5-2017

Vision-Based Control of a Full-Size Car by Lane Detection

N. Chase Kunz
Utah State University

VISION-BASED CONTROL OF A FULL-SIZE CAR BY LANE DETECTION

by

N. Chase Kunz

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Rajnikant Sharma, Ph.D. Major Professor	Donald Cripps, Ph.D. Committee Member
Xiaojun Qi, Ph.D. Committee Member	Mark R. McLellan, Ph.D. Vice President for Research and Dean of the School of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2017

ABSTRACT

Vision-Based Control of a Full-Size Car by Lane Detection

by

N. Chase Kunz, Master of Science

Utah State University, 2017

Major Professor: Rajnikant Sharma, Ph.D.
Department: Electrical and Computer Engineering

Autonomous driving is an area of increasing investment for researchers and auto manufacturers. Integration has already begun for self-driving cars in urban environments. An essential aspect of navigation in these areas is the ability to sense and follow lane markers. This thesis focuses on the development of a vision-based control platform using lane detection to control a full-sized electric vehicle with only a monocular camera. An open-source, integrated solution is presented for automation of a stock vehicle. Aspects of reverse engineering, system identification, and low-level control of the vehicle are discussed. This work also details methods for lane detection and the design of a non-linear vision-based control strategy.

(93 pages)

PUBLIC ABSTRACT

Vision-Based Control of a Full-Size Car by Lane Detection

N. Chase Kunz

Self-driving cars are an area of increasing investment for researchers and auto manufacturers. Integration has already begun for such vehicles in urban environments. An essential aspect of navigation in these areas is the ability to sense and follow lane markers. This thesis focuses on the development of a self-driving, full-size, electric car which uses only a camera and lane markers to stay on the road. A complete method for automation is shown for a stock vehicle. Discussion includes reverse engineering of the steering, braking, and acceleration signals, as well as methods for lane detection and full-vehicle control.

ACKNOWLEDGMENTS

First, I would like to thank my major professor, Dr. Rajnikant Sharma, for his constant guidance and kind encouragement in times of difficulty. I would also like to thank Dr. Ryan Gerdes for his meticulous attention to detail and willingness to troubleshoot at critical moments. Together, these professors have offered me a wealth of direction in matters relating to the project and invaluable mentorship beyond. I am also grateful for the help from members of my committee and ECE department faculty in automating a vehicle: Dr. Donald Cripps, Dr. Xiaojun Qi, and Dr. Jake Gunther.

Enough thanks cannot be given to my friend and primary research partner, Austin Costley. From beginning to end, he was always willing to dedicate time and energy where it was most needed for the research to progress. The final product would not have been possible without the reverse engineering work of the students and unsung heroes on the EV Automation team: Cameron Sego, Hunter Buxton, Aaron Kunz, Austin Goddard, Tyler Travis, Zach Garrard, Gregory Vernon, Daniel McGarry, David Petrizze, Ishmaal Erekson, and Jonathon Tousley.

For funding and use of the state-of-the-art facility I am grateful for the SELECT group. I would also like to thank the staff at the EVR, especially Ryan Bohm, Josh Rambo, and Paul Rau for providing us the tools and means by which to accomplish our goals.

I am grateful for my friends at the RISC Lab and Information Dynamics Lab: Anusna Chakraborty, Srijanee Biswas, Sohum Misra, Imran Sajjad, Soudeh Dadras, Abhishek Manjunath, Trevor Landeen, Sam Whiting, and Dana Sorensen. They offered wonderful support and comradery in research and coursework.

Finally, I would like to thank my family for a lifetime of encouragement. I am grateful for the sacrifices made by my parents, Nathan and Laura, to send me to school and for the support of my siblings: Christian, Halee, Levi, and Celeste.

N. Chase Kunz

CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

APP	accelerator pedal position
ABS	antilock braking system
BPP	brake pedal position
CAN	controller area network
CMU	Carnegie Mellon University
CRC	cyclic redundancy check
DAC	digital to analog converter
DARPA	Defense Advanced Research Agency
ECU	electronic control unit
EPAS	electric power assisted steering
EVR	electric vehicle roadway
GPIO	general purpose input/output
GPS	global positioning system
IMU	inertial measurement unit
IPM	inverse perspective mapping
MPH	miles per hour
OBD	on-board diagnostics
PCB	printed circuit board
PI	proportional integral
PRBS	pseudorandom binary signal
PSCM	power steering control module
PWM	pulse width modulation
ROS	Robot Operating System
RTK	real time kinematic
SELECT	sustainable electrified transportation
TCM	transmission control module
UART	universal asynchronous receiver/transmitter

USU	Utah State University
UPS	uninterpretable power supply

CHAPTER 1

INTRODUCTION

Control of autonomous vehicles is a topic of ever increasing relevance. Car producers are boosting investment in and research of autonomous capabilities at a rapid rate [1]. The race to bring self-driving cars to the market has begun, with major auto manufacturers promising to mass produce vehicles with full autonomy within the next five years [2]. These technologies give hope to a safer and more efficient transit system, increased mobility for the elderly and disabled, and a more productive commute.

Self-driving cars bring with them a varied set of challenges ranging from the social issues of public acceptance [3] to the technical capabilities of perception, planning, and control [4]. While some of the social aspects can only be address as autonomous vehicles gain traction in the public eye, commercial and academic research can help address barriers in cost and technological capabilities more immediately. For example, as self-driving cars enter public roadways, they must have the ability to interact with their environment. The capability of a vehicle to sense lane markings is a fundamental feature in autonomous driving. Lane detection using computer vision has gained increased attention since the mid 1980s [5] [6] [7]. This method is attractive as cameras are relatively cheap when compared to other sensors such as LIDAR [8]. There is also a demand for navigation in GPS denied environments [9]. Vision-based sensing can continue to operate without reliance on an external signal.

USU in partnership with SELECT [10] has developed a platform for the testing and prototyping of autonomous vehicles. In October 2015 the team received a stock 2013 Ford Focus Electric with the task of automating it with vision input. The project's purpose was to have a vehicle that could navigate a test track without human intervention and to a high degree of accuracy using only a camera. This thesis outlines contributions made to the field of vehicle automation using this platform in the following sections.

1.1 Complete Integration of a Full-size Vision-based Autonomous Car

The modern vehicle is a complex machine and, as such, requires many stages of development for automation. The following sections discuss each stage to automate a regular drive-by-wire vehicle with vision from reverse engineering to vision-based control. While many researchers focus on development of a specific stage, few offer integrated discussions for all stages of vision-based autonomous control as outlined in Chapters 2, 3, and 4 of this thesis.

- *Reverse Engineering and Low-level Control:* The first step in vehicle automation is to gain control of low-level steering, breaking, and acceleration abilities. Many development platforms use external actuators on the steering wheel, brake pedal, and accelerator pedal for this purpose [11]. Another more comfortable option is to make use of a vehicle's drive-by-wire capabilities to command internal actuators to this end [12]. In order to leverage these tools, control signals for each internal actuator must be reverse engineered. Chapter 2 provides the methods to do so for a 2013 Ford Focus Electric.

- *Low-level Controller Design and Model Identification:* A split-controller, proposed by Rajamani [13], separates control capabilities onto different levels. In the case of a vehicle, a guidance controller sends a desired velocity and steering wheel angle to lower-level controls which take care of the commands themselves. These take the desired speed or angle and perform actuation on the vehicle's hardware. This allows for easily swapping out guidance controllers without the need to re-design actuator controls for each new high-level controller. Chapter 3 discusses the necessary model identification and controller design for low-level controllers on a 2013 Ford Focus Electric.

- *Lane Detection:* A variety of methods have been researched and implemented concerning lane detection. Many approaches use edge information for lane markers [14] [15]. These usually involve filtering for lane size. Others users use particle filtering [16]

and pixel clustering [17] [18]. Various lane models have been used in vision based lane detection. Bertozzi et al. make use of polylines [19]. In [20], a hyperbola pair is calculated from lane boundaries using fuzzy logic and RANSAC. A B-Snake is used in [21]. This model is able to describe a wider range of lane structures since B-Splines can form any arbitrary shape by a set of control points. Similarly, a third-degree Bezier spline is used in [14].

A technique used by many is Inverse Perspective Mapping [14] [17] [19]. This approach allows one to remove the perspective effect when the acquisition parameters (camera position, orientation, optics, etc.) are completely known and when some knowledge about the road is given, such as a flat road hypothesis [15]. Lines which converge at the vanishing point (parallel lines on a road) are parallel on the image plane after the image transformation. Chapter 4 shows a method for lane detection using IPM.

- *Vision-based Control:* Vision based vehicle control has been a subject of research for decades. During the 1990s, CMU implemented various lane detection and control systems using their Navlab experimental platform. In 1991 UNSCARF used pattern recognition techniques to navigate unstructured roads with no lane markings [22]. The SCARF system controlled the vehicle through difficult roads and added the ability to recognize intersections without prior knowledge in 1993 [23]. In 1995, the RALPH system used a calculation of road curvature in combination with the vehicle's lateral offset in the lanes to send steering commands [17]. Controlling autonomous vehicles through neural networks has also been a topic of research since at least the same time period [18] [24]. Spurring research in the mid 2000s were challenges put on by the Defense Advanced Research Projects Agency. The DARPA Grand Challenge of 2005 confronted vehicles with off-road conditions [9] while the DARPA Urban Challenge presented contestants with the difficulties of urban environments such as lane markers, traffic signals, and other vehicles [25]. Chapter 4 shows the design of a vision-based controller.

1.2 Method of Acceleration Through CAN Message Injection

As vehicles rely more on digital input and communication for control, new attack surfaces arise through which an attacker may cause unintended vehicle responses. Building on the work of Checkoway et. al. [26] [27], Miller and Valasek depict attacks on a 2010 Ford Escape, 2010 Toyota Prius, and a 2014 Jeep Cherokee [28] [29]. The attacks vary in nature and function, making use of the vehicle's native ECUs and reverse engineering the communication protocols on the CAN bus. While their scope is limited, for example controlling steering under 5 mph, these attacks show the ability to control a vehicle by the understanding and manipulation of its native control signals. This thesis uses similar methods to reverse engineer CAN message commands on the 2013 Ford Focus Electric and cause unintended acceleration. The approach is detailed in Chapter 2.

1.3 Open-source and Low-cost Platform

The developed platform was designed to be cost-effective and open for collaboration and further research. It uses ROS and OpenCV open-source software and all other code is available online. Chapter 5 presents this platform and provides links to its software.

1.4 Outline

This thesis will proceed in the following manner: Chapter 2 discusses and methods of reverse engineering through CAN message injection and sensor emulation. Methods of model identification and the design of low-level control are outlined in Chapter 3. In Chapter 4, a method for lane detection by monocular vision and an accompanying vision-based controller are presented. Chapter 5 highlights the platform for development of a vision-based, autonomous vehicle using open-source software. Results of vision and control algorithms are presented in Chapter 6. Finally, an overview of the thesis and concluding remarks are provided in Chapter 7.

CHAPTER 2

REVERSE ENGINEER COMMUNICATIONS AND ENABLE REMOTE CONTROL

This chapter discusses the methods explored in reverse engineering the low-level communications of the vehicle. It examines data injection through the CAN bus as well as sensor emulation for control of steering, braking, and acceleration.

A team of undergraduate students in the Electrical and Mechanical Engineering programs at Utah State University (including the author of this work) was assembled to explore the 2013 Ford Focus Electric and reverse engineer the communication protocols. The work from this team is detailed in this chapter. The reverse engineering project lasted from November 2015 to May 2016. It is important to note the contributions of the EV Automation team. This chapter was prepared for a coauthored journal submission with Austin Costley [30], the author's research partner. It is included in this thesis for completeness.

2.1 Reverse Engineer Communications and Enable Remote Control

Modern vehicles use a Controller Area Network (CAN) bus system for module-to-module communication [31]. Electronic Control Units (ECU's) are the CAN modules that connect to the bus that send and receive information. A CAN module receives data from sensors, processes the data, and generates the appropriate CAN message to be broadcast on the bus using an analog-to-digital operation.

The research team for the current work used the 2013 Ford Focus Electric Wiring Guide [32] and the Auto Repair Reference Center Research Database from EBSCOhost [33] to understand signal path and critical connections. The wiring guide provided diagrams for most of the wires in the vehicle and included diagrams and pin-outs for the wiring connections. The Auto Repair Reference Center was particularly useful for reverse engineering the CAN protocols. It contains the CAN messages generated and received by each module, diagrams of the four CAN buses in the vehicle, and the module layout on each bus. The

CAN message information was an incomplete list of general messages sent between CAN modules. For example, the list would indicate that a message about the acceleration pedal position is sent from one module to another, but it would not indicate the structure of the message, the arbitration ID, or a conversion to useful units.

Using these resources, the team identified sensors and modules that could be used for vehicle control. Table 2.1 summarizes these findings. In addition, the team took a hands-on approach to vehicle exploration and verified the location and connections of these components.

The examination of this architecture led to the identification of the two possible controller insertion strategies, as shown in Fig. 2.1 . First, the CAN lines between the control module and the CAN bus could be cut, and a controller could be inserted to intercept and change messages being sent from the target control module. Second, the analog signal wires from the target sensor could be cut, and the controller could be inserted between the sensor and the control module. In either strategy, the controller would insert spurious data into the system to control the vehicle. The details and results of these two approaches are discussed in the following subsections.

In order to successfully implement the first controller insertion strategy and take advantage of the information on the vehicle CAN bus, the Vector CANalyzer [34] system was used for the initial CAN message identification process. This system provides an excellent visual tool for watching CAN messages in real time. The tool displays a table of CAN messages with the rows organized by arbitration ID. The first column of the table indicates the time since the last message with a given arbitration ID was received. The second column lists the arbitration ID, and the third column shows the value for each byte of the CAN message. If a byte is changed when a new message is received, the byte is displayed in bold. Over time the byte fades to a light gray if that value stays the same. This was useful in identifying messages such as the accelerator pedal position, brake pedal position, steering wheel angle, and vehicle speed. The CANalyzer system is a great resource, but it is expensive and closed-source. Cheaper alternatives such as the Peak Systems PCAN [35]

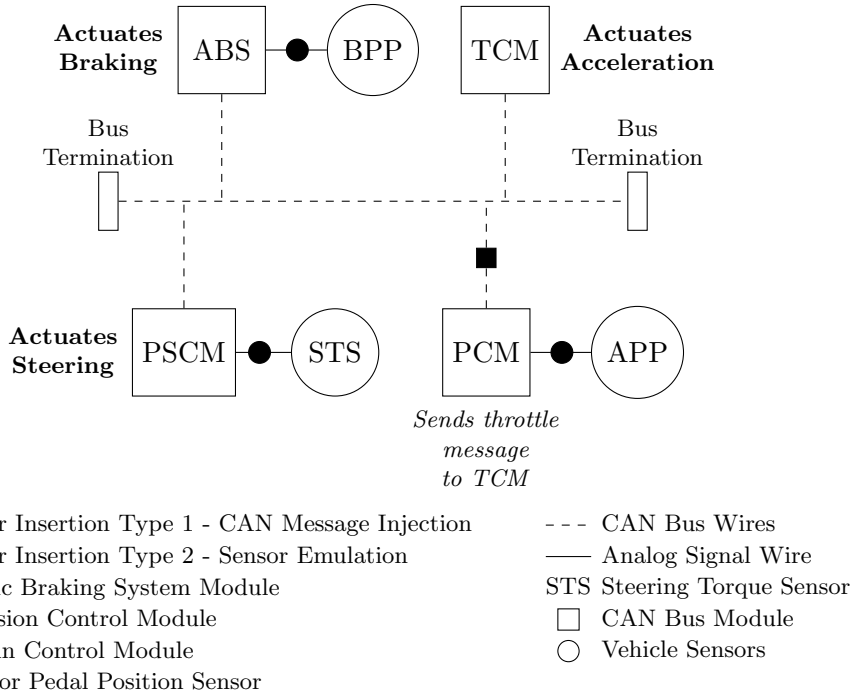

Fig. 2.1: Vehicle CAN bus and sensor architecture. Controller insertion type 1 filters CAN messages and replaces data with the message to be injected, and is represented by a filled black square. Controller insertion type 2 emulates sensor output signals, and are represented by filled black circles. The TCM controls the main drive motor of the electric vehicle and therefore actuates acceleration. The PSCM actuates the power steering motor. The ABS module actuates the hydraulic braking system.

Table 2.1: Sensor and Module Connections for Control Signals

Sensor	CAN Module	CAN Message	CAN Arbitration ID
Accelerator Pedal Position Sensor	Powertrain Control Module (PCM)	Accelerator Pedal Position	0x204
Brake Pedal Position Sensor	Automatic Braking System (ABS)	Brake Pedal Position	0x7D
Steering Torque Sensor	Power Steering Control Module (PSCM)	Steering Torque	Unknown
Steering Wheel Angle Sensor	Steering Angle Sensor Module (SASM)	Steering Wheel Angle	0x10

device can be used that have an open platform for development. An open-source solution for monitoring CAN traffic with the PCAN device is provided with the open-source software accompanying this work. Further discussion on the use of the PCAN device can be found in Section 5.1.1. Another alternative is to use a microcontroller with a CAN bus interface module to monitor and report CAN traffic [36].

Using the resources in the previous paragraphs, it was determined that CAN messages have two main functions: status and control. A status message reports the status of a vehicle component or condition, but does not control that component or condition. For example, a module will receive input from the wheel speed sensors and send the information on the CAN bus. Changing the data in this message will not result in a change of vehicle speed. A control message, however, is sent by a module to control a component or condition of the vehicle. For example, the movement of the wing mirrors is controlled by a CAN signal, when this message is changed the mirrors will move in response.

2.1.1 CAN Message Injection

A platform for injecting CAN messages was developed using the TI TM4C129XL microcontroller evaluation kit [36] and TI CAN transceivers [37]. The platform would connect to the CAN bus using the first controller insertion point, between the target module and the bus. The microcontroller was programmed to record and playback CAN traffic. More specifically, the microcontroller would receive the output from the control module and store it in memory. Upon user request, the output from the control module could be injected on the CAN bus. This platform was used to determine which messages, organized by arbitration ID, were generated by the target control module. Using information from the Auto Repair Center Research Database, it was determined that the Powertrain Control Module (PCM) sent a CAN message to the Transmission Control Module (TCM) regarding the

accelerator pedal position. In addition, the PCM was the only control module that received the sensor signal from the accelerator pedal. For these reasons, the PCM was chosen as the target module for vehicle acceleration.

The connector diagrams in the Wiring Guide were used to determine the CAN bus connections to the PCM. These wires were cut and routed to the CAN injection platform. The CAN messages output from the PCM were recorded and passed through to the CAN bus for an accelerator pedal press. The recorded data was then played back on the bus and the vehicle accelerated as expected. Additional code was added to the playback function to selectively playback messages based on the message arbitration ID. This was used to search the recorded data set and isolate the acceleration control message. The messages were separated into two groups based on their arbitration ID, and each group was played back to the vehicle separately. The group that resulted in vehicle acceleration was separated again into two smaller groups and the process was repeated until one message arbitration ID was left. The acceleration control message was determined to be the message with arbitration ID `0x11A`. Figs. 2.2 and 2.3 show successful CAN injections of the acceleration control message, and the resulting speed for a ramp and step input, respectively. Due to the similarities between the acceleration control message, and a throttle signal in a gas powered car, this message is called the throttle message for the rest of this work.

Another approach to identify useful CAN messages for vehicle acceleration is by correlating the CAN messages with the vehicle speed. This approach was attempted by recording an accelerator pedal press and processing the data in Matlab. The CAN modules for the 2013 Ford Focus EV broadcast messages at prescribed frequencies as opposed to broadcasting in response to another signal. The speed message for the vehicle is broadcast at 100 Hz, which is the highest frequency messages are broadcast for this vehicle. Correlation was performed on messages of the same frequency, and the speed data was downsampled to perform correlation with messages at lower frequencies. The correlation returned a value between -1 and 1 for each byte of every message to indicate how closely correlated that byte was to the speed message. If the byte did not change during the recording, the correlation

returned `NAN`. On the EV-HS CAN bus there are 102 different messages and each message contains 8 bytes. The bytes were sorted based on the absolute value of the correlation value to identify the highest positively or negatively correlated bytes. The bytes that had a `NAN` value were rejected and the final number of bytes being ranked was 341. The highest correlated byte of `0x11A` was byte 4, which had a correlation value of 0.2359 and was ranked 57 out of 341. The low correlation value and rank indicates that the throttle message would not have been identified using this strategy. In contrast, using the approach described in the previous paragraph, the throttle message was identified and was used to control vehicle acceleration.

The investigation of the braking system concluded that a CAN bus message about pedal position would not actuate the hydraulic braking system. The pedal signal is sent directly to the Automatic Braking System (ABS) CAN module, which is the only CAN module on the vehicle that is connected to the hydraulic brake lines. From this it was concluded that braking could not be fully controlled through the CAN bus.

The 2013 Ford Focus EV has an option to include park assist [38]. Though our specific vehicle did not include this option, it was determined that the Electric Power Assisted Steering (EPAS) system had the same part number and motor as the EPAS system in a vehicle with the park assist feature. This meant that the power steering motor would be powerful enough to turn the wheel at low speeds, and by extension, any speed. The Power Steering Control Module (PSCM) receives inputs from the CAN bus and the steering torque sensor located at the base of the steering column. The torque sensor uses a torsion bar to determine the amount of torque being applied by the driver, which is used by the PSCM to determine how much assist the power steering motor should provide. Simply, for a given torque input, less assistance would be provided by the EPAS system at higher speeds. Similar to the brake system, the input of interest is sent from a sensor to the module that performs the desired action. This led to the conclusion that the control of steering would have to be controlled through torque sensor input, and not through the CAN bus. In addition, the work by Miller and Valasek [29] [28] identifies the short comings of exploiting

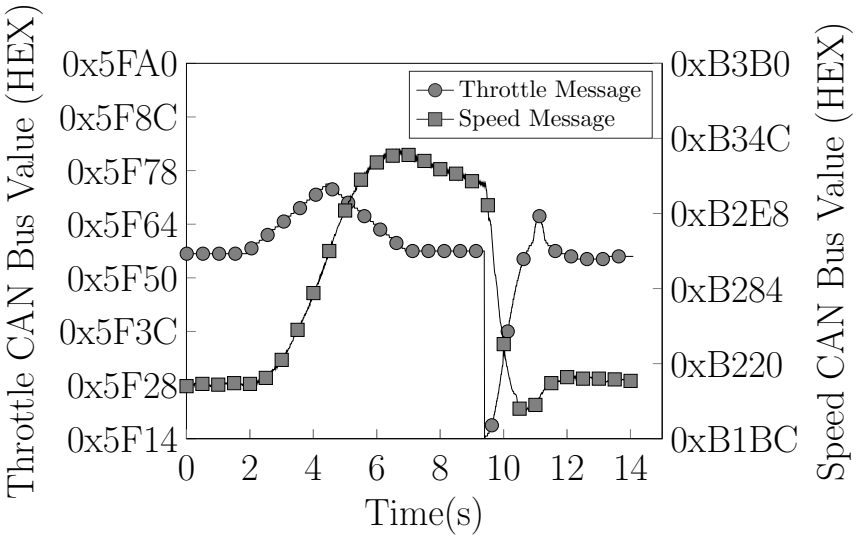

Fig. 2.2: CAN ramp injection from controller insertion point 1 and resulting vehicle speed. Data gathered from CAN bus and represented in hexadecimal format.

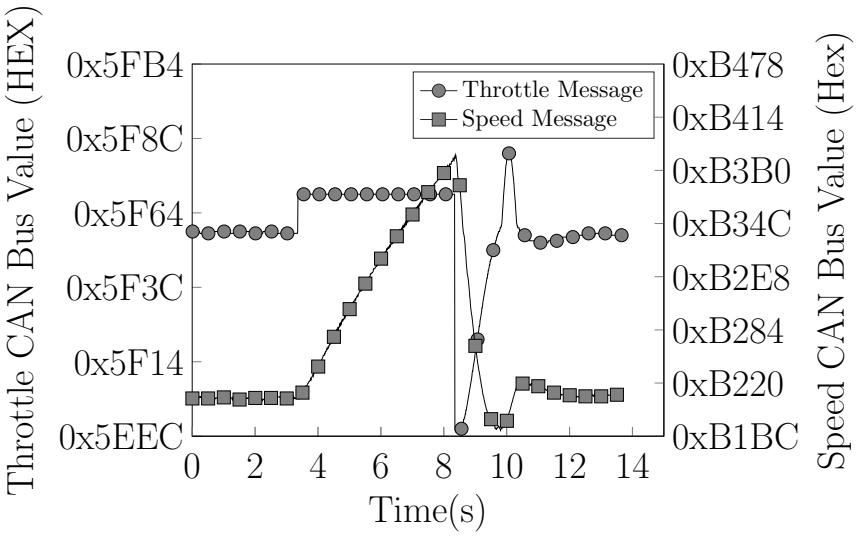

Fig. 2.3: CAN step injection from controller insertion point 1 and resulting vehicle speed. Data gathered from CAN bus and represented in hexadecimal format.

the park assist feature, namely, the park assist feature will cease to control the vehicle if the speed threshold is exceeded.

2.1.2 Sensor Emulation

The CAN bus injection method was unable to control braking and steering, so the sensor emulation method was explored. The accelerator pedal position sensor was analyzed to control vehicle acceleration, the brake pedal position sensor was analyzed to control the braking system, and the steering torque sensor was analyzed to control the vehicle steering. Fig. 2.4 shows these sensors and the following paragraphs discuss the analysis.

The accelerator pedal position (APP) sensor is located at the top of the accelerator pedal. There are six wires connected to the APP sensor, including two 5 V power wires with corresponding ground wires, and two signal wires. The sensor power pins were connected to a voltage source, and the signal wires were connected to an oscilloscope. It was determined that the sensor outputs two DC voltages similar to the output of potentiometers. Fig. 2.6 shows the voltage levels of the two output signals in response to a pedal press. The third signal on the graph is a multiplier that relates the two signals. It is seen that $V_1 \approx 2V_2$.

The brake pedal position (BPP) sensor is located at the top of the brake pedal. There are four wires connected to the BPP, including a 5 V power, ground, and two signal wires. The BPP was connected to the voltage source and oscilloscope in the same manner as the APP. However, the BPP outputs two PWM signals instead of DC voltage levels. When the brake pedal is not pressed the duty cycles of the signals settle at 89% for signal 1 and 11% for signal 2. During a braking event, the duty cycle for signal 1 decreases and the duty cycle for signal 2 increases at the same rate. Fig. 2.7 shows the two PWM signals for the BPP. The frequency of signal 1 is 533 Hz and the frequency of signal 2 is 482 Hz.

The steering torque sensor is located at the base of the steering column. The sensor connects to the Power Steering Control Module (PSCM) on the CAN bus, which determines the amount of power steering assist to provide. The assist is provided by an electric motor connected to the steering rack. In the sensor, a torsion bar is used to connect two parts of the steering shaft, where the rotational displacement can be measured to determine the

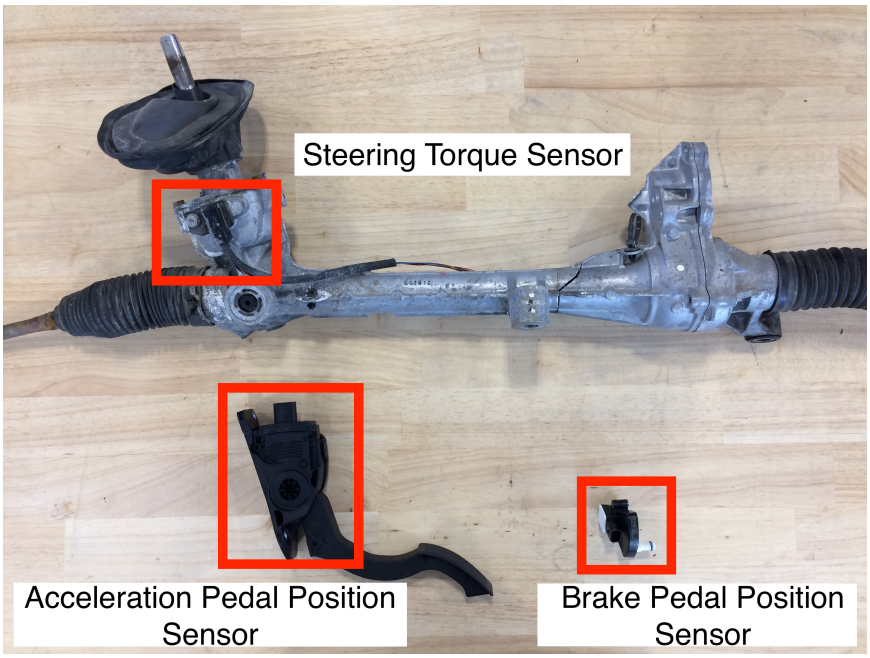

Fig. 2.4: The physical sensors emulated for vehicle control. *Top:* Steering rack for 2013 Ford Focus EV, the steering torque sensor is located at the base of the steering column and measures torque from driver. *Bottom Left:* The APP sensor located at the top of the accelerator pedal. *Bottom Right:* The BPP sensor that is usually mounted behind the brake pedal assembly and measures the brake pedal press.

Fig. 2.5: Aerial view of the Electric Vehicle Roadway and Research Facility (EVR) at Utah State University.

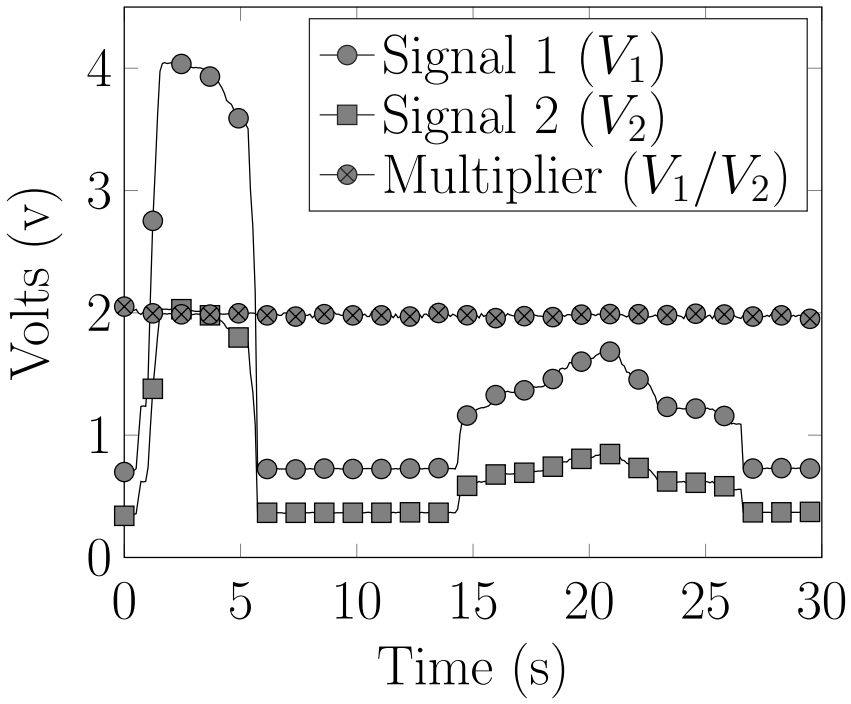

Fig. 2.6: Acceleration pedal position sensor output. Two analog voltage signals related by $V_1 = 2V_2$.

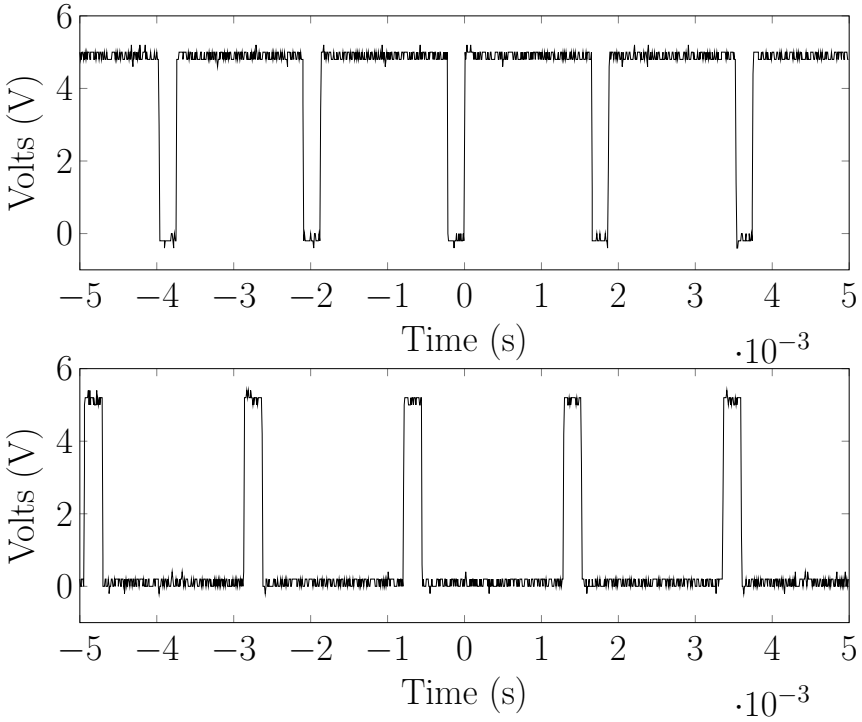

Fig. 2.7: Brake pedal position sensor output signals. Signal 1: 89% resting duty cycle at 533 Hz. Signal 2: 11% resting duty cycle at 482 Hz.

torque input by the driver [39]. Similar to the BPP, there are 4 wires connected to the steering torque sensor, including 5 V power, ground, and two signal wires. The steering torque sensor was connected to the voltage source and oscilloscope, and it was determined that the sensor outputs two PWM signals on the signal wires, where both signals settle at 50% duty cycle when no torque is applied on the steering wheel. Both signals have a frequency of 2.15 kHz. Similar to the brake PWM signals, the duty cycles always add to 100%, and the direction that the steering wheel is being turned determines which signal's duty cycle increases and which signal's duty cycle decreases. Fig. 2.8 shows the two steering PWM signals.

2.1.3 Safety and Security

In 1996, the OBD-II (On-Board Diagnostics) specification was required to be implemented on any new vehicle sold in the United States [40]. This specification gives owners and technicians the ability to diagnose issues on the vehicle. The specification standardized connectors, message formats, and frequencies. The OBD-II port on the 2013 Ford Focus EV connects to the EV-HS CAN bus, which is the same bus that the throttle message is sent from the PCM to the TCM.

An attack platform was developed to inject arbitrary throttle messages through the diagnostics port. This attack method was important because, if successful, it would demonstrate that the acceleration of the vehicle could be controlled with limited intrusion. This differed from the approach in Section 2.1.1, as it does not require access to the target module, or that the CAN wires be cut and re-routed. Instead, this platform could be plugged into the OBD-II port and monitor the bus for the target message arbitration ID. Also, it would show that if an attacker was able to inject messages from any module on the EV-HS CAN bus, then arbitrary vehicle acceleration could be caused. This would stand in contrast to the findings in [28], [29], [27], [26], where the acceleration of the vehicle could only be controlled under specific preconditions, and required intrusive access to the CAN bus.

The platform was connected to the CAN bus through the OBD-II port (other points on the bus could be used, as well) and monitored the traffic on the bus. The user determined a

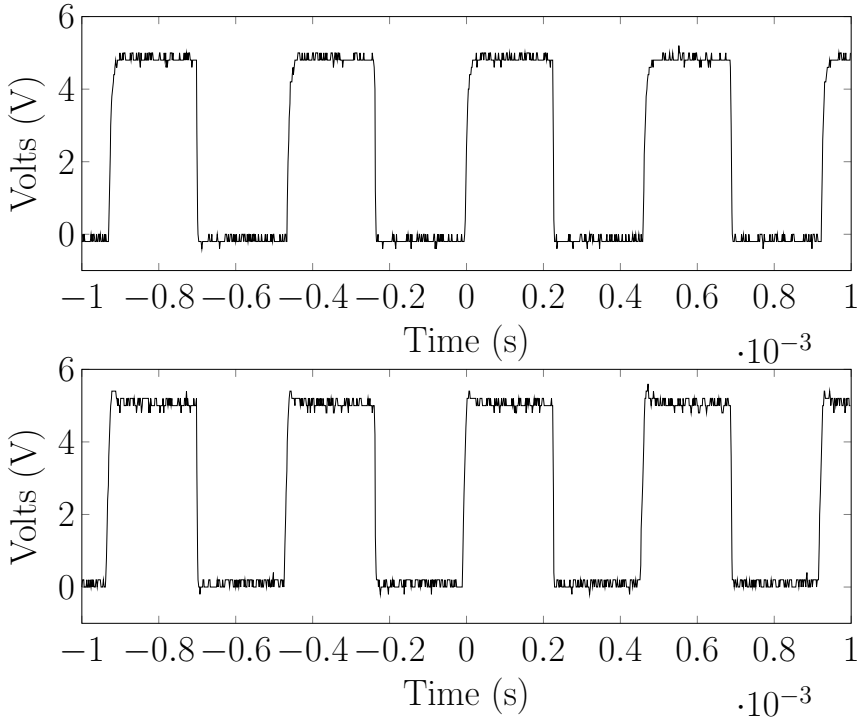

Fig. 2.8: Steering torque sensor output signals. 50% resting duty cycles at 2.15 kHz.

target message, in this case, the throttle message, and provided that message arbitration ID to the system. In Section 2.1.1, it was determined that the throttle message is included in the data frame associated with arbitration ID `0x11A`, and is broadcast at 10 Hz. The platform waited until a message was received with the corresponding arbitration ID, and would replace throttle message data with an arbitrary throttle command value. The platform was designed to only alter the parts of the message that relate to the throttle control. The inserted message would be sent 250 μs after the actual message, leaving 9.75 ms for the inserted message to be received and processed by the TCM. This allowed the inserted message to dominate the period and cause the vehicle to accelerate. Fig. 2.9 shows the successful ramp injection through the OBD-II port and the resulting vehicle speed. Thus confirming the hypothesis that vehicle acceleration can be caused by injecting CAN messages through the OBD-II port, and therefore, could be caused at any other point on the bus.

These results demonstrate a CAN bus security concern. If an attacker were able to access the CAN bus, physically, or by compromising another ECU, they would be able

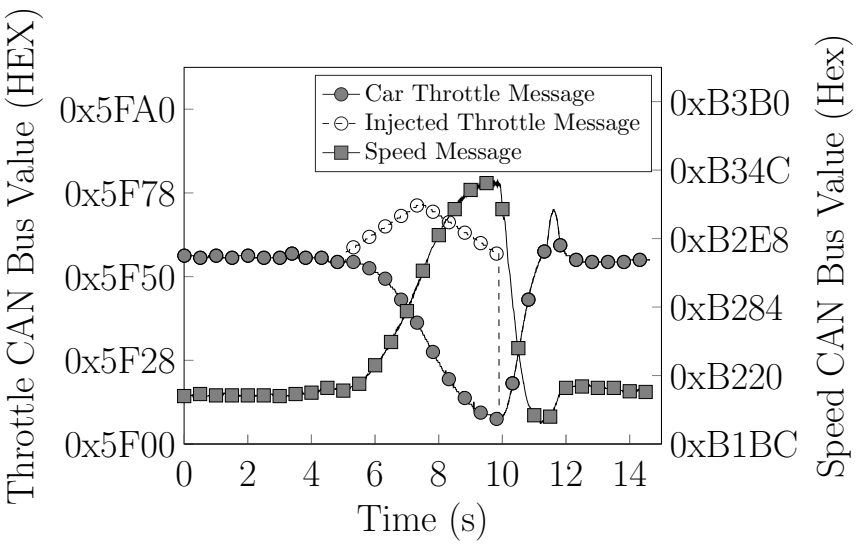

Fig. 2.9: CAN ramp injection through OBD-II port with resulting vehicle speed. Injected throttle message was sent on bus immediately following car throttle message. Values read from CAN bus and displayed in hexadecimal format.

to effect the acceleration of the vehicle without causing any errors. Remote access to the vehicle, but not necessarily the requisite CAN bus, could be effected by compromising the Telematic Control Unit (TCU) or a wireless Tire Pressure Monitoring Sensor (TPMS). The TPMS sends a signal to the Body Control Module (BCM), which in turn transmits a message on the medium speed CAN bus (MS-CAN), while the TCU is connected to the I-CAN bus (it is unlikely, however, that compromising a sensor would allow for injection of arbitrary CAN messages onto the I-CAN or MS-CAN bus). These busses are connected to the EV-HS bus through a gateway module; transmitting a message from one bus to another, which would be required for either the TCU or TPMS to impersonate the PCM by passing APP messages, was not explored in this work. Regardless of the access approach, the driver is able to stop the unwanted acceleration by pressing the brake pedal. However, other works indicate that it is possible to make the vehicle ignore braking requests [29] [28] [26] [27]. This was not investigated as part of this work. Another security concern is that of a malicious technician. Since technicians will often access the OBD-II port when a vehicle is being serviced, it would be quite simple for them to leave an OBD-II injection platform connected to the OBD-II port. The acceleration control could be initiated remotely or by a timer,

causing the vehicle to accelerate at a dangerous time.

We present two remediation strategies that could be employed to help protect against this vulnerability. First, a simple change in the acceleration system architecture, such that the APP sensor connects directly to the TCM, which is the actuating module. This would remove the need of a throttle message to be sent from the PCM to the TCM and effectively remove the attack surface. The second approach is through device fingerprinting for both the digital and analog signals [41] [42]. This would allow the receiving module to authenticate the transmitting module, and prevent this type of attack.

CHAPTER 3

MODEL IDENTIFICATION AND LOW-LEVEL CONTROLLER DESIGN

This chapter provides an examination of the methods used for model identification of the steering, braking, and acceleration systems in the 2013 Ford Focus Electric. Details are given of the low-level controller design of desired velocity and steering wheel angle.

The efforts discussed in this chapter were led by the author of this work and his research partner, Austin Costley [30]. It is important to note the collaboration effort with Austin, and identify his contributions. In particular, Austin was instrumental in model identification and low-level controller design. This chapter was prepared for a coauthored journal submission with Austin. It is included in this thesis for completeness.

3.1 Model Identification and Low Level Controller Design

A simple overview of the control structure for the automated vehicle platform is shown in Fig. 3.1. The high level controller plans the path and provides the desired vehicle speed, $v_{desired}$, and desired steering wheel angle, $\theta_{desired}$. The low level controllers discussed in this section are the inner loops that control vehicle speed and steering wheel angle. The vehicle commands are τ_{cmd}, APP_{cmd}, and BPP_{cmd}, and represent, steering torque, APP, and brake pedal position, respectively.

The first step in the development of the low level controllers was to determine a model of the system being controlled. A system model is expressed as a transfer function relating the input to the output of the system. Models were identified to relate the accelerator and brake pedal inputs to vehicle velocity, and steering wheel angle to vehicle heading. The following subsections review the model identification approach and low level controller design process for the Ford Focus.

3.1.1 Longitudinal Model

The longitudinal characteristics of the vehicle are affected by the APP sensor and the BPP sensor. These two systems were tested and identified separately, then implemented together as a complete longitudinal model.

In [43], Dias et al. perform longitudinal model identification and controller design for an autonomous vehicle. This approach was examined for the current work, however, a more straightforward classical controls technique using step responses was ultimately used. Once the acceleration and braking systems were identified, a control system was developed for each input device. The control loops for accelerator and braking were connected by switching logic to determine whether the accelerator or brakes should be used. A similar two loop control system with a switching logic component was used for the longitudinal controller in the current work. However, this work is an open-source project that uses the Robot Operating System (ROS) [44].

For the APP sensor system identification, the vehicle was placed on a dynamometer [45] and step inputs were initiated on the APP sensor from 4% to 15% at increments of 1%. Fig. 3.2 shows the step responses for some accelerator pedal inputs. It was observed that for a given APP percentage, the vehicle would eventually settle at a specific speed. The relationship between APP and speed can be described by a first order transfer function.

The general equation for a first order transfer function, $G(s)$, can be represented by

$$G(s) = \frac{K}{\tau s + 1}, \tag{3.1}$$

where K is the constant or equation that relates APP to vehicle speed, and τ is the system time constant. The equation for K was derived from a linear fit of the a scatter plot of max speeds from the step input, as shown in Fig. 3.3, and given by

$$f(x) = 3.65x - 9.7, \tag{3.2}$$

where $f(x)$ is the vehicle speed and x is the APP percentage. The test track (shown in

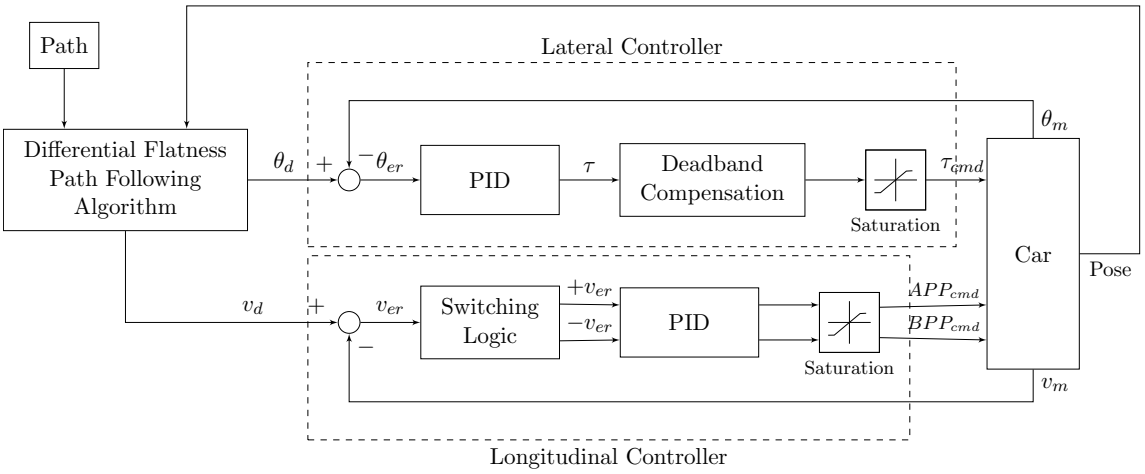

Fig. 3.1: High-level system block diagram. Shows low-level control loop for lateral and longitudinal control and high-level differential flatness path following feedback loop.

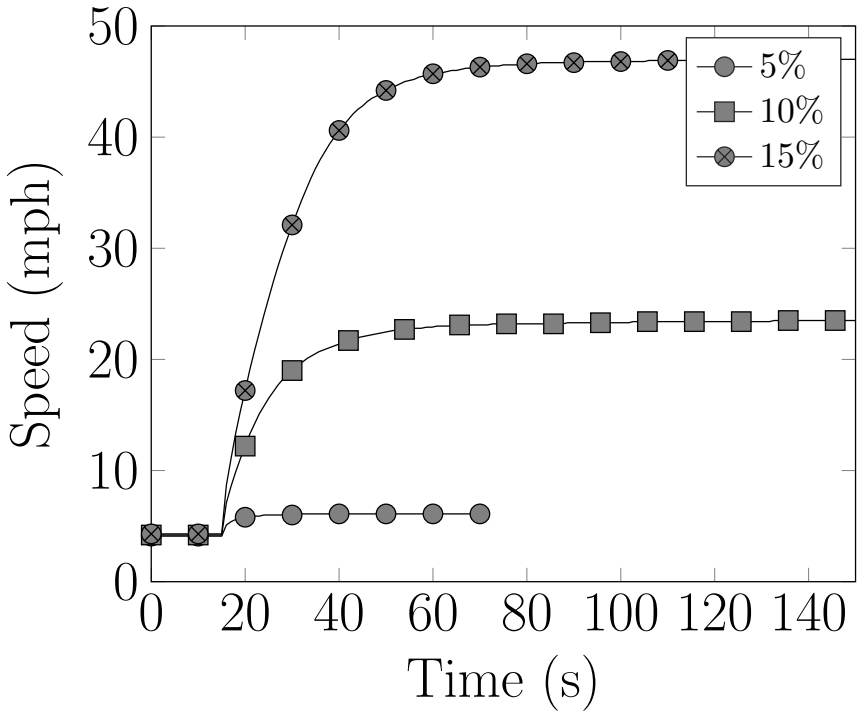

Fig. 3.2: APP step response for 5%, 10%, and 15% pedal presses. The graph shows a general first order speed response for a given pedal percentage.

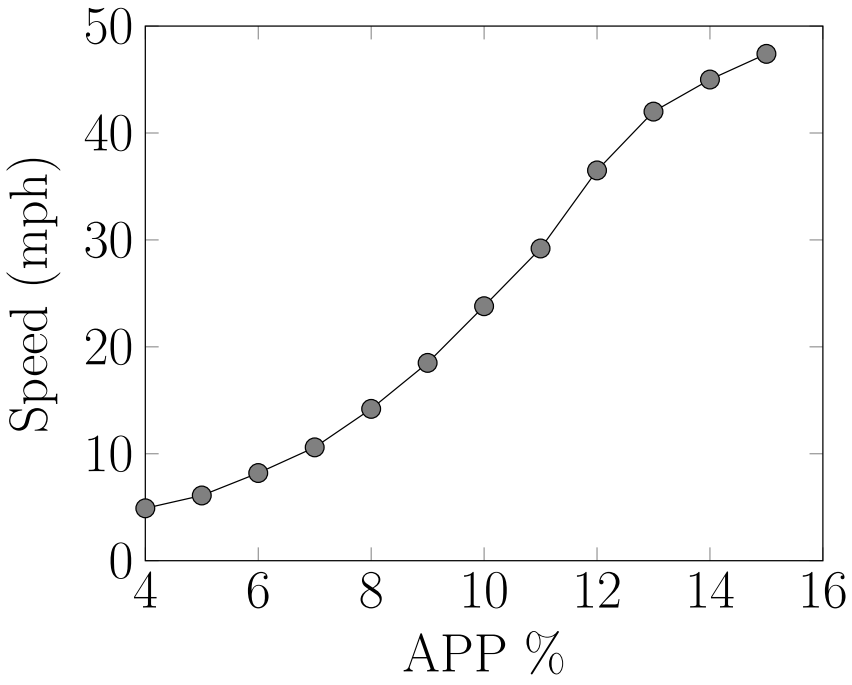

Fig. 3.3: Vehicle settling speeds for given APP step input percentages.

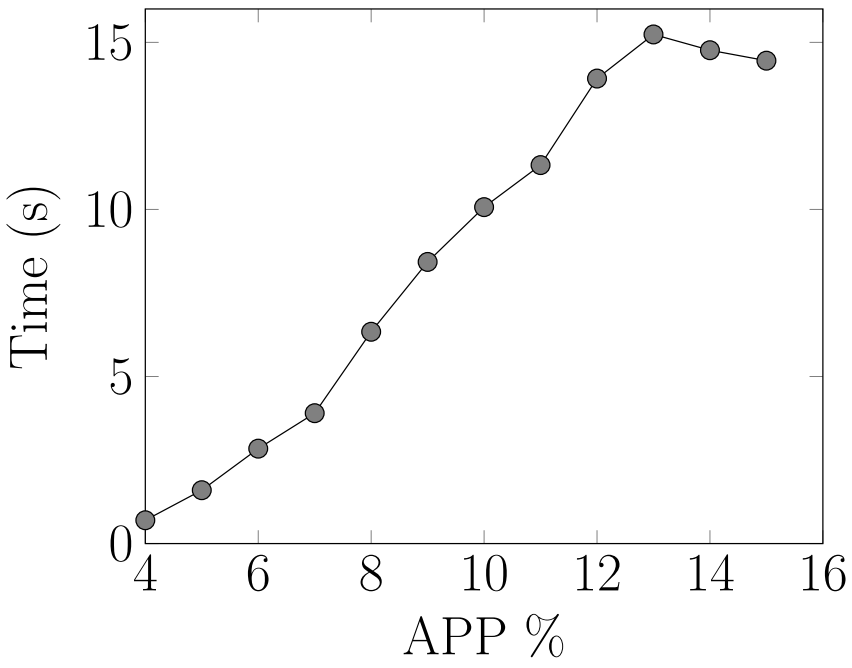

Fig. 3.4: Time constants for given APP step input percentages.

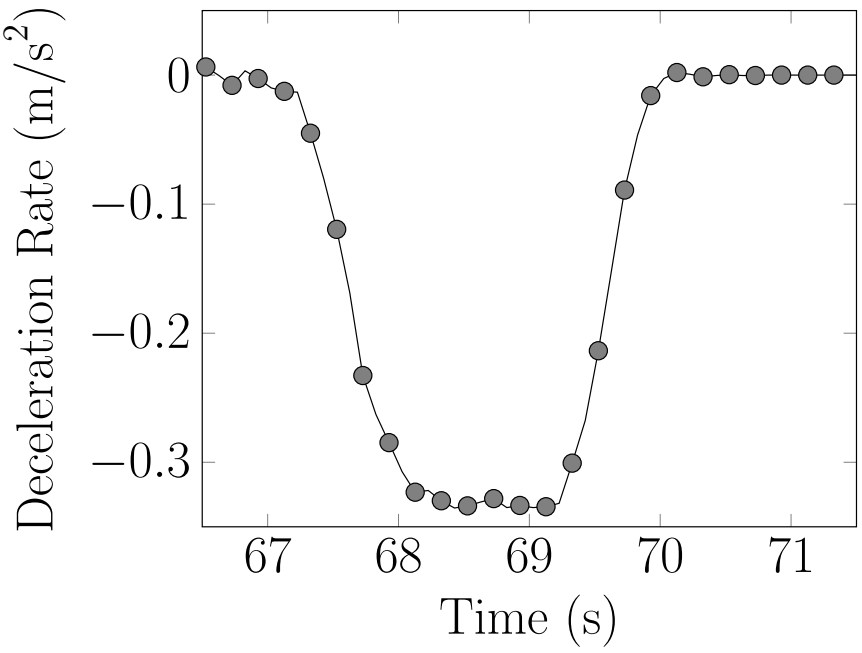

Fig. 3.5: Deceleration rate for BPP step input of 15%. The figure shows a first order relationship between BPP percentage and deceleration rate.

Fig. 2.5) where the vehicle was operating is an oval track with sharp corners on the north and south side. The sharp corners and the short straightaways limit the vehicle operating speeds to between 15 and 25 mph for the initial automation. The τ value that best represented the vehicle response between 15 and 25 mph was chosen as the time constant for accelerator pedal input in the longitudinal model. Fig. 3.4 shows the time constants for varying APP percentages. The time constant for the accelerator pedal input was chosen to be 7 seconds, as this best represented the system response for the nominal operating conditions. For BPP system identification, the vehicle was driven in a large, flat, asphalt area at speeds ranging from 5 to 25 mph at 5 mph increments. The vehicle was accelerated to the desired speed by a driver. Once the vehicle obtained the desired speed, an input to the braking system was initiated through the ROS setup discussed in Section 5.1. Step inputs were initiated ranging from 5% to 50% of BPP percentage at increments of 5% for each speed value. The speed data seemed to show a consistent rate of change for a given BPP percentage. To confirm this, the speed data was smoothed using a 5 point moving average, and the derivative of the

smoothed data was taken by calculating the difference between successive data points, and dividing by the elapsed time between data points. Fig. 3.5 shows the vehicle deceleration due to a braking event. It was observed that the settling value for the deceleration rate was consistent for a given BPP percentage and varying speeds, which concluded that the longitudinal model was independent of current vehicle speed. This speed independence can be seen in Fig. 3.6 where each line shows the deceleration rate for a given BPP percentage. At low BPP values, the lines converge, meaning that deceleration is unaffected by very small brake pedal percentages. However, at higher brake pedal percentages, the lines show distinct deceleration rates regardless of the vehicle speed. To show the relationship between BPP percentage and deceleration, an average was taken for each BPP value across each of the speeds. The result of this operation is shown in Fig. 3.7.

Similar to the APP model, the relationship between BPP and deceleration could be described by a first order transfer function. After analyzing the deceleration curves at different BPP percentages and for different speeds, the system time constant, τ, was calculated to be 0.3 seconds. The equation that relates BPP to deceleration was determined by finding a curve fit algorithm for the curve in Fig. 3.7. This would result in an equation that would provide a BPP percentage for a desired deceleration rate. The equation for K is given by

$$f(x) = -0.0018x^2 + 0.029x - 0.3768, \tag{3.3}$$

where $f(x)$ is the deceleration, and x is the BPP percentage. This equation is used to describe K from the general first order transfer function equation.

3.1.2 Lateral Model

The lateral model of the vehicle was determined by step response analysis. The model relates an input from the steering torque sensor to changes in the steering wheel angle. As discussed in Section 2.1.2, the torque sensor measures the torque applied by the driver, and sends that information to the PSCM. The PSCM activates the power assist motor that connects to the steering rack, and moves the wheels. The steering wheel angle is measured

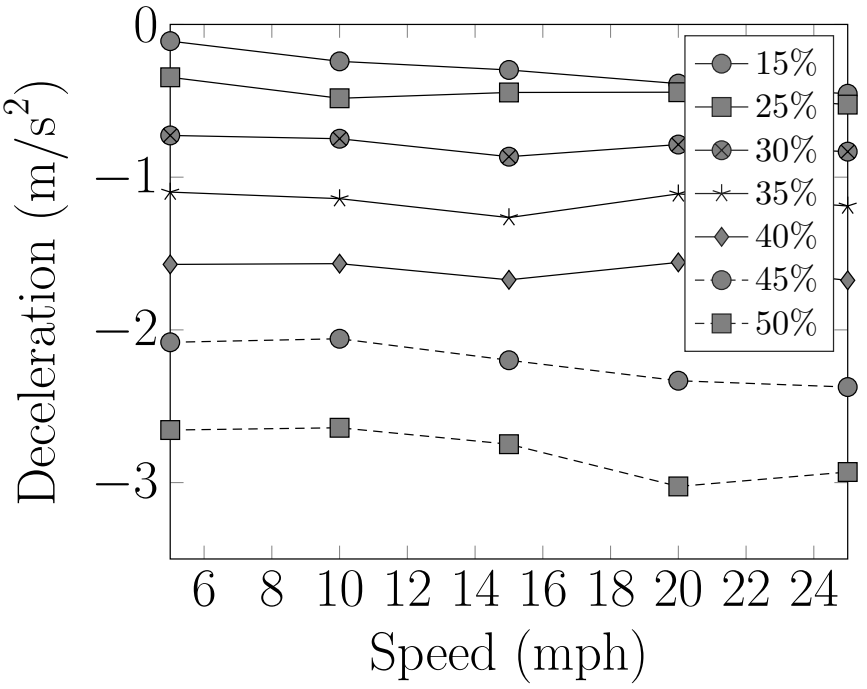

Fig. 3.6: Vehicle deceleration for BPP percentages at a variety of speeds. The values are the average of the deceleration rates settling point in response to a BPP step input. The lines for BPP percentages do not cross, and therefore, indicate speed independence for the brake model.

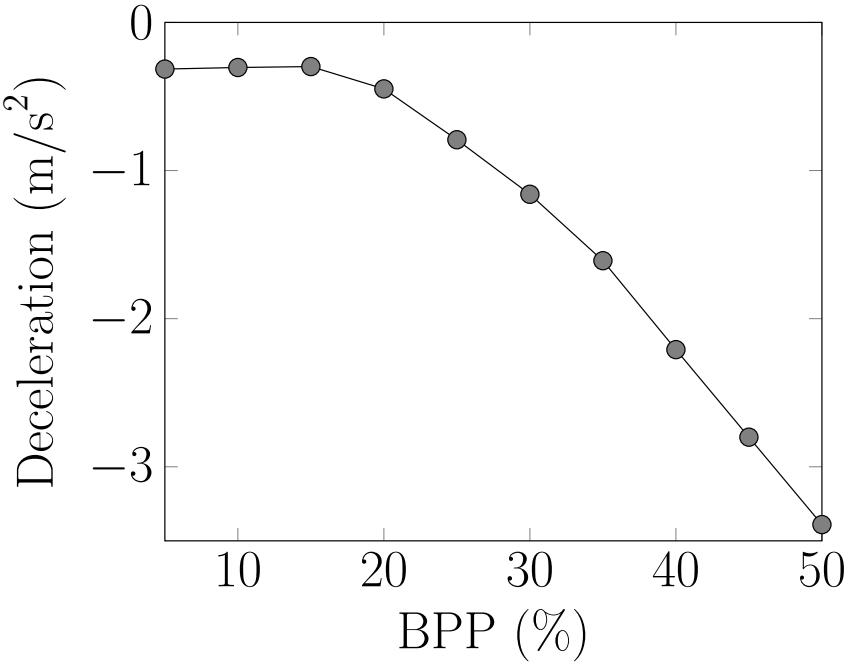

Fig. 3.7: Average deceleration settling rates due to BPP step input percentages.

by a sensor in the steering wheel and output on the CAN bus at a high level of precision.

Step inputs were initiated on the steering torque duty cycle signal ranging from 50% to 63% at 1% increments. Tests were performed at a large, flat, asphalt area with vehicle speeds ranging from 5 mph to 25 mph. Fig. 3.8 shows the results of the step input tests performed at 25 mph. It was observed that a general first order transfer function could be used to describe the relationship between steering torque duty cycle and steering wheel angle. However, at lower speeds and higher torque values, this observation is not valid. Fig. 3.9 shows the step response of the steering system at 15 mph. At the higher torque values, the steering wheel angles do not settle to a consistent steering wheel angle. It was also observed that the settling angles for a given steering torque duty cycle are not consistent for varying speeds. Therefore, the lateral model identification is speed dependent and would require a speed dependent limit on the steering torque duty cycle. Providing these characteristics, the system can still be modeled as first order transfer function for a given speed.

The steering data was analyzed in order to determine the gain equation, K, and the time constant, τ. Time constants were calculated for each step input response and for each speed. Fig. 3.10 *Bottom* shows the time constants for given steering torque duty cycles. Each of the lines indicates the speed at which the test was performed. It can be seen that at low speeds and low duty cycles the time constants are not consistent. But at higher speeds the inconsistencies lessen. A time constant, τ, of 0.2 seconds was chosen to optimize for typical vehicle operation.

Since the lateral system was found to be speed dependent, the gain equation K must also be speed dependent. The step input tests were performed at 5 mph increments so a gain equation K would be found for each speed value. These gain equations relate steering torque duty cycle to steering wheel angle. Fig. 3.10 shows the settling angles for varying steering torque duty cycles when the vehicle was traveling at 25 mph. A curve fit approximation was completed for this data set, and a solution was determined by solving the given equation. For this data set, the given equation for K is

$$f(x) = 59.4x^2 - 6802.7x + 195084.5, \tag{3.4}$$

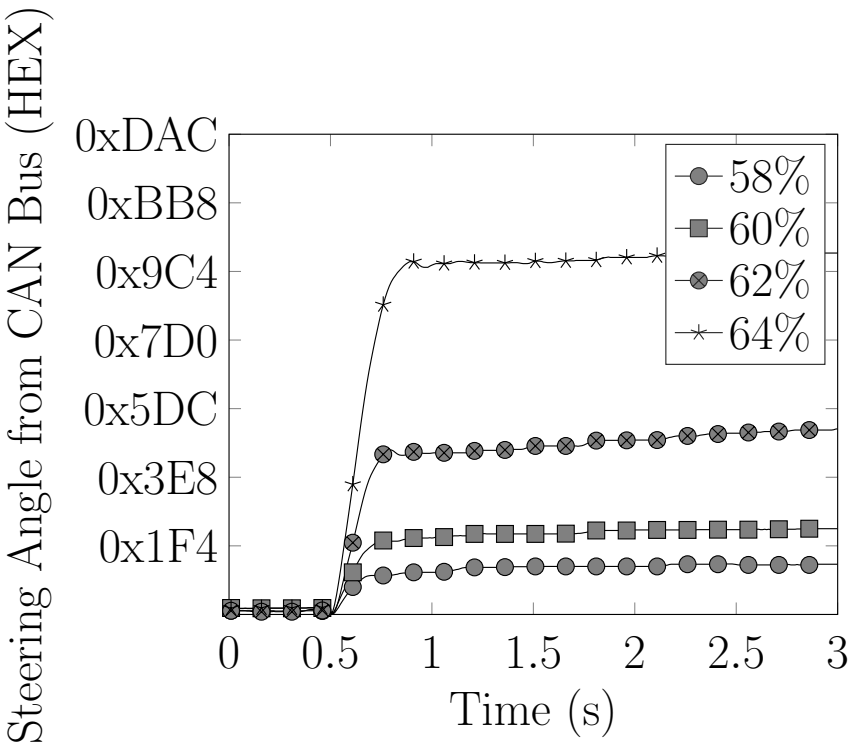

Fig. 3.8: Steering torque step response for 58%, 60%, 62%, and 64% duty cycles at a vehicle speed of 25 mph. The plot indicates a first order relationship between torque duty cycle input and steering wheel angle.

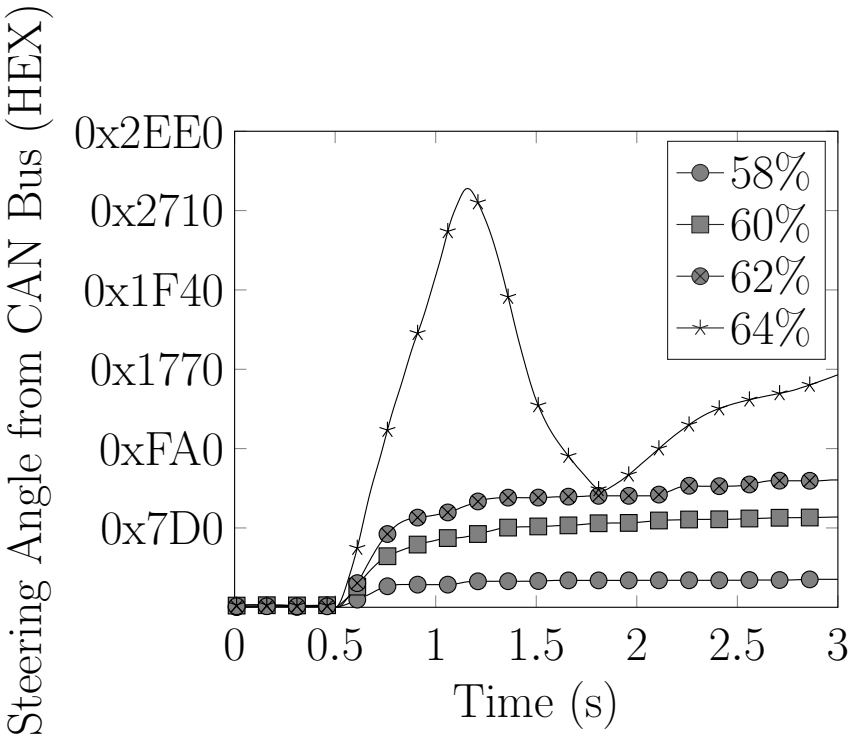

Fig. 3.9: Steering torque step inputs for 58%, 60%, 62%, and 64% duty cycles at a vehicle speed of 15 mph. The plot indicates a first order relationship between torque duty cycle input and steering wheel angles, however, at high duty cycle percentages the first order relationship is not valid.

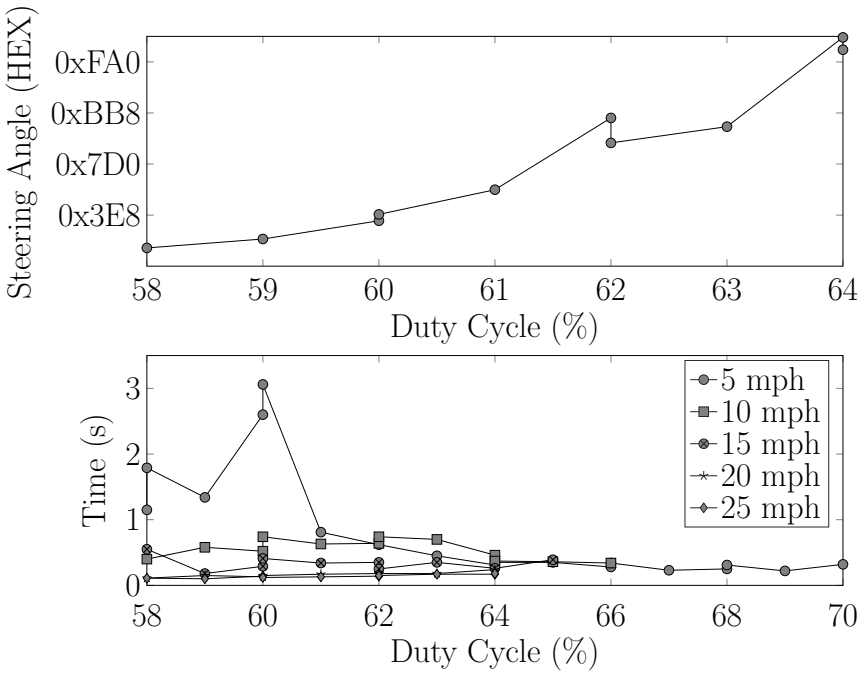

Fig. 3.10: *Top:* Steering angle settling values for given steering torque duty cycle step inputs. Values represented in hexadecimal format as received from CAN bus. *Bottom:* Steering angle time constants for given steering torque duty cycle step inputs at varying speeds. The time constants converge at higher speeds.

where $f(x)$ is the steering wheel angle and x is the steering torque duty cycle.

Figs. 3.8 and 3.9 show the step response of the vehicle due to steering torque input signals. The graphs do not include step input values below 58% because the step responses at such values had little effect on the steering wheel angle. This exposed a deadband in the response from the steering torque sensor input to the steering wheel angle. A deadband compensation algorithm was implemented to mitigate the effects of this non-linearity. As shown in Fig. 3.1, the deadband compensation code was executed just before the signal was sent to the vehicle. If the torque input value was greater than 50%, then

$$\tau_{cmd} = B_{max} + \frac{\tau - 50}{\tau_{max} - 50}(\tau_{max} - B_{max}) \tag{3.5}$$

was used to compensate for the deadband. If the torque input value was less than 50%, then

$$\tau_{cmd} = B_{min} + \frac{50 - \tau}{50 - \tau_{min}}(\tau_{min} - B_{min}) \tag{3.6}$$

was used to compensate for the deadband. Where τ_{cmd} is the torque command sent to the vehicle, τ is the value received from the PI controller, B_{max} is the upper limit of the deadband, B_{min} is the lower limit of the deadband, τ_{max} is the maximum allowed value for the steering torque signal, and τ_{min} is the minimum allowed steering torque signal. For the deadband on the 2013 Ford Focus EV, the upper and lower limits were 55% and 45%, and the maximum and minimum values for the torque signal were 64% and 37%, respectively.

3.1.3 PI Controller Design

Low-level control loops were designed to control vehicle speed and steering wheel angle. The desired speed and desired steering wheel angle would be input to the low-level control loops from a user or high-level controller. The low-level longitudinal controller interfaced with the accelerator and brake pedals to effect vehicle speed. A separate loop was designed for each vehicle input, and switching logic was used to choose whether the acceleration or brake loop would be used. The low-level lateral controller would receive the desired steering

wheel angle and determine the appropriate input to the steering torque sensor to achieve the desired angle.

A Proportional Integral (PI) Feedback Controller was implemented for longitudinal and lateral control. Fig. 3.11 shows a basic PI Feedback Controller for a first order system. The transfer function block represents the vehicle and contains the system model. The $\frac{1}{K}$ block effectively cancels out the gain equation K, and helps relate the speed error to a vehicle input. For example, in the longitudinal controller, the K equation receives the APP as an input, and outputs speed. Therefore, the input to the transfer function block must be an APP value. However, the control loop is calculating a speed error, so the output of the PI block is a speed value. The $\frac{1}{K}$ block translates the speed value into an appropriate APP value.

Since the K and $\frac{1}{K}$ can be combined to equal 1, they can be ignored in the loop equation. The open loop transfer function of this system is then given by

$$G_{OL}(s) = \frac{1}{(\tau s + 1)}. \tag{3.7}$$

Closing the feedback loop and adding the PI controller gives

$$G_{CL}(s) = \frac{\frac{k_p}{\tau}\left(s + \frac{k_i}{k_p}\right)}{s^2 + \left(\frac{1}{\tau} + \frac{kp}{\tau}\right)s + \frac{k_i}{\tau}}. \tag{3.8}$$

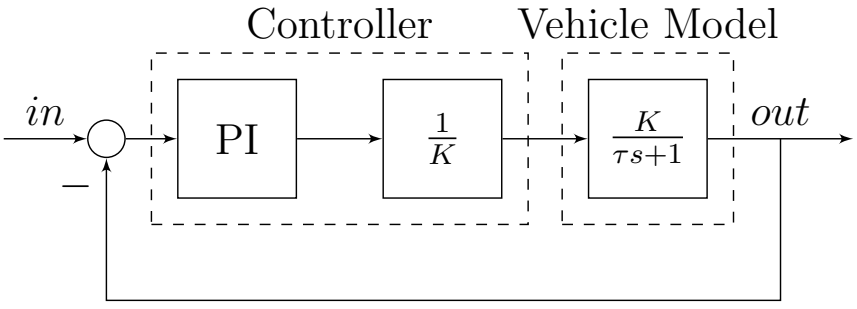

Fig. 3.11: General control loop for a first order PI controller.

The system is stable if the real part of the closed-loop poles are negative. Solving for the closed loop poles and zero yields

$$s = \frac{-\left(kp+1\right) \pm \sqrt{\left(k_p+1\right)^2 - 4k_i\tau}}{2\tau}, \tag{3.9}$$

and

$$s = -\frac{k_i}{k_p}, \tag{3.10}$$

respectively. From these equations it can be determined that if k_p, and k_i are positive the system will be stable.

The second order transfer function obtained through closing the loop can be written, in a general form, as

$$\frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}, \tag{3.11}$$

where ζ is the damping coefficient and ω_n is the natural frequency. The damping coefficient determines whether the system will be under-damped, over-damped, or critically damped and the natural frequency helps to determine the time constant for the system. The time constant, τ, is given by the equation $\tau = \frac{1}{\zeta\omega_n}$. Values were chosen for the damping coefficient and the time constant to define the system behavior. From these values one can determine the appropriate k_p and k_i for the system. The equations for k_p and k_i are given by

$$k_p = \tau\left(2\zeta\omega_n - \frac{1}{\tau}\right), \tag{3.12}$$

and

$$k_i = \tau\omega_n^2, \tag{3.13}$$

respectively. Table 3.1 shows the calculated values for each of the control inputs, where the τ_{car} column shows the time constants found during model identification and are internal vehicle parameters.

Table 3.1: Table of Values for Input Loops

Input	τ_{car}	ζ	τ	ω_n	k_p	k_i
Accelerator Pedal	7	1	0.5	2	27	28
Brake Pedal	0.3	1	0.5	2	0.2	1.2
Steering Torque	0.2	1	0.33	3	0.2	1.8

CHAPTER 4

LANE DETECTION AND VISION-BASED CONTROL

This chapter details the methods for lane detection and the design of an accompanying vision-based controller which uses lane information to navigate within the roadway.

4.1 Lane Detection

Lane detection is an essential part of navigating urban roadways. Although it is intuitive for humans to identify and follow lane markers, machines are plagued with many obstacles in the performing the task. Difficulties include varying road types and markers, changes in weather and lighting conditions, and other vehicles obscuring the view of the road. This work focuses on detecting the lanes on the closed circuit test track at Utah State University's EVR [46]. While there are no other vehicles to obstruct the lane markers, the track presents challenges in lane detection as shown in Fig. 4.1. Charging gutters in the middle section of the road are surrounded by cement strips similar in width and color to lane markers. Curves on either end of the track are sharper than those found in highway driving and most standard roadways. Charging equipment near the side of the road casts shadows on the lane markers, making them more difficult to detect. Extensive work on detecting lane markers in urban roadways was performed by Aly [14]. A modified version of his algorithm is presented in this section.

4.1.1 Inverse Perspective Mapping

Images received by a camera suffer from what is known as the perspective effect. Lines which are parallel in the world frame do not run parallel in an image taken from a camera mounted on a vehicle. Instead, they converge at the horizon as can be seen in Fig. 4.1. Furthermore, without knowing an object's size, there is not a measure of distance for a monocular image. Inverse Perspective Mapping is a technique which resolves the perspective

Fig. 4.1: Original image taken on EVR test track. Difficulties for lane detection are shown including shadows, a steep curvature, and shapes similar to lane markers in the center of the road

problem by creating a top-view perspective of a forward facing image. This bird's-eye view has distinct advantages in lane detection. After an IPM transform, lanes which run parallel in the world frame are also represented as parallel in the image frame. IPM give a sense of depth and maps a pixel to meter distance for the image. It also focuses on a subregion of the original image, decreasing the computational processing needed.

IPM is a geometric transformation which uses camera intrinsics and extrinsic to display a bird's-eye view of the image. Intrinsic parameters include camera focal length and focal center. Camera height, pitch, and yaw are taken into account for extrinsic parameters. Fig. 4.2 shows IPM operates in three frames: a world frame $\{F_w\} = \{X_w, Y_w, Z_w\}$ centered at the camera's optical center, a camera frame $\{F_c\} = \{X_c, Y_c, Z_c\}$, and an image frame $\{F_i\} = \{u, v\}$. Pitch angle α and yaw angle β are allowed for, but no roll angle. It is assumed the camera's X_c axis stays in the world frame's X_wY_w plane. The camera's height above the ground plane is defined as h. The road surface is assumed to be flat. From any point in the image plane ${}^iP = \{u, v, 1, 1\}$, it's projection on the road plane can be found by

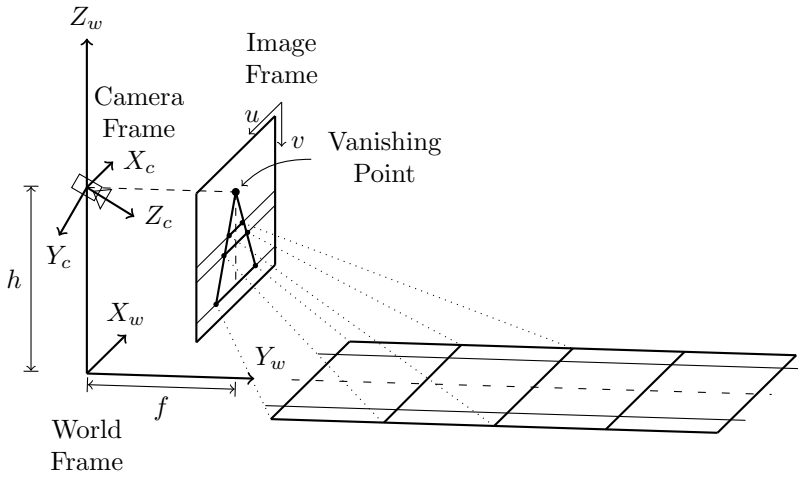

Fig. 4.2: Inverse Perspective Mapping (IPM) frames. IPM is a geometric transformation which maps a 2D image to a 3D world frame

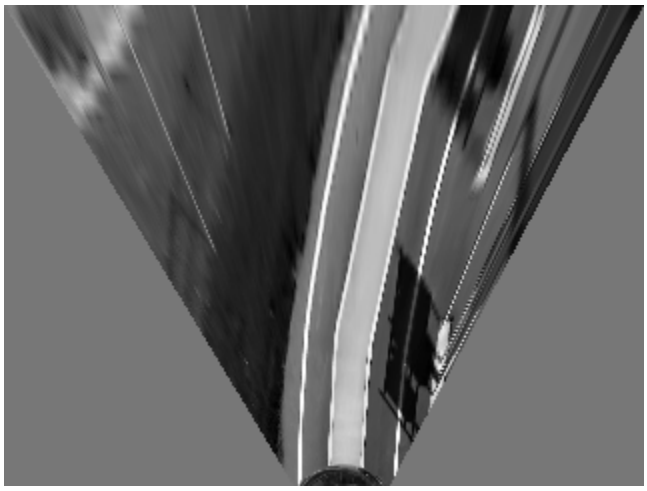

Fig. 4.3: Image with Inverse Perspective Mapping (IPM) applied

applying the homogeneous transformation

$$ {}^{g}_{i}T = h \begin{bmatrix} -\frac{1}{f_u}c_2 & \frac{1}{f_v}s_1 s_2 & \frac{1}{f_u}c_u c_2 - \frac{1}{f_v}c_v s_1 s_2 - c_1 s_2 & 0 \\ \frac{1}{f_u}s_2 & \frac{1}{f_v}s_1 c_1 & -\frac{1}{f_u}c_u s_2 - \frac{1}{f_v}c_v s_1 c_2 - c_1 s_2 & 0 \\ 0 & \frac{1}{f_v}c_1 & \frac{1}{f_v}c_v c_1 + s_1 & 0 \\ 0 & -\frac{1}{h f_v}c_1 & \frac{1}{h f_v}c_v c_1 - \frac{1}{h}s_1 & 0 \end{bmatrix} \tag{4.1} $$

where $c_1 = \cos\alpha$, $c_2 = \cos\beta$, $s_1 = \sin\alpha$, and $s_2 = \sin\beta$. The horizontal and vertical focal lengths are $\{f_u, f_v\}$ and the coordinates of the optical center are $\{c_u, c_v\}$. With this transformation, ${}^{g}P = {}^{g}_{i}T {}^{i}P$ is the point on the ground plane corresponding to ${}^{i}P$ in the image plane. The same holds for the inverse of the transform

$$ {}^{i}_{g}T = \begin{bmatrix} f_u c_2 + c_u c_1 s_2 & c_u c_1 c_2 - s_2 f_u & -c_u s_1 & 0 \\ s_2(c_v c_1 + f_v s_1) & c_2(c_v c_1 + f_v s_1) & -f_v c_1 - c_v s_1 & 0 \\ c_1 s_2 & c_1 c_2 & -s_1 & 0 \\ c_1 s_2 & c_1 c_2 & -s_1 & 0 \end{bmatrix} \tag{4.2} $$

where a point in the ground plane ${}^{g}P = \{x_g, y_g, -h, 1\}$ can be obtained in the image frame by ${}^{i}P = {}^{i}_{g}T {}^{g}P$, followed by a rescaling of the homogeneous part. Fig. 4.3 shows an example of taking the IPM transform of Fig. 4.1. The original image has a size of 1280x720 and the transformed image has a size of 360x240. As mentioned before, the lanes in Fig. 4.3 now appear parallel and have a fixed width.

4.1.2 Filtering and Thresholding

After the IPM transform, the image is filtered horizontally using a second derivative Gaussian filter: $f_u(x) = \frac{1}{\sigma_x^2} exp(-\frac{x^2}{2\delta_x^2})(1 - \frac{x^2}{\delta_x^2})$. This is designed to accentuate bright areas on a dark background, following our assumption of a lane marker's appearance in the image. The filter is tuned to keep vertical lines of a lane maker's width. The filter also keeps quasi-vertical lines. Fig. 4.4 shows the filter applied to Fig. 4.3. It can be seen that lane data is kept well along with other parts of the image with similar attributes. Although Aly [14]

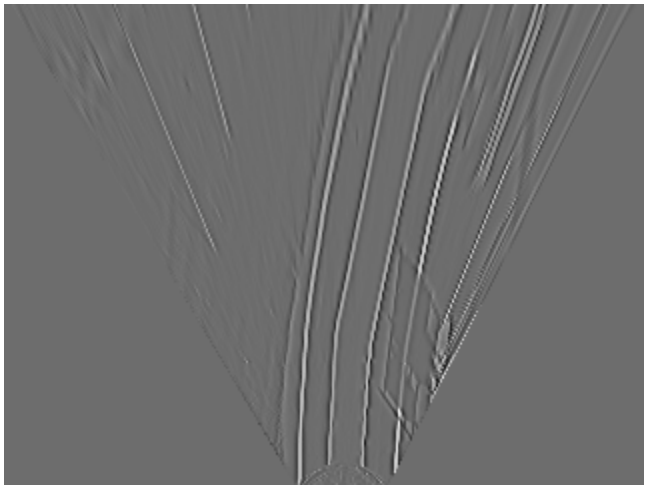

Fig. 4.4: Image filtered using a second derivative Gaussian horizontal kernel

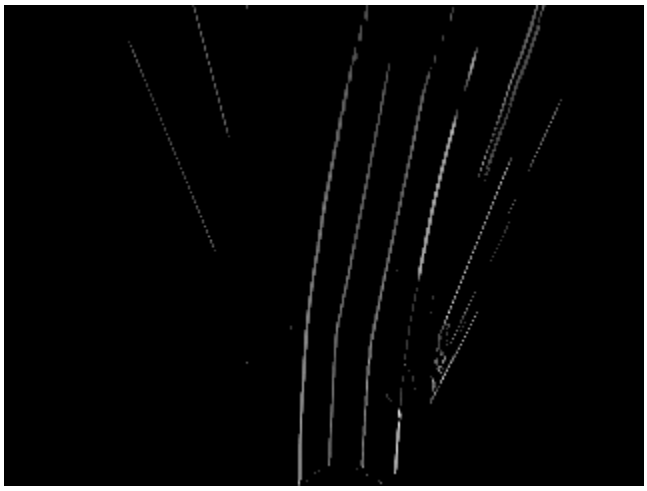

Fig. 4.5: Thresholded image retaining high intensity areas. The image is not binarized and keeps all intensity values above the threshold

suggests a vertical smoothing filter, one was not used in our implementation because it did not significantly increase the effectiveness of the filter.

Thresholding is performed on the filtered image, keeping only the highest values. The average pixel value is calculated for the filtered image, and values below it's q% are set to zero. A Value of $q = 97.5$ was used for the images in this chapter. It is important to note that the thresholded image is not binarized. Pixel values above the threshold are kept to preserve intensity data. Fig. 4.5 is the result of this thresholding method on Fig. 4.4. Lane markers are easily seen in the image with the exception of shadow areas.

4.1.3 Line Fitting

For the thresholded image, pixels are summed vertically for each column. The result is smoothed by a Gaussian filter and the local maxima are detected as shown in Fig. 4.6. The lines are grouped to account for multiple entries and the resulting lines (Fig. 4.7) are used as the basis for line and spline detection. A bounding box is taken around each of these lines and a line is fit using RANSAC for the data within the bounding box. Fig. 4.8 shows a line fit to a sub-image defined by a bounding box around one of the lines in Fig. 4.7. The same process is run for each of the bounding regions and produce a line fit by RANSAC for each. Fig. 4.9 shows the combination of all lines fit.

4.1.4 Spline Fitting

Algorithm 1 RANSAC Spline Fitting

```
1: for i=1 to numIterations do
2:     points=getRandomSample()
3:     spline=fitSpline(points)
4:     score=computeSplineScore(spline)
5:     if score > bestScore then
6:         bestSpline = spline
7:     end if
8: end for
```

In the previous step, a line was fit to the data within a subregion. In this step, a spline

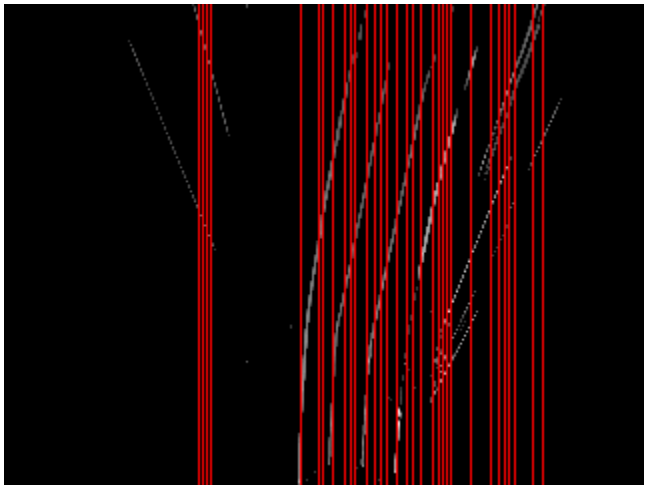

Fig. 4.6: Local maxima of image columns. Pixel values are summed vertically. Local maxima are indicated by the red lines

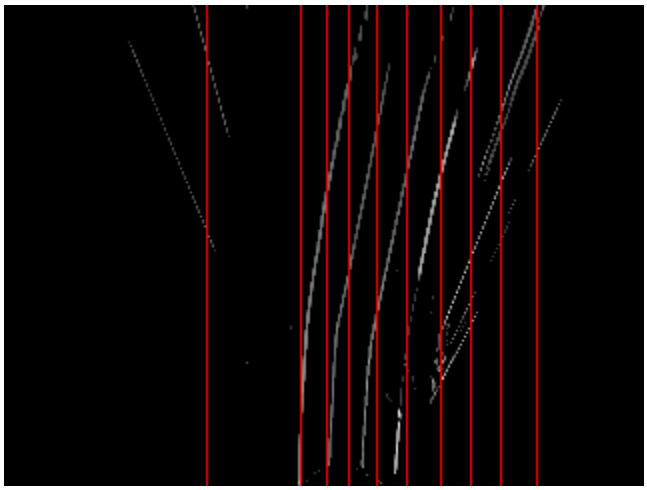

Fig. 4.7: Vertical lines indicating the subregions of interest in image. A bounding box is taken around each line and a curve is fit for all data within the bounding region

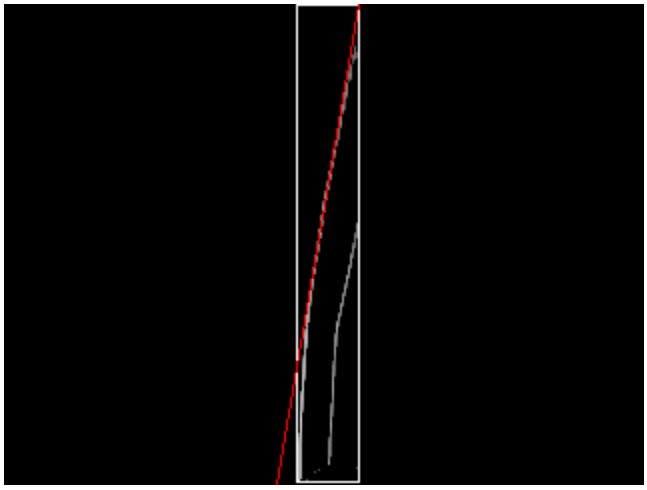

Fig. 4.8: Region showing line fit by RANSAC to potential lane data

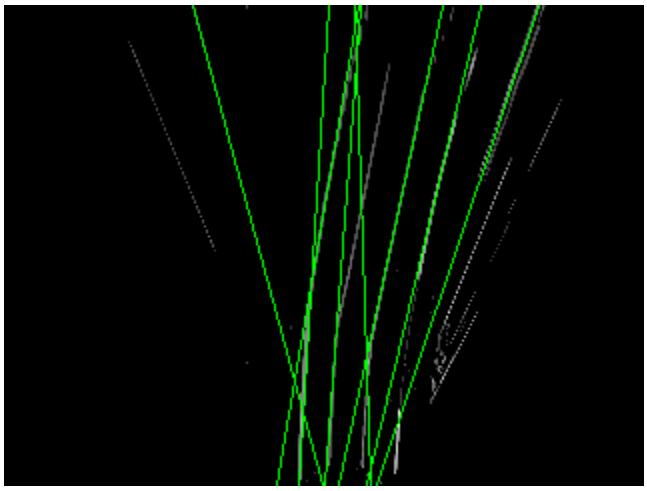

Fig. 4.9: Lines fit to subregions of image. These lines are used as initial guesses for RANSAC spline fitting

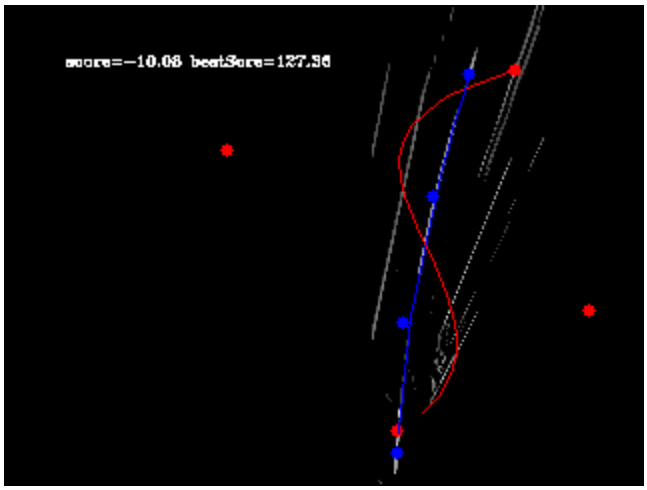

Fig. 4.10: RANSAC fitting of spline to subregion data. The current iteration is shown in red while the best fit spline is shown in blue

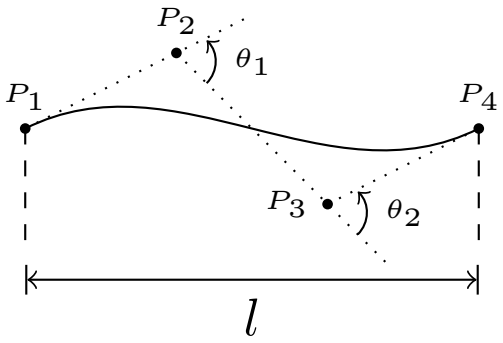

Fig. 4.11: Third degree Bezier Spline consisting of four control points. A measure of straightness is estimated through θ_1 and θ_2 and length is defined as l

is fit to that data, with the line used to determine a new bounding region. A RANSAC spline fitting method is used to robustly fit a spline. A third degree Bezier spline is used to fit this data and is defined by

$$Q(t) = T(t)MP \tag{4.3}$$

$$Q(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_0 \\ P_1 \\ P_2 \\ P_3 \end{bmatrix} \tag{4.4}$$

where $t \in [0, 1]$, $Q(0) = P_0$, $Q(1) = P_3$, and P_1 and P_2 are control points that determine the shape of the spline as can be seen in Fig. 4.11. The RANSAC method is outlined in Algorithm 1 and makes use of the following functions:

1. *getRandomSample()*: This function randomly selects n points from the sub-image Points are weighted according to the filtered and thresholded value such that points with a higher weight have a greater likelihood of being selected by *getRandsomSample()*.

2. *fitSpline()*: Points supplied by the previous function are fit to a third degree Bezier spline using a least squares method. For each of n points, a value $t_i \in [0, 1]$ is assigned to each point $p_i = (u_i, v_i)$ in the sample, where t_i is proportional to the cumulative sum of the euclidean distances from point p_i to the first point p_1. Define a point $p_0 = p_1$, for $t_i = 1..n$

$$t_i = \frac{\sum_{j=1}^{i} d(p_j, p_{j-1})}{\sum_{j=1}^{n} d(p_j, p_{j-1})} \tag{4.5}$$

where $d(p_i, p_j) = \sqrt{(u_i - u_j)^2 + (v_i - v_j)^2}$. This causes the first point $t_1 = 0$ and the last point $t_n = 1$. Define a matrix:

$$Q = \begin{bmatrix} p_1 \\ ... \\ p_n \end{bmatrix} \tag{4.6}$$

$$Q = \begin{bmatrix} t_1^3 & t_1^2 & t_1 & 1 \\ ... & ... & ... & ... \\ t_n^3 & t_n^2 & t_n & 1 \end{bmatrix} \tag{4.7}$$

and use the pseudo-inverse to solve for the P matrix:

$$P = (TM)^{\dagger}Q. \tag{4.8}$$

The matrix P contains the control points such that the spline minimizes the sum of squared error for the sample points in the sub-image.

3. *computeSplineScore()*: In this function each spline is given a score in order to determine which spline best fits the criteria for a lane marker. Normally, a spline score would be determined by the normal distance from every point to the third degree spline. A more efficient approach is used here in which the spline is rasterized using an iterative method. Pixel values are summed which belong to the spline to give it the raw score s. The final score also includes measure of the straightness and length of the spline:

$$score = s(1 + k_1 l' + k_2 \theta'). \tag{4.9}$$

The value l' is the normalized length of the spline. This is define as $l' = (l/v) - 1$ where v is the height of the image. Hence, a longer spline will return a value closer

to $l' = 0$ and a shorter spline will have a value closer to $l' = -1$. This gives a greater penalty to shorter splines. A measure of straightness is given by $\theta' = (\theta - 1)/2$ where $\theta = (\cos\theta_1 + \cos\theta_2)$, which is the mean of the cosine of angles between the lines joining the spline's control points as shown in Fig. 4.11. This measure gives a higher score to straighter splines. The values k_1 and k_2 are used for regulation in scoring. Fig. 4.10 shows the RANSAC spline fitting. The spline in red is the most recent iteration of the algorithm and the spline in blue is the best fit.

Algorithm 1 is run on each of the sub-images An example of its results can be seen in Fig. 4.12. After being fit using RANSAC, the splines go through post-processing steps of localization and extension. Localization is accomplished by choosing consistently sampled points along the spline and extending a normal line from them. Along this normal line, the local maximum is located and the point is moved to its location. Extension is performed in the IPM image forward and back from the spline's endpoints. The results of post-processing can be seen in Fig. 4.13.

4.1.5 Lane Selection

From the previous step, a set of splines is detected and extended in the IPM frame. This set can vary in size and range from zero to n splines. In the case that fewer than two splines are detected, lane selection is skipped because no center lane can be calculated without a pair of splines from which to base it. If exactly two spline are detected, the pair is checked for validity in the method presented below. When more than two splines are detected, the best pair is chosen based on a "goodness" criteria. Ideal splines are those corresponding to lane markers, and more particularly the lane markers corresponding to the lane in which the vehicle is currently driving. In order to determine a center lane to follow, two splines must selected, one to the right of the vehicle and one to the left.

To determine which pair of splines corresponded to lane markers in the vehicle's lane, a score was created. Each pair of splines was analyzed and scored on the following criteria illustrated in Fig. 4.14:

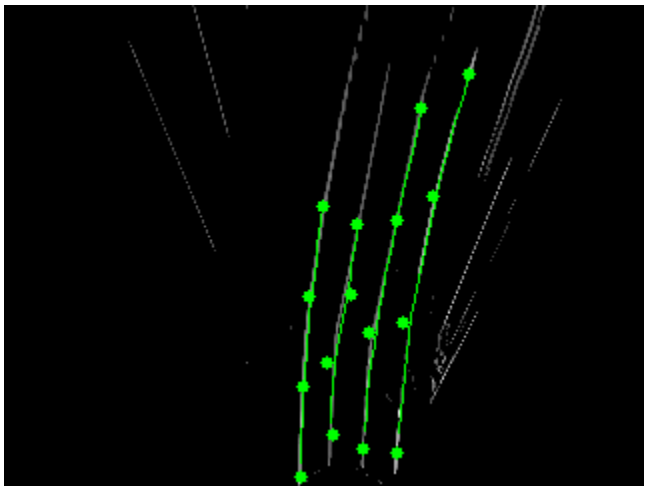

Fig. 4.12: Splines detected in the image. All are candidate lane markers

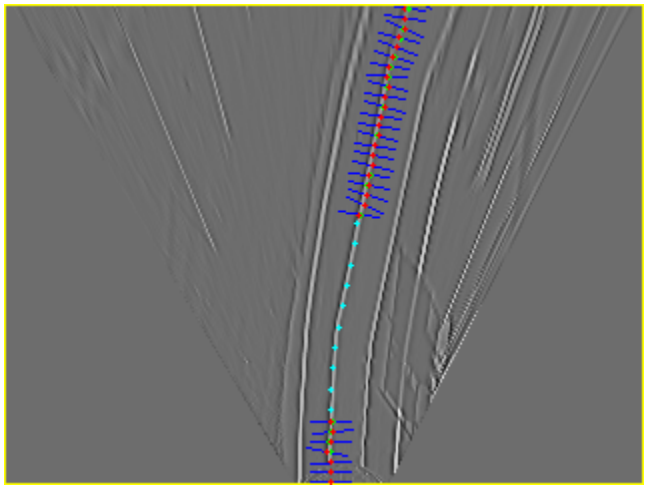

Fig. 4.13: Spline localization and extension in the IPM image

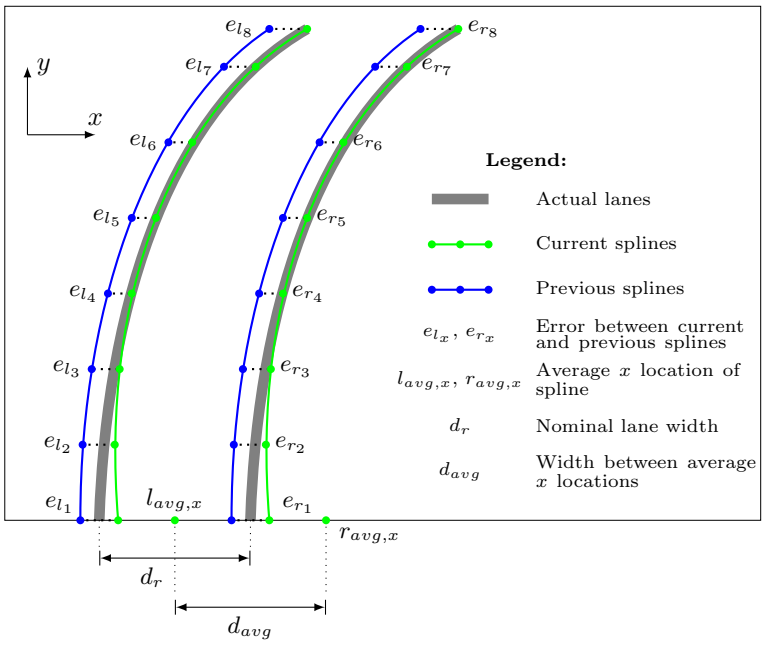

Fig. 4.14: Method for scoring pairs of lanes

1. *Distance*: A consistent number of n points were sampled from each spline. For each pair of splines, the average x value of the sampled points in the left lane, $l_{avg,x}$, and the right lane, $r_{avg,x}$, were taken. The average of these two values d_{avg} was calculated. This value was then subtracted from the nominal lane width d_r of the track. The distance error score was $e_d = d_r - d_{avg}$.

2. *Shape*: A measurement of shape was taken from the measure of straightness θ' discussed in Section 4.1.4. In this case, a difference in the measurements from each spline $e_{\theta'} = |\theta'_l - \theta'_r|$ was calculated, where θ'_l is the straightness of the left spline and θ'_r is the straightness of the right spline. Parallel splines will have a value of zero. The value of $e_{\theta'}$ increases for splines that are less similar in shape.

3. *Temporal Tracking*: If the vehicle is moving at regular speeds and the camera is sending images at a constant frame rate, it can be assumed that the lanes will not vary greatly from frame to frame. For each pair of lanes, the average x value of the

spline ($l_{avg,x}$ and $r_{avg,x}$) was used to determine which spline was on the left and which was on the right. The current left splines were than compared to the previous frame's left spline by calculating the euclidean distance e_{l_n} from each sample point. The same was done for the right splines giving e_{r_n}. The average error values

$$e_{t_l} = \frac{1}{n}\sum_{i=1}^{n} e_{l_i}, \quad e_{t_r} = \frac{1}{n}\sum_{i=1}^{n} e_{r_i} \tag{4.10}$$

were used to gauge how far the new candidate splines had strayed from the previous frame. The overall temporal error $e_t = (e_{t_l} + e_{t_r})/2$ gives a measure of how pairs of splines compared to the previous lane data. A low e_t signified a small deviation from the previous frame and a large e_t show the splines were far from those last seen.

For each of the three parameters listed above a small value was desired. A score was used

$$score = k_1(100 - e_d) + k_2(100 - e_{\theta'}) + k_3(100 - e_t) \tag{4.11}$$

that rewarded similar splines of an expected lane width that did not deviate far from the spline pair from the previous frame. A threshold was also set in the case that no two splines fit lane markers. Fig. 4.15 shows the rejected splines (red) and the accepted splines (green) that fit the criteria needed.

After two splines were validated and accepted as lane markers, a center lane was calculated between them. The calculated center lane was an average of the euclidean distances of the n sampled points in the two splines. Fig. 4.16 shows the chosen splines in white and the calculated center lane in red. This is the path sent to the vehicle to follow. After all of the calculations for desired path are calculated in the world frame, the detected lanes can be displayed in the image frame again by back-transforming them through IPM as shown in Fig. 4.17. The detected lanes are shown in green with the blue area between them representing the drivable area of the road.

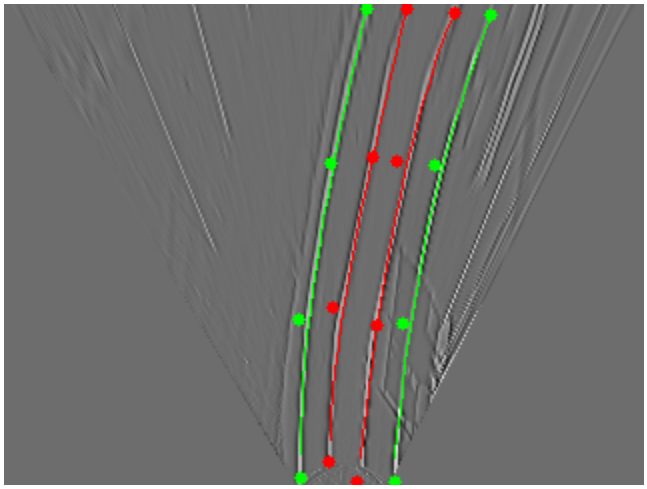

Fig. 4.15: Pairs of splines are checked for uniform distance and shape. Spline pairs with a distance close to nominal road width and parallel in form score higher. Accepted lane markers are shown in green while rejected splines are shown in red

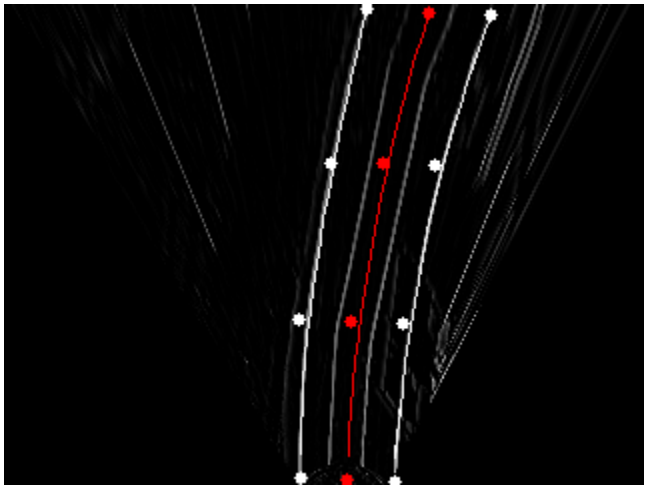

Fig. 4.16: Center lane calculation. The center lane is calculated as the average of corresponding points in the two splines. The center lane is shown in red and the detected lane markers are shown in white

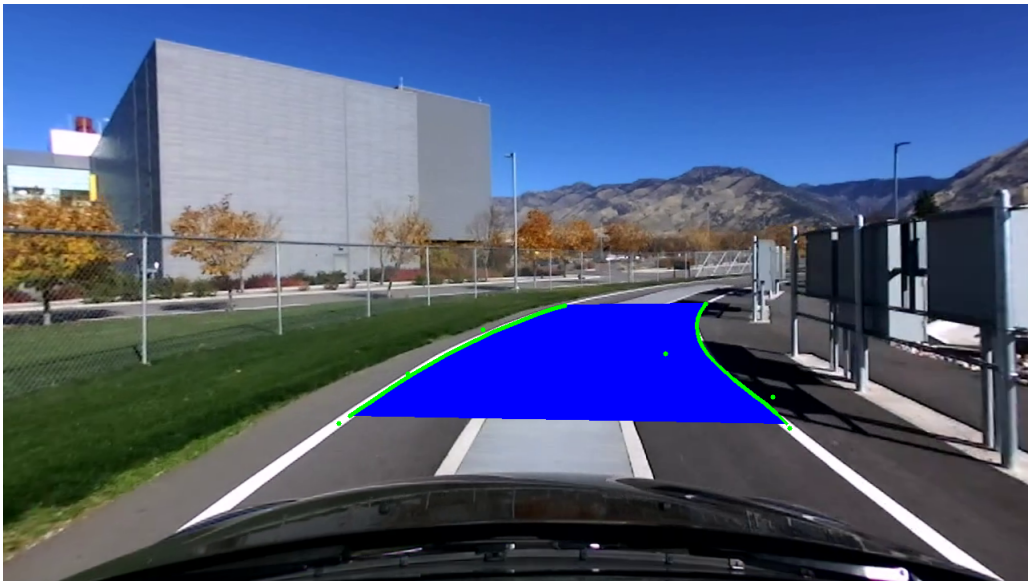

Fig. 4.17: Original image showing lane detection. Detected lanes are back-transformed through IPM and mapped to the image frame. Resulting lane markers are indicated in green and the drivable area of the road is shown in blue

4.2 Vision-Based Control

The vision-based controller receives a desired trajectory from the lane detection algorithm. The path it is designed to follow is the center of the two-lane road detected. Control of the vehicle can be separated into lateral and longitudinal components. For the purposes of this thesis longitudinal control simplified to constant velocity and the low-level longitudinal controller developed in Chapter 3 suffices. The lateral motion of the vehicle is the main concern when attempting to center between two lane markers. The following sections will discuss the lateral kinematics representing the vehicle's motion and the non-linear control law used to develop the lateral controller and are based on the work of Sotelo [47].

The objective of lateral control is to keep the vehicle centered between5 lane markers on either side. A vehicle must be able to stay within the lanes on straight roads as well as when entering curved segments. To accomplish this, the control algorithm must be able to anticipate curvature. Lateral and orientation errors (d_e and θ_e) are used to keep the vehicle within the lanes. The problem can be seen visually in Fig. 4.18. The vehicle is traveling with velocity v and position (x, y) centered on the rear axis. The vehicle is oriented with

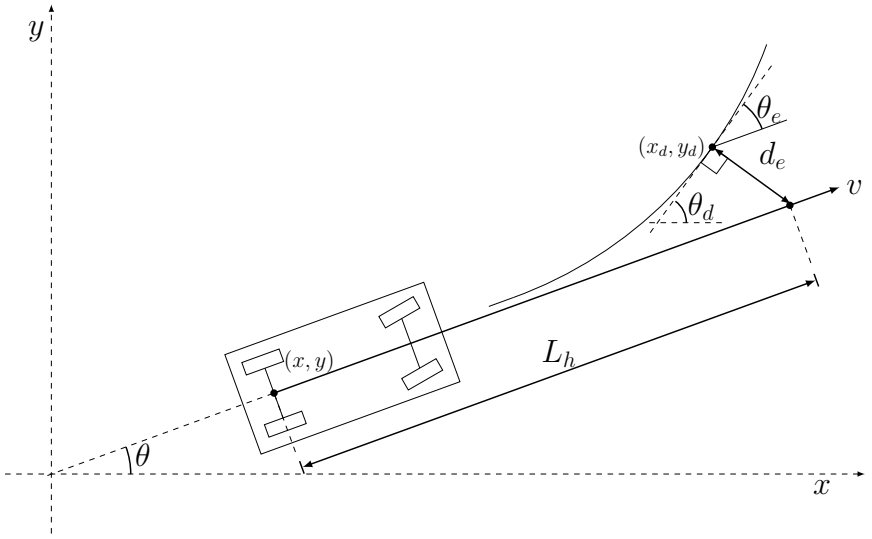

Fig. 4.18: Control parameters used as input to non-linear controller

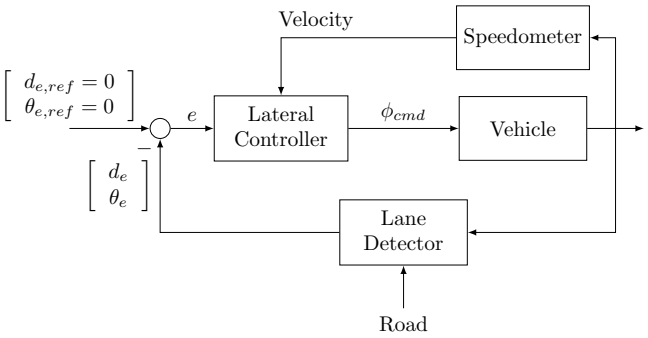

Fig. 4.19: Block diagram of control structure

a heading θ. A control point is established at a lookahead distance L_h in the direction of velocity of the vehicle. The point (x_d, y_d) is the nearest along the desired trajectory to the control point at L_h. The distance between the two points is denoted by d_e. This distance runs perpendicular to a tangent trajectory at (x_d, y_d) with an angle θ_e. The difference between the angle of vehicle travel and the desired heading tangent on the path is θ_e.

Fig. 4.19 shows the design of the control architecture. The vision algorithm senses the lane markers on the road and creates a desired trajectory along the center lane. The purpose of the lateral controller is to minimize the lateral and orientation errors d_e and θ_e. The controller receives these errors and the current speed of the vehicle from it's on-board speedometer.

4.2.1 Kinematic Model

To dynamically model the lateral and offset errors of the vehicle, an understanding of its kinematic behavior is needed. An Ackermann steering model is used, as shown in Fig. 4.20. Assuming that the two front wheels turn slightly differentially, a vehicle with wheelbase L and wheel angle ϕ will travel in a circle with radius R. Let the instantaneous curvature along the trajectory path $k(t)$ be defined as

$$k(t) = \frac{1}{R(t)} = \frac{\tan\phi(t)}{L} = \frac{\mathrm{d}\theta(t)}{\mathrm{d}s} \tag{4.12}$$

where s is path length and θ is the vehicle orientation in the global frame of reference. Orientation will change with time as a function of velocity v:

$$\dot{\theta} = \frac{\mathrm{d}\theta}{\mathrm{d}t} = \frac{\mathrm{d}\theta}{\mathrm{d}s} \cdot \frac{\mathrm{d}s}{\mathrm{d}t} = k(t) \cdot v(t) = \frac{\tan\phi(t)}{L} \cdot v(t). \tag{4.13}$$

Global position and orientation variables (x, y, θ) can by dynamically modeled with the commonly used equations

$$\dot{x} = \frac{\mathrm{d}x}{\mathrm{d}t} = v(t)\cos\theta(t), \quad \dot{y} = \frac{\mathrm{d}y}{\mathrm{d}t} = v(t)\sin\theta(t), \quad \dot{\theta} = \frac{\mathrm{d}\theta}{\mathrm{d}t} = v(t)\frac{\tan\phi(t)}{L}. \tag{4.14}$$

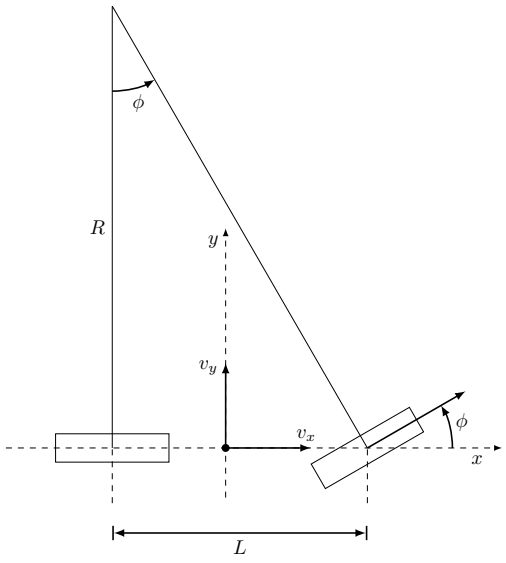

Fig. 4.20: Ackermann steering model

The lateral and orientation errors as shown in Fig. 4.18 and are calculated by

$$d_e = -(x + L_H \cos\theta - x_d)\sin\theta_d + (y + L_H \sin\theta - y_d)\cos\theta_d, \quad \theta_e = \theta - \theta_d. \tag{4.15}$$

The dynamic model of these errors is computed by taking their time derivative. In this case, x_d, y_d, and θ_d are inputs from the vision algorithm and are treated as constant between iterations of the control sequence:

$$\dot{d_e} = v\sin\theta_e + \frac{vL_h}{L}\cos\theta_e \tan\phi, \quad \dot{\theta_e} = \frac{v\tan\phi}{L}. \tag{4.16}$$

4.2.2 Non-linear Control Law

To keep the vehicle properly centered within the lane, the controller's objective is to track a reference trajectory. In particular, it attempts to minimize the lateral error d_e and orientation error θ_e. To accomplish this, the controller is designed to make use of chained systems theory [48]. The general theory is extended and applied to create a stable non-linear

controller for vehicles which follow an Ackermann steering model. A common method for control of non-linear systems is tangent linearization. This approach is not used as it is only accurate around a local linearized configuration and may be far from the path's initial conditions. To convert the non-linear system in Equation 4.16 into a desired linear one, a change of variables is used. The use of chained form allows for a nearly linear system for which linear control principles can be leveraged. Using state diffeomorphism and a change of control variables, d_e and θ_e can be transformed to

$$Y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \Theta(X) = \begin{bmatrix} d_e \\ \tan\theta_e \end{bmatrix}, \quad W = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \Upsilon(U) = \begin{bmatrix} v\cos\theta_e + \frac{vL_h\cos^2\theta_e\tan\phi}{L\sin\theta_e} \\ \frac{v\tan\phi}{L\cos^2\theta_e}. \end{bmatrix}. \tag{4.17}$$

New states y_1 and y_2 with control variables w_1 and w_2 are defined in chained form and create a mapping from the original states, d_e and θ_e. They are invertible so the original values can be obtained as long as v is not zero (the vehicle is moving) and θ_e is not $\pi/2$ (the vehicle's orientation in not perpendicular to the trajectory). Time derivatives of the new states are shown in Equation 4.18 and demonstrate chained form. The vehicle model is rewritten as

$$\dot{y_1} = \dot{d_e} = v\sin\theta_e + \frac{vL_h}{L}\cos\theta_e\tan\phi = w_1 y_2, \quad \dot{y_2} = \frac{\mathrm{d}(\tan\theta_e)}{\mathrm{d}t} = \frac{1}{\cos^2\theta_e}\cdot\dot{\theta_e} = \frac{v\tan\phi}{L\cos^2\theta_e} = w_2. \tag{4.18}$$

A differentiation with respect to distance is used to create a control law which is not velocity-dependent. A new variable ξ is defined that is related to the path length of the vehicle parallel to the reference trajectory tangent:

$$\xi = \int \left(v cos\theta_e + \frac{vL_h\cos^2\theta_e\tan\phi}{L\sin\theta_e} \right) \mathrm{d}t. \tag{4.19}$$

The state time derivative can be expressed in terms of the state variables with respect to ξ times ξ with respect to time:

$$\dot{y_1} = \frac{\mathrm{d}y_1}{\mathrm{d}t} = \frac{\mathrm{d}y_1}{\mathrm{d}\xi} \cdot \frac{\mathrm{d}\xi}{\mathrm{d}t} = y_1' \cdot \dot{\xi}, \quad \dot{y_2} = \frac{\mathrm{d}y_2}{\mathrm{d}t} = \frac{\mathrm{d}y_2}{\mathrm{d}\xi} \cdot \frac{\mathrm{d}\xi}{\mathrm{d}t} = y_2' \cdot \dot{\xi}. \tag{4.20}$$

The derivative of the vehicle model is taken with respect to ξ rather than time, yielding

$$y_1' = \frac{\dot{y_1}}{\dot{\xi}} = \tan\theta_e = y2, \quad y_2' = \frac{\dot{y_2}}{\dot{\xi}} = \frac{\tan\phi}{L\cos^3\theta_e + L_h(\frac{\cos^4\theta_e \tan\phi}{\sin\theta_e})} = \frac{w_2}{w_1} = w_3 \tag{4.21}$$

where y_1' is the derivative of y_1 with respect to the spatial variable ξ and y_2' is the derivative of y_2 with respect to ξ. The system now has a linear form and traditional linear control techniques can be applied. A classical PD approach is used to set the value of the new control input w_3. In doing so, many degrees of freedom are lost, but section 6.3 demonstrates that the design is sufficient for the purpose. The new control state is set to

$$w_3 = -K_d y_2 - K_p y_1, \quad (K_d, K_p) \in \Re^{+2}. \tag{4.22}$$

Combining Equations 4.21 and 4.22 and solving for y_1' gives the following second order equation:

$$y_1'' + K_d y_1' + K_p y_1 = 0 \tag{4.23}$$

where a linear dynamic behavior is exhibited by y_1 with respect to ξ. As ξ increases, the lateral and orientation errors (d_e, θ_e) tend to zero

$$\lim_{\xi\to\infty} d_e = \lim_{\xi\to\infty} \theta_e = 0. \tag{4.24}$$

This holds true as long as the integral term ξ is increasing. To assure this, the vehicle's velocity v must be greater than zero and its orientation error θ_e must be stay within the bounds of $-\pi/2$ and $\pi/2$. In practice, it is not difficult to abide by these restrictions.

The desired steering angle ϕ is obtained by substituting the control strategy for w_3 in Equation 4.22 into Equation 4.21 and solving for

$$\phi = \arctan\left[\frac{-L\sin\theta_e\cos^3\theta_e(K_d\tan\theta_e + K_p d_e)}{\sin\theta_e + L_h\cos^4\theta_e(K_d\tan\theta_e + K_p d_e)}\right]. \tag{4.25}$$

In real-world driving, saturation of the steering angle ϕ is desired to account for the physical limitations of the steering system. A sigmoid function is applied to Equation 4.25 to keep it within the desired bounds as outlined by

$$\phi = \arctan\left[-KL\cos^3\theta_e \cdot \frac{1 - exp^{-K(\sin\theta_e(K_d\tan\theta_e + K_p d_e)/\sin\theta_e + L_h\cos^4\theta_e(K_d\tan\theta_e + K_p d_e))}}{1 + exp^{-K(\sin\theta_e(K_d\tan\theta_e + K_p d_e)/\sin\theta_e + L_h\cos^4\theta_e(K_d\tan\theta_e + K_p d_e))}}\right]. \tag{4.26}$$

The steering wheel angle ϕ is saturated to ϕ_{max} through the variable K by

$$\phi_{max} = \pm\arctan(-KL). \tag{4.27}$$

Because the physical limitation of the vehicle's steering is $\phi_{max} = \pm(\pi/6)$, K is chosen such that it does not exceed this limit,

$$K = \frac{\tan(\pi/6)}{L}. \tag{4.28}$$

In Equation 4.23, it can be seen that variable y_1 dynamically follows the response of a second order linear system. Because of the application of the sigmoid function, the system is not truly linear, but can still be approximated as such. Thus, its coefficients K_d and K_p can be related to the parameters of a second order linear system ζ (damping coefficient) and ω_n (natural frequency):

$$\omega_n = \sqrt{K_p}, \quad \zeta = \frac{K_d}{2\sqrt{K_p}}. \tag{4.29}$$

Similarly, system overshoot M_p and settling distance $d_{s|2\%}$ can be designed using the properties of a second order linear response as shown in Equation 4.30. The system error

dynamics are described as a function of space variable ξ rather than time:

$$M_p = exp\left(\frac{-\zeta\pi}{\sqrt{1-\zeta^2}}\right), \quad d_{s|2\%} = \frac{4}{\zeta\omega_n}. \tag{4.30}$$

System constants K_p and K_d are designed with these parameters in mind. The desired maximum overshoot M_p should not exceed 10%. The settling distance d_s is determined based on settling time t_s and velocity v as shown in Equation 4.31. Given $t_s = 7s$, settling distance gives the following:

$$d_s = t_s \cdot v = 7v. \tag{4.31}$$

The value of K_d can be found from Equations 4.29 and 4.30 to be

$$K_d = \frac{8}{d_s} = \frac{8}{7v}. \tag{4.32}$$

In the same fashion, an expression for the damping coefficient ζ can be found from Equations 4.29 and 4.30:

$$\zeta = \sqrt{\frac{1}{[\pi/\ln 0.1]^2 + 1}} = \frac{K_d}{2\sqrt{K_p}} = \frac{4}{d_s\sqrt{K_p}}. \tag{4.33}$$

The constant K_p is found from Equation 4.33 giving

$$K_p = \left[\frac{4}{d_s\zeta}\right]^2 = \left[\frac{0.9666}{v}\right]^2. \tag{4.34}$$

At a nominal velocity of $v = 6.71m/s$ (15mph), the resulting gains are $K_p = 0.0208$ and $K_d = 7.6635$.

CHAPTER 5

AUTOMATION PLATFORM OVERVIEW

This chapter examines the implementation details of hardware and software components. Links are provided for the open source architecture.

The efforts discussed in this chapter were led by the author of this work and his research partner, Austin Costley [30]. It is important to note the collaboration effort with Austin, and identify his contributions. In particular, Austin was instrumental in PCB design, ROS controller structuring, and presentation of the information in this chapter which was prepared for a coauthored journal submission. It is included in amended form in this thesis for completeness.

5.1 Platform Overview

A versatile and robust platform is required to enable full-sized autonomous vehicle research. The platform was designed to enable vehicle automation for both the CAN injection and the sensor emulation approaches discussed in Section 2.1. For the CAN injection approach the platform was able to monitor the CAN bus and inject the desired packets, whereas for the sensor emulation approach, the platform required access to the output lines of the sensors to be emulated. In order to proceed to autonomy, the platform had the ability to sense the environment, determine vehicle location, communicate with the vehicle, and monitor the CAN bus. A computer running Ubuntu and the Robot Operating System (ROS) was used to communicate between the platform architectures. The computer was connected to a microcontroller to allow communication with the vehicle. Fig. 5.1 shows a diagram of the autonomous system, including the ROS software architecture, hardware connections to devices, and the vehicle interfacing hardware.

A ROS-based platform was chosen for ease of use, modularity, and sensor interfacing packages. ROS is an open source framework that encourages collaboration between

researchers. People can contribute to the ROS effort by creating software packages that interface with common sensors and provide tools for development. For example, an open-source software package for ROS was provided by Stereolabs to interface with the ZED camera [49], which helped expedite development time for this project. The ROS architecture components used in this project are packages, nodes, and topics. A ROS package is a collection of executable files used to complete a task. Generally packages are used to compartmentalize similar parts of a project. ROS nodes are the executable files in a ROS package that can be written in C/C++ or Python. A ROS topic is a way to transfer data between nodes. Any node can publish data to a topic, and any node can subscribe to that topic to receive the data. In this sense, the ROS topic acts as a bus to transfer data.

The following subsections detail the Interfacing Architecture, Sensing Architecture, and the Computational Architecture of the automation platform.

5.1.1 Interfacing Architecture

Interfacing devices are critical to the success of vehicle automation as they provide a way to send commands to the vehicle, and monitor the vehicle for feedback. A PCB (shown in Fig. 5.2) was designed to provide a connection between the microcontroller and the vehicle. The following subsections discuss the microcontroller and associated hardware, and the other interfacing devices used for this platform.

Microcontroller and Associated Hardware

The TI TM4C129XL evaluation kit was the chosen microcontroller platform because it offered multiple CAN bus interfaces allowing for a combination of CAN injection and sensor emulation from the same board [36]. The microcontroller receives input from the computer through a UART module. After performing the appropriate computations, PWM signals are generated and appropriate DC voltage levels are determined for vehicle input. The control signals pass through a variety of circuit components to prepare the signals for vehicle injection. The signals are terminated at solid state relays that select either the original vehicle signal, or the generated signal to be sent to the vehicle. The user determines

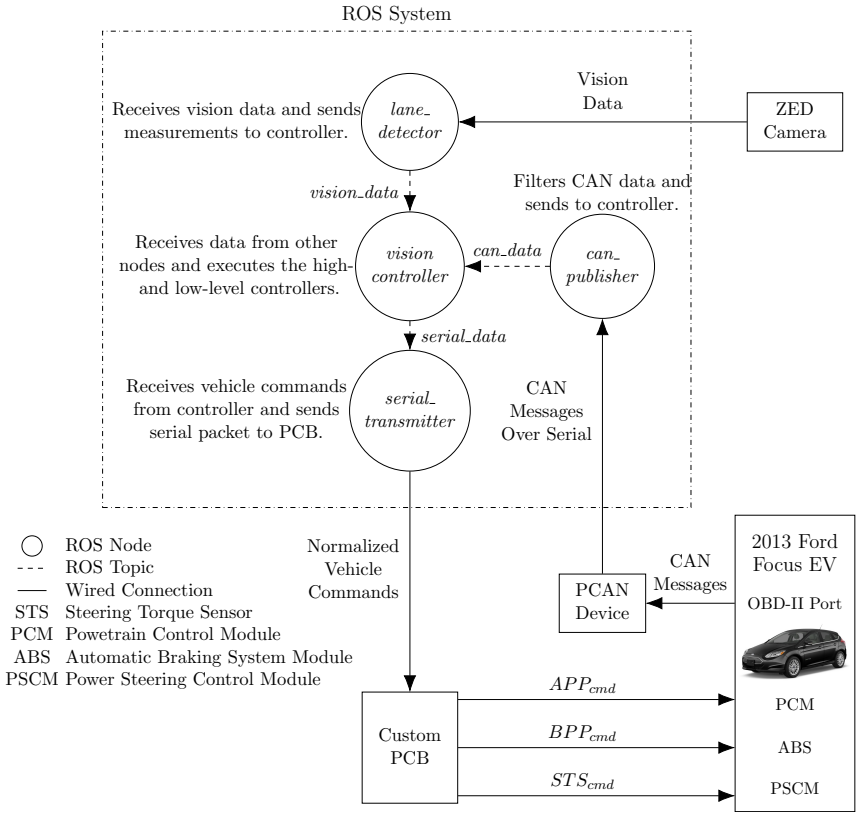

Fig. 5.1: Platform diagram including ROS system, hardware device interfaces, and vehicle interfaces.

which signal is sent based on a mode switch input to the microcontroller.

The sensor emulation approach requires four PWM signals to be generated by the microcontroller. The signals are passed through a level shifter to shift the amplitudes from 3.3 V to 5 V, and then sent through operational amplifiers in a voltage follower configuration to help drive the signals. The PWM signals are then terminated at the normally opened terminals of solid state relays.

The accelerator pedal input is generated by a Digital to Analog Converter (DAC) that receives an I2C signal from the microcontroller. The DAC converts the digital communication to an analog voltage level, and sends it to an active filter IC to clean the signal and perform a gain two operation to provide the two output signals. The generated signals then terminate at the normally open terminals of the solid state relays.

Another key feature of the PCB is the safety circuit. Next to the driver there is a

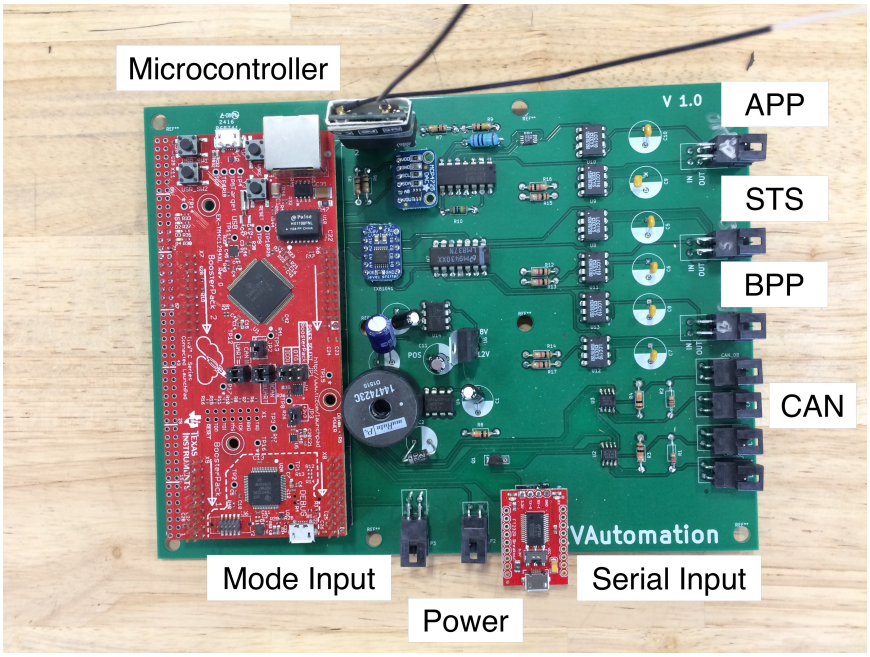

Fig. 5.2: Custom circuit board used to interface with the vehicle CAN bus and generate the signals required for the sensor emulation approach.

mode switch and an Emergency Stop button. The mode switch allows the driver to switch between Manual Driving Mode and Autonomous Mode. The power and solid state relay control signals are routed to the front of the vehicle so the safety driver can switch between operating modes or press the Emergency stop button to prevent power from reaching the circuit. The vehicle's original sensor signals are connected to the normally closed terminals of the solid state relays, so removing the power returns the vehicle to Manual Mode. Power to the circuit is provided by a NewMar DC Uninterruptible Power Supply (UPS) which connects to the 12 V car battery, and provides safe and stable voltage levels for the circuit operations [50]. The NewMar UPS also has an internal backup battery. The voltage level is stepped down to ± 8 V, 5 V and 3.3 V, and distributed across the custom PCB. The UPS is shown in Fig. 5.3.

Other Interface Devices

The Peak Systems PCAN device was chosen to monitor CAN traffic [35]. The PCAN device can connect directly to the vehicle's OBD-II port, and provides serial output over

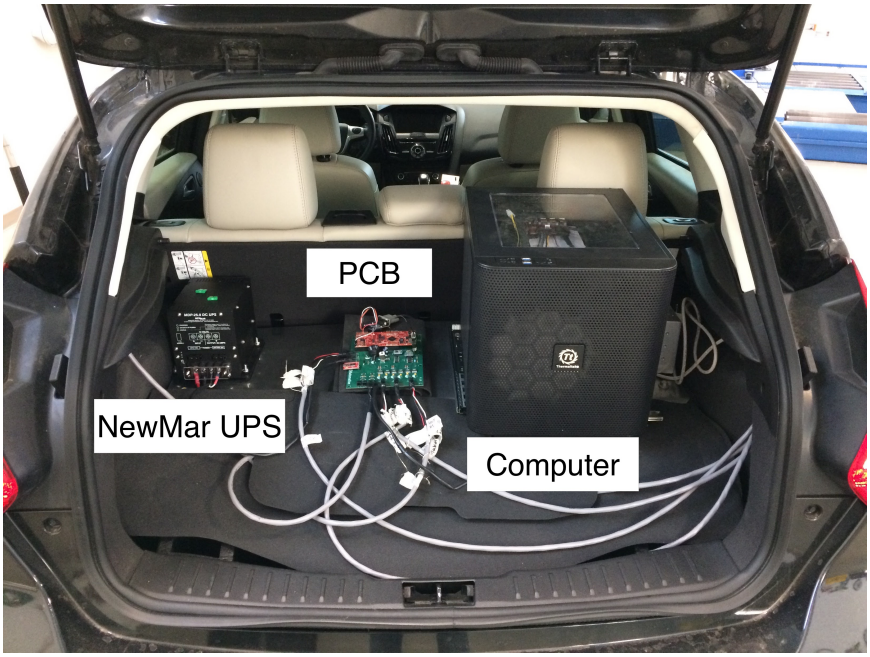

Fig. 5.3: Trunk of the 2013 Ford Focus EV with hardware setup. *Left:* The NewMar UPS that connects to the vehicles 12 V battery and supplies power to the circuit and computer. *Center:* The custom PCB for interfacing with the vehicle CAN bus and sensors. *Right:* Computer running Linux and ROS, connected to the ZED Camera, PCAN Device, and PCB.

USB. Every message on the connected CAN bus is received and sent serially to the computer. The user can determine which CAN data packets are important to operation, and ignore the rest. One approach to determine necessary CAN data packets is detailed in Section 2.1. Instead of using the CANalyzer system to monitor CAN traffic, the PCAN device can be used to record CAN traffic for a desired event (e.g. vehicle acceleration). The CAN data can be replaced section by section until a message or set of messages is isolated. Additional information on the use of the PCAN device for system feedback is given in Section 5.1.3.

A USB-to-serial device was used between the microcontroller and computer to enable communication. The control system determines the appropriate inputs to the vehicle and sends the commands to the microcontroller. The device receives the signal from the USB port and sends it to a UART module on the microcontroller. More information about the microcontroller and control system is given in Section 5.1.3.

The TI SN65HVD230 CAN Transceiver Breakout Board [51] [37] was used to connect

the microcontroller to the CAN bus. This board provided a direct connection with the vehicle CAN bus that can be used for monitoring and injection.

5.1.2 Sensing Architecture

The 2013 Ford Focus EV has an array of sensors on the vehicle that monitor everything from wheel speed to tire pressure. However, the vehicle does not have high precision wheel encoders, an RTK-GPS receiver, or inertial measurement sensors (IMU's) that could be useful for vehicle automation. The sensor data is typically received by a module and sent on the CAN bus. Using the PCAN device described in Section 5.1.1, the on-board sensor information can be provided to the rest of the automation platform. For autonomous driving, the accelerator pedal position sensor, brake pedal position sensor, steering wheel angle sensor, and vehicle speed data are used for feedback in the low-level controllers.

The sensor data broadcast on the CAN bus does not provide the information in empirical units, and sometimes the data is masked with other signals. An important aspect to the sensing architecture is the conversion from CAN bus messages to useful units. These conversions could be found experimentally for each message found on the CAN bus, but for the purposes of this platform the vehicle speed was the only message converted to empirical units (MPH). The vehicle speed is found on the message with arbitration ID `0x75` on bytes 7 and 8, and is represented by a 16-bit value where byte 7 is the upper byte and byte 8 is the lower byte. When the vehicle was not moving the vehicle speed was represented as `0xB0D4` on the CAN bus. The decimal representation of this constant, 45,268, is subtracted from the 16-bit vehicle speed to align the 0 MPH value. The vehicle was driven with the RTK-GPS units to provide a reference for the vehicle speed, and it was determined that the CAN value would then need to be divided by 54 to achieve an accurate measure of speed. This process is summarized by

$$v_{mph} = \frac{(b7 << 8) + b8 - 45268}{54}, \tag{5.1}$$

where $b7$ and $b8$ are the integer representations of bytes 7 and 8 from the CAN message with arbitration ID `0x75`, and the << operator represents a left bit shift.

ZED Camera

The ZED camera by Stereolabs was chosen for vision sensing. It features a stereo camera with a 100° wide field of view. While the camera is capable of frames rates up to 60 frames per second at 1080p resolution, only 15 frames per second at 720p were required due to image processing and control loop timing. The ZED camera outputs monocular images as well as depth information. Stereolabs has developed a ROS wrapper in conjunction with the hardware. Image frames, a depth map, and visual odometry data are published by the wrapper. Because of processing simplicity, only monocular images are used for lane detection through computer vision. The use of depth information is anticipated in the future.

5.1.3 Computational Architecture

The computational architecture includes the code required to combine sensor information, controller commands, and prepare command insertion. The two computational platforms used in this system are the microcontroller and ROS. These platforms are discussed in the following sections and code for these platforms can be found at https://github.com/rajnikant1010/EVAutomation.

Microcontroller Software

The code for the TI TM4C129XL was written in C and took advantage of the built in functionality of the TivaWare Peripheral Driver Library from Texas Instruments [52]. Table 5.1 shows the peripherals used and their functionality. The following paragraphs discuss the microcontroller code.

The microcontroller receives a serial packet from the computer in the form shown in Fig. 5.4. The first byte is always `0xFA`, the second byte gives the number of bytes in the payload, the payload contains the steering, braking, and acceleration commands to be sent

to the vehicle, and the last two bytes is a 16-bit Cyclic Redundancy Check (CRC) using the CRC16-CCITT algorithm to ensure data integrity. Once received, the CRC is calculated to ensure correct data, and the payload values are stored in appropriate variables. All input commands are normalized between zero and one. For PWM signals, the normalized value represents the duty cycle of the signal, where 0.5 represents 50% duty cycle.

OpenCV Library

The OpenCV library was chosen for the computer vision and lane detection algorithm because of it's open-source community support, cost efficiency, and speed. OpenCV was designed for computational efficiency and places a strong focus on real-time applications [53]. Developers can work in C/C++ or Python. OpenCV image structures and filtering in C++ were used to process lane data. Splines and debugging information were drawn and displayed using methods available in the OpenCV library. The OpenCV matrix structures proved useful when making calculation on images and data with multiple dimensions.

ROS Architecture

The ROS architecture consists of a series of packages, nodes, and topics [44]. A ROS package can be used to modularize code. For example, the four packages used for this project were *zed-ros-wrapper* (camera), *focus_serial* (serial communication), *pcan* (CAN interface), and *focus_vision* (high- and low-level control). Each of these packages has at least one node and publishes or subscribes to certain topics. A program file (either C/C++ or Python)

Table 5.1: Microcontroller Peripherals Table

Port	**Pin**	**Type**	**Purpose**
GPIO_F	2	PWM	Steering signal 1
GPIO_F	3	PWM	Steering signal 2
GPIO_F	1	PWM	Brake signal 1
GPIO_G	1	PWM	Brake signal 2
GPIO_K (I2C4)	6	I2C_SCL	Acceleration I2C SCL line
GPIO_K (I2C4)	7	I2C_SDA	Acceleration I2C SDA line
GPIO_C	4	Logic	Mode select signal from user
GPIO_C	5	Logic	Mode signal to system

`0xFA`	# of bytes (n)	Payload: Steering Torque, BPP, and APP Commands	CRC16-CCITT
1-byte	*1-byte*	*n-bytes*	*2-bytes*

Fig. 5.4: Serial message structure for communication with the microcontroller. The serial communication sends commands to the vehicle emulating the APP, BPP, and steering torque sensors. The second byte indicates the number of bytes in the payload, *n*.

is written for each node and when a node publishes information to a topic, other nodes can subscribe to that topic and receive the information that was published. The following paragraphs will discuss each of the ROS packages, nodes, and topics used for this system.

The *zed-ros-wrapper* package has only one node. This node initializes the ZED camera and publishes to a variety of topics. These topics include left and right images in color and black and white, a depth map, and a visual odometry message. Only the left color image was used for the project. It is published at 30Hz and is rebroadcast at 15Hz by a native ROS `throttle` node. The message is throttled because a slower frequency is called for by the lane detection algorithm.

The *pcan* package has one node called *can_publisher* that receives CAN data from the PCAN device and parses the requested data. The CAN data is translated to useful units for the given control strategy, and published to a ROS node called *can_data*. The *can_data* topic could provide as much CAN information as the user would like. For the purposes of automating this vehicle, the CAN data of interest is vehicle speed in MPH and steering wheel angle.

The *focus_vision* package has six nodes:

1. *LaneDetector64*: This node subscribes to the raw image data published by the ZED ROS wrapper. Lane detection is performed on the images using OpenCV, and control parameters d_e and θ_e are extracted. The node publishes the control information to the *sotelo_de* and *sotelo_thetae* topics to be used by the vision controller.

2. *vision_controller*: This node subscribes to the *sotelo_de*, *sotelo_thetae* and *can_data* topics. Using these parameters, it implements the high-level control algorithm outlined in Section 4.2. It publishes low-level control commands to the *desired_velocity* and *desired_velocity* topics for use by the lateral and longitudinal controllers.

3. *lateral_controller*: This controller node subscribes to the *desired_angle* topic. From the desired angle, it calculates a commanded steering value using the low-level PI controller discussed in Section 3.1.3. This vehicle command is published to the *lateral_command* topic.

4. *longitudinal_controller*: This controller node subscribes to the *desired_velocity* topic. From the desired velocity, it calculates a commanded brake or accelerator value using the low-level PI controller discussed in Section 3.1.3. This vehicle command is published to the *longitudinal_command* topic.

5. *controller*: This node subscribes to the lateral and longitudinal command data published on the *lateral_command* and *longitudinal_command* topics. The accelerator pedal position, brake pedal position, and steering torque duty cycle commands are received separately and then packaged and published to the *serial_data* topic.

6. *print_vision*: This logging node subscribes to all topics and prints a CSV log. The data printed is analyzed in MATLAB.

The *focus_serial* package has one node called *serial_transmitter*. This node subscribes to the *serial_data* topic and sends the accelerator pedal position, brake pedal position and steering torque duty cycle information to the microcontroller. Before the data is sent, a Cyclic Redundancy Check (CRC) is performed and a checksum is added to the serial message. The microcontroller checks the CRC to verify the accuracy of the data before sending the requested commands to the vehicle.

CHAPTER 6

RESULTS

This chapter details the results for autonomous driving. Experimental results of the low-level controller are presented, followed by results of high-level vision-based control.

The efforts discussed in this chapter were led by the author of this work and his research partner, Austin Costley [30]. It is important to note the collaboration effort with Austin, and identify his contributions. In particular, Austin was instrumental testing the low-level controller and the presentation of it's results in this chapter which were prepared for a coauthored journal submission. The results are included in this thesis for completeness.

Experiments were conducted to determine the results of the autonomous vehicle platform. The low level controllers were tested with given desired steering angles and velocities. The low level steering controller was improved by implementing a deadband compensation algorithm. The results for the low-level controllers are given in Section 6.1. A video of the low-level control can found at https://youtu.be/NpAUcNh4QUY.

After verification of the low-level controllers, experiments were conducted at a higher level using lane detection and the vision-based controller discussed in Chapter 4. Results of the vision-based controller are shown in Section 6.3. A video of autonomous vision-based driving can be found at https://youtu.be/7ohWIwb6KfM.

6.1 Low Level Controller

The low level controllers provide speed and steering wheel angle control through the user input signal. For the steering controller, the steering wheel torque sensor signal was used to change the position of the steering wheel. As discussed in Section 3.1, the control loop was designed such that, given a desired angle, the controller would change the steering torque value until the desired angle was achieved. This controller was tested using a step input and a graph of the result can be seen in Fig. 6.1. The y-axis is the steering wheel

angle as represented by a Hex value on the CAN bus. A desired steering wheel angle of `0x7D0` was used as an input to the controller node of the ROS platform. The step response had a maximum value of `0x891`, representing an overshoot of 9.65%. The resulting time constant of the system was 1.86 seconds. The desired behavior was for the system to be critically damped and have a time constant of 0.33 seconds.

After implementing deadband compensation, the lateral controller improved. Fig. 6.2 shows the step response of the lateral controller with deadband compensation. The maximum value for this response is 2,113, which represents a 5.65% overshoot, and has a time constant of 1.1 seconds.

6.2 Lane Detection

The lane detection algorithm searches for a pair of lane markers as discussed in Section 4.1. Target lanes are white lines which run parallel and at a lane's width around the vehicle. Successful lane detection can be seen in Fig. 6.3 with lane markers highlighted in green and the drivable area between them indicated in blue. It can be noted that the algorithm usually proved successful even under steep curvatures and shadow conditions. However, occasionally lanes would be only partially detected due to shadows or return false positives from the charging gutters in the center of the road as demonstrated in Figs. 6.4 and 6.5.

6.3 Vision-Based Controller

The vision-based controller centers the vehicle in the road based on the position of lane markers. Design of this controller is discussed in Section 4.2. Lanes are detected and a center lane is calculated by the lane detection algorithm outlined in Section 4.1. The vehicle's error is calculated based on a lateral error d_e and orientation error θ_e at a look ahead distance L_h in front of the vehicle as shown in Fig. 4.18. The lateral error d_e is the distance from the look ahead control point to the closest point on the trajectory (x_d, y_d). Tangential to this point along the trajectory is the angle of the path θ_d. The lateral error d_e is perpendicular to the tangent θ_d at this point. The orientation error θ_e is the angle

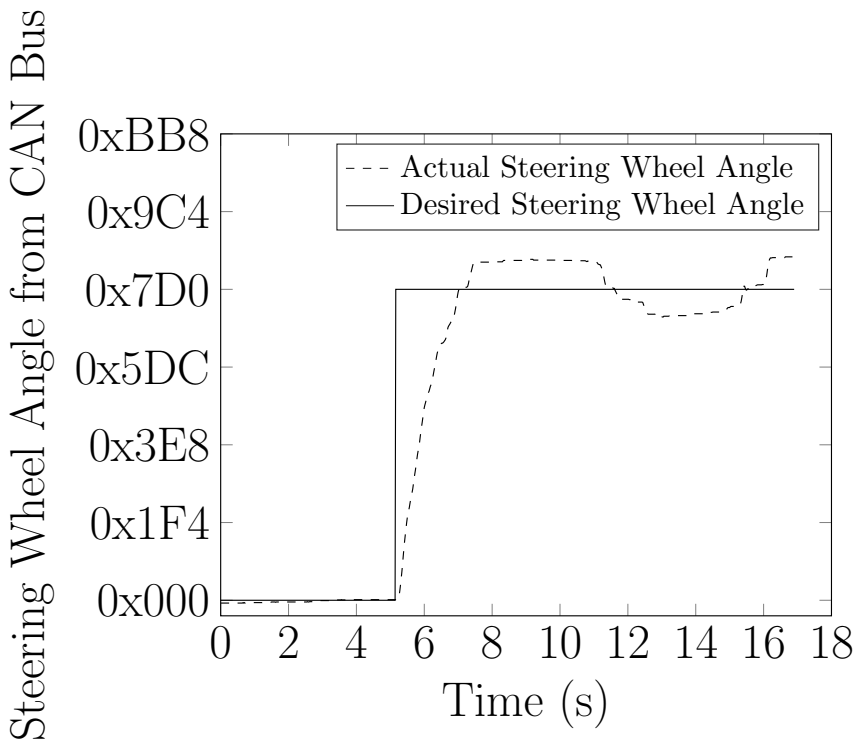

Fig. 6.1: Steering angle step response without deadband compensation.

difference between the vehicle's orientation θ and the tangent to the path θ_d. Experimental results taken from autonomous test track data can be seen for d_e in Fig. 6.6 and θ_e in Fig. 6.7. These errors represent the parameters calculated by the vision algorithm and are not true measures of path error at the vehicle, but rather at the look ahead distance L_h in front of it. True path errors were calculated by taking the middle distance between the two lanes. The average path error is 30cm. Fig. 6.8 shows position data for the center lane and the path the vehicle traveled as measured by RTK GPS. Vehicle path error from the center of the vehicle to the center of the lane is shown in Fig. 6.9.

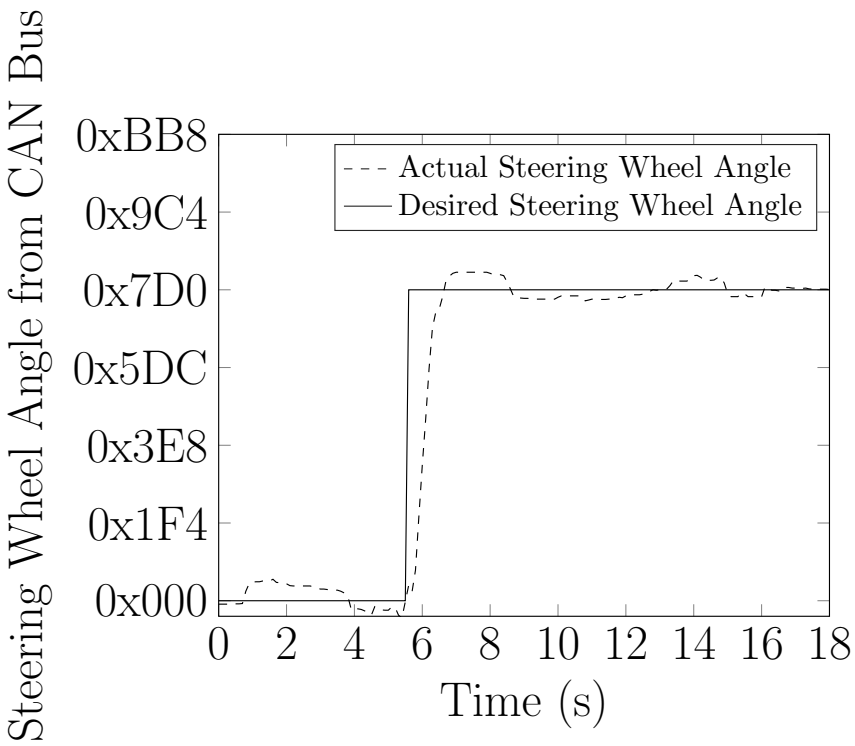

Fig. 6.2: Steering angle step input with deadband compensation.

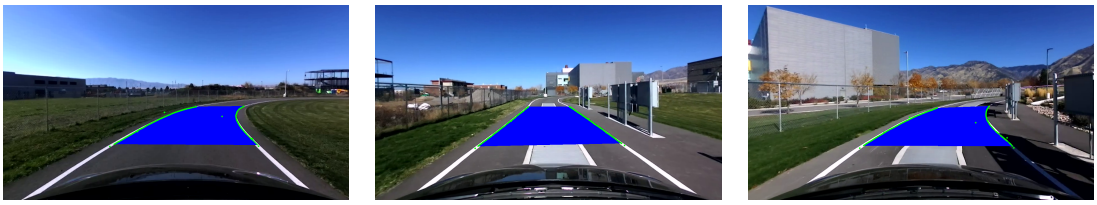

Fig. 6.3: Lane detection frames showing both lanes correctly identified

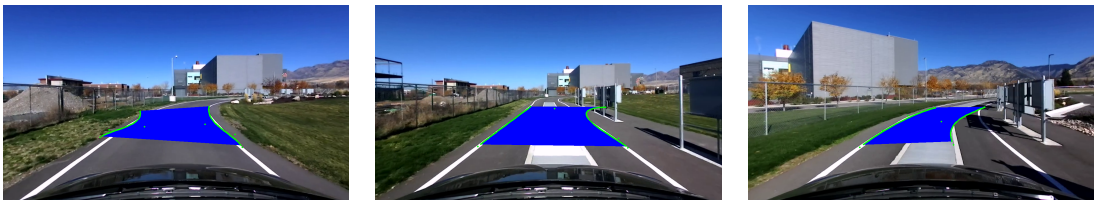

Fig. 6.4: Lane detection frames showing correct detection of one lane and imperfect detection of the other

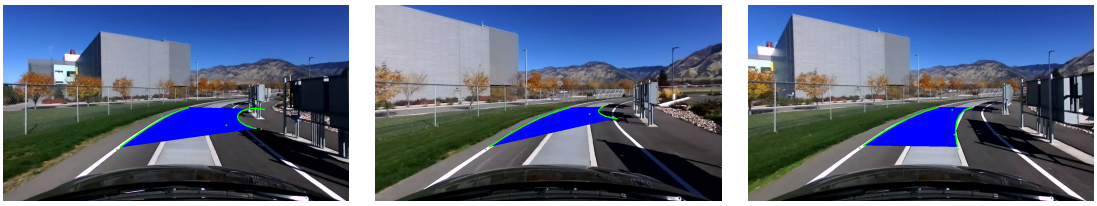

Fig. 6.5: Lane detection frames showing poor detection of lanes or false positives

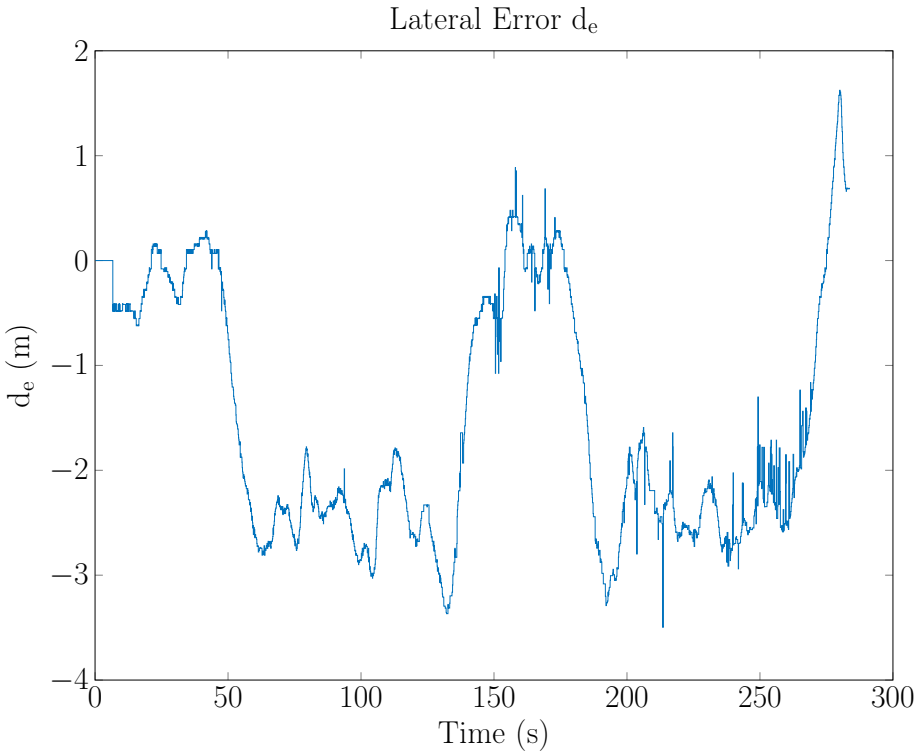

Fig. 6.6: Vision-based controller lateral error

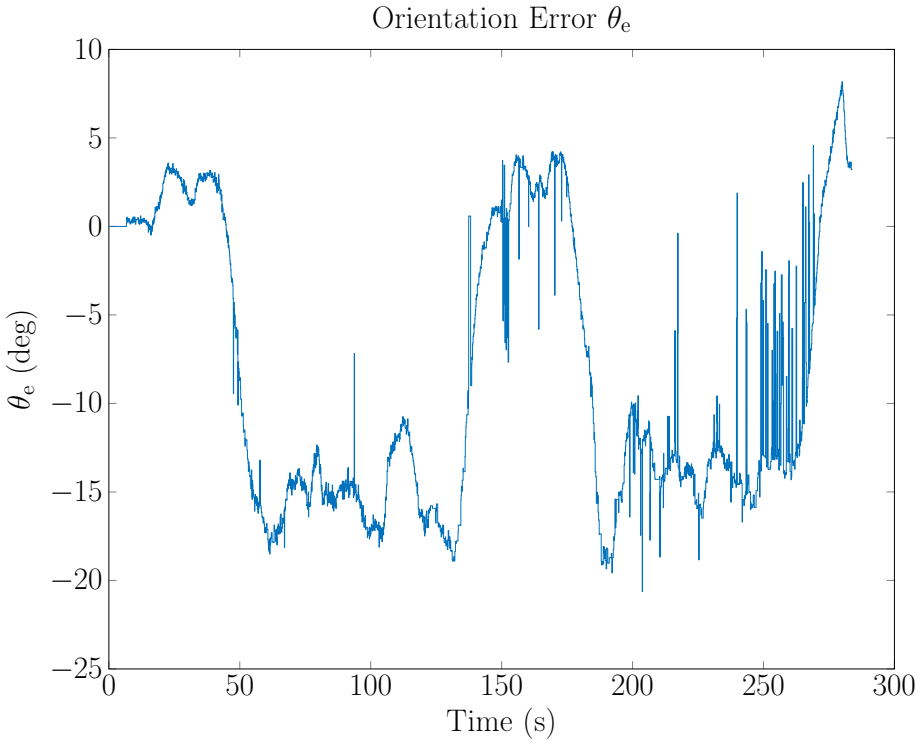

Fig. 6.7: Vision-based controller orientation error

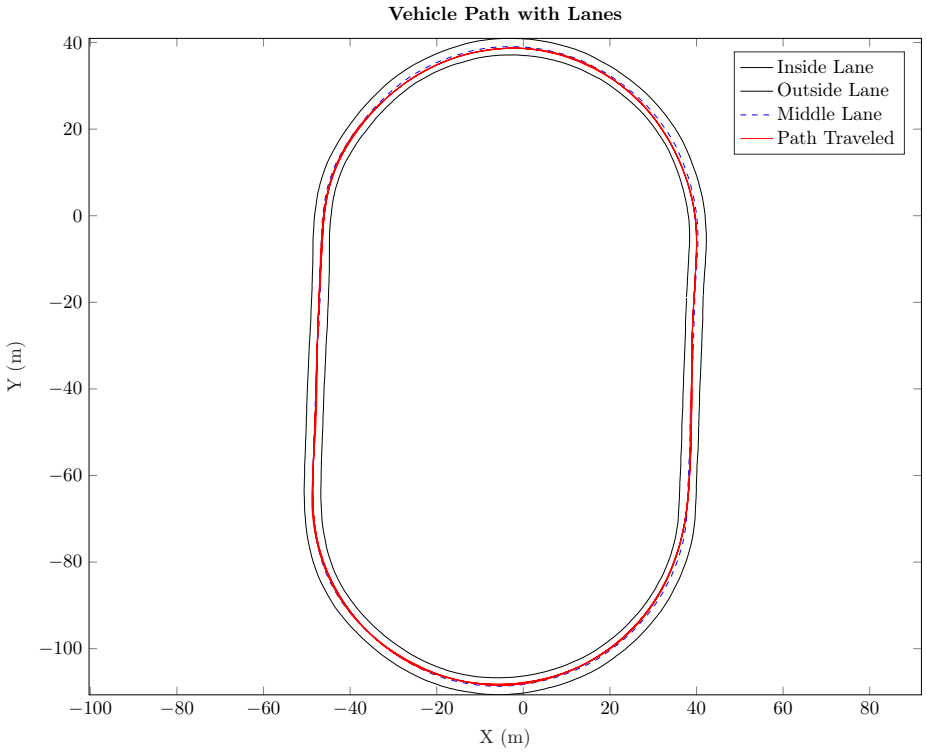

Fig. 6.8: Path error of the vision based controller

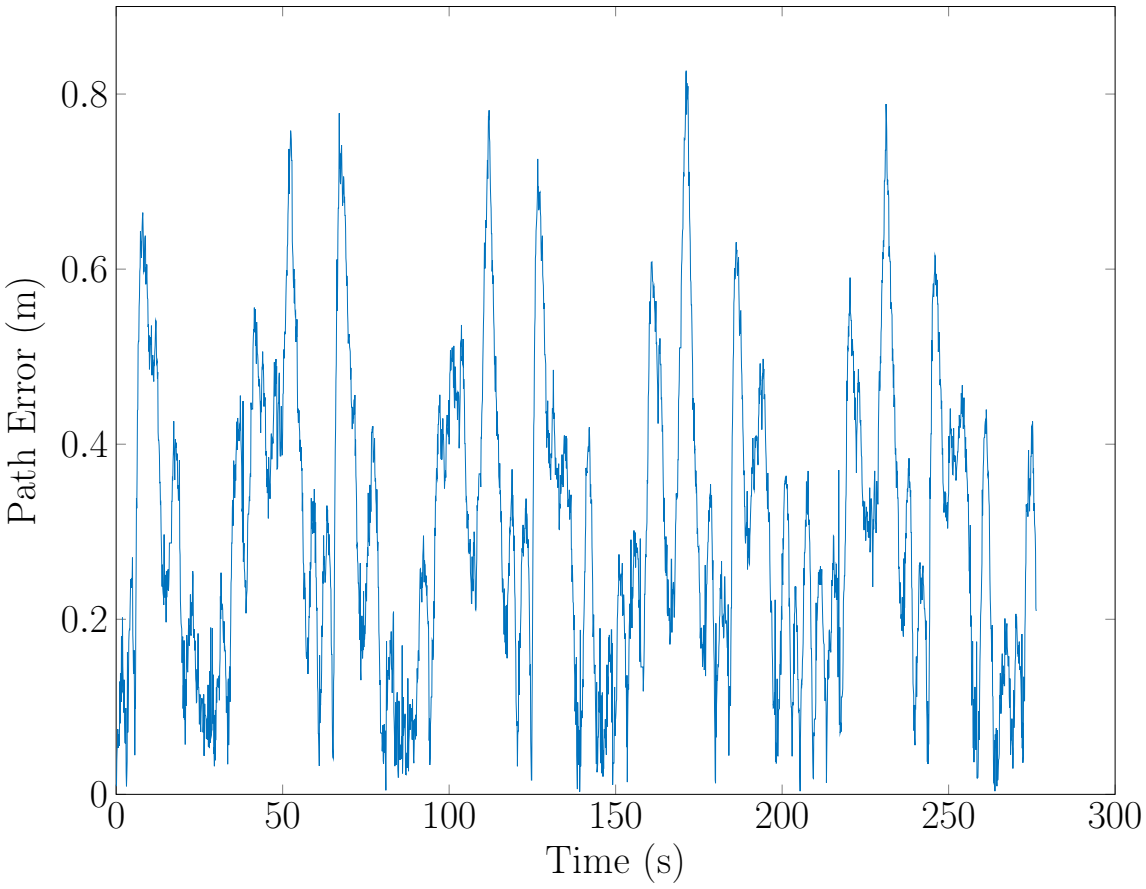

Fig. 6.9: Lateral path error from center lane

CHAPTER 7

CONCLUSION

This thesis presents a platform for automation by vision of a full-size car without the use of external actuators. The complete method for automation from a stock vehicle to lane-based vision control is shown. Chapter 2 discusses methods of reverse engineering low-level control signals and Chapter 3 details the design of their accompanying low-level controllers. Lane detection and vision-based control are discussed in Chapter 4. Finally, Chapter 5 outlines the open source platform used in the implementation of vehicle automation. Experimental results in Chapter 6 validate the design of the previous chapters. This thesis also presents new work on injection over CAN causing vehicle acceleration.

7.1 Limitations and Future Work

The lane detection algorithm has been tuned for the scope of driving at the EVR test track. The lane markers are solid white lines with no breaks and the roadway has no intersections. Implementation of detection of dashed lane marker and lane makers of different colors could be implemented. Robustness is also a concern in lane detection. The algorithm works well in overcast weather, but more poorly in bright, sunny conditions because of reflections cast from bright, straight objects such as poles. Shadows from the nearby charging boxes are also cast on the lane markers during sunny conditions. Future work could be done to increase the robustness of lane detection in more diverse lighting conditions.

The vision-based controller is limited by velocity. Although it is designed to be velocity independent, experimental tests have only been performed at velocities around 15mph. It also does not have the capability of start from stop. Future work could be done to improve the functionality of the vision-based controller with respect to velocity. The current lateral error is around 30cm. The goal of the project is to decrease this error to around only 10cm.

Currently, model identification was performed with first order responses. Using second order systems to model the vehicle could increase longitudinal and lateral accuracy.

REFERENCES

[1] B. Snavely. (2016) Detroit automakers ink deals for self-driving cars. [Online]. Available: http://www.usatoday.com/story/money/cars/2016/05/16/detroit-automakers-ink-deals-self-driving-cars/84438032/

[2] Ford targets fully autonomous vehicle for ride sharing in 2021; invests in new tech companies, doubles silicon valley team. [Online]. Available: https://media.ford.com/content/fordmedia/fna/us/en/news/2016/08/16/ford-targets-fully-autonomous-vehicle-for-ride-sharing-in-2021.html

[3] D. Howard and D. Dai, "Public perceptions of self-driving cars: The case of berkeley, california," in *Transportation Research Board 93rd Annual Meeting*, vol. 14, no. 4502, 2014.

[4] S. Thrun, "Toward robotic cars," *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.

[5] E. D. Dickmanns and A. Zapp, "A curvature-based scheme for improving road vehicle guidance by computer vision," in *Mobile Robots, SPIE Proc. Vol. 727, Cambridge, Mass.*, 1986, pp. 161–168.

[6] K. Kluge, "Extracting road curvature and orientation from image edge points without perceptual grouping into features," in *Intelligent Vehicles '94 Symposium, Proceedings of the*, Oct 1994, pp. 109–114.

[7] A. A. Assidiq, O. O. Khalifa, M. R. Islam, and S. Khan, "Real time lane detection for autonomous vehicles," in *2008 International Conference on Computer and Communication Engineering*, May 2008, pp. 82–88.

[8] M. McFarland. (2015) The $75,000 problem for self-driving cars is going away. [Online]. Available: https://www.washingtonpost.com/news/innovations/wp/2015/12/04/the-75000-problem-for-self-driving-cars-is-going-away/?utm_term=.c6a3c5dc5096

[9] U. Ozguner, C. Stiller, and K. Redmill, "Systems for safety and autonomous behavior in cars: The darpa grand challenge experience," *Proceedings of the IEEE*, vol. 95, no. 2, pp. 397–412, Feb 2007.

[10] "Select," http://select.usu.edu/, accessed: 2016-12-7.

[11] Autonomous desert-crossing robotic car. [Online]. Available: http://web.mit.edu/zacka/www/grandchallenge.html

[12] R. Isermann, R. Schwarz, and S. Stolzl, "Fault-tolerant drive-by-wire systems," *IEEE Control Systems*, vol. 22, no. 5, pp. 64–81, 2002.

[13] R. Rajamani, *Vehicle Dynamics and Control*, 2nd ed., ser. Mechanical Engineering Series. Springer US, 2012.

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

[15] M. Bertozzi and A. Broggi, "Gold: a parallel real-time stereo vision system for generic obstacle and lane detection," *IEEE Transactions on Image Processing*, vol. 7, no. 1, pp. 62–81, Jan 1998.

[16] N. Apostoloff and A. Zelinsky, "Robust vision based lane tracking using multiple cues and particle filtering," in *IEEE IV2003 Intelligent Vehicles Symposium. Proceedings (Cat. No.03TH8683)*, June 2003, pp. 558–563.

[17] D. Pomerleau, "Ralph: rapidly adapting lateral position handler," in *Intelligent Vehicles '95 Symposium., Proceedings of the*, Sep 1995, pp. 506–511.

[18] D. A. Pomerleau, *Neural network perception for mobile robot guidance*. Springer Science & Business Media, 2012, vol. 239.

[19] M. Bertozzi and A. Broggi, "Vision-based vehicle guidance," *Computer*, vol. 30, no. 7, pp. 49–55, Jul 1997.

[20] H. Wang and Q. Chen, "Real-time lane detection in various conditions and night cases," in *2006 IEEE Intelligent Transportation Systems Conference*, Sept 2006, pp. 1226–1231.

[21] 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.

[22] J. D. Crisman and C. E. Thorpe, "Unscarf-a color vision system for the detection of unstructured roads," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, Apr 1991, pp. 2496–2501 vol.3.

[23] ——, "Scarf: a color vision system that tracks roads and intersections," *IEEE Transactions on Robotics and Automation*, vol. 9, no. 1, pp. 49–58, Feb 1993.

[24] M. Rosenblum and L. S. Davis, "An improved radial basis function network for visual autonomous road following," *IEEE Transactions on Neural Networks*, vol. 7, no. 5, pp. 1111–1120, Sep 1996.

[25] "Darpa urban challenge," http://archive.darpa.mil/grandchallenge/, accessed: 2016-12-7.

[26] K. Koscher, A. Czeskis, F. Roesner, S. Patel, T. Kohno, S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham *et al.*, "Experimental security analysis of a modern automobile," in *2010 IEEE Symposium on Security and Privacy*. IEEE, 2010, pp. 447–462.

[27] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, and T. Kohno, "Comprehensive experimental analyses of automotive attack surfaces." in *USENIX Security Symposium*, 2011.

[28] C. Valasek and C. Miller, "Adventures in automotive networks and control units," IOActive, Tech. Rep., 2014.

[29] C. Miller and C. Valasek, "Remote exploitation of an unaltered passenger vehicle," IOActive, Tech. Rep., 2015.

[30] A. Costley, "Platform Development and Path Following Controller Design for Full-sized Vehicle Automation," Master's thesis, Utah State University, Logan, Utah, 2017.

[31] "Road vehicles - controller area network (can) - part 1: Data link layer and physical signalling," International Organization for Standardization, Geneva, CH, standard, 2015.

[32] *Wiring Diagrams: Focus Electric 2013*, Ford Motor Company.

[33] EBSCOhost, "Auto repair reference center," https://www.ebscohost.com/public/auto-repair-reference-center, Tech. Rep., 2013.

[34] Ecu analysis with canalyzer. Vector. [Online]. Available: https://vector.com/vi_canalyzer_en.html

[35] Pcan-usb. [Online]. Available: http://www.peak-system.com/PCAN-USB.199.0.html?L=1

[36] Arm cortex-m4f based mcu tm4c1294 connected launchpad. Texas Instruments. [Online]. Available: http://www.ti.com/tool/ek-tm4c1294xl

[37] Sn65hvd230. Texas Instruments. [Online]. Available: http://www.ti.com/product/SN65HVD230

[38] Active park assist. Ford Motor Company. [Online]. Available: https://owner.ford.com/how-tos/vehicle-features/convenience-and-comfort/active-park-assist.html

[39] O. Persson and G. Persson, "Torque sensor for automotive applicaitons." [Online]. Available: http://www.iea.lth.se/publications/MS-Theses/Full%20document/5352_full_document.pdf

[40] Obd-ii background. B&B Electronics. [Online]. Available: http://www.obdii.com/background.html

[41] P. S. Murvay and B. Groza, "Source identification using signal characteristics in controller area networks," *IEEE Signal Processing Letters*, vol. 21, no. 4, pp. 395–399, April 2014.

[42] K.-T. Cho and K. G. Shin, "Fingerprinting electronic control units for vehicle intrusion detection," in *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, 2016, pp. 911–927.

[43] J. E. A. Dias, G. A. S. Pereira, and R. M. Palhares, "Longitudinal model identification and velocity control of an autonomous car," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, no. 2, pp. 776–786, April 2015.

[44] Ros tutorials. [Online]. Available: wiki.ros.org/ROS/Tutorials

[45] Md-awd-150. Mustang Dynamometer. [Online]. Available: https://mustangdyne.com/md-awd-150/

[46] Utah state university builds the nations most advanced test facility for dynamic wireless charging. [Online]. Available: http://evr.usu.edu/news/press-releases/2014/dec1-usu-builds-nations-most-advanced-facility

[47] M. A. Sotelo, "Lateral control strategy for autonomous steering of ackerman-like vehicles," *Robotics and Autonomous Systems*, vol. 45, no. 3, pp. 223–233, 2003.

[48] J. Luo and P. Tsiotras, "Control design for systems in chained form with bounded inputs," in *American Control Conference, 1998. Proceedings of the 1998*, vol. 1. IEEE, 1998, pp. 473–477.

[49] Using zed with ros. [Online]. Available: https://www.stereolabs.com/documentation/guides/using-zed-with-ros/introduction.html

[50] Newmar powering the mobile network. NewMar. [Online]. Available: http://newmarpower.com/wp-content/uploads/2016/04/Newmar_Powering_the_Mobile_Network_Mobile_Catalog_2015-Web.pdf

[51] Sn65hvd230 can board. WaveShare. [Online]. Available: http://www.waveshare.com/sn65hvd230-can-board.htm

[52] Tivaware™for c series (complete). Texas Instruments. [Online]. Available: http://www.ti.com/tool/sw-tm4c

[53] Opencv library. [Online]. Available: http://opencv.org/