Development of an Autonomous Vehicle Platform

by

Hamid Tahir

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Mechanical and Mechatronics Engineering

Waterloo, Ontario, Canada, 2019

© Hamid Tahir 2019

Author's Declaration

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Autonomous vehicles and their related development are gaining a lot of traction as a promising up and coming technology. The Mechatronics Vehicle Systems lab at the University of Waterloo is well pioneered in the automotive industry and seeks to apply their knowledge and skills to autonomous vehicles. Having an autonomous vehicle development platform at the University allows for development and testing of state of the art algorithms that can potentially benefit the entire automotive industry.

An autonomous driving platform based on a Chevrolet Equinox is proposed in this thesis. Various types of sensors are installed on the vehicle and interfaced, allowing for full coverage of the surrounding environment. A software platform is developed which uses ROS and Matlab simultaneously, benefiting from the libraries, tools, and resources that come with both. The hardware platform is designed with simplicity and functionality in mind. Moreover, a simulation platform is used for testing various algorithms before real world implementation.

Various types of sensor calibrations are necessary to fully synchronize all the sensors on the platform spatially. A joint calibration method that allows for the simultaneous calibration of all 3D sensors sharing a common field of view is implemented. Specialized hand-eye calibration methods to calibrate the GPS navigation system to the LIDAR and camera sensors are explored. Furthermore, vehicle to everything interfacing is kept in mind and a calibration technique is presented in order to localize infrastructure mounted sensors to a GPS navigation system. The calibration techniques are tested and areas of improvement are revealed.

The developed platform is tested with the task of autonomous lane keeping. The steering wheel angle of the vehicle is controlled by the developed algorithm utilizing the camera and GPS navigation solution. The algorithm is tested in simulation with good results. Before real world testing, time synchronization between various devices on the platform, as well as testing of the actuators' controllers is performed. Finally, the lane keeping algorithm is tested on the developed platform on the University of Waterloo Ring Road. The system is able to autonomously steer around the majority of the road which is approximately a 2.5 km distance.

Acknowledgements

I would like to thank Professor Amir Khajepour and Professor William Melek, my supervisors, for their invaluable insights and guidance. Their knowledge and willingness to share it has kept me on the right track for the last two years.

I would also like to thank the lab technicians at the Mechatronics Vehicle Systems lab, Jeff Graansma and Jeremy Reddekopp, for their support and help in testing the developed platform. The co-operative undergrad student, Brandon Johnson, aided in the construction of hardware elements used on the platform for which I am also very grateful.

Finally, I would like to thank Professor Ehsan Hashemi and Dr. Chen Tang for their support in my research and providing insightful ideas and critiques regarding my work.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Autonomous vehicles are gaining traction in the automotive field due to the foreseeable improvements in driving safety, efficiency improvements and other benefits. The Mechatronics Vehicle Systems lab (MVSL) seeks to develop an autonomous vehicle platform for the development of state of the art autonomous driving and driver assistance algorithms.

1.1 Motivation

With many organizations developing their own autonomous vehicles and logging thousands of miles travelled autonomously, it has become clear that this technology is the next stage for the automotive industry. According to Transport Canada's National Collision Database, 1898 fatalities occurred in 2016 due to motor vehicle collisions [1]. Fully functional autonomous vehicles will significantly improve this statistic by removing the factor of human error. Alongside driving safety and crash reductions, the technology potentially improves driving efficiency and congestion as well as travel behavior. The improvements add up to around \$2000 to \$5000 of savings per year [2].

However, developing autonomous vehicles is a difficult task and comes with its own set of challenges. The complexity of a driving environment, coupled with the differing types and quantities of sensors required to fully understand that environment, often limit the safe driving speed of the autonomous vehicle. Moreover, a fully autonomous vehicle requires better capability and performance from the hardware systems which lead to an increase in development cost and time. Developing an autonomous vehicle for a limited scenario such as highway driving becomes an easier task where a smaller subset of sensors need to be used and vehicles generally follow a set of predefined rules. Tackling a smaller task also alleviates hardware requirements and costs.

The MVSL has developed solutions to many automotive control and systems challenges and intends to develop a platform used by students and professionals working on state of the art autonomous driving hardware and software. The lab seeks to develop a fully autonomous vehicle capable of high speed driving and desires to address the current limitations present in similar platforms. It is believed that the application of the skills of the MVSL will result in the development of an autonomous vehicle platform that addresses some of the gaps of similar existing platforms and will benefit the field of autonomous driving in general. Furthermore, the platform will provide students exposure to autonomous vehicles and their related technologies which is beneficial for the growth of the field.

1.2 Objectives and Contributions

The objective of this thesis is to present an autonomous driving platform developed for implementing and testing algorithms for self driving. The platform is split into the hardware and software components as well as the interfacing tasks required for everything to work together. Moreover, the sensors used in the platform are to be calibrated and their coordinate frames resolved with respect to a common coordinate frame to enable sensor fusion techniques. Finally, the developed platform will be employed for the task of autonomous lane keeping around the University of Waterloo Ring Road. The approach taken in the development of the platform is to get the platform working as quickly as possible with extensible and functional hardware and software architectures. Individual elements of the platform will be improved when specialized focus is given to them in the future.

The first contribution of this thesis is the platform design and implementation using popular sensors. The system is presented in full with sensor placement and interfacing explained. Hardware architecture of the platform is developed for a research focus which due to a lack of specialized components, is easily reproducible. The second contribution of the thesis are the insights gained from implementation of the sensor calibration techniques. Through the performed experiments, successful calibration methods as well as the limitations of the unsuccessful calibration methods can be identified. Lastly, the thesis contributes a simple lane keeping approach that utilizes the platform and interfaced sensors to steer autonomously on a known track.

1.3 Outline

Chapter 2 presents a review of existing literature pertaining to autonomous vehicles specifically relating to platform design, calibration between sensors, and lane keeping methodologies. The chapter also presents background concepts relevant to the full understanding of the thesis.

In chapter 3 the software and hardware platform designs and requirements are presented and explained. Next, interfacing between the software and hardware components and the necessary interconnections are outlined. Finally, the need for synchronization between sensors and possible approaches are discussed. The simulation platform used on for testing of the developed algorithms for the autonomous vehicle platform is discussed in chapter 4.

The topic of sensor calibration is presented in chapter 5. In this chapter, sensor calibration methods pertaining to the sensors used on autonomous vehicles are explained and their importance is mentioned. A calibration method relevant to infrastructure mounted stationary sensors is also presented. The methods are implemented in simulation and insights for implementation on the real platform are obtained. Successful calibration results will allows sensor fusion and vehicle-to-infrastructure interfacing techniques to be used on the platform.

Chapter 6 explains the implementation of the lane detection and lane keeping algorithms. These algorithms are tested and refined through simulation and made ready for real world use.

The experimental setup is explained in chapter 7. The hardware and sensors used in the platform are identified. Interfacing between hardware and software with regards to the actual sensors is also outlined. Moreover, tested calibration techniques are implemented and the sensors are localized to a common coordinate frame.

Chapter 8 presents the lane keeping results obtained on the developed platform. The issues encountered and the corrections made are presented step by step in order to achieve a successful final result.

Finally, chapter 9 presents a conclusion of everything covered in the thesis. Future work for the developed platform is also presented in this chapter.

Chapter 2

Literature Review and Background

An autonomous vehicle can be defined to be any mobile platform that safely accomplishes predefined tasks while facing unpredictable events. With such a loose definition, it is no revelation that autonomous vehicle platforms tend to be very different from one another, from different sensor selections to different algorithms trying to accomplish the required tasks. Moreover, autonomous vehicles can span a variety of different goals and are not only limited to street or highway driving. For example, a Mars rover is an autonomous vehicle but has very different goals than what the developed platform seeks to target. Therefore, the literature review for this thesis is limited to the topics covered by the thesis. First, a general review of some existing and famous autonomous vehicle platforms will be provided. Following this, calibration techniques covering the relevant sensor types will be covered. Finally, some of the existing lane detection and lane keeping techniques will be reviewed. Some of the specific algorithms and methodologies utilized in this thesis will also be covered including brief introductions for each of the common autonomous vehicle sensors.

2.1 Autonomous Vehicle Platforms

Many organizations have been developing autonomous vehicle platforms for over a few decades. The development of some of the most influential platforms was set into motion when the Defense Advanced Research Projects Agency (DARPA) issued the DARPA Grand Challenge for self driving cars in 2003. The first winner of the challenge in 2005 was the autonomous vehicle Stanley. Based on a Volkswagen Touareg R5, Stanley utilizes a DC motor for steering actuation and a custom interface for acceleration and braking actuation with data transfer through a Controller Area Network (CAN) bus interface. Stanley utilized an array of laser scanners, a video camera, radar, wheel encoders, a Global Positioning System (GPS) and an Inertial Measurement Unit (IMU) to navigate the off road terrain in the competition [3]. The platform supported mapping, planning, tracking, and control algorithms all working together to achieve first place. The development of Stanley led to a series of advancements pertaining to autonomous navigation that are still relevant today.

In 2007, DARPA held the Urban Challenge which involved vehicles driving through a mock urban environment featuring interactions with other traffic and required the vehicles to perform complex driving maneuvers. Successful vehicles tended to employ high precision GPS systems and utilized active sensors such as radar and LIDAR for obstacle detection and tracking over the more difficult to interpret vision data [4], [5], [6]. The competition led to the development and implementation of advanced motion planning techniques as vehicles performed complex maneuvers in traffic. Furthermore, the competition led to the development of open source datasets such as the popular KITTI dataset based on the AnnieWAY autonomous platform [7]. The KITTI dataset is widely used to test algorithms related to mobile robotics and autonomous driving providing LIDAR, camera, radar and high precision GPS and IMU data for a diverse set of diving scenarios. The KITTI dataset has led to the development of other datasets more focused on providing large amounts of rich data for learning based algorithms [8], [9]. These positive outcomes from the DARPA challenges have helped to advance the field of autonomous driving.

Over the years, more research purposed platforms have emerged. Many research platforms tend to implement all relevant sensor modalities including LIDARs, monocular and stereo cameras, radars, and inertial GPS systems [10], [11], [12]. These platforms facilitate students and researchers working on the entire range of mobile robotics and autonomous driving algorithms. However, industry designed autonomous platforms tend to have limited sensor modalities that are selected partly based on their ability to be used in a production vehicle. The Bertha Benz vehicle, based on a Mercedes Benz, utilizes only radars and cameras alongside accurate digital maps [13] for autonomous navigation. LIDAR systems are omitted in the platform even though they can augment the inertial GPS based localization and improve its accuracy [14].

2.1.1 Computing Platforms

There are several different options available for an autonomous vehicle computing platform each with its own advantages and disadvantages. This makes the choice of the platform difficult as there is no ideal choice suitable for all scenarios and many trade-offs need to be carefully considered. A main distinction in computing platforms is the choice of using standard computers vs. existing solutions provided by manufacturers and chip designers. Standard computers can be designed as per performance requirements and are scalable in performance and storage. Additionally, they provide ease of development since they allow the use of standard and familiar tools and libraries with well defined resources resulting in quicker implementations. Some of the downsides include high power consumption and heat dissipation. Moreover, the solution is not automotive grade and is not designed with driving safety in mind. On the other hand, solutions provided by manufacturers include systems based on Graphics Processing Units (GPUs), Digital Signal Processors (DSPs), Field Programmable Gate Arrays (FPGA), and Application Specific Integrated Circuits (ASICs) each with their own advantages and disadvantages to be considered. In general, these specialized computing platforms have lower power consumption and are often specifically designed for automotive applications. Some of the downsides of these solutions are that they lack the ease of use aspect that general computers have and generally take more time to implement and have higher costs and learning curves [15], [16].

While both options are viable, there is an argument for the use of general computers for research based platforms due to their versatility, developmental ease and scalability. This is typically the case in literature with computing platforms described in many autonomous vehicle papers. Team AnnieWAY uses an off the shelf quad core computer for software processing tasks [5]. The platform developed in [11] implements small form factor computers based on Intel QX9300 processors with necessary components implemented in a custom chassis in the trunk of the vehicle. The Deeva autonomous vehicle computing platform consists of 17 general computers and three embedded boards [17].

2.2 Sensor Calibration Techniques

Sensor calibration is a topic with a wide breadth of applications. It can refer to the techniques applied to a single sensor in order to characterize some aspect of the sensor. More generally, sensor calibration in the field of mobile robotics and autonomous driving refers to the techniques utilized in finding the rigid body transformations between the mounting locations of multiple sensors. Knowing accurate rigid body transformations between all of the sensors on a vehicle or robot allows for all information gathered by the sensors to be represented in a common reference frame. This can be crucial is knowing the accurate position and motion characteristics of objects of interest with respect to the vehicle or robot. Moreover, representing the locations of detections accurately with respect to one another is required for sensor fusion techniques to be successful. Overall, sensor calibration techniques are a core aspect of autonomous vehicles and mobile robotics.

Camera calibration is perhaps the most common type of calibration method and is utilized across many fields. Current calibration methods are based on the work of Zhang [18] who developed the technique for identifying important camera parameters based on images of various orientations of a planar board. Camera calibration algorithms model the lens distortion effects as well and the same calibration routine can simultaneously solve for distortion parameters. Zhang also contributed extrinsic calibration methods for calibration of a camera with a laser range finder [19]. The method determines the transformation between the camera and the range finder by solving for physical constraints present in the calibration design and minimizing a re-projection error for further refinement. Stereo camera extrinsic calibration is based on similar methodologies and principles.

Calibration of a pair of sensors at a time is common in literature. Due to the diverse sensor modalities and types of information being collected by each sensor, different sensor combinations tend to have differing calibration methods. Moreover, it may be required to calibrate a sensor from the same modality. Such is the case of an autonomous vehicle with multiple cameras and LIDARs covering different field of views around the vehicle. For that vehicle, possible pairwise calibrations include calibrating a camera to a camera, a LIDAR to a LIDAR, and a LIDAR to a camera, each with different techniques being employed.

When a series of pairwise calibrations are performed, errors can accumulate as sensor data is transformed to a common coordinate frame. Methods that can calibrate more than two sensors simultaneously can save a large amount of time, especially if the calibration needs to be repeated often, but can also reduce overall calibration error as fewer calibrations need to be performed. The method presented in [20] calibrates multiple cameras based on parameter adjustment and silhouette observations. In [21], it is proposed to create sensors groups that provide 3D observations of a calibration target and to simultaneously calibrate all sensor groups through the geometric constraints. However, it is not always possible to create these groups and a specialized pairwise calibration method may be simpler and easier to implement.

Calibrating sensors to inertial GPS systems is akin to hand-eye calibration of robotic manipulators where a sensor such as a camera is rigidly attached to a robotic arm. Early works provide closed form solutions for the method [22], [23]. More recent works provide numerical optimization methods to simultaneously solve for the rotation and translation [24], [25]. Hand-eye calibration is applied traditionally to calibrating sensors mounted on the end effectors of robotic arms. Recently, the method is being applied to autonomous vehicle sensor calibrations with GPS and IMU systems due to similarity in geometry to the traditional problem. The navigation system can be calibrated using this method with any sensor that allows for odometry to be estimated through its data. These generally include LIDARs and cameras. Moreover, recent advancements in such sensor calibrations are focusing on performing these calibrations automatically utilizing SLAM, visual odometry, and LIDAR odometry methods to determine the transformation while driving in a normal environment.

Calibration techniques also include automatic methods which generally interpret normal operation data for common features between the sensors involved and optimize some metric to calculate the transformation. Such methods are more difficult and involved compared to specialized calibration methods, but don't require a special calibration procedure to be performed. For the case of LIDAR to camera calibration, manual calibration methods employ specially designed targets from which corresponding features can be detected by each sensor [26], [27]. Automatic methods use environmental data and don't require specialized targets [28], [29]. Advanced methods can detect and correct minor drifts in transformations in real time.

2.3 Lane Detection and Lane Keeping Methodology

The lane detection perception problem has seen various sensor modalities being utilized over the years in the solution to the problem. One school of thought advocates the use of vision systems in solving the problem due to their similarities with human perception and ability to extract similar data as that used by human drivers. This is valid reasoning considering lane markings are designed to work in the vision modality for humans based mainly on colour and reflectivity. Moreover, there is a support for using vision sensors from an evolutionary point of view in that cameras are a mature technology available cheaply and readily even at the consumer level [30]. That being said, lane markings can be detected by light based detection and ranging (LIDAR) sensors based on the intensity cues gathered from the lane marking which otherwise have no structural cues. In fact, LIDARs operate with active light and do not suffer from lighting based issues that commonly affect vision sensors. This comes at the cost of expense as LIDARs, especially ones that produce very dense point clouds, are extremely expensive in relation to the cost of a machine vision camera. The stereo camera modality is a solution in between the vision and LIDAR modalities where two cameras are utilized in order to obtain depth information of the environment. This allows for additional cues to be used in lane marking detection but suffers since depth maps tend to be highly dependent on surface texture and lane markings are generally smooth. However, they are a cost effective way at obtaining a portion of the accuracy and reliability offered by the LIDAR sensor. Vision sensors tend to be the most utilized modality for lane marking detection but generally fail to fully solve all the complexity and variation that can be present in lane markings.

Of course the combined GPS and IMU system is another modality that can be used to navigate on a road without needing lane marking perception. The working principle is to follow a known map of the driving environment based on real time localization using the navigation system. This method sees common use in many systems due to its relative simplicity and common availability of GPS and IMU systems. That being said, the ability of a vehicle to follow a map well depends on both the accuracy of the map and the accuracy of the navigation system both of which increase the overall system cost.

Many different approaches to vision based lane detection exist. Methods differ in how they detect the lane as well as the type of model that is used to fit to the lane. Perhaps the most common detection method is based on the Hough transform which can isolate features in an image by shape. The method is utilized in [31] and [32] which fit a parabolic lane model to the detections and [33] and [34] which fit spline based models. The procedural lane detection methods typically begin by inverse perspective mapping the image to a birds eye view. Processing the birds eye view image is beneficial since perspective effects are removed and the lanes become parallel. Moreover, knowledge of the expected location of the lane markings can be utilized and sub regions in the image can be processed for optimization. Other detection methods include template matching in which a template model of the lane is identified with the camera image [35]. Moreover, recent approaches utilize neural networks which can detect more advanced and human used features [36], [37]. These approaches benefit from large amounts of training data to improve detection accuracy and can outperform non learning based methods when trained well.

For lane marking detection using LIDAR, methods tend to threshold the birds eye view point cloud based on reflectivity of the markings. In [38], the authors set a dynamic region of interest for candidate lane marking points in a birds eye view LIDAR point cloud. The region is based on predicted lane parameters from the previous detections. The general pipeline is simlar to that of the vision system with the main difference being the use of a different method for feature extraction. Both vision and LIDAR based solutions tend to employ model fitting and tracking of the lane markings in some way to add robustness and reliability to the lane marking detection.

Lane keeping methodologies often utilize PID steering control based on lateral error from lane center and heading error from lane heading. A popular controller is the Stanley controller where the control law utilizes a proportional gain to minimize the cross track error while following the heading of the road [3]. Advanced controllers use Model Predictive Control (MPC) for better results by predicting changes in the road in the near future [39], [40]. There are a slew of different techniques available that fall somewhere in between these methods often with minor differences between them.

2.4 Autonomous Driving Sensors

Standalone autonomous vehicles rely on on-board sensors for the necessary information required to function. Sensors are used to detect obstacles in the vicinity as well as to help the vehicle navigate. The sensors implemented on the platform are explained in the following sub-sections.

2.4.1 Camera

Camera sensors are one of the most explored sensors available. With applications in a multitude of fields this type of sensor benefits from cheaper cost, well defined resources, and well researched algorithms. The working principle of the camera is the redirection of light onto the image sensor which creates a 2D representation of the surrounding 3D scene. With an abundance in the types of cameras and lens available, the task of camera selection becomes an application specific task. The main goal of the camera, currently, is lane detection and lane keeping and so the camera and lens are selected with this task in mind.

Pinhole Camera Model

The pinhole camera model is an ideal model that relates 3D points in the camera field of view to their 2D projections on the camera image. Through the model, an intrinsic matrix, A, and an extrinsic matrix, H, are used in the mapping of the 3D scene coordinate, X, to the 2D image coordinate, x. The mathematical relationship derived from the model can be expressed as shown in Equation 2.1. The intrinsic matrix characterizes the image sensor with the focal point (f_x, f_y) and the camera center (c_x, c_y) and is expanded in Equation 2.2. The extrinsic matrix is expanded in Equation 2.3 and describes the camera's location in the world. The basic pinhole camera model can be expanded to include the nonlinear radial and tangential distortions.

$$x = AHX \tag{2.1}$$

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

$$H = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.3}$$

Calibration

Camera calibration is referred to the process by which the intrinsic and extrinsic matrices of the camera are determined. Software packages such as Matlab and OpenCV provide implementations of the camera calibration based on the work of [18]. This approach uses different poses of a planar calibration target, typically a chessboard, in order to compute the desired matrices. The very common calibration process for a single camera is shown in Figure 2.1. To perform the calibration, a planar chessboard is placed in the field of view of the camera. The camera pose is changed to excite the 3D rotation and translation along all the axis and a set of images of the chessboard are captured. Next, using Matlab's camera calibration tool, the camera is calibration and the various poses of the chessboard with respect to the camera are shown as well as the reprojection error. The reprojection error measures the point to point error of the chessboard corners projected onto the image using the calibration matrices with the detected corners and is an indication of how good the calibration is. Ensuring that the calibration target is planar with equal sized squares is paramount to a good calibration.

2.4.2 GPS and IMU

GPS provides continuous 3D position and timing data for a receiver. The position measurement is derived from the trilateration of distances of the receiver from multiple satellites. According to [41], the mean accuracy of standard smart phone GPS systems is approximately 5 meters in radius in an open environment.

Two common methods used to improve the accuracy are the Real Time Kinematic (RTK) and the Differential GPS (DGPS) systems. The RTK system uses a local base station that sends real time measurement corrections to the mobile receiver and is capable to achieve centimetre level accuracy. DGPS also uses corrections from a reference station, yet achieves an accuracy in the decimetre range due to an alternative calculation method [42].

IMU provide orientation data complimentary to the 3D position coming from the GPS. Current high end navigation solutions integrate GPS and IMU solutions using the Kalman

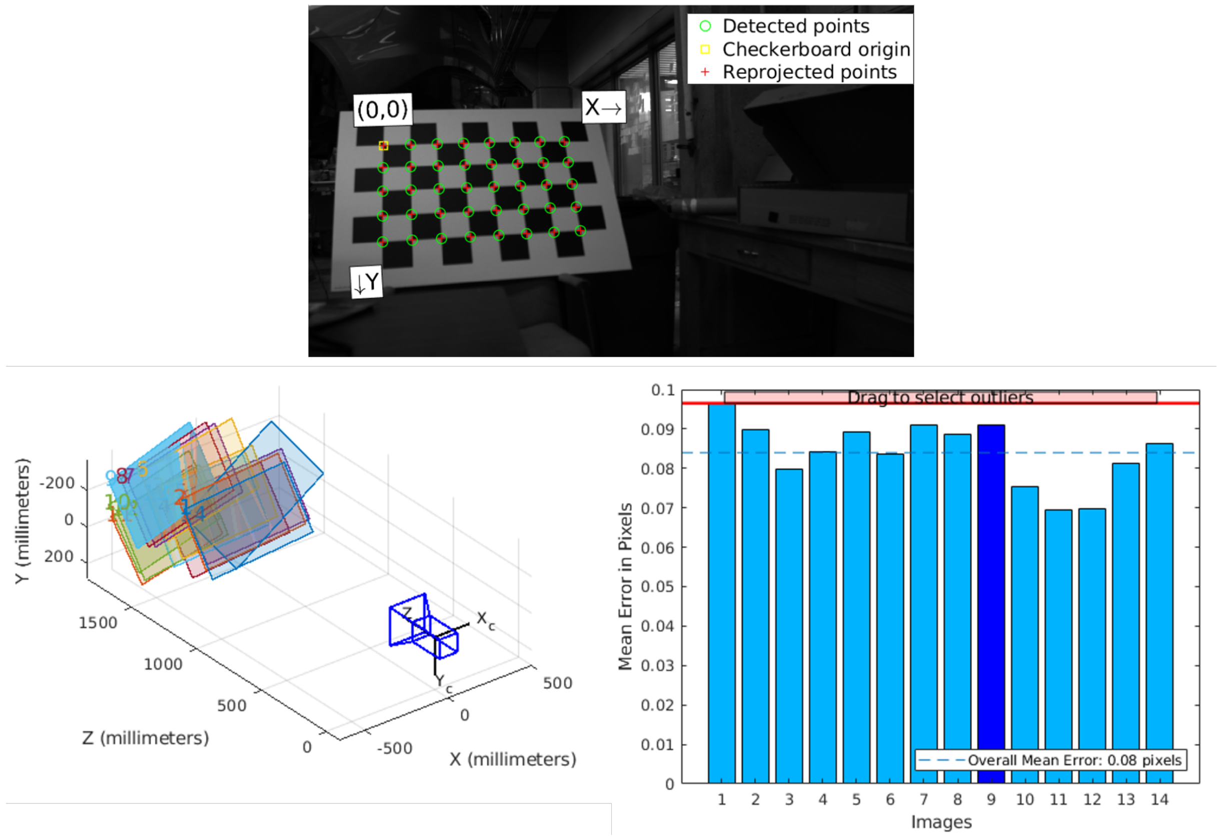


Figure 2.1: Camera Calibration Process in Matlab Depicting Chessboard Corner Detection (top), Chessboard Poses Used for Calibration (bottom left), Calibration Reprojection Error (bottom right)

filter to provide accurate position and orientation data robust to GPS dropouts. This can be achieved since the bounded accuracy of GPS data can be used to calibrate and provide a bound to the IMU data while dead reckoning using the IMU can estimate the position during GPS dropouts [43]. Moreover, the fused system is able to provide position and orientation at the high data rate of the IMU when the GPS system is slower.

Converting to UTM Coordinates

GPS systems provide latitude and longitude measurements in degrees based on an ellipsoidal model of the earth. It is not straightforward to integrate this with Cartesian data coming from other sensors. A commonly used solution is to project the GPS coordinates into universal transverse Mercator (UTM) coordinates which represent the earth as a cylindrical coordinate system discretized into a set of, approximately, Cartesian systems [44]. Functions to perform the conversion are available for software packages such as Matlab and are utilized in this work.

2.4.3 Radar

The basic operating principle of radars is the transmission of a signal and measurement of the returns in order to calculate distance to and relative velocity of objects in the field of view. Put simply, the signal when reflected off of a target is altered through the addition of noise, loss in signal strength, change in phase, etc. Additionally, the delay in receiving the reflected signal and in the case of a moving target, the Doppler affect imposed on the system, can be used to determine the target's range, velocity and, in advanced systems, characterise the target in a range of classes [45].

2.4.4 LIDAR

LIDAR is a relatively new technology and achieves a similar in nature result as Radar sensors, but utilizes pulses of light instead of radio waves [46]. Companies such as Velodyne have begun to utilize an array of lasers and detectors with a revolving head that can accurately create high definition maps of the surroundings. The technology enables for the creation of dense point clouds of the surrounding environment that provide position and intensity information encoded in each point. The LIDAR sensor can be used for many tasks ranging from obstacle detection to odometry.

2.5 3D Rigid Body Transformations

The autonomous driving platform utilizes multiple coordinate frames. Each sensor provides data relative to its own coordinate system. The vehicle may be represented by its own coordinate system, and there may be a map available that is defined based on a coordinate system fixed on the earth. In such a system it is often necessary to represent a point known in one coordinate system in another. For example, when fusing data from two sensors, it is important to represent the data in a common coordinate system instead of the separate coordinate system of each sensor.

Rigid body transformations can be used to represent the combined translations and rotations present in such geometric transformations. In this thesis, homogeneous transformation matrices will be used to represent scale invariant 3D rigid body transformations. As shown in Equation 2.4, to convert a point in coordinate frame A, p_A, to a point in coordinate frame B, p_B, the point is pre-multiplied by the transformation matrix T_A^B which is a 4 by 4 homogeneous matrix. Equation 2.5 shows the form of T_A^B if the transformation from frame B to A consists only of translations of a, b, c units along B's, x, y, and z axis, respectively. Equation 2.6 to Equation 2.8 show the form of T_A^B under a pure rotation of θ, about the x, y, and z axis of coordinate frame B, respectively [47]. Lastly, the translations and rotations may be combined by multiplying the homogeneous matrices together with the earliest transformation on the left hand side. An example is shown in Equation 2.9 where the transformation from B to A is a rotation around the x axis by 60°, followed by a rotation around the z axis by 25°, and finally a translation about the y axis of 4.5 units.

$$p_B = T_A^B \cdot p_A \tag{2.4}$$

$$trans(a,b,c)|T_A^B = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

$$rot_x(\theta)|T_A^B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$rot_y(\theta)|T_A^B = \begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$rot_z(\theta)|T_A^B = \begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$T_A^B = \begin{bmatrix} 0.91 & -0.42 & 0 & -1.90 \\ 0.21 & 0.45 & -0.87 & 2.04 \\ 0.37 & 0.78 & 0.5 & 3.53 \\ 0 & 0 & 0 & 1 \end{bmatrix} = rot_x(60) \cdot rot_z(25) \cdot trans(0, 4.5, 0) \tag{2.9}$$

2.6 Solving Homogeneous Transformation Equations of Form: $AX = XB$

In robotics, the matrix equation $AX = XB$ comes up in the calibration of robotic arms with sensors mounted on the end effector. It is used to figure out the transformation between the robot hand and the sensor. However, the same technique can be applied to cases of calibration pertaining to autonomous vehicles where there is a similar fixed rigid body transformation between two sensors. Such a calibration is referred to as hand-eye calibration and a multitude of methods exist to solve problems of this form.

The paper [24] presents a closed form solution to the problem which first solves for the rotation component of the homogeneous transformation matrix X, and then solves for the translation component. Rotations are represented as unit quaternions. Homogeneous matrices are used as they simplify such equations by combining the effect of rotation and translation in one matrix. If n different poses of the hand-eye device are used, there are $n-1$ equations available as shown in Equation 2.10, where A_{i-1i} denotes the transformation from position $i-1$ to i of the eye frame and B_{i-1i} denotes the similar transformation for the hand frame. The paper also presents a non-linear optimization technique to simultaneously solve for the rotation and translation components of the X matrix.

$$\begin{aligned} A_{12}X &= XB_{12} \\ &\vdots \\ A_{i-1i}X &= XB_{i-1i} \\ &\vdots \\ A_{n-1n}X &= XB_{n-1n} \end{aligned} \tag{2.10}$$

2.7 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt algorithm is a method for minimizing $F(x)$ where the function is in the form of a non linear least squares problem as shown in Equation 2.11 where the vector function $\mathbf{f}$ is a mapping from $\mathbf{R}^n$ to $\mathbf{R}^m$ with $m \geq n$. The algorithm is a specialized optimization method to solve the given problem and is more efficient and achieves faster convergence than general methods such as Newton's method. The Gauss-Newton method is the basis for the Levenberg-Marquardt algorithm which is essentially a damped version of the Gauss-Newton method as presented in [48]. The Gauss-Newton method performs a linear approximation of $\mathbf{f}$ near x using the Taylor expansion as the main differentiating factor from the general Newton's method.

$$F(x) = \frac{1}{2}\sum_{i=1}^{m}(\mathbf{f}_i(x))^2 \tag{2.11}$$

2.8 Iterative Closest Point Algorithm

The Iterative Closest Point (ICP) algorithm is a method for the registration of two point clouds. Many variants of the algorithm exist. The base algorithm registers the two point clouds iteratively by finding the closest point correspondences from the reference cloud to the target cloud and then minimizing a non-linear least squares objective function to solve for the rotation and translation that perform the rotation. This process is repeated until an error metric measuring the alignment of the two clouds reaches some tolerance and it can be shown that the ICP algorithm always converges to the nearest local minimum monotonically. The algorithm is fully presented in [49].

2.9 Inverse Perspective Mapping Algorithm

A forward facing view from a camera mounted on the dashboard or front windshield of the vehicle can be used for lane detection. However, such a view is distorted due to the perspective caused by its mounting location. The road lane markings, although parallel, appear to converge at the horizon amongst other perspective effects. Inverse perspective mapping (IPM) is an algorithm used to transform the image to a view point where these distortions are not present by utilizing perspective projection geometry. For the purpose of lane detection for autonomous vehicles, IPM is generally used to project 3D Euclidean space to a 2D planar [50].

2.10 Lane Marking Detection Algorithm

The approach taken to identify the lane markings in a birds eye view image of the road is based on [51]. A recursive Bayesian classifier is used in order to segment pixels belonging to the lane marking from the rest of the road. The classification is performed at the pixel level based on the Bayesian decision theory utilizing probability likelihood models for each classifier class. The likelihood model for the lane marking class is based on the assumption that lane markings are generally vertical, high intensity, pixels with lower intensity pixels surrounding them.

Chapter 3

Platform Design

Although autonomous vehicle platforms share common elements, diversity arises at the hardware and software levels based on differing design decisions, goals, and scopes. Platforms designed for industry tend to be significantly different in computation hardware from research platforms despite using similar sensor stacks. Companies such as Tesla, Nvidia, Intel and NXP have developed dedicated computing systems for autonomous driving aimed for integration in consumer vehicles with small form factors and chip sets often specially designed for machine learning operations. At the other end of the spectrum, research, educational and open-source platforms tend to employ powerful, but standard, computers or laptops often running a generic, well supported, operating system for developmental ease and versatility.

In this chapter, the design is presented for a university research platform. Students will be using the platform to test autonomous driving and Advanced Driver Assistance Systems (ADAS) related algorithms. The overall platform is shown in Figure 3.1. It consists of the hardware platform, software platform, and the interfacing layer which handles their interaction with one another. Section 3.1 presents the software platform design and Section 3.2 presents the hardware architecture design. Interfacing with sensors and actuators is detailed in Section 3.3. The platform design presented in this section is in abstract terms. Chapter 7 will demonstrate the realization of the abstract platform into a functional prototype.

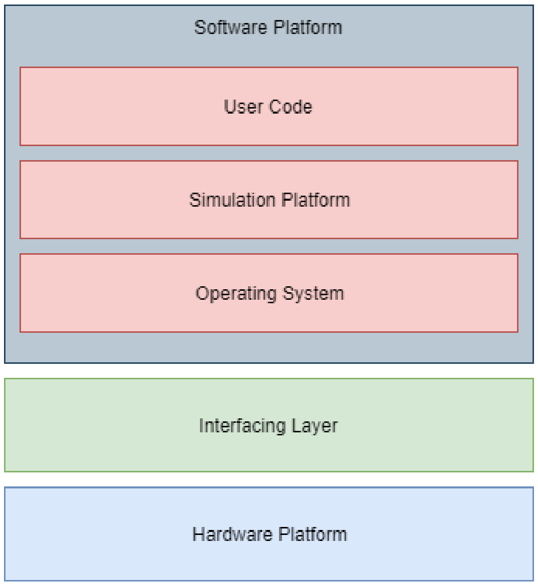


Figure 3.1: Breakdown of Autonomous Vehicle Platform Components

3.1 Software Platform

The design of the software platform covers three elements: the operating system (OS), the simulation platform, and the user code that utilizes the OS and simulation platform to achieve the require goals for the autonomous vehicle. The software platform also interfaces with the hardware platform including the sensor modalities on the vehicle through the interface layer. This is generally handled by the OS alongside a slew of other important tasks.

3.1.1 Operating System

In the context of an autonomous vehicle platform, there are a fair number of requirements from the OS. The primary requirements are the allocation of the resources on the computing system to the various autonomous driving systems and providing the interface for researchers to be able to utilize the various components of the vehicle. Internally, this includes a communication mechanism between the various systems and user code. The OS must additionally be able to interface with external sensors and devices through device drivers unique to the individual device used. Moreover, for any platform utilizing an array of sensors, the OS should handle some tasks related to data management. These tasks include data acquisition and recording as well as data visualization. There are also some requirements from the OS that are not mission critical, but are nice to have. These include libraries and functions for general autonomous vehicle development, the ability to run on arbitrary hardware, that the OS is a real time OS (RTOS) and the ability to interface with common industry tools such as Matlab.

Alongside the functional requirements from the OS, there are some general requirements of the software platform which also pertain to the OS. Firstly, the OS must allow user level code to be modular. For research platforms such as this, various different people typically work on different aspects of the system pertaining to their skills and goals. As such, the software platform and the OS must offer a way for all the different codes to work together by allowing modular user code that can communicate with one another through the OS. Next, since the platform pertains to an autonomous vehicle and safety is a critical issue, the software platform must be safe and secure. Generally these are handled through encapsulation of the various software modules for security and redundancy and monitoring tools for safety. Finally, the last general requirement of the software platform and OS is efficiency in operation. This includes reliable and fast data transfers so as not to hinder the working frame rate of user developed algorithms. The tasks desired from the OS are summarized in Table 3.1 with the required tasks marked.

Moreover, there are some requirements from a logistics point of view. First, the OS should support quick implementation on the platform that should be achievable by one person. Next, there must be an abundance of resources and support available. Lastly, the platform should be tested and used elsewhere for autonomous vehicle platforms thus proving its worth in the automotive field. Some nice to have features are that the OS is open source thus saving costs on the developed platform and that the OS supports distributed systems in case the autonomous vehicle has multiple computing systems.

3.1.2 Simulation Platform

A simulation platform is necessary for safely testing the algorithms used on the developed platform. Testing algorithms in a real world vehicle can lead to hazardous and unsafe situations if something goes wrong. Moreover, simulated systems can cover a lot more scenarios than it may be possible to test with a real system saving time and improving safety. For an autonomous vehicle platform the simulation platform requirements are as follows. The simulation software must be able to simulate not only the vehicle including the actuation systems, but also all the autonomous driving sensors. This will ensure full coverage over the range of autonomous driving algorithms. Another requirement of the

Table 3.1: Platform Operating System Tasks

Task Description	Required
Resource Allocation	✓
Communication Between Systems and User Code	✓
Interfacing with Sensors	✓
Data Acquisition	✓
Data Recording	✓
Data Visualization	✓
Development Libraries and Tools	
Ability to Run on Arbitrary Hardware	
Interface with Common Industry Tools	
Allow Modular User Code	✓
Encapsulation of Communication	✓
System Monitoring Tools	✓
Efficiency in Operation	
Real time	
Facilitates Fast Implementation	✓
Available Resources	✓
Tested in other Autonomous Vehicles	✓
Open Source	
Support Distributive Systems	

simulation software is the ability to perform hardware in the loop (HIL) testing using actual vehicle systems. This method of testing allows vehicle systems to be tested in simulated environments and improves the degree of confidence in the real platform prior to real world testing. Moreover, the simulation platform should allow for all aspects of real world driving to be simulated, including pedestrian, bicycle, and vehicle traffic, infrastructure including buildings, traffic signs, and light posts as well as all weather conditions. These ensure that the tested algorithms are robust and reliable.

There are several nice to have features of a simulation platform. First, it is nice if the platform can be coded in the same programming language and using the same tools and libraries as the actual code. As a result of this, the exact working code in simulation can be utilized in the real vehicle and development time can be saved. Another nice feature of the simulation platform is if realistic data can be generated by the simulation software. These include photo realistic pictures in the case of vision sensors, but for other sensors include the data injected with realistic noise to properly simulate real world sensors. Not only does this enable accurate testing, but photo realistic pictures can be used in the training of machine learning algorithms to be deployed in the real world.

3.1.3 User Code

User code is referred to code written on top of the OS which performs functions such as localization, object detection, path planning, etc. Due to the potentially large amount of researchers working on the developed platform, the user code is required to be modular. Although the OS covers the technical aspects and communications between modules, it is up to the developers to properly separate modules and adhere to good coding practices.

3.2 Hardware Platform

The hardware platform consists of the autonomous vehicle itself, the sensing devices, the actuating devices and the computing hardware. The key goal for the hardware platform is simplicity, with a focus on achieving quick implementation and operation. The reason for this is that the platform is currently an experimental prototype and will constantly be improved as needed. While things like extensibility and robustness are kept in mind, it is anticipated that the computing hardware and sensors used will change in the future making it too early to lock down the hardware platform design. Instead, quick implementation of a simple and operational platform will allow for faster testing of software algorithms and the capabilities of the platform which in turn will lead to the knowledge required for a robust and extensible future design.

In developing an autonomous vehicle platform, there are two possible approaches available for the base platform. The first approach is to purchase an existing autonomous vehicle base platform. The existing base consists of the vehicle, a drive by wire system allowing the vehicle to be actuated from software, sensors outfitted for various levels of autonomous driving, and a computing system already interfaced with the hardware and setup for quick development on the platform. The main benefit of this approach is the readily available platform that allows for immediate development and testing of algorithms. The drawbacks include higher costs, including both the purchase cost as well as regular maintenance fees that the companies providing such platforms require, and limitations in accessing the systems on the vehicle that are not exposed for use. Moreover, manufacturers, for security concerns, do not provide details of the sensors and control systems on the vehicle which hinder the use of existing autonomous vehicle platforms. The alternative approach is to purchase an ordinary car and outfit it with the required sensors, drive by wire and computing systems to make it autonomous. This solution is not only cheaper than the previous one, but also allows for full freedom in customization of the autonomous vehicle platform. Furthermore, being an educational institute developing a research platform, car manufacturers may be willing to provide access to vehicle controls and sensors information under a strict confidentiality agreement.

A requirement for the hardware platform is to have an electronic control unit (ECU) in addition to the computing system. The main reason for this is to have a separate computing system for low level safety critical systems that is not available for access to developers working on the main computing system. Additionally the ECU can be used as a filter for actuator commands to ensure the vehicle is not being commanded to perform a dangerous maneuver. This is achieved by having all the vehicle commands go through the ECU with algorithms on the ECU ensuring command validity. The main computing system is then responsible for high level autonomous driving algorithms and providing the control commands to the ECU.

The requirements for the actuation system on the vehicle is to allow control of vehicle steering wheel angle, and vehicle driving speed through software driven commands. The actuation system can be expanded to include control of other vehicle features later on, but these are critical to creating a usable autonomous driving platform. Moreover, the actuation systems have to adhere to a performance constraint for their response time to the commands. Ensuring that the vehicle responds to the commands quickly is critical to the success of the high level algorithms as delays can severely hinder the performance of the algorithms. Moreover, the actuators are also constrained on the rate of change of the commands. For example, supplying a steering angle command that is drastically different from the previous command should not induce a dangerously fast or slow rate of change to get to the commanded state.

Since the developed platform is a research platform, the requirements for the sensors used on the platform are only to provide full coverage of the surrounding environment. There is no limitation on the amount of sensors or redundancy in the sensors as different researchers will use different types and amounts sensors to test various different algorithms. Thus, one of the goals for sensor selection is to have some representation for all of the popular autonomous driving sensors. That being said, cost is a limiting factor to the upper bound of amount and type of sensor used on the developed platform.

3.3 Interfacing

Sensor data needs to be available for the computing system to be used by the software algorithms implemented on the autonomous vehicle. Moreover, the software algorithms need to be able to send commands to the vehicle actuators in order for the vehicle to perform tasks. Both of these require interfacing between the hardware and software platforms components.

For the developed platform, the interface consists of the physical connections between the hardware and software components as well as the software code necessary to receive/send and interpret the data coming from or going to the hardware. Since the interface is a crucial part of the system, there are some key requirements imposed on the interface to ensure reliability and extensibility.

Firstly, the interfaces are required to be designed in a way that allows for new sensors and actuators to be added easily. For the hardware interface, this means that the hardware connections support new connections to be made with the available ports on the computing system. For the software interface, this means that sensors share a common interface framework that can support the addition of new sensors and actuators with minimal effort. Moreover, since sensors and actuators use or provide different formats and types of data, the software interface must be able to support the various formats needed.

Next, the interfaces should allow for bi-directional communication when necessary as some sensors and actuators, in addition to providing data to the computing system, require commands to be sent to them specifying configuration options and parameters.

Moreover, the interfaces need to support the desired bandwidth and frame rates for operation. Sensors such as LIDARs and cameras can provide large amounts of data while other sensors such as radars provide a relatively lower amount per frame. This coupled with the frame rate of data transfer impose constraints on the physical and software interface to be able to support these data transfers reliably.

Finally, in an autonomous vehicle platform, interfacing is also required for the timing and synchronization signals between devices in order to ensure they operate at the correct times and that the timing between all the devices is based on the same source. This requires additional interfacing to allow for the transfer of signals necessary to accomplish this task.

Chapter 4

Simulation Setup

A simulation platform is necessary for safely and efficiently testing the algorithms used on the developed platform. Testing algorithms in a real world autonomous vehicle can lead to hazardous and unsafe situations if something goes wrong. In simulation, the sensors can be represented ideally without any noise or deviations in measurement. This is useful in checking for correct implementation of algorithms as the ideal results can be expected. Moreover, varying levels of noise can be injected into the measurements to test the failure point of algorithms, thus helping to characterise whether the algorithm will work with the real world setup.

In the automotive domain, there are a few industrial level simulation platforms such as CarSim, Panosim, and Prescan. CarSim finds a lot of use in the vehicle controls domain and is widely used in MVSL. However, CarSim only simulates some of the autonomous vehicle sensors with ideal models. Panosim and Prescan are more geared towards autonomous vehicles with realistic sensor models alongside ideal ones. Prescan also couples with Unreal Engine to provide photo realistic sensor data that can be used for training of learning algorithms. Among open source software, multiple applications based on Unreal and Unity gaming engines are available include Microsoft Airsim and CARLA Simulator. However, these tend to be geared towards machine learning algorithms and lack the completeness of features available from industry software such as Prescan. These include HIL capability and the ability to test real time implementations.

For the purpose of testing the algorithms developed for the autonomous vehicle platform, Prescan is chosen as the simulation platform. The main reasons for this choice is that it provides a complete set of features for autonomous driving simulation and is easy to use. Prescan allows users to simulate vehicles, pedestrians, infrastructure, driving scenarios, and autonomous driving sensors. It interfaces with Matlab so that the same algorithms tested in a simulated Prescan environment can be applied to the real platform. Figure 4.1 displays a small city scenario created in Prescan and Figure 4.2 shows an intersection of the city with traffic signals, pedestrians, vehicles and other infrastructure. An ego vehicle can be made to traverse this scenario utilizing on board and infrastructure mounted sensors using developed algorithms. The chosen simulation platform provides an end to end solution for autonomous vehicle simulation.

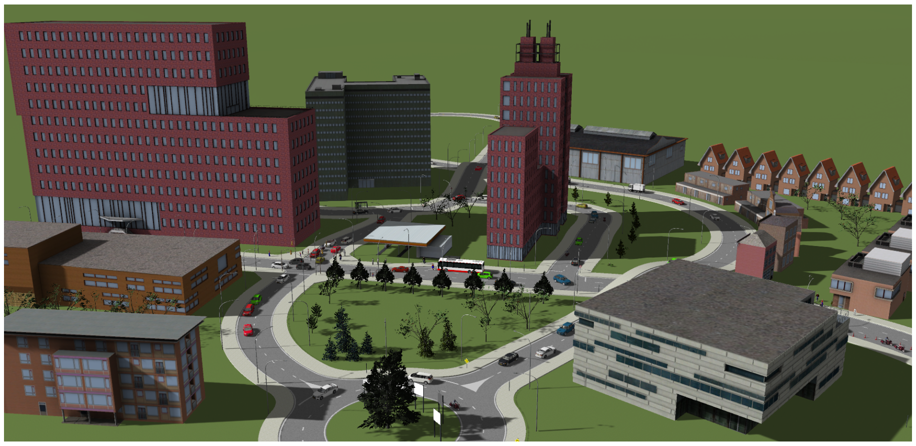


Figure 4.1: Example of City Environment Created in Prescan

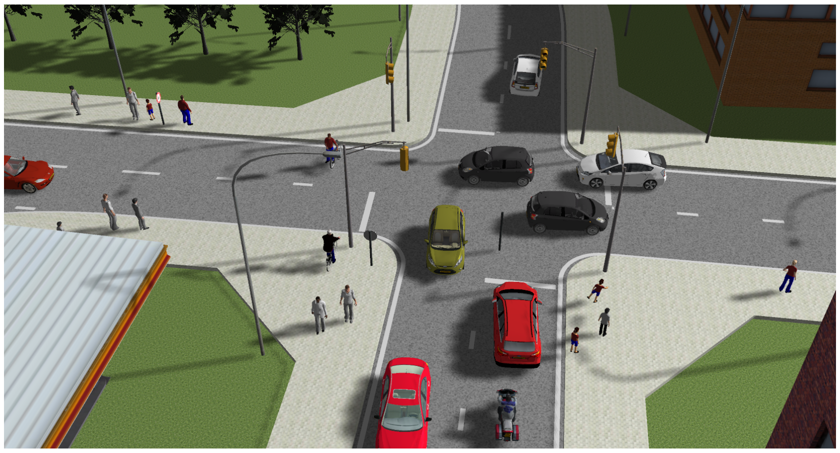


Figure 4.2: Example of City Intersection Created in Prescan

Chapter 5

Sensor Calibration

Sensors are mounted on the vehicle so that their field of views cover important sections of the surrounding road and environment. It is difficult to accurately measure the 3D rigid body transformations between all of the sensor coordinate frames using hand held tools such as measuring tapes, protractors, plumb bobs, lasers, etc. In practice, different calibration techniques are used in order to determine the transformations.

Since the sensors will not be commonly adjusted in their mounting positions and poses, offline calibration techniques are implemented for the platform as they are generally easier to implement. This is because specialized calibration targets can be designed and utilized, whereas online techniques seek to calibrate the sensors based on the information present in a typical driving environment which is generally harder to obtain and decipher. Instead, calibration routines are designed to be simple and easy to perform so that the vehicle may be calibrated quickly when needed.

In this chapter, three different calibration techniques are presented, each pertaining to different sensor types or configurations expected to be on the autonomous driving platform. In Section 5.1 a joint calibration technique is presented which allows for the simultaneous calibration of all sensors that return the 3D position of detections and share a common field of view with each other. In Section 5.2, a similar technique is used to calibrate infrastructure mounted sensors that return the 3D position of detections to a GPS sensor. This routine allows the sensors to be localized in a global coordinate frame positioned on the earth and is useful in systems involving vehicle to infrastructure interfacing. Finally, Section 5.3 presents the Hand-Eye calibration technique commonly used for calibration of sensors mounted on the end effectors of robotic arms to the arm base. However, this technique will be used on the platform to calibrate the inertial GPS system with sensors that can generate odometry information from their data. Some of the calibration techniques are tested using the chosen simulation platform prior to implementation on the real platform.

5.1 Heterogeneous 3D Joint Calibration

This calibration technique is based on the method provided in [21]. It is used to jointly calibrate sensors or groups of sensors that can measure the 3D position of a common target and have a shared field of view. For example, using the proposed method, a stereo camera system may be calibrated with a 3D LIDAR sensor. The result of the calibration will be the 3D rigid body transformation that should be applied to one of the sensor or sensor groups in order to have its detections represented in the other's coordinate system. The calibration is performed as follows. First, a calibration target is designed that is capable of being detected well by each sensor, as detailed in subsection 5.1.1. Then, the target is placed at different positions in the overlapping field of view and measurements of the position of the target are taken by each sensor. Alternatively, the target could be fixed in place and the vehicle moved. It is simplest to take measurements when both the vehicle and target are stationary, so that error is not introduced by motion occurring between the capture times of different sensors. Finally, an objective function is minimized to solve for all the homogeneous transformation matrices required.

The working principle of the optimization is that the transformed point (3D detection) from one sensor's coordinate frame to another sensor's frame should result in the corresponding points to be perfectly overlapped if the homogeneous transformation matrix has no error. That is to say the that distance between a transformed detection and the corresponding detection by another sensor should be zero. Moreover, the rotation matrices must be orthogonal by definition. With these, the objective function shown in Equation 5.1 can be written for a pair of sensors, A and B, where n is the number of points detected and ν is sufficiently large to enforce the orthogonal constraint. Note that the objective function can be expanded to include additional sensors so that the number homogeneous transformation solved for is equal to the number of unique sensor pairs. For example, adding another sensor, C, will result in 3 rotation matrices and 3 translation vectors that need to be solved for: $R_A^B, R_B^C, R_C^A, t_A^B, t_B^C, t_C^A$. The benefit of this method is that any number of sensors can be simultaneously calibrated as long as they fulfill the requirements.

$$
\begin{aligned}
min\frac{1}{2}\Sigma_i^n \parallel R_A^B \cdot p_{A_i} + t_A^B - p_{B_i} \parallel_2^2 \\
+\nu \parallel (R_A^B)^T \cdot R_A^B - I \parallel_2^2
\end{aligned}
\tag{5.1}
$$

The calibration may be improved, as mentioned in [21], if additional constraints such as distance preservation between targets or coplanar targets are used. These generally involve having multiple targets being detected in each frame and require more time and effort to perform the calibration. Furthermore, if the accuracy of calibrated sensors is sufficiently high, these constraints may not be necessary to achieve a satisfactory calibration result. Thus, it makes sense to hold off from utilizing these constraints unless proven necessary.

Since the objective function is in the form of a non-linear least squares problem, a trust region method such as the Levenberg-Marquardt (LM) method outlined in Section 2.7 may be used to find the local minimum of the objective function. It is possible that the solution found is a poor local minimum. To combat this, the authors in [21] restart the minimization after adding a random perturbation to the local minimum found in the previous iteration. Although it is not mentioned how many iterations are performed, it is sufficient to say that a good local minimum is found when the resulting minimum does not change significantly between iterations.

A simple way to estimate the calibration error is to transform a new set of measurements from each sensor into one frame such as the LIDAR sensor coordinate frame using the homogeneous transformation matrices constructed from the solved rotations and translations. Next, the distance between the corresponding measurement of each sensor with the LIDAR sensor measurement can be calculated using the euclidean distance. Finally, the average distance across a number of measurements can provide a metric for the error present in the calibration since ideally the average distance should be zero if the calibration is perfect. With this error metric, a threshold may be defined that is considered to be a good calibration. This threshold will typically be application specific based on the accuracy required from the calibrated system.

5.1.1 Calibration Target Design

The calibration target should be designed such that all sensors can detect the same point on the target. An example will be given for the joint calibration of a camera, radar, and 3D LIDAR system. Generally, for machine vision cameras, it is enough to use a standard calibration chessboard and have the target be one of the corners of the chessboard. For radar sensors, a radar reflector of appropriate size and material should be used. The reflector ensures a strong and consistent detection of itself by the radar sensors. A 3D LIDAR sensor will generally not need a specialized target when the point cloud is dense or the target is sufficiently close to the sensor. This is because in these scenarios, the target point may be visible in the point cloud and could simply be selected manually or automatically through an algorithm. If this is not the case, the ability of the 3D LIDAR to measure the intensity of each individual return may be utilized and an appropriate target can be designed. Therefore, the final target will consist of a calibration chessboard with a radar reflector attached behind the board rigidly mounted on a mobile platform so that the position of the target may be changed easily. An anticipated source of error for this calibration technique is that the target point detected by the individual sensors may not be the same point due to physical constraints and will be addressed if found significant.

5.2 Infrastructure Sensor and GPS Calibration

One of the features of autonomous vehicles is the ability to interface with infrastructure mounted sensors to acquire additional information about the environment. Knowing the location of the infrastructure sensor is useful in localizing the information attained. A simple and effective way to do this is to know the GPS location, latitude and longitude, of all relevant infrastructure sensors so that the received data can be localized with respect to the GPS system on the developed platform.

Since the coordinate systems for both the infrastructure sensor and the GPS sensor mounted on the vehicle are stationary, the motion of the vehicle does not affect the transformation between them. Therefore, Equation 5.1 can be used once again in the exact same way. Moreover, a similar approach to the calibration target design is employed. The target, detected by the infrastructure sensor, should be mounted at the location of the GPS reference point so that both sensors are measuring the position of the same point.

The calibration procedure is to drive the vehicle in a path that excites all the translation and rotation degrees of freedom and take measurements from both sensors at the exact same time. The simplest way to do this is to stop the vehicle when taking the measurement, but time synchronization and triggering of the sensors may also be employed. If the road surface does not allow the vehicle to excite all the degrees of freedom, an alternative is to mount the targets on a mechanical platform that can independently incur these motions on the targets.

5.2.1 Vision Sensor and GPS Calibration Simulation Results

The setup for the calibration between an infrastructure mounted vision sensor and a GPS is shown in Figure 5.1. A calibration chessboard is mounted on the roof of the vehicle and one of the corners of the chessboard is overlapped with the GPS reference point. This is the point on the vehicle that the GPS device measures the latitude and longitude of. Any infrastructure sensor that measures the 3D position of a target may be used, but the example uses a camera since this is the most common. The chessboard is mounted on a mechanical platform that provides sinusoidal oscillations along the roll and pitch axis of the chessboard as well as translation in the vertical direction. These, coupled with the planar translations and yaw motion of the vehicle itself, ensure that all the degrees of freedom are excited as measurements are taken. When all the degrees of freedom are excited, 3D homogeneous transformation matrix can be fully resolved. The yellow trajectory shows the path that the vehicle will take. The camera is mounted on the light post and its field of view is shown on the road.

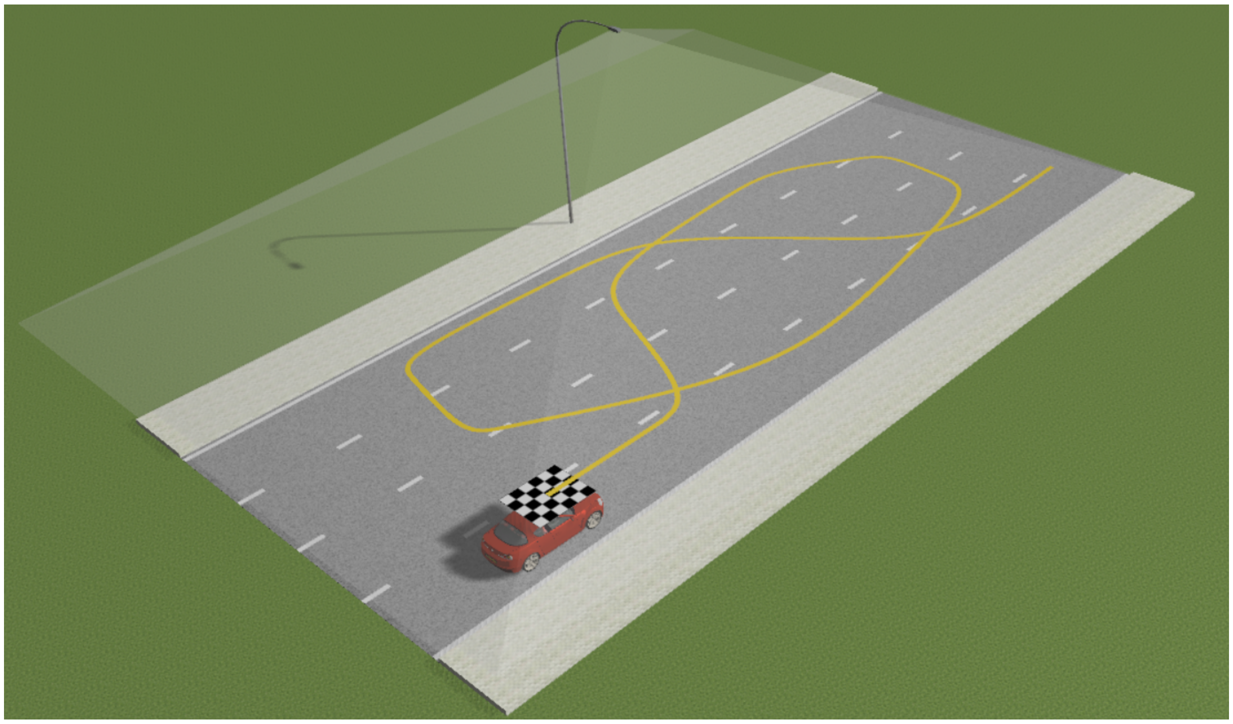


Figure 5.1: Simulation Scenario Setup for Infrastructure Sensor with GPS Calibration

The mounting location of the camera with respect to the global GPS coordinate frame is fixed and known. The simulation platform simulates the vehicle driving the trajectory as well as the chessboard sinusoidal motions. The camera takes an image at a fixed frame rate. At the exact same time, the latitude and longitude of the GPS system is measured. As a result, a number of images and coordinates are obtained as the vehicle traverses the trajectory. Roughly 15 to 30 well spread data points are needed for a good calibration based on anecdotal experience. Finally, the cost function shown in Equation 5.1 can be minimized. A value of 100000 is used for ν and works well. The calculated transformation matrix for the transformation from the GPS coordinate frame to the camera coordinate frame is:

$$\begin{bmatrix} -0.9987 & 0.0495 & 0.0100 & 64.3602 \\ 0.0496 & 0.9987 & 0.0070 & -59.4982 \\ -0.0096 & 0.0074 & -0.9999 & 8.9662 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

To measure the calibration error, the camera location with respect to the global GPS coordinate frame can be obtained using the solved transformation by transforming the point $[0, 0, 0]$ to the GPS coordinate frame. Then, the resulting coordinates can be subtracted from the known setup coordinates to find the error along each of the x, y, and z axis. Finally, the mean error can be calculated from these three axial errors. The axial errors are 0.1361 m, 0.3316 m, and 0.0267 m along the x, y, and z axis, respectively. The average of these errors is 0.1648 m.

Since, in the simulation environment the GPS measurement is completely error free, the main cause of error is the camera measurements. Since the calibration chessboard is far from the mounting location of the camera, the chessboard corners cannot be determined to a very high level of accuracy. That is to say, that the error can be reduced as the size of the chessboard is increased or the camera is mounted closer to the vehicle. That being said, the mounting height of the camera is strategic for other reasons and cannot be changed just for a better calibration. Moreover, in a real world GPS system, there is error present in the GPS measurement and so the calibration will be worse than it is in the simulation. Furthermore, it is difficult in the real world to mount a target at the exact GPS reference point. Instead, the target will be mounted at some arbitrary and easy to mount location, but the transformation will be measured. Then the transformed point may be calibrated to obtain the same result, however the measurement error in the transformation will also affect the accuracy of the results.

5.3 Hand-Eye Calibration

In the previously described calibration cases, all sensors are able to collect measurements of the 3D position of a calibration target in a common field of view allowing for the method employed to be successful. When these requirements are not met, specialized calibration techniques need to be considered on a case by case basis for each of the uncalibrated sensors. In the case of calibrating the GPS sensor with other vehicle mounted sensors such as the LIDAR or the camera, it is not possible to meet these requirements. The only way that a GPS system will measure the location of a calibration target is if the target is mounted at the reference location of the GPS, but since the reference location is static with respect to vehicle mounted sensors, it is not possible to obtain multiple measurements of the target by the vehicle mounted sensors. Therefore, the previously used technique becomes invalid.

Fortunately, it is possible to solve this calibration scenario through a method referred to as Hand-Eye calibration. At the core, this method utilizes separate calibration targets for each of the two sensors being calibrated. Although it may be possible to perform the Hand-Eye calibration with more than two sensors at a time, it is not necessary for the platform since the transformations of the remaining vehicle mounted sensors with respect to the GPS system can be calculated algebraically from the previous calibrations. Moreover, the joint calibration method only required the position of the calibration target to be detected by the sensors, but the hand-eye calibration requires the 3D orientation to be detected as well. Therefore, a GPS + IMU navigation solution should be utilized.

5.3.1 Camera and Navigation Solution

For calibration between a camera and the navigation solution, a chessboard can be used as the target detected by the camera. The chessboard is statically mounted somewhere in the environment and a coordinate frame is define based on one of the corners of the chessboard. Next, a static navigation reference frame can be defined for the navigation solution when the latitude, and longitude measurements are converted to UTM coordinates as per subsection 2.4.2. Finally, as depicted in Figure 5.2, the vehicle should be driven to different poses such that all the degrees of motion are excited and the chessboard is in the field of view of the camera during all poses. Hand-Eye calibration seeks to solve the homogeneous transformation equation of the form $AX = XB$, or some similar variant, where A, B, X are all homogeneous transformation matrices. $A_0, A_1, ...$ are obtained from the camera images of the chessboard and the intrinsic camera matrix as discussed in subsection 2.4.1. A series of A matrices can be calculated for two successive poses. Similarly, $B0, B1, ...$ are the transformation matrices from the navigation reference frame origin to each of the poses of the navigation reference point, and can be used to calculate a series of B matrices. X is the homogeneous transformation matrix from the navigation reference point on the vehicle to the camera coordinate frame, the entity of interest.

In order to solve the homogeneous transformation equation $AX = XB$, the method outline in Section 2.6 is used. Other closed form or optimization based solutions also exist and can be used. In [25], a solution is presented to the homogeneous transformation equation $CX = ZD$, which simultaneously solves for X and Z. Z is the transformation from the navigation reference frame to the camera target frame, C is A_0^{-1}, and D is B_0^{-1}.

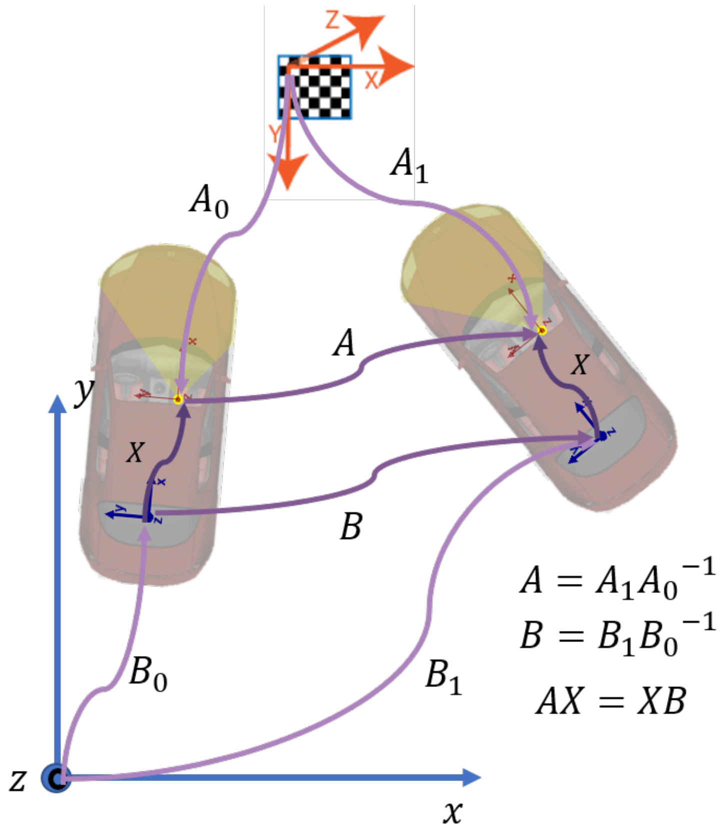
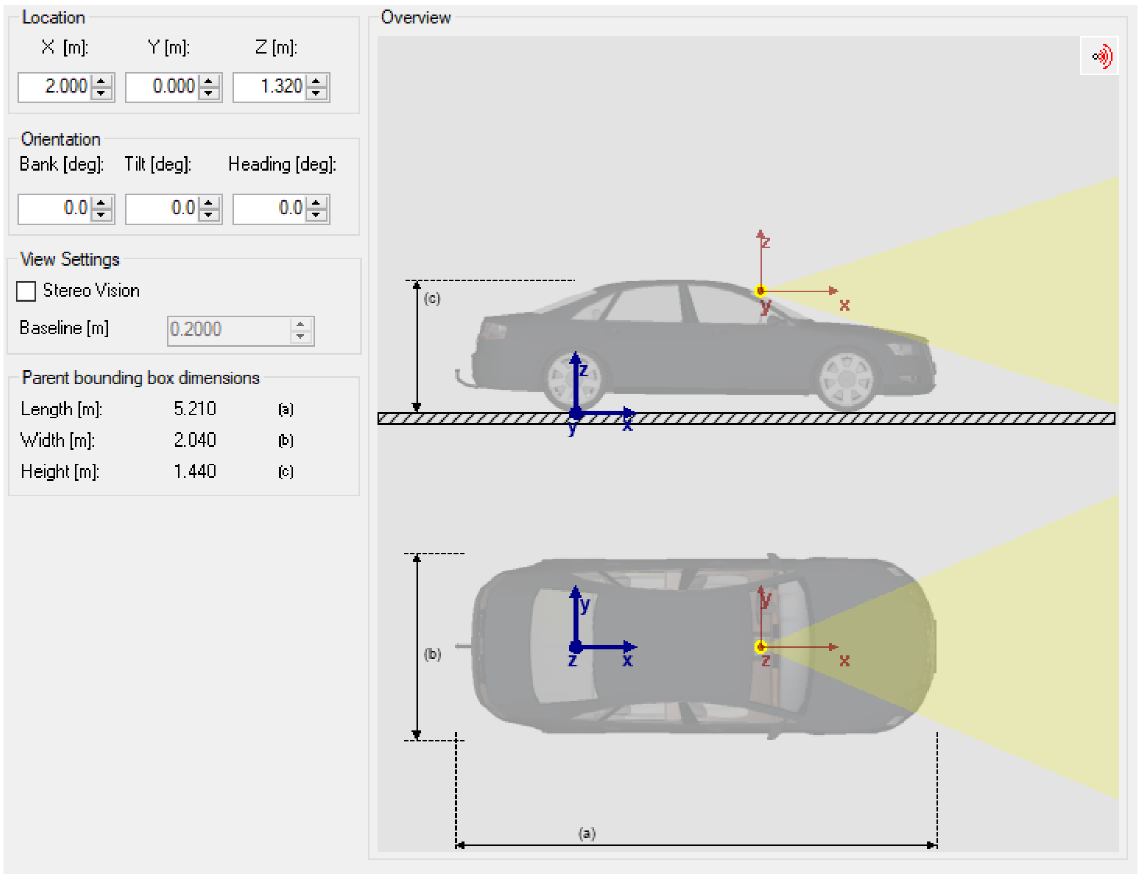


Figure 5.2: Basic Principle of Hand-Eye Calibration

Camera and Navigation Solution Calibration Simulation Results

In order to calculate the transformation from the navigation system reference point to the vehicle mounted camera, the Hand-Eye calibration technique is used. The method is tested in simulation with the setup as follows. A camera is mounted on the vehicle as shown in Figure 5.3. The blue coordinate system in the image is the navigation system reference point and the camera is mounted with respect to that point 2 m in the x direction and 1.32 meters in the z direction. A stationary calibration chessboard is setup in front of the vehicle and the vehicle is made to drive a trajectory that excites all the degrees of freedom but also keeps the chessboard in the camera field of view. Figure 5.4 shows some of the camera images obtained. A series of camera images and navigation solution samples are obtained and the method explained previously is utilized. The resulting transformation matrix obtained is:

$$\begin{bmatrix} -0.0003 & -0.0000 & 1.0000 & 2.0401 \\ -1.0000 & -0.0011 & -0.0003 & 0.1138 \\ 0.0011 & -1.0000 & -0.0000 & 1.3167 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$


Location

X [m]:	Y [m]:	Z [m]:
2.000	0.000	1.320

Orientation

Bank [deg]:	Tilt [deg]:	Heading [deg]:
0.0	0.0	0.0

View Settings

☐ Stereo Vision

Baseline [m] 0.2000

Parent bounding box dimensions

Length [m]:	5.210	(a)
Width [m]:	2.040	(b)
Height [m]:	1.440	(c)

Figure 5.3: Camera Setup WRT Navigation System Reference Point

It can be seen that the translation component of the transformation from the navigation system to the camera has an error of 0.0401 m, 0.1138 m, and 0.0033 m in the x, y,

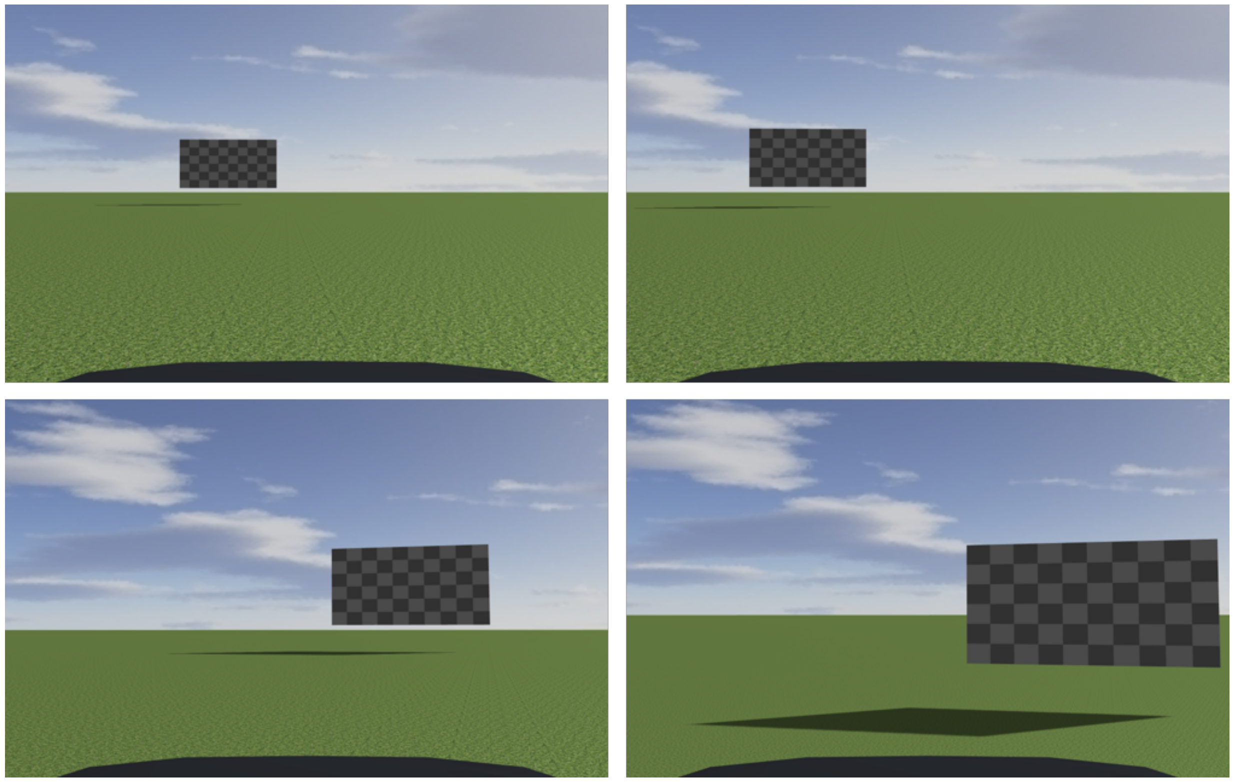


Figure 5.4: Camera Images at Various Stages in Vehicle Trajectory

and z directions, respectively. The rotation matrix of the homogeneous transformation pertains to euler angles of $-90.0148°$, $-0.0654°$, and $-90.0014°$ along the z, y, and x axis, respectively. Although the true value of the rotation matrix should be identity since there is no rotations between the two coordinate systems, this result occurs because of a coordinate frame difference in the Prescan simulation environment and that of the Matlab environment where the calibration algorithm is run. The difference of the coordinate systems is a $-90°$ rotation around the z axis followed by a $-90°$ around the x axis. Subtracting these from the values obtained after calibration results in absolute value rotational errors of $0.0148°$ around z, $0.0654°$ around y, and $0.0014°$ around x.

These results demonstrate the working principle of the Hand-Eye calibration algorithm and setup. Errors in simulation may be caused once again by the camera not being able to accurately detect the location of the chessboard corners. In simulation, a very large chessboard is used mitigate these errors.

5.3.2 LIDAR and Navigation Solution

The calibration process between a LIDAR and the navigation solution is the same, however, it is not as simple to obtain the series of A and B matrices. Although a common target point can be easily selected from a series of point clouds and the translation vector resolved, it is not so easy to determine the 3D orientation of the target point. A common way to determine the homogeneous transformation between two point clouds captured from different poses is to use the ICP algorithm outlined in Section 2.8. The algorithm determines the transformation by finding the point to point correspondences between the two point clouds and then solving for the best rotation and translation to align all of the points.

It is difficult to obtain a completely correct transformation from the ICP algorithm. The LIDAR itself has error in measurement of each individual point, and further noise is injected if the environment is not completely static between all the captured poses. Furthermore, the ICP algorithm may converge to a local minimum and not the global minimum. This coupled with errors in the navigation solution make this calibration very difficult to perform well.

Chapter 6

Lane Keeping System

In this chapter, the implementation of the lane keeping system is explained and tested using the simulation platform. Lane keeping generally consists of lane detection, in which the road lane markings and boundaries are detected, and lane following, in which the vehicle is controlled to stay within the lane markings and boundaries while driving. The lane keeping system will be implemented on the developed platform as a test for the platform and all on board systems.

With a highly accurate GPS and IMU system such as an RTK or DGPS system fused with an accurate IMU, it is possible to follow a prior path so that the vehicle can drive autonomously while keeping itself in a lane. However, the real world is dynamic and it is not always possible to follow a pre defined path. Moreover, these highly accurate navigation systems are expensive and not suitable for implementation in an affordable solution. Therefore, other sensors, such as a monocular camera, are often used for lane detection. In this work, a monocular camera is used alongside a navigation system for lane keeping. The navigation system serves as a fall back solution when situations arise where the lane cannot be detected, yet augmenting the navigation system with a vision system allows for the accuracy requirements from the GPS and IMU to be significantly reduced, to the extent that cheaper less accurate systems may be used in conjunction with the monocular camera.

A simple, but effective lane keeping controller is described in Section 6.1. The use of a GPS and IMU based navigation solution is employed with the controller in Section 6.2. Monocular lane detection is employed to accurately locate the lane markings with respect to the vehicle in Section 6.3. In Section 6.4, the vision system is augmented with the navigation solution for lane keeping when there is no lane present or when the vision system fails. Finally, the various approaches are tested on the simulation platform in Section 6.5 prior to real world implementation.

6.1 Lane Keeping Controller

The lane keeping controller is largely based on the controller used in [3] by the DARPA Grand Challenge winning robot: Stanley. The control law, shown in Equation 6.1, uses the lateral error, $l(t)$, and the heading error, $h(t)$, as the error metrics trying to be brought to zero. The desired steering angle, $\delta(t)$, is linearly proportional to the heading error with gain, kp_1.

Moreover, the desired steering angle is non-linearly proportional to the lateral error with gain, kp_2, and the vehicle speed, $u(t)$. When the lateral error is large or the vehicle speed is slow, the desired steering response is stronger. On the other hand, if the lateral error is small or the vehicle speed is fast, the desired steering response is weaker. These help to strike a balance between controller stability and speed of convergence. The purpose of the offset, $kp_3 \geq 1$, is to nullify large steering requests when the vehicle is near zero velocity.

As discussed in [3], the convergence of the controller is only limited by the velocity of the vehicle. Although this is a simple proportional controller, it is effective at the slow velocities that the autonomous vehicle is planned to be driven at.

$$\delta(t) = kp_1 \cdot h(t) + \arctan(\frac{kp_2 \cdot l(t)}{kp_3 + u(t)}) \tag{6.1}$$

6.2 Lane Keeping with Navigation System

The underlying prior for lane keeping based solely on GPS and IMU is an accurate map for the vehicle to follow. If such a map is available, the vehicle can be localized on the map in real time with data from an accurate on-board GPS. The error metrics can be determined based on this localization in conjunction with vehicle heading angle obtained from an IMU.

In this thesis, the map is a series of points, defined in a global coordinate system, that form a path. The origin of the coordinate system can be located at an arbitrarily chosen point in the world and should be kept consistent throughout all tests. It is desirable that the map is collected at a high frequency so that the distance between successive points is small. Larger separations between points on the map will result in less accurate error metric calculations that will then require interpolation techniques to be used. In the following subsections, interpolation techniques are not applied as the map is assumed to be collected at a high frequency.

Furthermore, it may be necessary to smooth the map if points have significant noise. In this thesis, it is assumed that measurement noise is at an insignificant scale and the map is accurate to $\leq$ 10 cm RMS error per point position. This is consistent with the typical accuracy of advanced RTK GPS systems and post processed DGPS systems.

6.2.1 Lateral Error Metric

Lateral error is defined as the distance from the vehicle position, $p_v = (p_{vx}, p_{vy})$, to the closest point, $p_c = (p_{cx}, p_{cy})$, on the map. In order to localize the vehicle position on the map, the latitude and longitude is converted to the coordinate system of the map using the process explained in Section 2.4.2. A method utilizing the Quickhull algorithm for convex hulls, based on the work of Barber [52], is employed to efficiently locate p_c. Finally, the euclidean distance can be calculated between the vehicle position and the closest point and is depicted in Figure 6.1.

6.2.2 Heading Error Metric

Human drivers tend to steer the vehicle based on a reference point some distance ahead of the vehicle. On straighter paths, minor adjustments are made and the vehicle is simply guided in the correct direction to keep it adequately centered in the lane. On highly curved paths, the reference point is brought a lot closer to the vehicle so as not to cut corners. To achieve a similar driving characteristic, a dynamic reference point, $p_r = (p_{rx}, p_{ry})$, is calculated. The heading error metric is calculated such that the control effort seeks to drive the vehicle into the reference point, and is depicted in Figure 6.1. With this approach, the lateral error metric becomes less significant in the control of the steering.

The dynamic reference point is calculated by finding the closest point on the map that is a dynamic distance away from the closest point, p_c, as calculated in the previous section. This dynamic distance is a function of the current curvature, k, of the road and is calculated as shown in Equation 6.2. The map is pre-processed and the curvature at each point is calculated as explained in Section 6.2.3. The dynamic distance is bounded by a lower and upper bound, d_{min}, d_{max}, which are determined through trial and error. These bounds ensure that the reference point does not cause a loss of stability by being too close to the vehicle, or cause the vehicle to cut corners by being too far from the vehicle.

$$d = \begin{cases} d_{min} & |k| \geq k_{max} \\ d_{min} + (k_{max} - |k|) \cdot (\frac{d_{max} - d_{min}}{k_{max}}) & |k| < k_{max} \end{cases} \tag{6.2}$$

Finally, the heading error metric is calculated as shown in Equation 6.4. θ_{nav} is the vehicle heading angle measurement coming from the navigation solution and θ_d is the desired heading angle for the vehicle.

$$\theta_d = \arctan \frac{(p_{ry} - p_{vy})}{(p_{rx} - p_{vx})} \tag{6.3}$$

$$h = \theta_d - \theta_{nav} \tag{6.4}$$

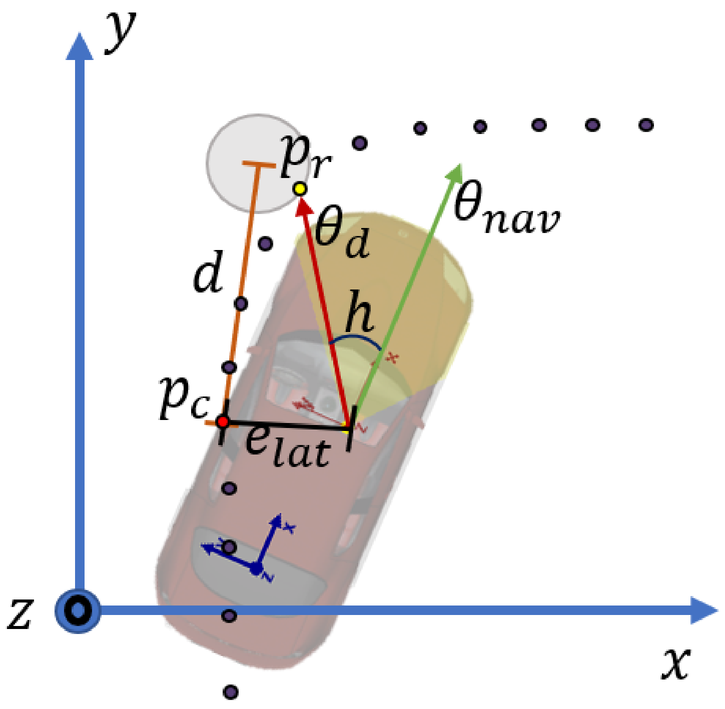


Figure 6.1: Visual Depiction of Navigation System Error Metrics

6.2.3 Map Pre-Processing

This pre-processing is performed in order to compute curvatures and distances of points on the map in order to optimize the live processing. Distances between adjacent points are computed using the Euclidean distance and is utilized in order to find the closest point at a specific distance from a reference point.

Curvatures are computed based on a sliding window section of the map. Points inside the window are fitted with a parabolic lane model using a Random Sample Consensus (RANSAC) based curve fitting. Window size is determined by trial and error to be sufficiently large for a good fit, yet not too large so that there is more than one inflection point in the window. If there are too many changes in curvature within the window, a parabola will not fit the data well and the curvature will not be accurate. From the parabolic fit, the curvature can be calculated using the curvature of a parabola equation as shown in Equation 6.5 with $x = 0$. The variables a and b are the parabolic coefficients from the equation $y = a \cdot x^2 + b \cdot x + c$. In this way, a curvature value can be computed for each point on the map and can be quickly looked up after localizing the vehicle on the map.

$$k = \frac{2 \cdot a}{(1 + (2 \cdot a \cdot x + b)^2)^{\frac{3}{2}}} \tag{6.5}$$

6.3 Vision Based Lane Detection

The final goal of the lane keeping algorithm is to autonomously drive the developed platform around the University of Waterloo Ring Road which is a loop of bidirectional road with a single lane in each direction. The known characteristics of the road can be used in the development of the lane detection algorithm and simplifying assumptions can be made. The road contains only one lane marking, separating the two lanes in the middle and is bounded on the ends by the curb. Therefore, the vehicle lateral position and heading will be determined with respect to the middle lane marking only. The impact of this on the control scheme will be discussed in Section 6.4.

The lane detection pipeline design is presented in Figure 6.2. The first step in a complete lane detection pipeline, after the standard distortion correction, is filtering of the raw image to remove lighting affects that can occur in a normal driving situation. The sun is the main cause of such undesired artifacts on the image and effects such as lens flares and bright rays of light on the image can be detrimental to accurate lane detection. Moreover, it is common for the image to have areas of non uniform light intensity caused by shadows from nearby infrastructure. Although software filtering can be applied to correct these effects, it is chosen to address them physically for simplicity and speed in implementation. The steps taken to address these issues for the real world platform will be discussed in the following chapter. Moreover, the software platform tests are designed to have ideal lighting conditions so that these effects are not an issue.

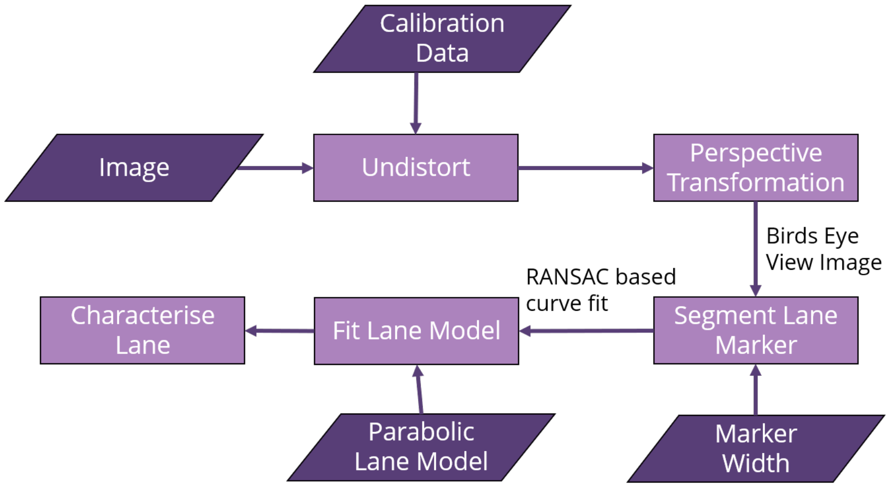


Figure 6.2: Lane Detection Pipeline Design

The next step of the lane detection pipeline is the utilization of the camera parameters in order to transform the original image frame into a top-down frame as shown in Figure 6.3. These parameters include the focal length, principal point, and image size, which are attained from the camera calibration process explained in subsection 2.4.1, and also the height and pitch angle of the mounted camera. Initially it is assumed that perturbations in pitch angle of the camera with respect to the road will not cause a significant effect on the lane detection result. As such, the initial pitch angle of the camera is used as a constant input to the transformation. Testing on real world roads may prove this assumption false, in which case the transformation will also implement dynamic pitch compensation. In the front facing image, parallel lane markings appear to converge at the horizon due to the perspective of the image. If the image perspective is transformed to the top-down view referred to as birds eye view, this perspective effect is removed and parallel lane markings are once again parallel. The birds eye view image is computed based on a rectangular sub section of the original image that can be arbitrarily defined. The idea is to utilize only the relevant area around the expected location of the lane marking for lane detection so as to reduce the processing times and any undesired objects in the processed image. The birds eye view image is produced using the inverse perspective mapping algorithm explained in Section 2.9.

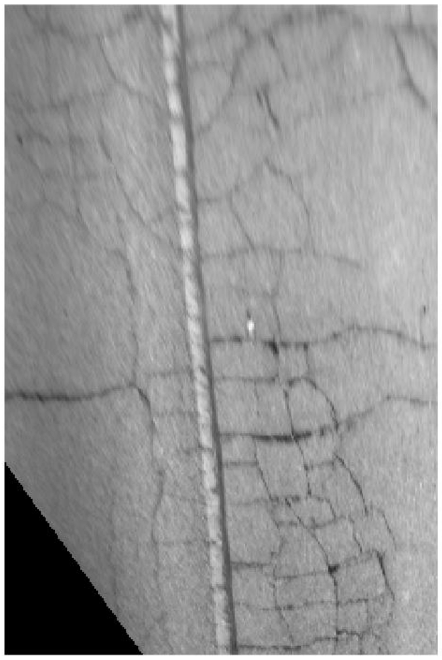


Figure 6.3: Birds Eye View Produced by Perspective Transformation

The next step of the pipeline is to identify the lane markings from the rest of the image as shown in Figure 6.4. The process employed is explained in detail in Section 2.10. In summary, the algorithm takes as input the approximate expected width of the lane marking and finds the markings in the image by searching for groups of pixels that have a high intensity contrast from the surrounding pixels and are near the specified width. Internally, this is achieve through a filter that expects a marking to have low intensity pixels neighboring it. Furthermore, the sensitivity to the lane markings can be controlled via a parameter. Any number of different lane markings detected by the algorithm can be utilized and the algorithm works just as well for a multi lane highway as it does for the single lane marking example shown.

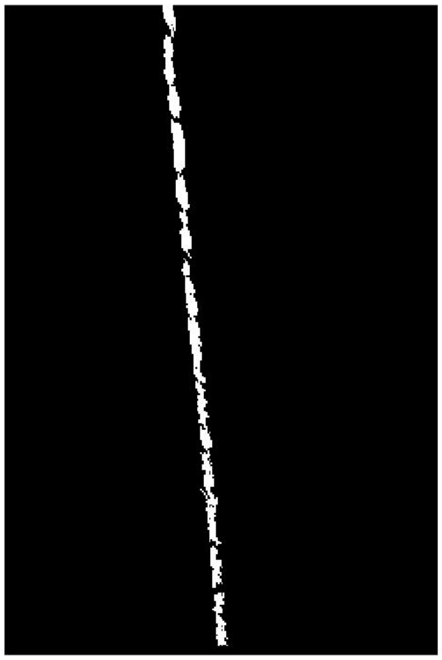


Figure 6.4: Segmentation of Birds Eye View Image

Next, parabolic or cubic lane models can be fit to the segmented lane markings which essentially fit the respective curves to the points making up the lane markings. A RANSAC based method is used for robust curve fitting. The parameters for the model are returned in the original camera coordinate frame and is shown in Figure 6.5.

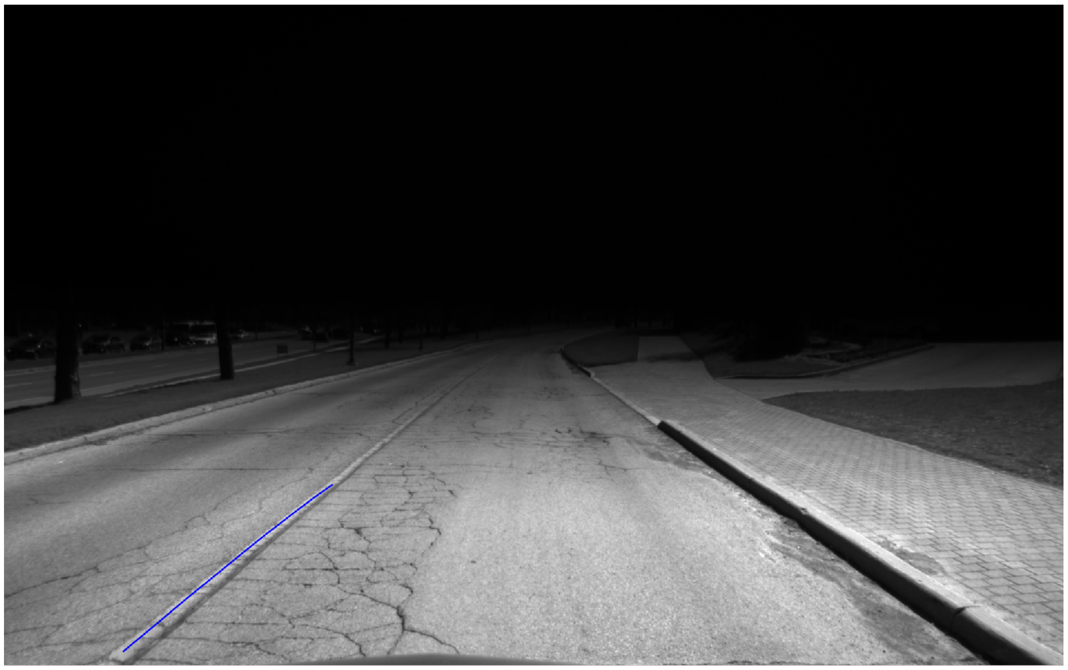


Figure 6.5: Polynomial Lane Model Fit Results on Original Image

6.4 Lane Keeping with Combined Vision/Navigation Solution

The experimental road is not suitable for lane keeping using only the vision system since there are a few stretches without any lane markings. If a lane marking is continuous for the full desired travel path, the vision lateral and heading error metrics as described in Section 6.4.1 and Section 6.4.2 can be used with the controller. Instead, a joint vision and navigation system is proposed to autonomously navigate the disjointed lane markings, yet still benefit from the more accurate information coming from the vision system.

The navigation strategy utilized is to use the heading error metric from the navigation solution and the lateral error metric from the vision solution. For the short sections where there is no lane marking, lane keeping is achieved based on only the heading error metric from the navigation solution. When a lane marking is present, both error metrics are used so as to better control the lateral position of the vehicle in the lane. Since there is only a left side lane marking, the vision lateral error metric always seeks to keep the vehicle at a fixed offset to the right of the lane marking.

The motivation behind this combined system utilized in this way is based on the observation of human driving habits. Human drivers tend to drive by mainly correcting for heading angle by looking ahead at some distance. Although adjustments are made when the vehicle moves too close to either side of the lane, minor lateral errors are not corrected and are deemed insignificant when the vehicle is near the center of the lane. As such, the heading angle must always be available to the autonomous system. Since the heading angle based on the vision system cannot be calculated if the lane marking is not present, the navigation system should provide the heading error metric. The vision system is used to provide the lateral error metric which can be discontinuous for short distances. When the lateral error metric is present, the vehicle will center itself well within the lane, but when it is not present, the correction based on the heading error will still keep the vehicle relatively on the correct path until the lateral error metric is available again.

Another motivation for using the vision system to augment the navigation solution in this way is that the required accuracy of the GPS can be reduced. As seen in Figure 6.1, the heading error metric coming from the navigation system relies on GPS only for localization in the pre defined map which is then used to find the closest point on the map from the vehicle in order to calculate the dynamic look ahead distance d and to calculate θ_d. It can be observed that increased error in the localized position, does not significantly alter θ_d, and only a large localization error would cause θ_d to be directed in a way that does not bring the vehicle towards the prior path as desired. The vision system further helps this by correcting for some of the added error by providing accurate lateral error metric which guides the final steering angle towards the right value.

6.4.1 Vision Lateral Error Metric

Since the parabolic lane model $y = a \cdot x^2 + b \cdot x + c$ is defined with respect to the vehicle frame as shown in Figure 6.6, the lateral position, l, of the vehicle from the lane marking is simply the c coefficient from the model with a positive value implying that the vehicle is to the right of the lane marking. To determine the error, the left marking offset, o, should be subtracted from the lateral position.

In order to further smooth the jitters present in subsequent lane marking detections, exponential smoothing is employed on the lateral error measurement as shown in Equation 6.6. The smoothing factor α controls the amount of smoothing of the lateral error at time i by adjusting it based on the lateral error at the previous time step $i - 1$.

$$c_i = (l - o) \cdot \alpha + c_{i-1} \cdot (1 - \alpha) \tag{6.6}$$

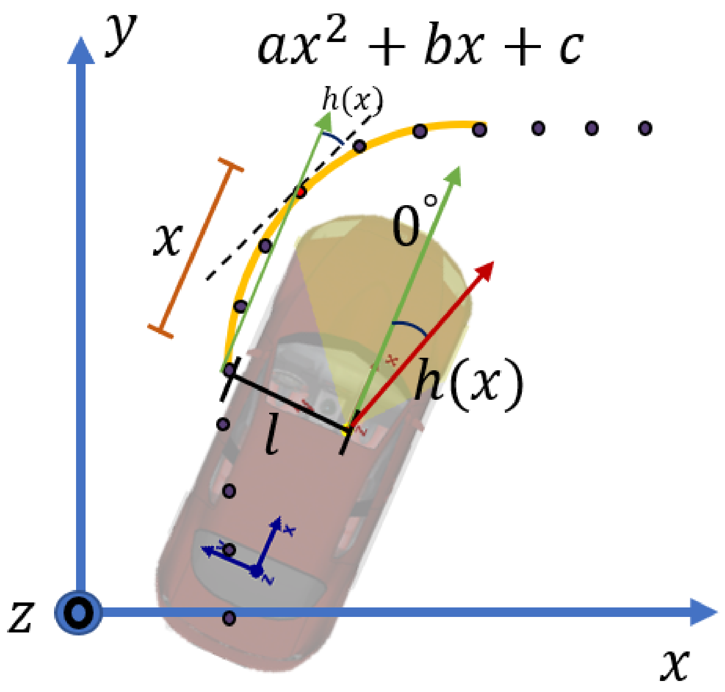


Figure 6.6: Visual Depiction of Vision System Error Metrics

6.4.2 Vision Heading Error Metric

The heading measurement can also be derived from the parabolic lane model. By definition of the vehicle frame, the vehicle heading is always 0°. The lane heading at an arbitrary distance x from the vehicle can be calculated by Equation 6.7. Therefore, a simple way to follow the lane is to set the vehicle heading to the lane heading $h(x)$. The distance can be dynamically calculated as shown in Equation 6.2 in order to prevent the vehicle from cutting corners.

$$h(x) = \arctan(2 \cdot a \cdot x + b) \tag{6.7}$$

6.5 Simulation Experiments

The simulation scenario created consists of a road with similar curvatures as that of the real world road. A section of the road is shown in Figure 6.7. It can be seen that areas of the road are made to not have a left lane marking to simulate this similar aspect of the real world road. The ego vehicle in the simulation has a forward mounted camera that is able to observe the road ahead. Moreover, the simulation software provides fully accurate GPS position of the vehicle with respect to a global coordinate system as well as the vehicle roll, pitch and yaw motion, similar to the navigation solution on the developed platform.

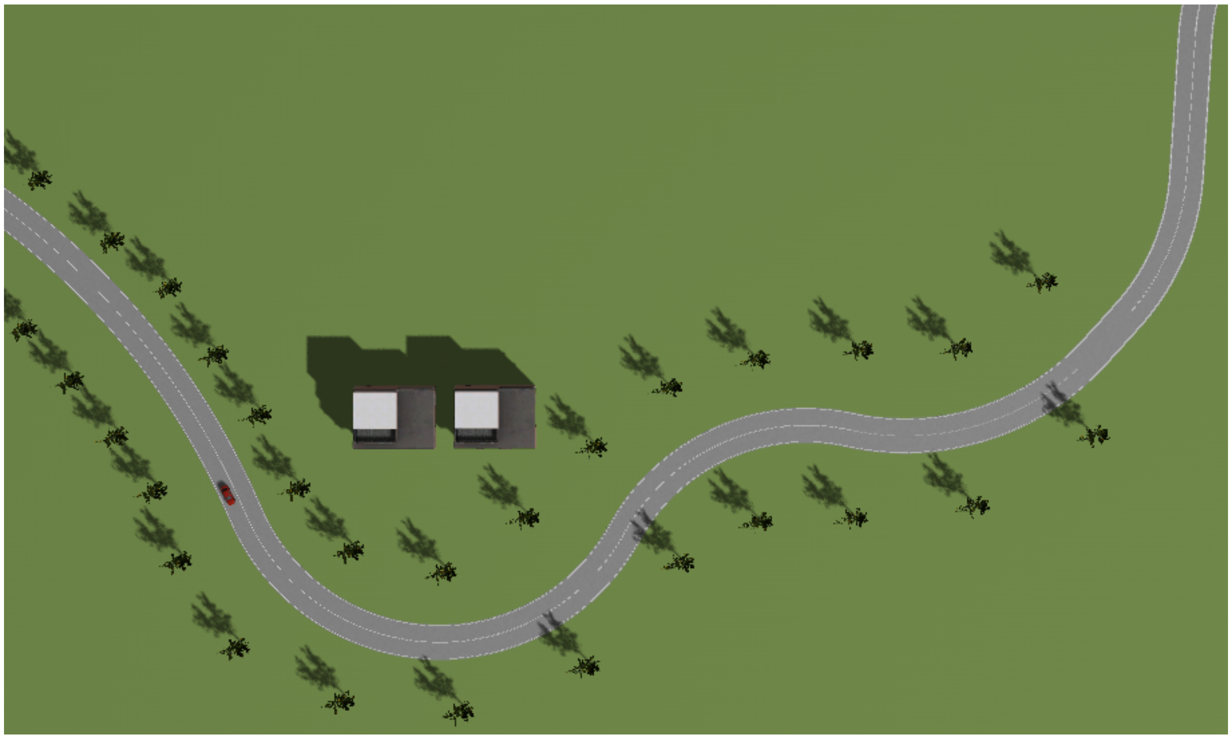


Figure 6.7: Top View of a Section of the Simulation Scenario

To generate the GPS map as required by the lane following approach, the vehicle is automatically driven through the course using the simulation software and the GPS way points are collected at 20 fps. The pre-processing explained in subsection 6.2.3, is conducted and Figure 6.8 shows the calculated curvature plot. As explained, this curvature is used to dynamically vary the look ahead distance with which the heading error metric is calculated, keeping the controller stable but also preventing the vehicle from cutting corners. It can be observed that the maximum absolute curvature of the simulated track is 0.044 $\frac{1}{m}$, similar to that of the experimental road.

The lane detection algorithm, as explained in Section 6.3, is performed on images from the simulated camera. Figure 6.9 shows the lane detection algorithm pipeline for a sample

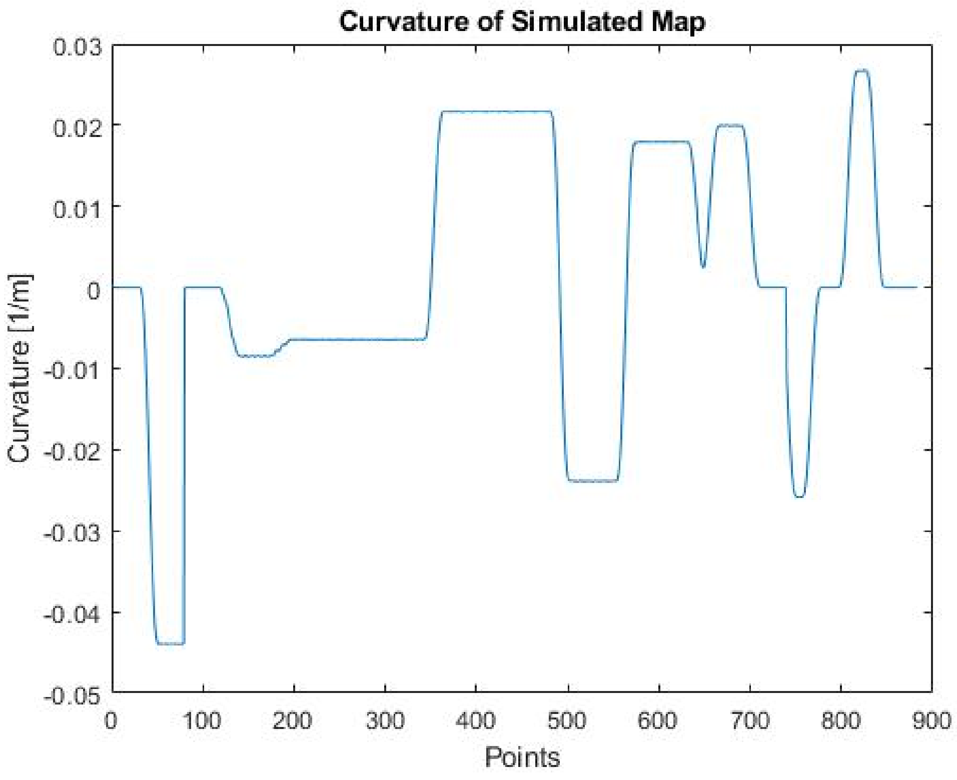


Figure 6.8: Curvature of Points for Simulation Scenario Map

frame. First, the distortion corrected image is transformed to a birds eye view area located around the lane marking. Next, the lane line is segmented from the image. Finally, a parabolic lane model is fit to the line and obtained in the camera coordinate frame.

For the simulation tests, the vehicle starts at standstill and a fixed throttle value is applied to the vehicle. That is to say that the vehicle speed is constantly increasing throughout the test. By the end of the track, the vehicle reaches approximately 50 $\frac{km}{h}$. This is consistent with the maximum speed of 40 $\frac{km}{h}$ on the real world road. Two tests are performed. In the first test, only the GPS and IMU system is used to navigate the vehicle using the prior known map. The gains of the lane keeping controller shown in Equation 6.1 are set to $kp_1 = 1$, $kp_2 = 0.1$, and $kp_3 = 1.2$. In the second test, the vision system is used alongside the navigation solution and kp_2 is raised to 2. This test also compares the results of the combined system with only the navigation solution based system. This will give more weight to the lateral error in the steering angle calculation and will emphasize

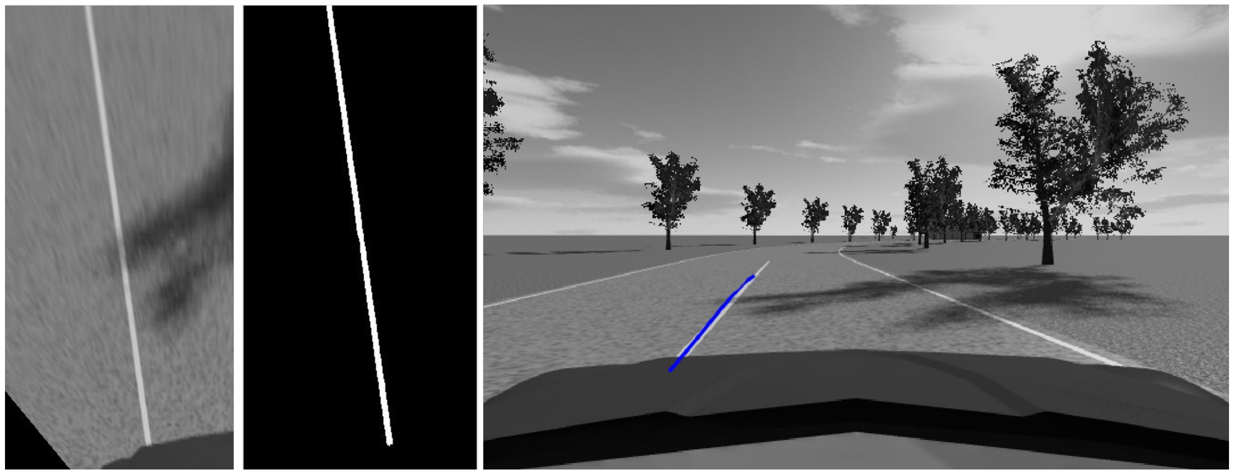


Figure 6.9: Lane Detection Pipeline. Left: Birds Eye View Image, Middle: Segmented Lane Marking, Right: Lane Detection Result after Parabolic Fit

the effect of the vision system being used to aid the navigation solution. The ratio of the steering wheel angle to the tire angle is accounted for and the steering wheel is constantly adjusted to the desired value at 20 Hz. The results are measured for a section of the track and is consistent across both tests. The section consists of curvy roads that change in direction often and have different amounts of curvatures.

Figure 6.10 shows the result of the first test. The test is performed a total of four times with different amounts of noise injected into the GPS position. Noise with a covariance of 10, 35, and 50 cm is injected and the no noise case is taken as the base result. In the no noise case, a small lateral error is present as the vehicle is following the curvature of the road. The initial lateral error is due to the starting position of the vehicle being offset from the path to be followed. Moreover, the oscillations present in the no noise case between 7 and 25 seconds are due to the frequency at which the map points are collected. Since interpolation techniques are not employed when calculating the closest point to the vehicle in the determination of the lateral error, the error tends to get oscillate due to the distance between the points on the map. When the vehicle speeds up, the effect becomes less prominent. The effect of the noise can be easily observed. As the noise level increases, the vehicle tends to oscillate more and more while following the lane. These oscillations increase in magnitude as the magnitude of noise increases. This is expected as the GPS is giving less accurate readings and the lateral error calculation will jump from frame to frame. In all cases, the total lateral error is within and absolute value of 0.4 meters from the planned path. That is to say, that the vehicle adequately tracks the path and navigates the track.

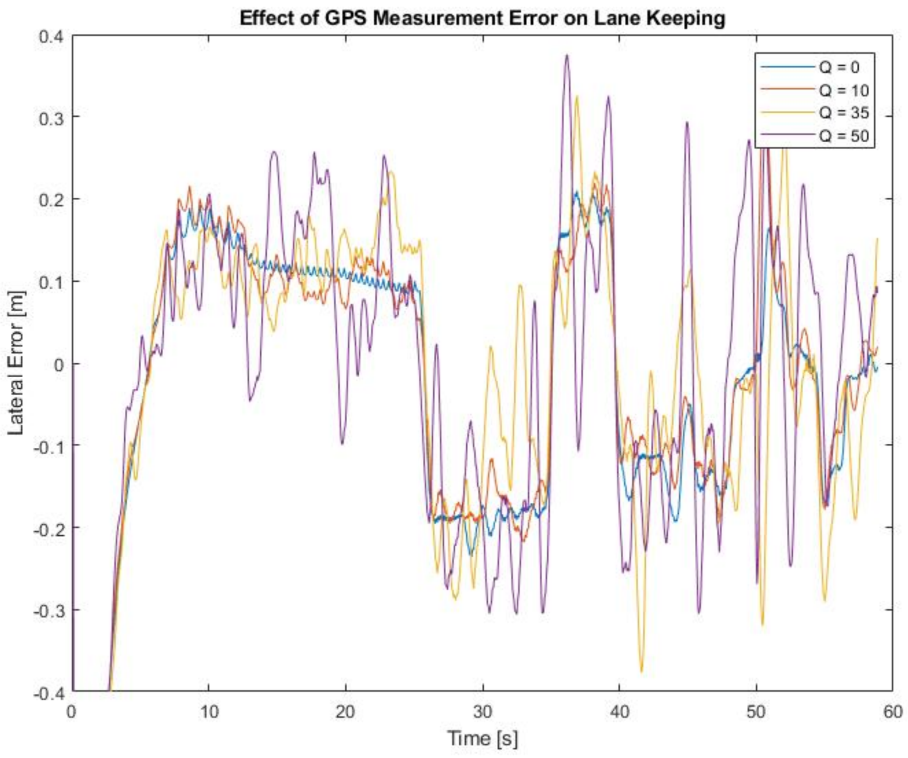


Figure 6.10: Effect of GPS Position Noise on Lane Following

Figure 6.11 shows the results of the second test. A noise covariance of 50 cm is injected into the GPS system. Since the lateral error gain is increased the magnitude of the oscillations caused by the GPS error have also increased. It can be seen that the oscillations reach a maximum magnitude of 2 meters. This is because the lateral position of the vehicle is given too much emphasis by the controller. In a more stable scenario, and corresponding to human driving, the heading error metric should be given more significance than the lateral error metric. However, increasing this gain helps to observe the effect of the combined vision and GPS system. It can be seen that the combined system has smaller oscillations and the lateral error is reduced in magnitude as the vehicle follows the path. This is because the vision system has a much higher accuracy than a GPS and IMU solution. These results show the benefit of the vision system being used as a source of more accurate lateral error data in the behaviour of the lane keeping of the vehicle. The

GPS only system under these conditions would have driven onto the curb or the adjacent oncoming lane resulting in a hazardous and unsafe situation. In comparison, the maximum lateral error of the combined system is around 35 cm deviation from the planned path, and the vehicle is still within the lane.

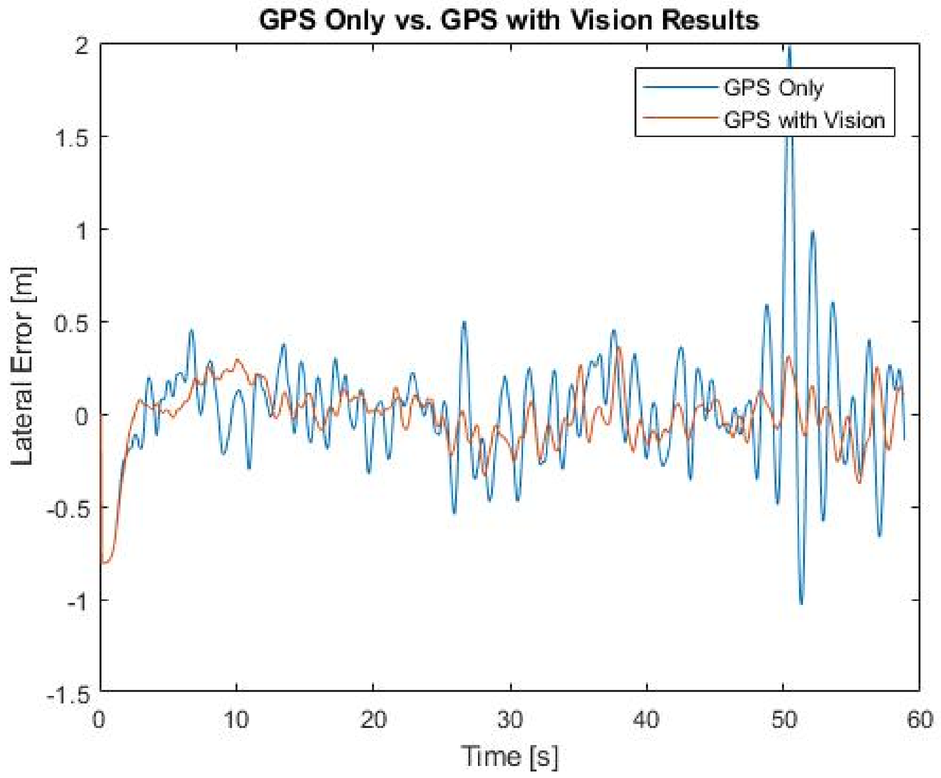


Figure 6.11: Comparison of GPS Only System with Combined GPS and Vision System

Chapter 7

Experimental Setup

In this chapter the autonomous vehicle platform is realized based on the design requirements presented in chapter 3 and is prepared for the lane keeping experiments. The developed platform is explained in Section 7.1 including details about the selected sensors, actuators, OS, and computing systems. Moreover, time synchronization of sensors for the platform is discussed in Section 7.2. Finally, the calibration algorithms are employed and tested in order to be able to represent all the different sensor data in a common frame in Section 7.3.

7.1 Experimental Platform

For the autonomous vehicle platform, it is decided to use an ordinary vehicle as the platform base and outfit it with all the necessary hardware and systems to make it into an autonomous vehicle platform. A preexisting autonomous vehicle platform is not purchased as they are costly and restrict the freedom in their use. In using the ordinary vehicle, any changes required can be made without consequence. Moreover, this allows for the selection of the hardware utilized on the vehicle whereas a preexisting platform may come with undesirable hardware packaged with the platform. The chosen vehicle base is the electronic Chevrolet Equinox and is shown in Figure 7.1.

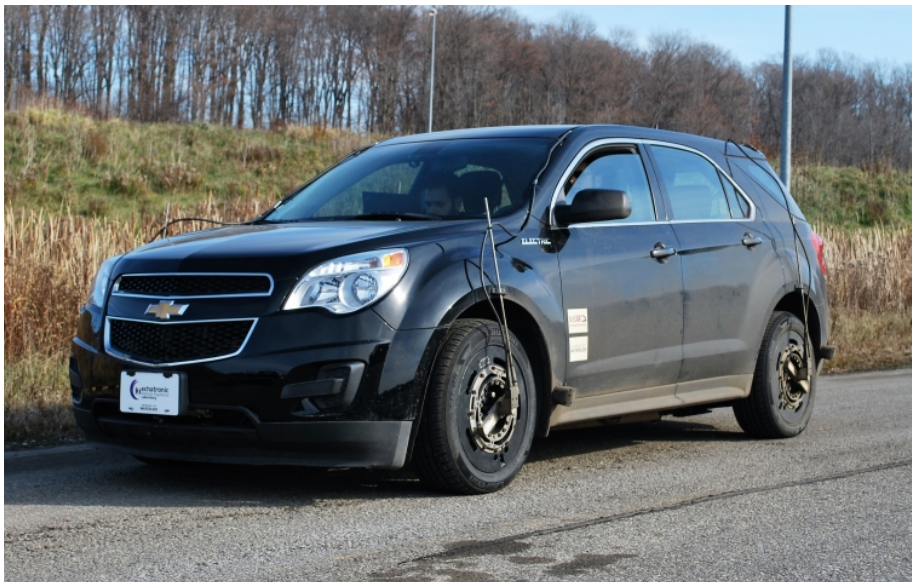


Figure 7.1: Autonomous Platform Base Vehicle: Chevrolet Equinox

7.1.1 Software Platform

The autonomous platform OS will be the main item of discussion in this subsection. The simulation platform selection is explained in chapter 4 and general best practices for user level code will be utilized and specific rules will not be placed on them for the development of this platform.

When it comes to industrial level operating systems there are a few available on the market. The first of these is the QNX Neutrino RTOS available from BlackBerry [53]. This real time OS is designed to be highly reliable and secure and is a good choice when these criteria are most important to the system. Moreover, the VxWorks RTOS from Wind River is another industrial real time OS conforming to various industry standards for reliability, safety and performance [54]. A slew of other paid systems are also available, but these seem to be the most popular.

In contrast, the Linux OS is an open source operating system that is widely used for autonomous vehicle platforms. The Ubuntu version of Linux is most widely used in the automotive field. The Apollo project from Baidu is an open source autonomous vehicle platform that is developed on a Linux kernel patch for real time capability [55]. Another open source autonomous vehicle platform is the Autoware project which is also built on top of a real time version of Linux. However, Autoware uses Robot Operating System

(ROS) on top of the Linux RTOS for additional benefits [56].

Since the developed autonomous vehicle platform is a research prototype platform, great importance is placed on speed of development and available resources for the OS. These immediately bring focus to using Linux and ROS for the developed platform since they are open source, have been used a lot for similar applications, and have an abundance of resources available for the same reasons. Therefore, the approach taken is to use ROS until it fails to be able to meet the goals. In hindsight, without the resources and libraries and tools available from ROS, such quick achievement on the developed platform would not be possible. A slew of other robotics frameworks that perform some of the tasks that ROS performs include: Player, YARP, Orocos, CARMEN, Orca, MOOS, Microsoft Robotics Studio, and Rock. These are not considered as they do not come close to the features provided by ROS for an autonomous platform and do not have nearly the same user base and resources.

The final software platform design is shown in Figure 7.2. ROS is a collection of open source software libraries and tools for robotics applications. Drivers to interface with sensors and actuators are readily available in ROS as well as the integration of major software libraries such as Open Computer Vision (OpenCV) for image processing tools and Point Cloud Library (PCL) for point cloud processing tools. Furthermore, ROS provides easy to use data acquisition, playback and visualization utilities with readily available support [57].

The basic building blocks of ROS are nodes. ROS nodes are processes that perform tasks. Nodes can perform a variety of tasks from interfacing with sensors to things like path planning. ROS nodes can communicate with one another through services or topics. Services are a one to one communication mechanism with two nodes establishing a connection and communicating through a request - receive scheme. Topics on the other hand are a one to many communication mechanism where a node can publish data on a topic and any number of nodes can subscribe to the data. The data transfer is formatted as a ROS message which is a dynamic data structure specific to the data being transferred. ROS message formats are discussed in subsubsection 7.1.3.

Although ROS is a complete platform for any kind of robotics development, pairing ROS with Matlab provides additional benefits. Matlab libraries and tools for algorithm development can aid in the development of the autonomous platform. It is generally quicker and simpler to develop and test algorithms in Matlab due to the availability of various functions for common algorithmic tasks. Moreover, university students in this field are well versed in Matlab from classes. Matlab provides a library of functions that allow users to communicate with ROS enabled devices. In essence, it allows ROS nodes to be

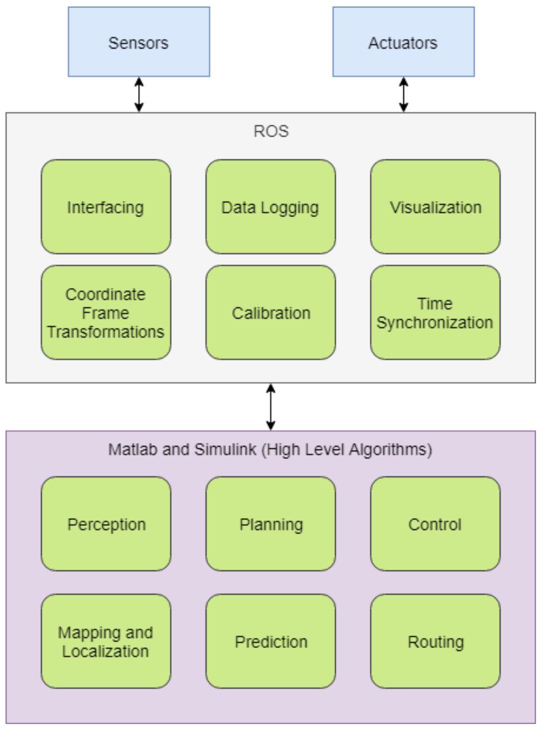


Figure 7.2: Software Architecture using ROS and Matlab/Simulink

developed in Matlab and to establish communication with the normal ROS nodes via topics and services. Therefore, Matlab and Simulink code can be used effectively with ROS.

The final software platform is as follows. ROS is used to interface with sensors and actuators via drivers written in ROS. The drivers publish topics with the sensor data available to other nodes and in the case of actuators the driver subscribes to topics with the actuator commands and communicates it to the actuators via the hardware. ROS also handles the data logging through the use of ROS bags. These bags essentially create a record of the published and subscribed topics and services with timestamps that can be played back in the future. The ROS Visualization (RVIZ) environment is used for visualization of data such as LIDAR pointclouds, radar returns, etc. When the sensors are calibrated, ROS also handles the coordinate frame transformations between sensors automatically. Furthermore, Matlab or ROS can be used for the implementation and testing of the higher level algorithms such as those for perception, planning, controls, routing, prediction, and mapping and localization. Researchers will develop these algorithms and use the established ROS communication to interface with the rest of the platform.

Table 7.1 shows the tasks performed by ROS in comparison to the desired tasks from the OS. Requirements not met include encapsulation of communication for security and system monitoring tools. However, it is possible to add these features to ROS with some additional work. In [58], Linux Container (LXC) is used to fix the security vulnerabilities. Moreover, free user written monitoring tools are available for ROS and can be added to the platform. Going forward, ROS should be used with a real time version of Linux thus allowing for use of high frequency controls algorithms without problems.

Table 7.1: OS Requirements Met by ROS

Task Description	Required	Met by ROS
Resource Allocation	✓	✓
Communication Between Systems and User Code	✓	✓
Interfacing with Sensors	✓	✓
Data Acquisition	✓	✓
Data Recording	✓	✓
Data Visualization	✓	✓
Development Libraries and Tools		✓
Ability to Run on Arbitrary Hardware		✓
Interface with Common Industry Tools		✓
Allow Modular User Code	✓	✓
Encapsulation of Communication	✓	
System Monitoring Tools	✓	
Efficiency in Operation		
Real time		
Facilitates Fast Implementation	✓	✓
Available Resources	✓	✓
Tested in other Autonomous Vehicles	✓	✓
Open Source		✓
Support Distributive Systems		✓

7.1.2 Hardware Platform

The developed hardware architecture is depicted in Figure 7.3 and shows the sensors, actuators, computing systems and their interfacing. The design is simple and able to be implemented quickly with the help of the lab technicians. The lab technicians handle the sensor and actuator installations as well as the ECU programming including the steering and speed controllers. They also expose data coming from base vehicle sensors such as the vehicle speed and steering angle.

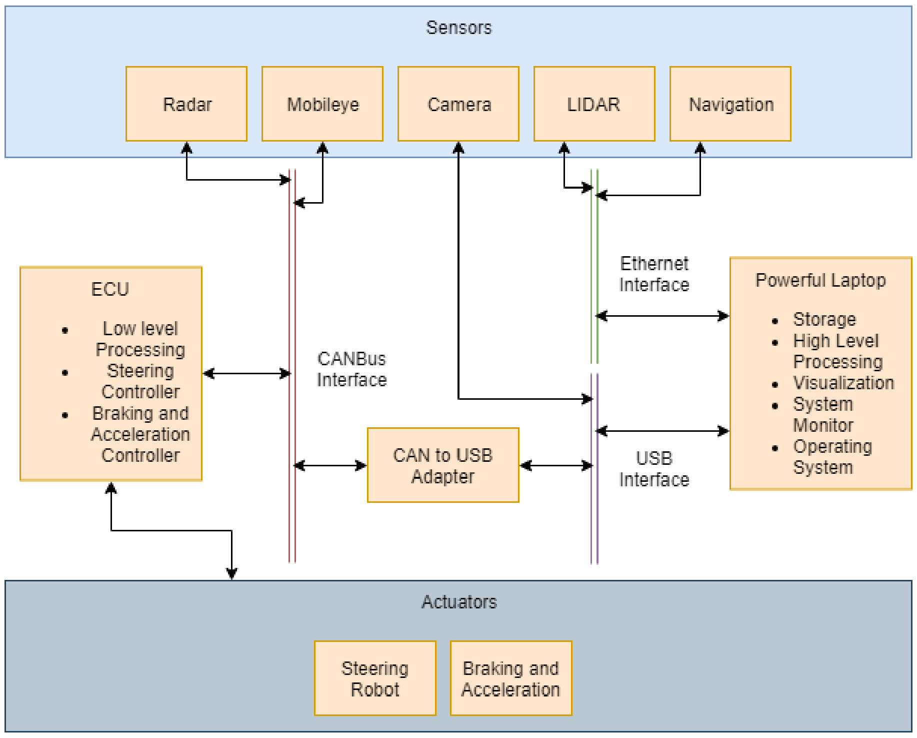


Figure 7.3: Hardware Architecture of Autonomous Driving Platform

Computing Systems

There are two computing systems part of the hardware platform. ECU is an embedded system that is used for low level control algorithms on the vehicle. The system installed on the vehicle is a dSpace AutoBox and is shown in Figure 7.4. It implements a steering controller capable of actuating the steering wheel to a specified angle by commanding the steering robot. Secondly, it interfaces with the vehicle Controller Area Network (CAN) bus to actuate the braking and acceleration inputs of the vehicle. The desired steering angle and vehicle speed can be send to the ECU through the CAN bus as a specific message generated by the laptop.

Figure 7.4: ECU Installed on Developed Platform

The laptop is a second computing system that is running the software platform previously explained. It uses an Intel Xeon E3-1535M v6 CPU with a clock speed of 3.10 GHz, 32.0 GB of RAM and an NVIDIA Quadro M2200 graphics card with 4.0 GB of VRAM. Sufficient storage space is present to record results. This machine runs ROS and Matlab simultaneously, runs the high level algorithms quickly, and interfaces with the vehicle CAN network through a CAN to USB adapter for actuation of the vehicle. Sensor interfacing is achieved directly with the laptop with different sensors transmitting data via CAN, USB, or Ethernet interfaces. Interfacing of sensors with the OS is explained in detail in Section 7.1.3. The laptop also handles visualization of data.

Sensor Selection and Installation

The sensors implemented on the platform include a navigation solution consisting of a coupled GPS and IMU system, a machine vision monocular camera as well as a Mobileye camera, a 3D LIDAR system, and six radar modules of varying ranges. Figure 7.5 shows the mounting locations of the sensors on the vehicle. Details on the exact sensors and some common specifications are presented in Table 7.2.

The camera used in this platform is a Basler acA1920-40um with a 2.3 MP resolution and a pixel size of 5.86 μm by 5.86 μm. It is a monochrome sensor with a maximum frame rate of 41 frames per second. A larger than average pixel size was chosen alongside a monochrome sensor so as to capture a broader light spectrum thereby improving the performance of the camera in low light conditions. Furthermore, these choices result in higher light sensitivity and benefits lane detection algorithms which seek to detect lane

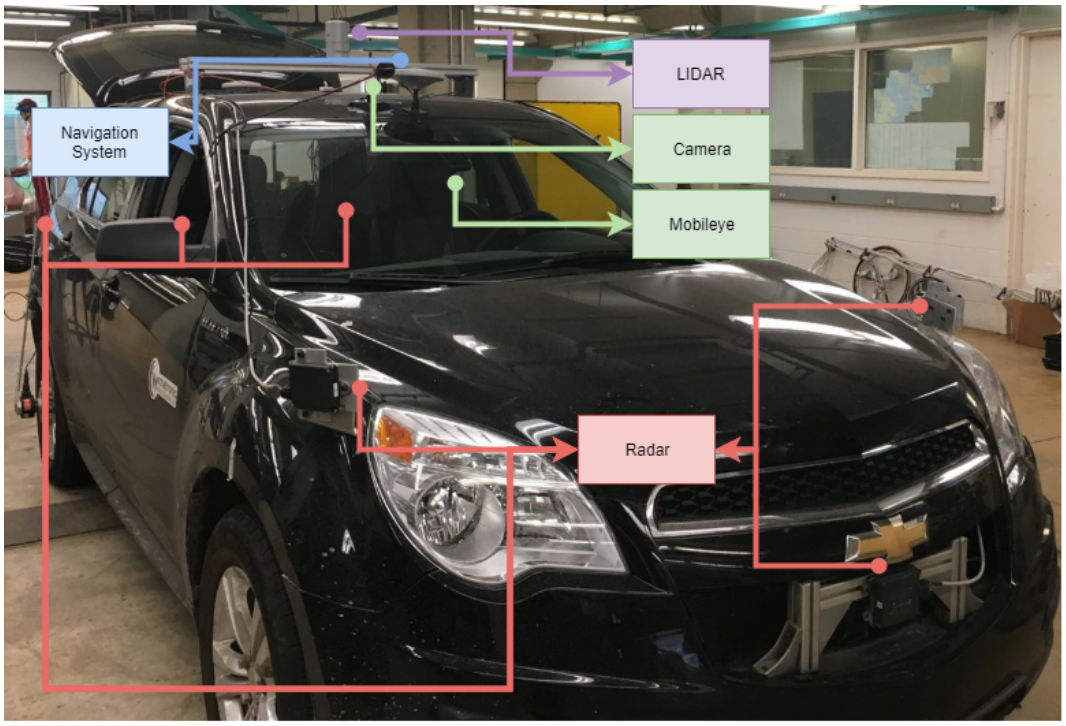


Figure 7.5: Sensor Locations on Developed Platform

markings based on their contrast with the road. Furthermore, the camera features a global shutter that allows for the entire image to be captured at the same time, removing motion blur effects that would occur otherwise.

The lens is selected in order to provide a large field of view at the working distance of the lane detection algorithm. The lens used is a Kowa LM8HC lens which allows the lane to be detected 2 meters in front of the vehicle. At greater distances, the field of view allows for the detection of multiple lanes.

The Mobileye sensor used in the platform utilizes a camera and on board algorithms to provide processed information about the surroundings. The unit can provide lane detection information, obstacle detection data including a characterization of the obstacle, and traffic sign information. It is centrally mounted on the windshield of the vehicle and monitors the area ahead of the vehicle.

The GPS system used on the platform is the differential system from Applanix called the POS LV. The POS LV system has a positional accuracy in the range of 35 cm to 50 cm as well as a heading angle accuracy up to 0.02°. Software post processing allows for

Table 7.2: Sensor Selection and Common Specifications

Sensor Name	Type	Data Rate	Field of View	Interface
Basler acA1920-40um Kowa LM8HC	Camera/Lens	41 Hz	2.8m @ 2m	USB 3.0
Mobileye 5	Camera	10 Hz	40° x 30°	CAN
Applanix POS LV	DGPS	200 Hz	∞	Ethernet
Continental ARS 408-21	Radar Long Range	16.7 Hz	4° @ 250m 90° @ 70m 120° @ 20m	CAN
Continental SRR 20X	Radar Short Range	30 Hz	40° @ 50m	CAN
Velodyne Hdl-32E	Lidar	10 Hz	360° $\leq$ 70m	Ethernet

recorded data to be processed to an accuracy $\leq 10cm$. The addition of a RTK system to this GPS would allow such accuracy in real time.

Radars can be designed to have different operating distances and field of views based on signal frequency. The developed platform utilizes two long range radars operating at a frequency of 77 GHz with an operating distance of 250 m and four short range radars operating at a frequency of 24 GHz with an operating distance of 50 m. Both types of radars have a distance accuracy < 0.5 m.

The developed platform uses a Velodyne HDL-32 LIDAR utilizing 32 lasers to generate point clouds. The sensor provides point clouds at a rate of 10 Hz and working distance of up to 70 meters. Figure 7.6 shows the installation points of some of these sensors close up.

The approximate field of views of the different sensors are depicted in Figure 7.7. It can be seen that the position of the sensors allow for coverage of the entire surrounding of the vehicle. The long range radar sensors mounted on the front and back of the vehicle have the purpose of obstacle detection at close and far distances and will be used in a cruise control system in the future. The four short range radars can detect obstacles in the lateral directions. The LIDAR is centrally mounted to detect everything going on around the vehicle and can provide redundant information to most of the other sensors usable with sensor fusion techniques to make the platform more robust. The machine vision camera is mounted at the front of the vehicle and is used for lane detection and lane keeping tasks. It can also be used for obstacle detection in the future. The Mobileye system provides another source of redundancy for lane detection and obstacle detection. Finally, the navigation system consisting of a differential GPS and IMU system provide reliable localization information for the vehicle.

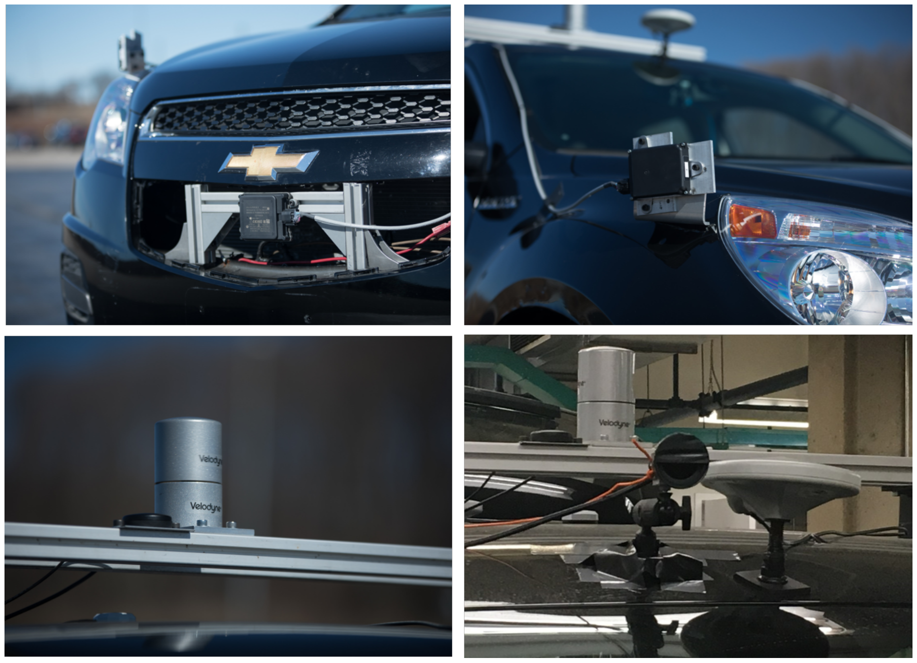


Figure 7.6: Installation View of Sensors with Front and Side Radar (top left), Side Radar and GPS Antenna (top right), LIDAR (bottom left), and Camera (bottom right)

Actuator Selection and Testing

The actuated systems on the vehicle include the steering robot controlling the steering wheel angle and the speed controller regulating vehicle speed. Both of these systems have low level feedback controllers that regulate the outputs to the commanded levels. These low level controllers and actuator device selection are not in the scope of this thesis and are handled by the lab technicians. Tests are performed to measure the response of these controllers under different kinds of commanded inputs. Figure 7.8 shows the response of the actuation under step inputs and Figure 7.9 shows the response of the velocity controller under a ramp input. From these results it can be seen that some tuning is required from the controllers. The steering controller controls the steering wheel angle to a fixed offset approximately 6° from the commanded angle and takes around 1 second to reach the value. For the purposes of the lane keeping controller, the response time should be improved for smoother lane keeping. Both of these issues are corrected by the lab technicians. The speed controller exhibits a noisy response that is also offset from the commanded value. The noise is hypothesized to be measurement noise from the sensor. Although the speed

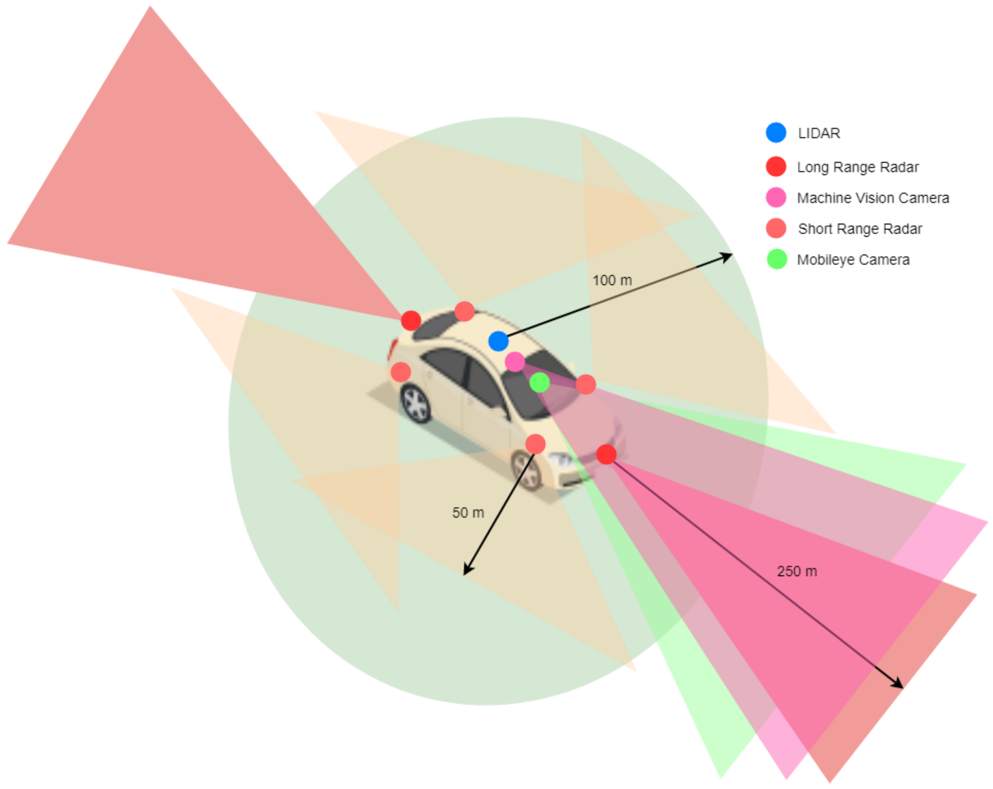
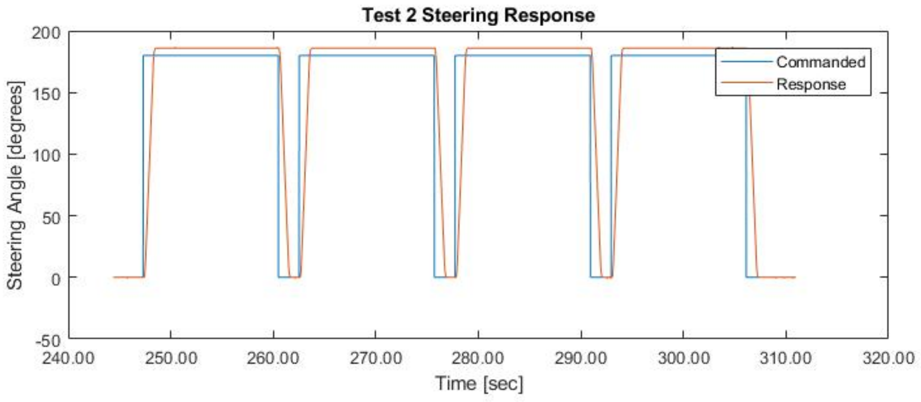


Figure 7.7: Approximate Sensor Field Of Views

controller is not used in this thesis, these issues will be addressed by the lab technicians for the future.

7.1.3 Interfacing

Sensor data needs to be available for the laptop computer to be used by the software algorithms implemented on the autonomous vehicle. Since, some of the sensors used are popular sensors, open source ROS drivers are available to be used. The Camera, LIDAR, and navigation system all have available ROS drivers. This allows for the real time transfer of data from each of these sensors to ROS directly through their respective hardware interface. The data is formatted as specialized ROS messages which are explained in this subsection. The remaining sensors as well as the actuators transmit or receive data via the CAN bus. Thus, it is beneficial to create a common way to interface with all such devices. In this subsection a communication bridge between ROS and the vehicle CAN bus is also



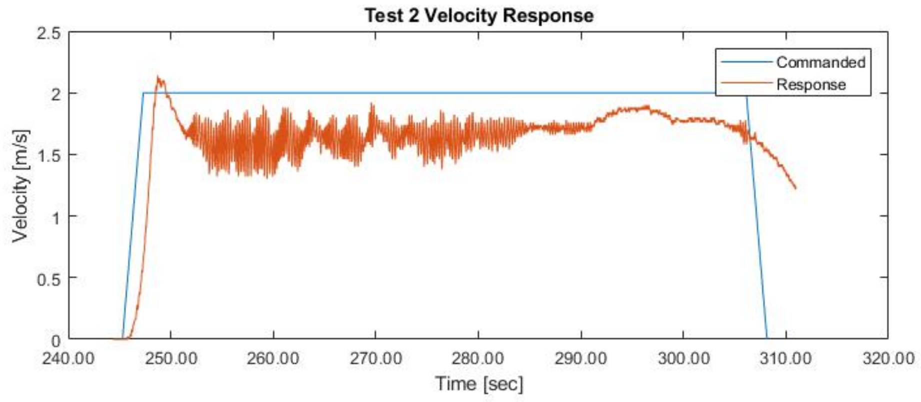


Figure 7.8: Actuator Response Under Step Input

created. This bridge is extensible to new devices communicating to the laptop via the CAN bus and only requires a common file specifying the data format for each device.

Bidirectional CAN Interface

The bidirectional CAN interface is designed with extensibility and functionality in mind. ROS has a CAN library, that interfaces with the linux CAN kernel level drivers, called *ros_canopen*. The *Socketcan Bridge* node is part of this CAN library and provides the raw CAN frames to be used in ROS applications. A generic node is created which can translate the raw CAN data into usable formats and vice versa using another open source library called *libcan_encode_decode*. This library provides functions that help to encode

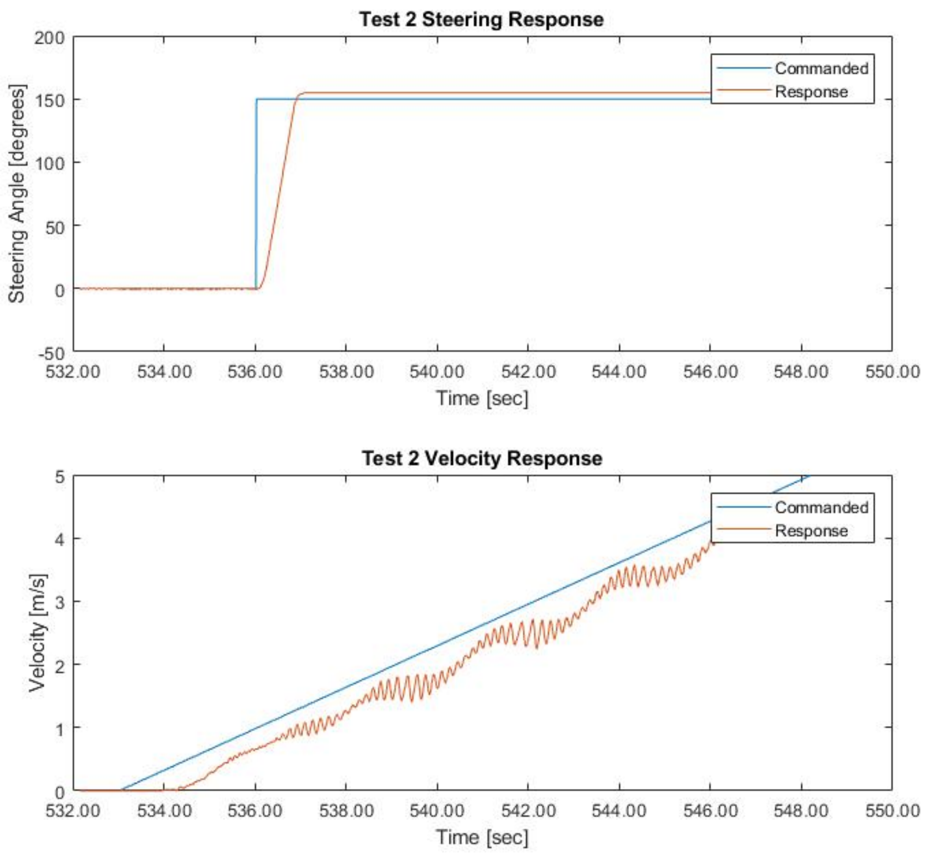


Figure 7.9: Actuator Response Under Step and Ramp Inputs

and decode CAN frames. The generic node utilizes a sensor specific database file which is a industry standard file that provides information about the CAN messages sent and received by the sensors. With this setup, any new sensor utilizing the CAN bus can be interfaced with by simply switching the database files. The generic node publishes the decoded sensor data as ROS topics available to the other processing nodes. Actuation is handled in a similar way using the same interfaces. A generic write node uses a actuator specific database file to encode the desired data to be sent to an actuator, and the *ros_canopen* library handles the low level data transfer.

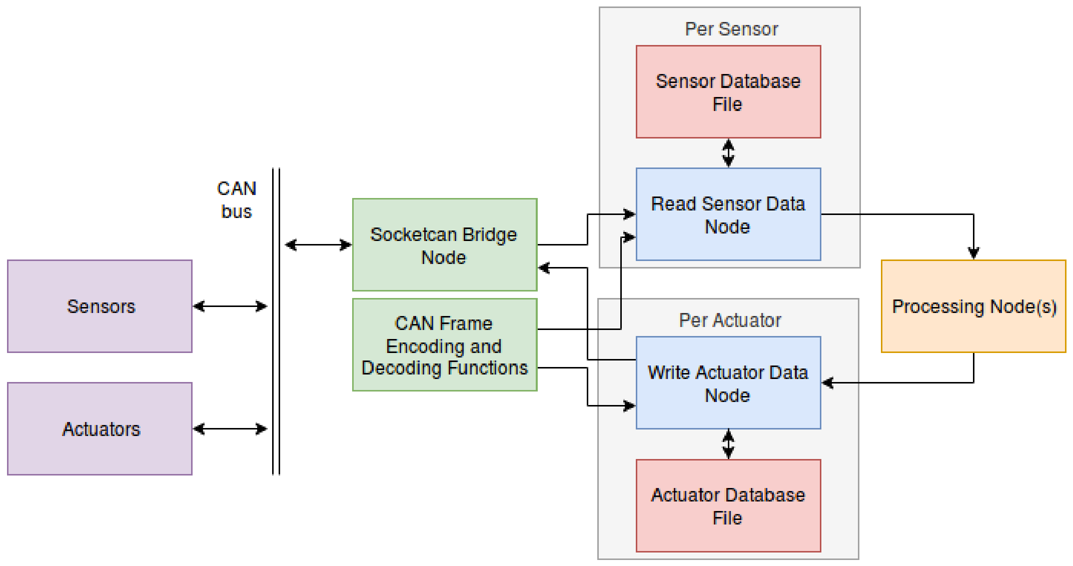


Figure 7.10: Diagram of CAN Communication Interface

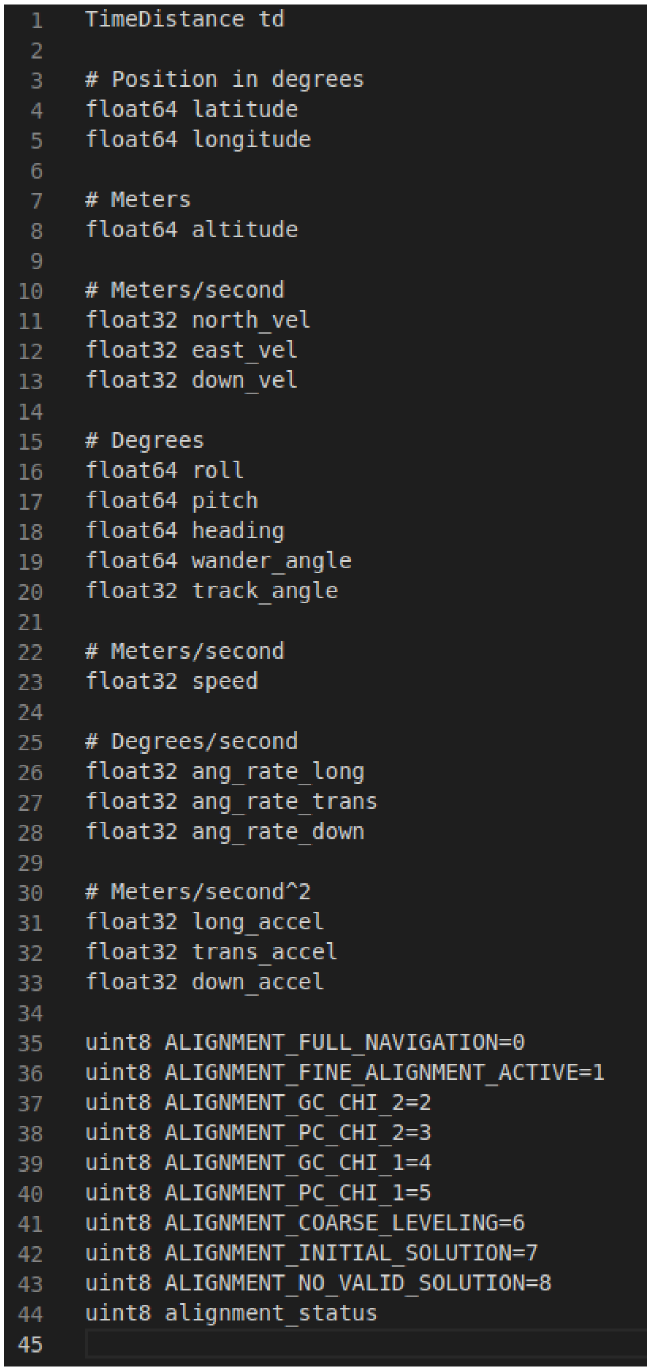
ROS Message Formats

ROS messages are a way of encoding data to be transmitted within the ROS network. ROS provides basic messages that are common to most robotics applications. Custom messages can be created by users encapsulating the basic message types. They can be viewed synonymous to structs in object oriented programming. When ROS drivers are used to interface with sensors, custom message formats are created for the driver. Figure 7.11 displays an example of a custom ROS message with the navigation message format used by the GPS and IMU system.

The CAN communication is designed to be extensible. Therefore, the custom messages required to publish the sensor data to the ROS network are auto generated using the database files. The processing nodes can further encapsulate sensor data into more usable formats if necessary.

7.2 Time Synchronization

Individual sensors operate using a local clock to trigger the data acquisition for each sensor. This is how a machine vision camera can acquire image frames at a set frequency or how a rotating LIDAR determines when to fire each laser to build a point cloud frame. When



```
TimeDistance td

# Position in degrees
float64 latitude
float64 longitude

# Meters
float64 altitude

# Meters/second
float32 north_vel
float32 east_vel
float32 down_vel

# Degrees
float64 roll
float64 pitch
float64 heading
float64 wander_angle
float32 track_angle

# Meters/second
float32 speed

# Degrees/second
float32 ang_rate_long
float32 ang_rate_trans
float32 ang_rate_down

# Meters/second^2
float32 long_accel
float32 trans_accel
float32 down_accel

uint8 ALIGNMENT_FULL_NAVIGATION=0
uint8 ALIGNMENT_FINE_ALIGNMENT_ACTIVE=1
uint8 ALIGNMENT_GC_CHI_2=2
uint8 ALIGNMENT_PC_CHI_2=3
uint8 ALIGNMENT_GC_CHI_1=4
uint8 ALIGNMENT_PC_CHI_1=5
uint8 ALIGNMENT_COARSE_LEVELING=6
uint8 ALIGNMENT_INITIAL_SOLUTION=7
uint8 ALIGNMENT_NO_VALID_SOLUTION=8
uint8 alignment_status

```

Figure 7.11: Navigation Message Format in ROS

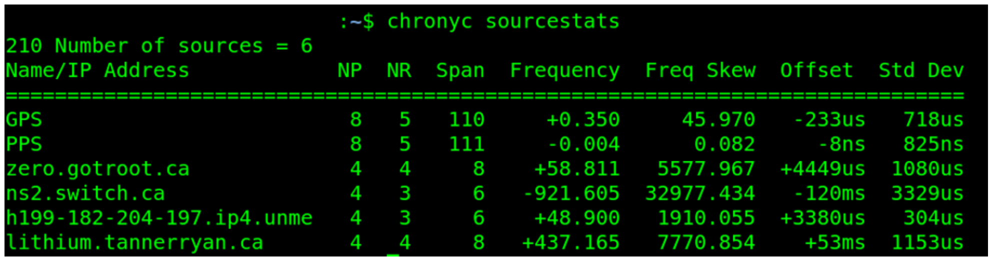
using sensors in a network, such as that of the autonomous driving platform, the local clocks of sensor nodes need to be synchronized. The precision of synchronization is application specific. In a stereo camera system employing two machine vision cameras, both cameras should trigger acquisition at nearly the same instance in time, or subjects in one camera's frame may have shifted in position from the other camera's frame, resulting in a noisy disparity image. However, when fusing data from multiple sensors operating at a very high frequency, it is often justified to simply use the latest frame from each of the sensors. If the data acquisition is sufficiently fast, there will not be a significant discrepancy in the timing.

For the task of lane keeping, it is sufficient to use the latest frames from all of the relevant sensors. ROS handles the data acquisition and provides data streams from all of the sensors. Algorithms may access the latest frame from any of the sensors as necessary. However, since the platform is designed as a test-bed for student work, exact synchronization between sensors may be necessary so that frames are captured at the exact same time. Moreover, individual sensor clocks tend to drift over time so that data acquisition may not happen at a consistent rate. To this end, the hardware used is synchronized as explained in the following subsections. The GPS system clock is used as the global clock for this synchronization since it is highly accurate. It will be used to provide hardware triggers to sensors that are capable to receive such triggers and will be used in conjunction with a Linux machine to synchronize the clocks of sensors to a common source.

7.2.1 GPS and Linux Computer Synchronization

The GPS navigation system creates two types of messages that will be utilized for time synchronization with the Linux machine. The first of these is a National Marine Electronics Association (NMEA) string which is a common and standardized format of transferring the position, velocity and time information computed by the GPS device. The second type of message is the Pulse Per Second (PPS) message which is a repeated signal with a rising edge at every second originated from the GPS clock. These signals can be used in order to synchronize the clock of the Linux machine to the reference clock.

A program called *gpsd* allows the Linux machine to listen to the signals from the GPS. A special cord is created to interface from the GPS output ports to the USB port of the laptop. *gpsd* allows the signals to be used on the laptop by other programs. Next, a program called *chronyc*, which implements the Network Time Protocol (NTP), synchronizes the Linux machine clock to the reference clock from the *gpsd* output. The synchronization is performed based on the NTP protocol which over time achieve synchronization that is robust to discontinuities and preserve monotonicity and chronoscopicity. Figure 7.12 displays the result of the synchronization achieved. It can be seen that the offsets from the synchronization with the GPS is 233 μs with a standard deviation of 718 μs, This is a more than sufficient level of synchronization for algorithms functioning under 100 Hz.



```
                            :~$ chronyc sourcestats
210 Number of sources = 6
Name/IP Address            NP  NR  Span  Frequency  Freq Skew  Offset  Std Dev
==============================================================================
GPS                         8   5   110     +0.350     45.970   -233us    718us
PPS                         8   5   111     -0.004      0.082     -8ns    825ns
zero.gotroot.ca             4   4     8    +58.811   5577.967  +4449us   1080us
ns2.switch.ca               4   3     6   -921.605  32977.434   -120ms   3329us
h199-182-204-197.ip4.unme   4   3     6    +48.900   1910.055  +3380us    304us
lithium.tannerryan.ca       4   4     8   +437.165   7770.854    +53ms   1153us
```

Figure 7.12: Time Synchronization Statistics between GPS and Linux Laptop

Moreover, the acquisition time recorded by ROS, which is based on the local computer clock, will now be accurate with respect to the GPS clock which does not drift as much as the local computer clock. At this point software based synchronization could be achieved between the laptop and the sensors connected to the laptop. Alternatively, if the sensors allow, they can directly be synchronized or triggered from the GPS system. These elements working together ensure that all devices on the platform will always work together correctly. Algorithms that expect ordered data at specific times or the accurate logging of information are some of the things that depend on accurate time synchronization.

7.2.2 Synchronization with Other Sensors

The sensors capable of being synchronized on the platform include the Velodyne LIDAR and the Basler machine vision camera. Both of these devices have built in capability to be triggered from hardware or synchronized from hardware using a General Purpose Input Output (GPIO) cable. To achieve this task, the appropriate signals need to be generated by the GPS and output at the desired frequency to the sensors. The GPIO cables need to specifically constructed to fit with the GPS output interface as well as the sensor input interfaces. Generally the sensor manuals indicate the signals needed and which pins on the input interface should be connected to these signals. On the platform, this process is followed for the LIDAR to be time synchronized with the GPS clock. This will ensure the LIDAR clock does not drift and the timestamps for the LIDAR pointclouds are accurate. The hardware triggering capability of the camera is not utilized currently. Instead, the camera is set to run at a fixed frame rate based on its local clock and the acquisition timestamps from ROS are used. The radar sensors and Mobileye unit do not support being synchronized or triggered from external hardware.

7.3 Calibration Results

In this section, the different calibration techniques are employed and tested to different levels of success. The infrastructure sensor with GPS sensor calibration is only tested in simulation in Section 5.2.1 as a physical setup with an infrastructure mounted sensor was not available. Moreover, this calibration technique is not required until interfacing with the environment is implemented.

7.3.1 Heterogeneous Joint Calibration

For the joint calibration of the developed platform, the front radar, LIDAR, and Mobileye camera are calibrated. At the time of the calibration, the machine vision camera was not purchased or installed. Another calibration may be performed in the future to include this sensor, or the missing transformations may be calculated from the Hand-Eye calibration between the navigation system and the camera. Moreover, the remaining radars are not currently used and therefore are not calibrated with the LIDAR system even though the same method may be used to do so.

The specially designed calibration target is shown in Figure 7.13. The mannequin is detected by the Mobileye sensor as a pedesrian. The radar reflector is mounted at the center of the target and is designed to reflect the radar waves directly back to the radar. This means that the signal strength of the return from the reflector is significantly stronger that many of the surrounding objects and therefore is easy to isolate. The key factors of the target in order to achieve a high strength return are size and material.

The algorithm is implemented as a ROS node using Ceres Solver. In this platform, Ceres Solver, which implements the LM algorithm, is used in order to perform the minimization, although any other software package may be used. A point selector ROS node is also written which allows for sensor points to be clicked in RVIZ and saved to a file. Live sensor data is recorded and visualized in RVIZ where the calibration target is moved to various positions in the region of interest. As this data is played back, the point selector node is utilized and a set 20 corresponding points from each of the three calibrated sensors are written to a file. Since the radar and Mobileye sensor do not provide the height of the detection, the LIDAR based height is used. That is to say that the Z value of the

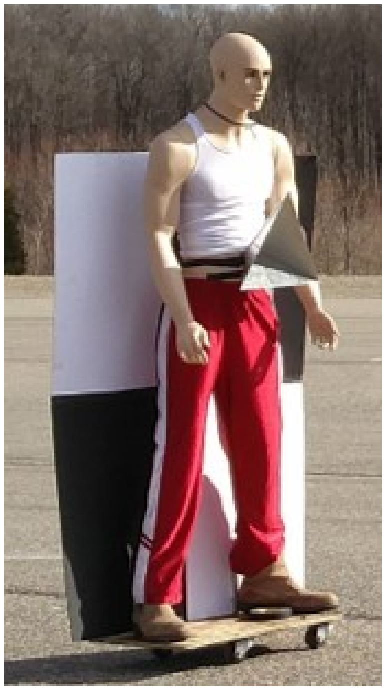


Figure 7.13: Calibration Target for Joint Calibration of Radar, LIDAR, and Mobileye Sensors

point selections comes from the LIDAR point. This is justified since the height of the detections are not important since it is generally assumed that the object is at the ground level. Finally, the calibration ROS node is run and the transformation matrices are found. The transformation from the LIDAR to the Mobileye and from the radar to the LIDAR, respectively, are:

$$\begin{bmatrix} 0.0415 & 0.9991 & 0.0032 & -3.5418 \\ -0.9982 & 0.0413 & 0.0444 & 0.1686 \\ 0.0443 & -0.0050 & 0.9990 & -0.0021 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

$$\begin{bmatrix} 0.0606 & -0.9963 & -0.0615 & 0.3260 \\ 0.9982 & 0.0605 & 0.0077 & 2.4745 \\ -0.0039 & -0.0618 & 0.9981 & 0.0056 \\ 0 & 0 & 0 & 1.0000 \end{bmatrix}$$

One simple way to test for correct calibration is to apply the transformation obtained in RVIZ and transform two of the sensors' detections into the frame of the third detection. In this case, the Mobileye and Radar detections are transformed into the LIDAR coordinate frame. Figure 7.14 shows the alignment of the detections after the transformations are applied. It can be observed that there is some error present since the points do no completely align.

The error metric explained in Section 5.1 is utilized to provide an estimate of the level of error present in the transformations. 10 frames similar to the ones shown are sampled with the distance from the Mobileye and Radar detections to the Lidar detection calculated. The average distance is taken as the error metric and is 0.42 meters. This level of error is currently sufficient as the autonomous vehicle can be given a margin with which to avoid any obstacles to ensure collision free travel.

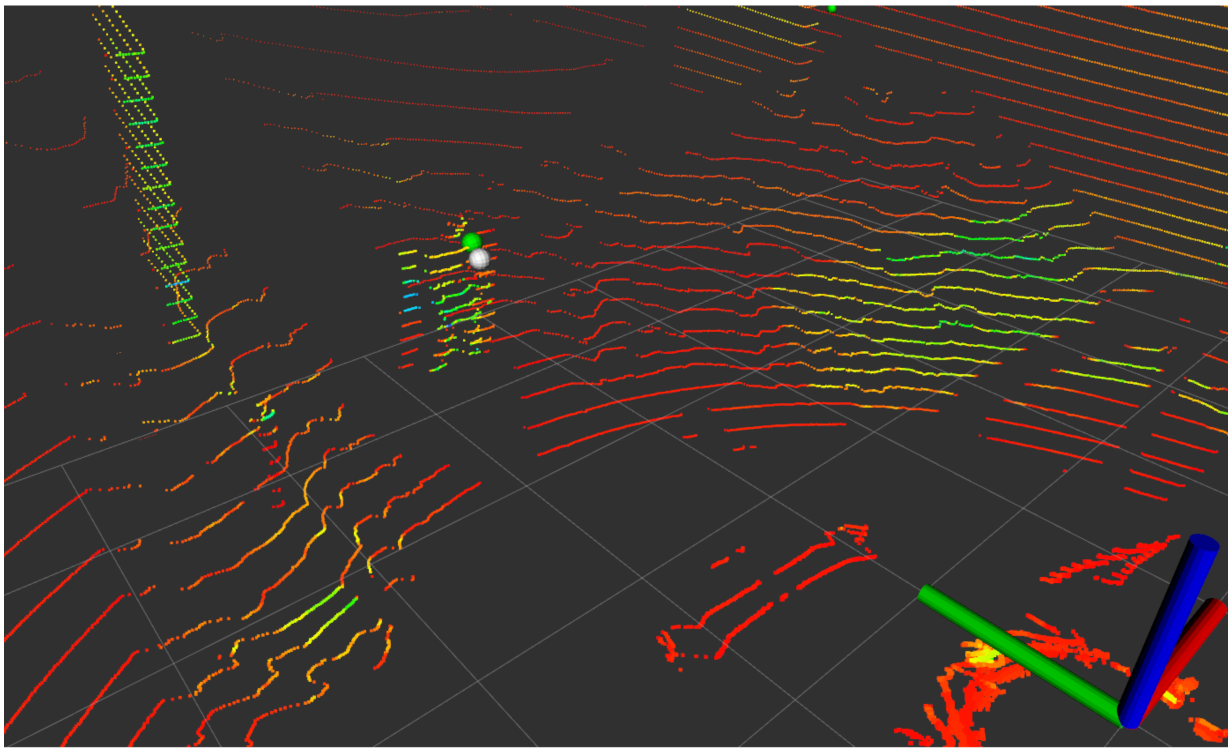


Figure 7.14: Joint Calibration Results. The Mobileye Detection (green) and the Radar Detection (white) are Transformed to the LIDAR Coordinate Frame

There are various causes for errors present in this calibration. First, at the sensor level, each of the sensors have a measurement accuracy that is built into the measurements. There is no way to obtain a more accurate measurement unless more advance and expensive sensors are used. From the corresponding data sheets, the approximate measurement accuracies of the LIDAR, radar, and Mobileye sensors used are 0.02 m, 0.1 m, and $\leq$2 m respectively. The accuracies presented pertain to the maximum distance of the calibration target during calibration. Generally, these measurement noises get worse with distance. Therefore, it is a good idea not to set the calibration target too far from the vehicle. Moreover, error is added into the system during the point selection stage. Ideally, the exact same point on the calibration target is measured by all three sensors. In reality, it is not known what point on the calibration target the Mobileye actually measures, whether it is the center or a corner of the bounding box of the pedestrian detection or some other arbitrary point. Next, the radar reflector is made sufficiently large to provide a strong reflected signal, however this also means that the radar is not detecting the exact center of the reflector all the time. Lastly, the LIDAR point pertaining to the center of the radar reflector is selected manually. However, since the LIDAR point cloud is of a fixed density, the exact center point of the reflector may not be picked up, especially as the target gets further away from the LIDAR. Therefore, the next closest point is selected, but error is introduced into the selection.

These sources of errors result in the calibration to be non ideal. Improvements can be made as follows. First, a machine vision camera could replace the Mobileye unit. When the raw image of the camera is available, a specialized chessboard pattern can be detected with a much higher accuracy as that coming from the Mobileye sensor. Not only that, the machine vision camera will also provide the height of the target point that will ensure some of the data is not compromised. The radar reflector design should be optimized between reflection strength and size. If the calibration is performed in a special area devoid of other objects that can be detected by the radar, the radar reflector can be made significantly smaller, thus providing a more consistent localization for the measurements as the target is moved. Finally, a LIDAR system that provides a denser pointcloud can be used so that the point selection becomes more accurate. Furthermore, interpolation techniques between LIDAR points would greatly help in allowing the selection of the point at the middle of the radar reflector.

7.3.2 Navigation System and Camera

The hand-eye calibration between the navigation system and vehicle mounted camera was attempted with the real world sensors. It can be seen from Figure7.15 that the calibration target size is relatively small. This results in inaccuracy in the corner detection algorithm since the squares of the checkerboard do not contain enough pixels to determine the exact corner locations to a high accuracy. Since the calibration had to be completed outdoors in order to obtain GPS signal, a sufficient way to mount the chessboard rigidly in the air was not possible. If the chessboard is rigidly hanging in the air, the vehicle can be driven very close to the target so that the target takes up a larger section of the image and corner detection error is decreased. Instead, the limitation of having the chessboard sitting on the chair restricted the size of the target in the image due to the mounting location of the camera being very far from the target. This problem can be solved by using a much larger calibration chessboard which is not available at the time of this work, or to somehow mount the chessboard in the air such that the vehicle can drive under it.

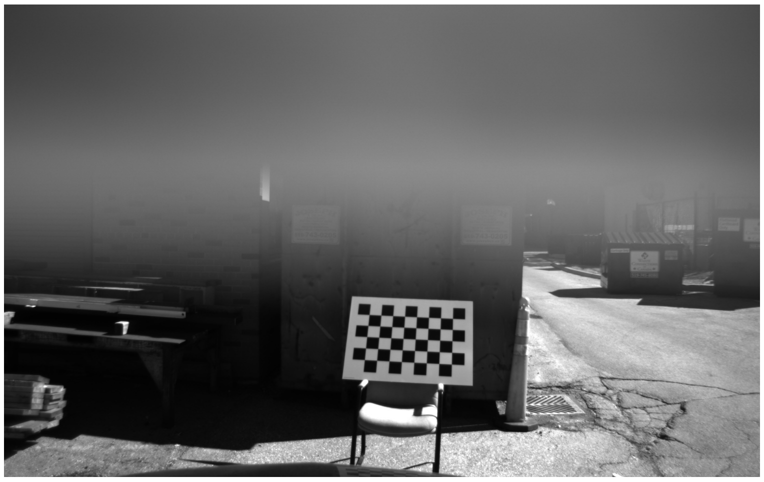


Figure 7.15: Cropped Camera Image from Hand-Eye Calibration Between Camera and Navigation System

Moreover, the navigation system has a typical error around 0.5 meters which is also significant for this calibration. Without an RTK GPS system, it is very difficult to obtain a higher accuracy result. Using the DGPS system on the vehicle, it is possible to post process the calibration routine for higher accuracy and should be utilized in the future. However, if calibration between a less accurate GPS system, without DGPS or RTK capability, and a camera needs to be performed, evidence shows that the hand-eye calibration method is not suitable for an accurate calibration.

Due to these sources of error, a useful calibration could not be performed. The transformation obtained through the calibration was highly inaccurate and is not able to be utilized. Instead, it is decided to measure the transformation by hand as precisely as possible when needed. In order to achieve a usable calibration with the implemented method, the calibration chessboard should be much larger and the navigation solution should have centimeter level accuracy.

7.3.3 Navigation System and LIDAR

The calibration between the navigation system and the LIDAR sensor is implemented with the developed platform. First, the implementation of the ICP algorithm is tested. Figure 7.16 shows the result of the implemented ICP algorithm explained in Section 2.8 for two point clouds from the Kitti dataset. It can be seen that the red and green point clouds are aligned through the ICP algorithm and that the ICP algorithm implementation seems to works well on this test set.

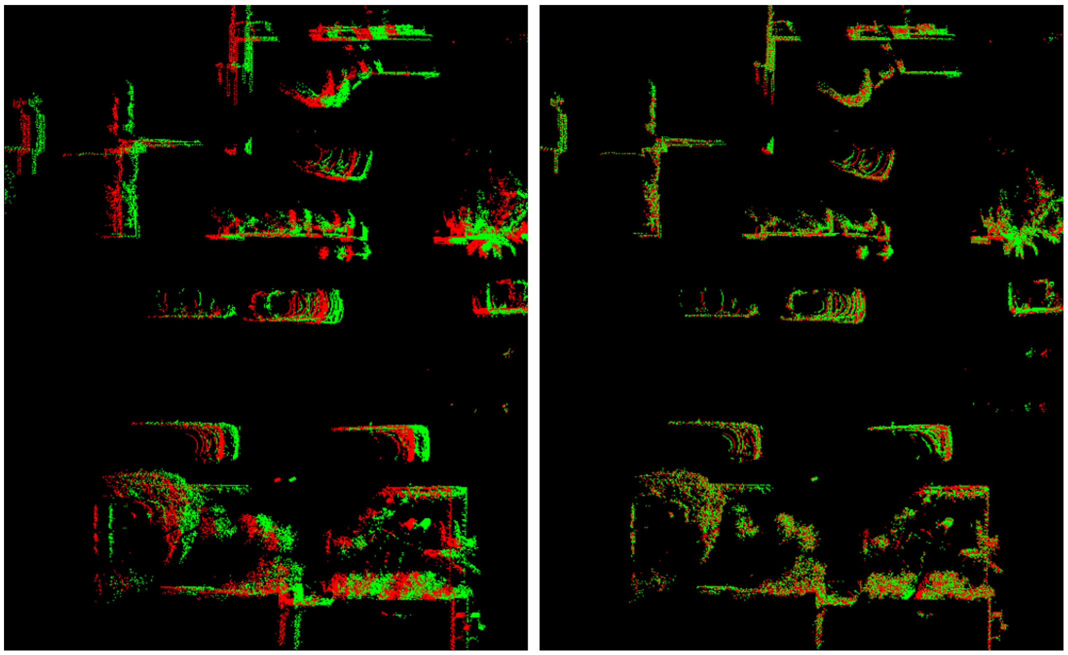


Figure 7.16: Result of ICP Algorithm

The developed platform is driven in a series of maneuvers while recording LIDAR and navigation data for performing the calibration. Figure 7.17 shows the navigation system position of the vehicle in blue and the same data extracted from the LIDAR point clouds using the implemented ICP algorithm. The vehicle is driven very slowly to remove motion effects. It can be seen that the odometry from the ICP algorithm is not as accurate as that from the navigation system.

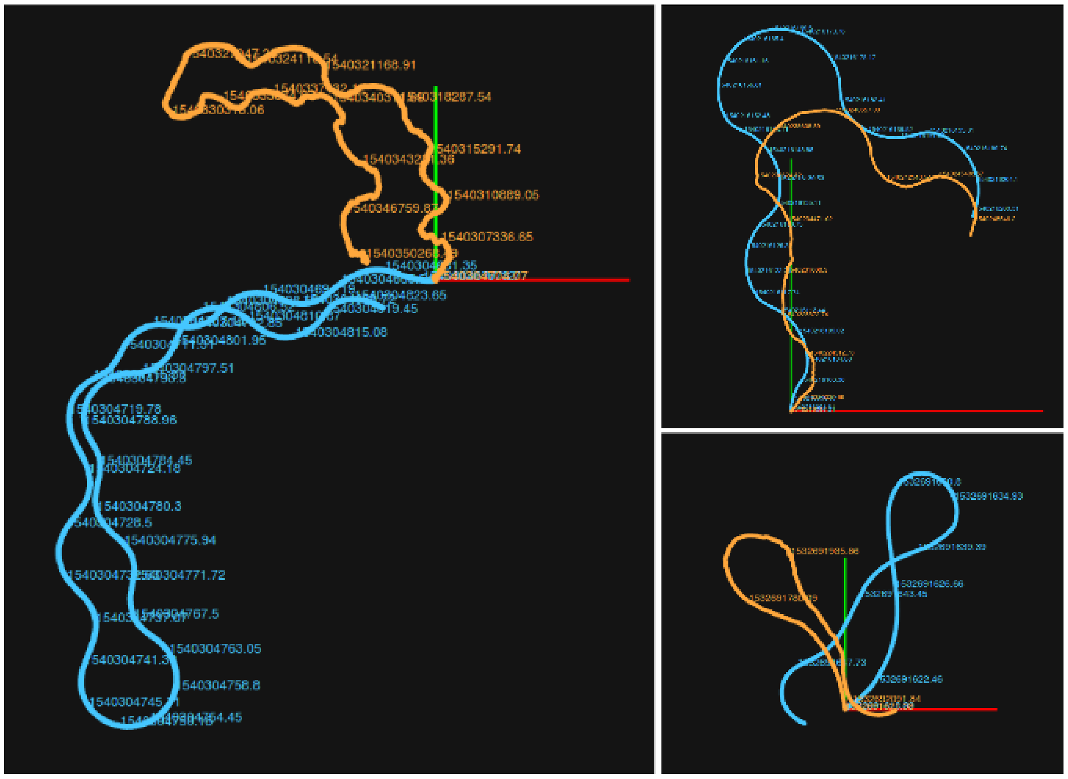


Figure 7.17: GPS (blue) and LIDAR Odometry (orange) for Various Calibration Maneuvers

The ICP algorithm is subjective to measurement errors from the LIDAR sensor. Since each point can have error in its position, the overall result of the ICP algorithm can never be perfect. Furthermore, ICP matching becomes worse when there is significant motion between the two frames or when the position of the LIDAR changes too much between frames. Both of these scenarios result in points that do not correspond to points in the second frame and can result in false correspondences to be found. Another ICP fails to perform well if there are not many usable feature points in the respective point clouds. Having a lot of infrastructure an buildings in the surroundings tended to result in more accurate point cloud matching.

Moreover, the calibration between the navigation system and LIDAR is also difficult to achieve due to the errors in the navigation system in measuring the accurate position and orientation of the vehicle. The typical error of the navigation system on the vehicle is too high for a usable calibration as mentioned in the previous section.

For these reasons, the calibration attempts between the navigation system and LIDAR sensor were unsuccessful. It is believed that the problems in calibration can be addressed by using a navigation system that provide centimeter level accuracy and using a motion free calibration environment for the LIDAR. A highly accurate and high resolution LIDAR system should be used to provide more and better correspondences for the ICP algorithm. Furthermore, the ICP algorithm itself could be improved to find a more accurate transformation between the LIDAR point clouds.

7.3.4 Conclusions

In conclusion, all of the calibration methods have various sources of errors that result in a less than perfect calibration. The joint calibration method provided decent results that can be used with the system. The infrastructure sensor calibration with the GPS system was successful in simulation and can be implemented with a real world setup when needed. The Hand-Eye calibrations need further work to be successfully completed. Sources of errors are identified and need to be addressed in future attempts.

Chapter 8

Experimental Results

In this section, the developed autonomous vehicle platform and lane keeping methodologies is tested on the University of Waterloo Ring Road. The lane keeping system design using the navigation system by itself as well as the combined vision and navigation system are both tested and compared. The issues encountered during testing are presented and corrected to eventually achieve the goal of autonomous lane keeping.

8.1 Vision Lane Detection Implementation

Many issues are encountered and corrected in the implementation and testing of the monocular lane detection. Firstly, light artifacts on the image due to the sun can create high intensity vertical lines that are lane like. It can be seen in Figure 8.1 that the artifact is brighter than the lane markings and is likely to be falsely detected as the lane causing the vehicle to adjust its position to the right. Also depicted in the figure, a lower quality color camera, converted to gray-scale for the algorithm, was initially being used as it was the only camera available at the time. It allowed for the experience necessary to determine the characteristics that a camera for lane detection should have.

To correct these issues, it is determined that a monochrome camera is necessary to improve the lane detection since monochrome camera sensors are capable of higher sensitivity and detail in an image as compared to their color counterparts. This is because the Color Filter Array (CFA) used to capture a particular color limits the amount of light being captured by each photo-site in the sensor by approximately a third. As a result, a new camera was purchased with an accompanying lens that better fit to the size of the lane. Moreover,

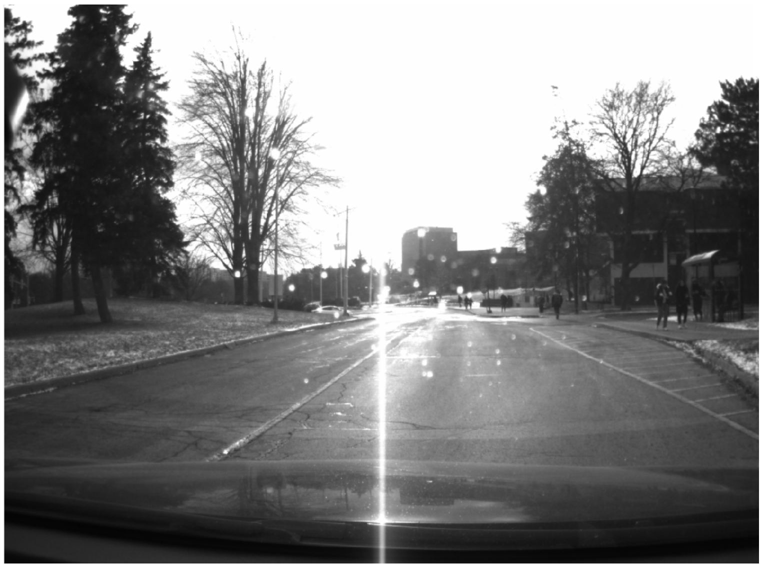


Figure 8.1: Light Artifacts on Image Due to Location of the Sun

the new camera allowed for higher number of pixels for the same amount of distance in the world allowing for objects and lanes in the background to be of higher quality. For correcting the issue with the light artifacts, a piece of tape is attached to the top half of the lens effectively blocking out much of the light captured for unused parts of the image. This coupled with a dynamic brightness adjustment algorithm available from the camera firmware allows the camera to automatically adjust the brightness of the image so that the average pixel intensity is held constant. This allows the camera to automatically adjust exposure settings when the vehicle enters a dark shadowy area so that the lane is always able to be detected. With these changes the problem of the light artifacts is solved and the overall quality and depth of information in the image is significantly improved as shown in Figure 8.2.

8.1.1 Dynamic Pitch Correction Design

The real world test road has extremely uneven surfaces in various sections caused by bumps, cracks and potholes. These uneven surfaces incur pitch motion on the vehicle and camera. If this pitch motion is not accounted for the lane marking position in the image frames will shift laterally resulting in erroneous error metric measurements. Therefore, the camera pitch angle must become a dynamic input in the computation of the birds eye view image.

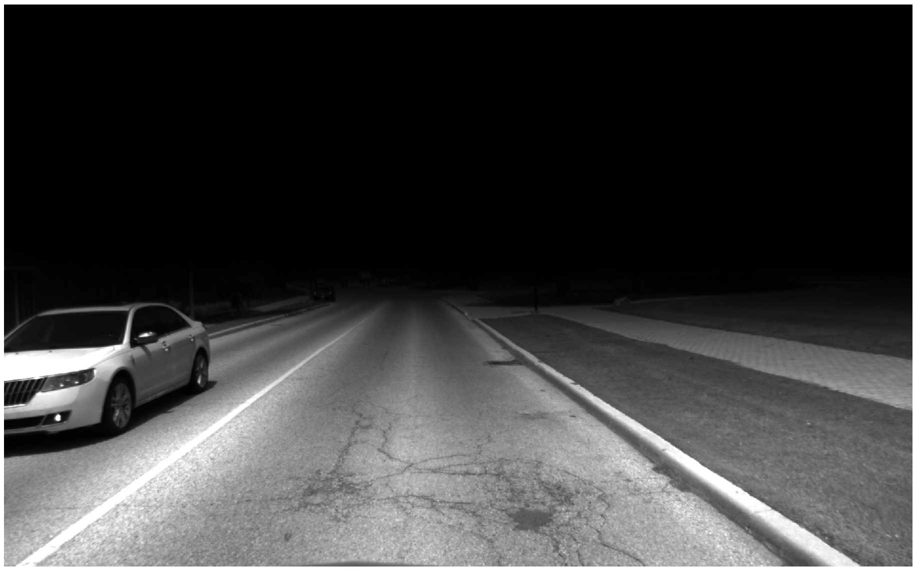


Figure 8.2: Improvement of Image with Tape Blocking Undesired Light Artifacts

A navigation solution which provides real time pitch angle of the vehicle may be utilized for dynamic pitch correction. If such a system is not accessible, estimation techniques can be used. In [59], the authors estimate vehicle pitch by characterizing the distortion present in the birds eye view lane markings. With the underlying assumption that lane markings in birds eye view are parallel, the width of the lane at two different distances from the vehicle is sufficient to geometrically characterize the distortion.

Assuming the pitch angle of the vehicle is available, the high frequency content needs to be extracted and the birds eye view adjusted. The low frequency change of pitch of the road should not be corrected for since these do not incur changes in vehicle pitch with respect to the road.

The transfer function of the simple first order high pass filter used is given in Equation 8.1 where $a = \frac{T}{\tau}$, T is sampling time, and τ is the filter time constant. The filter time constant is determined by inspection of the Fast Fourier Transform (FFT) of the pitch angles coming from the navigation solution while driving on a section of the test road. The filter is applied to a window of data up to 500 frames. Therefore, it is expected that the filter will not perform well when there is not enough data present but will improve as time goes on and the window size is achieved.

$$H(z) = \frac{1 + a + (a-1)\cdot z^{-1}}{1 + (a-1)\cdot z^{-1}} \tag{8.1}$$

8.1.2 Pitch Correction Results

In the simulated scenario, the road conditions were perfect. There were no irregularities, bumps, pot holes, etc. In the real world driving on Ring Road, these imperfect conditions incur undesired pitch motion on the vehicle that bias the vision system data and should be corrected. Figure 8.3 shows the results of the pitch correction algorithm on a section of Ring Road. A high pass filter is used to extract the high frequency pitch since the low frequency pitch should not be corrected. It can be observed that the high pass filter works well especially after the first fifty frames where the window size is too small to accurately extract the high frequency pitch. Moreover, it can be seen that the high frequency content has an average value around 0 degrees as the number of frames increases further showing the effectiveness of the filter for this type of data. Furthermore, the low frequency content approaches the average value of the measured pitch as the window size is being reached.

The lateral error measured by the vision system with and without pitch correction is shown for the same segment of Ring Road. It can be seen that with the pitch correction the magnitude of the error is slightly improved. However, this is not the level of improvement desired from the pitch correction. Significant effect of the pitch motion incurred on the vehicle and camera can still be observed since the road lane marking is smooth and continuous, yet the lateral error measured is noisy. It is believed that the reason for this is the slow frame rate of the pitch correction and vision processing pipeline. The processing cannot be accomplished at the necessary frame rate to address the full effect of the incurred pitch motion.

8.1.3 Lane Detection Improvements

Even with the dynamic pitch correction, the lane marking is not detected perfectly in every single image frame. Commonly occurring issues include false detections where cracks in the road or the curb may be detected as the lane marking as shown in Figure 8.4 and missed detections where the detection algorithm fails to find a lane marking in the frame. In order to mitigate these issues, adjustments are made to the lane detection algorithm. Overall, these additions improve the lane detection results shown in Section 8.2.

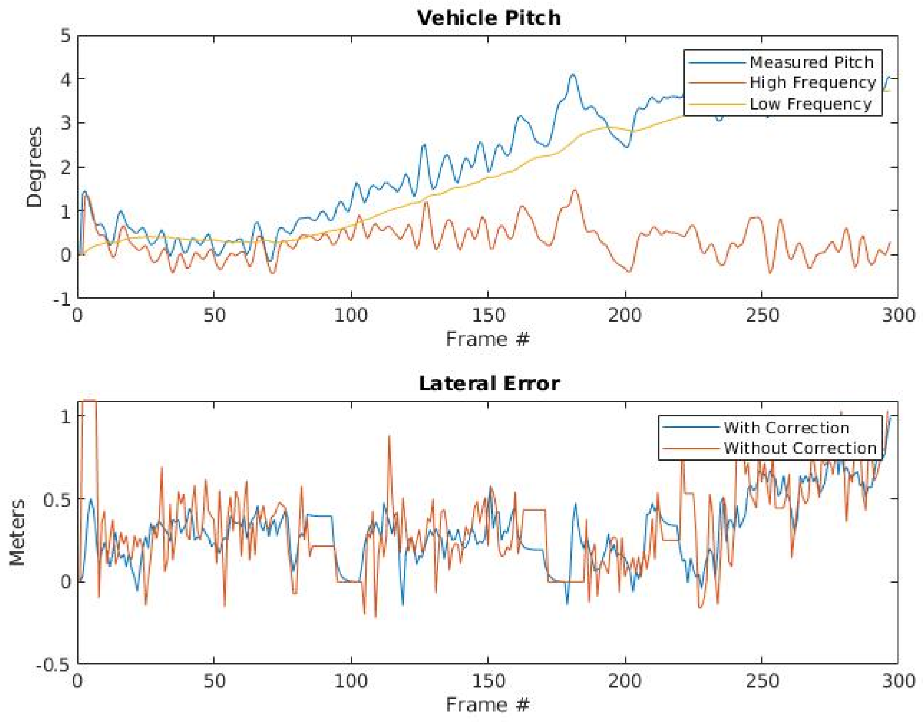


Figure 8.3: Lateral Error Noise Improvement Via Pitch Correction

Firstly, instead of transforming a large field of view to the birds eye image, a smaller localized area corresponding the the expected position of the left lane marking should be utilized. Not only does this reduce the processing time of the algorithm it also reduces the chance of false detections as the curbs and some of the cracks are removed prior to the detection stage as shown in Figure 8.5. Since the design of the control law seeks to keep the vehicle at a fixed offset from the left lane marking, the localized birds eye view area can simply be defined as a rectangle that is offset from the camera coordinate frame by a fixed, predetermined, amount.

False detections can be further mitigated by defining some criteria that have to be fulfilled in order for a detection to be considered valid. Road lane markings have a maximum allowed curvature that is regulated. Therefore, one check for a valid lane marking is to ensure that the curvature of the detection is under this maximum threshold. Next, markings of very short length should be invalid as they are likely to be false detections of cracks. Ring Road lane markings are solid and continuous unless there is an intersection.

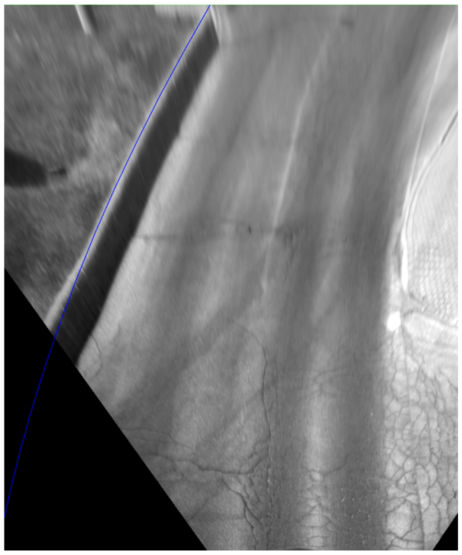


Figure 8.4: Detection of Curb Instead of Lane Marking in Birds Eye View Image

Furthermore, the recent detection history may be used to further validate detections. It is expected that lane markings change smoothly and tend to be near the location where they were in the previous frame. Therefore, validation criteria is set on the curvature difference and the mean distance from the previous detection.

Ring Road lane markings are faded and deteriorating resulting in missed detections. For a short duration, it is expected that the lane marking follows the characteristics of the short term history of detections. Therefore, for a fixed number of frames, missed detections can be replaced by the last valid detection.

8.2 Lane Keeping Results

At this stage of the development of the platform, the goal of the lane keeping system is to only control the vehicle steering angle to keep the vehicle within the lane. The speed of the vehicle is not controlled. Furthermore, dynamic path planning and routing is not currently employed and it is expected that the lane is always clear. A stopped or parked vehicle in the lane will require the test to be stopped as no obstacle avoidance logic is currently present on the vehicle.

The test paths are pre-driven in order to generate the required navigation map with

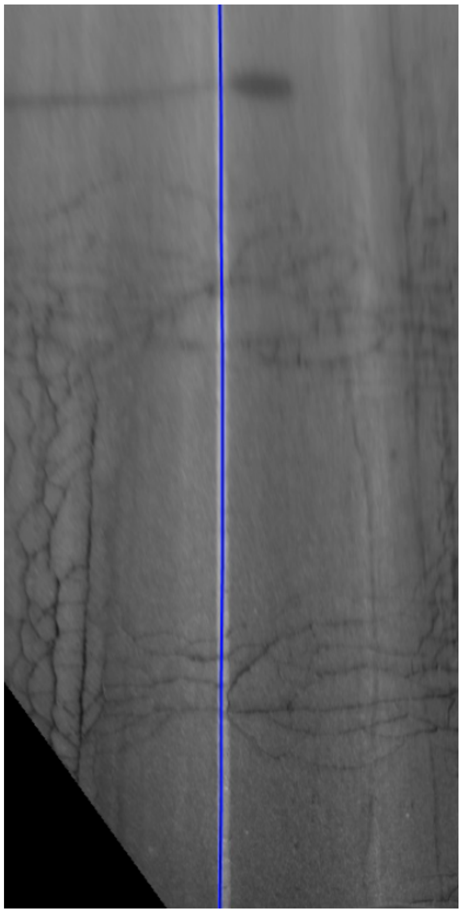


Figure 8.5: Improved Detection and Algorithm Speed by Limiting the Birds Eye View Processed Area

the vehicle driving in the position which is desired from the autonomous lane keeping. The system is tested in simulation prior to real world testing around Ring Road. Parameters such as controller coefficients are tuned through trial and error and the simulation parameters are used as a basis for the real world parameters.

It is important to note that the camera is mounted as close to the navigation system antenna as possible as shown in Figure 8.6. This antenna is the reference point for the navigation system measurements. Therefore, the vehicle is considered to be a single point at this location and both error metrics are measured from the same point. Furthermore, the map used as part of the navigation system solution should be collected by driving the vehicle at the same fixed offset from the left lane marking. This will ensure that the navigation and vision solutions are not trying to control the vehicle to two different points.

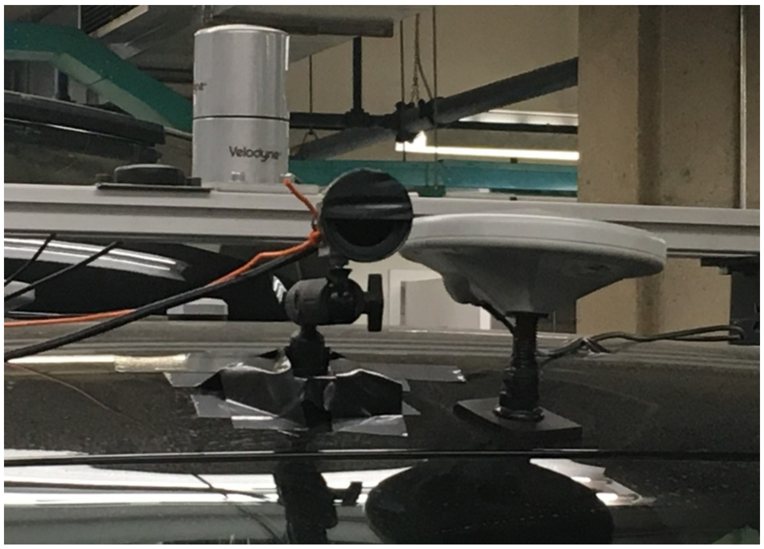


Figure 8.6: Camera Mount and GPS Antenna Location

8.2.1 Ring Road Experiments

The Ring Road experiment consisted of driving around the full road with the lane keeping system using the navigation solution only followed by using the combined vision and navigation solution. A lab technician applied the acceleration and braking inputs to the vehicle and stopped the lane keeping if the path was blocked by a stopped vehicle. Both tests successfully completed the majority of the Ring Road, but had to be stopped on one small stretch due to very busy conditions and vehicles blocking the planned path. Regardless, the majority of Ring Road was traversed with autonomous lane keeping and a comparison of the two systems could be made.

The gains used in the real world controller were $kp_1 = 1$, $kp_2 = 0.2$, and $kp_3 = 1.2$, for both control schemes. Figure 8.7 displays the results. The lateral error from the path is shown vs distance travelled and the data is aligned so that the distances for both tests correspond to the same location on the Ring Road. It can be seen that both tests are very similar and the maximum error magnitude is 20 cm for the navigation only solution and 25 cm for the combined solution. It was observed that increasing the lateral error gain decreased the smoothness of the lane keeping and more oscillations took place around the planned path. The navigation solution position measurement error covariance was between 35 cm and 50 cm and randomness is added to the lane keeping due to these fluctuations. The measurement error is effected by proximity to buildings, GPS signal strength, number of satellites available, etc. Another source of error is the prior map of the Ring Road. This map was collected with the same navigation solution used in the experiment, but with post processing the positional error was reduce to $\leq$ 7cm. This resulted in a highly accurate map, but still not as accurate as the simulated system.

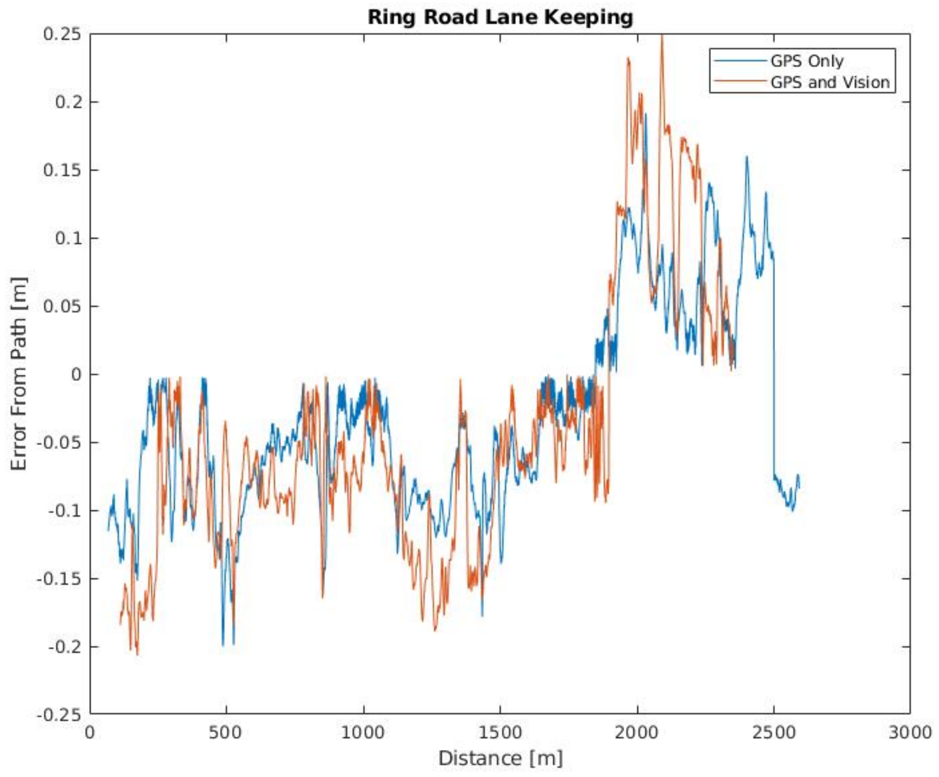


Figure 8.7: Lane Keeping Results on Ring Road of GPS and IMU System Only and Combined System

In the end, both real world Ring Road experiments were successful. It was shown that the navigation only solution allowed the vehicle to autonomously lane keep a 2.5 km section of Ring Road. Furthermore, the combined vision and navigation solution system also traversed a similar distance autonomously. The overall lateral error was under 25 cm and the vehicle followed very closely to the planned map.

8.2.2 Conclusions

In conclusion the lane keeping solutions implemented were both successful in simulation and the real world tests. The system relies on the navigation solution as a base lane keeping system and uses the vision system to provide greater accuracy in lane keeping. In the real world, pitch motion incurred by road irregularities reduce the effectiveness of the vision system. However, the system follows the lane effectively and the deviation from the lane center is relatively small.

These results indicate that the approach used for lane keeping allows for the use of less accurate on board GPS system. The GPS is only being used for localization on the pre defined map which in turn is only being used to calculate the heading angle error of the vehicle based on a point some distance in front of the vehicle. Even if there is a significant localization error, the heading calculated using a point far from the vehicle will still guide the vehicle in the correct direction in order to follow the lane. Moreover, using the lateral error metric from the vision system helps to alleviate accuracy requirements from the GPS by providing fine adjustment of the vehicle position within in the lane.

Improvements to the system could be made by sensor fusion of the vision and navigation sensors using a variant of the Kalman filter. Such a system would be robust to GPS or vision system dropouts and would better estimate the error metrics using both systems. This would utilize more information coming from each independent solution whereas the current system only uses the heading error metric from the navigation solution and the lateral error metric from the vision solution.

Chapter 9

Conclusions

An autonomous vehicle development platform was created and outlined in this thesis. The software and hardware components as well as the interfacing were developed. These systems were tested through the implementation of various calibration algorithms as well as the lane keeping algorithm. Successfully completed tests validate the proper functionality and usefulness of the developed platform for use by other students in testing their work. The goal of time synchronization between all the devices in the platform is well underway with the LIDAR and Linux laptop being synchronized from the GPS navigation solution. Moreover, interfacing with sensors and actuators was completed with all the necessary interfaces set up for future use.

All the necessary calibrations to fully resolve the different coordinate frames of the system were performed and tested. The joint calibration method has proved effective in simultaneously calibrating many sensors together. The effectiveness was proved when the Radar, Mobileye, and Lidar sensors were calibrated with an error under 0.5 meters. The same technique was employed in the synchronization of infrastructure sensors with a vehicle mounted GPS. The effectiveness of this calibration method was shown in simulation. The hand-eye calibration technique was employed to calibrate the GPS navigation system with the LIDAR and camera sensors. All aspects of the calibration were implemented, but limitations in hardware or errors arising in the real world did not allow for the calibrations to be useful.

Finally, the full platform was tested with the goal of achieving autonomous lane keeping around Ring Road. A lane keeping controller was employed based off of the Stanley controller. Error metrics were defined in order to extract useful information coming from the GPS and camera. The error metrics were used in conjunction with the control law to autonomously steer the vehicle in real time. The lane keeping algorithm was employed successfully in simulation and the developed platform.

9.1 Future Work

The autonomous platform was developed quickly, but effectively, and is capable to test many different algorithms as it stands. However, various things can be done to improve the platform. Firstly, the platform relies heavily on ROS for interfacing with sensors as well as visualization and data logging tasks. ROS itself has some flaws that are not addressed in this thesis, but should be addressed in future development of the platform. Firstly, ROS only has one master node handling the other nodes. Failure of the master node is not considered and should be handled since this can seriously hinder all software processes running on the platform. Moreover, ROS needs a monitor to check whether all the nodes are functioning properly and to fix any issues it detects. Next, there are some performance improvements that can be made to the platform. ROS itself has some performance degradation when broadcasting message, but when ROS and Matlab are used together, the data transfer between them can take significant time and cap the achievable frame rate of the software.

Improvements to the hand-eye calibration algorithms are necessary. As mentioned in the thesis, larger calibration targets, a RTK GPS system, as well as a LIDAR with more laser channels can benefit these calibrations. Moreover, the implementation of the ICP algorithm used in the thesis could be improved and tested better.

The lane keeping algorithm can also be improved in the future. Firstly, the pitch correction results were not great and the algorithm could benefit if the entire system was working at a higher frame rate. Moreover, lane tracking methods are not currently employed in the lane keeping system. These methods would also reduce the noise injected by incurred pitch motion by using the history of the lane markings to estimate the future markings.

The work on the lane keeping system leads to autonomous acceleration and deceleration as the natural next step. Since a map of the Ring Road is available, a quick implementation can use hard coded locations where the vehicle should stop for a stop sign or traffic light. The detection of signs can be made dynamic in the future. Moreover, object detection is a well researched problem and existing implementations can be used in conjunction with a basic cruise control system to autonomously control the speed while lane keeping. Furthermore, another natural extension of this work is to replace the highly accurate GPS system utilized for lane keeping with a less accurate and more affordable system. The lane keeping system design accommodates for this change, and it simply needs to be tested and GPS accuracy requirements need to be characterized. This would demonstrate the value of the lane keeping design utilizing an affordable and minimal sensor set.

References

[1] T. Canada, "Canadian motor vehicle traffic collision statistics: 2016," Apr 2018. Online, Accessed: 2019-07-22.

[2] D. J. Fagnant and K. Kockelman, "Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations," *Transportation Research Part A: Policy and Practice*, vol. 77, pp. 167–181, 2015.

[3] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, *et al.*, "Stanley: The robot that won the darpa grand challenge," *Journal of field Robotics*, vol. 23, no. 9, pp. 661–692, 2006.

[4] C. Urmson, J. Anhalt, D. Bagnell, C. Baker, R. Bittner, M. Clark, J. Dolan, D. Duggins, T. Galatali, C. Geyer, *et al.*, "Autonomous driving in urban environments: Boss and the urban challenge," *Journal of Field Robotics*, vol. 25, no. 8, pp. 425–466, 2008.

[5] S. Kammel, J. Ziegler, B. Pitzer, M. Werling, T. Gindele, D. Jagzent, J. Schröder, M. Thuy, M. Goebl, F. v. Hundelshausen, *et al.*, "Team annieway's autonomous system for the 2007 darpa urban challenge," *Journal of Field Robotics*, vol. 25, no. 9, pp. 615–639, 2008.

[6] J. Leonard, D. Barrett, J. How, S. Teller, M. Antone, S. Campbell, A. Epstein, G. Fiore, L. Fletcher, E. Frazzoli, *et al.*, "Team mit urban challenge technical report," 2007.

[7] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene under-

standing," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

[9] X. Huang, X. Cheng, Q. Geng, B. Cao, D. Zhou, P. Wang, Y. Lin, and R. Yang, "The apolloscape dataset for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 954–960, 2018.

[10] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *2011 IEEE Intelligent Vehicles Symposium (IV)*, pp. 163–168, IEEE, 2011.

[11] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi, "Towards a viable autonomous driving research platform," in *2013 IEEE Intelligent Vehicles Symposium (IV)*, pp. 763–770, IEEE, 2013.

[12] P. Grisleri and I. Fedriga, "The braive autonomous ground vehicle platform," *IFAC Proceedings Volumes*, vol. 43, no. 16, pp. 497–502, 2010.

[13] J. Ziegler, P. Bender, M. Schreiber, H. Lategahn, T. Strauss, C. Stiller, T. Dang, U. Franke, N. Appenrodt, C. G. Keller, *et al.*, "Making bertha drivean autonomous journey on a historic route," *IEEE Intelligent transportation systems magazine*, vol. 6, no. 2, pp. 8–20, 2014.

[14] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," in *2010 IEEE International Conference on Robotics and Automation*, pp. 4372–4378, IEEE, 2010.

[15] S. Liu, J. Tang, Z. Zhang, and J.-L. Gaudiot, "Computer architectures for autonomous driving," *Computer*, vol. 50, no. 8, pp. 18–25, 2017.

[16] K. Belcarz, T. Białek, M. Komorkiewicz, and P. Żołnierczyk, "Developing autonomous vehicle research platform–a case study," in *IOP Conference Series: Materials Science and Engineering*, vol. 421, p. 022002, IOP Publishing, 2018.

[17] A. Broggi, S. Debattisti, P. Grisleri, and M. Panciroli, "The deeva autonomous vehicle platform," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 692–699, IEEE, 2015.

[18] Z. Zhang, "A flexible new technique for camera calibration," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, 2000.

[19] Q. Zhang and R. Pless, "Extrinsic calibration of a camera and laser range finder (improves camera calibration)," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566)*, vol. 3, pp. 2301–2306, IEEE, 2004.

[20] H. Yamazoe, A. Utsumi, and S. Abe, "Multiple camera calibration with bundled optimization using silhouette geometry constraints," in *18th International Conference on Pattern Recognition (ICPR'06)*, vol. 3, pp. 960–963, IEEE, 2006.

[21] Q. V. Le and A. Y. Ng, "Joint calibration of multiple sensors," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3651–3658, IEEE, 2009.

[22] R. Y. Tsai and R. K. Lenz, "Real time versatile robotics hand/eye calibration using 3d machine vision," in *Proceedings. 1988 IEEE International Conference on Robotics and Automation*, pp. 554–561, IEEE, 1988.

[23] R. Y. Tsai and R. K. Lenz, "A new technique for fully autonomous and efficient 3d robotics hand/eye calibration," *IEEE Transactions on robotics and automation*, vol. 5, no. 3, pp. 345–358, 1989.

[24] R. Horaud and F. Dornaika, "Hand-eye calibration," *The international journal of robotics research*, vol. 14, no. 3, pp. 195–210, 1995.

[25] F. Dornaika and R. Horaud, "Simultaneous robot-world and hand-eye calibration," *IEEE transactions on Robotics and Automation*, vol. 14, no. 4, pp. 617–622, 1998.

[26] M. Vel'as, M. Španěl, Z. Materna, and A. Herout, "Calibration of rgb camera with velodyne lidar," 2014.

[27] Y. Park, S. Yun, C. Won, K. Cho, K. Um, and S. Sim, "Calibration between color camera and 3d lidar instruments with a polygonal planar board," *Sensors*, vol. 14, no. 3, pp. 5333–5353, 2014.

[28] G. Pandey, J. R. McBride, S. Savarese, and R. M. Eustice, "Automatic extrinsic calibration of vision and lidar by maximizing mutual information," *Journal of Field Robotics*, vol. 32, no. 5, pp. 696–722, 2015.

[29] J. Levinson and S. Thrun, "Automatic online calibration of cameras and lasers.," in *Robotics: Science and Systems*, vol. 2, 2013.

[30] A. B. Hillel, R. Lerner, D. Levi, and G. Raz, "Recent progress in road and lane detection: a survey," *Machine vision and applications*, vol. 25, no. 3, pp. 727–745, 2014.

[31] B. Yu and A. K. Jain, "Lane boundary detection using a multiresolution hough transform," in *Proceedings of International Conference on Image Processing*, vol. 2, pp. 748–751, IEEE, 1997.

[32] M. Aly, "Real time detection of lane markers in urban streets," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 7–12, IEEE, 2008.

[33] Y. Wang, D. Shen, and E. K. Teoh, "Lane detection using spline model," *Pattern Recognition Letters*, vol. 21, no. 8, pp. 677–689, 2000.

[34] Y. Wang, E. K. Teoh, and D. Shen, "Lane detection and tracking using b-snake," *Image and Vision computing*, vol. 22, no. 4, pp. 269–280, 2004.

[35] K. Kluge and S. Lakshmanan, "A deformable-template approach to lane detection," in *Proceedings of the Intelligent Vehicles' 95. Symposium*, pp. 54–59, IEEE, 1995.

[36] J. Li, X. Mei, D. Prokhorov, and D. Tao, "Deep neural network for structural prediction and lane detection in traffic scene," *IEEE transactions on neural networks and learning systems*, vol. 28, no. 3, pp. 690–703, 2016.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[38] T. Ogawa and K. Takagi, "Lane recognition using on-vehicle lidar," in *2006 IEEE Intelligent Vehicles Symposium*, pp. 540–545, IEEE, 2006.

[39] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on control systems technology*, vol. 15, no. 3, pp. 566–580, 2007.

[40] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat, "Mpc-based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, vol. 3, no. 2, pp. 265–291, 2005.

[41] F. Van Diggelen and P. Enge, "The worlds first gps mooc and worldwide laboratory using smartphones," in *Proceedings of the 28th international technical meeting of the satellite division of the institute of navigation (ION GNSS+ 2015)*, pp. 361–369, 2015.

[42] H. Landau, X. Chen, S. Klose, R. Leandro, and U. Vollath, “Trimble's rtk and dgps solutions in comparison with precise point positioning,” in *Observing our Changing Earth* (M. G. Sideris, ed.), (Berlin, Heidelberg), pp. 709–718, Springer Berlin Heidelberg, 2009.

[43] E. D. Kaplan and C. J. Hegarty, “Understanding gps: principles and applications second edition,” in *Artech House*, 2006.

[44] J. G. Manchuk and C. Deutsch, “Conversion of latitude and longitude to utm coordinates,” *Paper 410, CCG Annual Report*, vol. 11, 2009.

[45] B. Moran, “Mathematics of radar,” in *Twentieth Century Harmonic AnalysisA Celebration*, pp. 295–328, Springer, 2001.

[46] B. Schwarz, “Lidar: Mapping the world in 3d,” *Nature Photonics*, vol. 4, no. 7, p. 429, 2010.

[47] J. Kay, “Introduction to homogeneous transformations & robot kinematics,” *Rowan University Computer Science Department*, 2005.

[48] K. Madsen, H. B. Nielsen, and O. Tingleff, “Methods for non-linear least squares problems,” 1999.

[49] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611, pp. 586–606, International Society for Optics and Photonics, 1992.

[50] A. M. Muad, A. Hussain, S. A. Samad, M. M. Mustaffa, and B. Y. Majlis, “Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system,” in *2004 IEEE Region 10 Conference TENCON 2004.*, pp. 207–210, IEEE, 2004.

[51] M. Nieto, J. A. Laborda, and L. Salgado, “Road environment modeling using robust perspective analysis and recursive bayesian segmentation,” *Machine Vision and Applications*, vol. 22, no. 6, pp. 927–945, 2011.

[52] C. B. Barber, D. P. Dobkin, D. P. Dobkin, and H. Huhdanpaa, “The quickhull algorithm for convex hulls,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 22, no. 4, pp. 469–483, 1996.

[53] P. Leroux, “Exactly when do you need an rtos?,” *QNX Software Systems*, 2009.

[54] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of vxworks, linux, rtai, and xenomai in a hard real-time application," *IEEE Transactions on Nuclear Science*, vol. 55, no. 1, pp. 435–439, 2008.

[55] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[56] S. Kato, S. Tokunaga, Y. Maruyama, S. Maeda, M. Hirabayashi, Y. Kitsukawa, A. Monrroy, T. Ando, Y. Fujii, and T. Azumi, "Autoware on board: Enabling autonomous vehicles with embedded systems," in *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, pp. 287–296, IEEE, 2018.

[57] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[58] S. Liu, L. Li, J. Tang, S. Wu, and J.-L. Gaudiot, "Creating autonomous vehicle systems," *Synthesis Lectures on Computer Science*, vol. 6, no. 1, pp. i–186, 2017.

[59] H. Li and F. Nashashibi, *Lane detection (part i): Mono-vision based method.* PhD thesis, INRIA, 2013.