisFrame, 0);
            TNarray = [TNarray TNthisFrame];
            totalTargetsinGT = [totalTargetsinGT numDetectionsInGT]; %=TP+FN
            plotInd = plotInd + 1;
            %% Video Output Display
            detectedImg = insertObjectAnnotation(videoFrame,'rectangle',detectorOutput,'Target
Vehicle', 'Color', 'red');
            detectedImg = insertObjectAnnotation(detectedImg,'rectangle',groundTruth,'Ground
Truth', 'Color', 'cyan');
            imshow(detectedImg);
        end
    end
    figure
    TPR = TParray./(TParray+FNarray);
    FPR = FParray./(FParray+TNarray);
    [FPR, I] = sort(FPR);
    TPR = TPR(I);
    plot(FPR, TPR, trainingSetLineSpec{ii});
    title('Reciever Operator Characteristics of Haar Classifiers');
    xlabel('False Positive Rate');
    ylabel('True Positive Rate');
    %' trainingSets(ii).name(1:end-4)
End
```

Code Snippet 1: MATLAB Script to Generate ROC for Haar Classifiers

Note, the data extracted from the detection and ground truth needed to be reorganized in order to utilize MATLABs plotroc() function. This involved taking each ground truth annotation, false positive, and true negative and treating each as individual samples in a True/False Binary experiment.

The receiver operator characteristic (ROC) for the close-range classifier is shown in Figure 4.34. This shows that the classify is extremely sensitive, as it requires a high false alarm (false positive) rate to achieve high detection. Thus, if treated as a decision rule, this classifier will exemplify high type I error during operation.

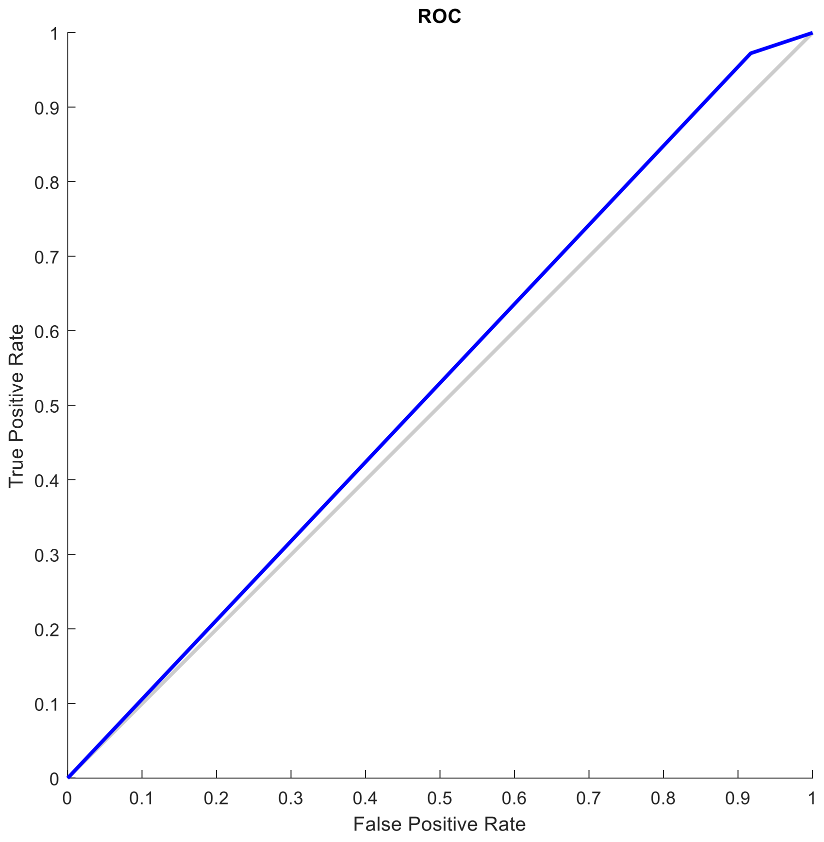


Figure 4.34: Receiver Operator Characteristic for Close-Range Haar Classifier

This was acceptable because additional filtering techniques can be taken to reduce false positives while maintaining the true positive rate. The ROC is powerful for evaluating classifiers, but in object detection there are other parameters that are important. ROC treats the classifier as binary, either correctly identifying the vehicle or not without taking into consideration the complexity of localization. The second classifier performed similarly, with ROC shown in Figure 4.36.

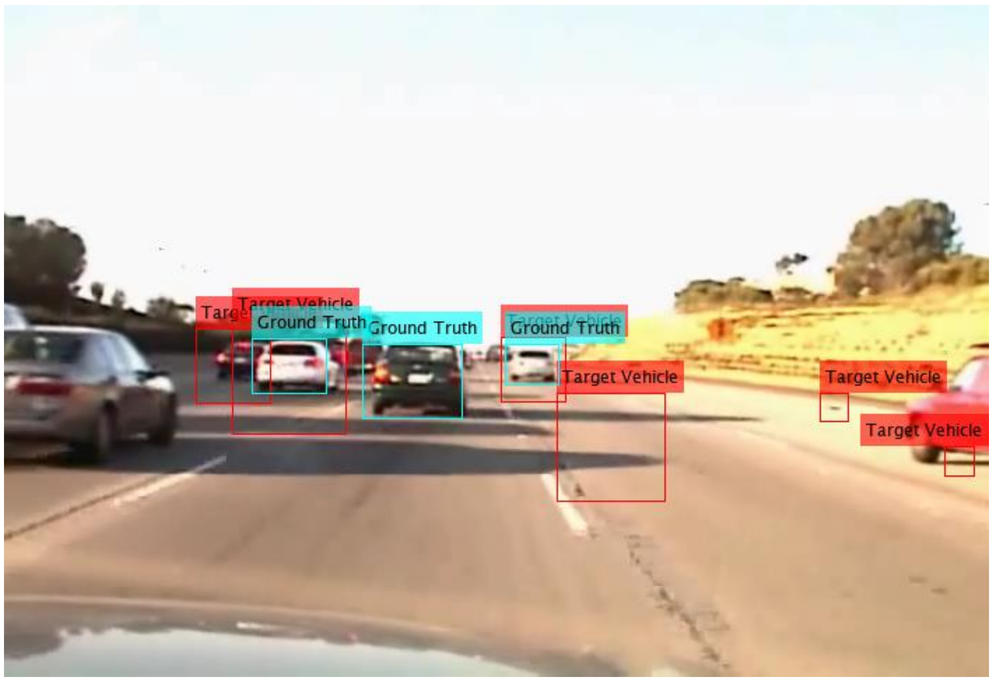


Figure 4.35: Output of Far-Range Classifier and Ground Truth on a Test Frame

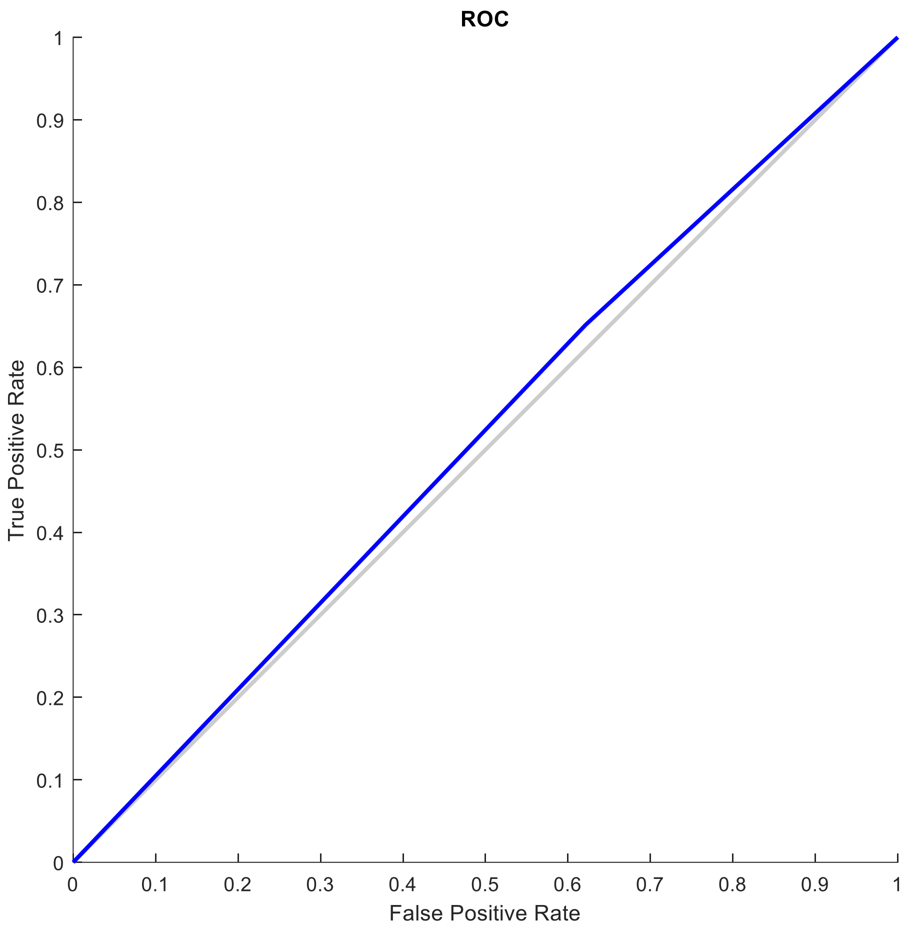


Figure 4.36: Receiver Operator Characteristic for Far-Range Haar Classifier

The Haar Classifiers were capable of 12fps, which was sufficient for real-time vehicle detection. However, they were overly sensitive which required additional filtering to produce the desired output. This will be discussed further in the *Trade-offs* section.

4.6.2 Convolutional Neural Network

A convolutional neural network (CNN) was also used for object detection. This algorithm forms the basis for the modern approach to object detection known as “deep learning, using layers of weighted functions determined by data exposure during training. For this application, the CNN was pre-trained based on the DriveNet architecture and created using University of California at Berkeley's Caffe Deep Learning Framework [39] for an open-sourced vehicle detection application. The network nearly always identified vehicles without false positives but was selective on which vehicles it identified. Based on experimentation, the network preferred close-range vehicles in adjacent lanes over lead vehicles in the ego lane. This was likely because of the additional information available to the network from the exposed vehicle profile in an orthographic view. This is illustrated in Figure 4.37, for example, where the network identifies only the vehicle on the far left.

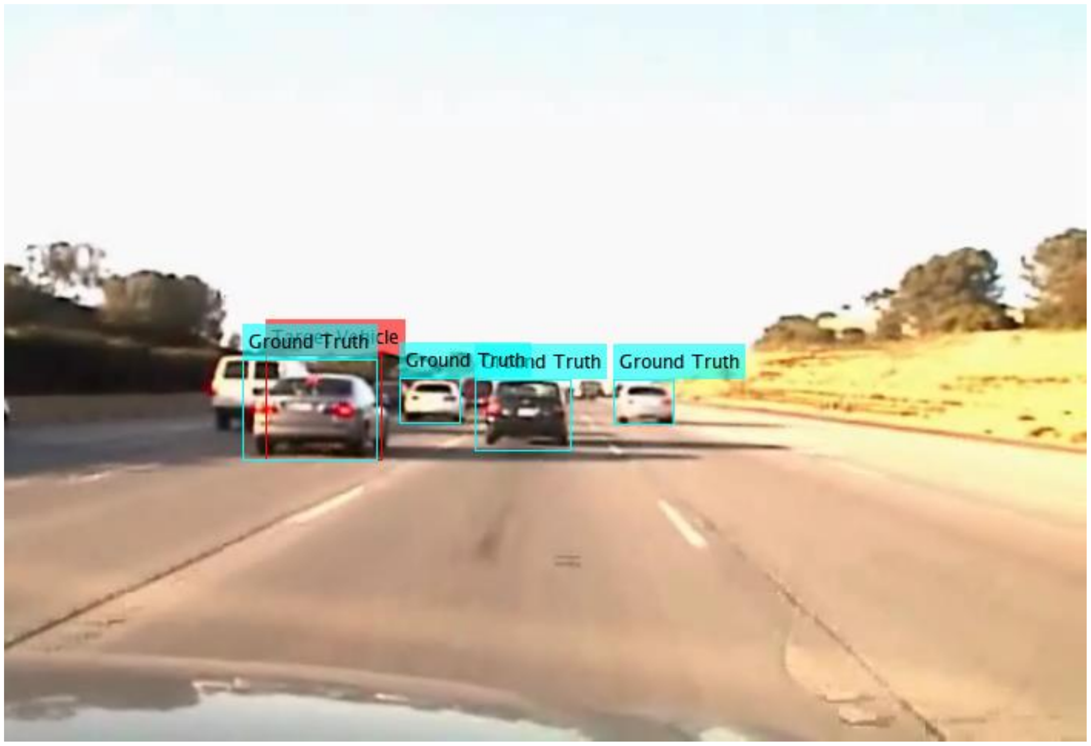


Figure 4.37: Output of CNN and Ground Truth on a Test Frame

Figure 4.38 shows the ROC for the CNN against the LISA test video shown in Figure 4.37. Here, we see a low hit rate of about 1 in 5 vehicles identified with 0 false positives for the video sequence. This network seemed to behave conversely to that of the Haar classifiers, however, an algorithm was developed to optimize true positives and minimize false positives and computation time using both detection methods.

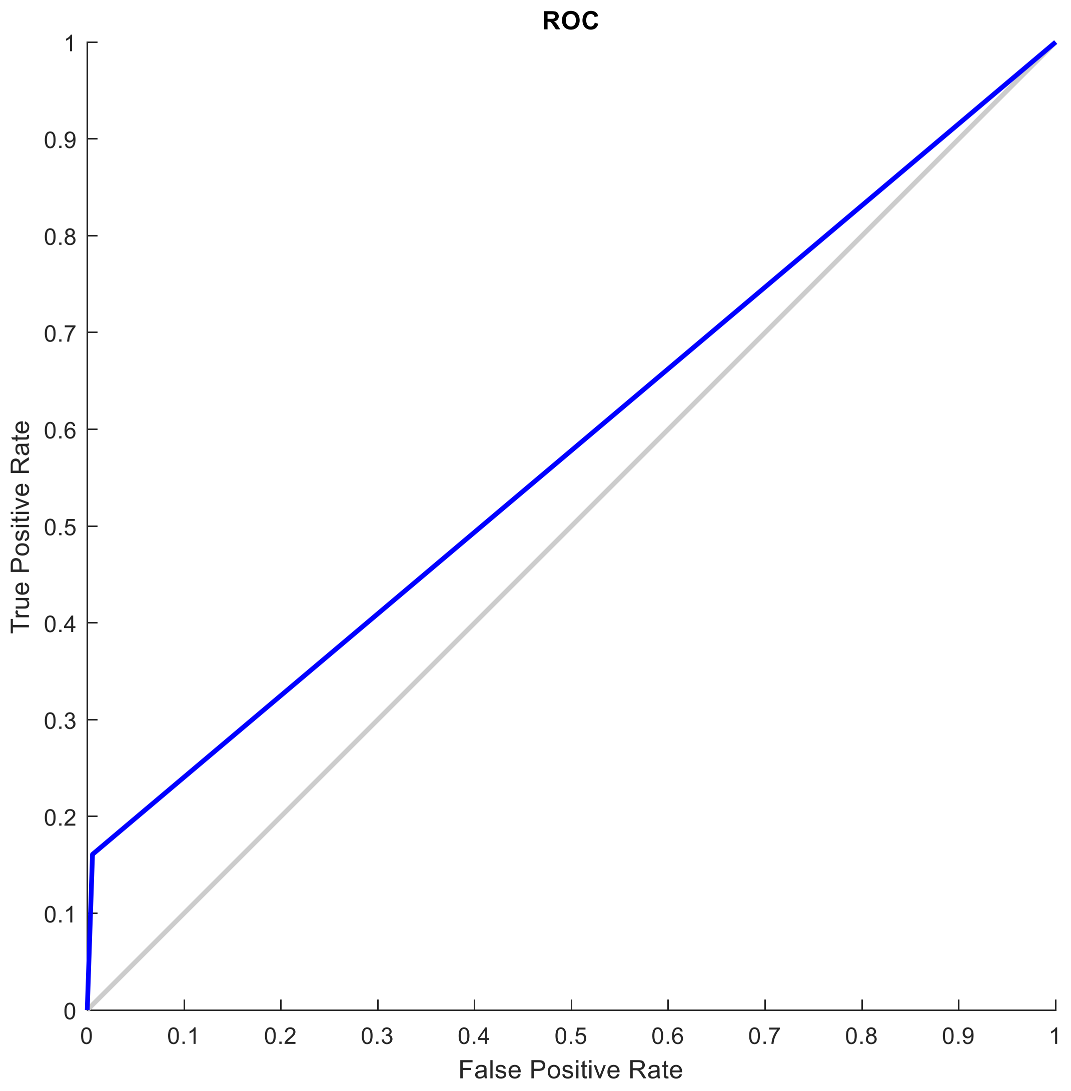


Figure 4.38: Receiver Operator Characteristic for CNN

4.6.3 Haar-CNN Interpolation

Due to the high sensitivity of the Haar classification method and the timely, low sensitivity of the CNN, a new algorithm was developed which utilized both algorithms when applicable. Table 4.1 shows a breakdown of strengths and weaknesses of both algorithms individually.

Table 4.1 Comparison of Haar Classifiers and CNN Object Detection Strategies:

Haar Classifier		CNN	
Strengths	Weaknesses	Strengths	Weaknesses
Rear Vehicles	Overly sensitive when noise is present	Identifying Vehicles in adjacent lanes	
Far range vehicles near image vanishing point		Identifying close vehicles	Rear vehicles centered new vanishing point

To compensate for high false positives and low hit rate, a new algorithm was created; referred to as Haar-CNN interpolation. This algorithm was implemented based on the following observations and solutions:

1. *Observation*: Haar is consistent at detecting vehicles near the vanishing point which are far range.

   Solution: The far range Haar classifier was restricted to an ROI, roughly one fourth the size of the image frame, centered about the vanishing point. This restricted the possibility of false positives as vehicles would occupy most of the ROI and noise wasn't much of a factor at this range.

2. *Observation*: The CNN was better at finding vehicles in adjacent lanes as well as close range vehicles in the ego-lane.

   Solution: The CNN was not restricted by ROI, thus operating on the entire image frame. This was due to its performance in all regions as well as the layer architecture which was trained on 480x640 images, thus reducing input resolution did not improve computation time.

3. *Observation*: Haar was less likely to produce a false positive when its ROI was restricted, thus Haar was better at confirming a vehicle was present in an ROI versus finding a new vehicle in its first occurrence.

   Solution: The CNN was used on every third frame in order to identify new vehicles as they came into view. Once a vehicle was identified, the Haar classifiers would be used within the ROI formed by the bounding-box returned from the CNN. This restricted the Haar classifier to a region in which a vehicle was known to be present, so that the Haar could verify its present on subsequent frames before the CNN was used again.

Using this technique, computation time could be saved by using the CNN sparingly while improving detection rates. Additionally, other filtering techniques were used to produce clearer output such as filtering bounding box estimates based on a box-area to vertical position map, which assumes a vehicle will become smaller as it approaches the vanishing point. Once filtering was applied, all bounding boxes from a 6-frame history were fed into a heat map filter, which stacks the boxes to form regions with higher detection intensity. A threshold is then set to ensure that selected outputs have been detection several times in the last half-second, thus the output is less likely to contain a false positive.

4.7 Vehicle Tracking

Object detection is necessary for camera-based perception systems; however, it does not form an all-in-one solution to state-estimation of target objects. Additional processing is needed to estimate the trajectory of the objects identified over time to gain additional insight into the environment. In this section, an optical flow-based tracking method is introduced and discussed in terms of tracking target vehicles within a camera frame.

4.7.1 Optical Flow

Optical flow uses the principle of point correspondences – either a point in space moves or the camera moves, to estimate a motion field – a velocity associated with each image point. To do this, several underlying assumptions are made. First, the object-of-interest must be rigid. Otherwise, the velocity estimates would be skewed by the change in size and shape of the object. Second, the timestep between image captures must be small so that pixel data is merely moving to a different pixel with each new capture and not escaping the image frame altogether. This is a difficult assumption to maintain true as the timestep is directly proportional to computation time. Third, illumination should not change. The light source should not change position, orientation, or luminance. This is because the flow algorithm uses change in pixel intensity to track motion.

A simplistic model for an optical flow estimation is derived as follows. First, a real-world coordinate, P, is defined in the camera frame of reference:

$$P = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = a\ world\ coordinate\ in\ the\ camera\ frame\ of\ reference$$

P is then mapped to a camera frame equivalent using the camera pinhole model discussed at the start of this chapter.

$$P' = \begin{bmatrix} u \\ v \end{bmatrix} = a\ pixel\ coordinate\ in\ the\ camera\ frame$$

Using the geometry of the pinhole model, we can continue to define an optimal flow model using the focal length of the camera, the optical axis, and their relationship with P and P’ as follows:

$$f = camera\ focal\ length$$

$$r_0 = distance\ of\ world\ point\ from\ the\ pinhole\ along\ the\ projection\ axis$$

$$r_1 = distance\ of\ pixel\ point\ from\ the\ pinhole\ along\ the\ projection\ axis$$

$$\alpha = angle\ between\ the\ optical\ axis, z, and\ the\ projection\ axis\ for\ point\ P$$

Thus, a depth estimate is derived for the pixel point, P’, using the camera pinhole model as before.

$$r_1 = f \cdot sec(\alpha) \qquad [21]$$

The distances of r_0 and r_1 are proportional as they represent the same distance in two coordinate systems. Thus, it stands that a velocity, v_i, is a pixel motion field based on the motion field, v_0, of real-world coordinates; v_0 and v_i are defined as follows:

$$v_0 = \frac{dr_0}{dt}, \qquad v_i = \frac{dr_i}{dt}$$

The derivation is intuitive; from a position and time estimate a velocity estimate is produced, but additional software is needed to ensure the optical flow implementation will work quickly for real-time implementation. This work utilized the Lucas Kanade Algorithm to estimate optical flow and produce an object tracker using the OpenCV implementation [40]. For an image sequence, I, with pixel dimensions u,v and time t, we can define optical flow as the change of the pixel points over time in the sequence.

$$optical\ flow = \begin{cases} u(x,y) \\ v(x,y) \end{cases}$$

Then we can create a model for the next image using the kinematics of the velocity, previous position, and delta time as shown in Equation 22.

$$I(x + u\delta t, y + v\delta t, t + \delta t) \approx I(x, y, t) \qquad [22]$$

By assumption 2, if the sampling rate of the images in the sequence is high enough, then the shift in pixel position along u and v (x and y) is minimal and the time difference is also minimal. With this assumption, we can assume that each point in each pixel moves only slightly to one of its nearest neighbors. This assumption simplifies calculations and helps compute optical flow in real time. A relationship between the images I(:, :, t_0) and I(:, :, t_1) in the sequence are found based on the value of camera frame coordinate (u,v).

$$I_x u + I_y v + I_t = 0 \qquad [23]$$

$$\left(I_x, I_y\right)(u, v) = -I_t \qquad [24]$$

From which we can obtain the direction, or orientation, and magnitude of the optical flow as shown in Equations 25 and 26, respectively.

$$(u, v) = -\frac{I_t}{(I_x, I_y)} \qquad [25]$$

$$\left|\left|u, v\right|\right| = \frac{I_t}{\sqrt{I_x^2 + I_y^2}} \qquad [26]$$

The optical flow motion field can be computed between any two frames in an image sequence. Figure 4.39 shows the output on a cropped image from the LISA dataset [36]. This shows how points that move uniformly with the motion of the ego vehicle create a laminar flow while points that move independently, due to their position on another moving object, demonstrate turbulent flow within the motion field.

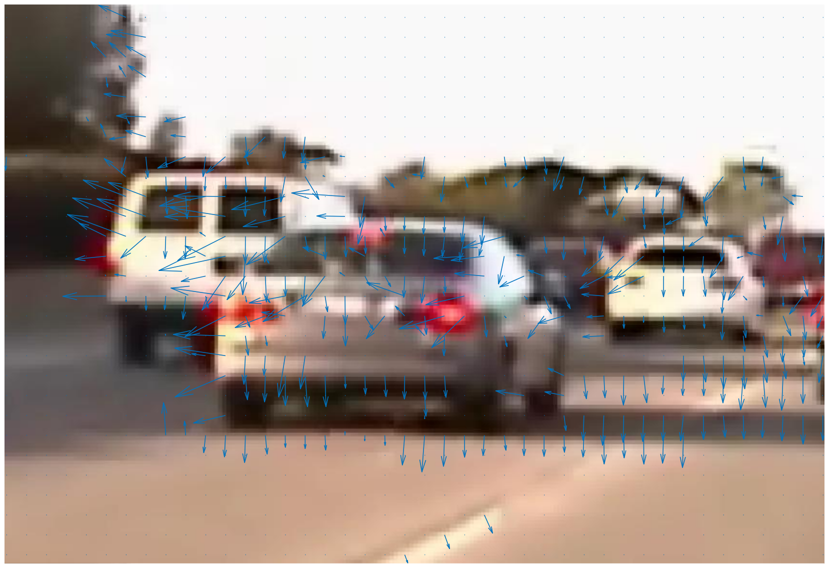


Figure 4.39: Laminar Flow

This algorithm was used to implement a tracker on each identified target vehicle. First, the centroid of the bounding box was determined and assigned as the initial value of the tracker "tag" along with a unique integer identifier. The tag was used as the point-of-interest on the target vehicle in which to track. With each passing frame, the optical flow of the tag was estimated to determine its new pixel location. This new location was then cross referenced against all new output bounding boxes to determine which box the tag fits into. When a correct match is found, the tag can be updated to match the new position of the centroid which mitigates skew in the flow estimate. This allowed the computer vision system to persistently track vehicles, even when the occasional missed detection occurred.

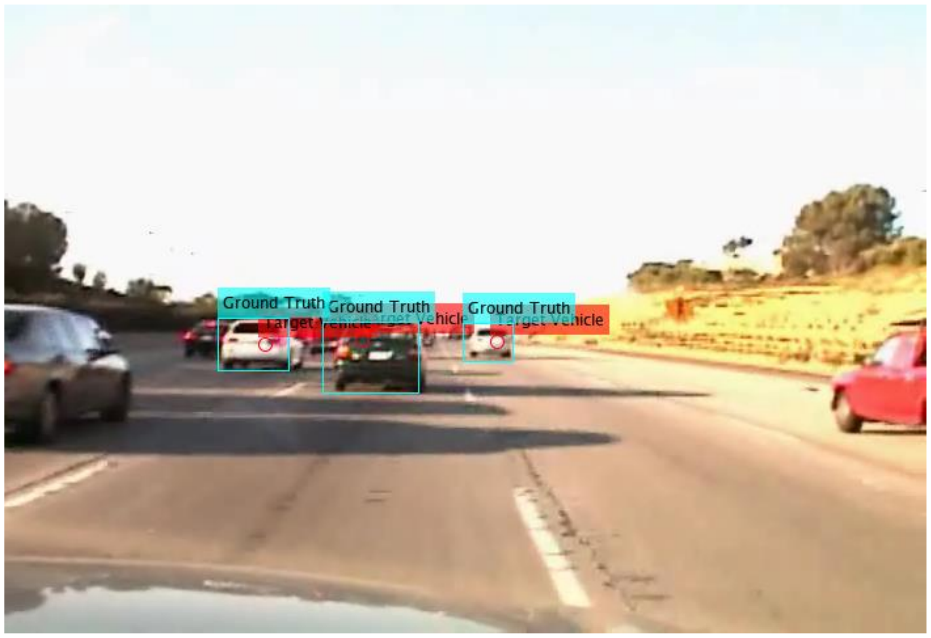


Figure 4.40: Output of Optical Flow Tracking using "Tags"

This algorithm required a high frame-rate to operate properly as one of the underlying assumptions of the model is that the time between images is small. However, this assumption breaks down in high-speed applications. The camera and target object are both moving with respect to world coordinates, which causes inaccuracies in velocity estimation. Additionally, change in camera position changes lighting conditions as new shadows, glare, and distortions are introduced. These problems were tuned using OpenCV parameters to compensate for error in the algorithm, however, the implemented model struggled with fast lateral movement due to the low frame-rate caused by the CNN detection method.

4.8 Summary

Using affine transformations, color masks, kernel filters, depth estimation, object detection, and object tracking methods discussed in this chapter, a computer vision pipeline is presented which can individually detect and track target vehicles and classify them based on lane lines position and longitudinal distance. Using this model of the forward-facing environment, kinematic and geometric relationships were formed to create ADAS applications such as LDW and FCW.

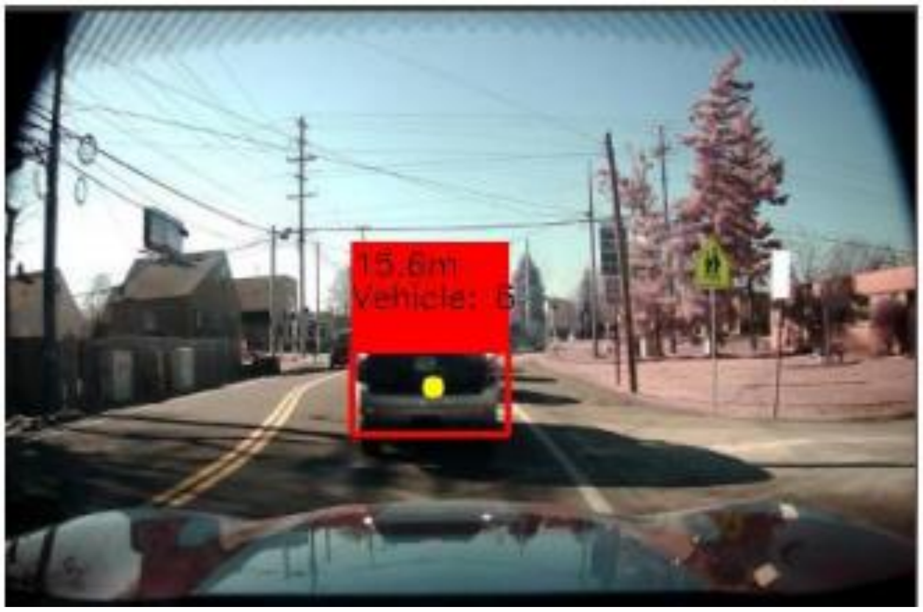


Figure 4.41: Object Tracking (Red/Yellow in Ego-Lane at <25m and >25m and green in another lane) [22]

Chapter 5: Validation and Results

Validation was performed using several layers of the X-in-the-Loop (xIL) process: SIL, HIL, and vehicle testing. xIL provides several advantages in validation. First, it allows the development team to test and catch errors in the software before it is built and deployed to target hardware in the vehicle. Second, it isolates variables. Removing the vehicle with powertrain, thermal, transient power, and other issues allows for isolated simulation in which the exact algorithm-of-interest can be evaluated in a closed system. Third, it saves time by running simulations in batches on a test bench without having to physically drive through a series of difficult-to-replicate driving scenarios. The levels of simulation and verification used in this work will be discussed as well as the results in terms of raw performance at the end-of-year EcoCAR 3 competition and in terms of contributions to automotive research. Technical challenges will be addressed along with potential solutions that are planned for the next EcoCAR project discussed in the *Conclusion* chapter.

5.1 Software-in-the-Loop (SIL)

Software portability played a critical role in the xIL process. The software architecture was structed such that it could be evaluated in a SIL simulation setup on generic Windows OS, a HIL simulation bench, and in-vehicle deployment without mass refactorization. To do this, calibration files were created for each simulation environment. The appropriate file was then found by the system and parsed at runtime so that algorithms were highly modular with respect to the simulation practice.

SIL was performed in one of two ways. First, VMs were used to emulate the appropriate hardware environment as part of the teams integrated development environment (IDE). This allows the team to test software iterations as they were produced, irrespective of the host OS and specifications. Test scenario videos, recorded on a mule-vehicle, were collected for real-time playback so new computer vision features could be tested using a database of realistic driving scenarios. C++ code was separated into compiler-specific, I/O code and generic algorithm code so that algorithms could be compiled and tested on most any Linux-based computer.

5.2 Hardware-in-the-Loop (HIL)

A custom computer vision simulator was built for low fidelity SIL and HIL simulation. Using this simulator, randomized driving scenarios could be generated and rendered in video format with a datafile containing ground truth data on target vehicle lane position and depth estimation.

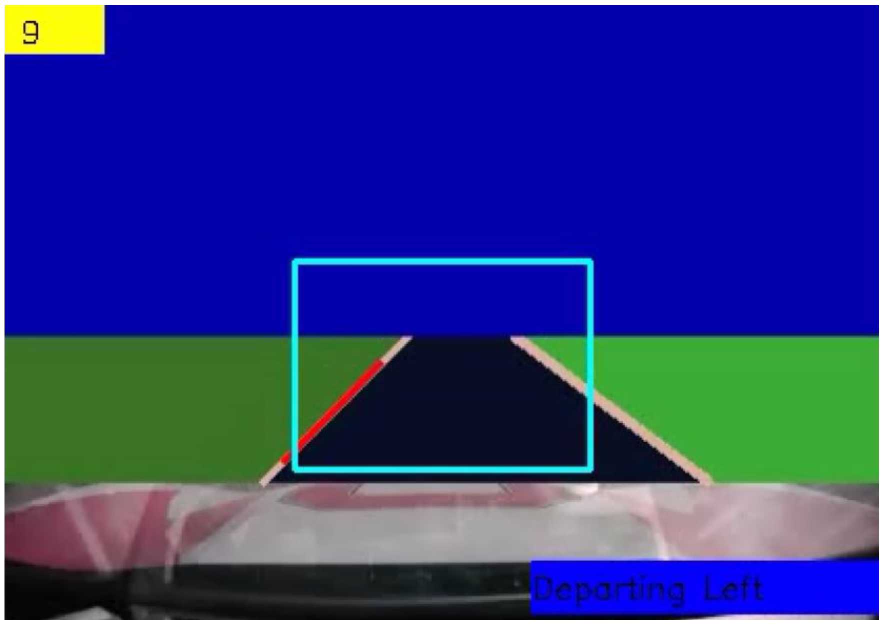


Figure 5.1: LDW Simulation using a Custom Simulator [22]

Similarly, an LDW simulator was derived to randomly create scenarios in which the ego-vehicle departed the host lane at a random angle of attack and speed. The lane lines could be any combination of white, yellow, dashed, and solid. An example scenario is shown in Figure 5.1 in which the video has been rendered and loaded into the SIL environment. The output of the lane detection algorithm is shown in red. As the line becomes increasingly close to the center of the image, the LDW algorithm determines that the ego-vehicle is departing the left side of the lane. A warning message is thus generated by the algorithm and displayed in the right corner of the screen in a blue box. Computer vision simulations were generally run in large batches with several dozen randomized scenarios. Since these scenarios were generally low fidelity, and not a realistic noise representation of real-world scenarios, the simulation results were used as a validation technique to ensure the development version of the code was still behaving as expected before vehicle testing. The simulator was often run on real hardware, an NVIDIA Jetson TX2, in a bench testing environment such that communication protocols could be validated across dozens of scenarios in a relatively short amount of time.

5.2.1 Component Validation

Similar to HIL, sensing components were isolated and evaluated experimentally. Fitment tests verified sensors and their mounts were sufficiently rigid to endure high speeds and vibrations. This was performed using a program to track displacement of a vibrant marker attached to the windshield. The marker was identified using a color threshold algorithm with the calibration shown in Figure 5.2.

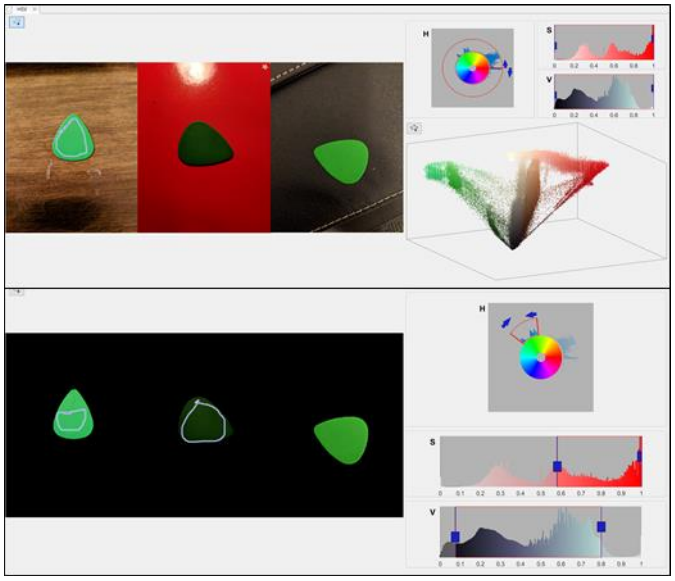


Figure 5.2: Color Threshold Calibration for Vibration Tracking

The displacement magnitude, between the markers perceived position in the camera frame and its calibrated position before testing, was evaluated over time. If the magnitude was above a determined noise level, it was due to a vibration causing relative movement between the camera and the windshield. A photo of the evaluation setup is shown in Figure 5.3.

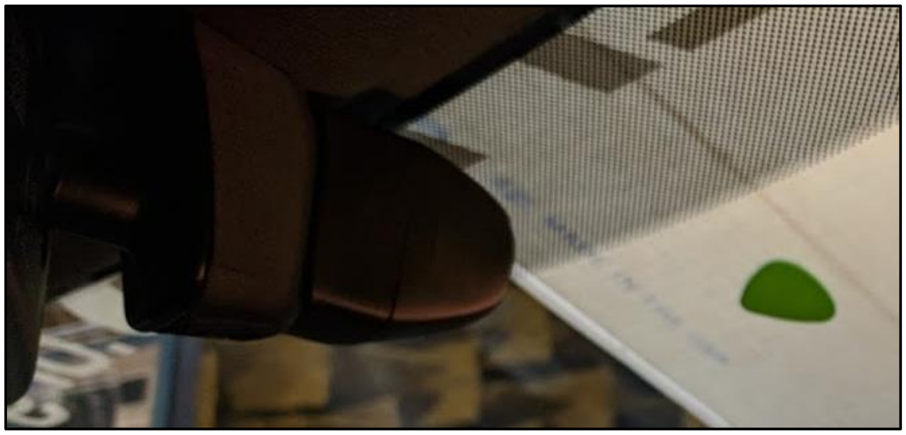


Figure 5.3: The Mono-Camera Detecting a Windshield Marker for Fitment Testing [22]

The vehicle was run on a chassis dynamometer in a baseline (stationary), realistic, and aggressive drive cycle to determine vibration patterns of the camera. The results of the test determined that there was no excessive vibration interfering with the camera's capturing ability.

5.3 Vehicle Testing

In-vehicle testing occurred during the last several months of the development process. During this time, startup/shutdown procedures, HMI use-cases, and ADAS features were tested. Startup and shutdown procedures were tested to ensure that ADAS controllers could repeatably initiate the expected boot-up sequence and begin processing sensor data. Similarly, the system was expected to perform a shutdown in a timely manner with uncorrupted log data. HMI use-cases were experimented to ensure the user could not crash the system with unexpected inputs. Dynamic testing was generally performed at Transportation Research Center (TRC) in Marysville, Ohio, where the team had access to several closed-course tracks. Single and multi-vehicle maneuvers were performed, such as the vehicle-following maneuver shown in Figure 5.4.

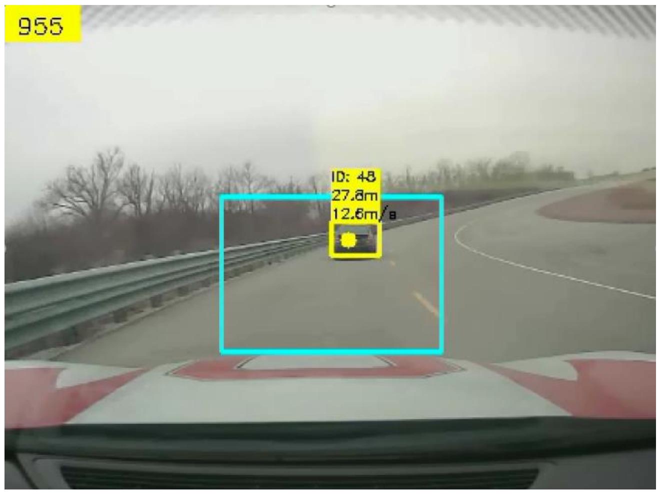


Figure 5.4: Testing the Perception System in the Ego-Vehicle on a Closed Track [22]

The system stored test logs with parameters, timestamps, and error messages from each test as well as a raw and post-processed video feed for each test performed. As the competition deadline approached, new version releases were created, weekly, as tests demonstrated stability in the software. Test logs were documented alongside each release to assist in calibrating perception parameters.

5.4 Results

The system was designed, developed, integrated into a vehicle, calibrated, and tested over the course of 1 academic year to compete in the end-of-year EcoCAR 3 Competition. The results will be discussed in terms of the system's relative performance during the ADAS Evaluation Event as well as its performance improvements during the development process.

5.4.1 EcoCAR 3 Year 4 Competition

The EcoCAR 3 Competition was split between dynamic events, performed using the teams' hybridized Chevrolet Camaros, and technical presentations. The dynamic events were largely performed at GM's Desert Proving Grounds in Yuma, AZ. This included the ADAS Evaluation Event in which each Camaro was to follow a target vehicle through a series of maneuvers performed on a flat oval track. Each vehicle was equipped with a GPS and IMU interface and was driven by a GM engineer. The GPU and IMU served as a method of ground truthing the measurements calculated by the Camaro's custom ADAS system. The system was evaluated on its ability to detect and track the target vehicle's relative position to the ego-vehicle and log it to the onboard diagnostics (OBDII) port. Additionally, points were allocated for driver feedback interfaces which included the HMI interface and ADAS applications discussed in Chapter 3: Figure 5.5 shows an annotated camera frame captured during the evaluation event.

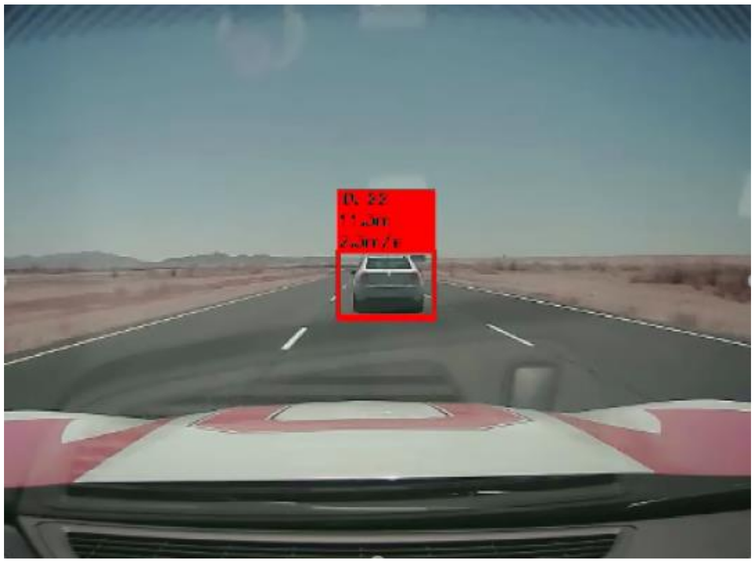


Figure 5.5: Vehicle Detection and Tracking during the Y4 Competition in Yuma, AZ.

The OSU EcoCAR team placed 4th place (of 16 schools) in this event. This was, in part, due to the accuracy of the computer vision and sensor fusion software as well as the HMI and vehicle integration of the system components.

The second half of competition was technical presentations which took place in Los Angeles, CA. Each team presented on engineering systems, project management, and communications strategies. For the ADAS presentation, the OSU team tied for 3rd place (of 16 schools). Combining year-long deliverables, the evaluation event, and the ADAS technical presentation, which summed to 100 points (10% of the competition), the OSU team placed 3rd overall with a score of 87/100.

5.4.2 Performance

The performance of the system was evaluated using three metrics. The first two, hit-rate and miss-rate, were introduced in Chapter 4: to measure the effectiveness of object detection and tracking. Leading up to competition, a set of basic one target vehicle maneuvers were performed to evaluate the average hit-rate and miss-rate with the latest software adaptations. The goal each week was to improve the object detection performance by increasing the hit-rate; thus, decreasing miss-rate as the two are complimentary. A plot of the detection performance over time is shown in Figure 5.6. This shows the improvement in object detection and tracking as new versions of the software were released.

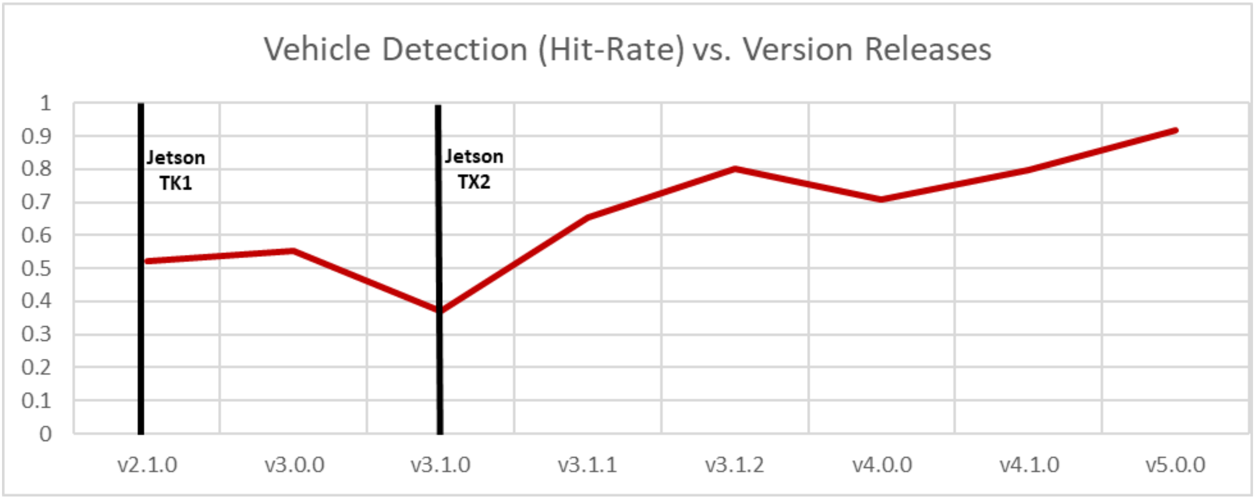


Figure 5.6: Hit-Rate Improvements over time during Development

Note, there are two vertical callouts indicating a change in hardware from an NVIDIA Jetson Tk1 to an NVIDIA Jetson TX2. The Tk1 was the original computer vision platform chosen for the system, however, it was determined that the Tk1 was incapable of running CNN algorithms due to incompatibilities between 64-bit deep learning frameworks and the Tk1’s 32-bit architecture. Thus, an upgrade to the 64-bit variant, TX2, was performed in order to run CNNs in the deployed software. This change created initial issues in the software, however, it presented the opportunity to implement new algorithms and filtering techniques which ultimately improved the hit-rate of the system.

Computational speed was also an important metric in evaluating the performance of the system. Figure 5.7 shows the computational speed, measured in FPS, as a function of version releases of the software over time.

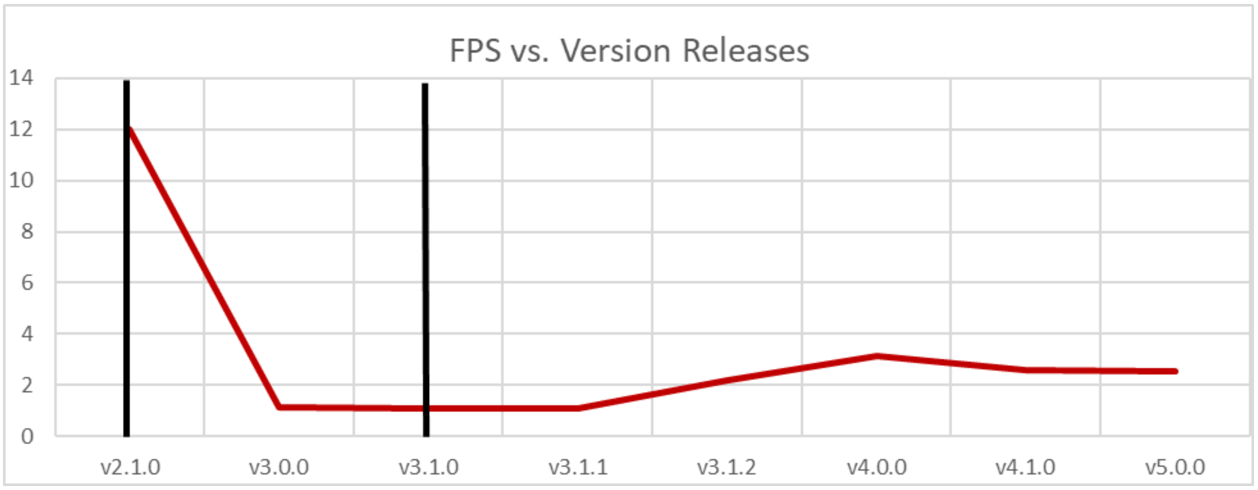


Figure 5.7: Frame-Rate Improvements over time during Development

Introducing a CNN into the software drastically reduced its FPS and only a small increase in frame-rate was able to be achieved. The low frame-rate was tolerable, but not desired. It had the largest effect on the optical flow-based tracker, which uses the underlying assumption that the time gap between frame captures is as small as possible. This had undesirable effects on tracking vehicles during lateral maneuvers, as their position changed rapidly in the camera frame.

5.5 Challenges Faced and Lessons Learned

The challenges faced in this work can be divided into two groups: technical challenges and design challenges. Technical challenges were implementation based, such as interfacing hardware and managing embedded processes. One of the more prominent technical challenges was organizing internal and external communication between controllers. A recommendation would be to use Robotic Operating System (ROS) as a middle-ware layer between embedded applications. This would parallelize processes and even sync devices across a well-defined network. It would also implement an inherent data logging tool [41]. Another technical challenge was bootup time. The ADAS components were only given power once the driver activated the ADAS system, which added latency while they waited for the system to become active. It is advised that the system, instead, receive power on vehicle ignition and enter a standby state until a feature is initiated.

Design challenges also presented a large development factor. The largest of which was the performance vs cost problem associated with achieving high detection rates in low computational time. The computer vision software did not take advantage of parallelization which meant that high-latency functions bottlenecked the whole pipeline. It is recommended that the team use threading or a middleware like ROS to communicate between programs. This way, a separate program could oversee reading frames from the camera, running complicated object detection functions, and returning bounding box coordinates without bottlenecking the rest of the software.

Mediocre simulation practices were used. It is recommended that the team use higher fidelity simulations which more accurately represent the scenarios expected by the system. A solution to this is to use model-based design using MATLAB & Simulink. In this project, MATLAB tools were used to assist development, but were not used as an integrated design solution. In future applications, it is advised that model-based design be used to ensure design decisions are data-driven based on performance in batches of simulation scenarios that follow industry standards.

After the conclusion of EcoCAR 3, a new competition, EcoCAR Mobility Challenge, began. EcoCAR Mobility Challenge is a 4-year competition to design a Mobility as a Service (MaaS) vehicle with both hybrid propulsion and Connected and Automated Vehicle (CAVs) technologies. Many of the lessons learned from the EcoCAR 3 ADAS project were implemented into design for a 360° perception architecture for CAVs. A component survey and mounting analysis was performed to evaluate sensors and positions that met project requirements. Using the results, architectures were formulated and evaluated using realistic scenarios taken from the home region of operation in Columbus, OH.

Table 5.1: Scenarios for System-Level Architecture Simulation

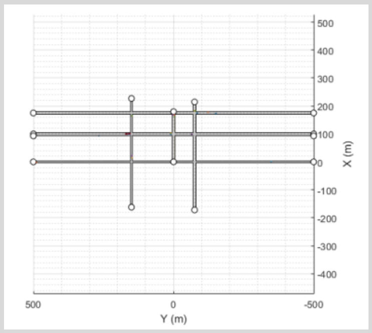
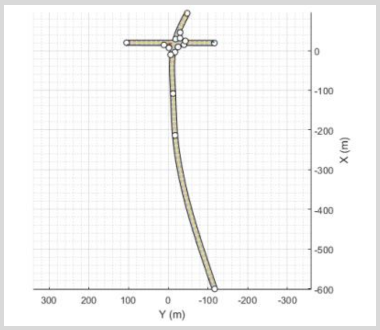
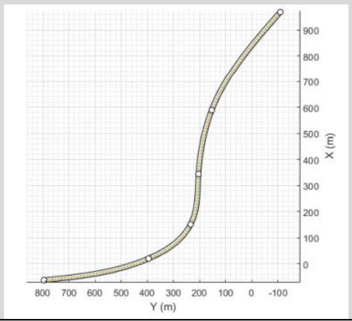
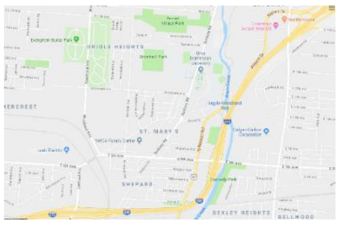
Downtown Columbus	Campus Area	Airport
Blocks between High Street and 4th Street 	Intersection of Olentangy River Rd and Ackerman Rd 	1000m section of I-670 towards John Glenn Columbus International 
		
Use Case: urban driving, heavy traffic	Use Case: suburban driving, campus driving	Use Case: ACC, highway driving

Table 5.1 shows three one-to-one replicas of scenarios used to evaluate sensor-set performance. Architectures were evaluated using models of supplier components evaluated in simulations with synthetically generated radar and camera data. Noise was also added to the measurements. This methodology was used to avoid many of the challenges faced in the ADAS project discussed in the main body of this work. This resulted in a much more refined design process which produced the sensor set architecture shown in Figure 5.8.

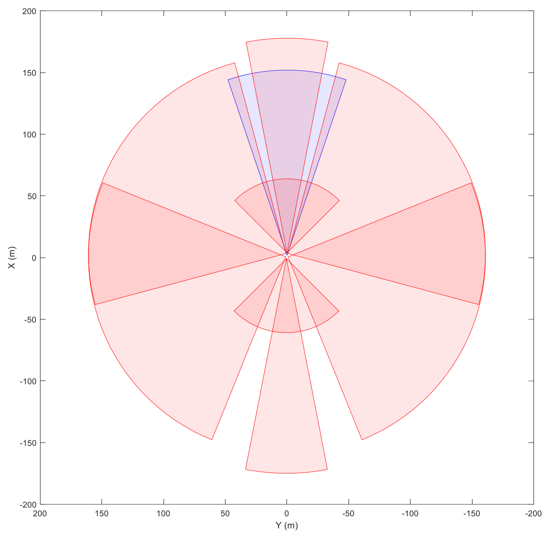


Figure 5.8: Perception System Architecture Birds-Eye-Plot

The team plans to use Simulink as the primary development platform such that algorithm design is driven by simulation data. Additionally, Simulink will be compiled using ROS and the Catkin toolchain such that multiprocessing, communication, and datalogging can be standardized across

ECUs in the system. These decisions were directly influenced by the challenges discussed in this section.

5.6 Summary

Using the xIL validation methods discussed in this chapter, the team was able to validate their software in various stages. This helped to isolate variables in the system and lead to more accurate algorithm design and calibration. Through the process, the team was able to refine their software and improve metrics like hit-rate and FPS. This was shown during the EcoCAR 3 Year 4 Competition in which the team placed 4th in the ADAS Evaluation Event. From this work, lessons learned were presented based on the technical and design challenges faced. These lessons played an important role in the design of a more complicated CAVs system which is likely to face additional technical challenges. However, the team has developed strong simulation practices using industry standards and software which will help the team meet design targets in algorithm design and calibration.

Chapter 6: Conclusion

This work outlined the scope and goals of a driver assistance system for a DoE-managed automotive technology competition, summarized the progression of driver assistance in the industry, and discussed the technical implementation of a vision-based ADAS and its performance. This serves as a useful overview of computer vision techniques such as image processing, filtering, object detection, and tracking which has direct relevance to many modern applications in driver assistance as well as autonomous vehicles and robotics.

A literature review outlined the state-of-the-art, showing the rise of “smart” technologies in the automotive industry. The industry is moving toward autonomous, electric vehicles in which vehicles themselves can sense their environment and making decisions to optimize their energy usage while getting passengers from point A to point B. To do this, perception systems will need to use the techniques discussed in this work to ensure the vehicles can generate a robust environmental model, consistently.

An overview of the experimental design and process is provided. This discussed the user features the system would perform and the components selected for the system. Detail was also provided for how the components were placed into the ego-vehicle and what software practices were used to develop, test, and deploy code.

The computer vision subsystem was explored in detail. Camera modeling was performed to characterize the interface between the vision software and the image sensor using the pinhole camera model. Next, a calibration process was performed to assign parameters to the camera model. This created a *camera matrix* of intrinsic and extrinsic parameters that characterize the mono camera used in this system. This was used to remove distortion from the input images as well as approximate depth of objects using camera geometry.

Image processing concepts including color spaces, transformations, and filtering were discussed in terms of their relevance to automotive computer vision. Color spaces were used to isolate objects of a certain color. Examples were provided for both affine as well as perspective transformations. Additionally, filtering examples were provided using notable kernel operators such as the Box Blur, sharpening kernel, and Gaussian Blur. These techniques were critical in the pre-processing stage of the software discussed in this work.

Next, automotive computer vision applications were presented. These applications use image processing, machine learning, and logic to extract information from images to better understand the environment. First, a lane line identification algorithm was presented which used color thresholding, region of interest isolation, Canny edge detection, and Hough transforms to model lane lines. Lastly, vehicle detection and tracking were discussed. Two detection algorithms, cascade classifiers and convolutional neural networks, were used to identify and localize vehicles within the camera frame using modern machine learning methods. Once detected, optical flow was used to develop a motion model of the detected targets to track their position from frame-to-frame with a unique identifier. Modeling lane line position and vehicle trajectories was a critical prerequisite for implementing lane departure warning and forward collision warning.

Validation techniques were discussed at several levels in the xIL process, namely: software-in-the-loop (SIL) and hardware-in-the-loop (HIL). These techniques were used to isolate variables and validate software in incremental levels of complexity. Furthermore, HIL allows the team to introduce error into the system and evaluate its response. Towards the end of the development cycle, the system was tested in-vehicle. This involved testing on a closed-course track at the Transportation Research Center in Murrysville, OH.

The system was evaluated during the ADAS Evaluation Event of the EcoCAR 3 Year 4 competition. Ohio State placed 4th in this event (of 16 schools). The performance results were as expected based on vehicle testing leading up to competition.

There were many challenges faced with the design and implementation of this system. However, at the end of the competition, the challenges faced were documented and reflected to determine future solutions that could help avoid and/or mitigate problems in the future. Many of these solutions have already been applied to the next EcoCAR competition, the EcoCAR Mobility Challenge, with a design of a connected and automated vehicles (CAVs) system. This work serves as a template for the design and implementation of a vision-based perception system with relevance to not only driver assistance, but autonomous vehicles and robotics as well.

References

[1] ARGONNE NATIONAL LABORATORY, "EcoCAR 3," Clique Studios, [Online]. Available: http://ecocar3.org/.

[2] NHTSA, "Human Factors Design Guidance for Driver-Vehicle Interfaces," NHTSA, 2016.

[3] NHTSA, *Society of Automotive Engineers (SAE) Automation Levels,* U.S. Department of Transportation.

[4] Grand View Research, "ADAS Market Size Worth $67.43 Billion By 2025 | CAGR: 19.0%," Grand View Research, 2018.

[5] SAE International, "Automated Driving, Levels of driving automation are defined in new SAE international standard J3016," 2014. [Online]. Available: http://www.sae.org/misc/pdfs/automated_driving.pdf.

[6] CARTELLIGENT, "Should your new car have active park assist?," 2018. [Online]. Available: https://www.cartelligent.com/blog/should-your-new-car-have-active-park-assist.

[7] B. Logan, "Police in the San Francisco Bay Area took an unusual approach to stop a Tesla operating on Autopilot as a drunk driver slept behind the wheel," *Business Insider,* p. 1, 2018.

[8] J. Clifton and W. Mattiace, "Nissan ProPILOT Assist technology reduces the hassle of stop-and-go highway driving, ready for U.S. launch," 20 July 2017. [Online]. Available:

https://nissannews.com/en-US/nissan/usa/releases/nissan-propilot-assist-technology-reduces-the-hassle-of-stop-and-go-highway-driving-ready-for-u-s-launch.

[9] "Discover Cadillac," 2018. [Online]. Available: https://www.cadillac.com/world-of-cadillac/innovation/super-cruise.

[10] "Discover Cadillac," 2018. [Online]. Available: https://www.cadillac.com/world-of-cadillac/innovation/super-cruise.

[11] AAA, "New Vehicle Technologies Double Repair Bills for Minor Collisions," 2018.

[12] IIHS, "IIHS Status Report Newsletter," IIHS, 2016.

[13] NHTSA, "Manufacturers make progress on voluntary commitment to include automatic emergency braking on all new vehicles," NHTSA, Washingto, DC, 2017.

[14] H. Green, "Radio-Controlled Automobile," *Radio News for November,* November 1925.

[15] E. E. Kimberly, "History of the Department of Electrical Engineering," The Ohio State University.

[16] Palm Beach Daily News, "This Automobile Doesn't Need a Driver," *Palm Beach Daily News,* 15 December 1966.

[17] S.-C. Lin, Y. Zhang, C.-H. Hsu, M. Skach, M. E. Haque, L. Tang and J. Mars, "The Architectural Implications of Autonomous Driving: Constraints and Acceleration," *ACM,* pp. 751-766, 2018.

[18] D. Grady, "The Vision Thing: Mainly in the Brain," *Discover,* June 1993.

[19] C. Wolff, "Doppler-Effect," [Online]. Available: http://www.radartutorial.eu/11.coherent/co06.en.html.

[20] University of Chicago Center for Cosmological Physics, "Radio Wave Basics," 2002. [Online]. Available: https://kicp.uchicago.edu/education/explorers/2002summer-YERKES/pdfs-sum02/background.pdf.

[21] A. Karpathy, *Unique mentions of deep learning frameworks in arxiv papers,* 2018.

[22] E. Stoddart, S. Chebolu and S. Midlam-Mohler, "System Engineering of an Advanced Driver Assistance System," in *SAE World Congress Experience*, Detroit, MI, 2019.

[23] Kalra, Nidhi; Paddock, Susan M., "Driving to Safety: How Many Miles of Driving Would It Take to Demonstrate Autonomous Vehicle Reliability," Rand Corporation, 2016.

[24] Waymo, "On the Road," 23 October 2018. [Online]. Available: https://waymo.com/ontheroad/.

[25] G. Motors, "For Crashes, Emissions and Congestion, Zero Is More," 2018. [Online]. Available: https://www.gm.com/our-stories/commitment/for-crashes-emissions-and-congestion-zero-is-more.html.

[26] AutonomouStuff, "Delphi ESR 2.5".

[27] Spinel, "Spinel 2MP full HD USB Camera Module".

[28] NVIDIA, "Harness AI at the Edge with the Jetson TX2 Developer Kit".

[29] NXP, "S32 AUTOMOTIVE".

[30] K. Hata and S. Savarese, "CS231A Course Notes 1: Camera Models," [Online]. Available: https://web.stanford.edu/class/cs231a/course_notes/01-camera-models.pdf.

[31] J.-Y. Bouguet, "Camera Calibration Toolbox for Matlab," Computational Vision at the California Institute of Technology, 2015. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib_doc/.

[32] Z. Zhang, "A Flexible New Technique for Camera Calibration," *Microsoft Research,* pp. 9-10, 1998.

[33] A. Rosebrock, "Find distance from camera to object/marker using Python and OpenCV," 19 January 2015. [Online]. Available: https://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv/.

[34] "OpenCV Canny Edge Detection," 18 December 2015. [Online]. Available: https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html.

[35] J. Canny, "A computational Approach to Edge Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 8, pp. 679-698, 1986.

[36] S. Sivaraman and M. M. Trivedi, "A General Active Learning Framework for On-road Vehicle Recognition and Tracking," in *IEEE Transactions on Intelligent Transportation Systems*, 2010.

[37] H. M. Zawbaa, H. El-Bendary, A. e. Hassanien and V. Snasel, "Semi-Automatic Annotation System for Home," in *10th International Conference on Intelligent Systems Design and Applications*, 2010.

[38] P. Viola and M. Jones, "Rapid Object Detection Using a Boosted Cascade of Simple Features," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Cambridge, Massachusetts, 2004.

[39] bpinaya, "DetectNetCars," [Online]. Available: https://github.com/bpinaya/DetectNetCars.

[40] OpenCV, "OpenCV Optical Flow," 2015. [Online]. Available: https://docs.opencv.org/3.1.0/d7/d8b/tutorial_py_lucas_kanade.html.

[41] Open Source Robotics Foundation, "About ROS," 23 October 2018. [Online]. Available: http://www.ros.org/about-ros/.