THE PENNSYLVANIA STATE UNIVERSITY
SCHREYER HONORS COLLEGE

DEPARTMENT OF MECHANICAL ENGINEERING

Path-Planning and Optimization of a High-speed 1/5-Scale Off-Road Autonomous Vehicle

ANDRES ENRIQUE ESPARRAGOZA
SPRING 2024

A thesis
submitted in partial fulfillment
of the requirements
for a baccalaureate degree
in Mechanical Engineering
with honors in Mechanical Engineering

Reviewed and approved* by the following:

Dr. Sean Brennan
Professor of Mechanical Engineering
Thesis Supervisor

Dr. Daniel Cortes
Associate Professor of Mechanical Engineering
Honors Adviser

* Electronic approvals are on file.

ABSTRACT

This thesis investigates path-planning for a high-speed off-road autonomous vehicle. A 1/5th scale Remote Control (RC) vehicle was modified with sensors to be able to track vehicle position, trajectory, velocity, and other parameters necessary while performing traversals on a specifically designed off-road course that challenges the vehicles maneuverability and velocity.

Advanced global position system (GPS) and encoder algorithms are implemented into the vehicles onboard microcontrollers to determine real time data necessary to dictate the necessary steering and throttle inputs. The sensor algorithms allow for the vehicle to measure its position and velocity, thereby enabling the foundation of path-following.

Using algorithms to control the two receiving functions, steering and throttle, the vehicle must traverse the course autonomously with the aim of increasing speed per traversal until reaching an optimal steady-state velocity profile. Using iterative learning control (ILC) and similar control strategies, the vehicle learns from prior traversals to find limiting levels of velocity to be able to repeatedly produce traversals faster with each lap.

The foundation for a 1/5th scale high-speed off-road vehicle path following simulation is created on MATLAB prior to vehicle implementation. The simulation successfully models constant speed along a path for both large-scale and 1/5th scale vehicles. The simulation is prepared for the implementation of ILC to determine the optimal steady-state velocity profile. The goal is for the simulation to achieve lap navigation faster than if a human were to control the vehicle via its remote-control unit.

This research discussion concludes by outlining the next steps for algorithm development and field testing. The overall goal is to contribute to the development of autonomous off-road

vehicles for use in a range of applications, including agriculture, mining, and search and rescue. The findings of this thesis seek to advance core algorithms and deployment technologies for autonomous systems in challenging off-road environments.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Firstly, I would like to express my gratitude to Dr. Brennan, my research supervisor, for guiding me throughout my undergraduate research experience. Dr. Brennan's expertise, encouragement, and support have been vital in shaping and refining my knowledge of intelligent vehicles and engineering. Dr. Brennan, your dedication to your research and students has been inspirational and motivating for me.

I'd also like to extend my appreciation to the Intelligent Vehicles and Systems Lab, whose resources and knowledge have played a role in this research. The lab's commitment to research intelligent vehicles has provided me with a captivating environment to explore, learn, and innovate in the field of off-road autonomous vehicles.

I am also thankful for my team of undergraduate engineers – Gabriel Gayoso, Isaiah Adu, Justin Kerr, Marius Tanase, Mustapha Salau, and Micah Delattre – who have been a source of inspiration and ideas. All your insights, discussions, and passion for intelligent vehicle systems have contributed significantly to this thesis's development.

Finally, I'd like to acknowledge my family and friends for their support, understanding, and encouragement throughout this academic journey. Your encouragement, love, and joy have helped me maintain a healthy balance of academics and pleasure.

This thesis is a culmination of the collective efforts, guidance, and support from everyone above. Thank you all for your support and for being a part of this major milestone in my academic career.

The work presented herein is supported by the National Science Foundation under grant numbers CNS-1932509, CNS-1931927, CNS-1932138 “CPS: Medium: Collaborative Research: Automated Discovery of Data Validity for Safety-Critical Feedback Control in a Population of Connected Vehicles”. Any opinions, findings, and conclusions or recommendations expressed in ix this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation. I gratefully acknowledge this support.

Chapter 1

Introduction

Motivation

The exploration and advancement of off-road autonomous vehicles is critical due to the condition of off-road driving. Off-road driving is often considered difficult and multifaceted due to a variety of factors such as variable traction, obstacles, uneven terrain, and many other conditions which limit the performance of a vehicle. Conversely, off-road driving's lack of definition allows for the user to have a greater margin for decisions when path-planning.

Several industries such as mining, agriculture, construction, forestry, and others heavily rely on off-road vehicles. Thus, research in autonomous off-road vehicles can significantly benefit these industries in the development of their vehicles and machinery they use to accomplish their respective tasks.

In mining operations, where rugged and often hazardous terrains are faced, off-road autonomous vehicles can enhance safety, efficiency, and productivity. Autonomous vehicles can also minimize human exposure to suboptimal conditions. Off-road autonomous vehicles in agriculture can revolutionize tasks such as planting, harvesting, and soil management. Generally, research in the field of off-road autonomous vehicles is vital for developing robust and adaptable systems capable of navigating challenging terrains and performing tasks autonomously.

The ultimate challenge of off-road vehicle control is understanding how to reach the highest performance, which usually is limited by the vehicle's maneuverability limits under these

unpredictable conditions. This maneuverability challenge can be measured via the goal of achieving the minimum time for a vehicle to traverse an off-road path successfully and safely. To reach this robust limit, many algorithms and systems must be explored and researched to ensure the optimal performance of the vehicle under the off-road conditions.

Goals of This Thesis

The goal of this thesis is to achieve the successful implementation of autonomous off-road path-optimization using a 1/5th scale RC vehicle. The objective of this research is threefold: first, to develop the vehicle system architecture; second, to conduct data collection through various testing methods; and third, to develop and improve algorithms aimed at enhancing off-road vehicle performance. The primary goal is to enhance autonomous off-road vehicle capabilities, surpassing human drivers in controlling high-speed off-road vehicles through algorithm development.

To develop these objectives, this thesis explains the substantial hardware improvements made to the off-road vehicle platform. Essential modifications have been implemented to ensure the vehicle can withstand the rigors of high-speed, high-impact off-road driving. Safety measures, including an emergency stop system, have been integrated to mitigate the risk of high-speed collisions and safeguard the vehicle and its surroundings. The electronics and connections for data collection have been adapted to facilitate repeatability of the data collection process.

The thesis also explores the development of data collection methods. Initially, the RC vehicle is manually controlled to traverse an intended path developed on the Penn State Test Track. While the vehicle traverses the course, an onboard Teensy 4.1 microcontroller collects

encoder and GPS data are collected storing it onto a micro-SD card. This dataset – encompassing parameters such as wheel angular velocity, linear car velocity, and relative position – is extracted for post-processing in MATLAB. These parameters serve as inputs in creating a course-following algorithm.

The final phase of the thesis focuses on the development of algorithms designed for off-road vehicle control. The long-term outcome of this work is an autonomous vehicle capable of swiftly navigating designed off-road courses, with the goal of driving quicker and more consistently than a human could. This approach aims to push the boundaries of off-road autonomous vehicle capabilities, contributing to advancements in autonomous systems for off-road environments.

Organization of This Thesis

The organization of the remainder of this thesis is as follows:

Chapter 2 reviews previous research and literature in the development of autonomous vehicles. This includes development in sensor hardware, system architecture, sensor algorithms, path-following, path-planning, and iterative learning control, the control theory which is further explored to determine the velocity of the vehicle.

Chapter 3 is an overview of the $1/5^{th}$ scale RC vehicle used in this thesis. Hardware modifications and other details regarding the hardware and vehicle itself are explored in depth. This chapter also explores the electronic circuitry and subsequent modifications made during this work to ensure polarization and protection for repeatable data collection cycles. Lastly, this chapter discusses the software dictating the vehicle dynamics and data collecting.

Chapter 4 discusses the steering and throttle limiting controls that are considered.

Chapter 5 discusses the data collection process regarding and the results of several field data collection studies accomplished. More specifically, the results of defining the official track trajectory and determining the vehicle's primary states: velocity, acceleration, yaw, and power.

Chapter 6 explores the primary path-following algorithms and development given different inputs. This chapter discusses the use of iterative learning control within simulation as well as the results of preliminary simulation models of path following.

Finally, Chapter 7 concludes the research accomplished during this thesis, next steps in operation, and future work to be done on the vehicle for the next cohort of undergraduate researchers.

Chapter 2

Literature Review

Autonomous Vehicles

Autonomous vehicles, also known as AVs, are quickly becoming commonplace in road navigation. Companies such as Tesla, Google, GM, and many others are researching ways to implement autonomous capabilities in their current fleet of vehicles. The primary motivation for this is due to the 1.24 million people worldwide who die yearly due to vehicle-related accidents [1]. The guiding view in the scientific community is that autonomous vehicles and intelligent transportation systems may reduce driver behavior-related crashes, thus minimizing fatalities.

Autonomous vehicles are simply robotic vehicles which have functionalities that often emulate a human operating a vehicle [1]. An autonomous vehicle measures its surroundings and, using these measurements, attempts to maneuver properly to its destination with minimal assistance or guidance from a human. An autonomous vehicle can accomplish this by using sensors, computers, data, and control systems to replace many human functions [1], [2].

Humans are incredibly good at making on-the-spot driving decisions, which is an inherent challenge of driving activities. However, humans are susceptible to making mistakes, which can lead to catastrophic outcomes when driving. Autonomous vehicles, on the other hand, are generally programmed with the goal of avoiding making the same mistakes as a human when driving. To accomplish this, autonomous vehicles are designed to have predictable algorithm performance. This predictability avoids error but can be challenged by on-the-spot changes while driving. One means of studying the interaction between a predictable environment and a

changing environment is to operate the vehicle such that it is starting with an incorrect model of its operational area, but the vehicle then must learn – as quickly as possible – how to change its performance. This is a key motivation to the methods and approach of this thesis.

Autonomous vehicles use intelligent transport system algorithms to optimize driving decisions to accomplish three applications, one of which being vehicle safety [2]. Figure 2.1 illustrates the three domains which include: safety – avoiding accidents, crashes, and fatalities, transportation efficiency- which is to optimize gas mileage and take optimal routes, and infotainment applications – enabling a human to engage in non-driving activities.

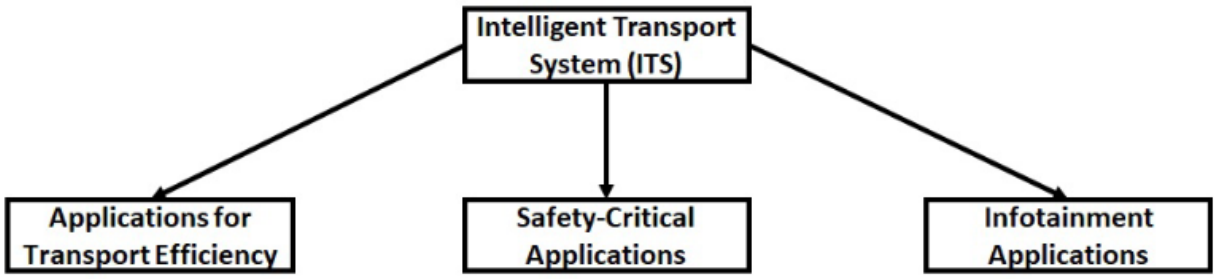


Figure 2.1 - Three Main Applications of Intelligent Transport Systems [2]

System Architecture of Autonomous Vehicles

The on-vehicle systems that allow autonomous vehicles to complete the tasks necessary to operate a vehicle are comprised of a complex array of sensors, processors, and actuators [3]. The sensors typically found surrounding an autonomous vehicle are either exteroceptive, which are for perception of surroundings, and proprioceptive sensors, which are for internal measurements [3]. Such sensor configurations are used to develop an observable environment in which the vehicle can engage in pathfinding by deciding its route based on sensor information.

Exteroceptive sensors are typically used in autonomous vehicles to be able to detect objects, identify the object's respective distance, and identify the object's current motion, if any. The exteroceptive sensors commonly found on autonomous vehicles include LiDAR, radar, cameras, and ultrasonic sensors. Most autonomous vehicles use a combination of these exteroceptive sensors in what is known as a sensor suite which work together to detect an object's location and motion. Figure 2.2, from Li J et al., demonstrates object detection accomplished from a sensor suite including a LiDAR, camera, and radar sensor on an autonomous vehicle [4].

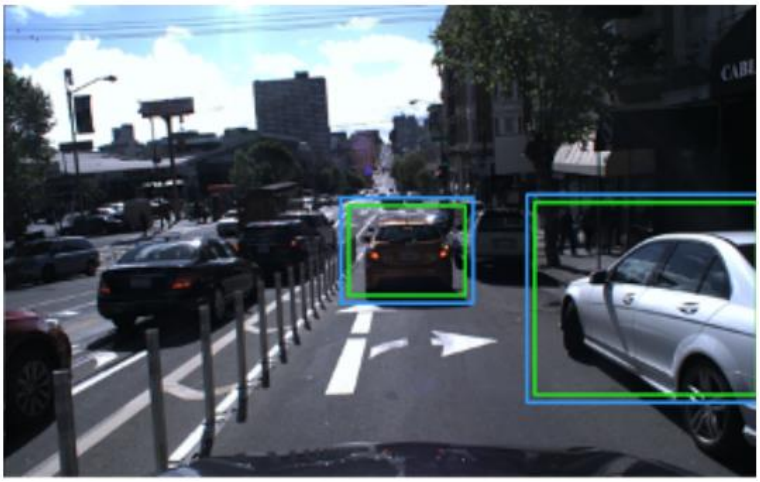


Figure 2.2 – Object Detection Accomplished with a Exteroceptive Sensor Suite [4]

LiDAR sensors, also known as light detection and ranging, use laser time-of-flight technology to measure distances and create 3D representations of their surroundings [2], [3]. LiDAR is commonly used on autonomous vehicles to detect other vehicles on the road, pedestrians, signs, and other objects to create a 3D world map of what to avoid.

Radar sensors use high-frequency radio waves to measure distances and velocities of objects [3]. Autonomous vehicles use radar to also detect objects and their motions relative to the autonomous vehicle itself.

Cameras intrinsically convert light emission into pixels arranged by the spherical angle of incoming light relative to the camera. The pixels may detect colors and, with time-of-flight cameras or stereo cameras, possibly specific depth perception [3], [5]. Cameras are often used in autonomous vehicles to read stop lights, road signs, lane lines, and local terrain.

Lastly, ultrasonic sensors use sound waves to measure the distances and velocities of relative objects [3]. Although cheap, ultrasonic sensors are uncommon in autonomous vehicles since they may be greatly affected by weather and atmospheric events, air temperature, and crosstalk from surrounding sensors. The typical AV's combination of LiDAR, radar, and camera perception is beneficial for the autonomous vehicle to register accurate details regarding what surrounds the vehicle.

Proprioceptive sensors typically measure internal vehicle-specific variables such as vehicle velocity, acceleration, forces, moments, and many other dynamic variables [2], [3], [5]. The proprioceptive sensors usually found on an autonomous vehicle include GPS, encoders, and Inertial Measurement Units (IMUs).

Encoders are a major hardware component in this thesis. Encoders, as imaged in Figure 2.3, use a rotational shaft to convert motion into electrical signals which can be read and processed by a controller [6].

Figure 2.3 – Rotary Wheel Encoder used in This Thesis [7]

Global positional systems (GPS) use satellite-based navigation for measuring local positions. IMU sensors can measure an object's acceleration (and net external forces), angular rate, and magnetic field. IMU's are typically built with three accelerometers, three gyroscopes, and three magnetometers in a compact printed circuit board to gather information along each of the axes in the typical XYZ cartesian coordinate system.

The sensors that form the infrastructure of an autonomous vehicle serve the purpose of collecting all the data necessary to capture what the vehicle is doing, where the vehicle is doing it, and what is near the vehicle [5]. The data package is responsible for the motion and decisions made by an autonomous vehicle. The amount of raw data collected by the sensors can be challenging to process. To simplify computation, land vehicle navigation relies on a complex process of data filtration to determine which data is worth using to help in imaging and navigation [5]. A key focus of previous research is to define chains of algorithms, called an "autonomy stack", that interrupt and filter corresponding data. The goal of this and similar research is to combine algorithms with sensors, processors, and actuator behaviors for consistent outcomes. A popular approach for sensor and data relationships is the multi-modal approach which uses multiple sensors to focus on different datasets. These datasets are filtered in parallel, and the results are merged at key processing steps [5]. Approaches, such as Kalman Filtering, are being applied in the research presented in this thesis.

Path-planning Fundamentals for Autonomous Vehicles

A key focus of current research related to on-road autonomous vehicles is to develop algorithms that achieve reliability rather than feasibility. Reliability in autonomous vehicle

driving is the ability of a vehicle to be able to drive as predictably as possible within the limits of its operational domain. Feasibility on the other hand seeks to demonstrate that the vehicle can operate at least once within a given situation or operational domain. Thus, the feasible operational domain must be much larger, usually, than the reliable operational domain, as feasibility tests seek to define the extent to where autonomous vehicles can perform with any success. Conversely, reliability tests seek to define the extent to which autonomous vehicles can perform without any failures.

On-road autonomous vehicles are designed to follow defined paths in predictable environments while making supervised decisions based on their surroundings [1], [2]. On-road autonomous vehicles currently follow well-paved roads with discernable markings and signs to indicate direction, rules of road, expectations on other vehicles, among others. In contrast, off-road autonomous vehicles do not always follow well-paved paths with discernable markings for directions [8].

Off-road and on-road autonomous vehicles thus can face very different challenges when it comes to considerations for programming and algorithms. Consequently, the algorithm types for on-road and off-road vehicles may be vastly different. For example, off-road driving must consider changes in terrain, steep slopes, weather, unpredictable obstacles, and other situations that an on-road driver rarely, if ever, must consider [8]. Thus, showing feasibility in off-road driving – for some situations – may improve reliability in on-road driving.

As another item of contrast between on-road and off-road AVs, the research behind off-road autonomous vehicles is focused on path-planning and following since the off-road setting often does not have a set path to follow. Thus, various algorithms for path-planning in unstructured environments have been created and researched in various settings. Such path-

planning algorithms depend on levels to control various aspects of the vehicle [9], [10]. The levels control all the parameters that determine vehicle movement and behavior including, but not limited to, position, throttle, steering, and trajectory. Humans instinctively decide what angle to take a turn, whereas the path-planning algorithm may attempt to accomplish the same level of success using a data-driven approach [5]. Figure 2.4, from K. Chu et al., illustrates the path candidate concepts that autonomous vehicles must decide on based on their anticipated path displacements [10].

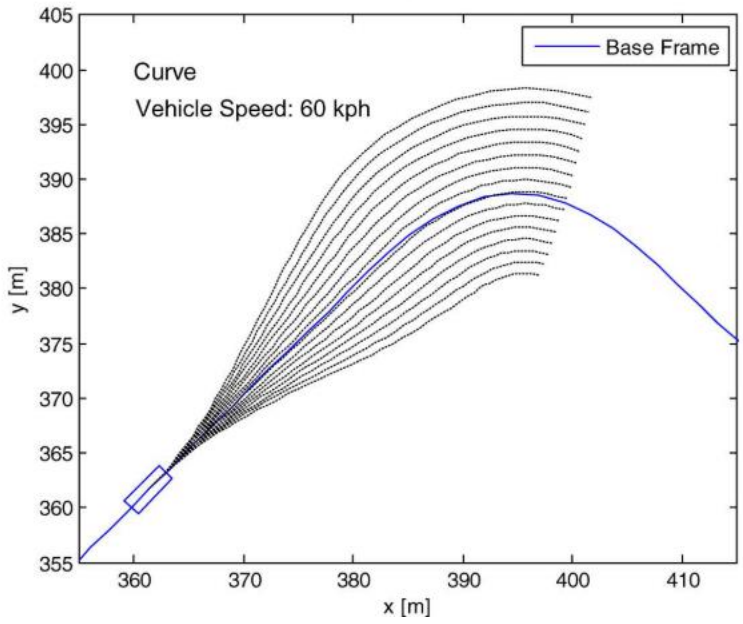


Figure 2.4 - Path Candidate Profile of an Autonomous Vehicle Taking a Turn [10]

Data dependency is the main difficulty with path-planning since the higher-level algorithms are usually designed to assume the world is static, and thus often lack a representation between changes in the world environment, certainty in the data, and decisions on when to re-plan paths. This interaction for human drivers is often intuitive. To address the lack of human intuition, prior research has found the use of cost functions to optimize how the vehicle should value different goals relative to each other [9], [10]. For example, when taking a turn, a vehicle

must consider speed, safety, barriers, and maneuverability. A vehicle could drive extremely fast to a corner, stop abruptly, and then turn sharply. The algorithm would detect such an event as a high-cost event since the vehicle had to stop and did not optimally turn. A low-cost event would be taking the turn from a wider angle and slower to maximize smoothness.

Lower-level algorithms include kinematics to calculate optimal trajectory and corresponding errors. Kinematic algorithms receive input from the sensors and can adjust the controls through feedback to follow the trajectory optimally [9], [10], [11].

The controls are simply the part of the autonomous vehicle which controls how the vehicle operates. The basic controls of a vehicle are throttle, braking, and steering. The controls of the autonomous vehicle of this research are manually controlled via a two-function remote control which consists of throttle/braking on one channel and steering on the other channel [12]. Autonomously, the vehicle is controlled via transponder control in which signals are sent directly to the transponder via a computer algorithm rather than a manual remote control. Path-planning algorithms are designed to communicate with the physical vehicle to activate the accelerator, brakes, steering wheel, and any other mechanical part of the vehicle.

Prior Research Contributions

Prior research has accomplished many direct tasks which have led to the growth of autonomous vehicle research both on- and off-road. This section explores various research topics and their major contributions to the autonomous vehicle path-planning realm.

A major component of autonomous vehicle development involves navigation using GPS technology. Leveraging GPS data allows the vehicle to pinpoint its location on the globe and

establish the foundation for its relative coordinate systems used for all dynamic algorithms. Research conducted at Stanford University created a GPS, Rate Gyro, Compass, and Odometer to create a visual navigation display [13] The use of the sensors and tools created the foundation for internal and external mapping systems for vehicular use. Dead reckoning sensors were also implemented into the system design to fill in gaps in GPS data [13]. The foundation laid by creating a low-cost and intelligent GPS has been vital to modern autonomous vehicle research. The research, however, did conclude that GPS is only accurate with the assistance of supplemental sensors and data which compensate for calibration errors.

Other sensors have been researched to facilitate the ability to create an autonomous vehicle. Research conducted in 1998 explored the use of millimeter wave radars in autonomous vehicle use for their ability to create a visual map [14]. The research created a system of 77 GHz millimeter wave radars and encoders to determine the active steer angle and velocity of the vehicle. Figure 2.5 illustrates the beacon observation from this research. This beacon setup facilitated the development of the algorithms necessary for the vehicle to understand what was in front of it, along with its respective distance and speed. Sensor setups like the one from this research are common in autonomous vehicles today to communicate to the controls of the vehicle in the case in which the steer angle or velocity must change.

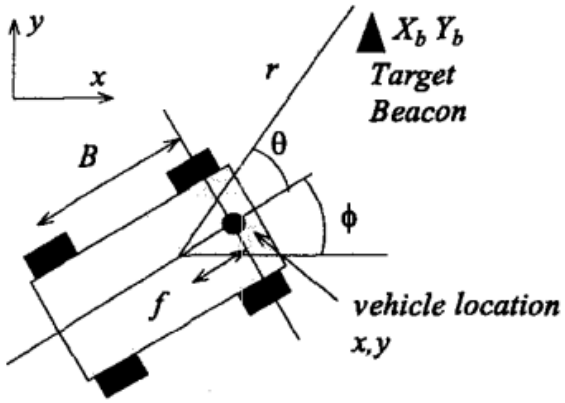


Figure 2.5 - Beacon Observation of a Vehicle using an Exteroceptive Sensor

Path-following is considered the precursor step to path-planning. Hongyan Guo et al. researched how to create path-following controls using model predictive control with considerations in road region and vehicle dynamics [11]. The sensor model used for this research includes cameras, radar, and real-time kinematic positions (RTK) which is a correction sensor for GPS systems [11], [13].

Prior autonomous vehicle research has laid out the foundations for object identification and intelligence. Research conducted at the Honda Research Institute has developed a way for a vehicle to identify its surroundings [4] using encoders, deep feature extractors, and cameras. Using the estimation of objects around the vehicle, the vehicle can determine what the object is and any tasks it might accomplish. This research has also laid the foundation of data sifting in object identification by filtering out unnecessary data and objects that might be considered insignificant or repetitive [4].

In the realm of object identification, research conducted by Mayuka et al. [15] was able to identify and decide what terrain type was optimal for the robot to operate. A skid steer robot equipped with cameras identified different terrain types and by previously gathering color and

texture patterns. Over several trials, the vehicle was able to successfully identify if the terrain ahead was gravel, wood, or grass and would decide which path was best for the robot to drive over [15]. Figure 2.6 demonstrates the vehicles design, sensor configuration, and path plan. The ability for the vehicle to learn from previous trials is notable and novel relative to prior work. The robot uses a form of artificial intelligence in its algorithm to be able to learn what type of terrain it is traversing as well as properties that would deem a terrain favorable or not.

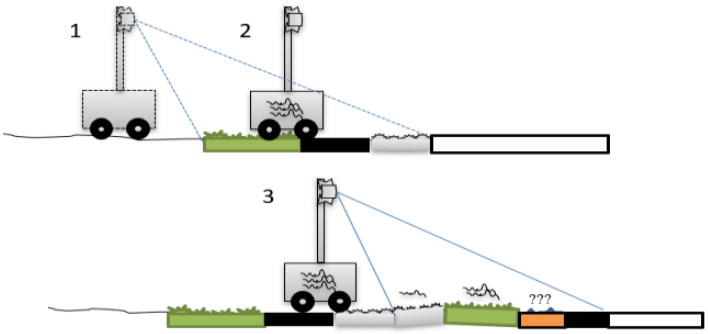


Figure 2.6 - Self-Supervised Learning Vehicle for Terrain Detection [15]

Research conducted by the Ford Motor Company has created 3D virtual driving environments for simulation purposes [16] This research has connected the virtual environment to MATLAB and Simulink to gather and sift data in a structured manner. The research used LiDAR sensors to detect total surroundings and create 3D maps that were controlled via the programs. 3D maps provide the vehicle with a perceptive component which helps in determining how close and far things are as well as what exactly are they doing relative to the test vehicle [16]. Simulation systems are becoming increasingly important for autonomous vehicle research since they must be able to create an eye and feel for the vehicle with as much precision as possible. Simulations allow for inexpensive and easy means of testing items without the need to be on track. The downside is that simulations lack the reality of off-road vehicular driving such

as constant changes in terrain friction, weather, and other variables which affect the driving experience.

Such real-world extreme off-road autonomous vehicle testing has been conducted but using small-scale replica vehicles. Using full-scale vehicles for autonomous off-road testing is expensive and very situation-specific; thus, most research testing is completed with simulations, hardware bench tests, or with surrogate systems such as small-scale vehicles [12], [17]. In the area of small-scale vehicles, Goldfain et al. created AutoRally to research aggressive driving in the off-road environment [17]. The AutoRally is a $1/5^{th}$ scale remote control vehicle equipped with cameras, GPS, IMUs, and other hardware for research. The AutoRally, as seen in Figure 2.7, is also highly designed to withstand crashes and other potential failures due to aggressive off-road driving. The program used to run AutoRally was in fact simulated initially, but ultimately, the actual vehicle was tested on an off-road track [17]. The research conducted in this thesis resembles that of AutoRally in which the goal is to follow an optimal path algorithm. This thesis and corresponding research intend to extend the study to include different learning control models along with path-planning.

Figure 2.7 - AutoRally Vehicle Undergoing Autonomous Testing at Georgia Tech [17]

Successes in Path-Following Algorithms

Path-following and planning is not entirely new. It has been successful on many occasions such as the research conducted by Guo et al. [11]. Their novel contribution to path-following was the consideration of vehicle dynamics into their algorithm. Previous research treats the vehicle as a point load object rather than a three-dimensional space. The model predictive control (MPC) algorithm could handle the additional constraints that prior algorithms were unable to do in path-following. The research created a highly optimized path with dynamic predictions which were very favorable, but MPC algorithms require a lot of computational power to run which is a potential downside to this algorithm.

Another success in path-following was research that investigated environmental detection and mapping for off-road autonomous driving [18]. This research explored the power of LiDAR and Vislab Embedded Lane Detection (VELD) to understand an obstacle course. Other sensors such as CCD Cameras, IBEO, and SICK scanners were implemented to read the total

surroundings of the vehicle. Like the research conducted by Mayuku et al., the vehicle had an intelligent component driven by the camera to decide where to go [15], [18]. Figure 2.8 demonstrates the image processing pipeline in Mayuku et al. research which successfully identified varying terrain. The LiDAR sensors were responsible for detecting obstacles and assigning a point cloud of data corresponding to it for the computer of the autonomous vehicle to evaluate. The evaluation simply determined how aggressively the object should be avoided [18]. The path was then created by following the line of objects that avoids all obstacles the most efficiently in terms of path desirability. The vehicle succeeded in avoiding obstacles and driving smoothly [18].

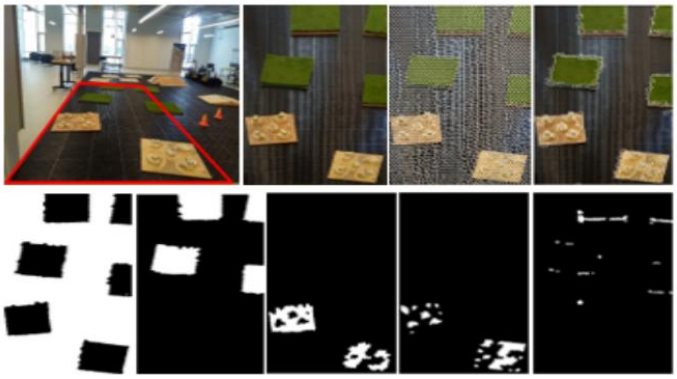


Figure 2.8 - Image Processing Pipeline Which Determined Terrain Types [15]

Gaps in Literature

All the prior research succeeded heavily in their specific goals and objectives. Prior research, however, includes gaps that are the forefront of this theses' goals. The largest gap in literature is that many previous paths following and planning algorithms only address one or two parameters that make up autonomous driving. The environmental detection research focused on being able to create plotted environmental maps of obstacles for the computer to understand [18].

The MPC model algorithm addressed vehicle dynamics mostly [11]. The 3D environmental simulation research only addressed the process of creating a simulation on Simulink of an environment for a point load autonomous vehicle to traverse [16]. The data-based approach for autonomous motion research studies focused entirely on organizing data and creating cost functions for identified obstacles [8], [10].

Another gap that much of the prior research lacks is the concept of having the robot learn from itself. The terrain identification research conducted by Mayuku et al. does include a level of artificial intelligence [15]; however, the AI is focused on external variables, the terrain, rather than internal variables such as the vehicles speed, yaw angle, and trajectory.

A form of learning which has been previously explored in other applications and currently being explored for autonomous vehicles is that of iterative learning control (ILC). ILC is an algorithm practice aimed at improving an operation through iterative repetition [19]. A simple analogy of ILC is that of practicing free throws in basketball. When shooting a free throw, the thrower receives immediate feedback whether the shot was made or not. If the shot was missed, there could be various reasons why. Such reasons include the ball was thrown too hard, too soft, or off trajectory in any direction. The thrower then can adjust their shot in subsequent attempts until the ball goes in. ILC follows the exact same principle as repeatedly shooting free throws [19]. ILC is an open-loop algorithm which can be used for a robot to learn from iteratively repeating a task. Figure 2.9 demonstrates the basic ILC configuration of a system per Ahn et al. [20]. The lack of ILC-related autonomous vehicle research is a gap in literature addressed in this thesis.

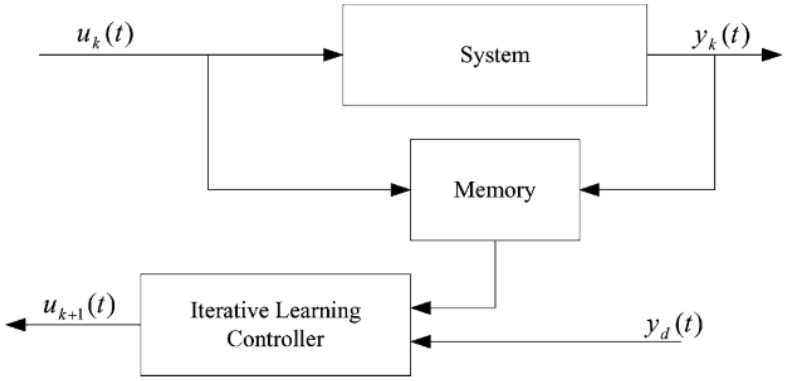


Figure 2.9 - Basic ILC I/O Model Configuration [20]

Ultimately, there is room for the exploration of combining previous algorithms, research, and novel considerations that can provide a new infrastructure for autonomous vehicles in the off-road setting. This thesis combines various forms of intelligence to learn how to optimize vehicle speed on an off-road multi-terrain course. The goal is to design a vehicle with the necessary infrastructure and intelligence to autonomously traverse a course repeatedly with increasing speed based on iterative learning control.

Chapter 3

Vehicle, Hardware, and Software Overview

Vehicle Overview

The vehicle subject of this thesis is the Losi DBXL-E 2.0 RC Buggy which is a 1/5th scale vehicle. The vehicle is powered by a brushless electric motor with four-wheel drive capabilities due to front and rear differentials. Figure 3.1 shows the Losi DBXL-E 2.0 RC Buggy as it comes stocked.

Figure 3.1 - Losi DBXL-E 2.0 RC Buggy Original Configuration [21]

The listed vehicle above has been heavily modified to maximize torque power, ensure means of data collection through sensors, and to achieve autonomous path-following. The hardware that has been added to the vehicle's assembly include four high resolution rotary encoders on each wheel, custom encoder mounts, a dual-band GPS receiver, an emergency stop system, an emergency stop system mount, and the on-board electrical circuit. The hardware that has been removed from the vehicle's assembly consists of only the front differential. Figure 3.2 shows the modified Losi DBXL-E 2.0 RC Buggy.

Figure 3.2 - Losi Buggy with Sensor Attachments as of March 2024

RC Car Platform

The Losi DBXL-E 2.0, with outer dimensions measuring 33.15 inches in length, 19.5 inches in width, and 12.5 inches in height, is designed as a 1/5th scale representation of an automobile. The vehicle weighs an estimated 30.5 pounds out of the box. The electric vehicle, propelled by a Spektrum Firma 4-pole 780Kv brushless motor managed by a Spektrum Firma 160-amp brushless smart electronic speed controller (ESC), is rated for a top speed of 50 mph. Designed for rigorous off-road driving, the vehicle is assembled with a four-wheel-drive drivetrain with front and rear differentials, coupled with four aluminum shocks in its suspension. Figure 3.3 shows the described system architecture of the Losi DBXL-E 2.0 RC Vehicle.

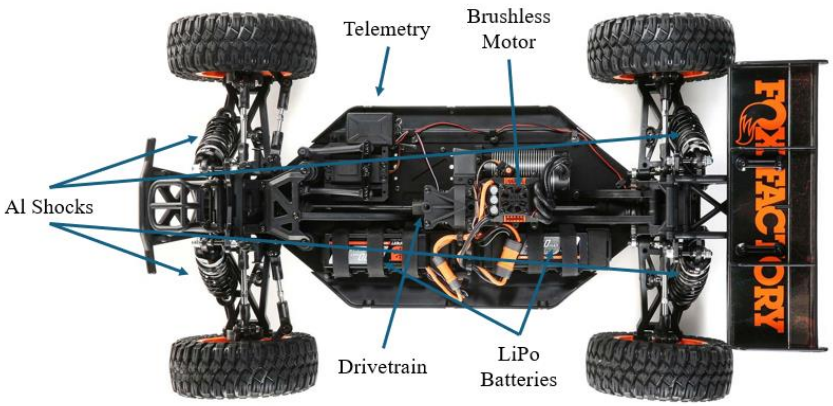


Figure 3.3 - System Architecture of the Vehicle [21]

The vehicle includes the Spektrum SR6100AT 6-channel AVC telemetry receiver and a Spektrum S906 1/5th -waterproof servo. The Spektrum DX3 2.4GHz remote transmitter facilitates communication with the on-board receiver, which transmits steering and throttle commands to the respective components.

In the context of this research, the vehicle's control system is explored further. A microcontroller, such as a Teensy 4.1 microcontroller, can interface between the receiver and the ESC and steering servo through a standard 3-pin pulse width modulation (PWM) connection. This interface allows for implementing steering and throttle control through the Teensy 4.1 microcontroller.

An electric RC vehicle was selected for this project to mitigate the potential hazards associated with gas-to-electric conversions and minimize the risk of fire or explosion during high-speed collisions. The DBXL-E 2.0 RC platform operates with two Spektrum 14.8V 5000mAh 4S Smart LiPo batteries as seen in Figure 3.4. Each battery cell has a nominal voltage of 3.7V and can achieve a maximum voltage of 4.2V per cell, resulting in a total maximum

voltage of 16.8 volts. These rechargeable batteries are directly connected in series to the ESC, providing power to the motor and ESC.

Figure 3.4 - LiPo Batteries Used to Power the Vehicle and Emergency Stop System[22]

Additionally, the vehicle's sensors, microcontroller, encoders, and related components are powered by two 3.7V Lithium Ion in 2000 mA batteries as shown in Figure 3.5. Voltage regulators play an important role in converting the battery output voltage to the required 5V and 3.3V for optimal functioning of the electronic components in the system. These details are explained further in subsequent sections of this thesis.

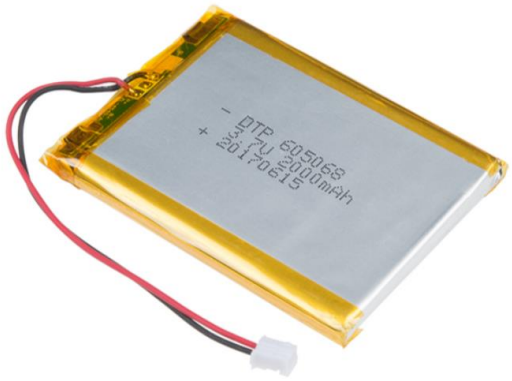


Figure 3.5 - Lithium-Ion Batteries Used to Power Data Collection Sensors [23]

Vehicle Modifications

In the pursuit of enabling autonomous path-following capabilities for the Losi DBXL vehicle, various modifications to the vehicle were implemented. In prior research on the vehicle, the front center drive shaft was removed, thereby disconnecting the front wheels from the drivetrain. This modification facilitated the installation of front disk brakes, allowing for independent braking of the front wheels [12]. To ensure continued power transmission to the rear wheels, a modified aluminum I-beam was employed to lock the front output of the center differential [12], [24]. Concurrently, the removal of the stock RC vehicle's spoiler and plastic passenger insert took place, while preserving the integrity of the base frame, roll cage, and plastic covering.

The implementation of an Anti-lock Braking System (ABS) was necessitated by prior thesis work conducted by Stephen Maransky [12]. Most electric RC vehicles typically rely on the electric motor to break all four wheels simultaneously. Consequently, the vehicle design incorporated front wheel brake installation. Subsequently, the front drive shaft and differential were reinstalled during Micah Delattre’s tenure with the vehicle [24]. A previous data collection revealed operational issues, particularly when driving off-road at the vehicle's maximum speed. Under these conditions, the back differential locked, rendering the vehicle inoperable. The aluminum I-beam, which initially locked the front differential, failed, causing severe vibration at a high frequency, and effectively locking the entire drivetrain. To address this issue, the front drive train and differential were reinstalled, restoring four-wheel-drive capability for Micah Delattre [24].

Due to assembly issues during the reinstallation process, the front differential often disconnected from the wheel shaft during data collection, thus causing the vehicle to operate

normally without the front wheels engaged. To prevent a front differential lock or unnecessary wear on the gear box redirecting the driveshaft, the front driveshaft and front differential were once again removed entirely, thus yielding the vehicle to be rear wheel drive. This decision is also substantiated by the changes in weight distribution of the vehicle. Subsequent additions to the vehicle greatly increase the weight in the rear of the vehicle. Increased traction due to rear wheel drive will greatly benefit the vehicle's acceleration and overall performance when in high-speed applications.

Another significant modification to the vehicle involved the incorporation of an emergency stop system, detailed in subsequent sections, which allows the operator to wirelessly shut down the vehicle with a simple push of a button.

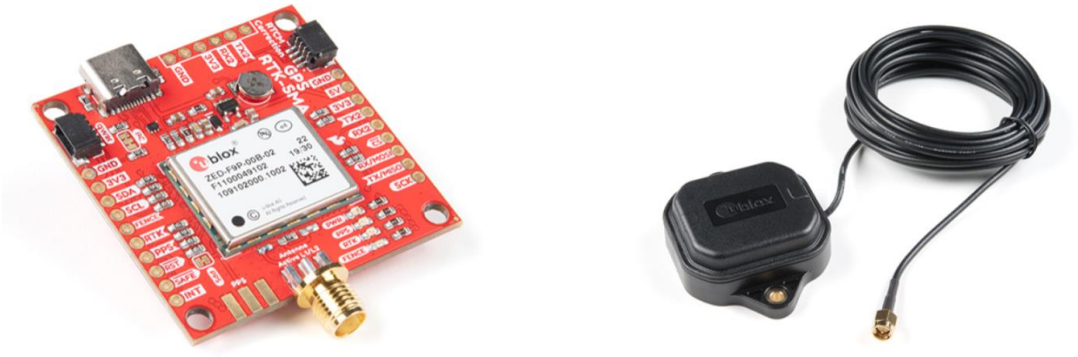
Hardware Overview

Sensor Implementation

A critical requirement for autonomous driving capabilities is high-resolution wheel rotation sensing. Such measurements play an important role in determining wheel position, speed, slip, rotation count, direction, and many other metrics involved in advanced algorithm development for autonomous vehicles. Wheel rotation data is obtained through a feedback signal generated by sensing devices like rotary encoders. The installation of US Digital H5 ball bearing optical encoders, as seen in Figure 3.6, on each wheel addressed this need. These industrial rotary encoder sensors provide high-resolution measurements with a resolution of .018 degrees and output 20,000 pulses per revolution, translating wheel rotation into a digital signal interpretable by a microcontroller.

Figure 3.6 - US Digital H5 Ball Bearing Optical Encoders attached to the Vehicle [7]

In the realm of autonomous driving functionalities, having a high-precision GPS positioning system stands as a vital requirement for the vehicle to know its own relative location. The capacity to know the precise location of the vehicle is important for the computation of velocity and accelerations. To fulfill this necessity, the vehicle is equipped with a SparkFun GPS-RTK Breakout Board, as seen in Figure 3.7, incorporating the ZED-F9P receiver and a multiband global navigation satellite system (GNSS) u-blox antenna dedicated to satellite positioning.



a) b)

Figure 3.7 - GPS Communication System a) GPS-RTK Receiver [25] **b) GNSS Antenna Mount** [26]

The accuracy of GPS measurements is dependent on various external factors such as radio interference, atmospheric conditions, and meteorological phenomena in general. These

external influences introduce distortions into the radio signals, thereby limiting the precision of GPS-derived positional data to the extent of multiple centimeters or even meters which is suboptimal for autonomous vehicle performance. Mitigating this challenge, the ZED-F9P receiver leverages the utilization of two distinct signal bands, denoted as L1C/A and L2C, which operate at different frequencies. This dual-band approach enhances the accuracy of the GPS positioning system, providing a more resilient solution in the face of signal disturbances induced by environmental and atmospheric variables.

The mitigation of distortions can also be achieved through the implementation of real-time kinematic positioning (RTK). To activate RTK, a stationary base station positioned at or close to the vehicle's designated test track is chosen and identified, with a predetermined and precisely known geographic location. The base station configuration encompasses an additional ZED-F9P receiver, a u-blox antenna, a telemetry radio, a 5V regulator, and a power supply comprising two 3.73V batteries arranged in series.

The wireless communication between the telemetry radio situated at the base station and its counterpart attached to the vehicle is operated by the HolyBro SiK Telemetry Radio V3, as seen in Figure 3.8, radios on each system. The telemetry radio has a rated output power of 100mW and operates at a frequency of 915MHz. The radio can also establish a fully transparent serial link with other radio stations. This transparency ensures that the signal input into one radio precisely mirrors the signal output by the other. Algorithm development related to the communication between the base station and the onboard GPS unit are discussed in subsequent chapters.

Figure 3.8 - HolyBro SiK Telemetry Radio V3 [27]

The base station's stationary GPS receiver generates corrections by recording signal distortions evident from perceived timing changes despite a stationary position fix. Simultaneously, these corrections are transmitted wirelessly to the vehicle via the HolyBro SiK Telemetry Radio V3. The onboard receiver collects these corrections, applying them to the satellite signals received to yield highly precise positioning measurements through the mechanism of the RTK positioning.

The stationary base station is adaptable to two configurations: a permanent base station or a temporary alternative. Due to the current absence of a permanent base station at the test track, the establishment of a temporary base station at the off-road course is necessary for current research needs. The configuration of this base station mandates the selection of a distinct and consistent location for antenna placement. At the designated off-road course area, which is discussed in more detail in subsequent chapters, a specific location marked by a permanent manhole cover was identified and chosen for the antenna's placement. Prior to setting up the antenna, GPS data was logged for an extended period exceeding four hours by Micah Delattre in previous research [24]. This data was subsequently uploaded to the Canadian Spatial Reference

System Precise Point Positioning Service (CSRS-PPP), which, after processing, yielded a precise position of the antenna with an accuracy of approximately 7 mm in latitude and longitude [24]. This precise position was then configured to the base station receiver, enabling the transmission of RTK corrections to the moving vehicle's receiver. It is imperative to note that, for each RTK-based GPS data collection, the base station antenna must be precisely positioned in accordance with the initially generated geographic coordinates. This was tested several times throughout the research process to ensure stability.

Connecting everything together, vehicle data acquisition is conducted by a Teensy 4.1 microcontroller, as seen in Figure 3.9, due to its compatibility with Arduino code programmed as well as its superior features. Different from the Arduino Uno by its enhanced features, processor speed, and computing power, the Teensy 4.1 operates at a speed of 600 MHz, possesses 55 interrupt-capable digital input pins, and operates on 3.3V logic. Notably, the Teensy 4.1 incorporates a built-in SD card reader, instrumental in the storage of all collected data onto a micro-SD card. This facilitates the subsequent retrieval of the micro-SD card for data post-processing which is executed on MATLAB.

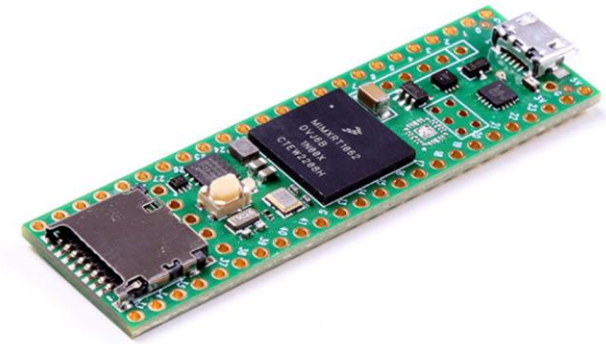


Figure 3.9 - Teensy 4.1 Microcontroller [28]

The use of SparkFun logic level converters is necessary to interface with the Teensy 4.1, given its 3.3V logic requirement, to allow for signal reading of the more conventional 5V logic utilized by most devices. This choice of logic level converters is motivated by their superior performance, avoiding electromagnetic interference issues inherent in alternatives such as voltage dividers. All sensors are meticulously connected through soldered wiring in a base printed circuit board (PCB) which is attached to the Teensy 4.1's via an electronic screw terminal block.

Mounting and Packaging

The installation of optical encoders on each wheel required the development of a custom mounting solution, characterized by the imperative criteria of rigidly holding the encoder during vehicle operation, facilitating complete encoder functionality, and ensuring durability to withstand operational stress. Following an iterative design process and experimentation, a solution meeting these prerequisites was achieved. The resulting solution comprised three distinct components mounting the encoders.

The first component, the wheel-nut adapter, was devised to securely grasp the wheel nut, thereby stabilizing the tire while also attaching directly to the encode shaft exteriorly. As the tire rotates, the adapter rotates correspondingly, thereby rotating the encoder shaft at the same angular velocity as the tire. For the secure affixation of the encoder read head, custom mounts were strategically positioned on both front and rear wheels. Figure 3.10 shows the external and internal features of the wheel nut adapter.

a) b)

Figure 3.10 - Wheel Nut Adapter a) Exterior Model b) Interior Model

On the front wheels, a metallic mount connects the encoder secured to the wheel adapter with the spindle carriers located on the wheel's interior as seen in Figure 3.10. Similarly, on the rear wheels, a multicomponent mounting system fixes the encoder to the wheel adapter with the rear suspension system. A 3D printed double jointed mount attaches directly to the rear suspension mount on the tire's interior. A laser-cut 5051 Aluminum bracket wraps around the rear of the vehicle directly onto the encoder secured to the wheel adapter as seen in Figure 3.11. The modeling and manufacturing process meticulously accounted for factors such as vibration, 3D print orientation, stress direction, weight considerations, and mount geometry.

Figure 3.11 - Rear Encoder Mount Metal Bracket

On the vehicle itself, the u-blox antenna and the HolyBro telemetry radio found their mounting locations on the roof. The magnetic antenna adhered to a circular 4-inch diameter steel ground plate, affixed to the plastic roof through bolting. Meanwhile, the telemetry radio was securely attached to the roof using an adhesive patch. Internally, within the RC vehicle's main chamber, an electronics packaging box is included, housing the onboard electrical circuitry detailed upon in subsequent chapters. To address challenges encountered during off-road data collection, where regular breadboard connections proved susceptible to vibration-induced disconnections, a waterproof metal box was adopted, featuring customized front and back panels with DB-9 Connector slots, as seen in Figure 3.12 below, to ensure constant polarity.

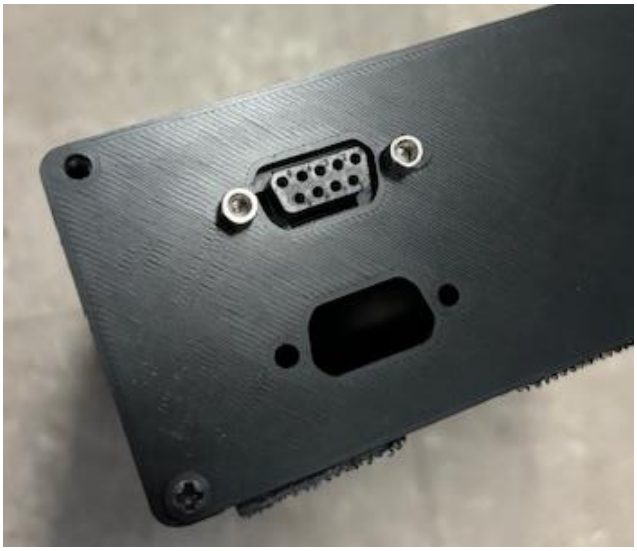


Figure 3.12 - DB9 Connector Inserted on Front Plate of Electronics Box

Notably, for enhanced repeatability in off-road data collection, all electrical connections within the circuit were soldered onto a blank PCB board. The switch, battery connections, and other components were steadfastly secured to the customized front panel with DB-9 connection points. This design approach, as shown in Figure 3.13, not only mitigated the issue of wires coming loose during high-speed off-road driving but also streamlined the data collection process while maintaining polarity in wire connections. A user, aiming to collect data, can simply secure the battery connections, secure the sensor connections, activate the power switch, and initiate the data log button.

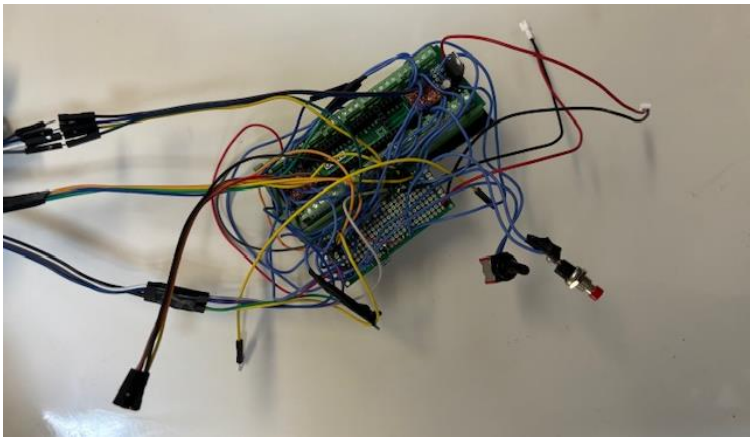


Figure 3.13 - Soldered PCB Electronics into a Push Pin Wiring Block

Also notable to the electronics box, a 3D printed baseplate with sliding insert capabilities was designed and implemented as shown in Figure 3.14. This allows a user to easily slide the internal electronics, which are attached to the baseplate via dual lock reusable fasteners, out of the waterproof box. The dual lock fasteners tightly secure the electrical components to the baseplate, preventing excessive vibration and internal collision within the box due to robust movements.

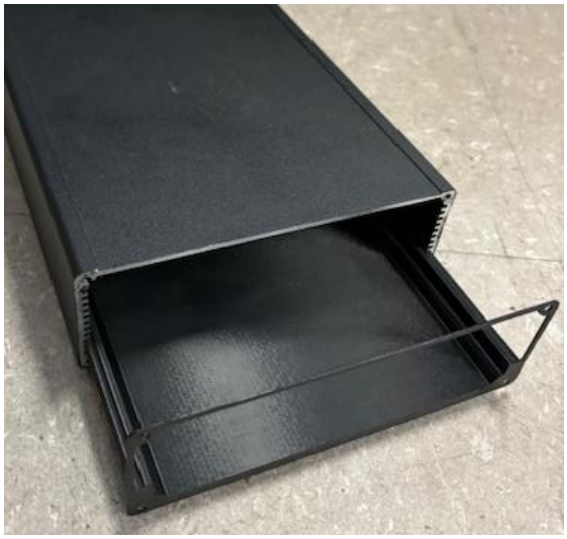


Figure 3.14 - 3D Printed Sliding Baseplate for Electronic Access

Control System Architecture

As previously mentioned, the vehicular data acquisition and recording is collected by a Teensy 4.1 microcontroller, operating on a 3.3V logic level. The Teensy interfaces with signals coming from the four rotary wheel encoders, the ZED-F9P GPS receiver, and the vehicle's receiver simultaneously.

The on-vehicle GPS receiver receives signals from both the on-vehicle GPS antenna and a corrected GPS signal transmitted via the telemetry radio. The system architecture flow chart delineates the intricate circuit wiring necessary for the seamless transmission of data signals. Figure 3.15 provides a visual representation of the schematic underpinning the vehicle's on-board electrical circuit. The circuit is powered by a tandem arrangement of two 3.7V LiPo batteries, serially connected, and further regulated to 5V. This regulated voltage powers the Teensy, wheel encoders, GPS receiver, and telemetry radio.

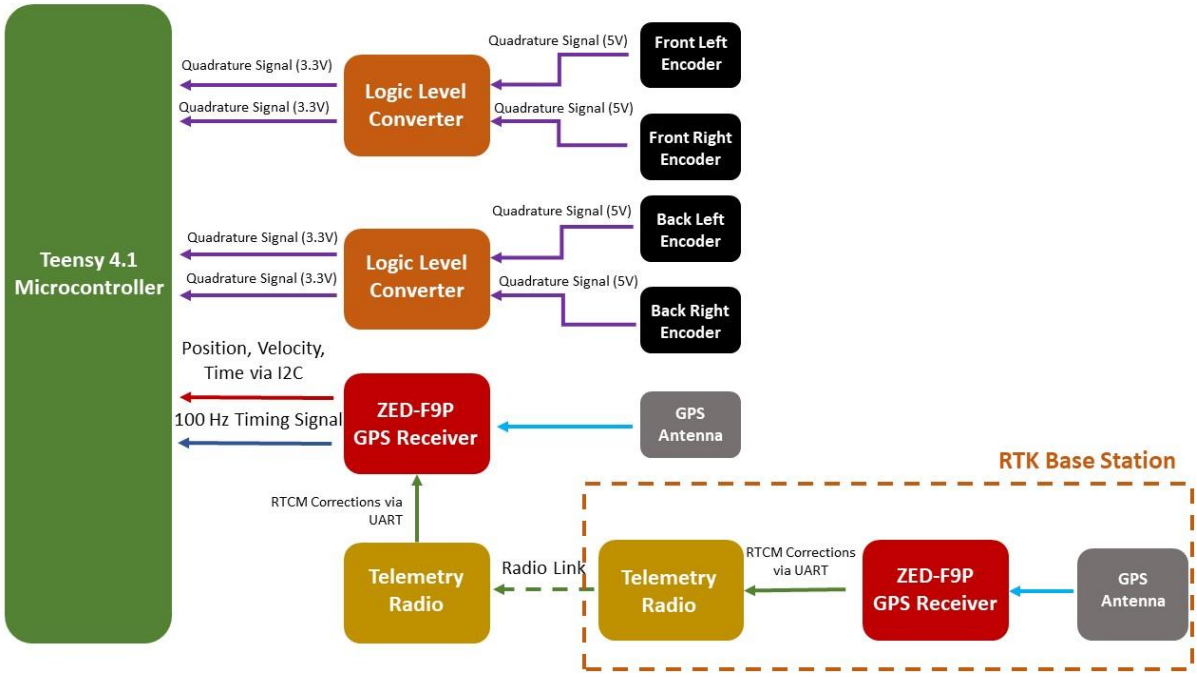


Figure 3.15 - Vehicle Data Collection Flow

Each wheel encoder encompasses two channels, denoted as A and B, emitting a quadrature signal characterized by an out-of-phase relationship which determines movement direction. This feature is favorable for vehicles capable of traversing in both forward and reverse directions. This research focuses on the forward direction; however, the rotation for the left and right wheels spins complementary to each other, thus the direction of the encoder matters. In essence, encoders on the right side of the vehicle will spin counterclockwise whereas encoders on the left side of the vehicle will spin clockwise. Wiring configurations entail connecting Channels A and B from each encoder to the 5V high voltage side of the logic level converter, while the 3V low voltage side is interfaced with specific digital pins on the Teensy 4.1 microcontroller. The rear encoders are linked to the circuit through shielded twisted pair cables, while the front encoders utilize unshielded CAT5 cables.

The telemetry radio's signal wires establish a direct connection with the GPS receiver, with the latter transmitting its signal to the Teensy through three distinct digital pins. Noteworthy is the strategic inclusion of capacitors throughout the circuit. These capacitors serve the crucial function of maintaining a stable voltage supply to the components, ensuring reduction of electrical noise and fluctuations. A full wiring schematic can be seen in Figures 3.16 and 3.17.

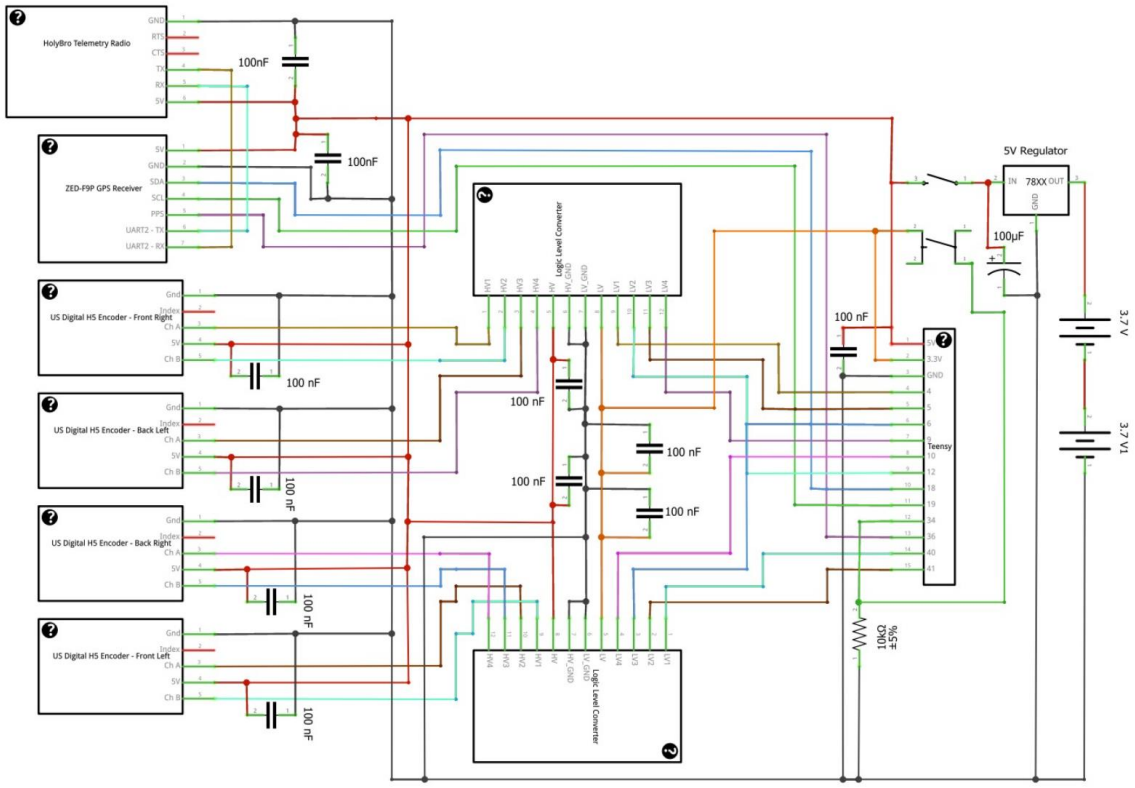


Figure 3.16 - Simplified Wiring Schematic of Objectives

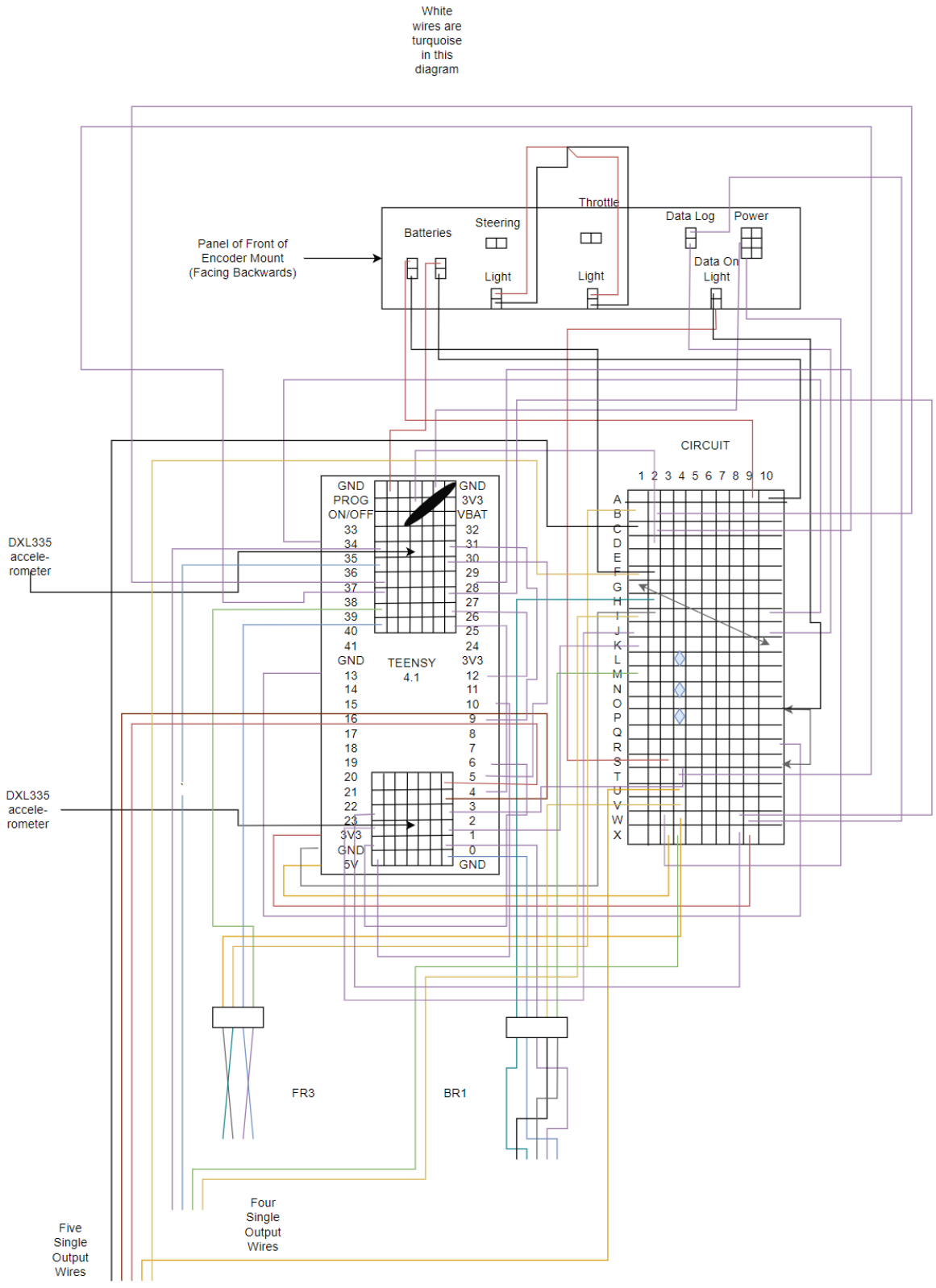


Figure 3.17 - True Wiring Schematic on the PCB and Screw Terminals

Emergency Stop System

In the pursuit of ensuring the safety of the Losi DBXL-E 2.0 RC vehicle, an emergency stop system was integrated. This system, acquired based on recommendations from the Intelligent Vehicles and Systems Lab Group at Penn State, features a Kar-Tech wireless emergency stop (E-Stop) system, as seen in Figure 3.18. The selected system operates on a 900MHz frequency, offering a range of up to 1.22 km. Components procured for this emergency stop system include a handheld wireless transmitter and a DC wireless E-stop 900 MHz receiver, both integral to the system's functionality.

a) b)

Figure 3.18 – Kar-Tech E-Stop System on Vehicle [29] **a) DC Wireless E-Stop Receiver b) Handheld Transmitter**

To augment this system, a Crydom D1D40 solid-state relay (SSR) was incorporated. This SSR, featuring 100VDC and 40A output capabilities, cuts power from the LiPo batteries to the Electronic Speed Controller (ESC) on the vehicle when the E-stop button is engaged. Prototyping was undertaken to validate the safety and efficacy of the system.

The initial benchmark circuit involved a simple LED connected to the E-stop receiver, demonstrating successful power interruption upon pressing the E-stop button. Subsequent tests

incorporated the SSR into the circuit, revealing the expected power cutoff both at the LED and a simulated ESC. The final test replaced the voltage supply with one of the 14.7V LiPo batteries, necessitating the introduction of resistors to align with the SSR's input current requirements. The prototype circuit functioned as intended.

Upon successful prototyping, the Kar-Tech emergency stop system was integrated into the vehicle. The wiring diagram of the emergency stop system illustrates the connection to the ESC and the power source as seen in Figure 3.19. The EC5 wired connectors were employed, with an important note regarding the color representation in the diagram.

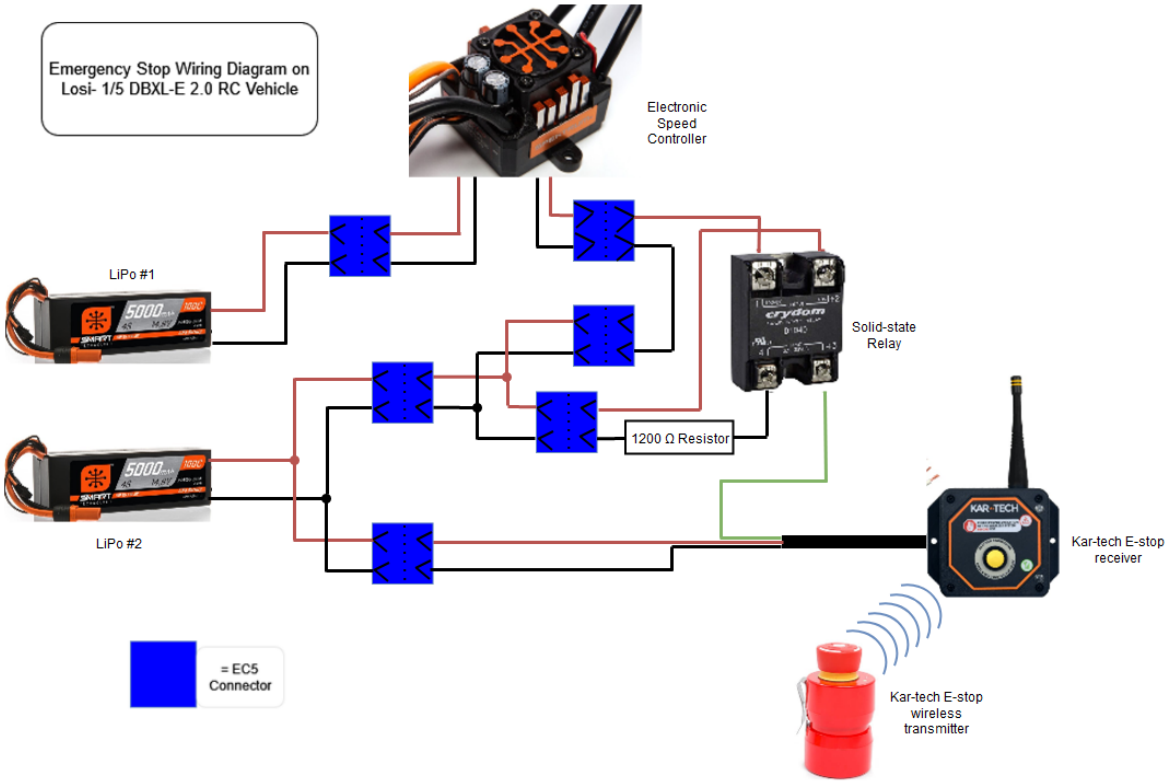


Figure 3.19 - Emergency Stop Wiring Diagram

During vehicle testing, the integrated emergency stop system functioned as expected with a confirmed range of roughly 640 m. When the E-stop button was pressed, power to the ESC and motor was promptly cut, halting the vehicle's motion. The E-stop receiver, mounted at the rear of

the vehicle on a 3D printed plate, further validated the system's practical implementation. This safety feature holds significant importance, particularly when operating the vehicle autonomously.

Software Overview

Data Parameters and Preprocessing

At system initialization, the Teensy 4.1 microcontroller executes a data-logging code programmed within the Teensy Integrated Development Environment (IDE). This code systematically acquires data signals from various sensors and logs each raw data parameter throughout the operational span of the vehicle. The recorded data is then stored onto the on-board microSD card in a text file format. This text file becomes the basis for subsequent post-processing operations performed on MATLAB.

The Teensy captures and logs 23 distinct data parameters, each at varying frequencies, as detailed in Table 3.1. The raw data is logged in a comma-separated value (CSV) format to facilitate high-frequency data collection. Upon initializing the data collection circuit, the Teensy executes the setup routine for the data logging code. This entails an initial check for the presence of an SD card. If detected, a .txt file is initialized, and an initial write of data commences. Subsequently, the code verifies the availability of RTK corrections and awaits the activation of the data logging button. Upon pressing this button, hardware interrupts are engaged for both GPS and encoder signals.

Table 3.1 - Data Parameters logged by the Teensy [24]

Parameter	*Unit*	*Frequency [Hz]*
Teensy Clock Time	Milliseconds	100
Encoder Count 1	Counts	100
Encoder Count 2	Counts	100
Encoder Count 3	Counts	100
Encoder Count 4	Counts	100
GPS Hour	Hour	20
GPS Minute	Minute	20
GPS Second	Seconds	20
GPS Millisecond	Milliseconds	20
GPS Nanosecond	Nanoseconds	20
Relative x	Millimeters	20
Relative y	Millimeters	20
Relative z	Millimeters	20
North Velocity	Millimeters / Second	20
East Velocity	Millimeters / Second	20
Up Velocity	Millimeters / Second	20
Vertical Acceleration	Millimeters / $(Second)^2$	20
Horizontal Acceleration	Millimeters / $(Second)^2$	20
Speed Acceleration	Millimeters / $(Second)^2$	20
Heading Acceleration	Millimeters / $(Second)^2$	20
Satellites in View	Dimensionless	20
Throttle	Microseconds	20
Steering	Microseconds	20

In this context, a hardware interrupt denotes an electronic signal received by the Teensy from a sensor, triggering a specific function within the code. For instance, each encoder emits a count signal every 10ms. The Teensy, responding to these signals, activates the corresponding code function, incrementing the encoder count for the respective encoder. The application of hardware interrupts is favorable in scenarios with multiple data parameters collected at different frequencies. Given that the Teensy is unable to simultaneously log all 23 parameters when signals arrive at rates of 100 kHz or higher, interrupts facilitate logging data at predetermined intervals, ensuring that data collection takes precedence over other code executions.

The subsequent loop segment of the code perpetually runs as long as data collection remains active. Within this loop, GPS and encoder parameters are logged to memory each time the GPS time interval interrupt is triggered. Time data is stored, and the Teensy continuously monitors the status of the data collection button. Upon pressing the data collection button, the data collection ceases, and the accumulated data is stored on the SD card. The circuit can then be powered down, and the SD card extracted for the processing of raw data into comprehensible graphs and figures. See Appendix A for datalogging code for the Teensy 4.1 microcontroller.

Post Processing

Post-processing of the data is executed in MATLAB. The raw data is initially imported, followed by the determination of encoder and GPS velocities. The GPS receiver logs velocities, which can be stored directly or subjected to further processing for data fusion. Encoder velocities are derived by calculating encoder counts per second, dividing by the corresponding time interval. Angular encoder velocity, in radians per second, is then computed by converting

encoder counts per second. Finally, linear encoder velocity is obtained by multiplying the encoder radians per second by the radius of the wheels. This assumes that a constant wheel radius is maintained across all wheels.

The GPS position data is acquired and plotted by extracting relevant parameters from the raw data file. Vehicle latitude and longitude can be visualized on a 2D map using MATLAB commands such as 'geoscatter' and 'geobasemap.' Examples of such figures are presented in Chapter 5.

Chapter 4

Steering and Throttle Control for Path-following

To enable autonomous path-following in the RC vehicle, steering and throttle control must be capable of receiving external signals from the microcontroller. Normally, these commands are input by the user to the transmitter unit of the remote-control vehicle, as depicted in Figure 4.1. The RC transmitter wirelessly transmits these commands to the receiver onboard the vehicle, typically connected to both the Electronic Speed Controller (ESC) and steering servo via a 3-pin wire connection. The receiver is powered by the battery connection to the ESC and distributes power to other servos, with the steering and throttle commands sent directly to the corresponding devices.

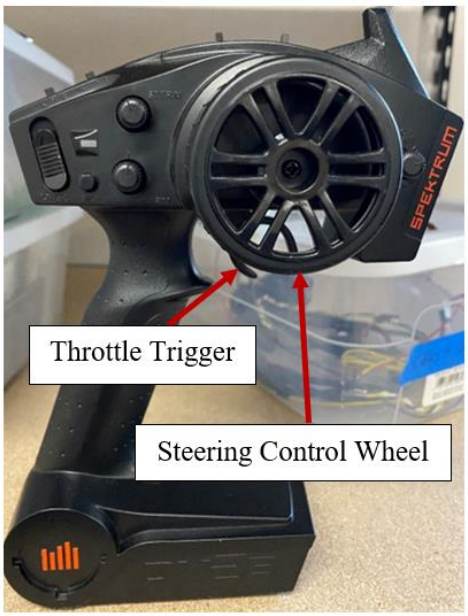


Figure 4.1 - Remote Control Transmitter for the Vehicle

The installed receiver is the SR61000AT AVC Technology Telemetry Receiver, as shown in Figure 4.2, featuring 6 channels, including a battery/programming port, a steering port

(channel 2), and a throttle port (channel 3). In the current configuration, only the steering and throttle ports are utilized. To facilitate steering and throttle control, the Teensy must establish connections between the receiver, steering servo, and ESC. The Teensy intercepts signals from the receiver, checks them using the steering and throttle limiting code, and then outputs limited signals to the ESC and servo. Figure 4.3 illustrates the schematic of the throttle and steering limiting control circuit, incorporating a double pull double throw switch (DPDT) for toggling between computer-driven and manually driven modes.

Figure 4.2 - SR61000AT AVC Technology Telemetry Receiver

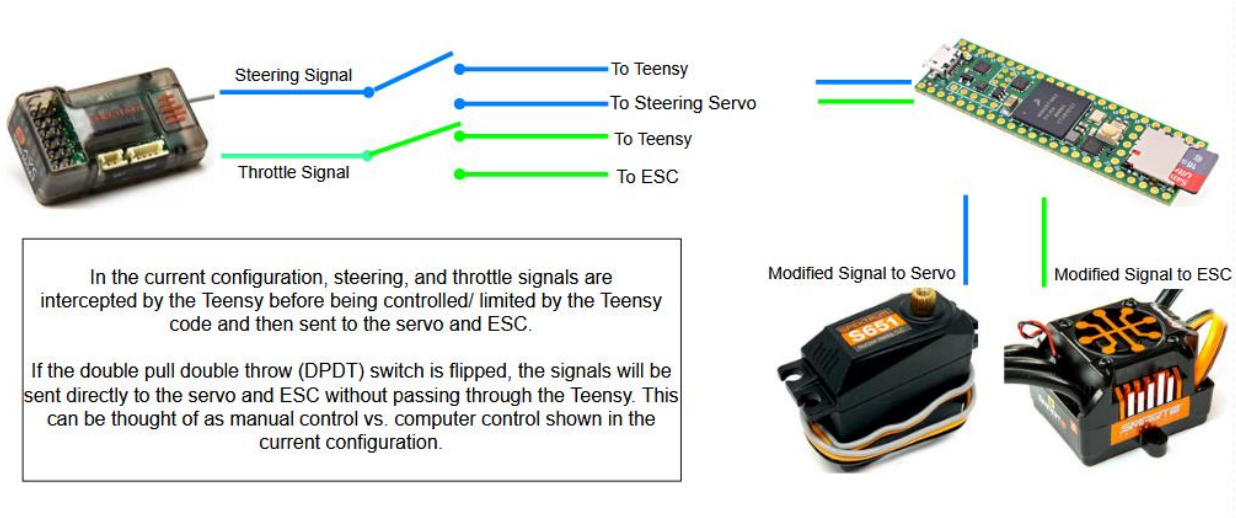


Figure 4.3 - Proposed Steering and Throttle Control Circuit Schematic

In computer-driven mode, the Teensy intercepts and modifies receiver signals, while in manual mode, signals are sent directly to the ESC and steering servo. The steering and throttle

limiting code defines steering and throttle signals as inputs, comparing them to predefined neutral, minimum, and maximum values stored in the Teensy permanent memory. If the input signal falls within the established range, it is output normally. If outside the range, the output is constrained to the nearest limit (minimum or maximum).

It is important to note that the connections between telemetry, Teensy, and servo have yet to be established due to project needs.

Chapter 5

Field Data Collection and Corresponding Results

Field data plays a critical role in analyzing the vehicle's dynamic system, designing robust algorithms, and further determining essential information regarding high-speed off-road autonomous driving. The information critical for the next steps of developing a highly intelligent system includes the states of the vehicle, such as velocity, acceleration, yaw, and power. The relationship between all the states ultimately determines how the vehicle maneuvers and its capabilities in each environment. The focus of the field data collection for this work was to fully define our states, course, and have sufficient information to accurately simulate the driving environment the vehicle is expected to endure. For this, a real-world test course was designed on an off-road course which challenges the vehicle's maneuverability while allowing the vehicle to reach maximum speeds. Before that, a relatively accurate estimation of the vehicle's maximum speed, acceleration, and distance to maximum speed was determined for maximum accuracy in designing our test course. Ultimately, the test course was then traced via GPS data of the vehicle conducting several laps. This data was refined, processed, and later simulated as will be discussed in subsequent sections and chapters.

Off-Road Test Course

The off-road autonomous vehicle is tested at the Penn State test track at a designated off-road area selected due to its variety in topological features. The selected course, as seen in Figure 5.1, for evaluating the autonomous vehicle is characterized by a steep inclined side hill, ditches,

irregular grassy terrain, and small hills. The course configuration deliberately incorporates various challenges to assess the vehicle's path-following capabilities comprehensively.

Figure 5.1 - Image of the Terrain for the Test Track

The development of the off-road test course underwent several iterations to achieve its finalized configuration. Initial iterations incorporated all the specified features, except the straight stretch, which fell short of meeting the crucial requirement for enabling the vehicle to attain its maximum speed. Ensuring that the vehicle reaches its peak speed on this straight stretch is important in achieving the high-speed component of the overall testing.

The original iterations of the course, designed by Stephen Maransky and Micah Delattre, were unable to fulfill the speed requirement. Figure 5.2a presents a plot of GPS data collected during the vehicle's operation on the initial off-road course design, highlighting the inadequacy in achieving the desired speed along the straight stretch. This observation prompted further refinement in subsequent iterations to optimize the course layout and address the identified limitations. Figure 5.2b presents a plot of GPS data collected during the vehicle’s operation of

the second off-road course design which also failed to achieve maximum speed along the straight stretch. This second course was primarily on asphalt which did not satisfy the off-road conditions desired.

a) b)

Figure 5.2 - GPS Data Plot of the Prior Off-Road Course Design

The current iteration of the course consists of several extended straight sections, with one elongated stretch allowing the vehicle to attain its maximum speed. To enhance the complexity of the course, deliberate inclusion of switchback segments necessitates the vehicle to execute significant deceleration for proper maneuvering during turns. Additionally, the course layout strategically introduces ditches and small hills, compelling the vehicle to moderate its speed to prevent unintended airborne trajectories or collisions with ditch walls.

A distinctive feature of the course involves a steep side hill where the vehicle must traverse vertically and horizontally, diagonal to the slope. This element adds a layer of difficulty to the autonomous path-following, requiring the vehicle to employ steering adjustments, slightly upslope, to maintain a constant horizontal trajectory across the inclined terrain. The multifaceted nature of the test course, as seen in Figure 5.3, aims to rigorously evaluate the autonomous vehicle's performance in diverse and challenging real-world scenarios.

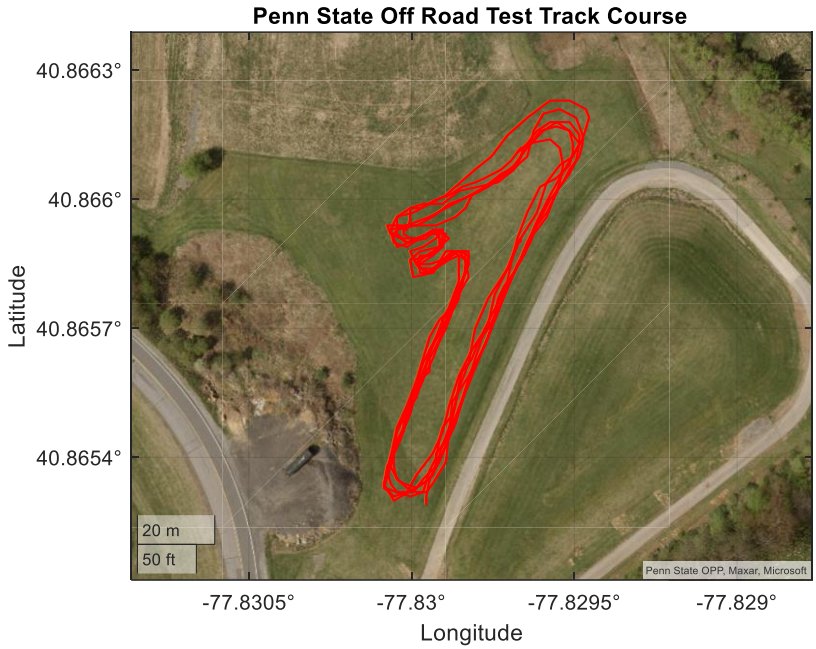


Figure 5.3 - Designed Test Track for 2024 Data Collection

Field Data Collection Process

Upon arriving at the test track, the initial step in the field data collection involved setting up the RTK base station. The tripod is positioned to align the center with the desired GPS position, verified using a plumb bob and a screw on the orange reflector. The GNSS antenna is centered on the tripod, followed by placing the RTK base station circuit box within the antenna's line of sight to all areas on the test course. Connecting the antenna cable to the GPS receiver in the circuit box and powering on the circuit completed the RTK base station setup.

Subsequently, the RC vehicle is positioned at the designated starting point on the course, with both LiPo batteries and the emergency stop system connected. The vehicle's onboard circuit batteries are also connected, the micro-SD card inserted, and the power switch is flipped on. To

conclude data collection, the data log power switch is flipped off, and the micro-SD card is removed for post-processing.

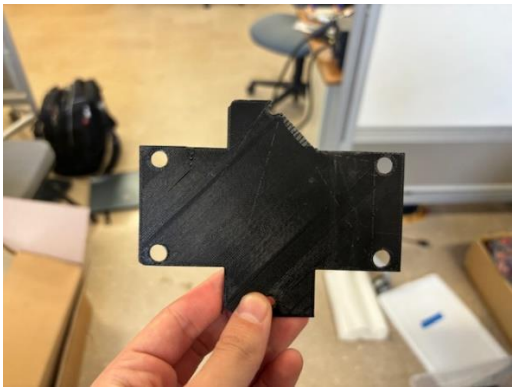
Depending on the test being performed, certain sensors can be isolated. GPS data can solely be collected by uploading only GPS datalogging capabilities to the Teensy Microcontroller. Likewise, can be performed for the encoders. An external accelerometer linked to a cloud-based system is also used in logging acceleration data which is isolated in its own nature.

On March 14th, 2024, a critical event occurred during the testing phase of the high-speed off-road autonomous vehicle project. The vehicle was subjected to a straight-line acceleration test, which will be discussed in subsequent sections, aimed at determining maximum speed and acceleration. Despite planning and preparation, the unforeseen loss of control by the vehicle near maximum speed resulted in a high-speed collision, causing significant damage to several components.

The collision led to the catastrophic failure of both the front left and back right encoder mounts, essential for precise speed and position sensing. Due to the nature of the front encoder's attachment, the wheel assembly came apart exposing the rotating front differential axle extension to hang loosely, thus removing control from the wheel. Additionally, the ESTOP mounting system, designed to keep the ESTOP receiver attached to the vehicle, was torn apart leading to the ESTOP to be dangerously dragged through rough terrain. This led to the ESTOP control button to be severely damaged. The aftermath of the collision posed further challenges as the GPS wires became dangerously entangled, and slight damage was found in several electronic connection points. Images of the wrecked components can be seen in Figure 5.3

a)

b)



c)

d)

Figure 5.4 - a) Detached and Broken Front Encoder Mount, b) Disconnected Front Wheel Axle, c) Torn Estop Mount, d) Dremel Repaired ESTOP Receiver

This incident emphasizes the importance of robust safety measures and highlights areas for improvement in both vehicle design and control systems to mitigate risks during high-speed operations. This incident also emphasizes the importance of testing with scaled vehicles compared to full-scale vehicles. Crashes like this can be costly and dangerous and thus it is difficult to commit research like this on full-scale vehicles. Future works on the vehicle will incorporate the addition of a roll cage, lateral bumpers, and object detection sensors to prevent similar damages.

GPS Traversal Data and Desired Trajectory

As previously seen in Figure 5.3, the off-road test track was outlined using GPS data from the vehicle as it traversed the stake-marked course. From this data, it is imperative to process the GPS data accordingly to create a desired reference trajectory for the vehicle. The desired reference trajectory is simply the track as well as its corresponding track limits such as one would see on a competitive motor speedway. Having this data is imperative for simulation purposes as well as programming purposes when it comes to intelligent path following and the subsequent path planning.

Figure 5.3 includes data from the GPS measured in latitude, longitude, and altitudes (LLA) coordinates. This leads to the first step of processing which is to convert the acquired GPS data from LLA coordinates to a coordinate system which is simpler to work with. For autonomous driving, this coordinate system is East, North, UP (ENU) coordinates given a relatively close reference point such as a base station. The conversion from LLA coordinates to ENU coordinates is essential for effective controller implementation in navigation and control applications within autonomous driving systems. ENU coordinates are easy to handle mathematically in terms of computing distance, velocity, and acceleration. ENU coordinates are also a more intuitive representation of the vehicle's motion.

Unlike the global LLA coordinate system, the ENU coordinate system provides a local tangent plane centered at a specific Earth surface point, with the positive *x*-axis aligned eastward, the positive *y*-axis aligned northward, and the positive *z*-axis oriented vertically upwards from the tangent plane. This transformation enables a clearer understanding of the vehicle's position and orientation relative to its direct plane of site. The reference surface point is located at the

Penn State Test Track near the utilities building. Figure 5.5 demonstrates the conversion from LLA to ENU coordinates.

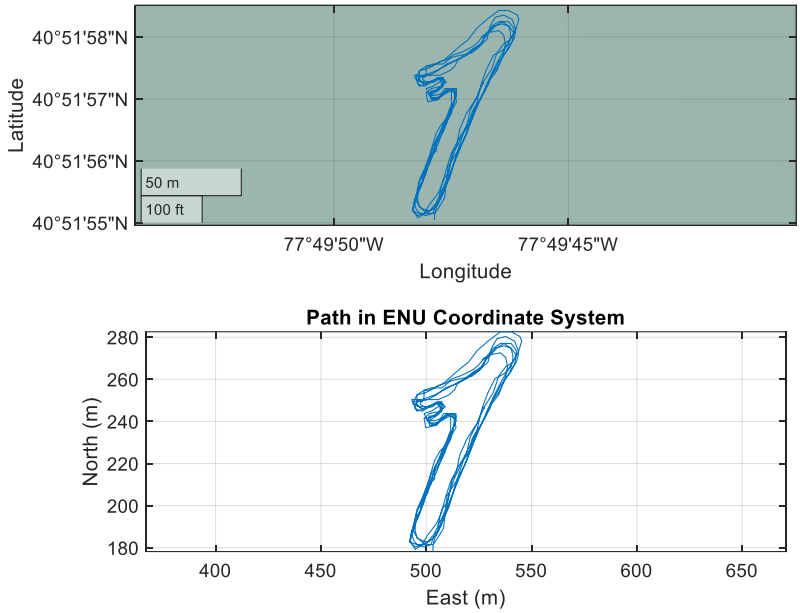


Figure 5.5 - LLA to ENU Conversion Plot

To extract a singular desired trajectory from the data and mitigate the issue of multiple overlaid traversals, the traversals must be broken down into laps. This segmentation is achieved through the establishment of zone definitions, which delineate regions marking the start and end of a single lap. These zone definitions are crucial in generating individual laps without needing the vehicle to hit precisely the same ENU point at the start of each lap. A line segment zone was defined for both the start and finish as can be seen by the green and red lines in Figure 5.6. The line segment zone allows our processing to trigger when the data starts a new lap and then subsequently ends a lap. A very broad line segment was chosen to ensure that all traversals comfortably fell within both zone definitions, and thus the function could then label the laps accordingly.

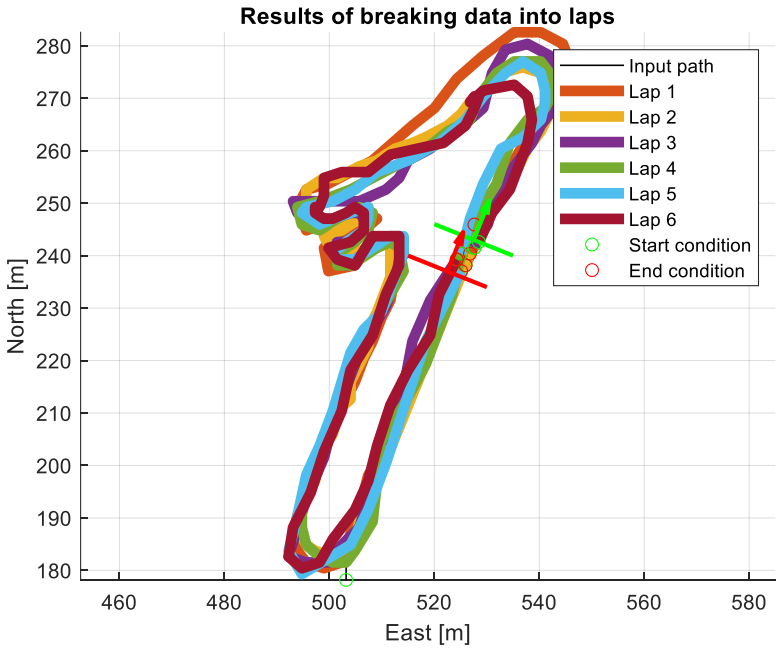


Figure 5.6 - Breaking the Data into Laps via Line Segment Definitions

At this point, the lap data has been further refined to include more points in between GPS points to ensure that there is a 10 cm (about 3.94 in) difference between all points. This is done to ensure consistency in points, provide excess data for more accuracy, as well as simplify the stations' processing steps. The lap data is converted to a station which is simply a one-dimensional distance term to define any point in the path. The station value at the starting index is zero, and the station value at the ending index is the ultimate path length. This one-dimensional representation of the path length along a traversal is beneficial in being able to determine the average reference traversal as well as giving us a known constant parameter that the vehicle is supposed to achieve in each traversal. One way to determine the average traversal of several laps is to define each lap into their station form of equal station lengths, then comparing corresponding stations and averaging out all ENU coordinates for a station, along every station, until one reaches the end of the track. This method can result in an extremely noisy average traversal due to the presence of outliers at any station, thus it was not utilized in this way

during processing. Note that a station is not a property of how far the vehicle has travelled, but rather a constant property of the track regardless of how the vehicle operates.

Given the lap data produced by the vehicle and later refined during processing, an average traversal was produced using orthogonal projections from a given lap in correspondence to its designated stations. This method was unsuccessful due to a sparsity of data points during the switchback segment of the track. In essence, the GPS failed to collect data rapidly enough relative to the vehicle's speed during the switchback segment, resulting in points of extremely acute angles as seen in Figure 5.7. These clusters of acute angles produced in the GPS data were unrecognizable by the orthogonal projection command, and thus required the average traversal to essentially decide to skip the switchback segment deeming that the best path given the collection of paths was to cut directly through the switch backs as if it were a single turn.

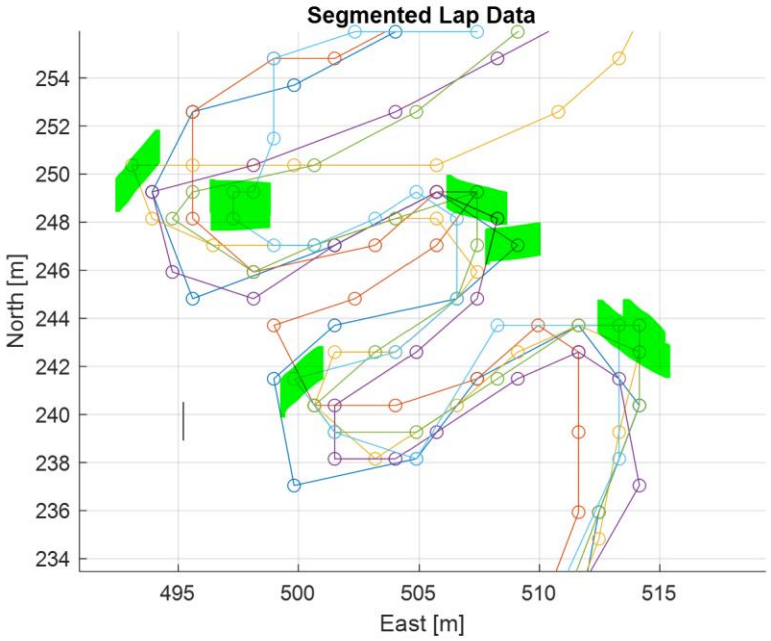


Figure 5.7 – Examples of Extreme Acute Angles in Switchback Data Highlighted in Green

Future iterations of this test should seek to drive the vehicle slower through these switchbacks to ensure more GPS points are collected to allow for a smoother reference map. Due

to the vehicle being temporarily inoperable due to the collision, a manual approach was taken to create the reference traversal used in Simulation. Using a manual review of the test area map using the ‘ginput’ function on MATLAB, nearly 400 points based on the previous lap traversals were manually selected. This ensured enough data was accounted for within turns to produce a smooth and realistic reference path traversal. Figure 5.8 illustrates the outcome of the manually selected points in green.

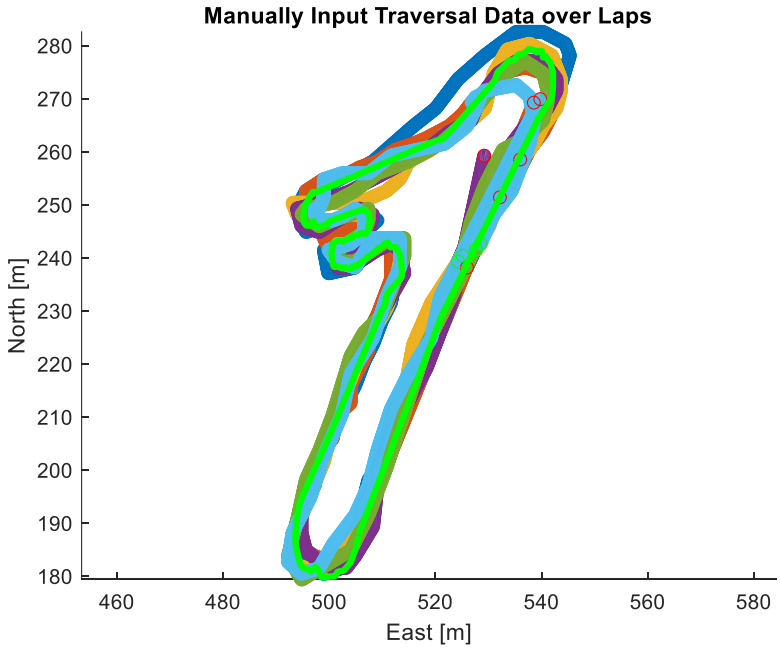


Figure 5.8 - Manually Input Traversal in Green

The manually input data similarly converted to a traversal, resampled for each point to be exactly 10 cm away from each other, and processed as a lap. Due to the noise and inconsistencies in manually inputting data, the traversal was smoothed using a distance-based zero-phase smoothing Butterworth filter.

To design the filter, conversion from time-domain filtering concepts to spatial-incremented domains. Knowing that each meter contained 10 points, derived from each point being 10 cm away from each other, we defined the spatial sampling frequency of this plot to be

10 spatial Hz, e.g. 10 points in every meter. Note that spatial frequencies are in units of spatial Hertz, defined as one sample per meter rather than the more commonly known definition of Hertz being one sample per second.

Per the Nyquist-Shannon sampling theorem, the Nyquist spatial frequency, or the minimum rate at which a signal should be sampled to accurately reconstruct the original signal without aliasing, is equal to half the spatial sampling frequency. This would yield a Nyquist spatial frequency of 5 spatial Hz. Using this information, the cutoff frequency for filtering was calculated as:

$$\omega_{cutoff} = \frac{\alpha}{\omega_{Nyquist}} \tag{5.1}$$

where α is an estimated cutoff ratio based on the desired filtration necessary. In this instance, α was deemed to be 1/10 to eliminate high-frequency noise while maintaining the integrity of the manually input lap traversal. With this calculation, a second-order Butterworth filter was applied to the manually input traversal using the 'butter' command and using the zero-phase 'filtfilt' command. The *x* and *y* coordinate data were filtered independently and converted back into a traversal. Figure 5.9 illustrates the newly acquired reference lap. The reference lap is a smooth representation of what the ideal lap should look like for the vehicle and will be an input in both simulation and path following applications. See Appendix B for GPS processing code.

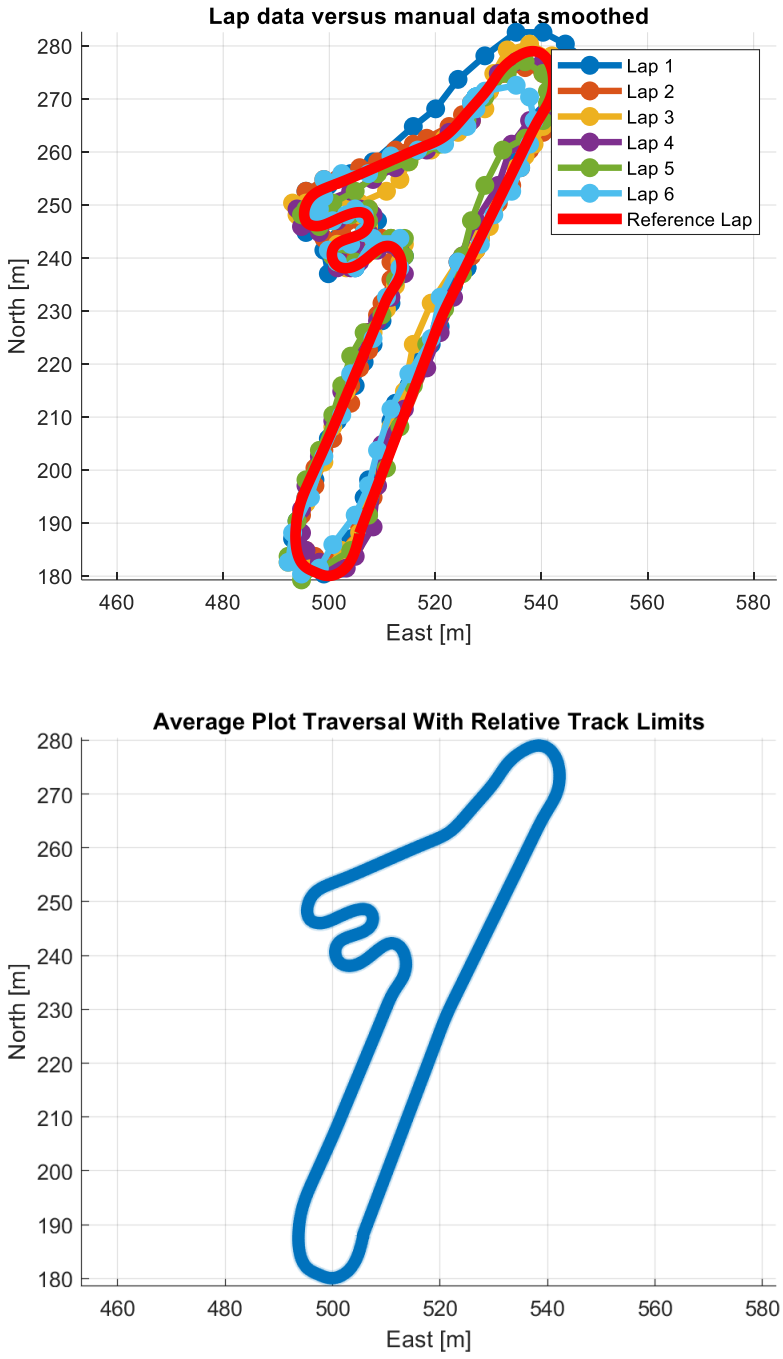


Figure 5.9 - Official Reference Lap for Simulation and Path Following

Acceleration Testing and Results

In the domain of autonomous vehicle development, comprehending and accurately quantifying the acceleration of the vehicle is a fundamental aspect for ensuring safe and efficient operation. The acceleration profile of an autonomous vehicle impacts its dynamic performance and influences its ability to navigate diverse driving scenarios effectively. Understanding the vehicle's acceleration capabilities is necessary as it directly influences factors such as trajectory planning, collision avoidance strategies, and maneuverability through a course like the one in Figure 5.3. Precise knowledge of acceleration aids in the calibration and validation of control algorithms such as the simulation model which will be discussed in a subsequent chapter, contributing to the robustness and reliability of autonomous driving systems. An important aspect of the research presented in this work is to understand acceleration to determine the distance it takes to reach maximum velocity. Achieving maximum velocity during a traversal is pertinent to the success of our research focusing on high-speed off-road autonomous driving. Therefore, it was pertinent to measure our vehicle's acceleration to understand its dynamic capabilities and design a track which maximized the vehicle's velocity capacity.

The determination of the distance to the maximum speed of the vehicle involved a preliminary on-road straight-line test and kinematics. The vehicle was positioned at a section of open, flat, pavement on the test track and accelerated to its maximum throttle while traversing the straight stretch of the test track. To gauge the distance to maximum speed, an accelerometer with readings at 30 Hz was strategically placed in the vehicle measuring the effective acceleration of the vehicle while performing the straight-line test. The data was processed on MATLAB by determining maximum acceleration segments and calculating the effective maximum acceleration at these points, effective velocity, and effecting position at the end of

segmentation when acceleration reached zero presumably at maximum velocity. This process was repeated twice with eleven acceleration segments used for calculation purposes. Plots of the acceleration data can be seen in Figure 5.10.

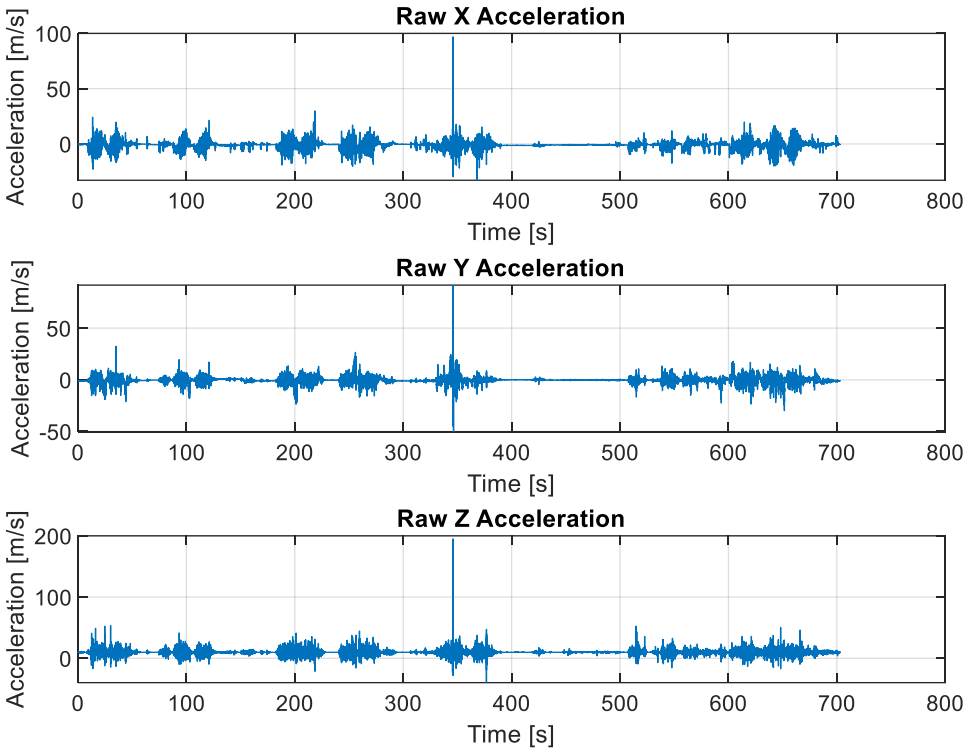


Figure 5.10 - Acceleration Data for the Two Acceleration Tests Performed

Note that the acceleration data exhibited in Figure 5.10 demonstrates the collision at around 350 seconds (about 6 minutes). The acceleration data presented after that major spike is due to a previously acquired acceleration data set which was combined with the data set that included the crash acceleration data. Information shortly before, during, and after the crash was intentionally omitted during processing. The best practice would have been to organize the acceleration data sets in order of their occurrence. However, due to time constraints they were

placed as such and processed accordingly. It is important to note that the order of appearance for the dataset has no effect on the yielded outcomes of this test.

In the domain of vehicular dynamics analysis, the *x* direction acceleration indicates the longitudinal acceleration aligned with the motion of the vehicle; the *y* direction acceleration encompasses lateral accelerations, extending to both left and right orientations; lastly, the *z* direction acceleration pertains to accelerations orthogonal to the ground plane, characterizing vertical movements of the vehicle. A model of this alignment can be seen in Figure 5.11. It is assumed that the accelerometer was consistently perfectly aligned with this relative coordinate system during the acceleration tests for processing purposes.

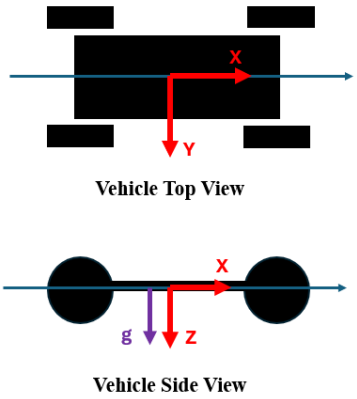


Figure 5.11 – Ideal Alignment of Accelerometer Coordinates for Acceleration Processing

Results regarding the *x* direction acceleration yielded that that the average acceleration of the vehicle on pavement equated to 4.40 m/s^2 with maximum acceleration data averaging between 3.10 m/s^2 and 4.90 m/s^2. Acceleration plots during the accelerated segments can be seen in Figure 5.12. Do note that the acceleration data is noisy due to high sensitivity of the instrumentation at such sampling frequency.

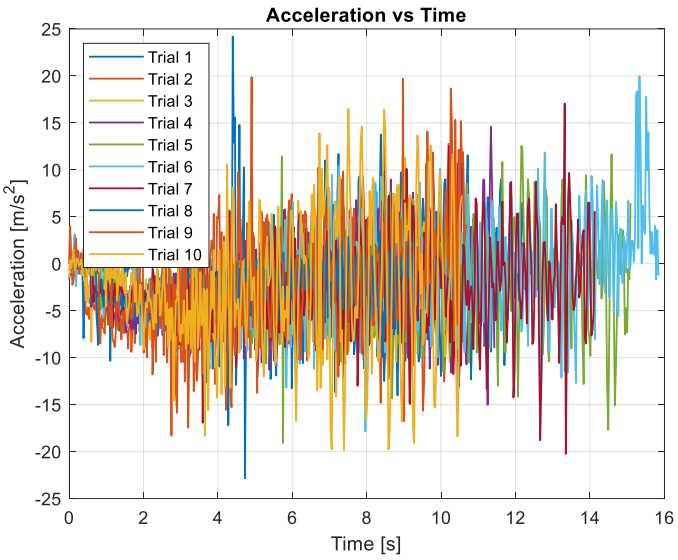


Figure 5.12 – Acceleration vs Time During the Various Segmented Intervals

The *x* acceleration data was integrated using a cumulative trapezoidal numerical integration command to yield estimated velocity data during the accelerated segments of the data set. Figure 5.13 demonstrates the velocity contours of the vehicle during the testing period per the estimations of the trapezoidal integration.

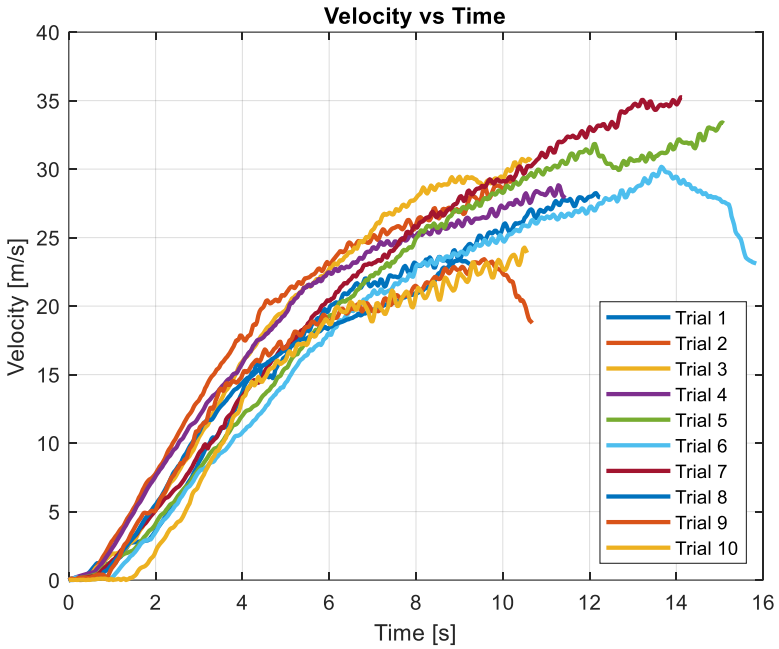


Figure 5.13 - Velocity vs Time During the Various Segments per Numeric Integration

With an estimated maximum velocity of 28 m/s, equating to approximately 64 miles per hour, as well as integrated results from the data plots, it was yielded that the average distance it takes to reach maximum speeds is 94 meters with the distance to reach maximum speed ranging anywhere from 59 meters to 146 meters per the yielded acceleration segments. Total recorded *x* acceleration data can be seen in Figure 5.14 and Table 5.1. See Appendix C for data processing code.

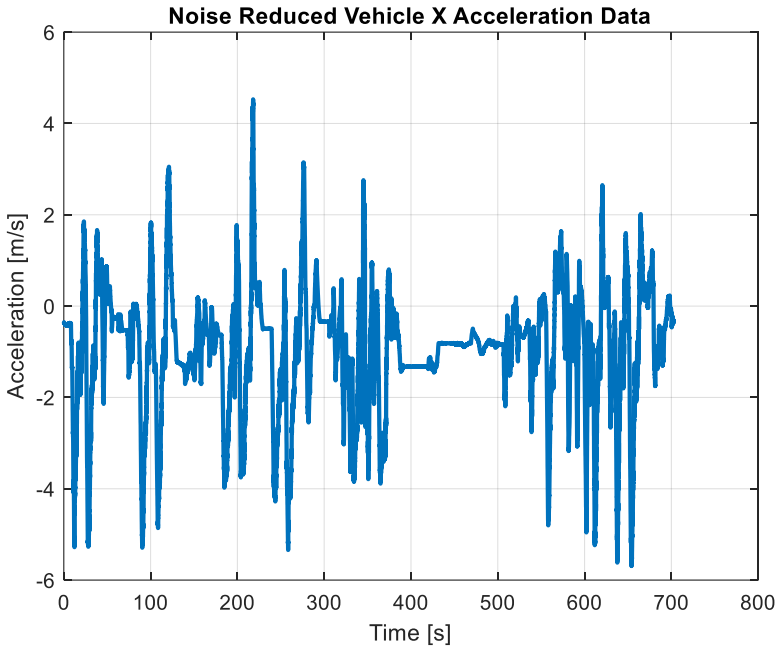


Figure 5.14 - *x* Acceleration Data Considered for Processing

Table 5.1 - Acceleration Data Analysis

Parameter	*Minimum*	*Maximum*	*Average*	*Standard Deviation*
Maximum Acceleration [m/s^2]	3.10	5.75	**4.40**	0.681
Maximum Velocity [m/s]	23.4	35.3	**28.7**	4.05
Distance to Max Velocity [m]	89.9	97.6	**93.8**	5.41

The provided data from the acceleration experiment provides both acceleration and velocity data which it is imperative to relate due to the significance of Force versus Velocity in vehicle dynamics. Understanding this relationship is good for optimizing the performance and control of autonomous vehicles. Force, as exerted by the vehicle's propulsion system, directly influences its acceleration capability. Higher forces lead to greater acceleration, enabling faster attainment of desired velocities. However, the relationship between force and velocity is more complicated than that; as velocity increases, the requirement for force is expected to decrease to

sustain or further increase acceleration. Power limiting and real-world conditions yield force vs velocity plots that are not as linear as can be seen in Figure 5.15. The various outcomes of Force vs. Velocity plots can be explained through causal means.

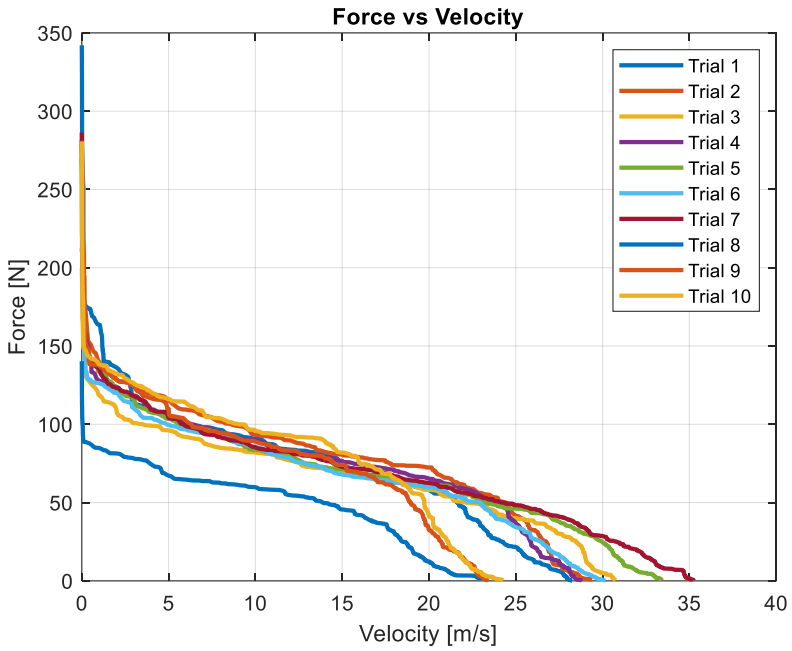


Figure 5.15 - Force vs. Velocity Plot for the Various Segments of Acceleration

Due to the nature of the acceleration test, the approximations and estimations calculated during processing must be considered only theoretically and not in practice. Further validation must be accomplished through more accurate testing methods to determine acceleration, velocity, and distance to reach maximum velocity. Testing acceleration without the assistance of other sensors can prove to be faulty and inaccurate due to sensitivity of such sensors compared to others as well as several calibration metrics which must be rigorously accounted for during testing periods.

One such calibration metric was consistently observed throughout the data and was assumed to be negligible during processing. However, this metric may be responsible for potentially overestimating the resulting data. Figure 5.11 demonstrates the idealized orientation

of the accelerometer during the entirety of the experiment. If the accelerometer were to be misaligned even to the slightest degree such as in Figure 5.16 below, then the processing methodology would need to consider the relative *x* and *y* axes offset in relation to the vehicles longitudinal acceleration denoted by a thin blue line.

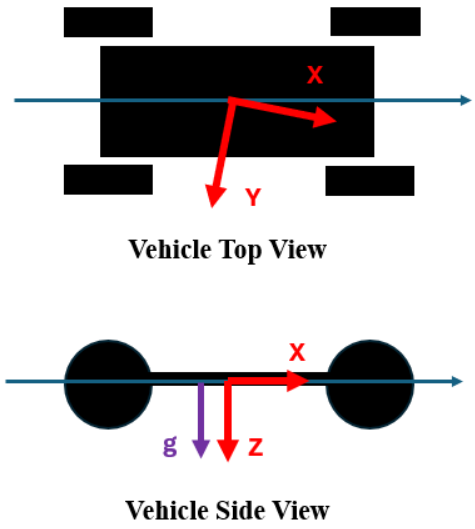


Figure 5.16 – Slight X-Y Axis Offset Which Could Lead to Inaccurate Results given Current Assumptions

To counteract this potential offset in the *x* and *y* direction acceleration, the accelerometer was meticulously and rigidly placed on the electronics bed using dual lock reusable fasteners as well as Loctite glued mounting to ensure rigidity of orientation. Although this calibration metric was able to be addressed beforehand, the more critical metric of an *x* and *z* direction offset was unable to be addressed and is likely responsible for overestimating the exact results. Like Figure 5.15, an offset in the *x* and *z* direction would also need to be addressed in the data processing stage. The difficulty in this offset is that gravity in the absolute $-\hat{k}$ direction is read by the accelerometer in both the *x*, *y*, and *z*. Ideally, as in Figure 5.11, the *z* acceleration data of the accelerometer will be consistently 9.8 m/s^2 and the *x* and *y* acceleration data will be 0 m/s^2 at

rest. If either of the cases presented in Figure 5.17 show up, then the *x* acceleration data at rest will contain a component of gravity of the form,

$$a_x = g \sin\theta \tag{5.2}$$

where θ is the angle of displacement of the *x* direction of the accelerometer and the vehicles longitudinal direction of motion.

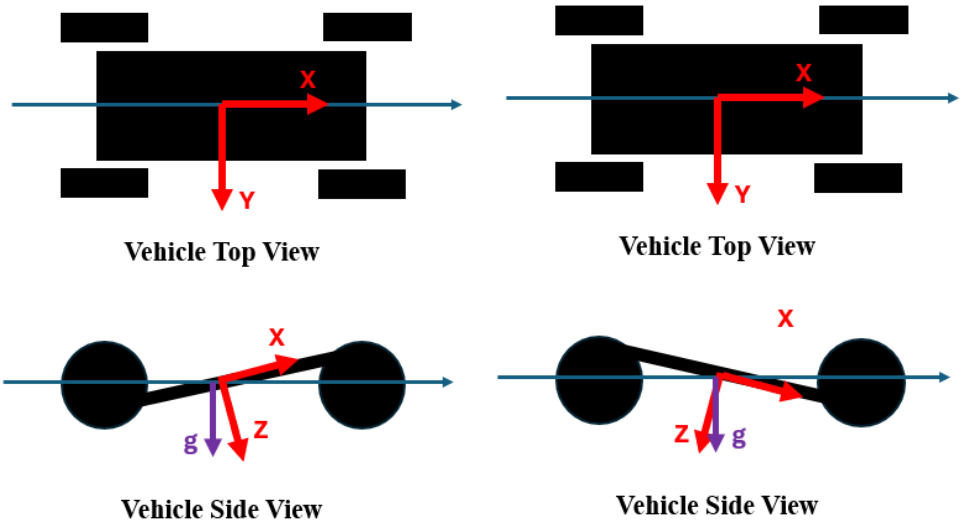


Figure 5.17 - Slight X-Z Axis Offset Demonstrated During the Vehicles Acceleration Test

This behavior is exhibited in the data consistently during periods of rest as seen in Figure 5.18. If the accelerometer's platform were perfectly level, then the data at rest would read 0 m/s^2. The variation in this at rest acceleration of the *x* direction data is most likely due to the suspension of the vehicle which is not calibrated to return to a level position during every rest but to find a steady state equilibrium after each full stop which can vary depending on the vehicle's previous behavior. Ultimately, this gravitational consideration leads to inaccurate results regarding the vehicle's true longitudinal acceleration, velocity, and distance to velocity.

Due to the nature of a vehicle's front bumper tilting up during acceleration, it is expected that the *x* acceleration data included a gravitational component. Although it was documented that

the accelerometer was aligned as shown in Figure 5.11, it is expected that the *x* axis was truly non-constant and teetered between both positions exhibited in Figure 5.17. Thus, this would conclude that the acceleration being documented by the accelerometer included a component of the vehicle's acceleration, previously estimated to be around 4 m/s^2, and a component of the acceleration of gravity denoted by Equation 5.2.

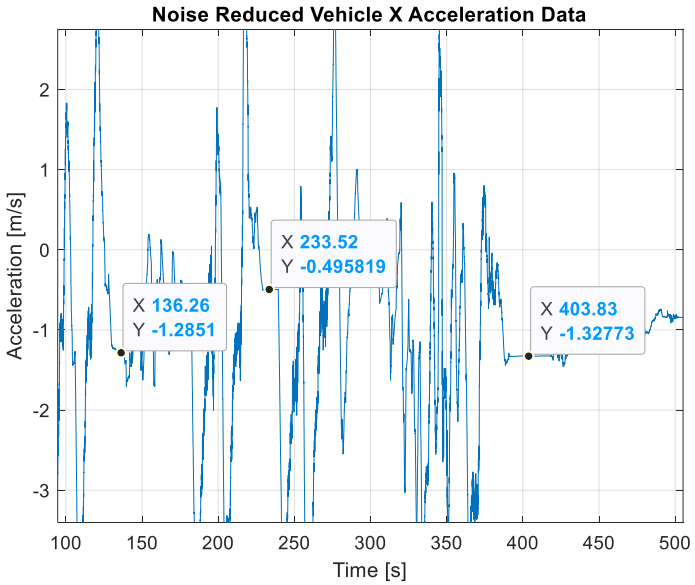


Figure 5.18 - Points of Rest where *x* Acceleration Demonstrated Behaviors of Including Gravitational Acceleration

Future iterations of this test will need to include the integration of additional sensors to enhance the calibration of accelerometer data, thereby improving the accuracy of results. The automotive industry standard for assessing vehicle acceleration uses timed GPS data and front wheel encoder data to obtain precise acceleration and velocity measurements. GPS data can gather distance and time parameters, which are then used in basic kinematic equations to determine velocity between two points. This data can be extrapolated and processed to estimate relative acceleration behaviors. Front wheel encoder data is preferred due to its reduced susceptibility to slip compared to the rear wheel in rear-wheel drive vehicles. The encoder

gathers angular velocity data over time, which can also be extrapolated and processed to derive longitudinal acceleration and velocity information, aiding in determining the distance required to attain maximum velocity. Employing an integrated approach incorporating data from multiple sensors is the most accurate method to calibrate and compute acceleration data for vehicular applications and will be applied in future works. Note that GPS data was not collected because the external accelerometer lacked GPS information.

Assuming the acceleration test yielded relatively accurate results, the on-road distance to maximum speed falls within the range of 90 meters, and thus the off-road course necessitates a comparable distance in relation to the dynamic coefficient of frictions of the different surfaces. It is assumed that for vehicles, the coefficient of friction for pavement and grass are 0.7 and 0.35, respectively. Accounting for the likelihood of a prolonged acceleration period on grass due to friction compared to paved roads, an estimated distance of 120 meters for the off-road straight stretch was considered. Subsequently, the off-road course was adjusted to incorporate a straight stretch of this estimated length, with the course layout meticulously outlined using numerous 3-foot-tall reflective driveway markers, as depicted in Figure 5.19.

Figure 5.19 - Starting Point for the Off-Road Track

Chapter 6

ILC Path Following Simulation

To achieve the goal of optimal path-planning, it is essential to master path-following. This can be achieved by creating an algorithm that models steering as a function of position dictating the motion of the vehicle throughout a traversal. This is then accompanied by a throttle control algorithm to control the speed of the vehicle.

To enable path-following with the actual RC vehicle, the development of a robust path-following algorithm requires careful design. An initial step in this process involves creating a simulation. Simulations offer valuable insights into the vehicle's behavior while following a designated path, enabling the testing and validation of control algorithms before implementing them on the physical vehicle. Additionally, simulations facilitate the fine-tuning of control algorithms, optimizing parameters such as speed, steering angle, and acceleration.

As previously mentioned, two key parameter inputs that heavily influence path following are steering and throttle. Steering as a function of position is a common practice in path following, however throttle control is more difficult to manage due to the factors that arise with high-speed driving such as robust maneuvers through sharp turns as well as look ahead control for the vehicle to make rapid decisions. For this, the vehicle’s velocity profile was simulated using data from previous chapters’ tests, coupled with iterative learning control methods of artificial intelligence.

Iterative learning control is implemented to determine the optimal speed of the vehicle based on the speed from previous iterations. Using iterative learning control, the vehicle uses different stations determined by the steering algorithm. This enables the determination for the

maximum achievable at various milestones based on previous iterations. This ultimately is the theory being applied to optimize the vehicle's velocity throughout any traversal.

Path-following Kinematics

For an effective path-following simulation, precise modeling of the vehicle's kinematics is essential. The RC vehicle employed in this study is characterized as a car-like mobile robot, representing a four-wheeled robot with front steerable wheels, resembling a typical car configuration. Car-like mobile robots, prevalent in ground robotics, are commonly conceptualized using a two-wheel model in their kinematic analysis. This model features a fixed rear wheel on the body, non-rotating concerning the base frame, while the plane of the front wheel rotates about the vertical axis for steering. The two-wheel model, like a bicycle model without sideways slip, assumes the wheels roll without lateral movement. It is imperative to note that when accounting for sideways slip, the model commonly used in general vehicle literature is termed the bicycle model. However, as slip is omitted in this analysis, this chapter intentionally refers to the vehicle as a two-wheel kinematic model.

Figure 6.1 illustrates the two-wheel kinematic model of a car-like mobile robot, depicting the assigned coordinate system for both the base frame and the vehicle. Additionally, the figure presents the instantaneous center of rotation, the point where the wheels cannot move. This center of rotation serves as a tool to determine the turning radius of the vehicle, influenced by the wheelbase, denoted as L, and the steering angle, represented as γ.

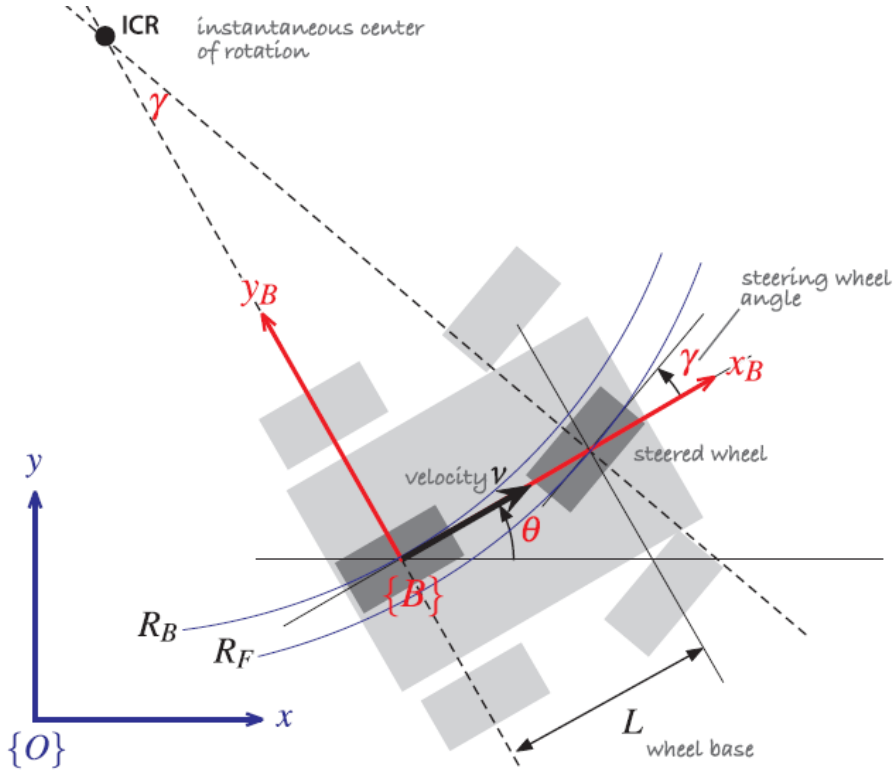


Figure 6.1 - Two-Wheel Model Vehicle Kinematics

The kinematic equations of motion about the two-wheel kinematic model are as follows:

$$\dot{x} = v\cos\theta \tag{6.1}$$

$$\dot{y} = v\sin\theta \tag{6.2}$$

$$\dot{\theta} = \frac{v}{L}\tan\gamma \tag{6.3}$$

Where $\dot{x}$ and $\dot{y}$ are relative velocities in the global base frame as measured from the center of the rear axle. $\dot{\theta}$ can be defined as the change of heading or change in relative steer angle.

The vehicle will typically have to traverse a trajectory of points in the *xy* plane. A viable strategy for trajectory tracking involves employing a pure pursuit steering and throttle controller. In this algorithm, the initial step is designating a specific goal point in the *xy* plane, denoted as (*x**, *y**), which serves as the target toward which the vehicle will navigate. The control

mechanism dictates that the robot's velocity is modulated proportionally to its distance from the designated goal point:

$$v^* = K_v\sqrt{(x^* - x)^2 + (y^* - y)^2} \tag{6.4}$$

The parameter K_v, denoted as the velocity gain, assumes a crucial role in the control algorithm. In control systems, a gain serves as a parameter for amplifying or attenuating an input signal, representing the ratio of the control system's output to the input signal. Subsequently, the algorithm governs the robot's heading and steering angle to facilitate alignment with the specified goal point, guided by the following control law. It is noteworthy that the heading gain, denoted as K_h, enables fine-tuning of the steering action. The desired heading is denoted as θ^*, and the corresponding steering angle is represented by γ:

$$\theta^* = \tan^{-1}\frac{y^* - y}{x^* - x} \tag{6.5}$$

$$\gamma = K_h(\theta^* - \theta) \tag{6.6}$$

Which denotes that that the steering input can is determined by the desired heading angle. In essence, the algorithms force the vehicle to face the next corresponding point at a given position.

Once the foundational control laws for point-to-point motion have been defined, there arises an opportunity for tuning to enhance trajectory or path-following. In addressing the trajectory following challenge, the vehicle's motion towards a point persists, albeit with the pursuit point dynamically updated over time. The maintained distance between the robot and the pursuit point is denoted as d^*. Leveraging this distance parameter, an error function, e, can be formulated to further characterize the system's performance:

$$e = \sqrt{(x^* - x)^2 + (y^* - y)^2} - d^* \tag{6.7}$$

Where this error function can be used to the iterative-learning control algorithm to determine the robot's velocity. Iterative learning control determines the next velocity to be had rather than the

current velocity through the traversal. Typically, an ILC algorithm will have an assumed initial velocity which will be altered by adding the product of the learning rate, α, and the error function:

$$v_{k+1} = v_k + \alpha * e \tag{6.8}$$

Where v_k is the velocity of the current iteration. The iterative learning control algorithm is discussed further in subsequent sections.

Iterative Learning Control

As discussed in Chapter 2, Iterative Learning Control is a control methodology widely applied in mechanical engineering in contexts where precise repetitive tasks are accomplished. At its core, ILC leverages past performance data to refine control inputs iteratively, aiming to enhance system performance with each iteration. The advantage of ILC lies in its ability to improve tracking accuracy over time, as it learns from previous errors and adjusts control signals leading to enhanced trajectory tracking and reduced tracking errors [19], [30]. Moreover, ILC often requires minimal additional hardware, making it a cost-effective solution in many applications. However, its effectiveness heavily relies on accurate modeling of the system dynamics and assumptions of repeatability in tasks, which can be challenging to achieve in complex real-world scenarios such as autonomous driving. Additionally, the convergence of ILC algorithms can be slow, demanding careful tuning and validation processes.

The parameter for which ILC representation is achieved is the vehicle's velocity profile throughout the traversal. In theory, the vehicle will learn from previous iterations to enhance its velocity at specific waypoints defined by the traversal itself. Using a base design of velocity as a

function of curvature and station, the vehicle profile will ultimately converge to an ideal steady state velocity profile that will be as fast as possible. This ILC model of velocity will essentially teach the vehicle to go fast through waypoints of low curvature and to go slow through waypoints of high curvature until the vehicle can repeatedly drive the traversal with essentially zero positional error and maximum speed.

Iterative learning control is most often associated with a time-based repetition model, where every iteration of an event ends in a fixed time of a duration such that the converging output is a function of a constant time. Due to the nature of this project, having a fixed time representation of the velocity profile would be counterintuitive since the goal is to minimize the time the vehicle traverses the path through a maximized velocity profile. Thus, the velocity ILC model will follow a fixed station representation for the velocity profile. The repeated constant is the station of the track. Again, note that station is not to be considered the length the vehicle traverses. The difference between station and path length is that path length can vary slightly depending on track limits and idealization, whereas station is a constant that represents segmented zones for the vehicle to pass through during a traversal.

To incorporate ILC, a simulation model was created in MATLAB which practices path following given an input traversal as well as vehicle parameters, controller parameters, and field traversal parameters. Initial conditions of the states are defined as longitudinal velocity, lateral velocity, yaw rate, angular velocity of the wheel, and relative initial position in east, north, and heading. The simulation was created using tools from Penn State Intelligent Vehicles and Systems Group GitHub libraries for Geometry, Path, and Vehicle Dynamics. Such MATLAB simulation code can be found in Appendix D.

Prior to creating running simulations regarding the subject vehicle of this research, the simulation created was validated and tuned using a proven large-scale vehicle model. The large-scale vehicle model is loosely based on the C5 Corvette with simplified parameters which allow for relatively easy tuning and simulation understanding. Table 6.1 details the differences between the large-scale vehicle model parameters compared to those offered by the subject vehicle of this research. All data regarding the subject vehicle was collected through general calculation models based on measurements from the actual subject vehicle as well as calculations based off the large-scale vehicle.

Table 6.1 - Large Scale Vehicle Compared to Subject Vehicle Parameters

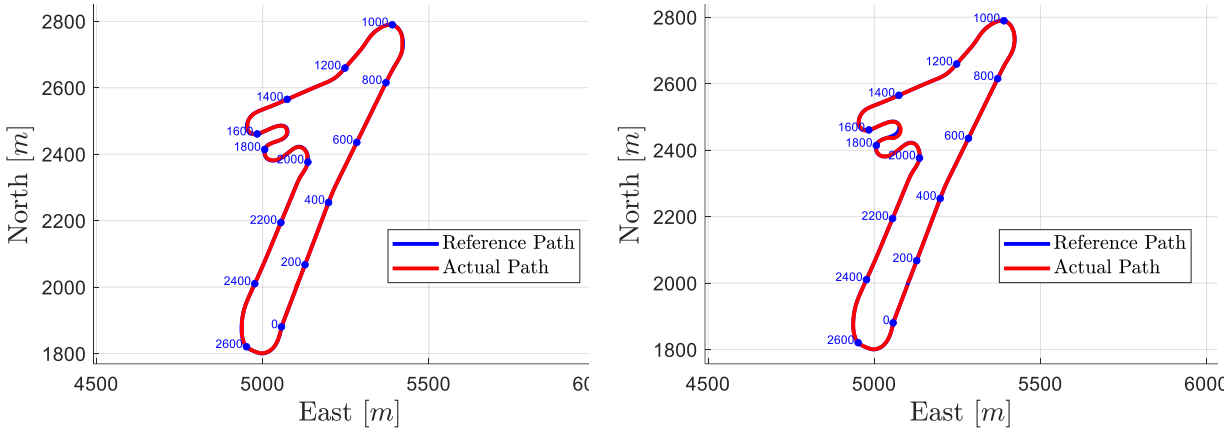
Parameter	*Large Vehicle*	*Subject Vehicle*	*Calculation*	
Mass [kg]	1,600	14.719	m	
Mass Moment of Inertia [kg m^2]	2,500	1.20761	$I_{zz} = m \cdot a \cdot b$	(6.9)
Wheel Mass Moment of Inertia [kg m^2]	1.2	1.2*10^{-3}	$I_w = I_{zz} \cdot 10^{-3}$	(6.10)
Effective Radius of Wheel [m]	0.32	0.09	r	
Center of Gravity to Front Axle [m]	1.3	0.26276	a	
Center of Gravity to Rear Axle [m]	1.3	0.31224	b	
Wheelbase [m]	2.6	0.575	$w_b = a + b$	(6.11)
Trackwidth [m]	1.5	0.44	w_t	
Height of Center of Gravity [m]	0.42	0.1	h_{cg}	
Wheel Corner Stiffness Front [-]	95,000	873.94	$C_{a,f}$	
Wheel Corner Stiffness Rear [-]	110,000	1011.9	$C_{a,r}$	
Longitudinal Stiffness [-]	65,000	597.96	C_b	
Contact Patch Length [m]	0.15	0.03	l_{cp}	

Large Scale Vehicle Simulation and Tuning

The large-scale vehicle simulation tuning validation test was conducted to ensure that the simulation conformed with expected performance criteria, allowing accurate plotting of anticipated behaviors. Additionally, the large-scale vehicle simulation provided a baseline for analyzing multiple input parameters and their consequent impacts on diverse output variables like slip angle, lateral tire force, lateral acceleration, and tracking error.

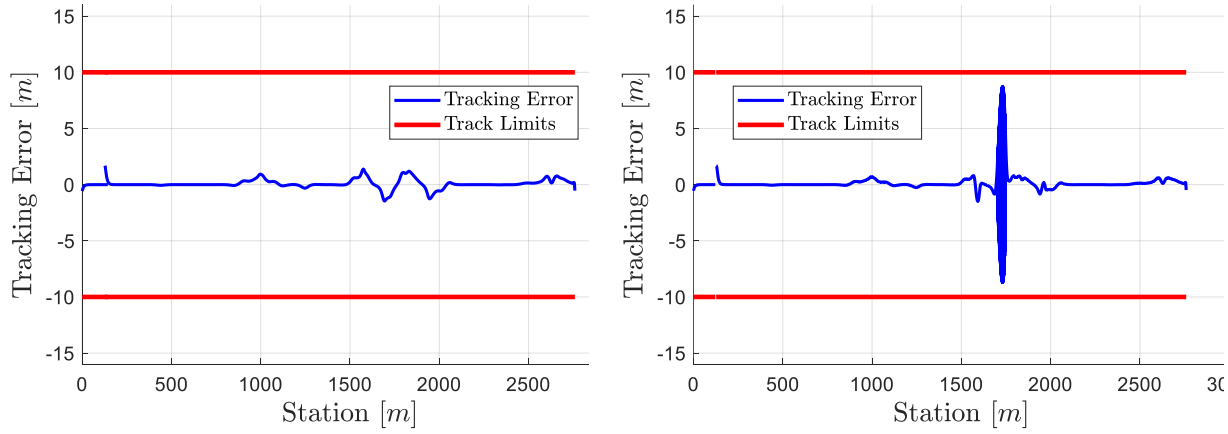
In the initial phase of simulation studies, the large-scale vehicle was programmed to follow a scaled version of the designated reference path outlined in Chapter 5, maintaining a constant velocity. This reference path, scaled to ten times the real-life track dimensions, ensured the feasibility of traversing the path with the larger vehicle model. The simulation included speeds of 10 m/s, 13 m/s, and 15 m/s to assess the effectiveness of path following. At 10 m/s, the large-scale vehicle autonomously traversed the path with zero tracking error or deviation. Conversely, at 13 m/s, the vehicle encountered significantly more tracking errors and deviations,

although it managed to complete most of the track. These observations are depicted in Figures 6.2 and 6.3.



a) b)

Figure 6.2 - Large Vehicle Model Simulation Path Success a) 10 m/s b) 13 m/s.



a) b)

Figure 6.3 - Large Vehicle Model Simulation Tracking Error a) 10 m/s b) 13 m/s.

Note that the reference path is annotated with corresponding stations along selected intervals. The switchback segment of the track garners interest due to its sharp curvature, leading

to challenges in path following at higher velocities. This segment, spanning stations 1500 to 2000, is designated as the switchback zone.

Note that the tracking error is related to the station closest associated to the vehicle during its traversal. Consequently, in regions characterized by significant tracking error, the vehicle may align with a different station than the anticipated next one, as evidenced by the thickened points in the case of 13 m/s which span across the x-axis. As anticipated, the areas exhibiting the highest tracking error and notable deviations in the plot lie within stations 1500 to 2000, previously identified as the switchback zone.

Further analysis was conducted at the constant speed of 15 m/s due to it demonstrating exceptional success in most of the track but exhibited behaviors during the switchback zone which are worth observing and identifying. Figure 6.4 illustrates the path following success at the constant speed of 15 m/s.

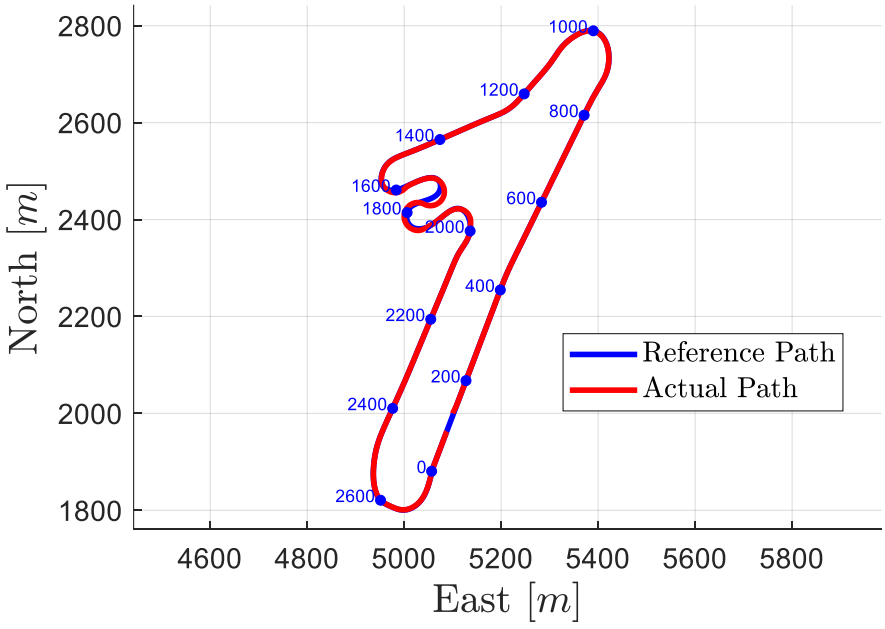


Figure 6.4 - Large Vehicle Model Simulation at 15 m/s around the Scaled Track

While the vehicle successfully completed the traversal, further post-processing and analysis of outputs reveal critical behaviors endured by the vehicle during the maneuver. These behaviors must be carefully considered by the ILC algorithm to ensure an optimal path traversal without adverse consequences.

One such adverse effect observed is complete tire slippage. Lateral tire force is a metric which determines the force the wheel is acting on in the lateral direction. Analysis of lateral tire force plots reveals instances where the vehicle's tires reach a plateaued maximum at various points along the switchback zone. The presence of these flat maximums, depicted in Figure 6.5, indicates a loss of traction, with the wheels potentially sliding sideways instead of solely moving along the vehicle's longitudinal axis. This subtle drift signals that the vehicle is navigating the turn at excessive speed, nearing the threshold of losing control.

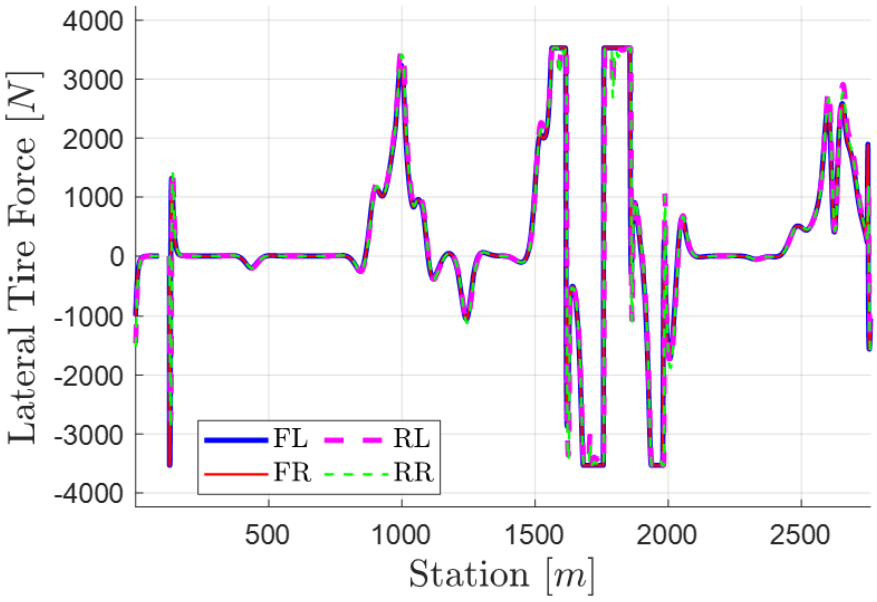


Figure 6.5 - Lateral Tire Force vs. Station Plot Exhibiting Sideways Slip

An investigation of the lateral acceleration thus exhibits a similar trend where maximum slip force is found. Figure 6.6 illustrates the lateral acceleration as well as lateral acceleration

limits defined by the friction coefficient of the track. In this case, lateral acceleration is not to exceed ± 8.83 m/s^2. During moments of high lateral force, it's observed that the lateral acceleration is nearing the limit, yet not exceeding it. Surpassing the acceleration limit could lead to severe skidding, increased risk of rollover, understeering, or oversteering depending on the tires that exhibit the behavior the most.

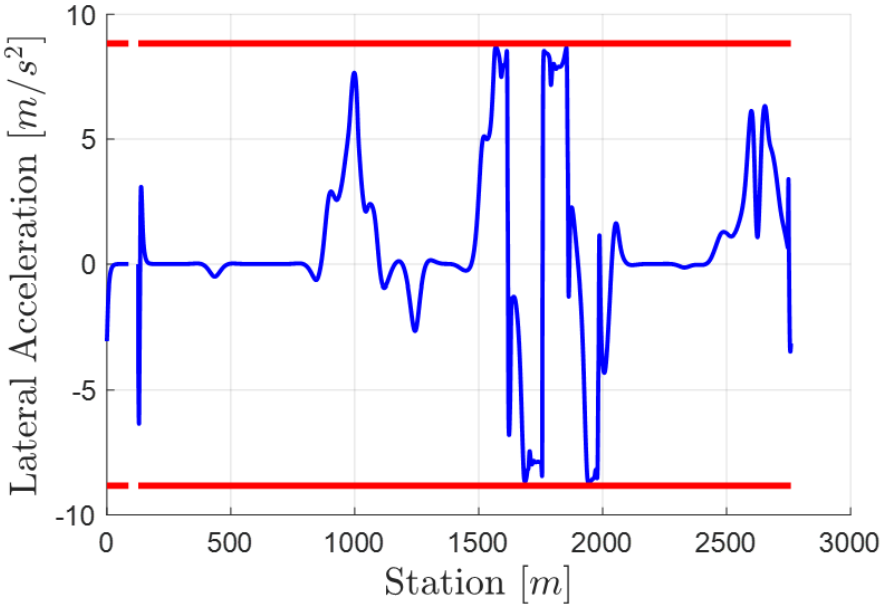


Figure 6.6 - Lateral Acceleration vs Station for 15 m/s Simulation.

This requires an examination of the slip angle for each tire, which quantifies the degree of sliding or drifting in radians. It is expected that the slip angle will exhibit an inverse relationship with the lateral tire force, a correlation that is verified and illustrated in Figure 6.7. The analysis reveals that the front tires undergo significant slippage, while the rear tires exhibit less, which yields more stability. This difference in behavior is expected, as the front wheels pivot in response to steering inputs, rendering them more susceptible to increased levels of slip. Slip in this case is assumed to be a suboptimal behavior which should be minimized to ensure smooth driving behaviors. Slip can sometimes be considered an optimal behavior in the form of

controlled drifting; however, this behavior is beyond the scope of this work and can be considered in future research.

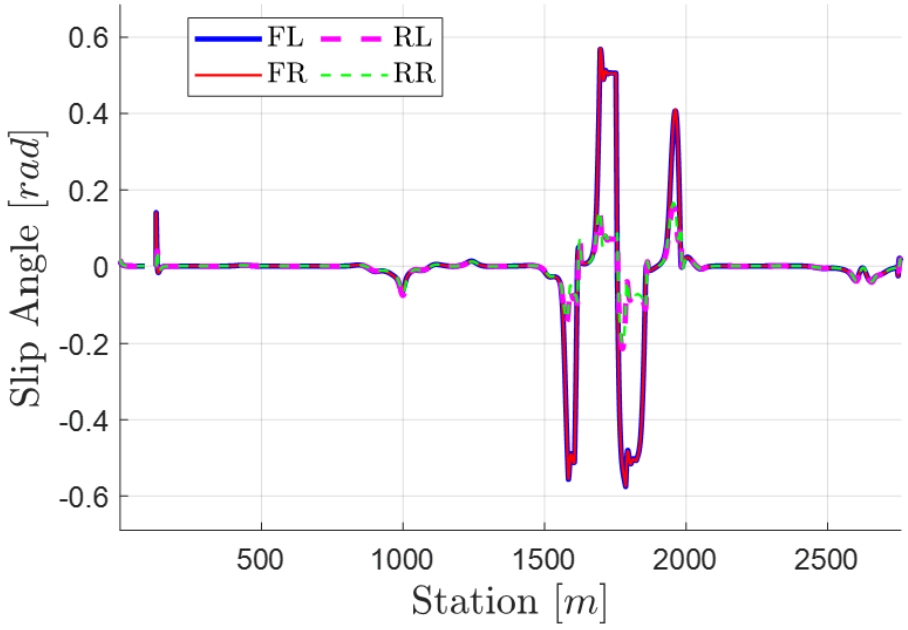
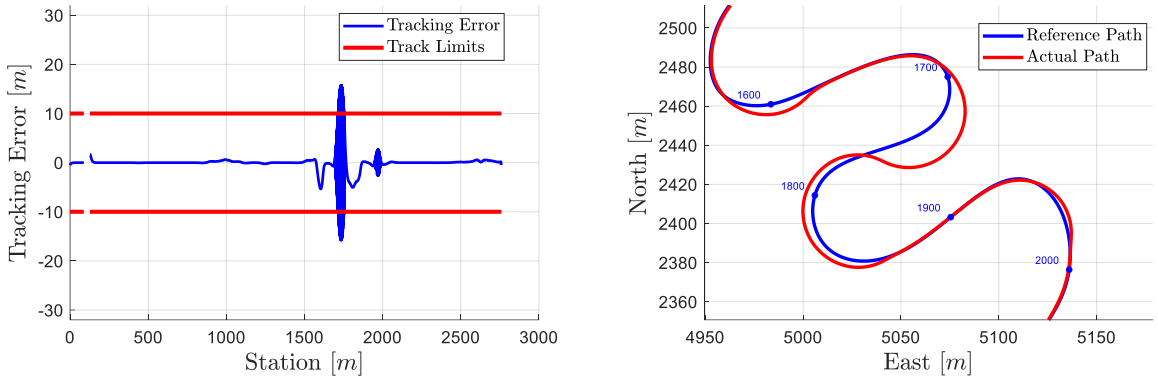


Figure 6.7 - Slip Angle vs Station for 15 m/s Simulation.

Lastly, considering the behaviors exhibited in the lateral direction of the wheels, it is crucial to assess their impact on the vehicle's ability to stay on track. Figure 6.8a provides an overview of the vehicle's tracking error in comparison to both the optimal path line—where tracking error equals zero—and the track limits, defined as points situated 10 meters away from the optimal path. As expected, the tracking error is most pronounced within the switchback zone, with several station points indicating a failure to remain within the track limits. This is particularly evident in a close-up examination of station 1700, where the vehicle significantly exceeds the track limits. As previously mentioned, a thick spot forms as the calculations yield the error on different sides of similar stations due to the proximity of stations. Such deviations from the optimal path represent suboptimal behaviors that must be mitigated via ILC to achieve optimal path following.



a) b)

Figure 6.8 - a) Tracking Error Plot, b) Switchback Zone Closeup

Comparing the plots obtained at 15 m/s with those at 10 m/s and 13 m/s yields valuable insights into optimal and suboptimal vehicle behaviors across all station points, particularly emphasizing segments where suboptimal behaviors are prevalent, such as the switchback zone. Ultimately, this comparative analysis sheds light on the various factors to be mindful of and their subsequent effects. This aids in refining the tuning process for the subject vehicle simulations, building upon the insights gained from the analyses described above.

Moreover, the large-vehicle simulation serves as a crucial tool for result validation. The simulations with the large-vehicle model effectively showcase expected behaviors, supported by reasoned explanations. They demonstrate the simulation's capability to achieve path following at constant speeds, laying a foundation for the inclusion, and tuning of more advanced path following scenarios such as variable speed cases and ultimately ILC implementation.

1/5th Scale Vehicle Simulation Validation

Following the validation and tuning of the large-scale vehicle simulation, it is imperative to confirm and validate the capabilities of the simulation when applied to a small-scale vehicle model. The performance of the small-scale vehicle simulation serves as a critical validation step to ensure the reliability and accuracy of the model across varying scales and conditions.

Unlike the initial phase of the large-scale vehicle simulation, the small-scale vehicle model is programmed to traverse the designated reference path as designed in normal dimensions, while maintaining a constant velocity. To compare with the results of the large-scale model simulation, similar parameters were investigated such as slip angle, lateral tire force, lateral acceleration, and tracking error. This comparative analysis provides insights into the scalability and robustness of the simulation across diverse operating conditions. The analysis also confirms the simulations accuracy and consistency.

Ultimately, the validation of the small-scale vehicle simulation is imperative for confirming the simulation's capability to accurately emulate real-world behaviors for the vehicle model being researched. Table 6.1 from earlier in the chapter illustrates the changes made to the simulation parameters to go from large-scale model to the $1/5^{th}$ scale vehicle model in simulation. After initial constant velocity tests, the vehicle successfully traversed the course at 2 m/s (or roughly 5 mph) with minimal tracking error as seen in Figure 6.9.

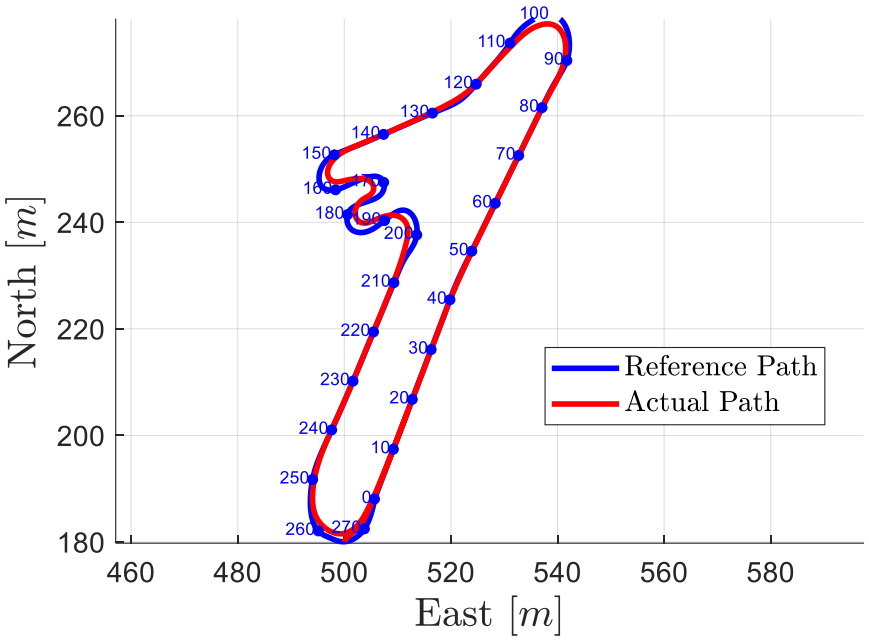


Figure 6.9 - 1/5th Scale Vehicle Simulation Result

This relatively low constant velocity makes sense due to the intense curvature of the switchback zone. The speed of 2 m/s is equivalent to the speed of a light jog, and from traversing the track by foot it is observed that the switchbacks at the speed of a light jog would be extremely difficult to accomplish consistently, thus the result can be inferred as possibly accurate. At the speed of 2 m/s, the tracking error comfortably stayed within the designated track limits as seen in Figure 6.10. The vehicle did experience some tracking errors at turns and in the switchback zone, which is expected. Note that switchback zone is denoted as stations 150 to 200.

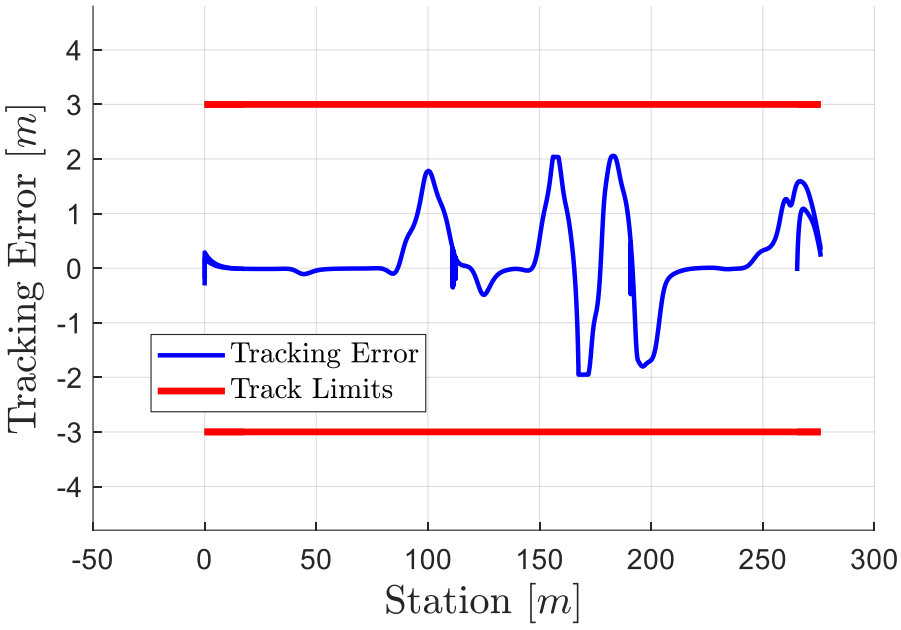


Figure 6.10 – Tracking Error vs Station for 1/5th Scale Vehicle Simulation

Comparatively, the lateral tire force acceleration and lateral acceleration were evaluated to ensure consistency with the large-scale vehicle model. Unlike the large-scale vehicle model, the rear tires were more susceptible to enduring large lateral tire force, as seen in Figure 6.11, however the general points of tire force shape look to be roughly the same with emphasis at the same stations. The rear tire force difference is observed when manually driving the Losi DBXL-E 2.0 Buggy in which the rear tires are noticeably more susceptible to slipping than the front tires. Similarly, lateral acceleration results corresponded with the outcomes derived from the large-scale vehicle model as seen in Figure 6.12

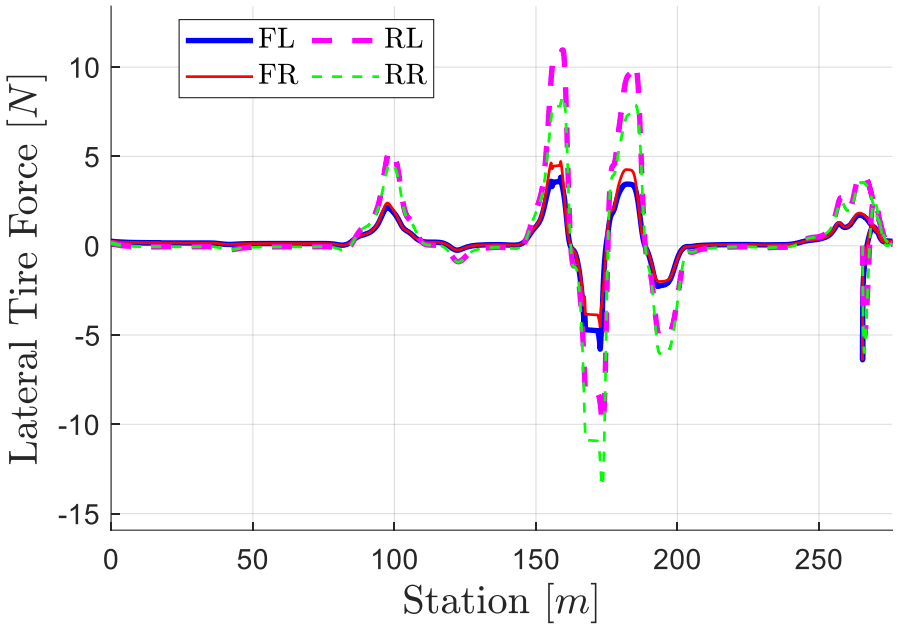


Figure 6.11 - Lateral Tire Force vs Station for 1/5th Scale Vehicle Simulation

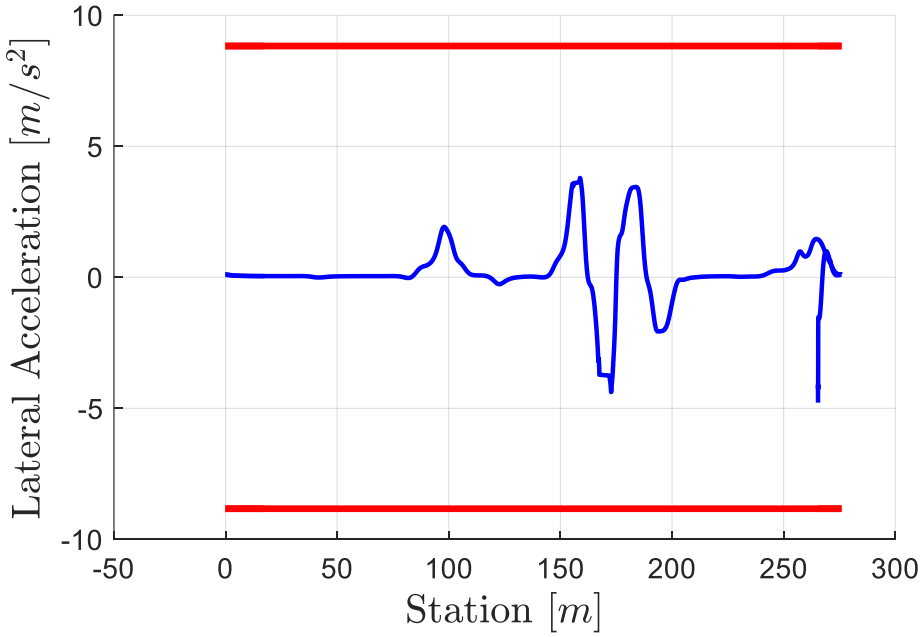


Figure 6.12 - Tracking Error for 1/5th Scale Vehicle

Once the constant speed velocity profile was compared for both large-scale and 1/5th scale vehicle models, the variable velocity phase of testing was initiated. Initially, variable speed was defined through a series of if statements which changed the velocity depending on the

previous station the vehicle has passed. Table 6.2 demonstrates the various segments of the track defined by their stations which were given different velocities.

Table 6.2 - Track Segment Zones and Their Corresponding Velocities

Zone Name	*Station Range*	*Relative Velocity*
Straightaway Zone	0-90	VERY HIGH
Hairpin One Zone	90-110	LOW
High Speed Corner	110-150	HIGH
Switchback Zone	150-200	VERY LOW
Acceleration Zone	200-250	HIGH
Hairpin Two Zone	250-End	LOW

Various velocity profiles were tested using Table 6.2's relative velocity using a hard coded estimation method. At a maximum velocity of 8 m/s during the straightaway zone, and minimum velocity of 2 m/s during the switchback zone, the vehicle successfully traversed the track with a variable velocity profile as seen in Figure 6.13. Due to the simulation being determined through a designated time, it can be observed the vehicle successfully completed a lap and was able to traverse at least half of a second lap due to overlay in the plotting data.

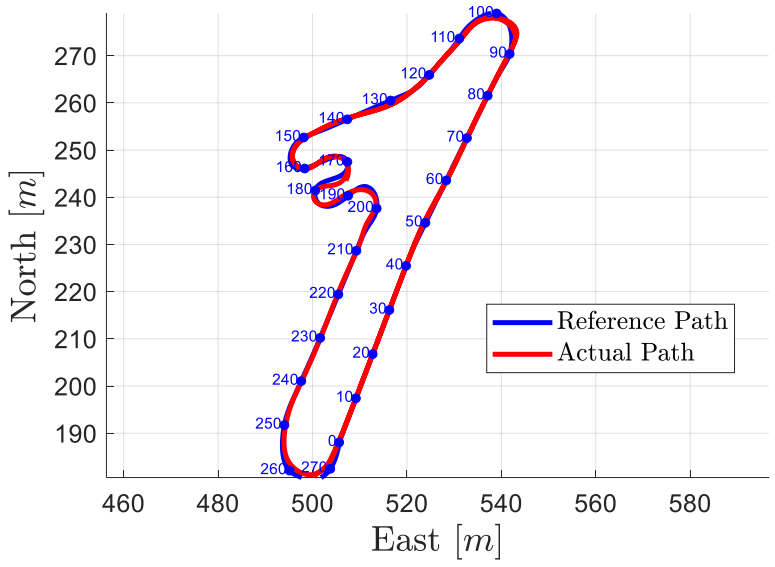


Figure 6.13 - 1/5th Scale Variable Velocity Profile Path Completion

It must be noted this variable velocity profile only succeeds at time steps greater than 0.1 seconds which is not a direct indication that this profile would succeed in real time. When tested at a more finite time step of 0.01 seconds, the vehicle experiences a serious slip in the straightaway zone and spins out of control in the form of a donut as seen in Figure 6.14. Thus, this indicates that in real-world situations, the vehicle is more likely to spin out given this velocity profile rather than not.

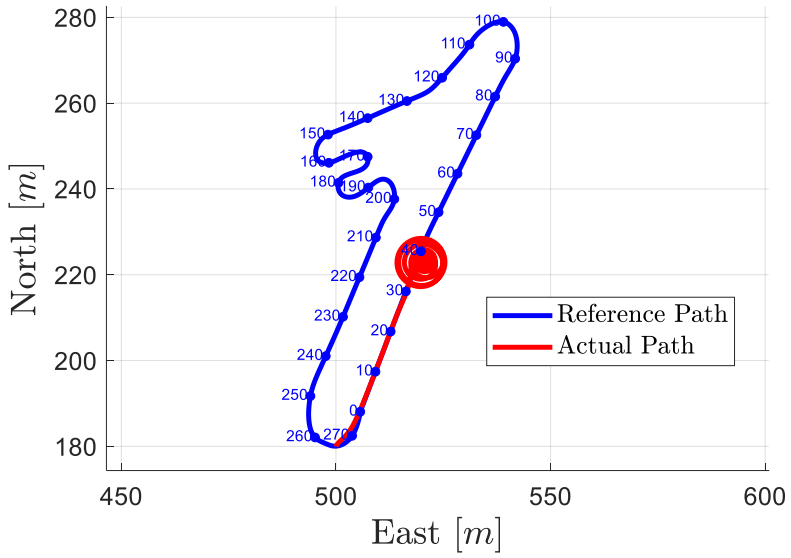


Figure 6.14 - 1/5th Scale Variable Velocity Profile Path Failure

Despite having the same velocity profile, the time step yields vastly different results. This is due to the simulation solver discretization which determines how often the steering is updated, and thus how aggressively the vehicle maneuvers in response to a change in the vehicle position. A simplified explanation of this phenomenon is like driving with the goal of evading potholes: at a fast time-step, a driver constantly adjusts maneuvers to achieve this goal, resulting in heightened sensitivity and very "jerky" vehicle behavior. Conversely, at a slower time step, the driver would drive blindly briefly prior to initiating new maneuvers, leading to a smoother trajectory compared to the more dynamically adjusted path. Ultimately, the increased time step acts as a low-pass filter for the vehicle maneuvers allowing for a smoother outcome. For this reason, the 'successful' plot is not further examined since it is likely not very accurate to a real-world outcome. These results do suggest that the concept of variable velocity profiles could have some degree of success, particularly in the use of ILC to tune and adjust the velocity profile.

Chapter 7

Conclusions

Ultimately, this thesis represents a comprehensive exploration and research into advancing high-speed off-road autonomous vehicle technology specifically in the realm of path planning. True high-speed off-road path planning requires three primary development components: First, the proper development of a vehicle capable of driving fast with capable maneuverability; second, the development of a hardware package with all necessary sensors properly mounted to the vehicle; and third, the development of advanced software algorithms which collect data intelligently and translate that information into intelligent path following subsequently path planning.

The Losi DBXL-E 2.0 RC Buggy used in this study is a vehicle renowned for its capacity to achieve high velocities and maneuverability through challenging off-road terrain. By implementing modifications and enhancements of the vehicle's hardware and mounts – specifically by compacting the electronics packaging with a printed circuit board, developing metal reinforced rear encoder mounts, and integrating polar electronic connectors for all connecting pieces – both reliability and usability were significantly improved, providing a foundation for ongoing research.

The acquisition and analysis of acceleration data provided invaluable insights into the dynamic behavior of the vehicle, facilitating the refinement of control algorithms and the optimization of the overall system performance.

Moreover, the design and creation of a new test-track, along with the processing of GPS data to generate a highly refined and accurate reference traversal, encompassing complicated geographic features and segments of high velocity and maneuverability offer a novel approach to

simulation and path following. This remarkable achievement contributes to the development of autonomous vehicle navigation systems and opens avenues for testing and validation in controlled environments.

Furthermore, the creation and validation of a reliable and complex simulation model on MATLAB using a large-scale vehicle model, and later validated with the $1/5^{th}$ scale vehicle model, represents an advancement in predicting the adaptability and robustness of autonomous systems in path-following. This simulation model characterized by simple input requirements – including a reference traversal, vehicle parameters, steering control parameters, and terrain parameters – yields intuitive data regarding the vehicle's states, associated vehicular forces on all tires, and even complex slip calculation data exhibited during high-speed turns.

The simulation model also enables the fine tuning of parameters prior to field testing and facilitating the establishment of a baseline understanding of potential outcomes that the vehicle might encounter across various segments of the track. Ultimately, the simulation serves as a critical tool for refining control algorithms and optimizing system performance. It also provides a relative baseline regarding outcome of the iterative learning control velocity model.

Overall, this research emphasizes the variety of interdisciplinary components essential for advancing innovation of high-speed off-road autonomous vehicles. Despite encountering setbacks, such as a high-speed collision that delayed vehicle testing, this study expands the boundaries of knowledge within both the vehicle under this study and the broader field of high-speed off-road autonomous vehicles.

Future Work

Significant tasks remain to fully integrate autonomous off-road path-following, and eventually path planning, into the vehicle. Further development of the simulation model needs to be accomplished to provide a more refined outcome regarding the small-scale vehicle itself rather than the large-scale vehicle. Predicted velocity profiles need and associated algorithms, such as velocity as a function of curvature, need to be further developed and linked to associated station structures in MATLAB for simulation.

The implementation of suitable internal GPS conversion from LLA to ENU along with steering angle mapping and control code onto the onboard Teensy is underway. This would allow for fundamental physical path-following tests such as following a straight line, following a curved line, then following an internalized path using an ENU-based coordinate system. Upon successful completion of these initial tests, the Teensy code should be enhanced to facilitate path-following.

The subsequent phase involves integrating a control block into the steering control within the path-following model. After tuning and rigorous testing, the variable velocities will be determined via real time iterative learning control such as the simulation predicts and subsequently incorporated into the path-following Teensy code. Following the successful implementation of autonomous steering and throttle control during on-road testing, time trials will be systematically evaluated. Upon the achievement of successful high-speed on-road path-following, a similar process will be completed at the off-road test track to enable high-speed off-road path-following.

The future for the vehicle encompasses further understanding of the vehicle itself. A full-fledged model of the vehicle's internal dynamic system is to be created and documented as well

as a full CAD model of the vehicle. This includes determining more accurate values for parameters such as acceleration, velocity, wheel cornering stiffness, longitudinal stiffness, and friction coefficient factors. This will allow for more accurate simulation results as well as design modification facilitation in the case of additional sensors or modifications.

Another plan for the vehicle includes the incorporation of off-road path-planning facilitated by an onboard computing system, such as a Raspberry Pi or NVIDIA Jetson Nano. To enable path-planning, additional sensors like LiDAR, cameras, IMU, etc., must be integrated to enable obstacle detection and avoidance.

Appendix A

RC Vehicle Datalogging Code

```
#include <SD.h>

/* IVSG RC Test Vehicle Datalogging V3.0

   Last Updated: 3/23/2022

   Microcontroller: Teensy 4.1

   For additional documentation, see "Vehicle Embedded Software" page of
Hardware_RCVehicle wiki

   (https://github.com/ivsg-psu/Hardware_RCvehicle/wiki/Vehicle-Embedded
Software#rc_vehicle_datalogging_v3)

   Required libraries:

   - Wire

   - SD

   - SparkFun_u-blox_GNSS_Arduino_Library (https://github.com/sparkfun/SparkFun_u-
blox_GNSS_Arduino_Library)

   Notes:

   - This program assumes the encoders only rotate in one direction to optimize performance.
Backwards rotation will be recorded as forward.

*/


// Libraries

#include <Wire.h> //For I2C to GNSS

#include <SD.h> //For writing to sd card
```

```
#include <SparkFun_u-blox_GNSS_Arduino_Library.h> //For interfacing with ZED-F9P

SFE_UBLOX_GNSS myGNSS; // Create GNSS object

File myFile; // Create file for reading/writing data
const int chipSelect = BUILTIN_SDCARD; // sets the SD card reader to be the onboard one

// Reduce defaultMaxWait for GNSS library operations to 250 from the default 1100. (This
program only uses I2C communication with the GNSS receiver,
// which does not require waits longer than 250 ms).
#define defaultMaxWait 250

// Define pins
#define Encoder1CHA 4 // Teensy pin connected to CHA pin of encoder 1
#define Encoder1CHB 12 // Teensy pin connected to CHB pin of encoder 1
#define Encoder2CHA 5
#define Encoder2CHB 9
#define Encoder3CHA 6
#define Encoder3CHB 10
#define Encoder4CHA 41
#define Encoder4CHB 40
#define ppsInterrupt 36 // Teensy pin connected to the PPS output of ZED-F9P GNSS receiver
#define buttonPin 29  // Connected to a button for starting and stopping datalogging
```

```
#define TeensyLED 13 // Pin for built in Teensy LED.

// Declare variables

// Master Count - updates with every encoder interrupt
volatile long encoder1MasterCount = 0;
volatile long encoder2MasterCount = 0;
volatile long encoder3MasterCount = 0;
volatile long encoder4MasterCount = 0;

// Polled Count - Updates once every GPS pulse
long polledEncoder1Count = 0;
long polledEncoder2Count = 0;
long polledEncoder3Count = 0;
long polledEncoder4Count = 0;

// Polled GNSS Variables
long latitude = 0;
long longitude = 0;
long alt = 0;
byte SIV = 0;
int gpsHour = 0;
int gpsMin = 0;
```

```
int gpsSec = 0;
int gpsMs = 0;
int gpsNs = 0;
int northVel = 0;
int eastVel = 0;
int downVel = 0;
int verticalAcc = 0;;
int horizontalAcc = 0;
int speedAcc = 0;
int headAcc = 0;

String dataString; // Stores next line to be written to file

// Flags
boolean doSave = false;
volatile boolean newPulse = false;
boolean buttonCheck = false;

// Other vars
byte RTK = 0;

// Time values
long startTime = 0;
```

```
long thisTime = 0;
long printTime = 0;
long lastButtonCheckTime = 0;
volatile long gpsPulseCount = 0;
volatile long interruptTeensyTime = 0;

void setup() {

  // Set the pins as inputs
  pinMode(Encoder1CHA, INPUT);
  pinMode(Encoder1CHB, INPUT);
  pinMode(Encoder2CHA, INPUT);
  pinMode(Encoder2CHB, INPUT);
  pinMode(Encoder3CHA, INPUT);
  pinMode(Encoder3CHB, INPUT);
  pinMode(Encoder4CHA, INPUT);
  pinMode(Encoder4CHB, INPUT);
  pinMode(ppsInterrupt, INPUT);
  pinMode(buttonPin, INPUT);

  // Set TeensyLED pin as output
  pinMode(TeensyLED, OUTPUT);
```

```
  // Note that Serial.begin() is not necessary for Teensy 4.1 code


  // Wait 1 second
  delay(1000);
  // see if the card is present and can be initialized:
  if (!SD.begin(chipSelect)) {


    // if the card is not present, freeze
    while (1);
  }


  myFile = SD.open("datalog.txt", FILE_WRITE); // to write to SD card need name, file will be called datalog.txt.


  if (myFile) { // Check that the file opened correctly
    myFile.println("----RC_vehicle_datalogging_v3 - 4 Encoders - GPS - SD Card Storage - Last Updated 3-23-22 ----");
    myFile.println("Time (ms), GPS Pulse Count, Encoder 1 Counts, Encoder 2 Counts, Encoder 3 Counts, Encoder 4 Counts, GPS Hour, GPS Min, GPS Sec, GPS Millisecond, GPS Nanosecond, Latitude, Longitude, Altitude, North Velocity, East Velocity, Down Velocity, Vertical Acc, Horizontal Acc, Speed Acc, Head Acc, SIV");
  }
  Wire.begin(); //Begin I2C comunication
```

```
  Wire.setClock(400000); // Increase I2C clock speed to 400kHz

  if (myGNSS.begin() == false) //Connect to the u-blox module using Wire port
  {
    //If connection to u-blox module failed, print debug message to file:
    myFile.println("u-blox GNSS not detected at default I2C address. Please check wiring.
Freezing.");
    while (1);
  }
  myGNSS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn off
NMEA noise)
  myFile.println();

  // Create storage for the time pulse parameters
  UBX_CFG_TP5_data_t timePulseParameters;
  // Get the time pulse parameters
  if (myGNSS.getTimePulseParameters(&timePulseParameters) == false)
  {
    myFile.println(F("getTimePulseParameters failed! Freezing..."));
    while (1) ; // Do nothing more
  }
  // Print the CFG TP5 version
  myFile.print(F("UBX_CFG_TP5 version: "));
```

```
  myFile.println(timePulseParameters.version);
  timePulseParameters.tpIdx = 0; // Select the TIMEPULSE pin

  // While the module is _locking_ to GNSS time, make it generate 100Hz
  timePulseParameters.freqPeriod = 100; // Set the frequency/period to 100Hz
  timePulseParameters.pulseLenRatio = 0x80000000; // Set the pulse ratio to 1/2 * 2^32 to
produce 33:67 mark:space

  // When the module is _locked_ to GNSS time, make it generate 100 Hz
  timePulseParameters.freqPeriodLock = 100; // Set the frequency/period to 100Hz
  timePulseParameters.pulseLenRatioLock = 0x80000000; // Set the pulse ratio to 1/2 * 2^32 to
produce 50:50 mark:space

  timePulseParameters.flags.bits.active = 1; // Make sure the active flag is set to enable the time
pulse. (Set to 0 to disable.)
  timePulseParameters.flags.bits.lockedOtherSet = 1; // Tell the module to use freqPeriod while
locking and freqPeriodLock when locked to GNSS time
  timePulseParameters.flags.bits.isFreq = 1; // Tell the module that we want to set the frequency
(not the period)
  timePulseParameters.flags.bits.isLength = 0; // Tell the module that pulseLenRatio is a ratio /
duty cycle (* 2^-32) - not a length (in us)
  timePulseParameters.flags.bits.polarity = 1; // Tell the module that we want the rising edge at
the top of second. (Set to 0 for falling edge.
```

```
  // Now set the time pulse parameters
  if (myGNSS.setTimePulseParameters(&timePulseParameters) == false)
  {
    myFile.println(F("setTimePulseParameters failed!"));
  }
  else
  {
    myFile.println(F("setTimePulseParameters successful!"));
  }
  myGNSS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT); //Save (only) the
communications port settings to flash and BBR
  myGNSS.setNavigationFrequency(20); //Send navigation solutions at 25 hz
  myGNSS.setAutoPVT(true);

  byte rate = myGNSS.getNavigationFrequency(); //Get the update rate of this module
  myFile.print("Current navigation update rate:");
  myFile.println(rate);

  // Blink the Teensy LED twice to indicate setup is complete and it is ready to collect data once
the button is pressed
  digitalWrite(TeensyLED, HIGH);
  delay(1000);
```

```
  digitalWrite(TeensyLED, LOW);
  delay(1000);
  digitalWrite(TeensyLED, HIGH);
  delay(1000);
  digitalWrite(TeensyLED, LOW);

  while (!buttonCheck) {

    buttonCheck = digitalRead(buttonPin);
    delay(600);
  }
  buttonCheck = false;

  // Check if the GNSS receiver  is currently generating positions with RTK corrections
  RTK = myGNSS.getCarrierSolutionType();
  if (RTK == 0) myFile.println(F("RTK Currently Off"));
  else if (RTK == 1) myFile.println(F("RTK Currently On (Floating))"));
  else if (RTK == 2) myFile.println(F("RTK Currently On (Fix)"));
  else myFile.println(F("Check RTK failed"));

  // Set interrupts
  attachInterrupt(ppsInterrupt, ppsFlag, RISING); // Attach an interrupt for GPS pulses (rising
edge only)
```

```
  attachInterrupt(Encoder1CHA, encoder1FlagA, CHANGE); // Attach an interrupt for state changes on channel A.
  attachInterrupt(Encoder1CHB, encoder1FlagB, CHANGE); // Attach an interrupt for state changes on channel B.
  attachInterrupt(Encoder2CHA, encoder2FlagA, CHANGE); // Attach an interrupt for state changes on channel A.
  attachInterrupt(Encoder2CHB, encoder2FlagB, CHANGE); // Attach an interrupt for state changes on channel B.
  attachInterrupt(Encoder3CHA, encoder3FlagA, CHANGE); // Attach an interrupt for state changes on channel A.
  attachInterrupt(Encoder3CHB, encoder3FlagB, CHANGE); // Attach an interrupt for state changes on channel B.
  attachInterrupt(Encoder4CHA, encoder4FlagA, CHANGE); // Attach an interrupt for state changes on channel A.
  attachInterrupt(Encoder4CHB, encoder4FlagB, CHANGE); // Attach an interrupt for state changes on channel B.

  // Turn the Teensy LED on to indicate that data is being collected
  digitalWrite(TeensyLED, HIGH);

  startTime = millis(); // Record startTime
} // End setup()
```

```
void loop() {


  if (newPulse) { // If data has not been printed since last pulse
    newPulse = false;
    printTime = interruptTeensyTime - startTime;
    if (myGNSS.getPVT()) { // If there is a new GNSS solution, get the values
      gpsHour = myGNSS.getHour();
      gpsMin = myGNSS.getMinute();
      gpsSec = myGNSS.getSecond();
      gpsMs = myGNSS.getMillisecond();
      gpsNs = myGNSS.getNanosecond();
      latitude = myGNSS.getLatitude();
      longitude = myGNSS.getLongitude();
      alt = myGNSS.getAltitude();
      SIV = myGNSS.getSIV();
      northVel = myGNSS.getNedNorthVel();
      eastVel = myGNSS.getNedEastVel();
      downVel = myGNSS.getNedDownVel();
      verticalAcc = myGNSS.getVerticalAccEst();
      horizontalAcc = myGNSS.getHorizontalAccEst();
      speedAcc = myGNSS.getSpeedAccEst();
      headAcc = myGNSS.getHeadingAccEst();
```

```
      dataString = ((String)printTime) + ", " + ((String)gpsPulseCount)  + ", " +
((String)polledEncoder1Count) + ", " + ((String)polledEncoder2Count) + ", " +
((String)polledEncoder3Count) + ", " + ((String)polledEncoder4Count) + ", " +
((String)gpsHour) + ", " + ((String)gpsMin) + ", " + ((String)gpsSec) + ", " + ((String)gpsMs) +
", " + ((String)gpsNs) + ", " + ((String)latitude) + ", " + ((String)longitude) + ", " + ((String)alt) +
", " + ((String)northVel) + ", " + ((String)eastVel) + ", " + ((String)downVel) + ", " +
((String)verticalAcc) + ", " + ((String)horizontalAcc) + ", " + ((String)speedAcc) + ", " +
((String)headAcc) + ", " + ((String)SIV);
    } // End if (myGNSS.getPVT())
    else {
      dataString = ((String)printTime) + ", " + ((String)gpsPulseCount)  + ", " +
((String)polledEncoder1Count) + ", " + ((String)polledEncoder2Count) + ", " +
((String)polledEncoder3Count) + ", " + ((String)polledEncoder4Count);
    }
    // if the file opened okay, write dataString to it:
    if (myFile) {
      myFile.println(dataString);
    }
  } // End if if (newPulse)

  thisTime = millis(); // Record the current time
```

```
  // Check if button is pressed once a second (1 hz), ignore button presses that take place in the
first 10 seconds after datalogging starts
  if ((thisTime - lastButtonCheckTime > 1000) && ((thisTime - startTime) >= 10000))
  {
    doSave = digitalRead(buttonPin);
    if (doSave) { //If button is pressed, save and pause
      if (myFile) {
        myFile.println("----------Save and pause----------");
        myFile.close();
        // Turn Teensy LED off to indicate that data is not being collected
        digitalWrite(TeensyLED, LOW);
      } // End if (myFile)
      delay(5000);
      // Wait for button press to resume dataLogging
      while (!digitalRead(buttonPin)) {
        delay(10);
      }
      // Turn on Teensy LED to indicate that data is again being collected
      digitalWrite(TeensyLED, HIGH);
      myFile = SD.open("datalog.txt", FILE_WRITE);

      if (!myFile) { // Check that the file opened correctly
        while (1);
```

```
      }
      delay(5000);
    } // End if (doSave)
lastButtonCheckTime = thisTime; // Record the last time the status of the button was checked
} // End if ((thisTime - lastButtonCheckTime > 1000) && ((thisTime - startTime) >= 10000))
} // End loop()
void encoder1FlagA() { // Interrupt function for state change in channel A
encoder1MasterCount++ ;
} // End encoder1FlagA()
void encoder1FlagB() { // Interrupt function for state change in channel B
encoder1MasterCount++;
} // End encoder1FlagB()
void encoder2FlagA() { // Interrupt function for state change in channel A
encoder2MasterCount++ ;
} // End encoder2FlagA()
void encoder2FlagB() { // Interrupt function for state change in channel B
encoder2MasterCount++;
} // End encoder2FlagB()
void encoder3FlagA() { // Interrupt function for state change in channel A
  encoder3MasterCount++ ;
} // End encoder3FlagA()
void encoder3FlagB() { // Interrupt function for state change in channel B
 encoder3MasterCount++;
```

```
} // End encoder3FlagB()
void encoder4FlagA() { // Interrupt function for state change in channel A
  encoder4MasterCount++ ;
} //End encoder4FlagA()
void encoder4FlagB() { // Interrupt function for state change in channel B
  encoder4MasterCount++;
} //End encoder4FlagB()
void ppsFlag() { //Interrupt function to run each time Teensy receives a GNSS pulse
  newPulse = true; // Set a flag to indicate that a new pulse has arrived
  gpsPulseCount++;
  polledEncoder1Count = encoder1MasterCount; // Record the counts for each encoder at this
instant
  polledEncoder2Count = encoder2MasterCount;
  polledEncoder3Count = encoder3MasterCount;
  polledEncoder4Count = encoder4MasterCount;
  interruptTeensyTime = millis();
} //End ppsFlag()
```

Appendix B

GPS Processing Code

```
%% Plot GPS Traces of Off-road Track
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  _____  _       _        _____ _____  _____
% |  __ \| |     | |      / ____|  __ \ / ____|
% | |__) | | ___ | |_    | |  __| |__) | (___
% |  ___/| |/ _ \| __|   | | |_ |  ___/ \___ \
% | |    | | (_) | |_    | |__| | |     ____) |
% |_|    |_|\___/ \__|    \_____|_|    |_____/
%
%
%  _______                                __
% |__   __|                              / _|
%    | |_ __ __ _  ___ ___  ___     ___ | |_
%    | | '__/ _` |/ __/ _ \/ __|  / _ \|  _|
%    | | | | (_| | (_|  __/\__ \ | (_) | |
%    |_|_|  \__,_|\___\___||___/  \___/|_|
%
%
%   ____   __  __                          _  _______                 _
%  / __ \ / _|/ _|                        | ||__   __|               | |
% | |  | | |_| |_ _ __ ___   __ _  __| |    | |  | |_ __ __ _  ___| | __
% | |  | |  _|  _| '__/ _ \ / _` |/ _` |    | |  | | '__/ _` |/ __| |/ /
% | |__| | | | | | | (_) | (_| | (_| |    | |  | | | | (_| | (__|   <
%  \____/|_| |_| |_|  \___/ \__,_|\__,_|    |_|  |_|  \__,_|\___|_|\_\
%
% https://patorjk.com/software/taag/#p=display&v=0&f=Big&t=
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all

GPS = load('20240314_GPS.txt');

allData_lat = [GPS(:,1)];
allData_lon = [GPS(:,2)];
allData_alt = [GPS(:,3)];

t = 1:length(allData_lat);

offset_Lon = 0.0000054;
offset_Lat = -0.0000008;

reference_latitude = 40.86368573;
reference_longitude = -77.83592832;
reference_altitude = 344.189;

h_geoplot = geoplot(reference_latitude, reference_longitude, '*','Color',[0 1
0],'Linewidth',3,'Markersize',10);
h_parent =  get(h_geoplot,'Parent');
set(h_parent,'ZoomLevel',19.125);
```

```
set(h_parent,'MapCenter',[40.865751663703335 -77.829815064107393]);
try
    geobasemap satellite

catch
    geobasemap openstreetmap
end
geotickformat -dd
hold on

% Plot latitude and longitude data
geoplot(allData_lat+offset_Lat, allData_lon+offset_Lon, '-','Color',[1 0
0],'Linewidth',1,'Markersize',10);
title('Penn State Off Road Test Track Course');

%% Convert data from LLA to ENU
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   _____                                 _          _       _
%  / ____|                               | |        | |     | |
% | |     ___  _ __  _____   _____ _ __| |_     __| | __ _| |_ __ _
% | |    / _ \| '_ \ \ / / _ \ '__| __|   / _` |/ _` | __/ _` |
% | |___| (_) | | | \ V /  __/ |  | |_   | (_| | (_| | || (_| |
%  \_____\___/|_| |_|\_/ \___|_|   \__|   \__,_|\__,_|\__\__,_|
%
%
%   __                          _      _
%  / _|                        | |    | |         /\
% | |_ _ __ ___  _ __ ___       | |    | |        /  \
% |  _| '__/ _ \| '_ ` _ \  | |    | |      / /\ \
% | | | | | (_) | | | | | | | |____| |____ / ____ \
% |_| |_|  \___/|_| |_| |_| |______|______/_/    \_\
%
%
%  _           ______ _   _ _    _
% | |         |  ____| \ | | |  | |
% | |_ ___    | |__  |  \| | |  | |
% | __/ _ \   |  __| | . ` | |  | |
% | || (_) | | |____| |\  | |__| |
%  \__\___/  |______|_| \_|\____/
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
allData_LLA = [allData_lat, allData_lon, allData_alt];
reference_LLA = [reference_latitude, reference_longitude, reference_altitude];

fig_num = 875576;
allData_ENU = fcn_GPS_lla2enu(allData_LLA, reference_LLA, fig_num);

% THIS SHOULD WORK ONCE THE GPS IS FIXED
% subplot(2,1,1);
% geotickformat -dd
% try
%     geobasemap satellite
%
% catch
```

```
%     geobasemap openstreetmap
% end

%% Break data into laps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  ____                 _       _____          _
% |  _ \               | |     |  __ \        | |
% | |_) |_ __ ___  __ _| | __  | |  | | __ _| |_ __ _
% |  _ <| '__/ _ \/ _` | |/ /  | |  | |/ _` | __/ _` |
% | |_) | | |  __/ (_| |   <   | |__| | (_| | || (_| |
% |____/|_|  \___|\__,_|_|\_\  |_____/ \__,_|\__\__,_|
%
%
%  _       _              _
% (_)     | |            | |
%  _ _ __ | |_ ___   ___  | |     __ _ _ __  ___
% | | '_ \| __/ _ \ / _ \ | |    / _` | '_ \/ __|
% | | | | | || (_) | | |__| (_| | |_) \__ \
% |_|_| |_|\__\___/  |______\__,_| .__/|___/
%                                | |
%                                |_|
%
https://patorjk.com/software/taag/#p=display&v=0&f=Big&t=Break%20%20Data%0Ainto%20Laps
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Convert ENU data into traversal types using the E as "X" and N as "Y"

% Convert paths to traversals structures. Each traversal instance is a
% "traversal" type, and the array called "data" below is a "traversals"
% type.
allData_traversal = fcn_Path_convertPathToTraversalStructure(allData_ENU);
allData_traveralArray.traversal{1} = allData_traversal;


% Plot the traversal
fig_num = 474764;
fcn_Laps_plotLapsXY(allData_traveralArray,fig_num);

% Call the Laps function to break data into laps
fig_num = 33737;
start_definition = [535 240; 520 246];
end_definition = [530 234; 515 240];
excursion_definition = []; % empty
title('Lap Input Data With Start and End Conditions')
xlabel('East [m]')
ylabel('North [m]')


lap_traversals = fcn_Laps_breakDataIntoLaps(...
    allData_traversal,...
    start_definition,...
    end_definition,...
    excursion_definition,...
```

```
    fig_num);
xlabel('East [m]')
ylabel('North [m]')

% Plot the lap traversals (should have 6)
fig_num = 323;
fcn_Laps_plotLapsXY(lap_traversals,fig_num);
title('Total Lap Data from GPS')
xlabel('East [m]')
ylabel('North [m]')

%% Find reference traversal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  ______ _             _
% |  ____(_)           | |
% | |__   _ _ __   __ _| |
% |  __| | | '_ \ / _` |
% | |    | | | | | (_| |
% |_|    |_|_| |_|\__,_|
%
%
%  _____        __
% |  __ \      / _|
% | |__) |___| |_ ___ _ __ ___ _ __   ___ ___
% |  _  // _ \  _/ _ \ '__/ _ \ '_ \ / __/ _ \
% | | \ \  __/ ||  __/ |  __/ | | | (_|  __/
% |_|  \_\___|_| \___|_|  \___|_| |_|\___\___|
%
%
%  _______                                 _
% |__   __|                               | |
%    | |_ __ __ ___   _____ _ __ ___  __ _| |
%    | | '__/ _` \ \ / / _ \ '__/ __|/ _` | |
%    | | | | (_| |\ V /  __/ |  \__ \ (_| | |
%    |_|_|  \__,_| \_/ \___|_|  |___/\__,_|_|
%
%
%
https://patorjk.com/software/taag/#p=display&v=0&f=Big&t=Find%20%0AReference%20%0ATra
versal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Resample Data
reference_traversal = lap_traversals.traversal{1};
interval = 0.2;
reference_station_points    = (0:interval:reference_traversal.Station(end))';
redecimated_reference_traversal = ...
    fcn_Path_newTraversalByStationResampling(reference_traversal,
reference_station_points);

% Redecimate all the traversals
redecimated_lap_traversals = lap_traversals;
for ith_traversal = 1:length(lap_traversals.traversal)
    redecimated_lap_traversals.traversal{ith_traversal} = ...
```

```
fcn_Path_newTraversalByStationResampling(lap_traversals.traversal{ith_traversal},
reference_station_points);
end

% Plot the redecimated lap traversals (should have 6)
fig_num = 3454;
figure(fig_num);
clf;
title('Manually Input Traversal Data over Laps')
xlabel('East [m]')
ylabel('North [m]')

fcn_Laps_plotLapsXY(redecimated_lap_traversals,fig_num);


%% Use path library to find average traversal
% % choose first one for traversal
% % reference_traversal = lap_traversals.traversal{2};
% num_iterations = 5;
% weight_for_averaging = 0;
% averaging_fig = 22;
%
%
% figure(averaging_fig);
% clf;
% hold on;
% grid on;
% axis equal;
%
% [path_average, closestXs, closestYs, closestDistances]  = ...
%
fcn_Path_findAverageTraversalViaOrthoProjection(redecimated_lap_traversals,reference_
traversal,num_iterations,weight_for_averaging,averaging_fig);


%% Use ginput to get manual "reference"
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   _____                    _
%  |_   _|                  | |
%    | |  _ __  _ __  _   _| |_
%    | | | '_ \| '_ \| | | | __|
%   _| |_| | | | |_) | |_| | |_
%  |_____|_| |_| .__/ \__,_|\__|
%              | |
%              |_|
%   __  __                          _
%  |  \/  |                        | |
%  | \  / | __ _ _ __  _   _  __ _| |
%  | |\/| |/ _` | '_ \| | | |/ _` | |
%  | |  | | (_| | | | | |_| | (_| | |
%  |_|  |_|\__,_|_| |_|\__,_|\__,_|_|
%
%
```

```
%   _______                                 _ 
%  |__   __|                               | |
%     | |_ __ __ ___   _____ _ __ ___  __ _| |
%     | | '__/ _` \ \ / / _ \ '__/ __|/ _` | |
%     | | | | (_| |\ V /  __/ |  \__ \ (_| | |
%     |_|_|  \__,_| \_/ \___|_|  |___/\__,_|_|
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% [X,Y] = ginput;
% reference_path = [X, Y];
% fprintf(1,'Copy the following and past into MATLAB script:\n');
% disp(reference_path);

% % Copy first 10 points to the end to force overlap
% manually_selected_XY_points = [manually_selected_XY_points;
manually_selected_XY_points(1:10,:)];
%
% % Keep only the unique points
% [manually_selected_XY_points] =
unique(manually_selected_XY_points,'rows','stable');
%
% figure(fig_num)
% hold on
% plot(manually_selected_XY_points(:,1),manually_selected_XY_points(:,2),'g-
','LineWidth',3)
%
% % Convert this into a traversal (for resampling)
% manually_selected_traversal =
fcn_Path_convertPathToTraversalStructure(manually_selected_XY_points);
%
% % Resample the path to exactly 10 cm spacing
% path_length = manually_selected_traversal.Station(end)+2;
% new_stations = (0:0.10:path_length)';
% fig_num = 22;
% [equal_spaced_manually_selected_traversal] =
fcn_Path_newTraversalByStationResampling(manually_selected_traversal,new_stations,(fi
g_num));


manual_data = load('20240328_Manual_XY.mat');

figure(fig_num)
hold on
plot(manual_data.manually_selected_XY_points(:,1),manual_data.manually_selected_XY_po
ints(:,2),'g-','LineWidth',3)

% Convert this into a traversal (for resampling)
manually_selected_traversal =
fcn_Path_convertPathToTraversalStructure(manual_data.manually_selected_XY_points);

% Resample the path to exactly 10 cm spacing
path_length = manually_selected_traversal.Station(end)+2;
new_stations = (0:0.10:path_length)';
```

```
fig_num = 22;
[equal_spaced_manually_selected_traversal] =
fcn_Path_newTraversalByStationResampling(manually_selected_traversal,new_stations,(fi
g_num));

% Clean up the path to make it "smooth" using a distance-based smoothing
% filter. The Nyquist spatial sampling frequency is 1/2 of the the spatial
% sampling frequency (e.g. 10 samples/meter or 10 "spatial" Hz), so our
% Nyquist frequency is 5 spatial Hz.

[B,A] = butter(2,(1/10)/5);
initial_X = [equal_spaced_manually_selected_traversal.X(1),
equal_spaced_manually_selected_traversal.X(1)];
initial_Y = [equal_spaced_manually_selected_traversal.Y(1),
equal_spaced_manually_selected_traversal.Y(1)];

filtered_manual_traversal_x =
filtfilt(B,A,equal_spaced_manually_selected_traversal.X);
filtered_manual_traversal_y =
filtfilt(B,A,equal_spaced_manually_selected_traversal.Y);

% Close off the ends before plotting
filtered_manual_traversal_x = [filtered_manual_traversal_x;
filtered_manual_traversal_x(1)];
filtered_manual_traversal_y = [filtered_manual_traversal_y;
filtered_manual_traversal_y(1)];

filtered_manual_traversal =
fcn_Path_convertPathToTraversalStructure([filtered_manual_traversal_x
filtered_manual_traversal_y]);

% Plot the redecimated lap traversals (should have 6)
fig_num = 22;
figure(fig_num);
filtered_manual_traversalArray.traversal{1} = filtered_manual_traversal;
fcn_Laps_plotLapsXY(filtered_manual_traversalArray,fig_num);

% Plot the results (nicely)
fig_num = 34784;
figure(fig_num);
clf;

hold on;
grid on;
axis equal;

handles = fcn_Path_plotTraversalsXY(lap_traversals, fig_num);
legend_strings = cell(1);
for ith_handle = 1:length(handles)
    set(handles(ith_handle),'LineWidth',3,'Marker','.','Markersize',20);
    legend_strings{ith_handle} = sprintf('Lap %.0d',ith_handle);
end
handle_filt = fcn_Path_plotTraversalsXY(filtered_manual_traversalArray,fig_num);
set(handle_filt,'LineWidth',5,'Marker','none','Markersize',20,'Color',[1 0 0]);
```

```
legend_strings{7} = sprintf('Reference Lap');
legend(legend_strings);
title('Lap data versus manual data smoothed');
xlabel('East [m]')
ylabel('North [m]')

reference_lap_x = filtered_manual_traversal_x;
reference_lap_y = filtered_manual_traversal_y;
%
% % NumLaps = length(lap_traversals.traversal);
% %
% % % Initialize the data
% % XY_lap_data{NumLaps} = cell;
% % for ith_lap = 1:NumLaps
% %     XY_lap_data{NumLaps} = [lap_traversals.traversal.X lap_traversals.traversal.Y
% % end
% fcn_Laps_plotLapsXY(lap_traversals,fig_num);

%% Plot deviation of laps versus reference
reference_traversal = filtered_manual_traversal;
reference_station_points = (0:1:reference_traversal.Station(end))';
flag_rounding_type = 3; % Use average of projections at end points
search_radius = 7;

[~, ~, closestDistances] = fcn_Path_findOrthoScatterFromTraversalToTraversals(
reference_station_points, reference_traversal, lap_traversals,
flag_rounding_type,search_radius);
%%
% plot the variances with the reference path, 1 standard deviation
variance_fig = 2343;
figure(variance_fig)
clf
axis equal
title('Average Plot Traversal With Relative Track Limits')
xlabel('East [m]')
ylabel('North [m]')
grid on;

legend('off')
% standard_deviation_in_path_following =
std(closestDistances(~isnan(closestDistances)),0,'all');
standard_deviation_in_path_following = 2;
fcn_Path_plotTraversalXYWithVarianceBands(reference_traversal,standard_deviation_in_p
ath_following,variance_fig);


% Plot the path errors
figure(3737)
clf;
hold on;
grid on;

for ith_lap = 1:length(closestDistances(1,:))
    plot(reference_station_points,closestDistances(:,ith_lap),'-');
```

```
end


%% Plot the results

% Plot the final XY result of orthogonal
path_points_fig = 33333;
figure(path_points_fig);
clf;
title('Segmented Lap Data')
xlabel('East [m]')
ylabel('North [m]')
hold on
grid on;
axis equal;


fcn_Path_plotTraversalsXY(lap_traversals,path_points_fig);
plot(reference_traversal.X,reference_traversal.Y,'Linewidth',4);
title('Original paths and final average path via orthogonal projections')
xlabel('X [m]')
ylabel('Y [m]')
```

Appendix C

Acceleration Processing Code

```
%% Acceleration Pre Processing
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                        _   _
%      /\                | |              | | (_)
%     /  \   ___ ___ ___| | ___ _ __ __ _| |_ _  ___  _ __
%    / /\ \ / __/ __/ _ \ |/ _ \ '__/ _` | __| |/ _ \| '_ \
%   / ____ \ (_| (_|  __/ |  __/ | | (_| | |_| | (_) | | | |
%  /_/    \_\___\___\___|_|\___|_|  \__,_|\__|_|\___/|_| |_|
%
%
%   _____                          _
%  |  __ \                        (_)
%  | |__) | __ ___   ___ ___  ___ _ _ __   __ _
%  |  ___/ '__/ _ \ / __/ _ \/ __/ __| | '_ \ / _` |
%  | |   | | | (_) | (_|  __/\__ \__ \ | | | | (_| |
%  |_|   |_|  \___/ \___\___||___/___/_|_| |_|\__, |
%                                               __/ |
%                                              |___/
% https://patorjk.com/software/taag/#p=display&v=0&f=Big&t=
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Mass of the Vehicle [kg]
m = 14.14;

% Load Acceleration Data
a = load('20240314_Acceleration.txt');

%{
Segmenting Data Per X Y Z
X is Vehicle Direction Acceleration
Y is Side to Side Acceleration of Vehicle
Z is perpendicular to Ground Plane
%}

% Acceleration Data in X Y Z
x_acceleration = a(:,1);
y_acceleration = a(:,2);
z_acceleration = a(:,3);
time = a(:,4);

%% Processing

% Reduced Noise Data
window_size = 75;
smoothed_x_acc = movmean(x_acceleration, window_size);
smoothed_y_acc = movmean(y_acceleration, window_size);
smoothed_z_acc = movmean(z_acceleration, window_size);
```

```
% Segmenting Data for First Test
starts1 = [9.03; 26.1; 87.57; 106.53; 182.28; 201.99; 239.91];
ends1 = [21.27; 36.3; 98.22; 117.96; 197.37; 217.83; 254.04];

% Segmenting Data for Second Test
starts2 = [125.46; 179.13; 203.97];
ends2 = [134.7; 189.81; 214.53];

% Combined Segmented Data
test_starts = [starts1; 430.65 + starts2];
test_ends = [ends1; 430.65 + ends2];

% Initializing Variables
acceleration =  zeros(1,length(test_starts));
distance = zeros(1,length(test_starts));
kinematic_distance = zeros(1,length(test_starts));
max_velo = zeros(1,length(test_starts));

% Processing For Loop For Each Segmented Data
for acceleration_test = 1:length(test_starts)

    start_time = test_starts(acceleration_test);
    end_time = test_ends(acceleration_test);

    delta_time = end_time - start_time;

    % Find indices corresponding to start and end times
    start_index = find(time >= start_time, 1);
    end_index = find(time > end_time, 1);

    % Extract segmented data
    segment_t = time(start_index:end_index);
    segment_x = x_acceleration(start_index:end_index);

    acceleration(acceleration_test) = mean(abs(segment_x));

    % Apply moving average filter
    window_size = 25; % Adjust window size as needed
    smoothed_x = movmean(abs(segment_x), window_size);

    % Velocity Integrations
    velocity_norm = cumtrapz(time(start_index:end_index), segment_x);
    velocity_smooth = cumtrapz(time(start_index:end_index), smoothed_x);

    % Distance Integrations
    distance_smooth = cumtrapz(time(start_index:end_index), velocity_smooth);
    distance_norm = cumtrapz(time(start_index:end_index), velocity_norm);

    % Calculations of Data
    distance(acceleration_test) = distance_smooth(end)-abs(distance_norm(end));
    max_velo(acceleration_test) = max(abs(velocity_norm));
    kinematic_distance(acceleration_test) =
max_velo(acceleration_test)^2/(2*acceleration(acceleration_test));
```

```
    % Plotting All Data
    tprime = linspace(0,delta_time,length(segment_x));

    hold on
    figure(70)
    plot(tprime, segment_x, 'Linewidth',1)
    title('Acceleration vs Time')
    xlabel('Time [s]')
    ylabel('Acceleration [m/s^2]')
    grid on

    hold on
    figure(71)
    plot(tprime, abs(velocity_norm), 'Linewidth',2)
    title('Velocity vs Time')
    xlabel('Time [s]')
    ylabel('Velocity [m/s]')
        grid on

    legend_labels{acceleration_test} = sprintf('Trial %d', acceleration_test);
%#ok<SAGROW>
end

figure(70)
legend(legend_labels, 'Location','northwest')

figure(71)
legend(legend_labels, 'Location','southeast')


distance_matrix = [mean(distance);
                   mean(kinematic_distance)];

average_acceleration = mean(acceleration);
std_acc = std(acceleration);

average_max_velocity = mean(max_velo);
std_velo = std(max_velo);

average_distance = mean(distance_matrix);
std_dist = std(distance_matrix);

calculated_distance = average_max_velocity^2/(2*average_acceleration);

percent_error = 100*abs(calculated_distance - average_distance)/calculated_distance;

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                                                       _   _
%      /\                |  |                |  | (_)
%     /  \   ___ ___ ___| | ___ _ __ __ _| |_ _  ___  _ __
%    / /\ \ / __/ __/ _ \ |/ _ \ '__/ _` | __| |/ _ \| '_ \
%   / ____ \ (_| (_|  __/ |  __/ | | (_| | |_| | (_) | | | |
%  /_/    \_\___\___\___|_|\___|_|  \__,_|\__|_|\___/|_| |_|
```

```
%
%
%   _____  _       _
%  |  __ \| |     | |
%  | |__) | | ___ | |_ ___
%  |  ___/| |/ _ \| __/ __|
%  | |    | | (_) | |_\__ \
%  |_|    |_|\___/ \__|___/
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% USE BELOW PLOTS TO FIND STARTING AND ENG INDICES OF ACCELERATION
% UNCOMMENT AND DEFINE START AND END POINTS TO SEGMENT DATA ACCORDINGLY

% Plot original and smoothed data
% figure;
% plot(t(start_index:end_index), abs(seg_x), 'b-', 'LineWidth', 1);
% hold on;
% plot(t(start_index:end_index), smoothed_x, 'r-', 'LineWidth', 1);
% xlabel('Time');
% ylabel('X');
% title('Original and Smoothed Data');
% legend('Original', 'Smoothed');
% grid on;

% figure
% plot(seg_t, abs(seg_x));
% xlabel('Time');
% ylabel('Value');
% title('Segmented X Acceleration');
% grid on;

% Plots of All Acceleration Data
% figure(6123)
% subplot(3,2,1)
% plot(time,smoothed_x_acc,'Linewidth',1)
% title('X Acceleration Noise Reduced')
% xlabel('Time [s]')
% ylabel('Acceleration [m/s]')
% grid on
%
% subplot(3,2,3)
% plot(time,smoothed_y_acc,'Linewidth',1)
% title('Y Acceleration Noise Reduced')
% xlabel('Time [s]')
% ylabel('Acceleration [m/s]')
% grid on
%
% subplot(3,2,5)
% plot(time,smoothed_z_acc,'Linewidth',1)
% title('Z Acceleration Noise Reduced')
% xlabel('Time [s]')
% ylabel('Acceleration [m/s]')
% grid on
```

```
subplot(3,1,1)
plot(time,x_acceleration)
title('Raw X Acceleration')
xlabel('Time [s]')
ylabel('Acceleration [m/s]')
grid on

subplot(3,1,2)
plot(time,y_acceleration)
title('Raw Y Acceleration')
xlabel('Time [s]')
ylabel('Acceleration [m/s]')
grid on

subplot(3,1,3)
plot(time,z_acceleration)
title('Raw Z Acceleration')
xlabel('Time [s]')
ylabel('Acceleration [m/s]')
grid on


% X Acceleration Only
figure(237846)
plot(time,smoothed_x_acc,'Linewidth',2)
title('Noise Reduced Vehicle X Acceleration Data')
xlabel('Time [s]')
ylabel('Acceleration [m/s]')
grid on
```

Appendix D

MATLAB Path Following Simulation Code

```
%% Vehicle Path-following Simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %
 % __      __  _     _      _
 % \ \    / / | |   (_)    | |
 %  \ \  / /__| |__  _  ___| | ___
 %   \ \/ / _ \ '_ \| |/ __| |/ _ \
 %    \  /  __/ | | | | (__| |  __/
 %     \/ \___|_| |_|_|\___|_|\___|
 %
 %
 %  _____      _   _            __       _ _               _
 % |  __ \    | | | |          / _|     | | |             (_)
 % | |__) |_ _| |_| |__ ______| |_ ___ | | | _____      ___ _ __   __ _
 % |  ___/ _` | __| '_ \______|  _/ _ \| | |/ _ \ \ /\ / / | '_ \ / _` |
 % | |  | (_| | |_| | | |     | || (_) | | | (_) \ V  V /| | | | | (_| |
 % |_|   \__,_|\__|_| |_|     |_| \___/|_|_|\___/ \_/\_/ |_|_| |_|\__, |
 %                                                                 __/ |
 %                                                                |___/
 %   _____ _                 _       _   _
 %  / ____(_)               | |     | | (_)
 % | (___  _ _ __ ___  _   _| | __ _| |_ _  ___  _ __
 %  \___ \| | '_ ` _ \| | | | |/ _` | __| |/ _ \| '_ \
 %  ____) | | | | | | | |_| | | (_| | |_| | (_) | | | |
 % |_____/|_|_| |_| |_|\__,_|_|\__,_|\__|_|\___/|_| |_|
 %
 %
% See: https://patorjk.com/software/taag/#p=display&v=0&f=Big&t=
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The goal of this section is to use the vehicle dynamics (VD) class
% library of codes to demonstrate the ability of a vehicle to follow a
% reference XY path that we defined above. The results should be a plot of
% The vehicle's XY position on top of the reference path, showing that
% path-following is working, and that the vehicle has some error. Another
% goal is to calculate and plot the path-following errors so we can analyze
% Them later in ILC. Finally, we want to generate plots of tracking error
% versus vehicle speed, showing that the path error gets larger with
% increasing speed.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   _____                   _
%  |_   _|                 | |
%    | |  _ __  _ __  _   _| |_ ___
%    | | | '_ \| '_ \| | | | __/ __|
%   _| |_| | | | |_) | |_| | |_\__ \
%  |_____|_| |_| .__/ \__,_|\__|___/
%              | |
%              |_|
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%% BELOW VEHICLE PARAMETER FOR 1/5th Scale HSOV

% Define vehicle and controller properties
% Define a MATLAB structure that specifies the physical values for a vehicle.
% For convenience, we ask that you call this stucture 'vehicle'.
% massratio = 14.719/1600;    % mass ratio between large vehicle and small
% vehicle.m   = 14.719; % mass (kg)
% vehicle.Izz = 1.20760570275114; % mass moment of inertia (kg m^2)
% vehicle.Iw  = 0.00120760570275114; % mass moment of inertia of a wheel (kg m^2)
% vehicle.Re  = 0.09; % effective radius of a wheel (m)
% vehicle.a   = 0.262756405577049; % length from front axle to CG (m)
% vehicle.L   = 0.575; % wheelbase (m)
% vehicle.b   = 0.312243594422951; % length from rear axle to CG (m)
% vehicle.d   = 0.44; % track width (m)
% vehicle.h_cg = 0.1; % height of the cg (m)
% vehicle.Ca  = massratio*[95000; 95000; 110000; 110000]; % wheel cornering
stiffnesses
% vehicle.Cx  = massratio*[65000; 65000; 65000; 65000]; % longitudinal stiffnesses

% BELOW VEHICLE PARAMETERS FOR LARGE SCALE VEHICLE

% Define inputs to the vehicle model
% scaling factor is the difference between a full-scale and small-scale
% vehicle
scalingFactor = 10;
% Define vehicle and controller properties
% Define a MATLAB structure that specifies the physical values for a vehicle.
% For convenience, we ask that you call this stucture 'vehicle'.
vehicle.m   = 1600; % mass (kg)
vehicle.Izz = 2500; % mass moment of inertia (kg m^2)
vehicle.Iw  = 1.2; % mass moment of inertia of a wheel (kg m^2)
vehicle.Re  = 0.32; % effective radius of a wheel (m)
vehicle.a   = 1.3; % length from front axle to CG (m)
vehicle.L   = 2.6; % wheelbase (m)
vehicle.b   = 1.3; % length from rear axle to CG (m)
vehicle.d   = 1.5; % track width (m)
vehicle.h_cg = 0.42; % height of the cg (m)
vehicle.Ca  = [95000; 95000; 110000; 110000]; % wheel cornering stiffnesses
vehicle.Cx  = [65000; 65000; 65000; 65000]; % longitudinal stiffnesses


vehicle.contact_patch_length = 0.15; % [m] % divide by 5 for small vehicle
vehicle.friction_ratio = 1;

controller.look_ahead_distance = scalingFactor; % look-ahead distance [meters]
controller.steering_Pgain = 0.1; % P gain for steering control

% Define the reference trajectory
% Make the trajectory 10 times larger, because the full-sized vehicle is 10
% times larger. If we used the real track size, the vehicle couldn't make
% the curves.

filtered_manual_traversal_x_scaled = filtered_manual_traversal_x*scalingFactor;
```

```
filtered_manual_traversal_y_scaled = filtered_manual_traversal_y*scalingFactor;
filtered_manual_traversal =
fcn_Path_convertPathToTraversalStructure([filtered_manual_traversal_x_scaled
filtered_manual_traversal_y_scaled]);
reference_traversal = filtered_manual_traversal;

% Plot and check the "BIG" reference traversal
figure(4747);
clf; hold on;
grid on;

plot(reference_traversal.X,reference_traversal.Y,'r','Linewidth',3);
title('Scaled Reference Traversal to be used for Path-following Tests')
xlabel('X [m]')
ylabel('Y [m]')
axis equal

X_trajectory       = reference_traversal.X;
Y_trajectory       = reference_traversal.Y;
Yaw_trajectory     = [reference_traversal.Yaw(1);reference_traversal.Yaw];

Station_trajectory = reference_traversal.Station;
inputTrajectory = scalingFactor*[X_trajectory Y_trajectory Yaw_trajectory
Station_trajectory];

vdParam.fieldsTrajectory.east = 1;
vdParam.fieldsTrajectory.north = 2;
vdParam.fieldsTrajectory.yaw = 3;
vdParam.fieldsTrajectory.station = 4;
vdParam.searchDistance = 4; % [meters]
vdParam.trajectorySize = size(inputTrajectory);
vdParam.contactPatchLength = 0.15;
vdParam.frictionCoefficientRatio = 1;
vdParam.sampling_time_gps = 0.01; % [seconds]
vdParam.sampling_time_imu = 0.01;
vdParam.longitudinalTransfer = 1;


% Define initial conditions
% Parameters and initial conditions for matlab model
U0 = 15; % longitudinal velocity [m/s]
U = U0;  % longitudinal velocity [m/s]
V = 0; % lateral velocity [m/s]
r = 0; % yaw rate [rad/s]
omega = U0*ones(4,1)/vehicle.Re; % angular velocity of wheel [rad/s]

% initial pose
east = 5100;
north = 2000;
heading = Yaw_trajectory(1);

% road properties
road_properties.grade = 0; road_properties.bank_angle = 0;
friction_coefficient = [0.9, 0.9, 0.9, 0.9];
```

```
% Define initial conditions
% initial.longitudinalSpeed = 3; % longitudinal velocity of vehicle [m/s]
% initial.wheelSpeeds = initial.longitudinalSpeed*ones(1,2)/vehicle.Re; % angular
velocity of wheel [rad/s]
% initial.east    = 0; % initial pose
% initial.north   = 0;
% initial.heading = 0; % [rad]
% road_properties.grade = 0; road_properties.bank_angle = 0; % road properties
% friction_coefficient = [0.8, 0.8];

% Define load transfer conditions
type_of_transfer = 'both';
% type_of_transfer = 'default';

% Define inputs to the vehicle model
wheel_torque = [0; 0; 5; 5]; % wheel torque [Nm]

% Define items used to determine how long to run sim
TotalTime = Station_trajectory(end)/U0; % This is how long the simulation will run.
deltaT = 0.01;
N_timeSteps = floor(TotalTime/deltaT)+1; % This is the number of time steps we should
have

% Main code
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    __  __       _
%   |  \/  |     (_)
%   | \  / | __ _ _ _ __
%   | |\/| |/ _` | | '_ \
%   | |  | | (_| | | | | |
%   |_|  |_|\__,_|_|_| |_|
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Run the simulation in MATLAB
% variables to store outputs of Matlab simulation
matlab_time = nan(N_timeSteps,1);
matlab_States = nan(N_timeSteps,9); matlab_pose = nan(N_timeSteps,3);
matlab_alpha = nan(N_timeSteps,4); matlab_kappa = nan(N_timeSteps,4);
matlab_Fz = nan(N_timeSteps,4); matlab_Mz = nan(N_timeSteps,4);
matlab_Fx = nan(N_timeSteps,4); matlab_Fy = nan(N_timeSteps,4);
matlab_friction = nan(N_timeSteps,4);
matlab_station = nan(N_timeSteps,1);
matlab_trackingError = nan(N_timeSteps,1);

global flag_update global_acceleration
global_acceleration = zeros(7,1);
input_states = [U0;V;r;omega;east;north;heading]; % initial conditions
counter = 1;
for t = 0:deltaT:TotalTime
    % Print every 100 iterations
    if 0==mod(counter,100)
        fprintf(1,'Time: %.3f of %.3f\n',counter*deltaT,TotalTime);
```

```
end
matlab_time(counter) = t;
matlab_States(counter,1:7) = input_states(1:7)';
matlab_pose(counter,:)     = input_states(8:10)';

% Controller: Steering
pose = matlab_pose(counter,:)';
[closest_xy,closest_station,~] = ...
    fcn_Path_snapPointOntoNearestTraversal(pose(1:2)',...
    reference_traversal);
matlab_trackingError(counter) = sum((closest_xy - pose(1:2)').^2,2).^0.5;
matlab_station(counter) = closest_station;
target_lookAhead_pose = fcn_VD_snapLookAheadPoseOnToTraversal(pose,...
    reference_traversal,controller);
target_lookAhead_pose = real(target_lookAhead_pose);
steering_angle = fcn_VD_lookAheadLatController(pose,target_lookAhead_pose,...
    controller);

% Estimate Slips for time 't'
% Slip Angle/Lateral Slip
slip_angle = fcn_VD_dtSlipAngle(U,V,r,steering_angle,vehicle);

% Wheel Slip/Longitudinal Slip
wheel_slip = fcn_VD_dtWheelSlip(U,V,r,omega,steering_angle,vehicle);
matlab_alpha(counter,:) = slip_angle';
matlab_kappa(counter,:) = wheel_slip';

% 7-DoF Vehicle Model
flag_update = true; % set it to to true before every call to RK4 method
[~,y] = fcn_VD_RungeKutta(@(t,y) fcn_VD_dt7dofModelForController(t,y,...
    steering_angle,wheel_torque,...
    vehicle,road_properties,friction_coefficient',type_of_transfer),...
    input_states,t,deltaT);

% Extract ODE solver solution out
U = y(1);

% Overwrite U to force a constant speed
U = U0;
y(1) = U0;

V = y(2);
r = y(3);
omega = y(4:7);

input_states = y; clear y;
matlab_States(counter,8:9) = global_acceleration(1:2)';

% Estimate Normal Forces for time 't'
if 1==1 %counter
    normal_force = fcn_VD_dtNormalForce([0;0],vehicle,road_properties,...
        type_of_transfer);
else
    normal_force = fcn_VD_dtNormalForce(matlab_States(counter-1,8:9)',vehicle,...
```

```
            road_properties,type_of_transfer);
    end
    matlab_Fz(counter,:) = normal_force';

    % Estimate Tire forces for time 't'
    tire_force = fcn_VD_dtTireForceBrush(slip_angle,wheel_slip,normal_force,...
        friction_coefficient',vehicle);
    matlab_Fx(counter,:) = tire_force(:,1)';
    matlab_Fy(counter,:) = tire_force(:,2)';

    % Aligning Moment
    aligning_moment = fcn_VD_dtAligningMomentBrush(slip_angle,normal_force,...
        friction_coefficient',vehicle);
    matlab_Mz(counter,:) = aligning_moment';

    % Friction Estimation
    estimated_vel = matlab_States(counter,1:3)'+...
        [normrnd(0,1.4142*0.25)*0.01; normrnd(0,1.4142*0.25)*0.01; ...
        normrnd(0,1.4142*0.009)*0.01];
    if 1==counter
        estimated_accel = [0; 0]+...
            [normrnd(0,0.5)*(0.01^2); normrnd(0,0.5)*(0.01^2)];
    else
        estimated_accel = matlab_States(counter-1,8:9)'+...
            [normrnd(0,0.5)*(0.01^2); normrnd(0,0.5)*(0.01^2)];
    end
    slip_angle = fcn_VD_dtSlipAngle(estimated_vel(1),estimated_vel(2),...
        estimated_vel(3),steering_angle,vehicle);
    normal_force = fcn_VD_dtNormalForce(estimated_accel,vehicle,road_properties,...
        type_of_transfer);
    friction_estimate = fcn_VD_estimateFrictionCoefficient(slip_angle,...
        normal_force,aligning_moment,vehicle);
    matlab_friction(counter,:) = friction_estimate';

    counter = counter+1;
end

%% Plots to check MATLAB simulation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   _____  _       _   _   _
%  |  __ \| |     | | | | (_)
%  | |__) | | ___ | |_| |_ _ _ __   __ _
%  |  ___/| |/ _ \| __| __| | '_ \ / _` |
%  | |    | | (_) | |_| |_| | | | | (_| |
%  |_|    |_|\___/ \__|\__|_|_| |_|\__, |
%                                    __/ |
%                                   |___/
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all

% Fix station jump
jump_indicies = find(diff(matlab_station)<0);
matlab_station(jump_indicies) = nan;
```

```
fcn_VD_plotTimeSlipAngle(matlab_time,matlab_alpha);
fcn_VD_plotStationSlipAngle(matlab_station,matlab_alpha);

% fcn_VD_plotTimeWheelSlip(matlab_time,matlab_kappa);
% fcn_VD_plotTimeNormalForce(matlab_time,matlab_Fz);
% fcn_VD_plotTimeLongitudinalTireForce(matlab_time,matlab_Fx);

fcn_VD_plotTimeLateralTireForce(matlab_time,matlab_Fy);
fcn_VD_plotStationLateralTireForce(matlab_station,matlab_Fy);

% fcn_VD_plotTimeAligningMoment(matlab_time,matlab_Mz);

% fcn_VD_plotTimeLongitudinalAcceleration(matlab_time,matlab_States(:,8));

friction = 0.9;  % Check this
fcn_VD_plotTimeLateralAcceleration(matlab_time,matlab_States(:,9),friction);
fcn_VD_plotStationLateralAcceleration(matlab_station,matlab_States(:,9),friction);

fcn_VD_plotTimeTrackingError(matlab_time,matlab_trackingError,10);
fcn_VD_plotStationTrackingError(matlab_station,matlab_trackingError,10);

% fcn_VD_plotTimeLongitudinalVelocity(matlab_time,matlab_States(:,1));
% fcn_VD_plotTimeLateralVelocity(matlab_time,matlab_States(:,2));
% fcn_VD_plotTimeYawRate(matlab_time,matlab_States(:,3));
% fcn_VD_plotTimeWheelSpeed(matltime,matlab_States(:,(4:7)));

fcn_VD_plotCompareTrajectory([reference_traversal.X, reference_traversal.Y],...
    'Reference Path',matlab_pose(:,[1,2]),'Actual Path',Station_trajectory);

% fcn_VD_plotTimeYaw(matlab_time,matlab_pose(:,3));

% fcn_VD_plotTimeFriction(matlab_time,matlab_friction);
```

BIBLIOGRAPHY

[1] M. V. Rajasekhar and A. K. Jaswal, “Autonomous vehicles: The future of automobiles,” *2015 IEEE International Transportation Electrification Conference, ITEC-India 2015*, Jan. 2016, doi: 10.1109/ITEC-INDIA.2015.7386874.

[2] M. N. Ahangar, Q. Z. Ahmed, F. A. Khan, and M. Hafeez, “A Survey of Autonomous Vehicles: Enabling Communication Technologies and Challenges,” *Sensors 2021*, vol. 21, no. 3, p. 706, Jan. 2021, doi: 10.3390/S21030706.

[3] S. Campbell *et al.*, “Sensor Technology in Autonomous Vehicles : A review,” *29th Irish Signals and Systems Conference, ISSC 2018, Belfast*, Dec. 2018, doi: 10.1109/ISSC.2018.8585340.

[4] J. Li, H. Gang, H. Ma, M. Tomizuka, and C. Choi, “Important Object Identification with Semi-Supervised Learning for Autonomous Driving,” *Proc IEEE Int Conf Robot Autom, Philadelphia*, pp. 2913–2919, 2022, doi: 10.1109/ICRA46639.2022.9812234.

[5] S. B. Kim, S. Y. Lee, T. H. Hwang, and K. H. Choi, “An advanced approach for navigation and image sensor integration for land vehicle navigation,” *IEEE Vehicular Technology Conference*, Milan, vol. 60, no. 6, pp. 4075–4078, 2004, doi: 10.1109/VETECF.2004.1404844.

[6] J. Ban, G. Chen, L. Wang, and Y. Meng, “A calibration method for rotary optical encoder temperature error in a rotational inertial navigation system,” *Meas Sci Technol*, vol. 33, no. 6, p. 065203, Mar. 2022, doi: 10.1088/1361-6501/AC4C67.

[7] “H5 | US Digital.” Accessed: Mar. 01, 2024. [Online]. Available: https://www.usdigital.com/products/encoders/incremental/shaft/H5

[8] H. Rastgoftar, B. Zhang, and E. M. Atkins, “A Data-Driven Approach for Autonomous Motion Planning and Control in Off-Road Driving Scenarios,” *Proceedings of the American Control Conference, Milwaukee*, vol. 2018-June, pp. 5876–5883, Aug. 2018, doi: 10.23919/ACC.2018.8431069.

[9] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel, “Path Planning for Autonomous Vehicles in Unknown Semi-structured Environments,” *Int J Rob Res*, vol. 29, no. 5, pp. 485–501, 2010, doi: 10.1177/0278364909359210.

[10] K. Chu, M. Lee, and M. Sunwoo, “Local path planning for off-road autonomous driving with avoidance of static obstacles,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 4, pp. 1599–1616, 2012, doi: 10.1109/TITS.2012.2198214.

[11] H. Guo, D. Cao, H. Chen, Z. Sun, and Y. Hu, “Model predictive path following control for autonomous cars considering a measurable disturbance: Implementation, testing, and verification,” *Mech Syst Signal Process*, vol. 118, pp. 41–60, Mar. 2019, doi: 10.1016/J.YMSSP.2018.08.028.

[12] Stephen Maransky and Sean Brennan, “Dead Reckoning by Path Averaging in an Instrumented Small-Scale Test Vehicle,” Thesis, The Pennsylvania State University, 2022.

[13] E. Abbott and D. Powell, “Land-vehicle navigation using GPS,” *Proceedings of the IEEE*, vol. 87, no. 1, pp. 145–162, 1999, doi: 10.1109/5.736347.

[14] S. Clark and H. Durrant-Whyte, “Autonomous land vehicle navigation using millimeter wave radar,” *Proc IEEE Int Conf Robot Autom, Leuven*, vol. 4, pp. 3697–3702, 1998, doi: 10.1109/ROBOT.1998.681411.

[15] O. Mayuku, B. W. Surgenor, and J. A. Marshall, “A Self-Supervised Near-to-Far Approach for Terrain-Adaptive Off-Road Autonomous Driving,” *Proc IEEE Int Conf Robot Autom, Xi’an,* vol. 2021-May, pp. 14054–14060, 2021, doi: 10.1109/ICRA48506.2021.9562029.

[16] A. Jayaraman, A. Micks, and E. Gross, “Creating 3D Virtual Driving Environments for Simulation-Aided Development of Autonomous Driving and Active Safety,” *SAE Technical Papers*, vol. 2017-March, no. March, Mar. 2017, doi: 10.4271/2017-01-0107.

[17] B. Goldfain *et al.*, “AutoRally: An Open Platform for Aggressive Autonomous Driving,” *IEEE Control Syst*, vol. 39, no. 1, pp. 26–55, Feb. 2019, doi: 10.1109/MCS.2018.2876958.

[18] J. Choi *et al.*, “Environment-detection-and-mapping algorithm for autonomous driving in rural or off-road environment,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, no. 2, pp. 974–982, 2012, doi: 10.1109/TITS.2011.2179802.

[19] D. A. Bristow, M. Tharayil, and A.G Alleyne, “A survey of iterative learning control,” *IEEE Control Syst*, vol. 26, no. 3, pp. 96–114, Jun. 2006, doi: 10.1109/MCS.2006.1636313.

[20] H. S. Ahn, Y. Q. Chen, and K. L. Moore, “Iterative learning control: Brief survey and categorization,” *IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews*, vol. 37, no. 6, pp. 1099–1121, Nov. 2007, doi: 10.1109/TSMCC.2007.905759.

[21] “1/5 DBXL-E 2.0 4X4 Desert Buggy Brushless RTR with Smart, FoxBLACK | Losi.” Accessed: Mar. 03, 2024. [Online]. Available: https://www.losi.com/product/1-5-dbxl-e-2.0-4x4-desert-buggy-brushless-rtr-with-smart-fox/LOS05020V2T1.html

[22] “Spektrum SMART 14.8V 5000mAh 4S 100C Smart Hardcase LiPo Battery: IC5 | Horizon Hobby.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.horizonhobby.com/product/14.8v-5000mah-4s-100c-smart-hardcase-lipo-battery-ic5/SPMX50004S100H5.html

[23] “Lithium Ion Battery - 2Ah - PRT-13855 - SparkFun Electronics.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.sparkfun.com/products/13855

[24] M. Delattre, “Off-Road Autonomous Path Following In An Instrumented Smallscale Test Vehicle,” Undergraduate Thesis, The Pennsylvania State University Schreyer Honors College, State College, PA, 2023.

[25] A. Broekman and P. J. Gräbe, “A low-cost, mobile real-time kinematic geolocation service for engineering and research applications,” *HardwareX*, vol. 10, p. e00203, Oct. 2021, doi: 10.1016/J.OHX.2021.E00203).

[26] “GNSS L1/L2 Multi-Band Magnetic Mount Antenna - 5m (SMA) - GPS-15192 - SparkFun Electronics.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.sparkfun.com/products/15192

[27] “SiK Telemetry Radio V3 - 915MHz, 100mW - WRL-19032 - SparkFun Electronics.” Accessed: Mar. 26, 2024. [Online]. Available: https://www.sparkfun.com/products/19032

[28] “Teensy 4.1 without Ethernet - DEV-20359 - SparkFun Electronics.” Accessed: Mar. 18, 2024. [Online]. Available: https://www.sparkfun.com/products/20359

[29] “Wireless E-Stop System - Multiple Transmitter Commanding Multiple Receivers.” Accessed: Mar. 26, 2024. [Online]. Available: https://kar-tech.com/purpose-built-systems/wireless-e-stop/wireless-estop-system-multiple-receiver.html

[30] R. J. Li and Z. Z. Han, “Survey of iterative learning control,” *Kongzhi yu Juece/Control and Decision*, vol. 20, no. 9, pp. 961–966, Sep. 2005, doi: 10.1109/MCS.2006.1636313.

ACADEMIC VITA

Andres Enrique Esparragoza
andespar15@gmail.com | aee5212@psu.edu

EDUCATION

The Pennsylvania State University (August 2020 - May 2024)
Schreyer Honors College, College of Engineering, Bachelor of Science in Mechanical Engineering
Dean's List | The President's Freshman Award | Evan Pugh Junior Scholar Award

WORK EXPERIENCE

John Deere (June 2022 -August 2022)
Product Engineer Intern - 5000 Series Vehicle Design Engineer - Chassis Design
- Used **Creo** to design, create, and analyze chassis components to fulfill customer needs
- Used **Creo** and **CreoView** to determine design changes necessary for high trim front axles on 5M Tractors
- Partook in 4 **CAD** design projects to enhance future 5M Tractor and assist the 5000 Series team

Delta Air Lines (January 2022 - August 2023)
Component Engineer Co-Op - Landing Gear Engineer
- Saved over $11 Million by designing & authoring repairs for over **200 out-of-scope landing gear components**
- Aided the Mechanical Systems Shop, Wheel & Brake Shop, and Landing Gear Shop for cost effective solutions
- Used **SQL** to create, edit, and transfer the Landing Gear database to streamline workflow
- Created Data based solutions to preclude repetitive work scopes from entering shop

Engineering Entrepreneurship Consulting Internship (October 2021 - May 2023)
Entrepreneurship Intern - Penn State EN-tern Program
- Marketed the Engineering Entrepreneurship Dept. using student to student engagement
- Enhanced the dept campus presence by leading engagement activities

Goza Engineering Design (September 2021- May 2022)
Founder & Consultant
- Consulted local businesses on engineering design decisions and tutored engineering students on **CAD**
- Used **SolidWorks, OnShape,** and the **Ender 3 Pro 3D Printer** to create parts for clients

RESEARCH EXPERIENCE

High Speed Autonomous Vehicle Research | *Programming, Design, Mechatronics* (January 2023 - May 2024)
- **Programmed** RC vehicle to follow ILC and RC commands of intelligent autonomous motion on off road terrain
- **Simulated** advanced ILC implementation for velocity optimization for path following
- Designed electronic packaging for **teensy**, **arduino**, and **Encoder** application to automate a scaled vehicle
- Wrote technical thesis on my research regarding **off road high speed autonomous vehicle**s

Drone Research Project | *Design, Iteration,* (June 2019 - August 2019)
- Designed and tested custom functioning drones with **3D printers** and **SolidWorks**
- **Manufactured** 10 different drones with 3 completing a level of sufficient controlled flying
- Adapted the drones to begin surveying land with camera technology

ADDITIONAL ACTIVITIES

Formula SAE (August 2021 - May 2022)
- Designed, tested, and created aerodynamic and structural chassis components on **SolidWorks** for the vehicle
- Collaborated with the aero and suspension subsystem to create a monocoque to withstand vehicular torsion

Engineering Ambassadors (August 2021 - Present)
- Support and inspire underrepresented and economically disadvantaged groups in engineering
- Present and communicate Engineering topics to the next generation of engineers

Society of Hispanic Professional Engineers (September 2021 - Present)
- Network and inspire with engineers of my culture

RELATED SKILLS

SolidWorks, CATIA V6, MATLAB, 3D Printing, CAD Modeling, Excel, GD&T, AutoCAD, FEA, CREO, CreoView

LANGUAGES

English, Spanish, French, Italian