Development of a Control System for an Urban Electric Vehicle Compatible with Autonomous Driving Features

by

Fernando Andre Barrios

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

Waterloo, Ontario, Canada, 2019

AUTHOR'S DECLARATION

Abstract

As cities continue to grow, the roads only become more congested, leading drivers to spend more time commuting, while vehicle accidents continue to rise. The Mechatronics Vehicle Systems Laboratory has developed its own urban electric vehicle in an attempt to combat congestion, as well as pollution concerns in urban areas. The focus of these thesis is a fault tolerant control system to improve the robustness of the vehicle, as well as allow for future autonomous vehicle functions to be tested on it.

The development of the control system began with deriving the requirements from the vehicle's hardware, the goal of the project and foreseeable changes in the future. From there, the processors were selected, electronic control units were ordered, the vehicle's necessarily electrical connections were completed, and the control software was deigned. Once every part of the project was completed, the control system was integrated, and testing was completed at the university.

The goal of the control software was to allow the vehicle to be drivable if a motor, motor controller or steering actuator failed due to an electrical, mechanical or feedback fault. Due to the vehicle's modular design it allowed other components to compensate for failures, while maintaining vehicle controllability and predictable vehicle dynamics. Furthermore, the control system allows for autonomous vehicle functions to be added and tested on the vehicle. This was tested through the addition of a lane following models, along with the control system's path following function.

The developed control system was shown to improve vehicle controllability when faults occurred, as well as allowed lane following to work effectively with the system structure and communication channels chosen. Overall, validating the design as well as allowing future mechanical designs or autonomous functions to be tested on the vehicle.

Acknowledgements

My thesis project, and my time at the University of Waterloo would not have been possible without the support structure around me. Firstly, I want to thank Professor Amir Khajepour for giving me the opportunity to work with him, as well as providing me with invaluable insight, and guidance along the way. Thanks to his management style, patience and expertise, he gives every member of his large research group the foundation to grow and flourish.

I would also like to thank all the members of the Mechatronic Vehicle Systems Lab that have shared their knowledge with me and helped to make the vehicle testing possible. Starting with the lab technicians, Jeff Graansma and Jeremy Reddekopp, who were always there to help in the lab when assistance is needed and provide unique suggestions along the way. The lab's co-operative undergraduate students, Brandon Johnson and Bao Anh Nguyen, for their help and expertise when assembling various portions of the vehicle. The lab's research engineer, Rhyse Maryniuk, for his continued support and always being available to answer questions with regards to the vehicle's design. Yubiao (Gary) Zhang, for his guidance, support and assistance during my research. Minghao Ning, who created the lane detection algorithms to make the control system testing possible. As well as various other colleagues that I have had the pleasure of working with including; Neel Bhatt, Bruce Huang, Davis Dao-Vu, Nathanael Jordan, and Mirco Woidelko.

I want to thank my family and close friends who have always been there to support me in my journey. Finally, a big thank you to my parents, Carlos and Damaris Barrios, for their patience and unconditional support.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

BEV	Battery Electric Vehicle
BMC	Body Module Controller
CAN	Controller Area Network
CMC	Corner Module Controller
CNN	Convolutional Neural Network
DC	Direct Current
ECU	Electronic Control Unit
FCNN	Fully Convolutional Neural Network
FPS	Frames Per Second
FTC	Fault Tolerant Control
HV	High Voltage
ICEV	Internal Combustion Engine Vehicle
ICNet	Image Cascade Network
IoT	Internet of Things
IoU	Intersection over Union
MBC	Model Based Control
MVSL	Mechatronic Vehicle Systems Laboratory
NRE	Non-recurring Engineering
OTA	Over The Air
PCB	Printed Circuit Board
PWM	Pulse Width Modulation
RCNN	Recurrent Convolutional Neural Network
SBC	Single Board Computer
SCNN	Spatial Convolutional Neural Network
UEV	Urban Electric Vehicle
VSC	Vehicle Supervisory Controller

Chapter 1
Introduction

With rising fuel prices, increased congestion and an increasing amount of fatal accidents, the automotive industry is undergoing radical changes in terms of propulsion, as well as autonomous driving features. The technological advancements of the combustion engine and improvement of transportation networks have resulted in great benefits to humanity, but has had detrimental effects on the environment. With heightened environmental concerns due to climate change, the push for alternative fuels and efficient transportation has never been greater. On the forefront of this transition are electric vehicles, including urban electric vehicles.

1.1 Urban Electric Vehicles

Urban electric vehicles are smaller electric vehicles designed for city driving with lower range, and a lower price point than long range electric vehicles, such as the Chevrolet Bolt. Urban electric vehicles have the potential to decrease congestion in major cities, while also reducing greenhouse gas emissions and noise pollution. Their smaller footprint not only allows more vehicles to travel on the road at once, but also greatly decreases the amount of parking area that is required.

Electric vehicles benefit from reduced weight from the transmission and motor, because they do not require a gearbox or only have a fixed gear transmission. The motor also has a higher power to weight ratio than an engine. Battery electric vehicles (BEVs) still tend to be heavier than conventional vehicles due to their large battery pack, and the fact that as energy is consumed, the weight of the vehicle stays constant instead of decreasing. However, when compared to long range electric vehicles urban vehicles require a much smaller battery pack, but still benefit from reduced weight of the transmission and motor. Their lower weight also helps to improve their efficiency and reduce the amount of energy required to travel a specified distance. BEVs already benefit from almost a two-fold efficiency improvement with a well-to-wheel efficiency of around 27%, compared to an internal combustion engine vehicle (ICEV) which only has about 15% well-to-wheel efficiency [1].

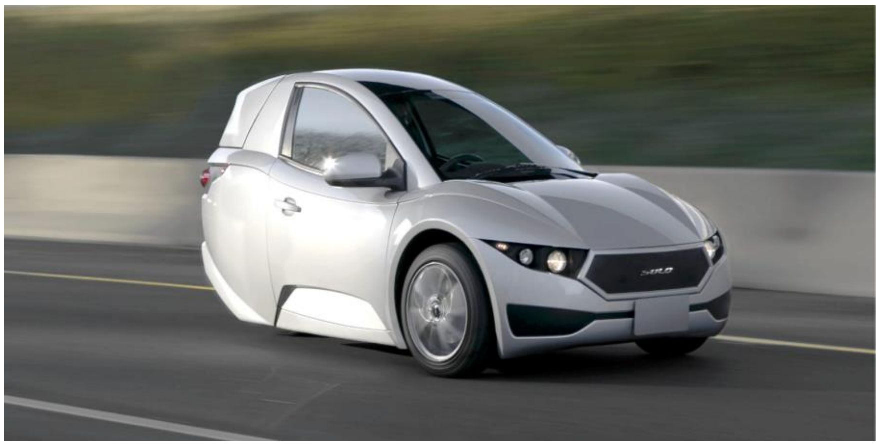


Figure 1.1 Urban Electric Vehicle sold and manufactured by Electra Meccanica

Furthermore, urban electric vehicles address many of the concerns that consumers, businesses and governments commonly have about electric vehicles. Well-to-wheel emissions of an urban electric vehicle (UEV) is about 2.5 times less than a gasoline vehicle [1]. This is based on the European Union electricity mix that only includes 27% renewable energy [2]. Therefore, this will be further improved as renewable energy sources become more efficient and widely used. Their smaller energy usage when compared to heavier long-range BEVs also means that UEV will put less stress on the electrical grid as more people adopt the technology. Finally, range anxiety will be reduced when these vehicles are used for short, urban driving distances where regenerative braking will be heavily utilized.

1.2 Research Urban Vehicles

Another purpose of urban vehicles is for conducting vehicle research. The main objective for research vehicles is to be able to rapidly validate various prototype designs [6]. This is achieved through the use of modular designs that allow various components to be exchanged without having to make major adjustments to the entire system.

The MVSL Urban Vehicle has served as a great modular research vehicle for which many electrical, mechanical, as well as software designs have been tested and validated. The vehicle has a novel modular drive and suspension system, commonly called the corner module [6]. This corner module design has facilitated the testing of various suspension designs, including various stiffness levels and an air suspension. It has also facilitated the testing of hydraulic, as well as electric steering components for the purposes of comparison. On top of that, it has served as a platform to validate four iterations of PCB designs and two different processors that have decreased the vehicle's cost while improving performance. Finally, two iterations of control software have been tested on the vehicle, the vehicle now contains a robust control system that has been designed with the goal of autonomous driving in mind.

1.3 ADAS and Autonomous Driving

The concept of vehicle safety, and occupant protection began almost a century ago when General Motors performed the first crash test in 1934 [1]. It is not until recently that vehicle manufactures have begun to implement advanced driver assistance systems that focus on collision prevention instead of occupant protection. As population density increases around cities, major highways and roads only become more congested [2]. In turn, this is constantly increasing the amount of time that we spend behind the wheel and it is now estimated that the average person spends about 200 hours every year commuting to their jobs alone [3]. The excessive amount of time behind the wheel inevitably introduces increased fatigue for the driver, putting not only their own life at risk, but also the lives of other road users such as pedestrians and cyclists.

It is clearly evident that the increasing quantity of vehicles on the road, along with increasing congestion, and longer commutes are negatively affecting the way we drive. Since 2013, every year except for one has seen an increase in the amount of vehicle collisions [4]. To alleviate some of the burden of driving and reduce the amount of traffic related incidents, an increasing amount of advanced driving assistance features are being introduced, eventually leading to a future of autonomous driving. Today, an average of 102 people are killed daily due to vehicle collisions in the United States alone [5]. The United States Federal Administration also reports that lane departure accidents account for about 51% of all fatal crashes [6]. Therefore, not only is lane detection vital for autonomous driving, but if every vehicle was equipped with lane departure warning or a lane keeping system, more than half of the accidents today could potentially be avoided.

Chapter 2
Literature Review

Fault tolerant control systems and autonomous vehicle systems are currently popular research topics. Therefore, before beginning the design stage of the system, a thorough literature review was conducted to evaluate different techniques that have been applied and how they may benefit the lab's urban electric vehicle. The topics that were of particular interest include; MVSL Urban Electric Vehicle Architecture, Fault Tolerant Control Systems, Model Based Design, Lane Line Detection Methods, and Path Tracking Methods.

2.1 MVSL Urban Electric Vehicle Architecture

This section will provide an overview of the UEV which was built in lab, both from literature and observations. The Mechatronic Vehicle System's Laboratory Urban Vehicle integrates a powertrain output of 30kW continuous power, 90kW peak power, along with a 3.24 kWh, 72V battery pack. The urban vehicle also features; front disc brakes, is capable of regenerative braking, has a hydraulic system supplying 2 GPM continuous for the front steering, electric rear steering, designed for drive-by-wire, and processor communication over CAN [6]. The UEV is a three-wheeled vehicle in a tadpole formation designed for a reduced footprint and increased efficiency when compared to the vehicles on the road today.

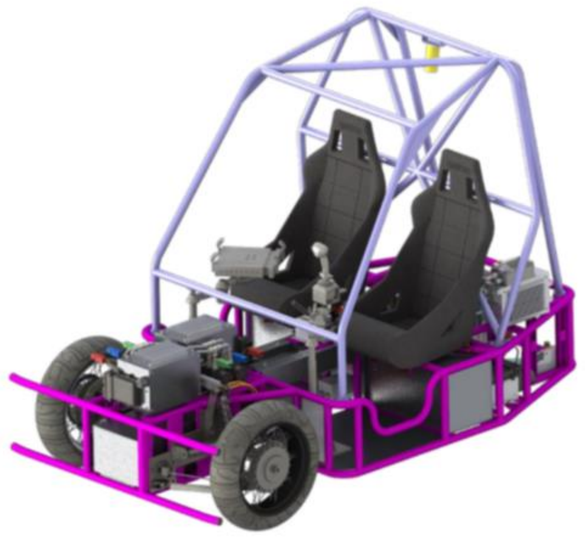


Figure 2.1 Mechatronics Vehicle System Laboratory Urban Vehicle [6]

The UEV was built upon the concept of corner modules, which gives the vehicle unique advantages. Corner modules are independent units that contain the complex components of the vehicle, allowing for different chassis designs to be implemented with this concept while decreasing the non-recurrent engineering (NRE) cost. The vehicle has three corner modules in total, two at the front and one in the rear. The front corner modules contain a wheel with in-wheel motors, a motor controller, a pre-charge circuit along with the high voltage contactor, a suspension system and two stepper motors to control the hydraulic fluid to the two hydraulic cylinders for camber and steering. The rear corner module contains the same components, except it does not contain stepper motors to control the hydraulics, instead it contains one linear actuator to steer the wheel. It is important to note that only the front wheels are capable of cambering.

The vehicle is powered by three in-wheel EnerTrac MHM603 electric motors, which are driven by the manufacture recommended Kelly KLS12301-8080I motor controllers [6]. As mentioned earlier, the front steering and camber are controlled by two hydraulic cylinders per side, being powered by a hydraulic pack. The hydraulic pack contains two electric pumps, a 32 oz. charge piston style accumulator, and an automatic release valve [6]. The hydraulic pack also has a dedicated 12V automotive lead-acid battery along with a DC-DC converter that provides 13.5V at 30 Amps from the high voltage pack. The rear corner module steering is controlled through a Thomson Electrak HD that provides feedback and is controlled through CAN using the J1939 protocol. The linear actuator is powered by a 12V battery, using a step up transformer to provide 24V. The same battery that powers the linear actuator powers all the ECUs and motor controllers, with its own DC-DC converter also supplying 13.5V at 30 Amps. Finally, the vehicle's high voltage lead acid pack provides 72V nominal, with a capacity of 3.24kWh.

In terms of operating the vehicle, as well as human machine interface there are four main points of focus including a bank of switches, a directional switch, a mechanical brake lever and a joystick. The bank of switches contains a total of four switches which power the vehicle's low and high voltage, power the hydraulics, begin the motor pre-charge and enable the motors. The directional switch was designed to allow the driver to select either forward or reverse once the vehicle is in a drive state. Beside the directional switch there is also an emergency stop button that will switch off the high voltage system, no longer allowing the motors to provide torque. Next to the emergency stop button the driver will find the mechanical brake lever. In the current configuration, the brake lever must be pulled to provide hydraulic brake pressure to the front corner modules and stop the vehicle. There is a 4-axis CAN Bus joystick purchased from Pran Systems that contains, four buttons, two rockers, x-axis and y-axis inputs, in a configuration as shown below in figure 2.2.

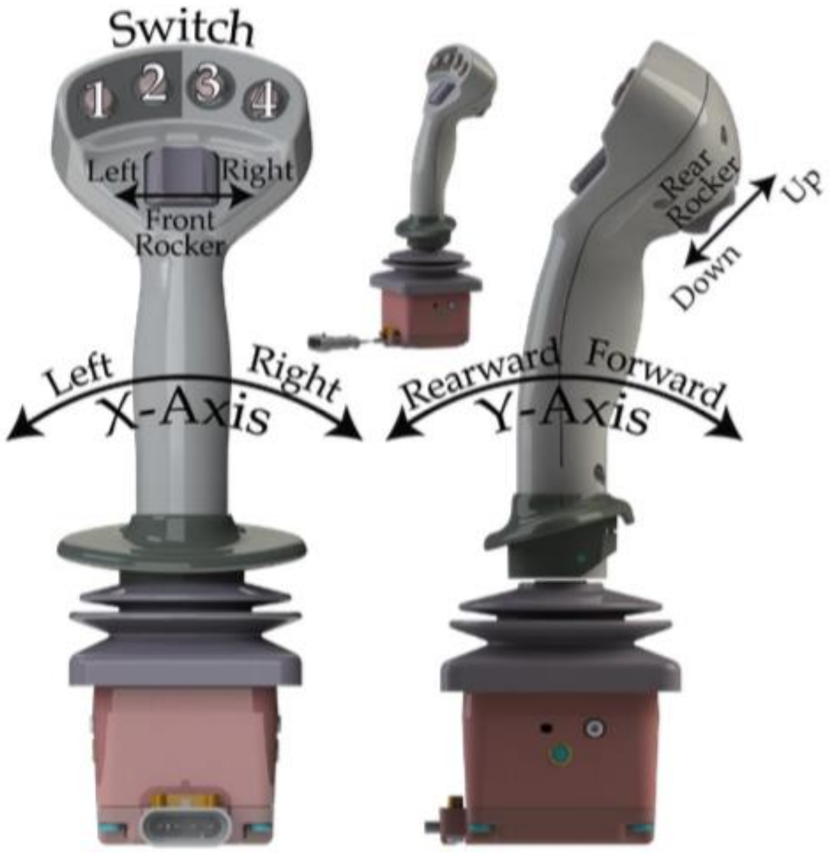


Figure 2.2 Urban Vehicle Joystick by Pran Systems [6]

2.2 Fault Tolerant Control Systems

The main objective for fault tolerant control systems is to ensure that the vehicle remains in a safe state when faults occur, decreasing the probability of injury to the passengers. The easiest way to achieve this is by using redundant hardware, therefore when a fault occurs the responsibility can be transferred to the secondary component. However, this is an inadequate solution for a high volume industry such as the automotive industry where a small addition over millions of units quickly adds up to a big cost.

The solution to this has been achieved through the use of "intelligent" software that prevents faults from developing into critical failures [3]. The software will attempt to keep the plant operating at full or reduced capacity when failures occur. In an automotive control system, the safety of the vehicle's occupants is paramount, therefore the focus of the system becomes maintaining control of the vehicle, even when certain actuators or systems may be unavailable. To achieve this controller transition smoothness after a failure, as well as sensitivity to control parameter variations must be taken into account [8]. In extreme cases, the plant may have to default to a safe state to ensure the safety of the occupants. The first task of a fault tolerant control system is fault detection, the control system must be able to identify the fault and quantify the severity of the fault. Then, once a fault has been detected the focus is then shifted to fault mitigation. The full control strategy includes; fault diagnosis, remedial action selection, and finally implementation. A common approach is to have a supervisory-level control that can adjust the vehicle's control based on the faults and their severity [4] [8].

The task of the supervisory control is to monitor subsystems and send requests to them, while monitoring that control objectives are met. If the objectives are not met, the supervisory control shall calculate a revised control objective, with a new control structure and/or parameters to meet the revised objective [3]. The level of complexity in the supervisory control can itself be an increased risk for software failure, therefore completeness and correctness of the implementation is vital. To achieve this, it is important implement a supervisory control that is modular, where each functionality can be designed, implemented and tested separately [3][7].

The architecture of the supervisory control shall allow for the implementation of the following features [3][7]:

- Support of overall coordinated plant control different phases of the controlled process; startup, normal operation, event triggered operation with different control objectives, shut-down.
- Support of all control modes for normal operation and modes of operation with foreseeable faults.
- Autonomous monitoring of operational status, control errors, process status and conditions.
- Fault diagnosis, accommodation and reconfiguration as needed. This is done autonomously, with status information to plant-wide coordinated control.

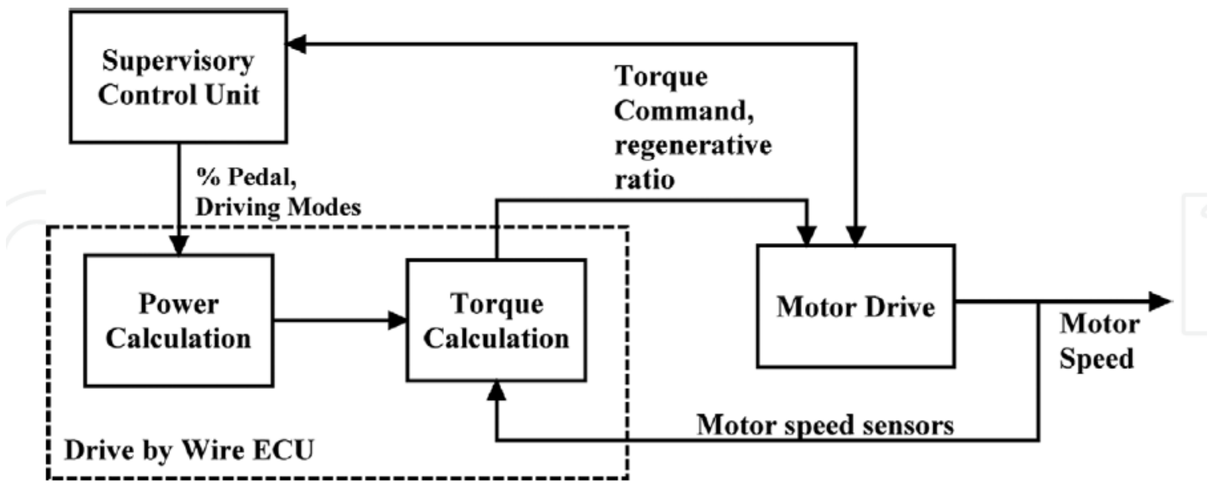


Figure 2.3 Sample Supervisory Control Architecture [9]

Available control schemes are generally classified into either active or passive fault tolerant control. Active FTC aims to reconfigure the controller during run-time, after the fault has been detected and quantified. On the other hand, the passive FTC is a robust controller that is expected to deal with all the expected faults. Passive FTC has the disadvantage of only being able to deal with the faults that it is expecting, however it has a much faster response than active FTC [5]. Faults are discrete events, each one being separate and distinct, however each fault that occurs changes the properties of the system. Due to the discrete nature of faults and control system reconfiguration, a vehicle's FTC system is naturally a hybrid between active and passive control [7].

2.3 Model-Based Design

Stringent performance requirements and shorter development cycles are driving the use of modeling and simulation [18]. Model-Based Design is a development process that enables faster, more cost-effective development of dynamic systems. Not only does it have the ability to reduce time to market, but it enables these products to have higher safety and reliability. In MBD, a model is at the center of the development process, from requirements development, through design, implementation and testing [11]. Simulation is used at each process step to verify whether the design meets the requirements [12].

Model-Based Design utilizes math based visual methods for designing complex control systems, examples of visual languages include; Simulink, Labview, Statemate or similar [11][14]. The visual languages, along with an architecture that fulfills the requirements facilitates the readability, understandability and modularity of the system. Code generation is utilized to implement the complex logic, thus saving time and cost by reducing error introduced by writing source code by hand [9][12]. Furthermore, MBD allows for early validation and testing, allowing flaws to be identified at an early stage [10][13]. Model-Based Design allows you to improve efficiency by various other factors including [11]:

- Using a common design environment across different teams
- Linking designs directly to requirements
- Integrating testing with design to continuously identify and correct errors
- Refining algorithms through multi-domain simulations
- Automatically generating documentation
- Reusing designs to employ systems across multiple processors and hardware targets

Model-Based Design can bring significant cost savings, but only with a well-chosen model-based development process [17]. The V-Model represents a software development process which may be considered an extension of the waterfall model. However, the V-Model demonstrates the relationship between each phase of the development life cycle and its associated phase of testing [15]. The V-Model is ideal for applications where the requirements may change, proper validation must take place at each phase, the tester is involved from the early stages of development, and the project is large [16]. The V-Model is made up of two branches, as shown below in figure 2.4. The left branch starts with system-level requirements of design, which are partitioned into subsystems and components. The right branch represents the realization and testing of the subsystems, components and final integration [18].

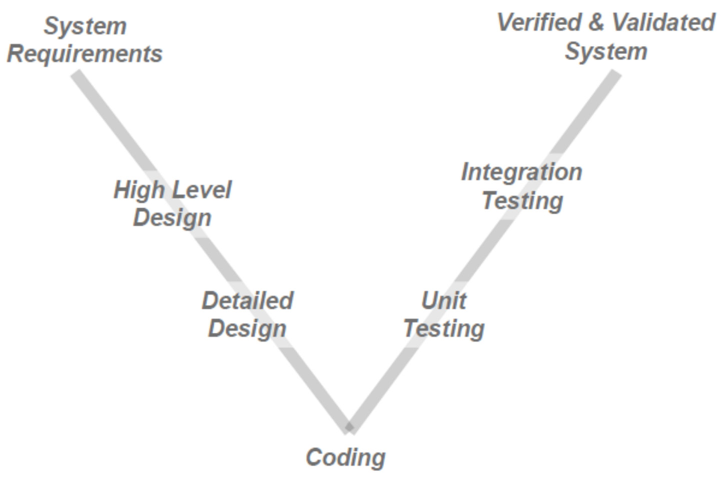


Figure 2.4 The V-Model [18]

2.4 Lane Line Detection Methods

There are currently two state-of-the-art methods that are utilized for lane line detection: feature-based algorithms and machine learning based models [31]. This section will provide an overview of both methods, as well as an overview of machine learning models that are suitable for lane detection.

Feature-based algorithms use low-level image features and filters to reduce the noise to successfully identify the lanes [32]. The features that are mostly commonly used are color and edge detection, assuming that there is a large enough contrast between the road and the road markings. The advantage of feature-based algorithms is that they are less computationally expensive, however they are more susceptible to errors when the environment is not optimal creating difficulties for real-world applications [33]. In many cases when driving, the environment is not optimal to use feature-based lane detection. Some of these situations may be encountered when there are shadows present, the vehicle's camera's view may not be clear due to sun's glare or other weather conditions. Other factors that can affect this include when lane markings may not be clear, or the contrast between the lane lines and the surface is not adequate, which may be the case on concrete road surfaces. On top of that, feature-based algorithms cannot be expanded to reliably detect other vehicles or pedestrians.

In contrast, machine learning based models can be expanded to detect all objects of interest, while also increasing the accuracy of the lane detections in real-world scenarios. Once the lane has been identified, the lane position as well as its orientation can be identified by the low level features that the neural network has extracted. Studies have shown that neural networks can increase the detection accuracy of lanes by up to 10% compared with traditional image processing methods [33]. Furthermore, neural networks can even help to connect the lanes that are partially covered by an object, and achieve up to 99% detection accuracy [34]. Neural networks have been the focus of autonomous driving for many years now. With various technology companies spending billions of dollars into the research of deep learning, new unprecedented improvements and new neural network architectures are being released every year [35]. The biggest challenge is that generally, there is a tradeoff between detection speed and accuracy, as well as robustness of the detections. An example of this is a RCNN, which produces some of the best results in terms of accuracy, however due to the amount of classification queries and inefficient use of convolutions, it remains impractical for real-time applications [43].

There are a vast amount of machine learning models for object detection, however they are not all suited for lane detection due to their network architecture. There are also three methods that items can be localized including: object detection, semantic segmentation and instance segmentation, as seen in figure 2.5 shown below. Object detection will provide a bounding box around the objects of interest, whereas segmentation will label each pixel of the image with its corresponding class. The difference between the two types of segmentation is that semantic segmentation will just label each pixel depending on which class it belongs to, whereas instance segmentation will label each instance of each class uniquely [41].

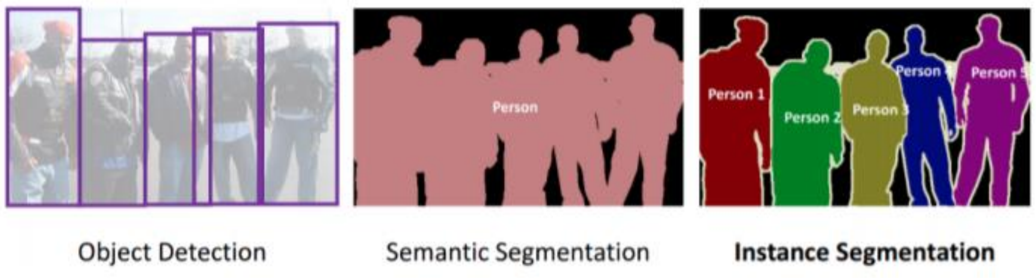


Figure 2.5 Differentiation between Object Localization Methods

The advantage of semantic segmentation networks is that they clearly label every object in the image. For example the Mapillary training set for semantic segmentation networks contains 66 visual object categories that it can differentiate between, these include; road markings, animals, infrastructure categories, signs, different vehicle types, sidewalks, bike lanes, parking and road sections [45]. However, the detection rate is generally slower than typical object detection. Figure 2.6 shown below provides an overview of the performance of semantic segmentation neural networks, outlining that most segmentation networks cannot provide predictions at more than 1fps. As seen in the graph ICNet is the only segmentation network that maintains a high accuracy while being able to compute in real-time due to its down sampling, shrinking feature maps and conducting model compression. The network takes three image inputs, one of low resolution, medium resolution and high resolution, then uses a cascade feature fusion unit to identify all the features of interest. First, the network gets semantic extraction using the low resolution input, then it refines the prediction using the medium and high resolution inputs. This makes the architecture must faster than others that segment the high resolution input, such as Fully Convolutional Networks (FCNNs) [44].

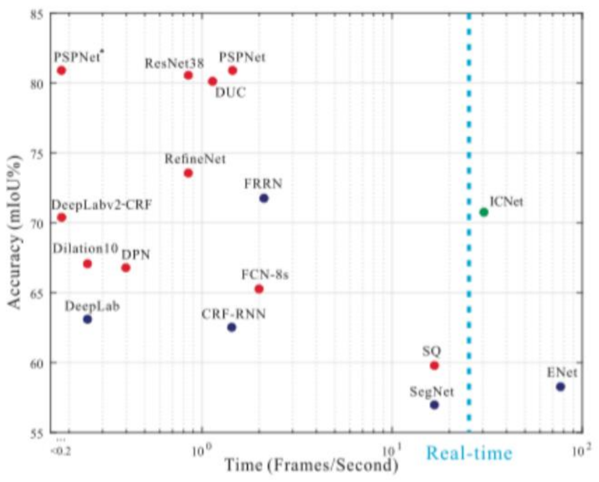


Figure 2.6 Performance Characteristics of Popular Segmentation Networks [42]

Convolutional networks have been able to perform object detection with speeds greater than 40fps [41]. To apply CNNs to lane detection it is important to capture spatial relationships of pixels across rows and columns of the image, which traditional CNNs are not designed for. However, Spatial CNN (SCNN) is particularly suitable for traffic lanes because it can detect long continuous shapes or large objects, with a strong spatial relationship, but less appearance clues. In a CNN, a convolution layer receives input from previous layers, applies a convolution operation and nonlinear activation, and sends the result to the next layer. In contrast, the SCNN views rows or columns and applies a convolution operation, nonlinear activation and sum operations before sending the result to the next layer. The SCNN has achieved impressive results, including winning first place on the TuSimple Benchmark Lane Detection Challenge, with an accuracy of 96.53%.

2.5 Path Tracking Methods

Lane detection is vital to reduce the amount of accidents on the road, however to increase the benefit of the lane detection model, a path tracking model or steering control algorithm can be added to keep the vehicle within the lanes. Together, the lane detection and steering control can form the basis of an autonomous vehicle. The main goal of the steering control is to minimize the error between the vehicle's location and the intended path, after the path has been defined, in this case by the lane detection module.

Overall, it has been stated that none of the path tracking methods perform well in all of the applications, however they contain complementary characteristics. This analysis lead to the conclusion that for more general applications it would be better to take a combination of the various path tracking methods for optimal performance [36]. PID steering control has been omitted from this section because without a heuristic function the controller is unstable even at moderate speeds. It can also clip corners and has demonstrated poor performance in tight curves [38]. This section will focus on path tracking using geometric models, as they are suited for moderate driving conditions similar to those that the lab's urban vehicle will be expected to maneuver. Slippery conditions, aggressive or emergency maneuvers may require a more accurate vehicle model with predictive control [37].

One of the most popular classes of path tracking methods is geometric path tracking, more commonly the Pure Pursuit and the Stanley methods. These path tracking methods use a bicycle model making two key simplifications. The first simplification is that the front and rear wheels are combined to make a two-wheeled model. The second simplification being that the vehicle can only move on a plane. Together these two simplifications result in a simple geometric relationship between the front wheel steering angle and the curvature that the rear wheel will follow [36].

In a similar way to humans, Pure Pursuit picks a look ahead distance in front of the car and heads towards this goal point (g_x,g_y) on the given path, as seen in figure 2.7 shown below. The look ahead distance then changes as we drive to reflect the curvature of the road [39]. The method consists of calculating the curvature of the circular arc that connects the vehicle's rear axle location to a goal point on the path ahead. The vehicle's steering angle (δ) can then be determined just by using the location of the goal point, as well as the angle between the vehicle's heading vector and the look ahead vector (α) [36].

On the other hand, the Stanley method calculates the cross track error (e_{fa}) by measuring the distance from the center of the front axle to the nearest path point (C_x,C_y), as seen in figure 2.7. There are two terms that when added together determine the steering wheel angle of the vehicle. The first term simply keeps the wheels aligned with the given path by setting the steering angle (δ) equal to the heading error. The second term adjusts the steering angle such that the intended trajectory intersects the path tangent from (C_x,C_y) at kv(t) units from the front axle [36].

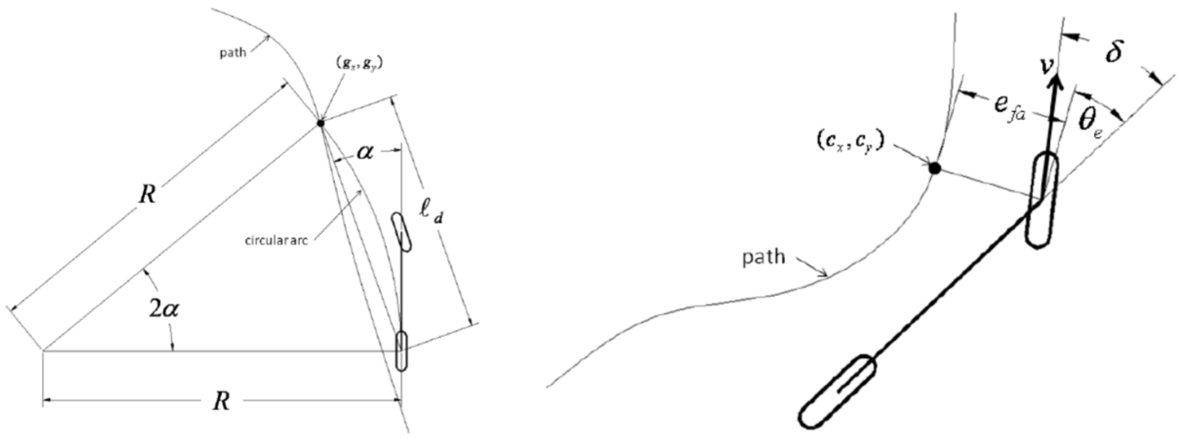


Figure 2.7 Pure Pursuit (left) and Stanley Geometry (right) [36]

Based on empirical comparison of both tracking methods, pure pursuit is recommended for slow driving and/or on discontinuous paths, whereas the Stanley method was recommended for smooth highway driving and/or parking maneuvers. Pure Pursuit works well, it is robust to large errors or discontinuous paths, but it is unclear how to pick the look ahead distance. Commonly, the approach taken is to vary the look ahead distance based on vehicle velocity, but it could also be a function based on path curvature. The Stanley method is the simplest method of the two, however it outperforms the Pure Pursuit controller in most scenarios. The drawback is that it is not as robust to large errors and non-smooth paths [36].

Chapter 3
System Requirement Specification

The design of the urban vehicle control system was completed based on the hardware that was chosen for the vehicle, but also keeping in mind the long-term vision of the project. The system requirements specification describes the features and behavior of the control system to achieve the intended vehicle performance, as well as support future additions to the system. The first step is to understand the vehicle hardware, as well as its interface to the control system. From there the rest of the requirements will be derived from the desired vehicle performance. This phase of the project is crucial to ensure a clear understanding of the goals to be achieved and for continued success throughout the project.

3.1 Vehicle Subsystem Analysis

The MVSL Urban Vehicle is composed of various subsystems that together have the capability to achieve the vehicle's original performance objects. In this section, each subsystem will be explored in detail, including their communication requirements. This in turn will help to formulate the hardware and software requirements for the vehicle's control and autonomous driving systems.

3.1.1 Brake System

This section will focus on the vehicle's mechanical brake system, and regenerative braking will be discussed in the next section. The vehicle cannot rely solely on regenerative braking because the battery pack can only receive a certain amount of current primarily based on its state of charge and pack temperature. Regenerative braking is also much less effective at low speeds than it is at high speeds. These factors result in varying performance results from regenerative braking, making regenerative breaking an unreliable option as the only source of braking torque.

The MVSL urban vehicle is fitted with a mechanical braking system that supplies brake pressure to two disc brakes located on the two front corner modules, applying negative torque to the front wheels. In its current state, a hand brake is utilized to provide pressure to the cylinders at the corner modules as seen in figure 3.1 below. Both brake lines contain a pressure sensor, which will provide the system with vital feedback as to when the driver is attempting to slow down. This will be especially useful when the vehicle is equipped with autonomous driving features or if the driver is commanding a torque, with the mechanical brakes applied. The hydraulic brake pressure sensors are TDH30 Pressure Transducers from Transducers Direct that output a 0-5V analog signal and have a pressure reading range of 0-3000 PSI. The vehicle is also equipped with an ABS module which will potentially improve the braking performance in the future.

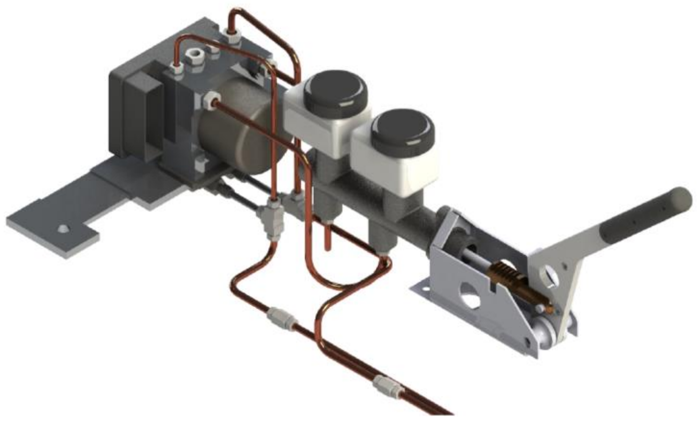


Figure 3.1 Urban Vehicle Mechanical Brake System [6]

3.1.2 Torque Producing System

The MVSL urban vehicle contains a total of three in-wheel EnerTrac MHM603 motors, the two at the front are air cooled, while the one in the rear is not cooled. The motors are each rated for 10KW of continuous power at 25 degrees Celsius, and up to 30KW peak power [20]. Each motor is controlled by the manufacture recommended KLS12301-8080I Kelly Motor Controller, in either torque or speed control modes.

The Kelly Controller is a vital part of the control system providing information regarding the motor, as well as controlling the electric current to the motor. There are five digital inputs, one is to power the controller, another one is to enable throttle and the last three are to set a drive mode including; forward, reverse and regenerative braking. The Kelly controller also has two analog inputs (0-5V), one is used as the command for regenerative braking torque, while the other one is a torque or speed request, depending on the controller's internal parameters [21].

The feedback from the Kelly Controllers is received through the vehicle's CAN bus, allowing feedback to be received very quickly and reliably. The following information is broadcasted by the Kelly Controller; low/high voltage faults, throttle signal feedback (re-iterating the 0-5V throttle input), sensor errors, controller and motor temperature, along with over temperature faults [22]. This allows the ECU to ensure that the throttle signal that is being sent out agrees with the throttle signal that the controller is reading. It also provides important temperature and sensor faults, which may require a motor to shut down.

Furthermore, each Kelly Controller has a dedicated precharge/discharge circuit that allows the ECU to control when HV power will be delivered to the controller and to disallow the inrush of current to the controller. This circuit expects two digital inputs, one to start the precharge at a rate of 120mA, and another to close high voltage contactor once the desired voltage has been reached. Once the high voltage contactor is closed it effectively connects the controller directly to the battery. If neither of these two inputs are given, the circuit will discharge at the same rate of 120mA [6].

3.1.3 Steering System

As mentioned earlier the rear steering is available through an electric linear actuator, whereas the front steering is hydraulic. The linear actuator is fully controlled and provides feedback through the CAN bus, making it easy interface with. The actuator provides feedback regarding its current position, current usage, and faults that may have occurred. The main faults that are continually reported include an overload fault, fatal error fault, voltage fault and temperature fault.

The front steering is available through two hydraulic cylinders in each corner module that control camber and steer angle. The flow of hydraulic fluid is controlled by two Nema 34HS59-5004S stepper motors on each corner module, to allow the cylinders to retract or extend. The stepper motors each have their own two-phase Leadshine DM556 drivers that expect two inputs; a digital input that will signal the direction of rotation and a PWM signal that determines the speed of rotation. The drivers are powered by a 20-50V voltage supply and draw 0.5-5.6A of current.

Each hydraulic cylinder has its own feedback device to estimate the current extension, provided by a TE Connectivity SP1-12 string potentiometer. The String potentiometer can extend up to 12.5 inches and can be powered by up to 30V, it will then return an analog output depending on the extension [23]. For example, if the string potentiometer is powered by 5V and is extended 6.25 inches, then the analog output of the potentiometer is expected to be 2.5V.

3.1.4 Energy Storage System

The urban vehicle's energy storage system is made up of three main batteries. It features two low voltage batteries outputting a nominal 12V each and one high voltage battery outputting a nominal 72V. The first low voltage battery is used solely to power the pumps in the hydraulic pack, due to the substantial amount of current required to power the pumps. The second low voltage battery was designed to power the controllers, as well as the rear steer actuator with the help of a 24V DC-DC converter. The low voltage batteries each have a DC-DC converter to allow them to charge from high voltage pack, both DC-DC converters have a relay to enable them. Finally the high voltage battery was designed to power the four front stepper motors for the hydraulic steering, and the three wheel hub motors to drive the vehicle.

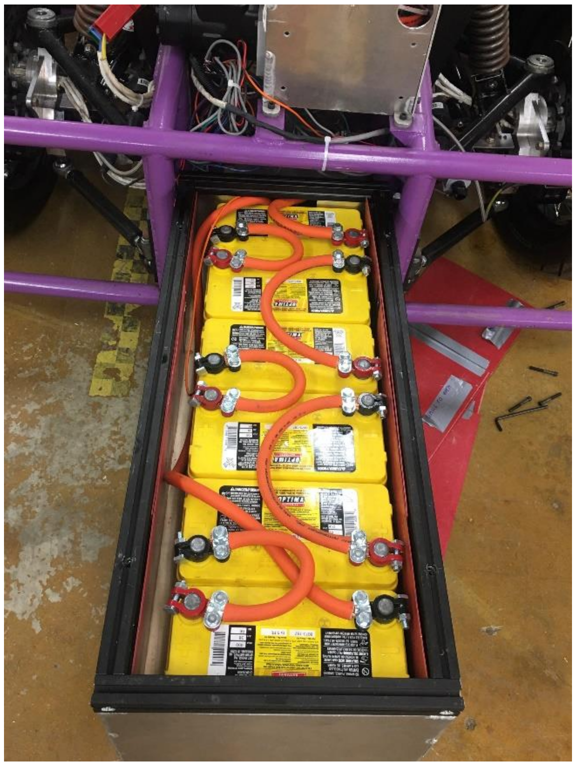


Figure 3.2 The Urban Vehicle's High Voltage Battery Pack

The high voltage battery pack is made up of six standard vehicle lead acid batteries as shown in figure 3.2 above. The total capacity of the pack is 3.24 kWh, which would be enough to complete small drive cycles on campus. The lead acid batteries were chosen due to their dependability and their relatively low cost. The battery does not contain a battery management system (BMS), in a similar way that traditional vehicles do not contain a BMS. The battery pack did have to be charged slowly to ensure that all cells would charge to full capacity, as each individual battery's state of charge would not always be the same.

3.1.5 Summary

The MVSL urban vehicle has a variety of different actuators, as well as sensors, requiring a mix of analog, digital and CAN signs to work effectively. This analysis of the vehicle's hardware provides restraints that became vital to formulate the ECU requirements, and control system requirements.

3.2 Control System Requirements

System requirements are critical for the design phase, and provide a standard to which the system can be verified, as well as validated. For the Urban Vehicle it was important to clearly set the safety requirements due to the unconventional technologies that are being used because it will be used as a future prototype platform. Furthermore, the design of the system architecture was dependent on the faults that the system is expected to deal with.

This section will cover the ECU requirements, followed by the requirements for the processor within the ECU, and the requirements for the vehicle's control software to complete the control system. The requirements are broken down into functional and non-functional requirements. Functional requirements focuses on the systems inputs, outputs and their behavior interrelationships. Whereas the non-functional requirements are defined as the general qualities of the intended product including; timing, reliability, safety, security, usability and maintainability [24].

3.2.1 Electronic Control Unit Requirements

The vehicle's electrical control units are responsible for executing the controls software. It is essential that the ECUs support both the compiled software, as well as provide the appropriate interface to the vehicle's actuators and sensor. The lab would be making a proprietary ECU for various projects and these requirements were formulated to ensure that the board ECUs would be compatible for the Urban Vehicle. The ECU requirements have been derived from the vehicle's software requirements, communication protocols, current hardware and foreseeable hardware changes. The non-functional requirements for the ECU are as follows;

- Automotive enclosure of the size 20cm x 20cm x 6cm or smaller
- The final cost for each unit shall be under $300 CAD to produce, excluding the NRE cost
- All components required for vehicle operation shall be automotive grade, with an operating temperature range of -40°C to 125°C
- All integrated circuits embedded in the ECU shall have AEC Q100 rating, and all passive components embedded in the ECU shall have a AEC Q200 rating
- The ECU enclosure shall have at least an IP67 waterproof rating

The functional requirements for the ECU include;

- The ECU shall be powered by a voltage range of 8 – 18V and resilient to electrical malfunctions such as reverse polarity, short circuits, under voltage and voltage spikes
- The ECU shall have at least one 5V output pin and at least two ground pins
- The ECU shall have at least five analog inputs configurable over a voltage range of 0V to 20V and with voltage spike protection
- The ECU shall allow for at least four digital signals as inputs
- The ECU shall have at least two CAN bus interfaces, with bit rates up to 1 Mbps, compliant with physical layer standard ISO 11898-2, supporting both standard (11-bit) and extended identifiers (29-bit)
- The ECU shall include the option to include a CAN bus terminating resistor (120 ohm)
- The ECU's outputs shall provide a minimum of 0.5A of current and can connect the output to either 0V, 5V or battery voltage, as well as capable of operating PWM with a frequency of up to 20 kHz
- The ECU shall have at least fourteen independent digital outputs, with at least four of them being capable of producing PWM signals

3.2.2 Electronic Control Unit Processor Board Requirements

The ECU Board will contain the processor, along with various I/O functionalities that will be mounted to a PCB board to provide the interface between the vehicle's electrical system and the board. In the future, these designs can be merged into one board, however being able to test them separately can provide many advantages. The vehicle's processors will execute the control software of the vehicle and compute the values necessary to read the sensors, as well as send commands to the actuators. It is critical that the processor can execute the control program at the refresh rate specified, that it can read all the inputs, it can produce the appropriate outputs and that the models can be compiled to the processor. The non-functional requirements for the processor are as follows;

- The processor board shall fit within the ECU size requirements
- The processor board shall cost less than $100 CAD each
- The processor board components shall be automotive grade with at least an ISO 26262 ASIL level B rating
- The processor board shall be able to operate within the temperature range of -40°C to 125°C
- The processor board manufacture shall provide adequate support to program and utilize the processor's functions that are required
- The processor board manufacture shall guarantee a minimum of 10 years of supply
- The processor board shall have the ability to be powered by a minimum of 5V
- The processor shall be capable of real-time processing of the critical logic (refresh rate of 10ms)

The functional requirements for the processor board are as follows;

- The processor shall allow for at least two CAN modules
- The processor should allow for code generation programming using a Model-Based Design language

- The processor board shall be able to handle the inputs and outputs as set out by the ECU requirements in the previous section

3.2.3 Control Software Requirements

The control software of the vehicle would become vital to handle the faults that may occur and to allow the lane following to work as intended. The following non-functional requirements were identified for the urban vehicle's control software;

- System development shall follow the V-Model software development processes
- The system software shall have a modular approach, to allow future maintainability and addition of new features
- The system shall be designed for the addition of autonomous driving features
- The system software shall be inaccessible outside of the MVSL
- The control system shall have an update rate of at least 10ms
- Reliable and robust against possible errors or miscalculations

Functional requirements identified for the urban vehicle's control software are outlined in table 3.1 below.

Req. ID	**Requirement**
1.1.1	Each ECU that makes torque decisions shall maintain its own internal control state
1.1.2	The vehicle shall be in only one of four states at any given time; park, reverse, forward or autonomous mode
1.1.3	The control software shall ensure that pre-charge is completed upon startup
1.1.4	Each corner module shall have the ability to act independently of the other modules
1.1.5	The control system shall have parameters to limit the torque and velocity of the vehicle if desired

1.1.6	The control system shall monitor the hydraulic system to ensure that 1300-1400 PSI of pressure is being provided for the front steering to operate
1.1.7	The control software shall only utilize inputs available through the joystick and switches for all functions
1.1.8	The control software shall allow the vehicle to perform front and rear steering during low speeds, but linearly decrease the rear steering until it is unavailable at speeds over 20km/h to maintain vehicle stability
1.1.9	The control software shall monitor the charge of the low voltage batteries, to charge them from the high voltage pack when required
1.2.1	The control software shall monitor the motor controller temperature to ensure it is under 100°C, if the temperature is within 20°C of the limit the performance shall be decreased
1.2.2	The control software shall monitor the motor temperature to ensure it is under 130°C, if the temperature is within 20°C of the limit the performance shall be decreased
1.2.3	The control software shall monitor all steering and motor feedback to determine if the actuator or feedback has failed
1.2.4	Each fault detected within the subsystems that affects the dynamics of the vehicle shall be reported to other ECUs
1.2.5	The control system shall enforce a 15km/h speed limit when a steering actuator has failed, to increase vehicle stability and allow all actuators to function
1.2.6	The control software shall cut torque to the motors if steering feedback has failed
1.2.7	The control software shall cut the torque if two or more steering actuators have failed
1.2.8	The control system shall take corrective actions to allow the vehicle to be drivable when a steering actuator has failed
1.2.9	The control system shall allow the vehicle to be drivable if one motor fails
1.2.10	When a steering actuator fails, torque shall no longer be sent to that wheel

1.3.1	The control system shall continuously monitor the vehicle's faults and only allow the driver to enter an autonomous state if the function is available, there are no faults that change the dynamics of the vehicle and the IMU/GPS is available
1.3.2	When in autonomous mode, the control software shall allow the driver to take control of the vehicle if it detects intent from the driver to do so
1.3.3	When in the autonomous state, the control module shall monitor the inputs from the autonomous system to ensure they are within the expected range, or else the vehicle shall return to a manual control state

Table 3.1 Vehicle Control Software Requirements

3.3 Lane Following Software Requirements

The vehicle's lane following software will provide the base for the autonomous software of the vehicle. The lane following algorithm will be made-up of two parts, a perception software detecting the lanes and the path tracking (control) software steering the vehicle within the lanes. The control portion of the system is safety critical, whereas the perception algorithm is not safety critical as the vehicle can still be operated without it. The non-functional requirements for these systems are as follows;

- System development shall follow the V-Model software development processes
- The design shall be modular to allow additional autonomous driving features to be added or the autonomous algorithms to be swapped
- The control algorithm shall be compiled on the safety critical ECUs
- The perception algorithm shall have an update rate of at least 4fps

The functional requirements for the lane following system are outlined in table 3.2 below.

Req. ID	**Requirement**
2.1.1	The lane lines detected by the perception software shall be made up of at least five points ahead of the vehicle

2.1.2	The perception software shall accurately detect at least one lane for the entire drive cycle around Ring Road
2.1.3	The autonomous software shall send the control system the lane locations in meters referenced from the vehicle's current position
2.2.1	The control software shall be able to maintain the vehicle on its intended path
2.2.2	The control software shall be tuned for minimal jerk to increase passenger comfort
2.2.3	The control system shall continuously monitor the autonomous system to determine if it has failed or it is unavailable
2.2.4	The control system shall determine the vehicle's path when lane lines are detected

Table 3.2 Lane Following Requirements

3.4 Software Test Cases

The software test cases were derived from the requirements to ensure that all of the requirements for the vehicle were met. The test cases can be broken down into unit testing, and validation testing of the vehicle. To bridge the gap between the two testing procedures integration testing was also performed as presented in the V-Model. Table 3.3 below provides a sample of the critical software test cases that the vehicle was required to complete, along with the requirement that the test case corresponds with.

Test ID	**Req ID**	**Procedure**
Unit Testing:		
1.1	**1.1.1, 1.1.2, 1.1.7, 1.3.1**	The following simulated inputs must be provided to the function responsible for vehicle state; joystick buttons, IMU/GPS availability, autonomous functions availability and steering faults. The expected behavior is; start in park state, then enter reverse/forward after drive button is pressed. Finally enter autonomous state when selected if IMU/GPS is available, autonomous functions are available and there are no steering faults.

1.2	**1.2.1, 1.2.2**	The following simulated inputs must be provided to the function responsible for motor torque; motor temperature, motor controller temperature and joystick torque command. Then the torque request from the motor must be monitored to ensure that it is limited when the motor or motor controller is approaching its temperature limit.
1.3	**2.2.1, 2.2.2**	For this unit test the vehicle will have a path being fed to it, the function responsible for path following will then command a steering output, the jerk and error of the vehicle's path tracking will then be monitored. Finally, the function will be tuned to reduce jerk and the vehicle's path tracking error.
System Testing:		
1.4	**1.2.3 1.2.5, 1.2.8, 1.2.10**	This test is to test the steering fault tolerance. To perform this test, a constant steer position will be sent to a corner module to simulate an actuator failure. In all three cases the vehicle shall still be drivable and be predictable.
1.8	**1.2.3, 1.2.9**	This test is to ensure the vehicle can continue to function after a motor fails. To perform this test, one of the motors shall be disconnected in the middle of the drive cycle, the vehicle must be able to operate with the motors that are operating, even if performance is reduced.
1.5	**1.3.2**	This test is to ensure the driver can exit from autonomous state. To perform this test the vehicle must be in an autonomous state, then when the driver attempt to take over control of the vehicle the control shall be transferred to the driver immediately.
1.6	**2.1.2, 2.2.1, 2.2.3**	This test is to validate the lane following function. The vehicle shall be driven in autonomous mode for one lap around Ring Road, the test will be passed if the vehicle can complete the lap without driver intervention while staying within the lane.

Table 3.3 Sample System Test Cases

Chapter 4

System Design

With the vehicle configuration finalized and the expected performance of the control system clearly outlined, the detailed design of control system was started. The modular design of the system was vital for this stage to ensure functions could be reused and that they would be able to work together. This section will focus on the selection of the processor, design of the control system and design of the lane following system.

4.1 Electronic Control Unit Board Selection

The processor boards to be used within the vehicle are a crucial component as they are tasked with controlling all functions of the vehicle and are expected to last the life of the vehicle. Any failure of the processors during vehicle operation can lead to a catastrophic system failure, and potentially an accident.

The requirements for the control system processor board are included in section 3.2.2. Along with the ECUs control system requirements it was determined that it would be advantageous if the ECU had IoT capabilities. The main idea behind this was to allow the MVSL to be able to collect vehicle data, as well as update the control software efficiently and much quicker than previously possible. To allow this the processor would not only have to meet the strict safety requirements previously outlined, but also allow for internet communication. To achieve this goal, various avenues were explored including; automotive processor boards, industrial computers, single board computers (SBC), and embedded processor boards.

The first option that was explored were automotive processor boards as they were the obvious choice. The two preferred manufactures with proven automotive processor boards were Texas Instruments (TI) and NXP, both utilizing ARM Cortex processors. The automotive boards met all the safety and real-time requirements outlined for the control system. However, they did not have expandable memory or enough memory to store vehicle data, and they also were not IoT capable.

Secondly, the possibility of utilizing an embedded processor board with IoT capability was explored. TI and STM both offered embedded boards that are suitable for controls applications and are IoT capable. At first glance this seemed like the optimal solution, however there were many issues with this approach. Firstly, the boards did not support CAN communication, and did not meet the robustness requirements of the automotive boards, such as operating temperature, as well as ASIL rating as defined by the ISO 26262 automotive standard. Furthermore, these boards still did not provide enough memory or expandable memory to store vehicle data.

Then, the possibility of utilizing SBCs was explored with no shortage of options with offerings from various companies such as; Asus, Raspberry Pi, Arduino, LattePanda, and Minnowboard to name a few. All of these SBCs included capable processors, large or expandable memory, internet connectivity and plenty of I/O. However, similarly to the embedded processor boards they did not have a CAN interface, and they were not automotive grade. Most of the boards also run an operating system that only provides a best effort response time, but not a guaranteed real-time response. This makes the boards unsuitable to run the control software, as failures and missed timing requirements are likely to occur.

Next, the possibility of using industrial computers was explored. The main offerings explored were provided by Portwell, Advantech, and Connect Tech Inc. Many of the same problems were still present, the computers were either an 'IoT Gateway' not supporting real-time control applications, while others had CAN and real-time support, but not IoT capabilities. Additionally, the cost of the industrial computers was higher than our price range and did not include an automotive ASIL rating.

In conclusion, after determining that none of these options could fulfill the full list of the requirements it was determined that the optimal solution would be to have two boards mounted on the PCB inside the ECU enclosure communicating with each other to fulfill the requirements. It became clear that the automotive boards could fulfill the crucial control requirements, while the SBCs could fulfill the desired IoT requirements. The architecture is summarized in figure 4.1 below.

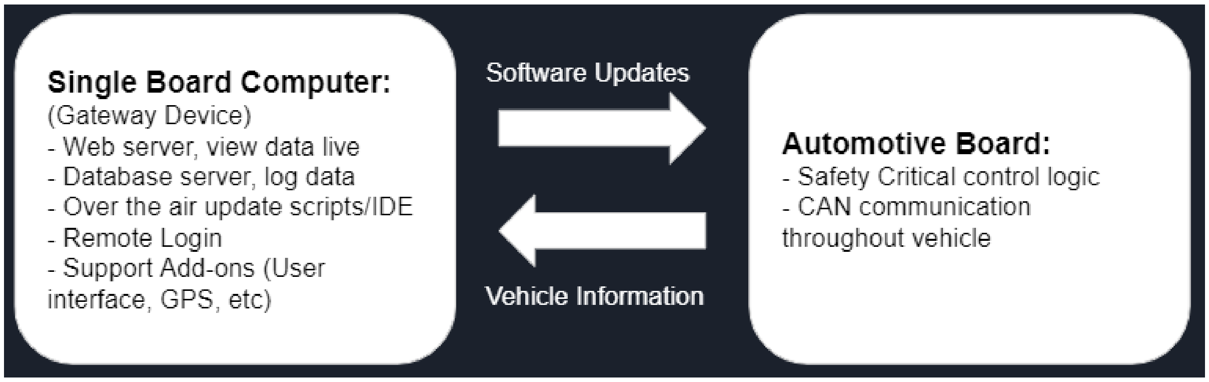

Single Board Computer:
(Gateway Device)
- Web server, view data live
- Database server, log data
- Over the air update scripts/IDE
- Remote Login
- Support Add-ons (User interface, GPS, etc)
Software Updates
Vehicle Information
Automotive Board:
- Safety Critical control logic
- CAN communication throughout vehicle

Figure 4.1 Electronic Control Unit Proposed Architecture

Once the processor architecture of the ECU was selected, the SBC and the automotive processor board were chosen, to meet all of the requirements, as outlined in the previous chapter. The requirements and processors of choice where handed to a third-party to complete the design, as well as the manufacturing process. The full design is unavailable as it is proprietary to the MVSL.

4.2 High-Level Control System Design

With all the vehicle components selected and clear requirements outlined the design of the control system began. When designing the control system it was very important to continue the modular approach to allow the design to be extended to other vehicles in the future. It was also very important to ensure that only the necessary information would be communicated with other controllers to decrease bus traffic and handle as many functions as possible locally. The control system will be fundamentally split into three crucial levels; the lower level control, the vehicle supervisory control and the autonomous functions.

It was decided early that to continue the modular approach, as well as the ability to add more fault tolerant features to the vehicle that each corner module would have an ECU. These ECUs will be responsible for the operation of that corner's steering system, motors and being able to detect the faults that may occur. There will also be a body control module that would be responsible for the operation of the hydraulic power pack, the DC-DC converters, as well as communicate with the brake pressure sensors, and any other sensors that may be added to the vehicle in the future. The three corner module controllers and the body controller complete the lower level control of the vehicle.

Most commonly, a fault occurs due to the failure of an actuator or a sensor. Detection of an actuator failure is made simple by the fact that the actuators typically have feedback loops, either internally or sensors that are incorporated within the system. Sensor failures are detected when sensor readings become erratic, change at an unexpected rate, or are outside their expected range. If an estimation is available, sensor failures can also be detected when the estimations do not agree with the sensor readings, and a faulty sensor's readings may even be replaced by this calculation. Due to the fact that the lower level controllers are interfacing directly with the sensors and actuators, they are responsible for providing feedback to the vehicle level controller regarding faults that occur.

Next, there will be a vehicle supervisory control that will evaluate the inputs from the driver or the autonomous system and will be focused on fulfilling these requests. Inputs to the supervisory controller include; the joystick commands, the commands from the autonomous system, the feedback from the lower level controllers, the IMU, as well as the GPS. When the failures do occur the supervisory controller is responsible for detecting, evaluating and mitigating failures that can impact the dynamics or drivability of the vehicle. In these cases, the performance may have to be reduced to ensure passenger safety, which is paramount. When engaged in an autonomous mode, the supervisory controller is responsible for ensuring that no failures are present that will affect the vehicle's drivability, and that the commands received from the autonomous software are viable. The intended flow of information within the system is shown in figure 4.2 below.

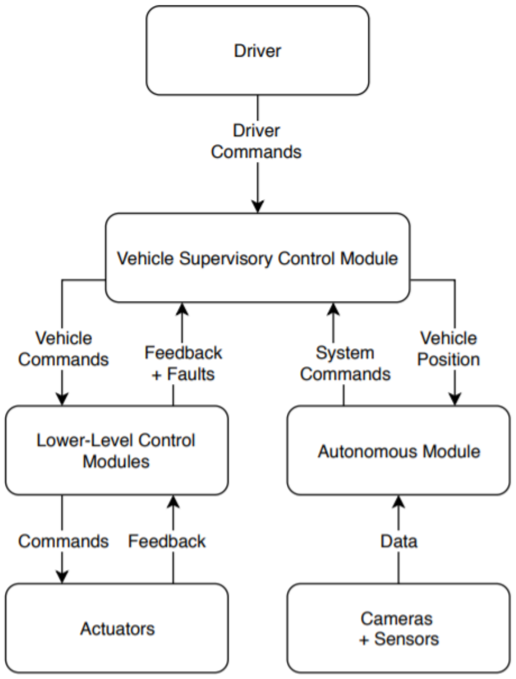
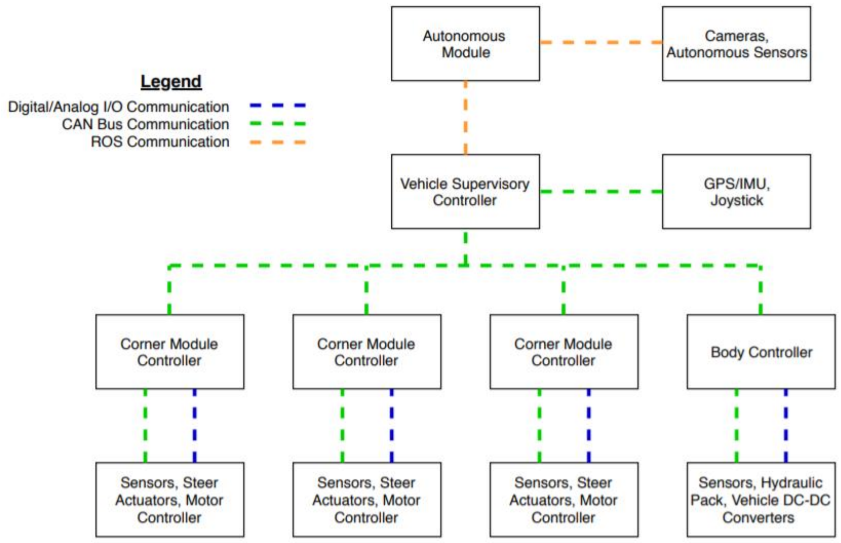


Figure 4.2 Control System Information Flow

Finally, there is the vehicle's autonomous functions which are no longer safety critical for the drivability of the vehicle, but must be in a safe state when being used. For the scope of this project the autonomous driving functions will consist of lane detections using a machine learning model. In this case, the path will be generated from within the vehicle supervisory control based on the locations of the lanes. However, the system has been designed in a way that it will be very easy to replace this lane detection model by full autonomous software that just sends a path with the respective velocities to the vehicle's supervisory control system.

In terms of communication there are two main communication protocols that will be used for the safety critical control system; digital or analog I/O, and various Controller Area Network (CAN) buses. The digital or analog I/O will only be utilized to communicate with the sensors and actuators at the lower control level when required. For example, the motor controller specifies that a 0-5V signal must be received as a throttle signal. The remaining actuators, sensors and all ECUs will then communicate with each other over a CAN bus. The CAN bus makes it easy to add nodes in the future, it reduces the cost of wiring, and it is robust against failures of subsystems, as well as interference.

The vehicle's autonomous functions will mainly communicate using the Robot Operating System (ROS). This will allow communication between the autonomous module with the vehicle's supervisory control, as well as between the autonomous module and corresponding hardware, such as cameras, radar and LIDAR. ROS was chosen for the communication of the autonomous module because it is independent of programming language, it is open sourced and has an abundance of resources available. ROS is a collection of open source software libraries with drivers readily available to interface with sensors, and cameras. ROS also provides various advantages when it comes to testing, as it is easy to use data acquisition, playback and visualization functions [40]. The final architecture of the vehicle's control system hardware is as seen in figure 4.3 below. The electrical connections required for each ECU to fulfill this architecture are summarized in Appendix A.



Legend
Digital/Analog I/O Communication
CAN Bus Communication
ROS Communication
Autonomous Module
Cameras, Autonomous Sensors
Vehicle Supervisory Controller
GPS/IMU, Joystick
Corner Module Controller
Corner Module Controller
Corner Module Controller
Body Controller
Sensors, Steer Actuators, Motor Controller
Sensors, Steer Actuators, Motor Controller
Sensors, Steer Actuators, Motor Controller
Sensors, Hydraulic Pack, Vehicle DC-DC Converters

Figure 4.3 Vehicle Control Architecture and Communication

The high-level architecture of the control system achieves all of the goals set out for the vehicle. The modular approach allows each corner module to be controlled separately, making the addition of new actuators or sensors easy to test and prototype. The same concept is used for the vehicle's body controller, allowing the braking system, hydraulic pack and other sensors to easily be added or removed. The dynamics of the vehicle are completely isolated in the vehicle's supervisory control that is only concerned with meeting the drivers or autonomous systems commands. Then, the autonomous functions can be edited completely independent from the rest of the control system, requiring only minor changes in the supervisory controller to expand the lane following algorithm to a fully autonomous system. The final signals to be communicated between the modules are summarized in table 4.1 below.

Module	**Inputs (Origin)**	**Outputs**

CornerModule (CMC)	MotorFeedback(Kelly Controller)
	ActuatorFeedback(Sensors/CAN)
	ActuatorFaults(CAN, if available)
	SteerCmd(VSC)
	TorqueCmd(VSC)
	Joystick(CAN)
		SteerFeedback
		SteeringFaults
		MotorFaults
		MaxTorque
		MotorAvailable

Body Module (BMC)		HydPackFault

		BrakeApplied BrakePressure LowVoltFault

VehicleSupervisory (VSC)	HydPackFault(BMC)
	BrakeApplied(BMC)
	BrakePressure(BMC)
	LowVoltFault(BMC)
	SteeringFaults(CMC)
	MotorFaults(CMC)
	SteerFeedback(CMC)
	MaxTorque(CMC)
	MotorAvailable(CMC)
	Joystick(CAN)
	IMU/GPS Data (CAN)
	AutonomousPath(AM)
		SteerCmd
		TorqueCmd
		IMU/GPS

AutonomousModule	IMU/GPS Data(VSC) CameraData(ROS)	AutonomousPath

Table 4.1 Electronic Control Unit Communication Summary

4.3 Fault Tolerant Control

Mechanically each corner module is completely independent allowing the vehicle to mitigate failures through the use of other actuators. In terms of the fault tolerant control there are three main areas that the control system focuses on, sensor failure, actuator failure, and motor failure. Each one will be covered in this section, along with how the potential failures will be mitigated or how the vehicle will be brought to a safe state.

Sensor failures can be particularly dangerous because the vehicle's control system no longer has a feedback loop. The sensors with the highest probability of failure are the front steering string potentiometers because they are exposed, and not as resistant to the elements as the other sensors. Unfortunately, an estimation for the wheel position, and camber would be beyond the scope of the project as it is dependent on the changing hydraulic pressure, the stepper motors controlling fluid flow, and the forces that each wheel is experiencing. The rear steering has a more robust system as it has an internal control loop and reports all of its feedback through the CAN bus. If there is no feedback or the feedback is outside of the expected range for the steering actuators, then the vehicle will detect the fault and will disallow torque to the motors to protect the occupants.

Motor failures or motor controller failures for the vehicle are less likely, but essential to mitigate. Feedback from the motor controllers is received through the CAN bus. To ensure that these connections are reliable, the control system will monitor that there is consistent feedback received from the motor controllers and ensure there are no faults being broadcasted. Based on motor availability, motor temperature and motor controller temperature, the vehicle will select one of three different drive configurations: all-wheel drive, front wheel drive or rear wheel drive.

The final task of the control system is to mitigate steering actuator failures. The first type of steering actuator that can fail is the camber actuator, which is an added complexity to the front steering, but can provide benefits in terms of vehicle balance. These are vital to keep the vehicle from rolling over, and ensure that the front of the vehicle is at an appropriate height. If these actuators are detected to be outside their safe operating range, the control system will disallow torque to all motors in an attempt to keep the occupants safe.

The second type of steering actuator encompasses the three steering actuators that solely control the steering of each wheel. For the front actuators these failures are detected if the vehicle's actuator length is not within an error tolerance range or the error is not decreasing. For the rear actuator the faults will be monitored via the CAN bus and the actuator length will be monitored to be within an error tolerance range. Some rear steering failures can be handled locally within the corner module, for example the overload failure that just requires a quick reset of the actuator. However, if a hydraulic front steering actuator is immobile or a serious fault occurs in the rear, the supervisory controller must act to control the vehicle's dynamics. To simplify the problem to just account for lateral tire forces, when a steering actuator fails the corresponding motor's torque will be disabled. If a steering actuator in the front fails the other front actuator will steer to the same angle, but in the opposite direction as the actuator that failed to compensate for the lateral force. If the rear steering fails, a vehicle model will be used to estimate the lateral force provided by the rear wheel and calculate the steering angle required by the front wheels to compensate.

This calculated value will be the vehicle's temporary neutral steering position, and the driver will be able to steer the vehicle with the remaining degrees of range of the front steering.

4.4 Vehicle Model

As mentioned earlier the vehicle model will be vital to ensure that the control system can compensate for rear steering faults. The vehicle model has been derived from the bicycle model using a linear tire model to allow the system to estimate the forces at each wheel, the vehicle's yaw, and lateral velocity. The model provides an estimation, but does not account for vehicle roll or pitch motions. The equations will correspond to the bicycle model with rear steering, however the equations have been extended to account for two wheels in the front creating the tadpole vehicle formation. The following set of equations defines the model;

$$m(\dot{v} + ur) = F_{yf1} + F_{yf2} + F_{yr}$$

$$I\dot{r} = a\big(F_{yf1} + F_{yf2}\big) - b\ (F_{yr})$$

$$F_{yf} = C_f \alpha_f$$

$$F_{yr} = C_r \alpha_r$$

$$\alpha_f = \delta_f - \frac{v + ar}{u}$$

$$\alpha_r = \delta_r - \frac{v - br}{u}$$

Where, v is lateral velocity, u is longitudinal velocity, δ_f/δ_r are the wheel angles, C_f/C_r are the tire cornering stiffness, F_{yf}/F_{yr} are lateral tire forces, r is yaw, m is mass, I is mass moment of inertia, and a/b are distances to the CG of the vehicle as seen in figure 4.4 below.

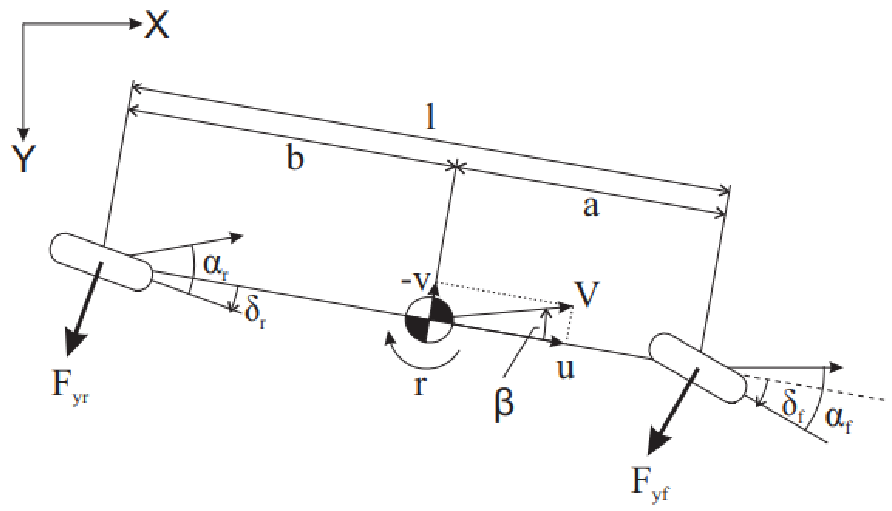


Figure 4.4 Bicycle Model with Active Rear Steering [46]

The model utilized for the vehicle's estimation is valid within the linear region of the lateral force, namely when the tire's slip angle is small. This will be the case in the driving scenarios that the urban vehicle will encounter as it will not be driven in slippery scenarios, at high speeds or for aggressive driving maneuvers. The constant parameters for the vehicle model were already available through previous research done in the MVSL. The vehicle model will be vital when the rear steering fails, using the vehicle model a new neutral position will be selected for the front wheels to compensate for the lateral force that the rear wheel is providing.

4.5 Detailed Control Software Design

After defining the vehicle's control architecture, as well as the hardware to be utilized, the next phase of the design process was to create a detailed design of the software functions within the control system. In this section, each control module from the high-level design will be presented along with the software functions that it will contain. Each software module's responsibility will be well defined, along with the inputs and outputs to ensure there is no confusion in the intended interface with other system modules. This will also allow for unit testing of each module independently before integration of the system begins.

At a high-level each module will contain very similar functions that can be broken down into four different groups; data input functions, system diagnosis functions, system control functions, and data output functions. Both the data input and data output subsystems will have the same functions in all modules. They are responsible for processing all the data being passed into and returning from their respective software module. The data input functions will decode, scale, and change the data type of all the inputs to maximize the processor's computational speed. The data output functions will once again scale the data, ensure the module outputs are within the expected range to reduce the probability of hardware failure, perform data type conversions, and finally encode the data before sending it out. The responsibilities and flow of information is summarized in figure 4.5 seen below.

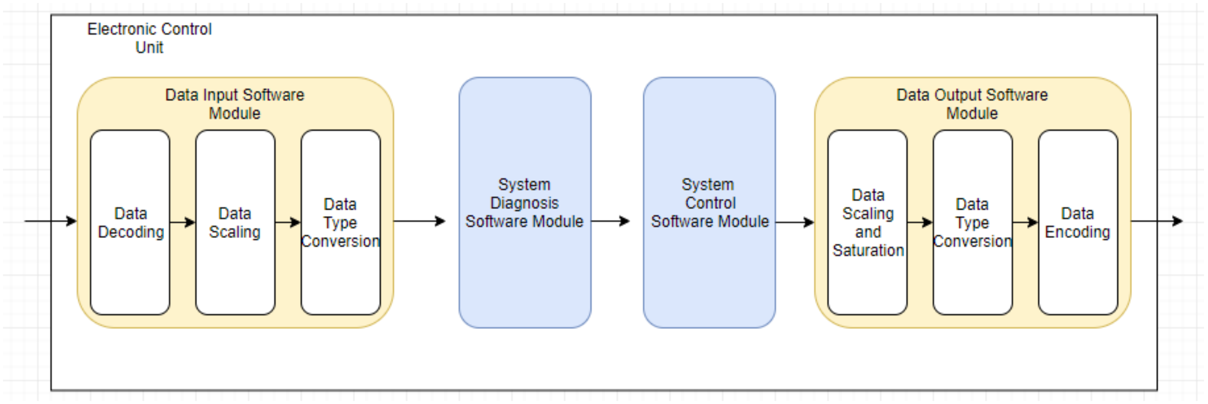

Electronic Control Unit
Data Input Software Module
Data Decoding
Data Scaling
Data Type Conversion
System Diagnosis Software Module
System Control Software Module
Data Output Software Module
Data Scaling and Saturation
Data Type Conversion
Data Encoding

Figure 4.5 Controller Functions Summary

The functions of the system diagnosis, as well as system control will be reviewed separately, and in greater detail for each module because they vary greatly. A formal Module Interface Specifications (MIS) was created to highlight the externally observable function of each module, along with its states, constants and interfaces between them. In the following sections a summary of these will be presented.

4.5.1 Corner Module Control Software

The corner modules of the vehicle will not contain the same hardware, however they will contain the same functions, demonstrating the modularity of the software. The corner modules are vital because they interface with the hardware directly, through CAN, digital or analog signals. For the vehicle's corner modules the system diagnosis functions will be responsible for detecting faults, determining the vehicle's state, and providing estimations to the system control functions. Table 4.2 below summarizes the responsibility of each software module, along the expected inputs, outputs and the requirements that it maps to.

Function	**Responsibility**	**Inputs**	**Outputs**	**Req. ID**
System Diagnosis Functions:				
VehicleState	Determine the vehicle's state as either park or drive state	Joystick buttons	Vehicle State (Uint8)	1.1.1, 1.1.2
MotorFaults	Determine if motors are available, as well as the maximum torque that can be requested based on motor and inverter temperature	Kelly Motor Controller Feedback	Motor available (Boolean), Max Torque (Uint8)	1.1.3
ActuatorFaults	Determine if any actuators have failed, on a temporary or permanent basis	ActuatorPos, Faults (if available)	TempFault (Boolean), PermFault (Boolean)	1.2.3
FeedbackFaults	Determine if the devices providing feedback are operating correctly, without erratic behavior	ActuatorPos	SensorFault (Boolean)	1.2.3
SteeringAngleEst	Determine the wheel's current angle based on the length of the actuator	ActuatorPos	SteerAngle (Uint16)	1.2.3
System Control Functions:				
MotorControl	When motor is available fulfill the torque request, broadcast torque percentage available from the motor based on temperatures	MaxTorque TorqueCmd	MotorTorq (Uint8)	1.1.4, 1.2.1, 1.2.2
SteeringControl	Fulfill the steer command quickly and with minimal overshoot	ActuatorPos, SteerCmd	Motor1Dir Motor2Dir (Boolean) Motor1Pulse Motor2Pulse (Uint8)	1.1.4

Table 4.2 Corner Module ECUs Main Function

4.5.2 Body Control Software

The vehicle's body controller is critical to control or receive feedback from subsystems that would service the corner module or vehicle supervisory controller. The body control module is the final ECU that will be interfacing with hardware directly. These include, the brake system, the DC-DC converters and the hydraulic power pack. The functions contained within the body control module are summarized in table 4.3 below.

Function	Responsibility	Inputs	Outputs	Req. ID
BrakeMonitor	Monitor the brake pressure sensors and indicate when the brakes are being applied	BrakePressure1 (analog) BrakePressure2 (analog)	BrakePress (Boolean)	1.3.2
DCDCPower	Monitor the low voltage batteries on the vehicle and close the relay to the DC-DC when the voltage is low to allow the batteries to charge	BatteryV1 (analog) BatteryV2 (analog)	DCRelay1 (Boolean) DCRelay2 (Boolean)	1.1.9
HydraulicPack	Monitor the pressure of the hydraulic steering, and close the relay of either one or both pumps to turn them on when the pressure has dropped below the threshold	Hydraulic Pressure (analog)	HydRelay1 (Boolean) HydRelay2 (Boolean)	1.1.6

Table 4.3 Body Control ECU Main Functions

4.5.3 Vehicle Supervisory Control Software

The vehicle's supervisory control will be critical to allow the vehicle to overcome hardware faults that occur, as well as allow the autonomous functions to take control of the vehicle. This control module is more concerned with the dynamics of the vehicle and ensuring that the vehicle is always in a safe state. The vehicle supervisory controller is also responsible for ensuring that when the autonomous module provides a path, the vehicle will follow the intended path. It is important to note that the supervisory control will contain the vehicle model to create the required estimations, as well as the Pure Pursuit path tracking algorithm, that can easily be swapped. Table 4.4 below will summarize the functions found within the supervisory controller.

Function	**Responsibility**	**Inputs**	**Outputs**	**Req. ID**
VehicleState	Determine the vehicle's state as either, park, supervisory control or autonomous control	Joystick buttons, Steering fault, Autonomous Fault, IMU Availability	Vehicle State (Uint8)	1.1.1, 1.1.2, 1.3.1
VehicleModel	Estimate the vehicles Vy, Yaw and tire forces to make corrections when failures occur	Steering angle of all wheels, Vx	Vy (Single) Yaw(Single) TireForces (Uint16)	1.2.8
DesiredSteer	Take the joystick input and translate it to a desired wheel angle	Joystick X position, neutral, left and right	DWheelAngle (Int16)	1.2.8
PathTracking	When given a path, determine the steering angle required to keep the vehicle on the path	Path (5 points), Vx	SteerCmd (Int16)	2.2.1, 2.2.2
Autonomous Fault	Continually monitor the autonomous path to ensure it is feasible, and monitor that the driver is not trying to take control of the vehicle	Path (5 points), JoystickPos, BrakePress Vx	AutoFault (Boolean)	1.3.2, 1.3.3, 2.2.3
FrontSteer	Based on faults, vehicle model, and joystick command determine the steering angle for each of the front wheels	Front and Rear Steer Faults, Rear Wheel Force, Vx, Vy, Yaw, Steer Command	FLSteerAngle FRSteerAngle (Int16)	1.1.8, 1.2.8
RearSteer	Based on faults, vehicle model, vehicle velocity, and joystick command determine the steering angle for the rear wheel	Front and Rear Steer Faults, Vehicle State, Vx, Steer Command	RearSteer-Angle (Int16)	1.1.8, 1.2.8
MotorTorque	Monitor motor availability, limits, and faults to distribute torque to the three wheels	Front and Rear Steer Faults, Motor Faults, Available Motor Torque, Torque	TorqueFL TorqueFR TorqueRear (Uint8)	1.1.5, 1.2.5, 1.2.6, 1.2.7, 1.2.9

		Limit, Speed Limit		

Table 4.4 Supervisory Control ECU Main Functions

4.6 Lane Detection

For the vehicle's lane detection it was decided early to take the machine learning model route because it is more robust to noise and disturbances found in the real-world. Due to the fact that the vehicle would be tested here on campus around Ring Road, this was vital because it is common for students, shadows, vehicles, buses and cyclists to occasionally cover or block the lanes. Once this was decided the next phase was to decide on a machine learning model that could accurately detect the lanes on Ring Road. The main goal was to find a model that would accurately detect the lanes, and with a detection rate that was adequate for the driving speeds of the urban vehicle as outlined in the requirements. The faster the detection rate the better, to allow the lane detection model to be utilized in other vehicles in the MVSL that are capable of driving at faster speeds.

After completing a thorough overview of machine learning models available it was clear that there were two models with the potential to meet the urban vehicle's requirements and provide benefits that could allow them to be used in other vehicles in the lab. It is important to note that transfer learning could be applied to these models to attempt to further improve accuracy. The first was ICNet that produces semantic segmentation, pre-trained with 30 different classes of objects and providing over 70% accuracy using the intersection over union (IoU) method at a real-time rate. The IoU method measures the overlap between the ground truth pixel labeling and the predicted pixel labeling. ICNet does not explicitly label each lane, however it labels the entire road which is sufficient for Ring Road which only has two lanes. The main advantage of ICNet is its ability to not only detect the road, but many other objects that will be required to be detected for future autonomous vehicle detection. On the other hand, the processing rate of the model would have to be tested on our hardware, as well as the detection accuracy of the road surface. This is due to the model being relatively slow and the detection of the road accuracy must be much higher than 70% for it to be useful in our scenario.

The other model that was further examined was SCNN, which only provides the detection of the lanes, but it has been proven to work with great accuracy. The only drawback is that unlike ICNet, the neural network architecture is very specific to lane detection and cannot be utilized to detect other object. A solution to this could be to use two distinct networks, one to detect the lanes and another to detect the objects of interest.

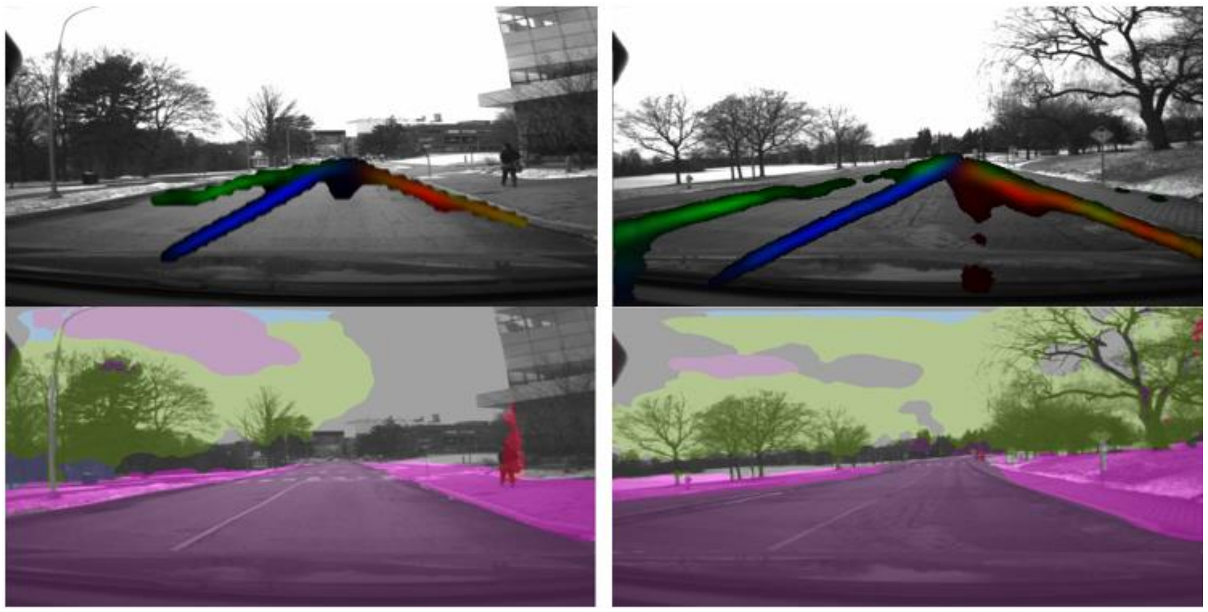


Figure 4.6 Lane Detection Model Sample Results

Data had been collected previously while driving around Ring Road, these videos were used to test both machine learning models for a comparison of accuracy, robustness and speed of detections. The ICNet will output an image the same size as the original except each pixel will be the color of the corresponding class of the object in that location. For example, if there is a sidewalk in the original image, in the output it will just be pink pixels. The original and output image are then overlaid with some transparency to easily visualize what has been detected. On the other hand, the SCNN creates a pixel level probability map which then creates a prediction of existing lane markings using cubic splines. It returns four different classes, one for each lane line each marked in a different colors allowing it to identify up to three lanes on the road. Both machine learning models performed very well in good lighting, when the curb is clearly visible and the road is relatively straight. Figure 4.6, which is shown above exemplifies a comparison of both models under these more ideal condition.

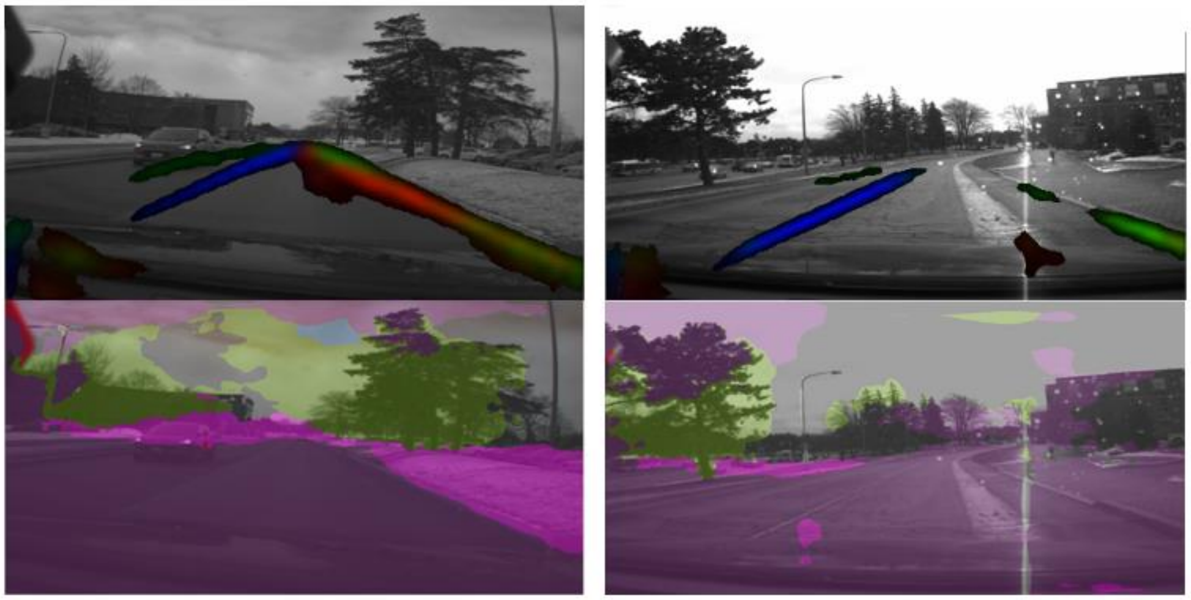


Figure 4.7 Lane Detection Model Sample Results Continued

However, when conditions were far from ideal SCNN outperformed ICNet providing much more reliable predictions. Usually ICNet's performance would suffer when the image was darker, the curbs were not visible or the height of the sidewalk blended with the road. Figure 4.7 above provides examples of scenarios where SCNN outperforms ICNet. The first image on the left is darker than most, and ICNet has trouble labelling the road on the left side, marking the oncoming vehicle as part of the sidewalk and the sidewalk as a road surface. The second image on the right ICNet has trouble detecting both lanes, and creates a dangerous situation by labelling the entire sidewalk as a road surface. They were both also tested on the same platform to gauge the computational efficiency of each model, they were both very similar in this regard. The ICNet performed at an average speed of 11.7 fps, where as the SCNN performed at 10.2 fps.

Due to its increased robustness, reliability and comparable speed SCNN was selected as the neural network of choice for the vehicle's lane detection. However, without performing any transfer learning the neural network was still having difficulty detecting the correct lane through a few sections of Ring Road, as well as detecting only one or no lanes through the intersections. The lab had already created a detailed map with coordinates of Ring Road. Then, Minghao Ning, a visiting student at the MVSL, created an algorithm that would take the map of Ring Road, GPS input, IMU input and the input from the selected SCNN to create the best possible prediction. The GPS creates a prediction of the vehicles position with respect to the map, the SCNN would provide a prediction of the lanes, while the IMU help create a prediction of the new lane location based on the previous time step.

The SCNN model has four classes, meaning that it can detect up to four distinct lane lines. The model then returns an array of points on each time step. The array will include a point for every 10 pixels in the y-direction that the model has identified a line. This array must be split up into four to be able to identify each lane individually. Once each lane has been identified the two closest lanes to the middle of the image are identified. Then for each lane line, the first point of line, the last point of the line and three other evenly distributed points are selected. These points are then translated into lane locations with respect to the vehicle using camera height and camera angle. Finally, these two sets of five points, each set representing a lane, are returned to the control system to meet the input requirements. Minghao Ning also provided a copy of his algorithm that relies solely on the SCNN to make the lane predictions. Both of the the lab's lane detection models would be tested to validate the control system.

4.7 Path Tracking

A path tracking control algorithm was selected to ensure that the vehicle would be able to follow the intended path when the vehicle is in the autonomous mode. After carefully reviewing path tracking algorithms the pure pursuit algorithm was chosen because it performed best in slow driving situations, its path error robustness, as well as discontinuous paths. This would be vital to help the vehicle stays within the lanes even if the lane detection algorithm fails and the path becomes discontinuous. When in autonomous mode the vehicle will not use rear steering to simplify the calculation of the required steering to follow the desired path. As discussed earlier, Pure Pursuit finds the vehicle's steering angle (δ) using only the angle (α) between the vehicle's heading vector and the look-ahead vector, as well as the goal point which is determined by the look-ahead distance (l_d) [36]. First, the curvature (κ) of the circular arc must be calculated, and then the steering angle can be calculated using the vehicle's wheelbase (L). Applying the law of sines the derivation of curvature calculation is as follows [36];

$$\frac{l_d}{\sin(2\alpha)} = \frac{R}{\sin(\frac{\pi}{2} - \alpha)}$$

$$\frac{l_d}{2\sin(\alpha)cos(\alpha)} = \frac{R}{\cos(\alpha)}$$

$$\frac{l_d}{\sin(\alpha)} = 2R$$

$$\kappa = \frac{2\sin(\alpha)}{l_d}$$

Then, using the bicycle model the steering angle can be found using the following equation;

$$\delta = \tan^{-1}(\kappa L)$$

The final portion of path tracking is a function to get the goal point that the Pure Pursuit requires. These goal points will generate the overall path that the vehicle is to follow. This function will also output a fault if there are no lanes detected, there's a failure in the goal point calculation, no look-ahead distance is given or the driver attempts to steer the vehicle. The vehicle state function will then return the control to the driver if this fault is received or no IMU/GPS is available, therefore covering all the fault mitigation requirements originally outlined for the vehicle.

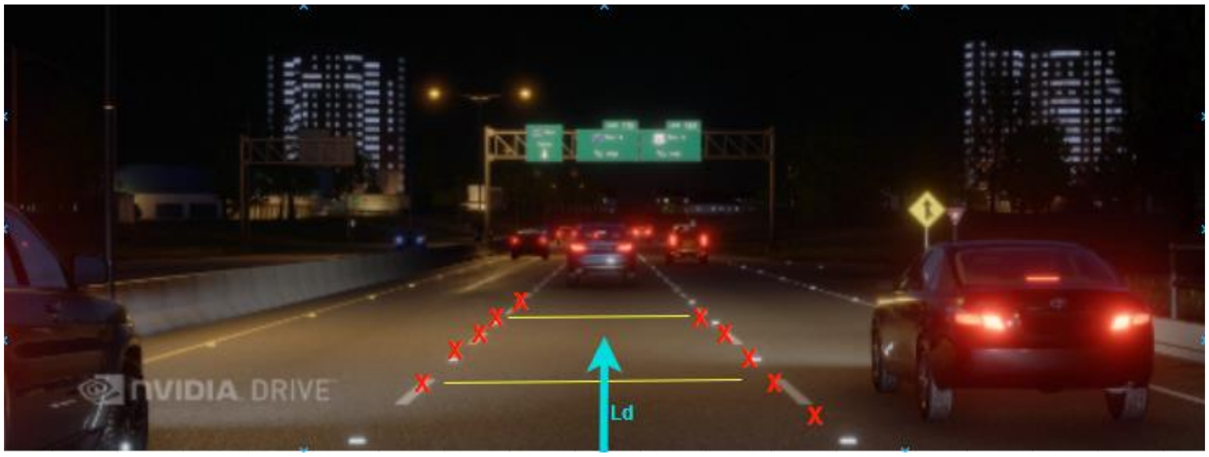


Figure 4.8 Illustration of Path Following Algorithm

The lane detection algorithm sends the control system five points for each lane, marking the beginning of the lane, the end of the lane and three points in between. Therefore the goal point function receives ten points, along with a look-ahead distance which is a distance in front of the vehicle. An example has been illustrated in figure 4.8 below, where each red "X" illustrates a point received from the lane detection algorithm, and the blue arrow signifies the look-ahead distance. At the look-ahead distance it will interpolate the lane points to recreate the lane line and find the distance to the right and left lane. Therefore, the look-ahead distance must be a distance away where both the left and right lanes have been detected, these are illustrated by the yellow lines seen in figure 4.8 above. Then, the point in between both of the lanes at the look-ahead distance becomes the goal point. If the look-ahead distance is outside of the distance where the lanes are detected, then the goal point will by default be the closest point to the vehicle where both lanes are available. This will ensure that the vehicle will not cut any corners, however vehicle jerk may be increased temporarily.

Chapter 5 System Implementation and Testing

After all of the vehicle’s systems were designed they were implemented, then testing began in simulation, followed by integration and finally vehicle testing. In this section the implementation of the designed system will be reviewed. Then, the unit testing, integration testing, followed by vehicle validation will be presented.

5.1 Implementation

The implementation of the vehicle's control system was slightly modified from the original design due to delay of the control boards that were ordered by the MVSL. The board would have to be designed, manufactured and tested by the supplier, then tested once more once they arrived at the lab. The first prototypes of the boards were tested in lab and implemented on the urban vehicle, allowing the corner modules to be independently tested using the joystick. However, these models would require some revisions before more boards could be ordered for the final product, and this delay lead to a change of hardware.

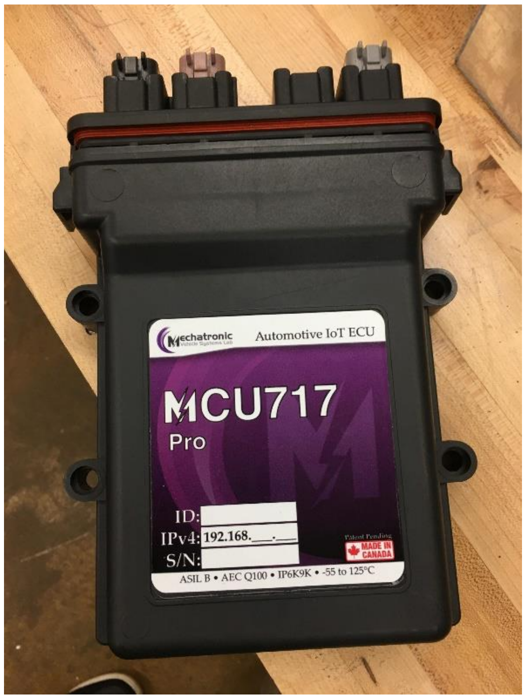


Figure 5.1 MVSL IoT ECU Prototype Board

For the testing phase, two arm based processing boards that were already owned by the lab were utilized to handle all of the analog and digital signals required by the vehicle's hardware, while a laptop executed the control system in Simulink in real-time. One board is strictly dedicated to the hydraulic power pack, while the other receives the front steering feedback, control the front steer actuators, send throttle voltages mapping to the requests and communicate to the laptop via CAN bus. This board receives torque requests, as well as front steer requests from the laptop and sends back the front steering position. The laptop executes the control software in Simulink, receiving the feedback from the rear steering, motor controllers and front steering directly all via the CAN bus. It then sends out front steer commands, rear steer commands and motor torque commands to be carried out.

From a software perspective, this meant that the fault detection logic that was originally designed to be included as a part of the corner modules is now a part of the supervisory control logic. This way the supervisory control logic could be executed on the laptop, doing fault detection, mitigation, and path following. The control system refreshes on the laptop every 10ms meeting the requirements originally set out for the vehicle. This simplified the hardware, without sacrificing functionality or requiring any major software changes as the same functions would be utilized.

For the lane detection, the camera used on the vehicle was a Basler acA1920-nm40um USB 3.0 with the Sony IMX249 sensor CMOS sensor delivering up to 41 frames per second at 2.9MP [47]. Basler provides a ROS driver package called pylon that allows the autonomous laptop to easily interface with the camera. The lens used with the camera was a Kowa LM8HC which allows for ample vision of the lanes, and even multiple lanes depending on the angle that the camera is mounted. This camera and lens combination was chosen because it was already available in the lab. They were also previously used to test feature-based lane detection algorithms, guaranteeing that they would provided an adequate field of view.

To process the SCNN lane detection model a secondary laptop would be used to reduce the computational burden across two platforms. This decision was made primarily because the control system running on the primary laptop is safety critical, and it would have to run under a timing constraint. These two computers communicate via ROS using an Ethernet cable, to ensure that both systems meet the timing constraints.

The control software for the vehicle was implemented using Matlab/Simulink, this would allow us to run it on the laptop and also use code generation to test on the MVSL's ECUs. Advantages of using Simulink also include rapid prototyping, the ability to quickly test through simulations and access to many support packages. For example, Simulink's CAN communication, as well as

ROS support packages would be important to establish the communication channels required. The lane detection software was completed in Python using Tensorflow, as an SCNN implementation was already available in Python and Python also has a large variety of packages to choose from.

5.2 Testing

The final testing of the vehicle was a thorough process that included testing the vehicle's hardware, the electrical connections for the control system and finally the system's software. In this section each portion of the testing will be discussed, starting from unit testing of the software, to integration testing where the electrical and mechanical components were also tested. The general approach was to test as soon as possible to determine where extra attention would be required, as well as take a bottom up approach to build up upon what had already been tested. Simulations were also heavily used to validate the software functions, and tune the controls software as much as possible before testing on the vehicle. This section will first cover the unit testing and integration testing, focusing on the electrical and control software components. Then, the testing for the lane following portion will be presented followed by full vehicle validation.

5.2.1 Unit Testing

The first step in the testing process would be unit testing of the control software. There are two main types of unit testing that were carried out, black box testing and white box testing. Black box testing was carried out to ensure that all of the requirements were met. Since the control software was implemented in Simulink it was very easy to carry out black box testing of individual software functions using constants or functions that were manually changed to simulate real world scenarios. The outputs were then monitored using a display or a scope to ensure the requirements outlined were met. Figure 4.2, provides an example of the setup when unit test the steering control, in this case to ensure the steering angles are within the expected range when a fault occurs. In a similar way, it was used to ensure that the motor controller would always provide the torque to the right motors and within the limits depending on the temperature to avoid overheating, as well as test that the remaining requirements were fulfilled by their respective function.

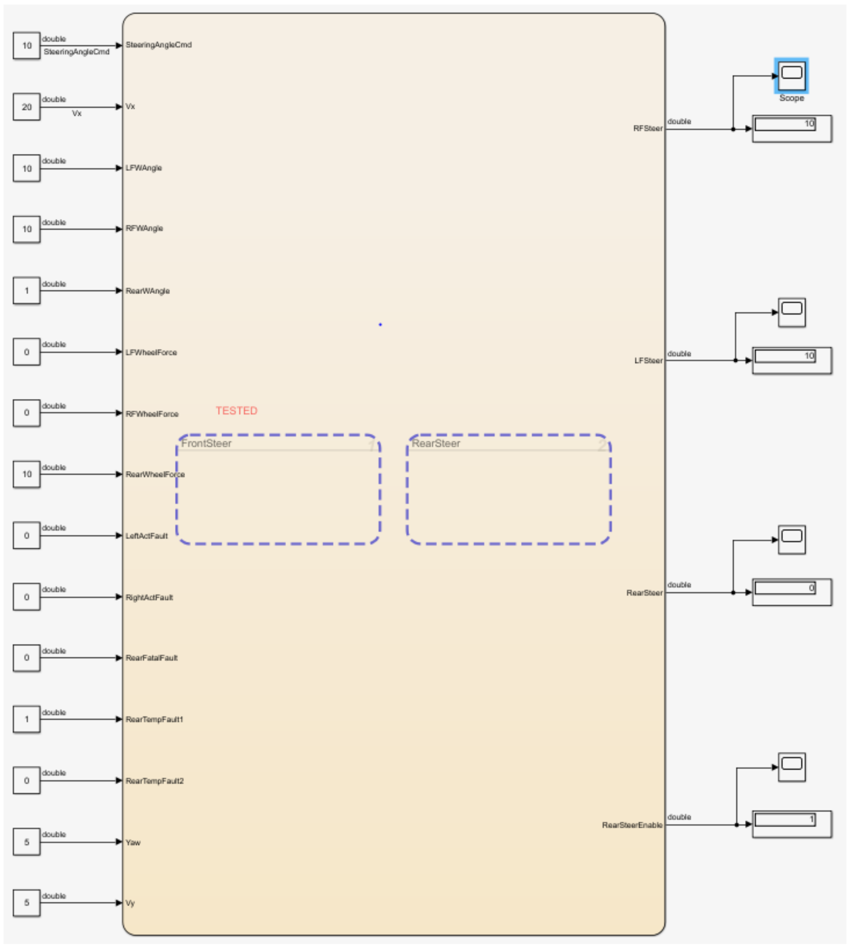


Figure 5.2 Unit Testing of Steering Controller

White box testing was carried out to ensure that the implementation fulfilled the detail design. It was carried out in a very similar way as black box testing, except the internals of each function were examined to ensure the appropriate variables were being used in each case. For example, whit box testing was also used to ensure that the vehicle would always be in the correct state and that the appropriate inputs were being taken into account. State charts provided an easy interface where the inputs can be changed without having to restart the simulations and the internal state transitions can be clearly labeled as seen in figure 5.3.

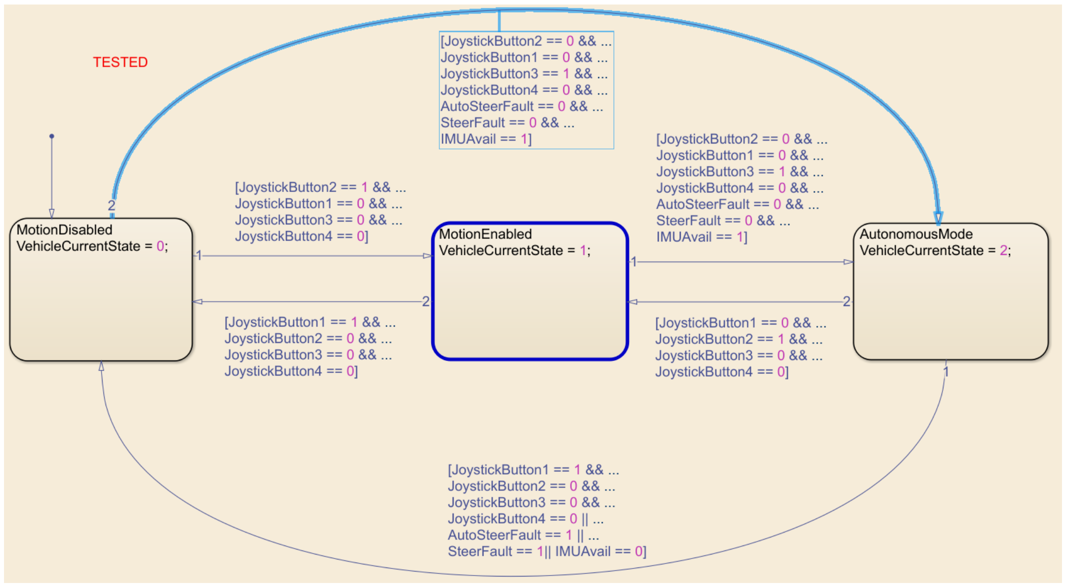

TESTED
MotionDisabled
VehicleCurrentState = 0;
MotionEnabled
VehicleCurrentState = 1;
AutonomousMode
VehicleCurrentState = 2;
[JoystickButton2 == 0 && ...
JoystickButton1 == 0 && ...
JoystickButton3 == 1 && ...
JoystickButton4 == 0 && ...
AutoSteerFault == 0 && ...
SteerFault == 0 && ...
IMUAvail == 1]
[JoystickButton2 == 1 && ...
JoystickButton1 == 0 && ...
JoystickButton3 == 0 && ...
JoystickButton4 == 0]
[JoystickButton2 == 0 && ...
JoystickButton1 == 0 && ...
JoystickButton3 == 1 && ...
JoystickButton4 == 0 && ...
AutoSteerFault == 0 && ...
SteerFault == 0 && ...
IMUAvail == 1]
[JoystickButton1 == 1 && ...
JoystickButton2 == 0 && ...
JoystickButton3 == 0 && ...
JoystickButton4 == 0]
[JoystickButton1 == 0 && ...
JoystickButton2 == 1 && ...
JoystickButton3 == 0 && ...
JoystickButton4 == 0]
[JoystickButton1 == 1 && ...
JoystickButton2 == 0 && ...
JoystickButton3 == 0 && ...
JoystickButton4 == 0 || ...
AutoSteerFault == 1 || ...
SteerFault == 1|| IMUAvail == 0]

Figure 5.3 Testing the Vehicle State Function

5.2.2 Integration Testing

Once all the software unit testing was complete, the system had to be integrated together. Before the full control software could be deployed to the vehicle, the vehicle's hardware and electrical connections for the control system were tested. The vehicle's wiring and connectors that were required for the control system were all done in-house, the tools and most of the materials required were already in lab. Each of these were tested by using simplified control logic, where each electrical function could be narrowed down to ensure the communication to the hardware and the sensors was reliable. This helped to isolate issues and a multi-meter was also used to validate the connections by checking the voltages, as well as resistances. To begin, each corner module, the hydraulic pack and batteries were each tested separately using the prototype MVSL IoT ECU.

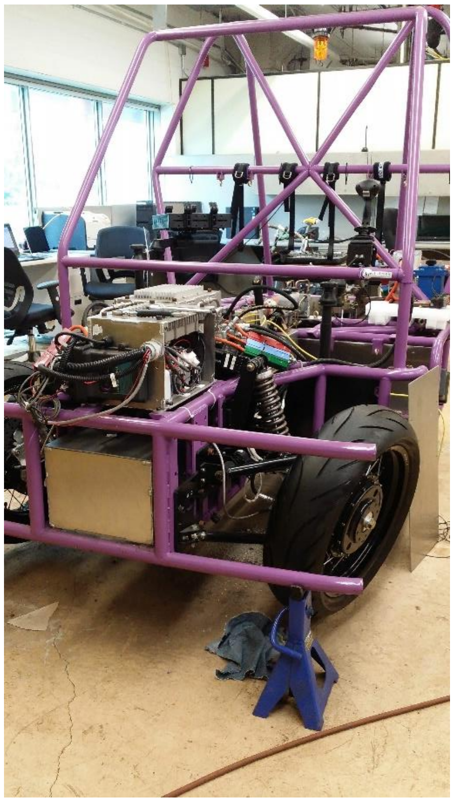


Figure 5.4 Urban Vehicle Hardware Testing

Once all of the vehicle's hardware and electrical components were tested, the integration of the final control software would begin. First, the two lower level controllers were tested for functionality. The hydraulic pack controller was tested to ensure it would read the hydraulic pressure and it would turn on the hydraulic pumps when the pressure was too low. The other controller was tested to ensure it could read the string potentiometers for the front steering feedback. Then it was tested to ensure that it would send the correct output voltages to the motor controllers for the throttle and the pulse/direction signals for the stepper motors to control the front steering. This controller also contains a proportional controller with a deadband to adjust the angle and velocity of the stepper motors based on the actuator length error. A Vector CAN CASEXL with CANoe was used to send out actuator length and torque commands to the lower level controller to ensure that the vehicle responded as expected.

Once the lower level controllers were functioning as expected, the integration of the supervisory control on the laptop was initiated. The CAN bus for the supervisory control unit was wired and tested to complete the electrical portion of the control system. Then, to integrate the software modules that had completed unit testing, a new Simulink file was started, and one set of functions would be added one by one. The six different sets of functions that were added are as follows; CAN In, Fault Detection, Estimation, Vehicle State, Vehicle Control, and CAN Out. Every time a set of functions would be added, it would be tested on the vehicle's hardware to ensure all inputs, outputs and vehicle functions were operating as expected. Then, before the CAN Out set of functions was integrated, the full control system response was monitored based on inputs while insuring that the vehicle remained in a safe state.

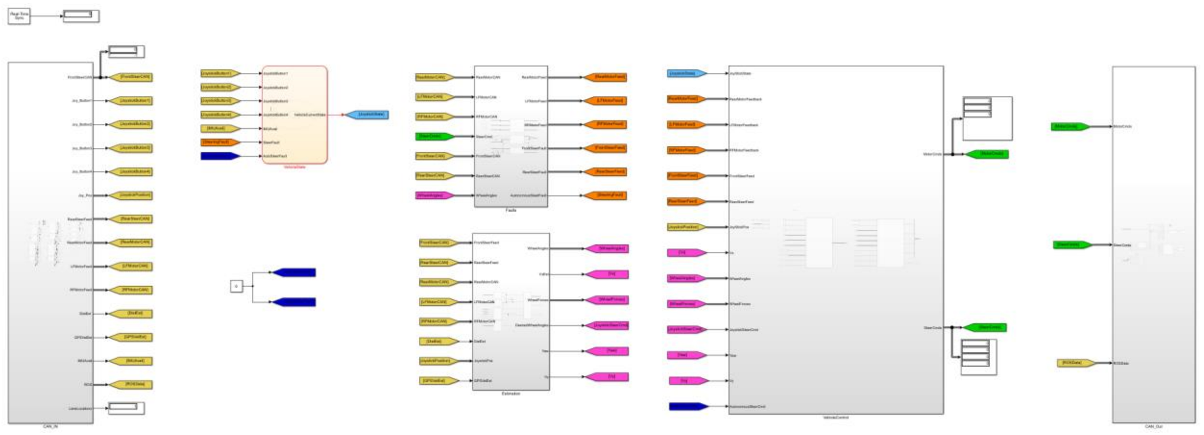


Figure 5.5 Full Control System Integration

Once all of the control software was integrated and all of the components were functional, the tuning for the steering would have to be completed before the vehicle could be driven. First, the mapping for the steering actuator length to angle would have to be improved, as an estimate was originally used. This mapping is vital to the vehicle because the model requires a wheel angle. Through measurements and experimental testing, the tuning for the steering was complete. Finally, the fault detection mechanisms for the steering was tuned. The front steering does not report faults, therefore the detection of faults had to be robust and thoroughly tested.

5.2.3 Fault Tolerance Testing

Once the vehicle was fully integrated and tuning completed, the fault tolerance testing began to ensure that the control system responded as expected. Fault tolerance testing began with fault detection. Most of the fault detections could be easily simulated and verified because either the actuator would report a fault or no feedback would be received, however this was not the case with the front steering actuators that are basic hydraulic cylinders. With the front wheels constantly adjusting due to the play in the mechanical setup, hydraulic pressure and hydraulic leakage, it was important to leave a time, as well as error buffer to ensure that the control system wouldn't have false fault detections. These had already been tested in simulation, however these tests involved starting all the vehicle systems and manually inputting faults to see how the vehicle would respond.

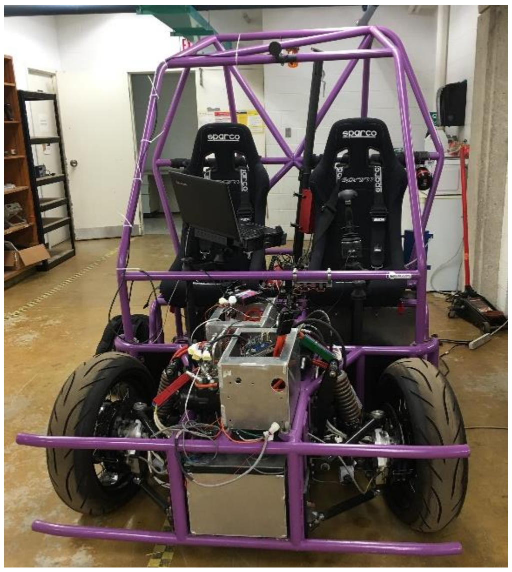


Figure 5.6 Urban Vehicle Steering Actuator Fault Testing

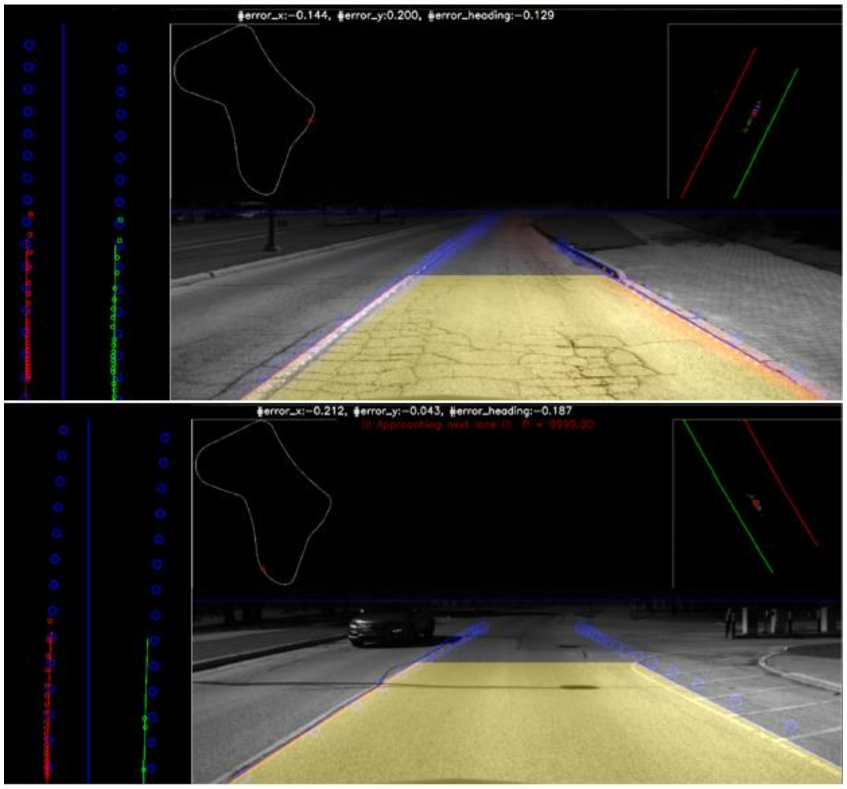
The front steering failure was detected through two ways, steering angle error and change in steering angle error. If the vehicle's wheel angle error was not decreasing or within a two degree threshold of the desired wheel angle for two seconds the front wheels would enter a fault state. This large threshold was implemented to avoid false detections because the steering is not very precise and the constant wheel angle shifting. When fault detection was working as expected the focus shifted to fault mitigation. In the fault state, the functioning wheel turns at the same angle, but opposite direction of the one that is in a fault state to balance out the lateral forces, an example of this is seen in figure 5.6 above. The front steering will stay in this state until the functioning wheel responds to the desired commands.

The rear steering failure would be tested in the validation phase as the vehicle will have to be on the road and driving for the vehicle model to calculate the corrected front steering angle. During the testing process the right front motor controller encountered a hardware fault, making it unavailable. The vehicle switched to rear wheel drive to compensate and the remaining tests were carried out in that configuration.

5.2.4 Lane Following Testing

At this point, all of the safety critical control had been tested, integrated and operating as expected, therefore the focus shifted to testing the lane following. There would be two parts to this, first to test the final version of the perception software to ensure that the lanes would be detected throughout the drive cycle. Secondly, to test the chosen path following algorithm that would allow the vehicle to stay within the lanes.

The perception software testing was straight forward because data had been previously recorded into a ROS bag when one of the lab's vehicles was driven around Ring Road with an accurate OXTS RT2000 IMU/GPS unit installed. This meant that the ROS bag would just be played, while the algorithm executed and was subscribed to the ROS topics, to test it using real-world data. To further test the robustness of the algorithm and help to gauge whether a GPS unit with less accuracy could also be utilized, Gaussian noise was added to the GPS signal. The algorithm performed very well and was able to detect the lanes through the entire drive cycle, a sample of this is seen in figure 5.7 below. The model relying only on SCNN was tested in the same way, mostly to ensure that the interface between the model and the control system were as expected, the SCNN performance was already evaluated as explained in section 4.6.



```
#error_x:-0.144, #error_y:0.200, #error_heading:-0.129
```

```
#error_x:-0.212, #error_y:-0.043, #error_heading:-0.187
!!! Approaching next lane !!! R = 9999.00
```

Figure 5.7 Lane Detection Model Testing

The blue lines mark the ground truth labelling of the lanes on Ring Road, while the red and green lines mark the predicted lanes. The algorithm breaks down the lines into points as shown on the left side of the image, five of these points are sent to the supervisory vehicle controller to calculate the vehicle's goal point. As seen on the image above the model can now detect the curb, even when it is at the same altitude as the road. The algorithm can also handle other tricky scenarios, for example when only one lane is visible or when crossing a cross walk, as seen in figure 5.8 below.

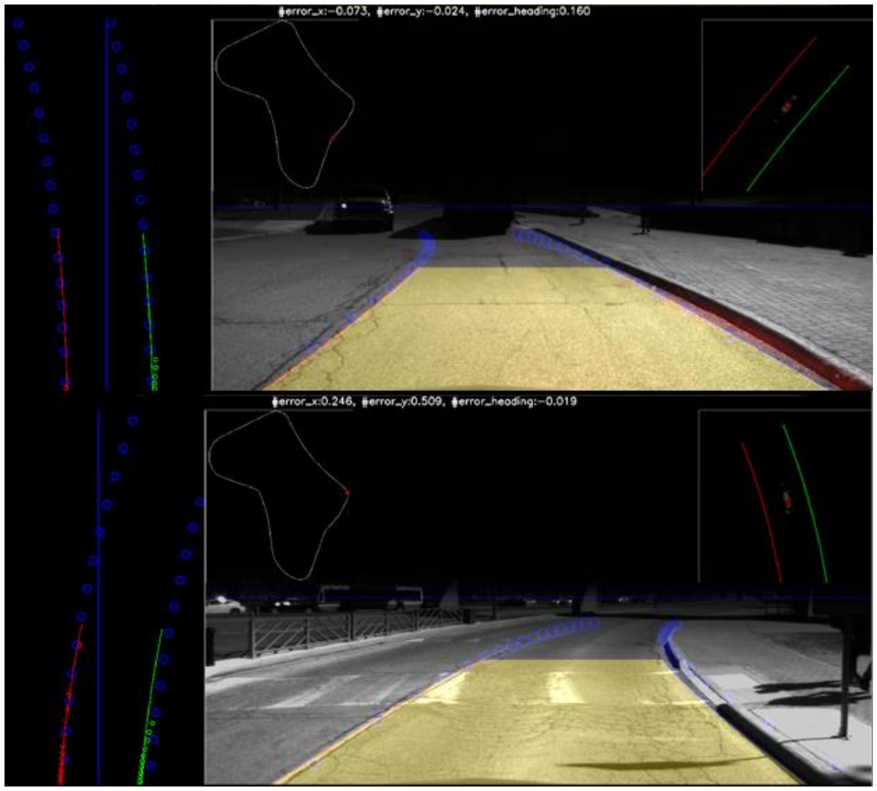

error_x:-0.073, error_y:-0.024, error_heading:0.160
error_x:0.246, error_y:0.509, error_heading:-0.019

Figure 5.8 Lane Detection Model Testing Continued

The second part of the testing would focus on the vehicle's path tracking algorithm to ensure that it receives the lane data, processes it, and sends the appropriate vehicle steering commands. To evaluate this, the perception models were once again executed on a laptop using the ROS bag from Ring Road, while another laptop subscribed to the ROS node outputting the lane data to collect it. The lane data was collected and stored in Matlab as time series to allow the data to be replayed again at the same rate. This data was then passed into the function to get the goal point, and the output of the pure pursuit controller was examined. A vehicle dynamics model was added at the end to visualize the effects that the steering would have on the vehicle at about 20km/h as seen in figure 5.9 below.

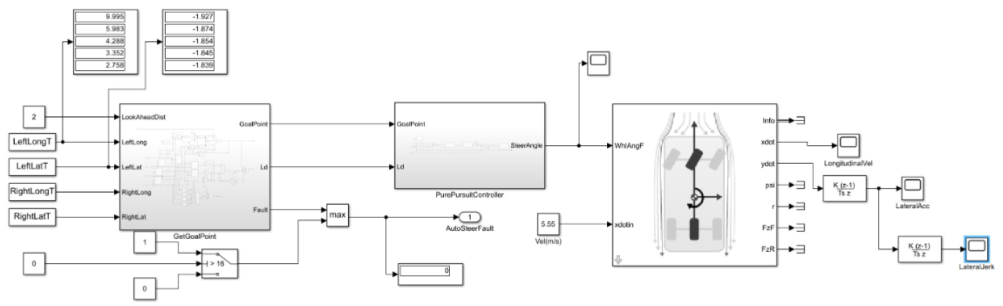


Figure 5.9 Path Tracking Simulation

The results for the testing were very promising, although it was difficult to gauge how well the physical vehicle would perform because during the drive cycle the vehicle was driven very close to the middle of the lane. Upon completion, the path tracking function was able to successfully process the lane data and the steering response was proportional to the error, as well as in the correct direction. This can be seen in figure 5.10 below, where the path error is negative meaning that the vehicle is closer to the left lane, and the vehicle steers to the right, which is positive in this case. Using the vehicle model, it was shown that as expected, the vehicle jerk would increase as the look-ahead distance would decrease. However, if the look-ahead distance was too large the steering corrections would be minuscule, and unlikely to get the vehicle back on its intended path. This would be tuned further on the vehicle to ensure that the vehicle to ensure that the vehicle stays on its intended path, while keeping the jerk values low to ensure passenger comfort.

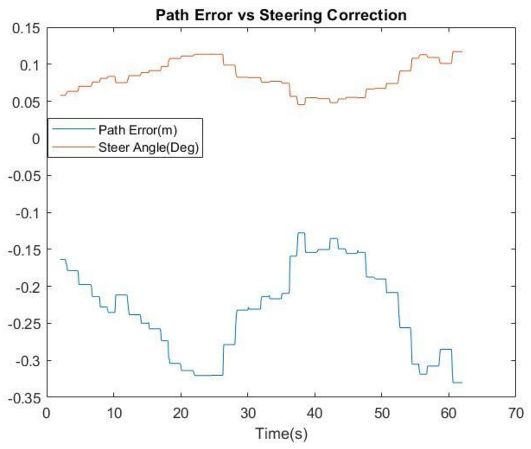


Figure 5.10 Path Following Simulation

5.2.5 Validation

Validation is the process of ensuring that the right product is being developed through the use of dynamic tests to certify that the product has met all of the requirements originally outlined. In this case, the vehicle was taken for on-road testing to validate its fault tolerance, as well as lane following performance. In terms of fault tolerance, the motor/motor controller failure, front steer actuator failure, and rear steer actuator failure scenarios were tested as they were the main areas of interest. To begin, the vehicle was driven on campus without any actuator or motor failures being created to ensure that the vehicle was functioning as expected. Unfortunately, as mentioned earlier one of the front motor controllers had experienced a current measurement failure, but all other actuators performed as expected. The control system redirected power only to the rear wheel to account for this front motor controller failure. Therefore, the validation testing continued with the steering failures.

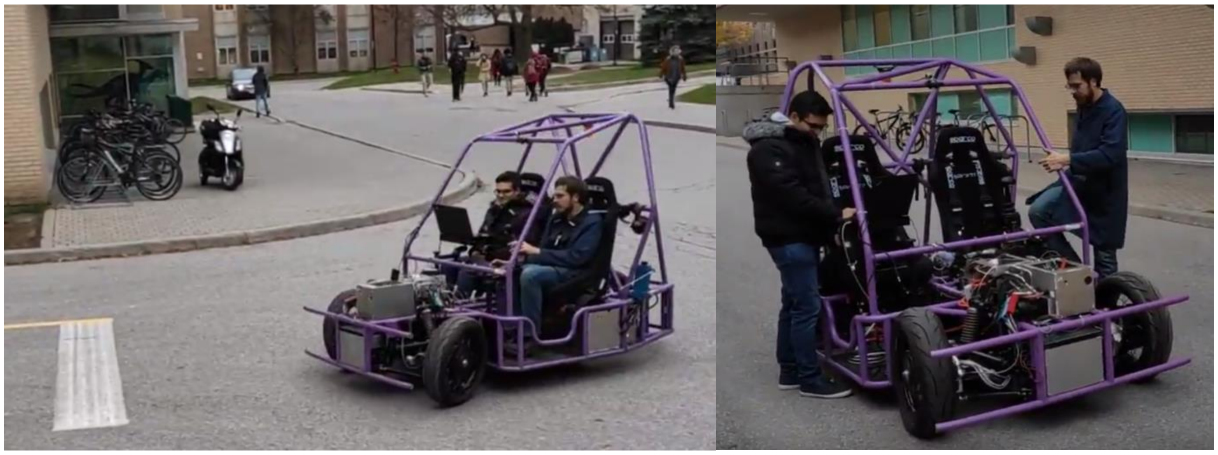


Figure 5.11 Vehicle Road Testing

Once the vehicle was driving, the steering actuator failure detection and mitigation would be tested. To do this, a constant actuator position command was sent to the right steering actuator in the front. The vehicle was driven to see if the fault would be detected, and then that the corresponding fault mitigation action would be observed. The data for the front right steering actuator fault is seen in 5.12 below. In the data it is evident that the fault detection is working as expected, and the left wheel follows the position of the right wheel. However, there is some overshoot in the left wheel position due to the feedback delay, PID and the constant readjustment of the right wheel. The rear steering follows the commands sent out by the joystick, although the wheel angle is less than the joystick command because as the vehicle's speed increases the range of motion of the rear steering decreases proportionally. This helped to stabilize the vehicle and allowed the driver to maintain control, but the vehicle was slowed down significantly due to the front wheel providing extra resistance.

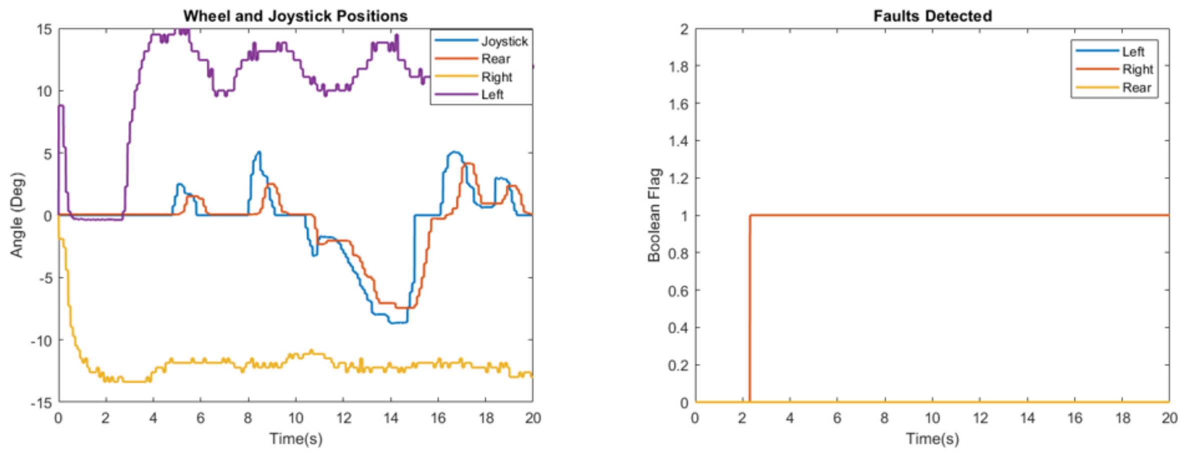


Figure 5.12 Front Actuator Failure Data

For the rear steering failure, the vehicle model would be used along with the measured wheel positions to calculate the angle of the front wheels required to make up for this force. However, when observing the model throughout the drive cycle the model's lateral velocity, yaw rate, and wheel forces would continually increase to large values. This was observed because the vehicle is constantly readjusting the front wheel angles, even if a constant wheel angle is requested as seen in figure 5.13 shown below. The model required integrating to find the lateral velocity and yaw rate, which is inputted for the model's next time step, leading the error to continually add up. The model had to be reset to zero and recalculated whenever the rear steer error occurred. However, eventually the model would still reach an unstable state and the vehicle's front steering would be forced to a full left or right position to compensate. This effect can be seen in figure 5.14, where the model stabilized temporarily before forcing the front wheels to do a full right steer.

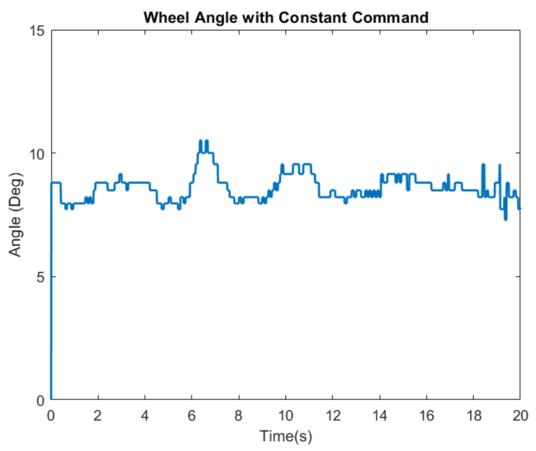


Figure 5.13 Front Steer Accuracy with Constant Angle Request

The response rate for the rear steering fault was unacceptable as the driver would lose full control of the steering at that point. As mentioned earlier, the cause of the error was determined to be the constant readjustment of the front wheels. On the other hand, it was evident that the rear wheel was much more stable and was able to maintain the steering angle requested. In a failure scenario, the rear wheel would then stay at a constant angle and the wheel's position would be accurate due to the precise feedback provided by the actuator. Therefore to provide better results, instead of using the front wheel angle measurements in the vehicle model, the front steer commands were used. This provided the failure in the rear to be acknowledged by the vehicle model but also have a much more stable front wheel angle to avoid error in the model. The output of the model for the same failure at about 8km/h can be seen in figure 5.15 below, where the model readjusts the wheels to a stable state of just under 3 degrees after the failure is detected after two seconds. This allowed the vehicle to drive in a straight line with minimal driver input and still leaves the driver with about 12 degrees of freedom in the front wheels to steer in the desired direction.

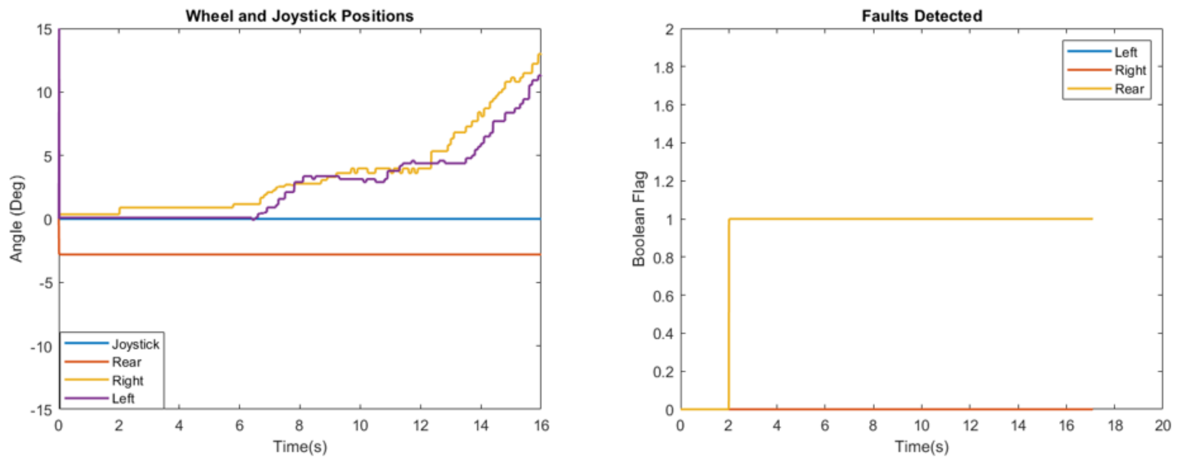


Figure 5.14 Rear Steer Failure Data

Overall, the vehicle was able to reliably detect and compensate for the faults that were introduced into the system. The on-road testing was able to confirm that the control strategy, as well as the model was effective for the urban vehicle. The final phase of validation testing would involve the lane following testing.

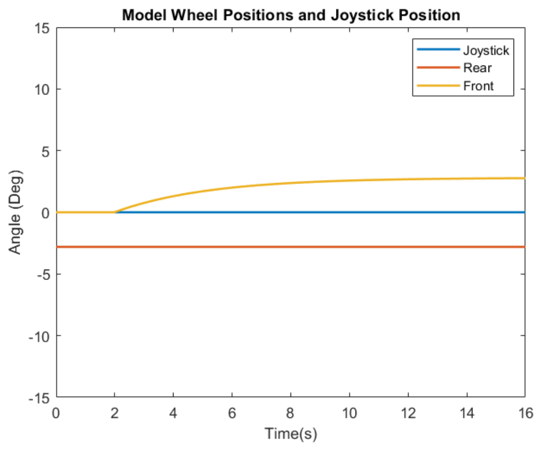


Figure 5.15 Revised Model Output

The lane following testing was carried out on the lab's electric Chevrolet Equinox. This vehicle benefitted from having much more range than the urban vehicle, allowing for more testing per charge. The Equinox was also instrumental in helping us test in different weather environments. To do this, the path tracking portion of the control system on the urban vehicle was copied to the Equinox's control system. The urban vehicle's hardware was also transferred over to the Equinox including the perception laptop, control laptop and the camera to accurately capture how it would function on the urban vehicle. The vehicle setup can be seen in figure 5.16 below. For these tests the lab technician would control the velocity of the vehicle, while the control system would provide the steering requests.

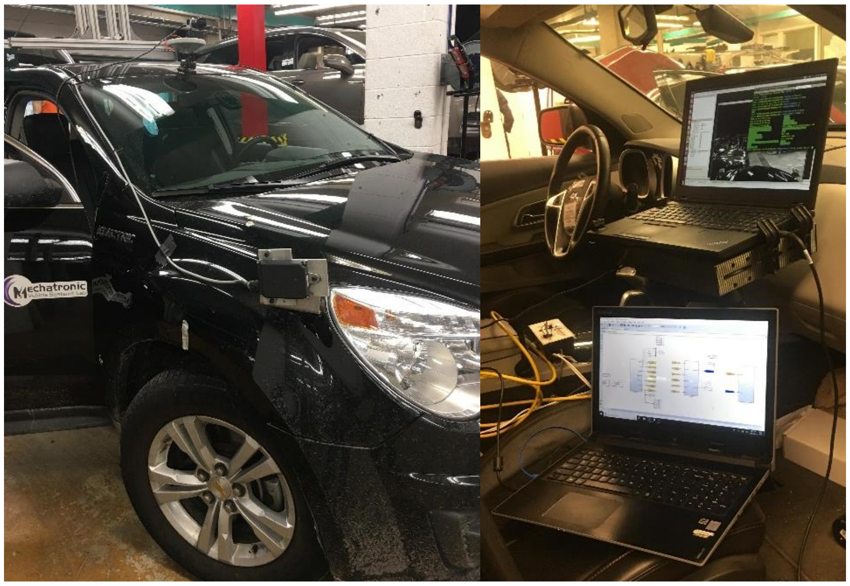
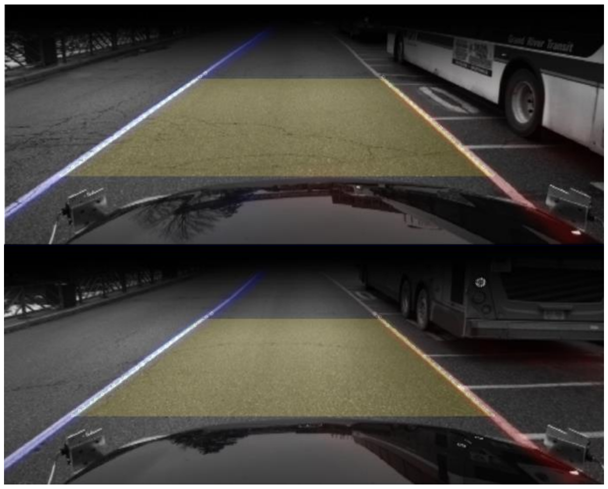


Figure 5.16 Lane Following Vehicle Setup

The lane following testing began with the lab's hybrid lane detection algorithm. This algorithm performed best when tuned for the data that had previously been collected, however it was having difficulty prediction the lanes when tested on the vehicle. The algorithm was dependent on the vehicle's position as it was in the dataset, but this was not the case as the GPS's accuracy changes based on weather conditions and other disturbances. Due to this error the algorithm had difficulty identifying which of the two lanes is the vehicle corresponding lane, and predicting the curves of the road.

Then, the SCNN-based lane detection model was tested on the vehicle. This method relies solely on vision and it proved to be very effective when two lanes were visible. The model could clearly detect both lanes, allowing the path to be created between the lanes and the vehicle steered effectively to stay within them. On the other hand, when only one lane was visible and the curb was on the other side the model had difficulty detecting the curb, leading the control to have difficulty defining the correct path. Eventually, the vehicle would slowly veer outside of the lane.

This effect could be due to the fact that the curbs were partially snow covered. In the scenario shown in figure 5.17 below, the vehicle was able to drive from one intersection to the next, through the bus station, and past various buses without requiring driver intervention. Although the perception model was running at about 4.52 frames per second, the performance was best at speeds under 25km/h.



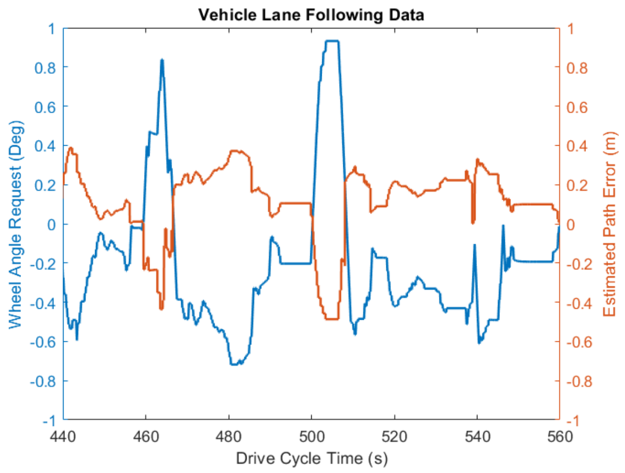


Figure 5.17 Lane Following with SCNN

To improve on these results, the algorithm that determines the vehicle's path was modified. Instead of searching for the path between both lanes, the left lane location would be taken to predict the location of the curb on the right. Using the left lane location and the estimated location of the curb on the right, the control calculates the goal points. This worked well to allow the vehicle to navigate through most of the lanes on Ring Road at low speeds. However, when the curvature of the road became high the perception algorithm would lose sight of the left lane.

Chapter 6
Conclusion and Future Work

As a part of this project, the vehicle's control system was designed, implemented and tested. Different hardware and software variations were tested on the vehicle including: two MVSL IoT ECUs, two machine learning based lane detection models, and two algorithms to determine the vehicle's path for lane following were tested, demonstrating the modularity of the design. After the completion of the project, the urban vehicle can serve as a platform to validate autonomous vehicle models or controls algorithms due to the system's modular design. The control system's electrical connections, software, and path tracking algorithm were completed as a part of the project along with the interfaces to test lane detection models. These systems were tested through simulation environments, integration testing on the vehicle and on-road testing. However, there are many improvements that can be made to the urban vehicle, which is not uncommon for a prototype vehicle of this nature.

6.1 Future Work

This section will highlight key areas of improvement that are recommended for the urban vehicle. These recommendations are to improve the vehicle's performance, and the modular design means they can be changed without having to make large changes to the overall system.

6.1.1 Control System

The control system has performed reliably as implemented, however there are some possible improvements that can be made. It is recommended that the control architecture as design included using the MVSL's IoT ECUs is still implanted on the vehicle as they would provide many advantages. The overall vehicle would be more modular, there are also many functions that can be handled by these ECUs instead of having physical switches, also simplifying the vehicle wiring and improving the controllability. For example, the precharge circuit, motor enable, forward, and reverse signals could all be controlled through the corner module ECUs.

The MVSL's IoT ECUs would also enable the capability of over-the-air (OTA) updates, as well as remote data collection. OTA updates would allow lab members to log into the vehicle and update the control system or autonomous system whenever required, or before a drive cycle. The data collection would also be very useful as the variables that need to be collected can just be saved onto the ECUs data, and the uploaded to the MVSL' data drive once the vehicle returns from the drive cycle. For this to occur, there must also be a firewall, as well as various safety measures that must be taken to ensure that the vehicle can only be updated while parked, and only by MVSL members.

To further improve the fault tolerant component of the control system, a more accurate vehicle model with a model predictive controller can be implemented. This would extend the fault tolerant control to be effective in low tire friction scenarios of when aggressive maneuvers are performed. Another control strategy that may be adopted in conjunction or to replace the current strategy is to examine the vehicle's yaw to detect faults. If the drivers intended yaw rate or path is not what is expected, this can be detected and corrected. Finally, the pure pursuit controller can also be further tuned by adding an actively changing look-ahead distance. If the map or trajectory of the path is known ahead of time this can also help to predict the optimal look-ahead distance for each portion of the drive cycle.

6.1.2 Vehicle Powertrain

The vehicle's powertrain has performed well during the testing phases, but there are a few key improvements that would make it easier to work with. The most important of these improvements being the brake system. It is currently manually operated through the lever providing hydraulic pressure, which was sufficient for our tests as the driver always had control of the vehicle's speed. However, to increase the autonomy of the vehicle, the vehicle must be able to monitor and adjust its own speed. The required solenoid and pumps are already installed on the vehicle, they just require a PCB to interface with them, as well as a DC-DC converter with an output of 3.3V at 8A to power the solenoids. To achieve this a PCB would have to be created to interface with the solenoid and pumps that are currently installed as a part of the vehicle's brake system. Then the Body Controller ECUs could interface with the PCB to control the braking. This would also allow for regenerative braking and mechanical braking to be combined, extending the vehicle's range.

The next change that is recommended is to upgrade the high voltage pack for the vehicle, as the current one only allows for about 15 minutes of driving around campus at low speeds. A redesigned battery pack could utilize the same connections, but by increasing the capacity of the pack or increasing the voltage of the system to reduce losses, the vehicle's driving range would be improved. The stepper motors that control the front steering alone will drain the pack in under two hours when left on, which leads into the next recommendation.

The final recommended change is to replace the hydraulic front steering actuators with electric actuators. At the beginning of the project the hydraulic rear steering was replaced for electric rear steering using a linear actuator. Overall, this has been a successful improvement as it requires less wires, reports faults, it is quieter, removes the concern of hydraulic oil leaks, it is easier to control and more precise. Furthermore, if the hydraulics for the steering were removed including the hydraulic pack, then the vehicle would be much lighter, more energy efficient, and less susceptible to failures. The two biggest considerations when making the switch are the input voltage requirement, stroke length, speed and force provided by the actuator. The rear steer occasionally experiences an overload fault, signifying that the linear actuator does not have enough force to execute the desired command due to the weight of the vehicle and the resistance provided by the tire rotation.

6.1.3 Autonomous System

The autonomous system is one that is expected to change and improve, as this is a popular research area in industry, as well as academia. To start, the lane line detection must be improved to detect lanes with greater curvature. The SCNN seems to pick up on these lanes, however the function built on top of the network does not recognize them. It makes the false assumption that the lane should be ahead and relatively straight.

After an improved lane detection model is implemented, the autonomous system should be improved by adding a machine learning model for object detection. This would allow the system to determine a path, as well as velocity based on the obstacles and lanes that have been identified. This system could be further improved through the addition of radars to accurately determine the distance to each object that has been detected. Research is currently being conducted in the MVSL for similar systems, and could be easily implemented on the vehicle.

References

[1] N. Sadek, “Urban electric vehicles: a contemporary business case,” *European Transport Research Review*, vol. 4, no. 1, pp. 27–37, Jan. 2012.

[2] “Overview of electricity production and use in Europe,” *European Environment Agency*, 15-Dec-2016. [Online]. Available: https://www.eea.europa.eu/data-and-maps/indicators/overview-of-the-electricity-production-1/assessment. [Accessed: 05-May-2019].

[3] M. W. Blanke, C. J. Frei, F. undefined Kraus, R. undefined Patton, and M. undefined Staroswiecki, “What is Fault-Tolerant Control?,” *International Federation of Automatic Control*, vol. 33, no. 11, Jun. 2000.

[4] R. T. Meyer, S. C. Johnson, R. A. Decarlo, S. Pekarek, and S. D. Sudhoff, “Hybrid Electric Vehicle Fault Tolerant Control,” *Journal of Dynamic Systems, Measurement, and Control*, vol. 140, no. 2, Sep. 2017.

[5] M. Benosman, “Passive Fault Tolerant Control,” *Robust Control, Theory and Applications*, pp. 283–308, Apr. 2011.

[6] R. Maryniuk, “Development of a Modular Urban Electric Vehicle,” *UWSpace - Waterloo's institutional repository*, 14-Dec-2017. [Online]. Available: http://hdl.handle.net/10012/12746. [Accessed: 15-Sep-2018].

[7] M. Blanke, M. Staroswiecki, and N. Wu, “Concepts and methods in fault-tolerant control,” *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, Jun. 2001.

[8] D. Diallo, M. Benbouzid, and A. Makouf, “A Fault-Tolerant Control Architecture for Induction Motor Drives in Automotive Applications,” *IEEE Transactions on Vehicular Technology*, vol. 53, no. 6, pp. 1847–1855, Dec. 2004.

[9] A. Ukaew, "Model Based System Design for Electric Vehicle Conversion," *New Trends in Electrical Vehicle Powertrains*, Nov. 2018.

[10] R.-A. Marchant, "Model-Based Design: Design with Simulation in Simulink," *Mathworks*, 2016. [Online]. Available: https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/company/events/conferences/matlab-conference-australia/2016/proceedings/design-with-simulation-in-simulink.pdf. [Accessed: 03-Jun-2019].

[11] T. Kelemenová, M. Kelemen, Ľ. Miková, V. Maxim, E. Prada, T. Lipták, and F. Menda, "Model Based Design and HIL Simulations," *American Journal of Mechanical Engineering*, vol. 1, no. 7, pp. 276–281, Nov. 2013.

[12] T. Egel, M. Burke, M. Carone, and W. Jin, "Applying Model-Based Design to Commercial Vehicle Electronics Systems," *SAE International Journal of Commercial Vehicles*, vol. 1, no. 1, pp. 392–396, Jul. 2008.

[13] N. He and H.-W. Huang, "Use of model-based design to teach embedded systems programming," *2017 IEEE International Conference on Electro Information Technology (EIT)*, pp. 91–94, Mar. 2017.

[14] A. Lebel, "Development of a Control System for a Series-Parallel Plug-In Hybrid Electric Vehicle," *MacSphere*, 2017. [Online]. Available: http://hdl.handle.net/11375/21240. [Accessed: 05-Jul-2019].

[15] R. S. Yadav, "Improvement in the V-Mode," *International Journal of Scientific & Engineering Research*, vol. 3, no. 2, pp. 1–8, Feb. 2012.

[16] S. Balaji and M. S. Murugaiyan, "WATEERFALL vs V-MODEL vs AGILE: A COMPARATIVE STUDY ON SDLC," *International Journal of Information Technology and Business Management*, vol. 2, no. 1, pp. 26–30, Jun. 2012.

[17] M. Broy, S. Kirstan, H. Krcmar, and B. Schätz, “What is the Benefit of a Model-Based Design of Embedded Software Systems in the Car Industry?,” *Emerging Technologies for the Evolution and Maintenance of Software Models*, 2012.

[18] J. Abraham, “Verification and Validation Spanning Models to Code,” *AIAA Modeling and Simulation Technologies Conference*, Jan. 2015.

[19] “TDH30 Pressure Transducer - Low Cost: Transducers Direct,” *Transducers Direct*. [Online]. Available: http://www.transducersdirect.com/products/pressure-transducers/tdh30-pressure-transducer/. [Accessed: 22-Nov-2018].

[20] “MHM602/MHM603,” *EnerTrac*. [Online]. Available: https://www.enertrac.net/product.php. [Accessed: 12-Oct-2018].

[21] “Kelly KLS8080I/IPS Motor Controller User’s Manual,” *Kelly Controller* . [Online]. Available: https://e-vehicle.eu/KLS8080I IPS Opto-isolated Sinusoidal BLDC V1.10.pdf. [Accessed: 12-Oct-2018].

[22] “Sinusoidal Wave Controller KLS Broadcast CAN Protocol 1.1,” *Kelly Controller*. [Online]. Available: https://kellycontroller.com/wp-content/uploads/kls-8080i-ips/Sinusoidal-Wave-Controller-KLS-D-8080I-8080IPS-Broadcast-CAN-Protocol.pdf. [Accessed: 12-Oct-2018].

[23] “SP1 Compact String Pot • Voltage Divider ,” *TE Connectivity* . [Online]. Available: https://www.te.com/commerce/DocumentDelivery/DDEController?Action=showdoc&DocId=Data Sheet•SP1•A•pdf•English•ENG_DS_SP1_A.pdf•CAT-CAPS0067. [Accessed: 15-Oct-2018].

[24] T. G. Kirner and A. M. Davis, “Nonfunctional Requirements of Real-Time Systems,” *Advances in Computers*, vol. 42, pp. 1–37, 1996.

[25] “History of Car Safety,” *Crash Test Vehicle Safety and Accident Prevention*. [Online]. Available: https://www.crashtest.org/history-car-safety/. [Accessed: 04-Mar-2019].

[26] G. Saldivia, "Stuck In Traffic? You're Not Alone. New Data Show American Commute Times Are Longer," *NPR*, 20-Sep-2018. [Online]. Available: https://www.npr.org/2018/09/20/650061560/stuck-in-traffic-youre-not-alone-new-data-show-american-commute-times-are-longer. [Accessed: 04-Mar-2019].

[27] S. Berger, "These are the states with the longest and shortest commutes — how does yours stack up?," *CNBC*, 23-Feb-2018. [Online]. Available: https://www.cnbc.com/2018/02/22/study-states-with-the-longest-and-shortest-commutes.html. [Accessed: 04-Mar-2019].

[28] A. J. Hawkins, "Fewer people died in car crashes in 2017, but the outlook is still grim," *The Verge*, 03-Oct-2018. [Online]. Available: https://www.theverge.com/2018/10/3/17933536/traffic-death-crash-statistics-nhtsa-us-2017. [Accessed: 04-Mar-2019].

[29] "CrashStats," *National Highway Traffic Safety Administration*. [Online]. Available: https://crashstats.nhtsa.dot.gov/#/. [Accessed: 05-Mar-2019].

[30] "Lane Departure Accidents: Car Accident Injury: Huntsville, AL," *Martinson & Beason, P.C.* [Online]. Available: https://www.martinsonandbeason.com/practice-areas/car- accidents-huntsville-alabama/types-of-car-accidents/lane- departure-accidents/. [Accessed: 05-Mar-2019].

[31] M. Aly, "Real time detection of lane markers in urban streets," *2008 IEEE Intelligent Vehicles Symposium*, Jun. 2008.

[32] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," *Computer Vision – ECCV 2016 Lecture Notes in Computer Science*, pp. 21–37, Dec. 2016.

[33] Y. Xing, C. Lv, L. Chen, H. Wang, H. Wang, D. Cao, E. Velenis, and F.-Y. Wang, "Advances in Vision-Based Lane Detection: Algorithms, Integration, Assessment, and Perspectives on ACP-Based Parallel Vision," *IEEE/CAA Journal of Automatica Sinica*, vol. 5, no. 3, pp. 645–661, Apr. 2018.

[34] J. Hui, “MAP (mean Average Preci- sion) for Object Detection,” *Medium*, 07-Mar-2018. [Online]. Available: https://medium.com/@jonathan hui/map-mean-average-precision- for-object-detection- 45c121a31173. [Accessed: 05-Mar-2019].

[35] J. Li, H. Cheng, H. Guo, and S. Qiu, “Survey on Artificial Intelligence for Vehicles,” *Automotive Innovation*, vol. 1, no. 1, pp. 2–14, Jan. 2018.

[36] J. M. Snider, “Automatic Steering Methods for Autonomous Automobile Path Tracking,” *Carnegie Mellon University*, Feb. 2009. [Online]. Available: https://www.ri.cmu.edu/publications/automatic-steering-methods-for-autonomous-automobile-path-tracking/. [Accessed: 17-Sep-2019].

[37] B. Paden, M. Cap, S. Z. Yong, D. Yershov, and E. Frazzoli, “A Survey of Motion Planning and Control Techniques for Self-Driving Urban Vehicles,” *IEEE Transactions on Intelligent Vehicles*, vol. 1, no. 1, pp. 33–55, Apr. 2016.

[38] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, “Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing,” *2007 American Control Conference*, Jul. 2007.

[39] R. C. Coulter, “Implementation of the Pure Pursuit Path Tracking Algorithm,” *Carnegie Mellon University*, Jan. 1992.

[40] H. Tahir, “Development of an Autonomous Vehicle Platform,” *UWSpace - Waterloo's institutional repository*, Aug. 2019. [Online]. Available: http://hdl.handle.net/10012/14960. [Accessed: 03-Oct-2019].

[41] H. Lamba, “Understanding Semantic Segmentation with UNET,” *Medium*, 17-Feb-2019. [Online]. Available: https://towardsdatascience.com/understanding-semanticsegmentation-with-unet-6be4f42d4b47. [Accessed: 06-Oct-2019].

[42] P. LeBeau, “Pedestrian Deaths Hit 28-Year High, and Big Vehicles and Smartphones are to Blame.,” *CNBC*, 28-Feb-2019. [Online]. Available:

https://www.cnbc.com/2019/02/28/pedestrian-deaths-hita-28-year-high-and-big-vehicles-and-smartphones-are-toblame.html. [Accessed: 31-Mar-2019].

[43] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, "An Empirical Evaluation of Deep Learning on Highway Driving," Apr. 2015.

[44] H. Zhao, X. Qi, X. Shen, J. Shi, and J. Jia, "ICNet for Real-Time Semantic Segmentation on High-Resolution Images," *Computer Vision – ECCV 2018 Lecture Notes in Computer Science*, pp. 418–434, Aug. 2018.

[45] G. Neuhold, T. Ollmann, S. R. Bulo, and P. Kontschieder, "The Mapillary Vistas Dataset for Semantic Understanding of Street Scenes," *2017 IEEE International Conference on Computer Vision (ICCV)*, Jul. 2017.

[46] T. J. Veldhuizen, "Yaw rate feedback by active rear wheel steering," Aug. 2007. [Online]. Available: https://research.tue.nl/en/studentTheses/yaw-rate-feedback-by-active-rear-wheel-steering. [Accessed: 10-Sep-2019]

[47] "Basler Ace User's Manual for USB 3.0 Camera," Jul. 2016.

[48] "NVIDIA Introduces DRIVE Constellation Simulation System to Safely Drive Autonomous Vehicles Billions of Miles in Virtual Reality," *NVIDIA Newsroom Newsroom*, 27-Mar-2018. [Online]. Available: https://nvidianews.nvidia.com/news/nvidia-introduces-drive-constellation-simulation-system-to-safely-drive-autonomous-vehicles-billions-of-miles-in-virtual-reality. [Accessed: 17-Oct-2019].

Appendix A Electronic Control Unit Connection Mappings

This section provides a mapping of the connections of each electrical control unit designed to be on the urban vehicle. Each connection will be found in the chart along with a small explanation of its function, the corresponding pin on the ECU, the corresponding variable within the processor and the expected voltage, where zero volts corresponds to a low-side drive signal. With the information in the chart, the MVSL's IoT ECU's can be easily configured for this use case.

Connection Name	**Function**	**ECU In**	**Processor Variable**	**Drive Strength**
Rear Corner Module ECU:				
CAN0_Low(Rx)	CAN0 – Vehicle	J1-45	PTB0	-
CAN0_High(Tx)	CAN0 – Vehicle	J1-48	PTB1	-

CAN1_Low(Rx)	CAN – Corner Module	J1-40	PTA12	-
CAN1_High(Tx)	CAN – Corner Module	J1-41	PTA13	-
PreCharge	Begin motor precharge	J1-36	PTC13	0V
HighVoltage	After precharge, leave high voltage on	J1-35	PTD1	0V
MtrCtrlPower	Turn on Motor Controller	J1-42	PTD15	12V
TorqCmd	0-5V signal for Torque Request	J1-29	PTC14	5V
BrakeCmd	0-5V signal for Regen Brake Request	J1-25	PTB10	5V
FwdSW	Motor set in forward direction	J1-37	PTD8	12V
RevSW	Motor set in reverse direction	J1-38	PTE15	12V
BrakeSW	Motor set for regenerative braking	J1-39	PTC2	12V
ThrottleSW	Throttle enabled	J1-34	PTB2	12V
Car_GND	ECU GND	J1-8	-	-
12V In (Battery)	ECU Power	J1-9	-	-
Front Corner Module ECU:				
CAN0_Low(Rx)	CAN0 – Vehicle	J1-45	PTB0	-
CAN0_High(Tx)	CAN0 – Vehicle	J1-48	PTB1	-
CAN1_Low(Rx)	CAN – Corner Module	J1-40	PTA12	-
CAN1_High(Tx)	CAN – Corner Module	J1-41	PTA13	-
PreCharge	Begin motor precharge	J1-36	PTC13	0V
HighVoltage	After precharge, leave high voltage on	J1-35	PTD1	0V
MtrCtrlPower	Turn on Motor Controller	J1-42	PTD15	12V
TorqCmd	0-5V signal for Torque Request	J1-29	PTC14	5V
BrakeCmd	0-5V signal for Regen Brake Request	J1-25	PTB10	5V
FwdSW	Motor set in forward direction	J1-37	PTD8	12V
RevSW	Motor set in reverse direction	J1-38	PTE15	12V
BrakeSW	Motor set for regenerative braking	J1-39	PTC2	12V
ThrottleSW	Throttle enabled	J1-34	PTB2	12V
Car_GND	ECU GND	J1-8	-	-
12V In (Battery)	ECU Power	J1-9	-	-
SteerPos1	Feedback from String Pot One (Top - Steer)	J1-1	PTB13	5V
SteerPos2	Feedback from String Pot Two (Bottom - Camber)	J1-2	PTA1	5V
FeedbackPower (2)	Potentiometer Power	-	-	5V
FeedbackGND(2)	Potentiometer GND	-	-	GND
SM1_PUL-	Stepper Motor One Pulse Signal	J1-22	PTC0	5V
SM2_PUL-	Stepper Motor Two Pulse Signal	J1-24	PTB8	5V
SM1_DIR-	Stepper Motor One Direction	J1-26	PTE4	5V

SM2_DIR-	Stepper Motor Two Direction	J1-27	PTB4	5V
SM1&2DIR+&PUL+	PUL-High and DIR-High Voltage for Stepper Motor One and Two	J1-21	-	5V
SM1_GND &SM2_GND	Stepper Motor Grounds	J1-12	-	GND
Body Controller ECU:				
Car_GND	ECU GND	J1-8	-	-
12V In (Battery)	ECU Power	J1-9	-	-
CAN0_Low(Rx)	CAN – Vehicle	J1-45	PTB0	-
CAN0_High(Tx)	CAN – Vehicle	J1-48	PTB1	-
Brake Pressure 1	Front Left Brake Line Pressure	J1-1	PTB13	5V
Brake Pressure 2	Front Right Brake Line Pressure	J1-2	PTA1	10V
Hydraulic Pressure	Hydraulic Pressure in Power Pack	J1-3	PTE6	10V
12V PP Reading	Measure Voltage of Power Pack Battery	J1-4	PTA16	14V
5V Power	Power to pressure sensors + IMU	J1-21	-	-
Sense GND	GND to IMU, pressure + hydraulic sensors	J1-12	-	-
Pump Relay1	Turn on Hydraulic Pump One	J1-37	PTD9	GND
Pump Relay2	Turn on Hydraulic Pump Two	J1-38	PTE16	GND
Battery Relay	Turn on hydraulic pack battery DC-DC for charging	J1-42	PTD16	GND
Supervisory Controller ECU:				
Car_GND	ECU GND	J1-8	-	-
12V In (Battery)	ECU Power	J1-9	-	-
CAN0_Low(Rx)	CAN0 – Vehicle	J1-45	PTB0	-
CAN0_High(Tx)	CAN0 – Vehicle	J1-48	PTB1	-