2022

Off Road Autonomous Vehicle Modeling and Repeatability Using Real World Telemetry via Simulation

Matthew Paul Spencer

OFF ROAD AUTONOMOUS VEHICLE MODELING AND REPEATABILITY USING REAL WORLD TELEMETRY VIA SIMULATION

By

Matthew P. Spencer

A THESIS

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical and Computer Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2022

This thesis has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical and Computer Engineering.

Department of Electrical and Computer Engineering

Thesis Advisor:	*Dr. Jeremy P. Bos*
Committee Member:	*Dr. Michael C. Roggemann*
Committee Member:	*Dr. Aurenice M. Oliveira*
Department Chair:	*Dr. Glen E. Archer*

Dedication

To Jesus Christ, family, friends, and Grandma Spencer

whose support and love fuels my work each day.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to specifically acknowledge Derek Chopp of the Keweenaw Research Center for building the Blender script used in this experiment to initially parse the point cloud terrain data. I would also like to acknowledge all the students and full-time employees at the Keweenaw Research Center for their work in data acquisition regarding the terrain point cloud, texture map, and HMMWV telemetry.

A large amount of gratitude is also owed to the university and professors who continue to impart on me the required wisdom and knowledge to do research work and further contribute to the field of electrical and computer engineering.

Definitions

α	Point cloud scaling factor
d	Set of saved path coordinates
d_ϕ	Look-ahead time for Pure Pursuit controller
δ	Steering angle applied to vehicle
δ_{turn}	Calculated average steering angle for the vehicle
f_{sim}	Set frequency of the executing simulation run
f_{trace}	Frequency of the recorded GPS and IMU trace data
Hm	Height-map file set
k_ϕ	Number of time-steps used for finding the minimum look-ahead value
k_t	Number of time-steps used in the averaged steering angle calculation
L	Height-map Tile
m	Current (x,y) position the vehicle
N_p	Total number of coordinates in current path
N_t	Total number of points in KRC point cloud
N_{tiles}	Total number of tiles to be created by UE4 KRC Terrain Plugin
P	Padded destination area
ϕ	Look-ahead time for Pure Pursuit controller
$\phi_{\max}$	Maximum look ahead time allowed

$\phi_{\min}$	Minimum look ahead time allowed
s	Total time of 1 time-step in seconds
Tx	Texture-map file set
X	KRC point cloud NAD83 easting coordinates
X_L	UE4 virtual X tile position
Xp	Current vehicle path NAD83 easting coordinates
x	Zeroed KRC point cloud NAD83 easting Coordinates
xp	Zeroed vehicle path NAD83 easting Coordinates
Y	KRC Point Cloud NAD83 northing coordinates
Y_L	UE4 virtual Y tile position
Yp	Current vehicle path NAD83 northing coordinates
y	Zeroed KRC point cloud NAD83 northing Coordinates
yp	Zeroed vehicle path NAD83 northing Coordinates

List of Abbreviations

API	Application Programming Interface
AUV	Autonomous Underwater Vehicles
AV	Autonomous Vehicle
CPU	Central Processing Unit
CSV	Comma Separated Values
CTE	Cross-Track Error
DLC	Double Lane Change
DNF	Did Not Finish
FPS	Frames per Second
GPS	Global Positioning System
GPU	Graphics Processing Unit
HMMWV	High Mobility Multipurpose Wheeled Vehicle
IMU	Inertial Measuring Unit
KRC	Keweenaw Research Center
LiDAR	Light Detection and Ranging
MUONS	Michigan tech Unstructured and Off-road Navigation Stack
OS	Operating System
PID	Proportional Integral Derivative

RAM	Random Access Memory
SLAM	Simultaneous Localization and Mapping
SS	Step Steer
UE4	Unreal Engine 4

Abstract

One approach to autonomous control of high mobility ground vehicle platforms operating on challenging terrain is with the use of predictive simulation. Using a simulated or virtual world, an autonomous system can optimize use of its control systems by predicting interaction between the vehicle and ground as well as the vehicle actuator state. Such a simulation allows the platform to assess multiple possible scenarios before attempting to execute a path. Physically realistic simulations covering all of these domains are currently computationally expensive, and are unable to provide fast execution times when assessing each individual scenario due to the use of high simulation frequencies (> 1000Hz).

This work evaluates using an Unreal Engine 4 vehicle model and virtual environment, leveraging its underlying PhysX library to build a simple unmanned vehicle platform. The simulation is demonstrated to run at low simulation frequencies (< 1000Hz) when performing multiple off road driving maneuvers. Real world path telemetry is used as input to drive the unmanned vehicle's integrated Pure Pursuit and PID autonomous driving control algorithms within the simulation. Cross-track-error and vehicle heading error between the simulation and real world telemetry is then observed after each maneuver's execution.

It is concluded after running multiple different vehicle maneuvers in real time at low simulation frequencies, a lower threshold frequency of 190Hz was shown to reliably control the virtual vehicle model with minimal average cross-track-error and heading angle deviation. Higher simulation frequencies approaching 400Hz, the recorded sampling frequency of the real world telemetry for each maneuver, had little change in system performance. Setting the simulation to execute at lower frequencies < 190Hz resulted in a point of exponential increase in both the overall average cross-track-error and heading error. Additional simulation failures were also observed when setting the AV to travel at higher velocities with set simulation frequencies < 190Hz.

Chapter 1

Introduction

This chapter details the problem and motivation for the thesis along with a brief analysis of its results. A quick discussion of current work being performed in the field of autonomous off road vehicle simulations is presented, and past tools and systems are explained.

1.1 Overview

As the age of commercial and private autonomous vehicles looms over the horizon, many hours of research have been poured into a myriad of aspects relating to the AV industry. Governments and corporations over the past few decades have put significant effort into improving the capability of AV artificial intelligence, cameras, and sensor systems to reliably guide AVs through challenging terrain (e.g. mud, fields, dirt roads, etc.) regarding unstructured off road environments [33] [34]. AVs operating in these environments cannot rely on pre-existing road signs, speed regulations, and detailed maps to successfully traverse to their destinations.

The applications that could utilize off road AVs are diverse, adding a substantial amount of variability to an already complicated task of structured on-road AV environments which can utilize pre-existing landmarks, signs, and information while in operation [1]. Current methods involving path planning and object detection being used to achieve successful off road AV traversal are quickly changing and improving [35], establishing the need for faster and cheaper ways of testing the implementation of newer methods. Computer vision algorithms used in Simultaneous Localization and Mapping (SLAM) techniques implementing modern camera and LiDAR technologies are being thoroughly explored to test AV operations in real time [10]. Simulations

being utilized for these purposes are growing within the fields of digital image processing and artificial neural networks [12]. The incorporation of large data sets generated either within or outside of these simulations for the purpose of real world evaluation is becoming a specific key area of research and investment [2].

Many well known 3D computer simulation tools and game engines such as ANVEL, Car-Sim, Unreal Engine, Gazebo, Unity, etc. [3] have been assessed to provide data precise enough to model AVs and test various autonomous scenarios. Most simulation systems are layered, requiring a federated approach incorporating multiple unique simulations specializing in different types of physics-based models running in parallel with visual modeling to achieve desirable results [4]. Federated systems coupled with high simulation frequencies leave legacy computer simulation architecture to require heavy computing power [9]. This generates long computational wait times between executing each step of the simulation, voiding their use within time-dependent real time applications. With the recent advancements in modern computers and technology, researchers have been pushing against these limitations by utilizing popular open-source software libraries to achieve realistic virtual worlds and simulation environments using a single simulation platform such as UE4 [6].

One way to speed up the execution time of a single simulation platform would be to decrease its simulation frequency. The set simulation frequency is also known as the simulation's frames-per-second (FPS) [45]. After the engine computes the needed

underlying physics and graphical calculations for the next time step of the simulation, the next image (frame) of the simulation is outputted. This process is called the "rendering pipeline" [46]. The less calculations the simulation has to do (lowering the simulation frequency), the faster it performs in real time. The more calculations the simulation has to do (raising the simulation frequency), the slower the simulation will perform in real time. Higher simulation frequencies result in smoother and more accurate simulated results, which is why high simulation frequencies are chosen for modern AV simulation solutions. Lowering the simulation frequency is desirable for more time-pressed applications such as path planning though, where a greater number of paths need to be calculated preferably faster than real time.

Real world trace path telemetry for each maneuver is generated by capturing synchronized GPS and IMU trace data sampled at 400 Hz from a real world High Mobility Multipurpose Wheeled Vehicle (HMMWV) platform. The HMMWV platform was used due to the accuracy of the supplied virtual model measurements compared to a real HMMWV provided by the Keweenaw Research Center (KRC). A human driver performs each test maneuver within the environment that the virtual world is modeled to replicate. The UE4 game engine is programmed using C++ to control a virtual HMMWV vehicle model, recreating each driving maneuver within the simulated environment using the trace telemetry as input. The programming language C++ is used as a result of UE4 being natively built to interface with multiple C-based graphic API libraries such as OpenGL and Vulkan for better performance [42] [43] [44]. A

modeled AV driver uses an integrated Pure Pursuit and PID controller to steer the vehicle along each path, moving along the specified coordinates at the desired target velocity [17].

The UE4 simulation telemetry is recorded at a sampling rate corresponding to the set simulation frequency during each execution. After each simulated driving test, the recorded simulation telemetry is then compared to the original trace path telemetry. Total CTE and heading angle deviation for each maneuver is then averaged and plotted over the tested set of simulation frequencies. Comparisons between the average CTE and heading angle error against the real telemetry is done to see if lower simulation frequencies (< 1000Hz) can be deemed reliable, meeting a CTE and heading error as close to 0 as possible. Simulation step frequency is lowered for each test until a point of failure is observed and the current driving maneuver cannot be performed by the modeled HMMWV AV.

In the following chapters, procedures for importing a real world environment into UE4 using colored geo-referenced point cloud data is explained. The KRC's outdoor vehicle test course was used to model the real-to-virtual UE4 world environment due to the high resolution geo-located point cloud data they provided[1]. The HMMWV model and its UE4 vehicle dynamic properties is then described along with the virtual AV driver using the UE4 Pure Pursuit and PID control C++ implementations.

[1]The KRC supplied the HMMWV model, geo-located 3D point cloud data, and the real HMMWV path telemetry used to drive the modeled AV for this thesis.

1.2 Problem Statement

By integrating common AV velocity and steering control algorithms within a single visual and physics computer simulation platform, a low simulation frequency needs to be found that demonstrates reliable CTE and heading angle error performance between the recorded path telemetry of a real vehicle platform and its virtual counterpart for effective path planning in real time. In order to understand the spatial and temporal resolution regarding modeled off road AV control within a single simulation platform using real world telemetry, this work evaluates multiple off road vehicle maneuvers performed by a real HMMWV platform [5] within the simulation. This thesis then demonstrates a low overall average CTE (< 6.87cm) and overall average heading angle error (< 4.89 degrees) of a single simulation platform running at low simulation frequencies (< 1000Hz) when modeling multiple real AV off road path following maneuvers.

Simulation work has been the focus for researching off road AV applications involving isolated individual sensors and algorithms to be incorporated within real hardware separately at a later date [36] [37]. This isolation is due to the uncertainty and challenge of simulating the entire system of real vehicle dynamics involved with AVs operating in unstructured off road environments [32]. One of the main downsides of

many existing simulation architectures for off road simulation is their heavy computational cost and reliance upon co-simulation [28]. These simulations are also known to be computationally expensive [26], utilizing multiple interconnecting complex physics systems and graphical libraries to complete a single data point experiment using a high simulation frequency to achieve desirable results [9].

Modeled AV path planning simulations that provide reliable CTE results exhibit evidence that the AV system can reach its target destination when following a selected path without accident [30]. CTE has commonly been used as a benchmark for path planning performance [31] and as an active input for path following control algorithms (e.g. Stanley method) [47]. A common method for operating path finding systems is to quickly analyze multiple possible routes prior to path execution in real time within a replicated simulated world of an AV's environment [3]. By analyzing the CTE and AV's heading angle error between the simulation and real world vehicle, comparisons can be made between the virtual and real world models to observe the simulation's reliability at lowered simulation frequencies.

Real time off road AV path planning simulation systems have been a proof-of-concept affair at the time of writing due to the stated use of a high simulation frequency and co-simulated architecture [38]. Simulations can take hours, or even days, in real time to complete when using modern high-end computing hardware [29]. A single reliable

modern integrated visual and physics platform executing at low simulation frequencies would therefore be beneficial when simulating real AV off road path following maneuvers. This eliminates the need for co-simulation between multiple platforms running at high simulation frequencies, and eliminates added computational costs.

A recent build of Unreal Engine 4 (UE4) game engine with its underlying PhysX architecture [13] is used in this thesis as the integrated visual and physics computer system for the AV simulation model. Telemetry gathered from a real vehicle platform is used as input to drive the virtual AV controllers within a modeled real world environment. The replicated outdoor virtual world allows performance comparisons between the simulation's path following accuracy to that of a vehicle platform in the real world. A low average CTE coupled with low heading error deviations indicates that the simulated vehicle model is able to execute the same path taken by the real world vehicle [49].

The results at the end of this experiment analyze a single virtual UE4 HMMWV model configured to run at low simulation frequencies. The simulation is programmed to stay within a chosen (1m x 1m) path waypoint CTE threshold during path execution. This $1m^2$ padding was chosen to be less than the horizontal HMMWV width between the front or rear tires.

After all tested paths are executed within the simulation, average CTE and heading error is discussed and compared to the real world path telemetry. The data is

quantiled and a two-term Gaussian fit-line applied. CTE Gaussian analysis is used commonly when measuring performance of AV algorithm control [48] [49], where a low CTE close to 0 is desired to indicate that the AV stayed near the target input path. A low heading error also indicates stabilization between the simulation environment physics, e.g. gravity, the AV's simulated controllers and the model's actuator state [50].

The end results demonstrate that future path following AV systems relying on a single simulation platform like UE4 can execute potential routes for real AV platforms running at lower simulation frequencies, resulting in low overall average CTE and heading error. Frequencies $<$ 190Hz were observed to have exponentially growing average CTE and heading error, further evidenced by an exponential rise in the slope of the Gaussian best-fit-line assigned to each error data set. Simulation failure also became more common when executing at higher speeds during more complex off road vehicle maneuvers. Setting the simulation frequency $>$ 190Hz, approaching the original input telemetry sampling frequency of 400Hz, produced minimal average CTE and heading error by comparison. Higher simulation frequencies resulted in simulations that were able to follow the supplied input path with greater accuracy compared to simulations set $<$ 190Hz.

Chapter 2

Background

This chapter discusses background information on building blocks used to set up the foundation of the following computer simulation path following experiment. Previous literature is discussed along with differences between this research and aforementioned literature. The simulation software and framework is explained in context with off road autonomous vehicle research, along with the justification of using a Pure Pursuit controller for the virtual AV model.

2.1 Unreal Engine 4

UE4 is currently used as a visual simulation tool within the public and private AV research and development community [25] [26]. There still remain questions about the robustness of its deterministic performance regarding the underlying physics implementation, such as object hit detection or computation lag [2], but its popular use among video game developers provides a highly populated online community which regularly gives feedback and documentation alongside steady UE4 engine updates [13]. The source code is also made publicly available by the developers, allowing programmers with a good understanding of C++ and the necessary skills to develop engine specific tools and plugins to achieve specific simulation related goals [14]. The underlying physics framework, PhysX, is also well known and documented [24], offsetting some of the prior concerns when compared to other simulation tools such as "MuJo" [4].

Much work has been done experimenting in creating simulation scenarios of real world events created specifically for the UE4 game engine. Environments can be hand-made and created to represent believable scenarios to study various autonomous technologies involving neural network training [10], pedestrian traffic avoidance [11] [27], specific LiDAR SLAM techniques [12], and many more. The need and interest is therefore outgrowing the pace at which AV path planning and safety systems are

being developed.

A recent UE4 build (v4.26) with the default UE4 mid-sized template vehicle model has been shown to autonomously drive within a virtual environment using the MUONS path planning stack developed at Michigan Technological University. It was determined that UE4 coupled with MUONS handled mid-sized vehicles with complex suspension systems well when compared to using a simpler HUSKY robot platform [15]. MUONS can also virtually incorporate the robot operating system (ROS) libraries in order to publish and subscribe to virtual sensors required to path plan adequately within complex unstructured environments, further indicating UE4's ability to be modified as needed. The environment created for the latter MUONS experiment was generated using a portion of real world point cloud data taken from the KRC's outdoor test course [16]. This emphasizes the value of the KRC course for correlating further real world to virtual world testing path planning experiments.

For past UE4 experiments mentioned in this section, all telemetry for path planning control and repeatable algorithm testing was created virtually inside of the simulation. Paths generated for an AV platform were executed virtually within the simulation, with no guarantee that the path could be executed on a real AV vehicle. This solely virtual limitation does not allow correlation between the virtual AV's predictive reliability and the real world vehicle platform it is trying to simulate. This thesis implements this additional piece of real world information to make an evaluation on

the repeatability when modeling potential AV scenarios using UE4 and PhysX.

2.2 Previous Simulation Work

Computer simulations to visually model AVs and their attempts at solving problems associated with real time path following date back to the early 1990s, when the United States Navy funded research into potential solutions for simulating autonomous underwater vehicles (AUVs) [8]. It was shown that in order to achieve a complete 3D visual representation of the simulation, many computers had to be networked together to provide a successful AUV simulation. Although the complex simulation architecture is crude by today's standards, an elegant solution to finding a reliable simulation to model fast and comprehensive real time AV path planning scenarios is still being sought after today [2].

Simulations are not only being developed in regards to AUVs and grounded AVs, but are also being developed within the field of aerial drone autonomy. In 2018, a drone flight controller was modeled and integrated within a virtually controlled drone, which was tested in a published study utilizing a UE4 C++ plugin called AirSim [9]. The UE4 simulation was fed prerecorded virtual world coordinate telemetry for path following, showing proof-of-concept for potential real world applications. The simulation frequency of the UE4 engine when sampling the drone path data

was set to be 1000 Hz, providing waypoint path data 1ms apart from each point for playback within the AirSim controller. Lower threshold values regarding the simulation frequency rate were not considered, nor any attempt at testing the system on a real aerial AV platform outside of the simulation. These questions are brought to the forefront for this thesis which uses real telemetry, and analyzes lower UE4 simulation frequencies within the context of AV simulation.

A real world example of a modeled AV UE4 simulation interacting with a real AV has been demonstrated to be a challenging endeavour as of late 2021 [38]. A simple two-way road was modeled within UE4 representing a real two-way AV test site. The experiment used two virtual vehicles placed on the simulated road facing the same direction as each other. One of the simulated vehicles was controlled by a real AV operating on the actual test course, and the other simulated vehicle was controlled by a real user equipped with a driving wheel controller connected to the simulation. The purpose of this experiment was to test the real AV's crash avoidance system by having the human controlled virtual vehicle cut into the AV's lane. This was done in order to find a solution to prevent the risks involved when testing an AV safety system in real life. The experiment proved that this virtual-to-real AV interaction was feasible, as the real AV was shown to be able to respond to the virtual vehicle and its simulated environment. However, the response from the real AV to avoid the simulated vehicle was found not reliable or accurate enough for use in testing actual AV safety systems.

Other 3D simulation platforms that are similar to open source environments such as UE4 are being used in the field as well. One example is the simulation software Vortex, which has been used within the last year to test off road vehicle wheel-soil interaction models [33]. This software has shown to be promising in isolated scenarios, but does not have the support of a wider development community and consistent updates from the first-party developers that exist on the UE4 platform. For this reason, UE4 was chosen for the thesis experiment instead of other modern simulation solutions such as Vortex.

What is not addressed by these experiments is the real computation time needed when running each simulation at a set simulation frequency, and how the simulation frequency impacts the simulation's overall performance. AV models have been created within UE4 and similar platforms [32], virtual data has been programmed to be used as an input source during execution [9], and real world AVs have been shown to be able to react to virtual input data [38], but to draw further conclusions regarding repeatability of simulated models within the real world, a simulation would also need to use real world data outside of the virtual environment. This thesis sets up the experiment and process to use UE4 alongside real world data to analyze AV simulation performance at different simulation frequencies.

2.3 Pure Pursuit

Pure Pursuit is an established autonomous steering control method, wherein the correct steering angle needed for the vehicle to move towards the next look-ahead waypoint is calculated [17]. The curvature between the current vehicle position and waypoint position is found, thus setting the vehicle's needed steering angle on this found curvature [21]. The calculated steering angle is set to achieve a more "human-like" smooth response compared to more simplistic alternatives such as the Carrot Chasing control algorithm [23], which continually snaps the steering angle to face the next waypoint at a fixed look-ahead distance.

Pure Pursuit's ease in implementation while retaining the "human-like" driving characteristics was preferred over other more complicated path following algorithms such as the Stanley method, which involves calculating CTE and desired yaw rate to feed into the controller during operation for lateral stability [22].

A visualization of the Pure Pursuit implementation for this thesis's experiment within UE4 can be seen in Fig. 2.1 [17]. The yellow line represents the look ahead vector to the next waypoint marked with a red circle, and the green line is the calculated path curve needed to be taken.

More modern methods to incorporate lateral and velocity input control have been

integrated into the underlying Pure Pursuit algorithm for use on real world vehicles to find optimal look-ahead values. This is done by smoothing out turns by averaging angles needed to turn the vehicle to reach its waypoint destination [18]. The Pure Pursuit implementation in this thesis relies on the additional lateral control combined with engine specific UE4 functions, wherein the look-ahead vector is shortened or lengthened depending on the upcoming steering angle needed to turn the vehicle. The look-ahead vector is shortened when approaching a heavy turn for example, and lengthened if following an estimated straight path.

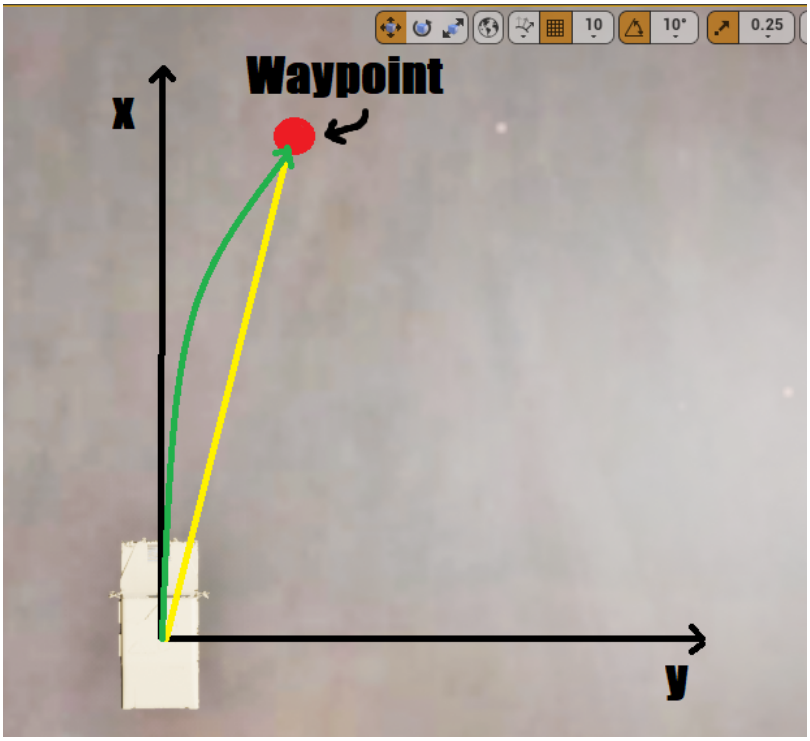

Figure 2.1: Pure Pursuit Example

The following chapter explains the simulation configuration for this experiment in more detail, describing the AV Pure Pursuit control algorithm with a flexible look-ahead vector. Modeling of the HMMWV and KRC test course within UE4 is also discussed, showing the importation process and procedures to align the real world coordinate system with that of the virtually simulated world.

Chapter 3

Simulation Setup

This chapter explains the general overview in setting up the UE4 simulation experiment. The required hardware and software specifications, terrain data importation methods, Pure Pursuit algorithm implementation, and the process for controlling the virtual AV using real telemetry is explained. Each simulated driving test maneuver is also visualized, plotting the paths taken by the real HMMWV during each maneuver.

3.1 Hardware Specifications

The following Table 3.1 outlines relevant computer hardware components used for executing the simulation. The parts used are common and commercially available at the time of writing.

Table 3.1
Hardware Specifications

Type	***Spec***
CPU	Intel i9-7920X 2.9 GHz
RAM	128 GB DDR4
GPU	NVIDIA GeForce RTX 2070 FE
PWR	1000W
OS	Windows 10 64-bit
UE4 version	Unreal Engine 4.26.2

3.2 Terrain Importation

Specialized land surveying equipment was utilized by the KRC to create a geo-located point cloud LiDAR data set of the KRC's outdoor test course. Drone equipment was also used along with geo-located aerial images of the test course. The point cloud and texture data was then combined and transformed into a single .fbx file. This process was done internally at the KRC and provided for use within this thesis. The .fbx file was then imported into the mesh editing software Blender [19].

Each point in the original point cloud (X, Y), of size N_t, is zeroed using its minimum coordinate $(x_{min}\ y_{min})$ to place the origin of the transformed mesh at the (0, 0) location as seen in [eq. (3.1)]. This zeroing process is done to ensure that the terrain is compatible with UE4's world coordinate system, orienting the origin point of the terrain at the simulated world's default virtual origin point, (0,0).

$$(x_i, y_i) = (X_i - x_{min}, Y_i - y_{min}) \quad \text{for} \quad i = 0, 1, 2, ..., N_t \tag{3.1}$$

A Python script utilizing Blender's API is then executed on the mesh, scaling the terrain mesh using a scaling factor α and dividing the mesh into a user selected number of corresponding heightmap and texture tiles seen in Fig. 3.1 and Fig. 3.2. Each tile produced has its heightmap file paired with the overlaid texture image of the mesh [eq. (3.2)]. This scaling was performed to use UE4's native landscape terrain type instead of a single point cloud mesh, trading overall mesh vertex resolution for simulation performance.

$$(X_i, Y_i) = (X_i, Y_i) * \alpha \quad \text{for} \quad i = 0, 1, 2, ..., N_t \tag{3.2}$$

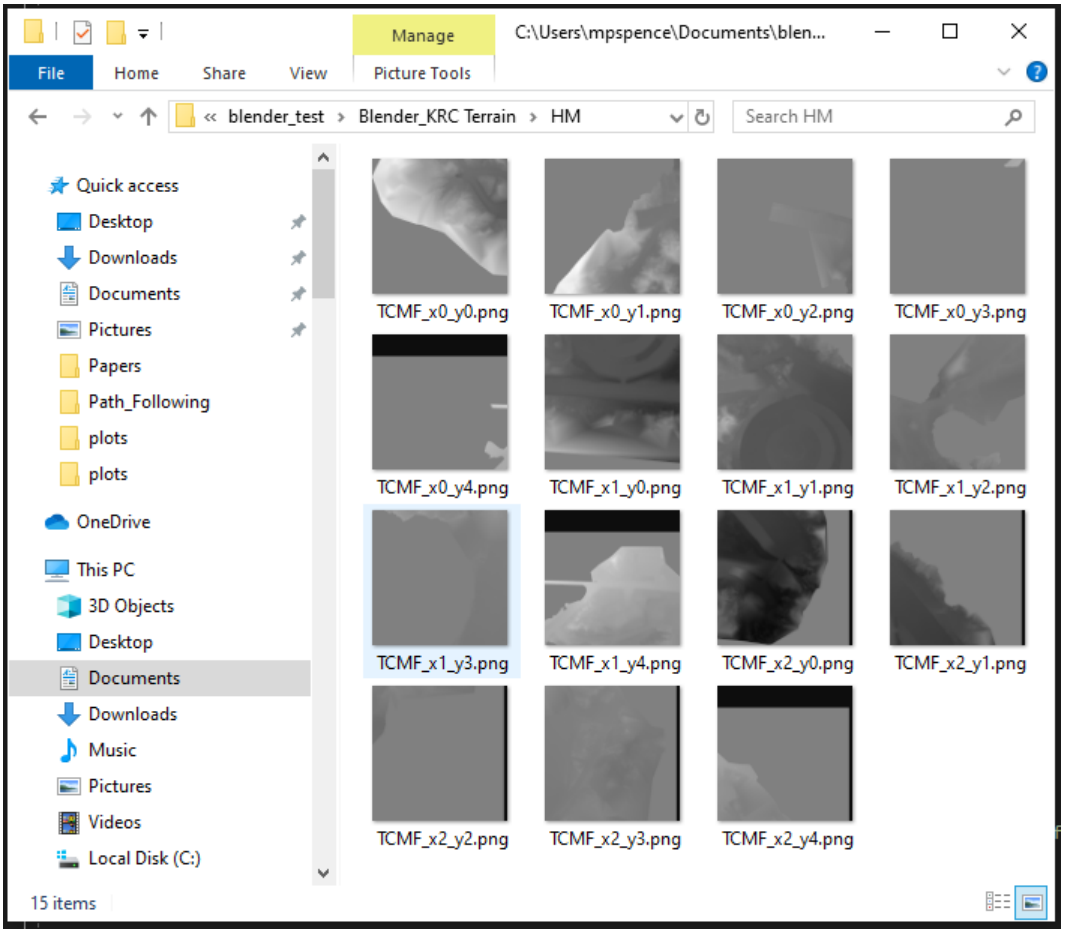

Figure 3.1: Height-map Files

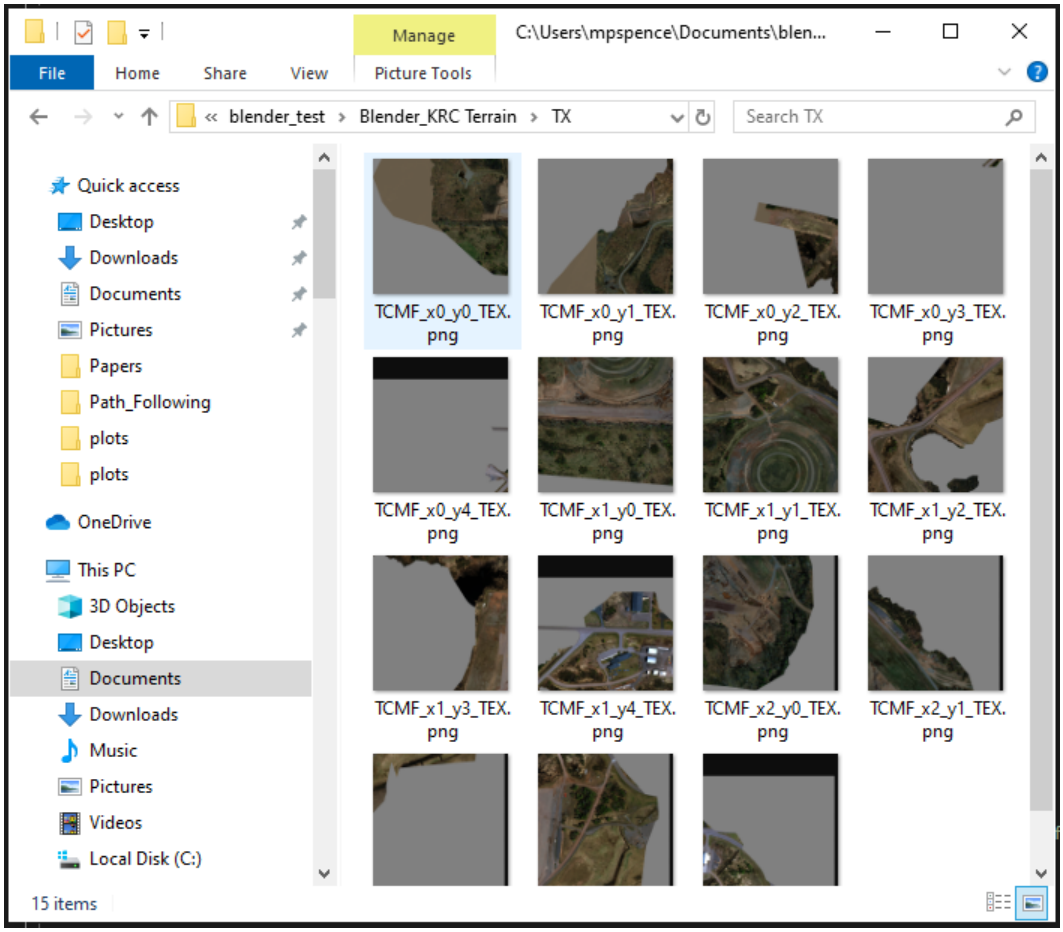

Figure 3.2: Texture Files

The tiles were loaded into UE4 using a custom proprietary C++ plug-in tool "KRC Terrain Creator" built using the UE4 editor API landscape creation functionality and internal engine function library [Fig. 3.3].

Figure 3.3: KRC Terrain Creator UE4 Plugin

The current tile being built has each heightmap pixel and its corresponding pixel texture color loaded into a buffer within the UE4 editor's heightmap landscaping level builder using internal UE4 C++ functions. The tile is then created as a UE4 landscape object and re-scaled and textured internally within UE4 using the plug-in.

Since the original units of the mesh were centimeters, no additional conversion was needed to match UE4's coordinate and unit systems. UE4 uses the metric system as its default measuring system for distance and length [39], where 1 "unreal unit" equates to 1cm. Depending on the tile's location denoted by its (X,Y) filename, it is spawned and placed in its proper location within the virtual world. This process is explained in [Algorithm 1].

Algorithm 1 UE4 KRC Terrain Plugin

$Hm \leftarrow$ Set of heightmap files
$Tx \leftarrow$ Set of texture files
$N_{tiles} \leftarrow$ Number of tiles to be created
$i \leftarrow$ iterator
for $i = 1$ to N_{tiles} **do**
    (X_L, Y_L) $\leftarrow$ Get (x,y) tile location using filename of heightmap file Hm_i
    $L \leftarrow$ UE4::CreateHeightmap(Hm_i, Tx_i) // Create Landscape tile using UE4 engine functions.
    (X_L, Y_L) = (X_L, Y_L) * α + shift // Apply scaling factor and shift tile origin location to proper placement.
    UE4::LoadTile(L, X_L, Y_L) // Load Tile into UE4 world at shifted position using UE4 editor functions.
end for

A flowchart explaining the terrain importation workflow is outlined in Fig. 3.4, showing a high-level overview of the point cloud and texture data importation process.

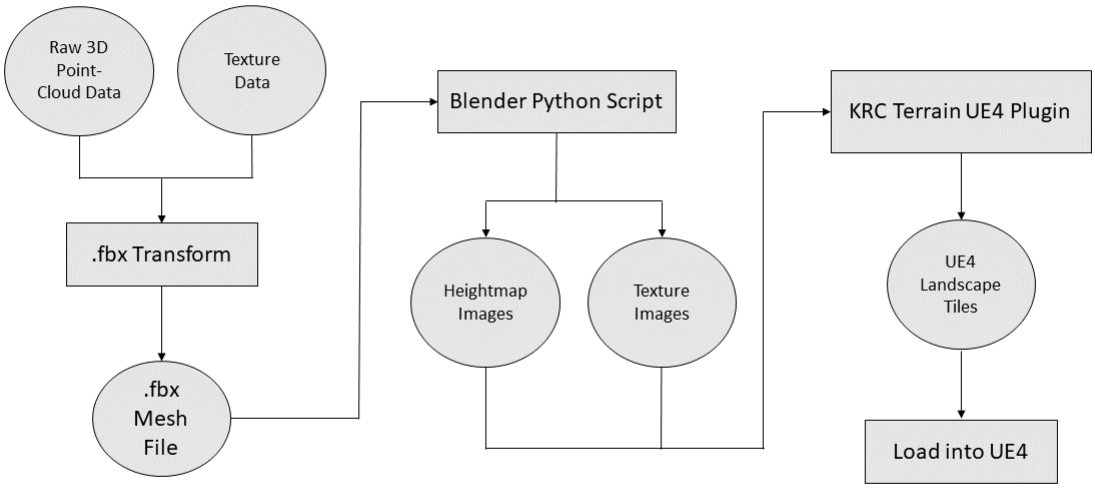

Figure 3.4: UE4 Terrain Importation Workflow

The resulting fully generated terrain landscape shown within the UE4 editor window is displayed in Fig. 3.5.

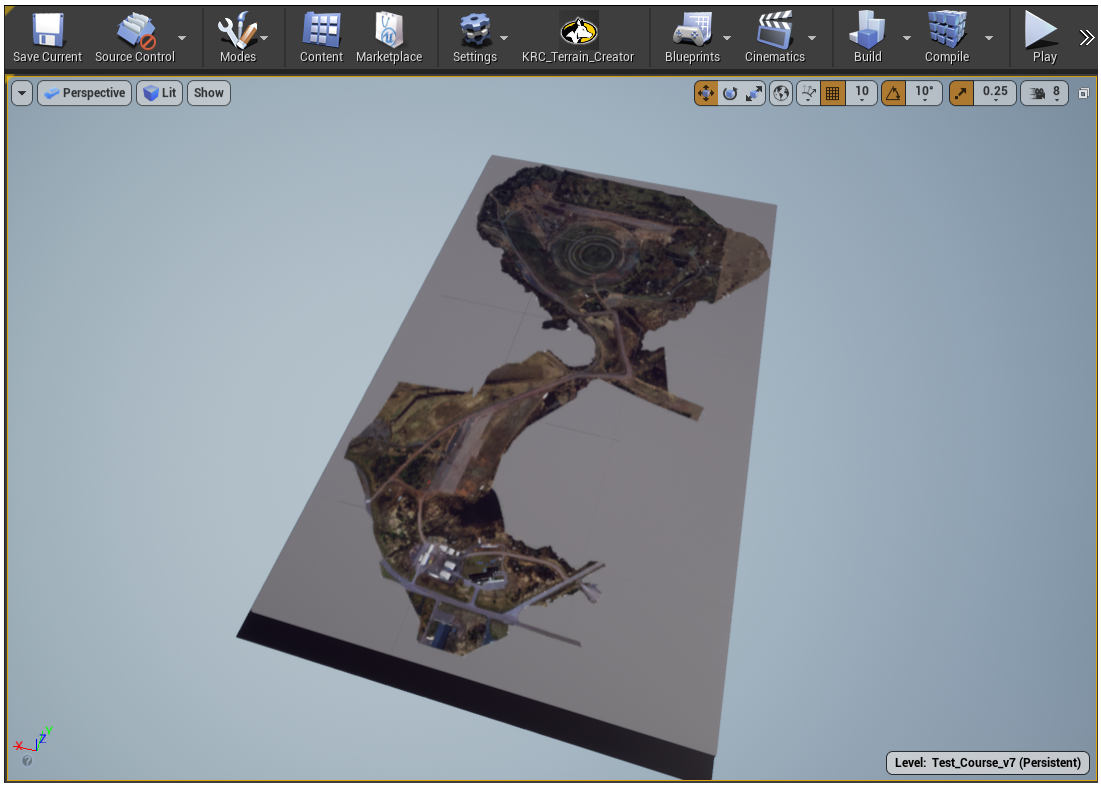

Figure 3.5: KRC UE4 Landscape

3.3 Vehicle Model

A HMMWV vehicle model supplied by the KRC was imported into UE4 [Fig. 3.6] and configured based on the default advanced vehicle template supplied by the engine. Basic UE4 vehicle parameters were tuned alongside default UE4 vehicle model values to correlate with real HMMWV properties as seen in Table 3.2.. The KRC requested that specific information be redacted regarding the supplied HMMWV vehicle due to its active use in the military.[1].

Table 3.2
HMMWV UE4 Model Parameters

Parameter	***Setting***
Vehicle Mass	[redacted] kg
Center of Mass	([redacted], [redacted]) cm
Wheel Base Length	[redacted] cm
Wheel Base Width	[redacted] cm
Wheel Radius	[redacted] cm
Wheel Width	[redacted] cm
Wheel Mass	[redacted] kg
Wheel Max Lat Stiff	2.0
Wheel Lat Stiff	17.0
Wheel Long Stiff	1000
Wheel Friction Scale	1.0

[1]Any HMMVW-style vehicle model using the default UE4 PhysX vehicle template can be used to recreate the experiment presented in this thesis.

Figure 3.6: HMMWV UE4 Model

UE4 vehicle values are directly tied to the PhysX model represented underneath the visual mesh, calculating the physics interactions between objects in the virtual world (such as tire-soil interaction)[40]. The focus of this experiment was not to measure specific physical vehicle dynamic interactions, but rather to focus on the performance of off road path following maneuvers using estimated UE4 vehicle physics property values assigned to the model during simulation. [Table 3.2].

3.4 Real-World Telemetry

A real HMMWV was equipped with GPS and IMU sensors to record a driver performing multiple different maneuvers at different speeds within an off-road environment. The data was recorded at rate of 400Hz, time synchronized across the necessary GPS

and IMU devices. The resulting output data is processed into a MATLAB .mat file format, which was then converted into a CSV file for use as input for the UE4 simulation. 400Hz was chosen due to the need for additional simultaneous testing for an unrelated HMMWV's on-board controller needing a 400Hz signal refresh rate.

The KRC test course point cloud data was created in the NAD83 Northing and Easting coordinate system, while the GPS data for the HMMWV was captured in the WGS84 latitude and longitude coordinate system. Thus a conversion was made on each coordinate, utilizing the "PROJ" C++ library [20] and an up-to-date packaged transform database, to convert WGS84 coordinates into required NAD83 coordinates for use as waypoints within the simulation [Fig. 3.7].

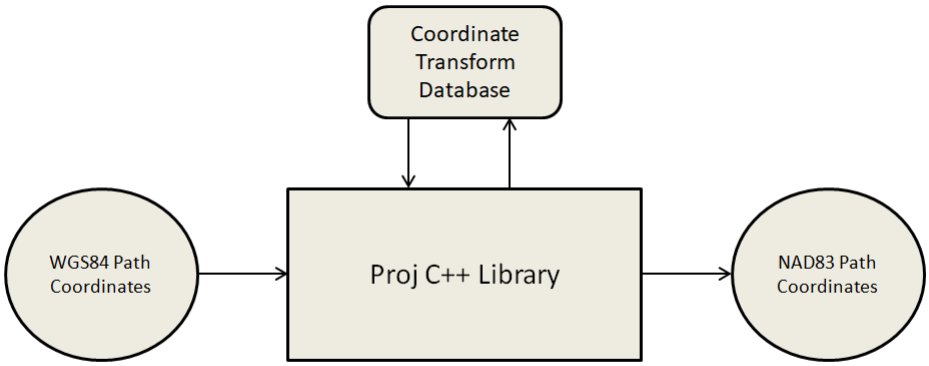

Figure 3.7: Coordinate Transform

To match up the total number of path coordinates, N_p, with the virtual world's origin

point at (0,0), each path coordinate (Xp, Yp) is re-zeroed and scaled with the same process used when importing the terrain seen in [eq. (3.3)].

$$(xp_i, yp_i) = (Xp_i - x_{min}, Yp_i - y_{min}) * \alpha \quad \text{for} \quad i = 0, 1, 2, ..., N_p \tag{3.3}$$

Velocity of the vehicle model in UE4 is controlled by a hand-tuned C++ PID controller, which adjusts the acceleration of the virtual vehicle until the recorded velocity is reached, as indicated by the real world trace telemetry at the given time step. This creates a virtual "cruise control" that adjusts the speed as needed. Default UE4 PhysX brake and engine torque is used to apply acceleration and brake pressure to meet the target velocity. The PID values used for the acceleration controller are shown in Table 3.3[2].

Table 3.3
PID Values

Variable	***Value***
P	6.0
I	0.25
D	0.0

[2]Plots showing the PID controller following the vehicle velocity of each maneuver's input path telemetry is presented at the end of this thesis in Appendix A.1.

3.5 Pure Pursuit Implementation

Pure pursuit finds the curvature needed to steer the vehicle towards the next waypoint chosen by the look ahead distance [17]. The algorithm is generalized for use in UE4 as shown in Algorithm 2.

At each time step, [Algorithm 2] is run to set the navigated vehicle's steering angle, δ, to point towards the next destination waypoint area, P, which is a padded (1m x 1m) area centered around the next destination point, d_ϕ, chosen using the look-ahead value, ϕ. The total time of one time-step, s, is found using the frequency of the simulation, f_{sim} [eq. (3.4)], and is used to move the current time-step forward when searching for P.

$$s = \frac{1}{f_{sim}} \tag{3.4}$$

The steering angle is then found. As t is the current time-step of the simulation, d is the set (x,y) trace path destinations, and K is the total number of time-steps used for the calculation. The steering angles for the next K time-steps are found for the current vehicle's location m and averaged together to acquire the next steering angle [18] [eq. (3.5]. A visualization of this can be seen in [Fig. 3.8].

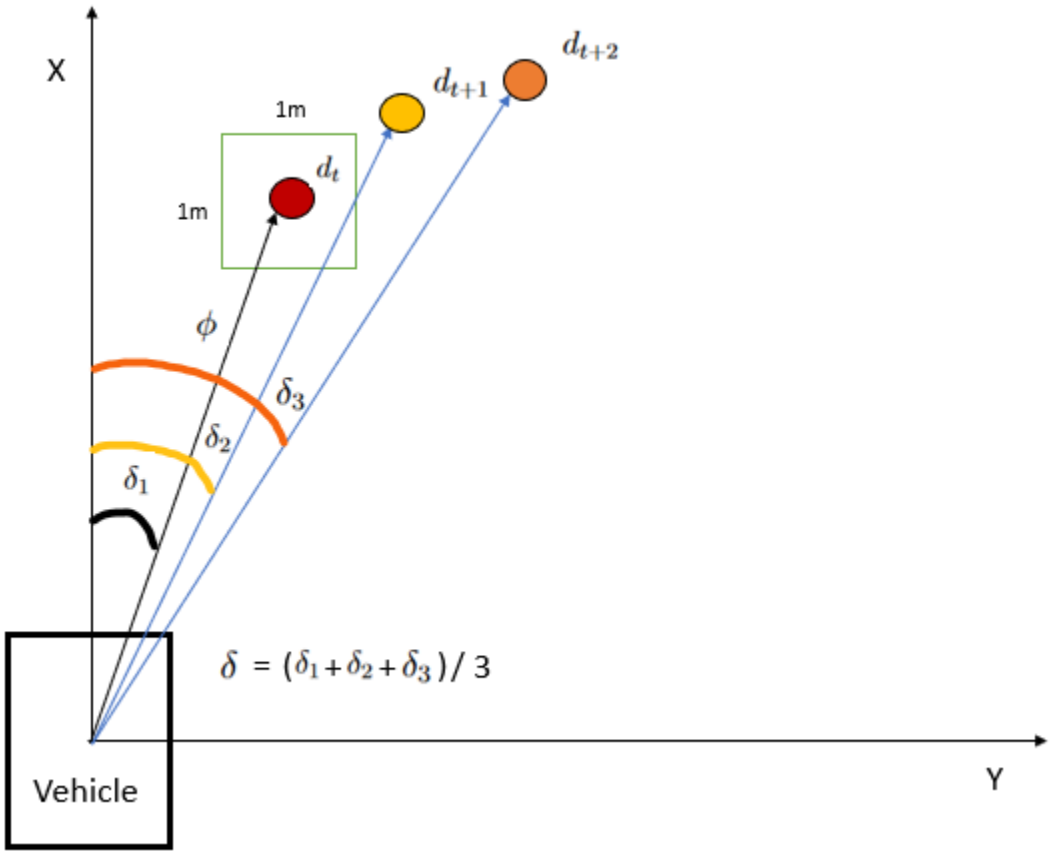

Figure 3.8: Steering Angle Calculation

$$\delta_{turn} = \frac{1}{K} * \sum_{i=t}^{K} \delta_i \tag{3.5}$$

The implementation described in Algorithm 2 pads each upcoming waypoint destination d_ϕ to form the target area P, giving the vehicle a threshold of forgiveness to prevent it from missing the pixel representing the destination waypoint coordinate

placed onto the imported terrain. The area chosen for the padding was set and arbitrarily determined before the experiment to be $1m^2$, as to not exceed the width of the vehicle from its CG point. Thus, a target padding threshold of (1m x 1m) was configured for each waypoint during the simulation, allowing a maximum square CTE area of 1m around any given waypoint destination.

Algorithm 2 Pure Pursuit

while waypoints exist **do**
  $m \leftarrow$ Current vehicle (x,y) position
  $\phi \leftarrow$ Get look-ahead distance using [Algorithm 3]
  $d_\phi \leftarrow$ Get Destination waypoint based on ϕ
  $P \leftarrow$ Pad waypoint creating a (1m x 1m) destination area centered around waypoint d_ϕ.
  while m is not within the destination area P **do**
    $m \leftarrow$ Refresh current vehicle location
    $\delta \leftarrow$ Calculate steering angle using [eq. (3.5)].
    Set vehicle steering $\leftarrow \delta$
    $t = t + s$ // Increase time-step of the simulation
    if m misses P **then**
      $\rightarrow$ Simulation Failed
    end if
  end while
end while

Missing the destination area will result in the vehicle attempting to return to the trace path, causing the controller to sharply turn the wheel in an attempt to steer and reorient the vehicle back towards the missed waypoint. The resulting action causes the vehicle to endlessly spin around in circles when traveling at high velocities. This action is considered to be a point of failure for the simulation, and therefore the

simulation is ended and noted as "Did-not-Finish" (DNF).

The look-ahead distance is not fixed, and is set to shorten in the event of an approaching turn or lengthen along a straight path as described in Algorithm 3. A starting minimum look-ahead value, ϕ_{min}, is set by taking the floor value of a chosen number of time-steps, k_ϕ, over the original sensor frequency at which the trace data was sampled, f_{trace} [eq. (3.6)]. The steering angle needed is calculated for each future destination time-step at the current look ahead value, d_ϕ.

$$\phi_{min} = \lfloor \frac{k_\phi}{f_{trace}} \rfloor \tag{3.6}$$

If the resulting steering angle, δ, is larger than a preset turning angle, δ_{turn}, a turn is detected and the look ahead value is found. If the look ahead value reaches the max look ahead, ϕ_{max}, the current set of destinations is found to be nearly a straight line. The look-ahead value is returned to Algorithm 2.

This flexible look-ahead value helps prevents the vehicle from oscillating, gradually re-adjusting the steering angle for every time step. This is common for many path following AV algorithms and found implemented in many out-of-the-box solutions, such as MATLAB's Pure Pursuit controller [41].

Algorithm 3 Look-Ahead Calculation

1: $\phi \leftarrow$ Set to minimum look-ahead distance of ϕ_{min} found using [eq. (3.6)]
2: $d_\phi \leftarrow$ Get Destination waypoint based on ϕ
3: $\delta \leftarrow$ UE4::SteeringAngle(d_ϕ) // Get current steering angle of vehicle towards destination using UE4 engine function.
// Loop until max look-ahead is reached, or a turn is detected.
4: **while** $\phi < \phi_{max}$ or $\delta < \delta_{turn}$ **do**
5: $\phi \leftarrow \phi + 1$ // Increase look ahead time-step
6: $d_\phi \leftarrow$ Get new Destination waypoint based on new ϕ
7: $\delta \leftarrow$ UE4::SteeringAngle(d_ϕ) Get new steering angle.
8: **end while**
9: Return ϕ // Return look ahead after loop

3.6 Simulation Tests

This section will outline the three sets of tests and their respective paths fixed at a simulation frequency, f_{sim}, which is be referred to as the simulation's frames-per-second (FPS) by UE4. The set frequencies for each test start at 60Hz and increase f_{sim} by 10Hz for each test until 400Hz is reached. Table 3.4 lists the test number, the maneuver being performed, and the maximum speed of the vehicle during execution.

Table 3.4
Simulation Tests

Test	***Maneuver***	***Max Speed(mph)***
1	Straight	5
2	Left	10
3	Left	15
4	SS	20
5	SS	25
6	SS	30
7	DLC	30
8	DLC	35
9	DLC	40

Figure 3.9 displays a straight path taken by Test 1 (*St5*) within the first set of calibration tests 1, 2, and 3. These tests are used to calibrate the PID control values for proper behavior[3], as well as make sure the Pure Pursuit controller's flexible look-ahead distance works as expected. The HMMWV travels in a straight line for *St5* at a low speed of 5mph. Figure 3.10 displays Tests 2 (*Left10*) and 3 (*Left15*), where a sharp left hand turn is done at the end of *St5* moving at slightly higher speeds of 10 and 15mph.

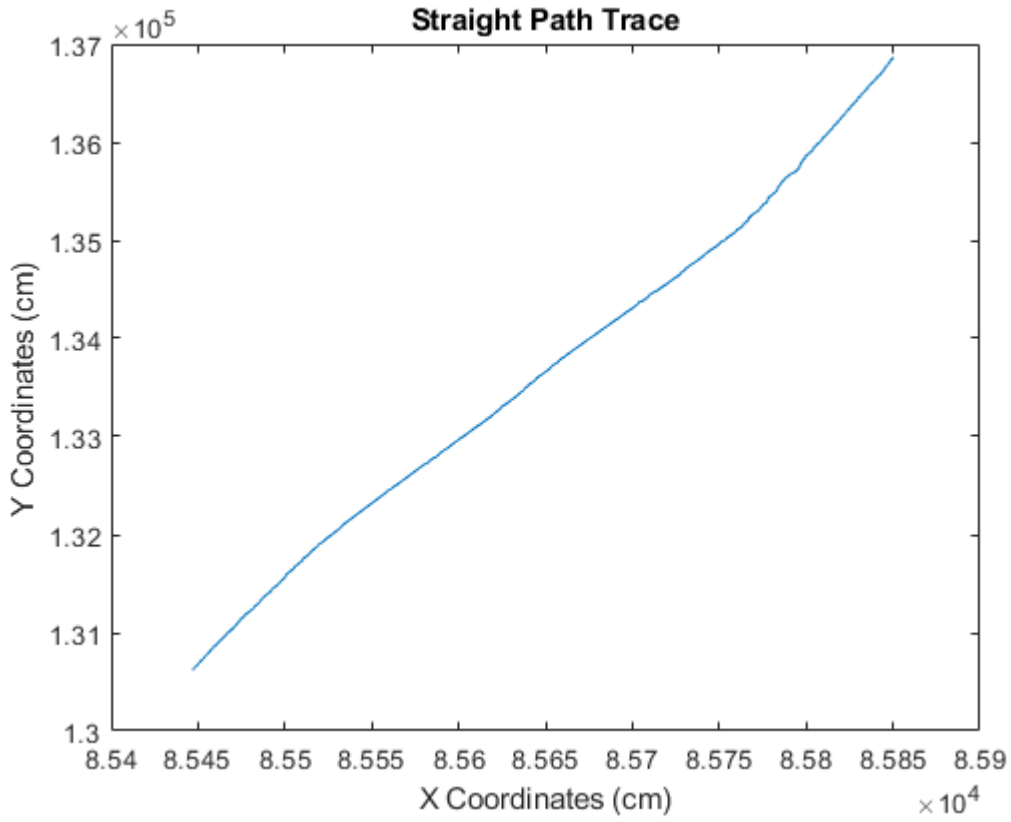

Figure 3.9: Straight Path Trace

[3]Velocity trace plots used when tuning the calibration tests are shown in Appendix A.1

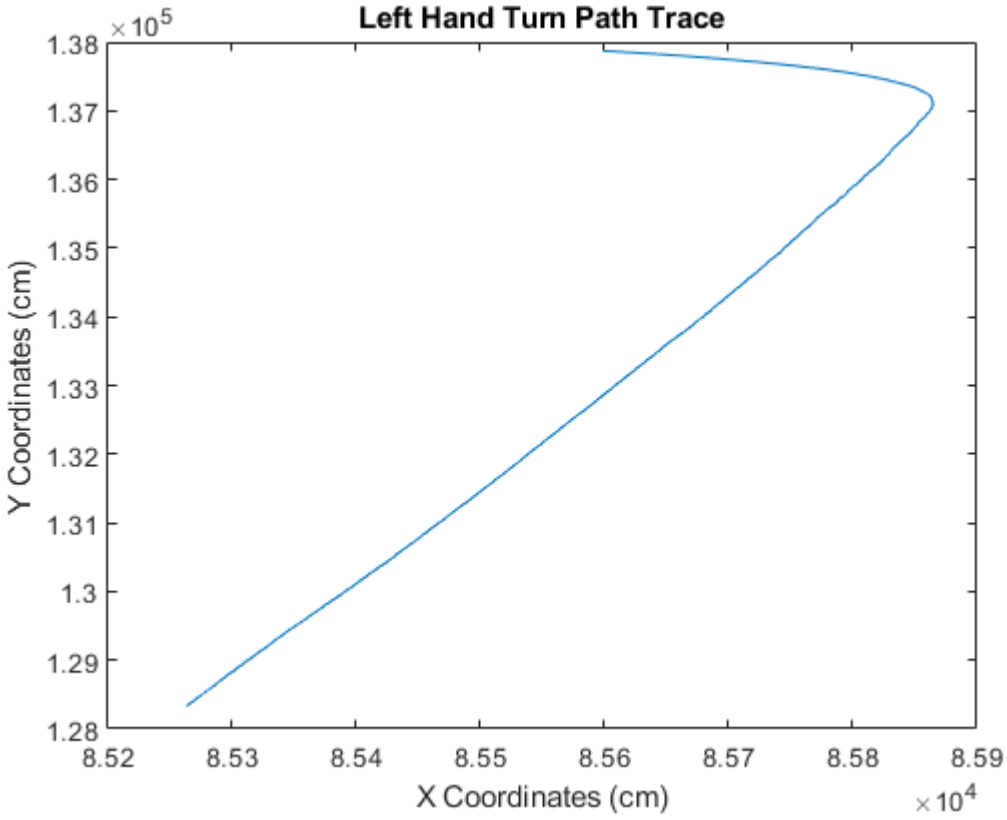

Figure 3.10: Left Hand Turn Path Trace

Figure 3.11 displays the path taken by the secondary set of tests 4 (*SS20*), 5 (*SS25*), and 6 (*SS30*). The HMMWV takes a wide sweeping left-hand turn called a Step Steer (SS) going at higher speeds of 20, 25, and 30mph. These tests demonstrate the simulation being able to execute a controlled sweeping turn of the vehicle 180 degrees while moving at higher velocities in an off road environment.

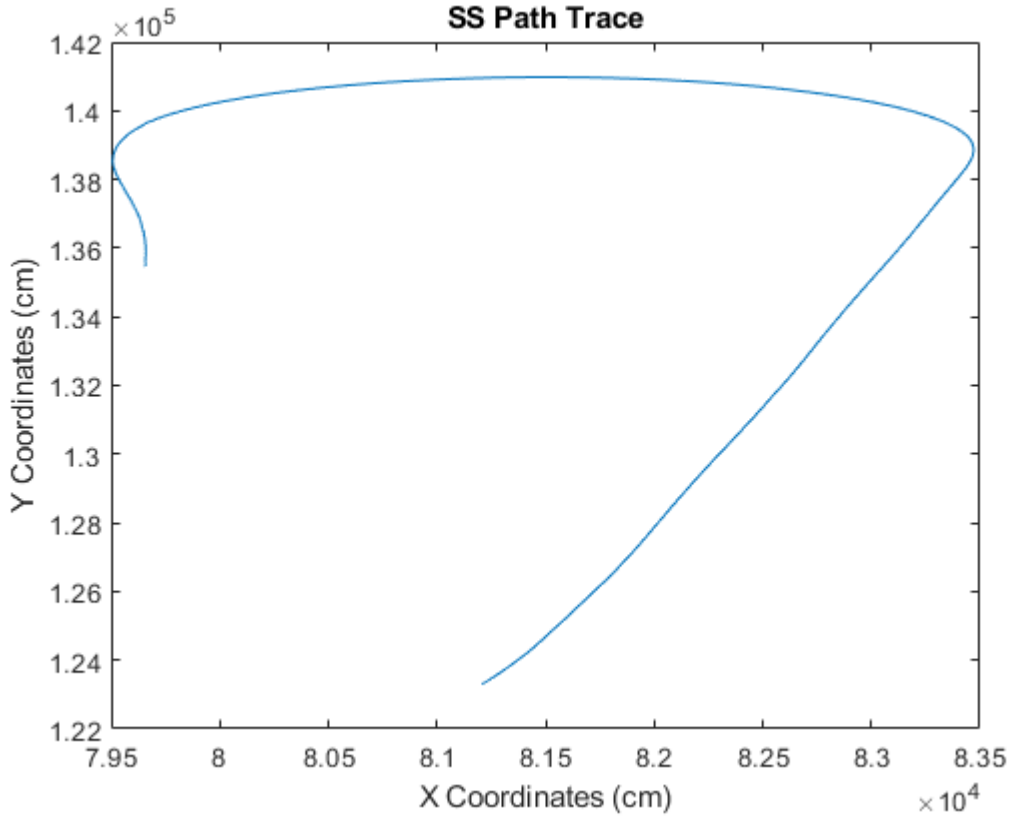

Figure 3.11: SS Path Trace

Figure 3.12 displays the last maneuver taken by the final set of tests 7 (*DLC30*), 8 (*DLC35*), and 9 (*DLC40*). The HMMWV performs a Double Lane Change (DLC) moving the vehicle at higher speeds of 30, 35, and 40mph. These three tests demonstrate the ability for the simulation to execute a more difficult steering path taken at higher velocities within off road environments.

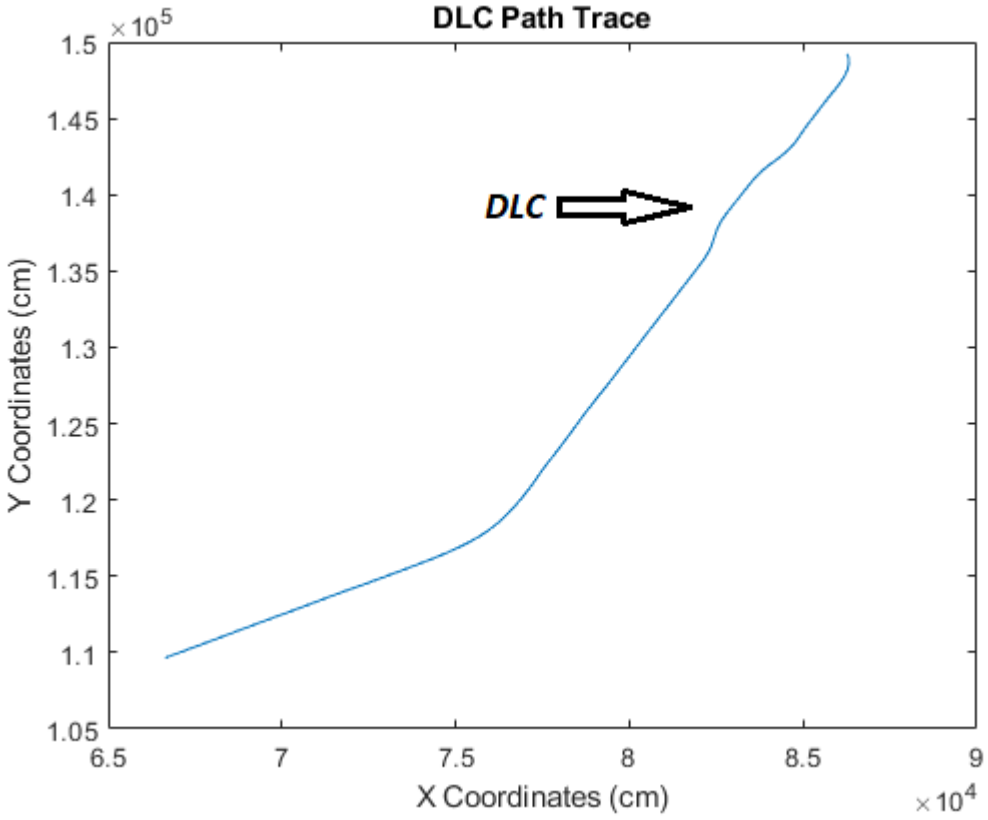

Figure 3.12: DLC Path Trace

Chapter 4

Results

This chapter details the CTE and heading error results after simulating the three sets of maneuvers described previously in Table 3.4. Velocity error was not taken into consideration, as additional vehicle dynamic tuning aside from the default brake torque and engine RPM configuration supplied by UE4 was out of scope for this thesis. Each test maneuver and resulting data set has a corresponding table and MATLAB generated line plots. Simulations that failed at specified frequencies are marked as DNF.

4.1 Overall Results

Table 4.1 and Table 4.2 display the quantiles for the overall average CTE ($\overline{CTE}$) and Heading error ($\overline{Head.Err.}$) for all tested simulation frequencies during each maneuver seen in Fig. 4.1 and Fig. 4.2.

Table 4.1
Average CTE Quantiles

Quantile	$\overline{CTE}(cm)$
Q1	6.3389
Q2	6.4189
Q3	6.8727

Table 4.2
Average Heading Error Quantiles

Quantile	$\overline{Head.Err.}(deg.)$
Q1	2.7372
Q2	3.6950
Q3	4.8862

A two-term Gaussian curve was fit to the overall $\overline{CTE}$ data over each tested frequency for analysis. $\overline{CTE}$ is seen to grow linearly until an exponential growth point happens in the Gaussian's slope [Fig. 4.1]. The frequency before the rise in slope and before the first $\overline{CTE}$ value above Q3 was chosen as the threshold frequency, 190Hz. This threshold ensures all maneuvers are able to complete down to the lower frequency of 190Hz.

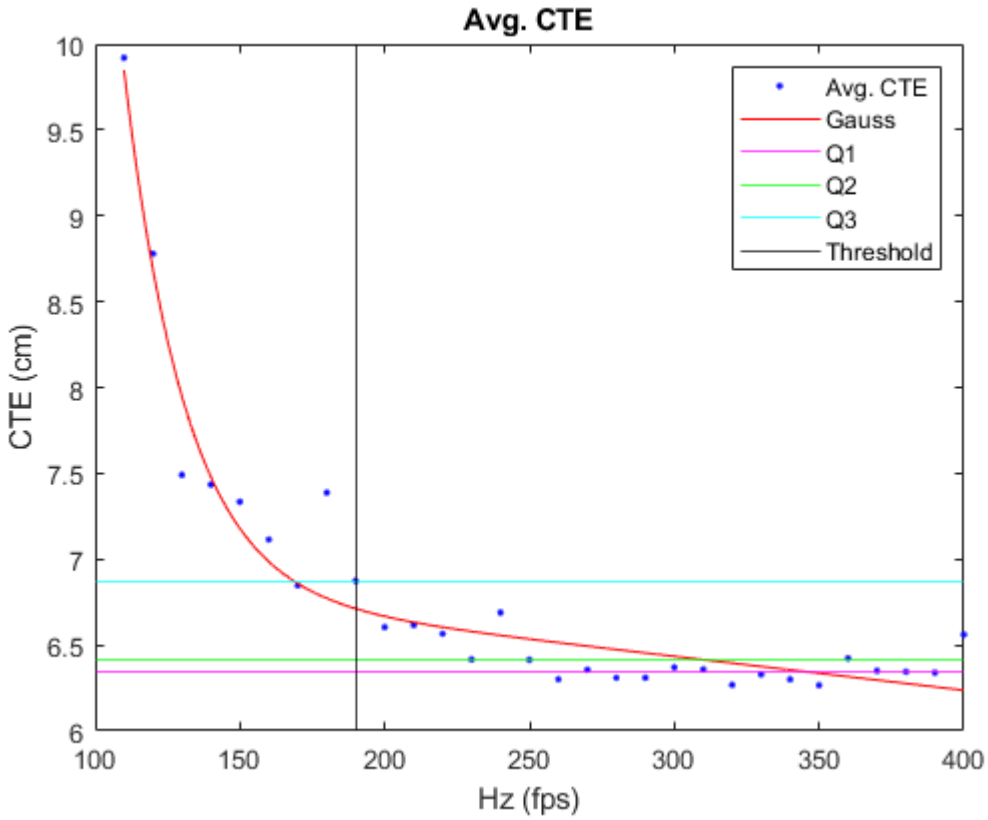

Figure 4.1: Overall Average CTE

A two-term Gaussian curve was again fit to the overall $\overline{Head.Err.}$ data over each tested frequency for analysis. $\overline{Head.Err.}$ is seen to grow linearly until an exponential growth point happens in the Gaussian's slope [Fig. 4.2]. The frequency before the sudden rise in slope and before the first $\overline{CTE}$ value above Q3 was chosen as the threshold frequency, which was again 190Hz. This threshold ensures all maneuvers are able to complete down to the lower frequency of 190Hz.

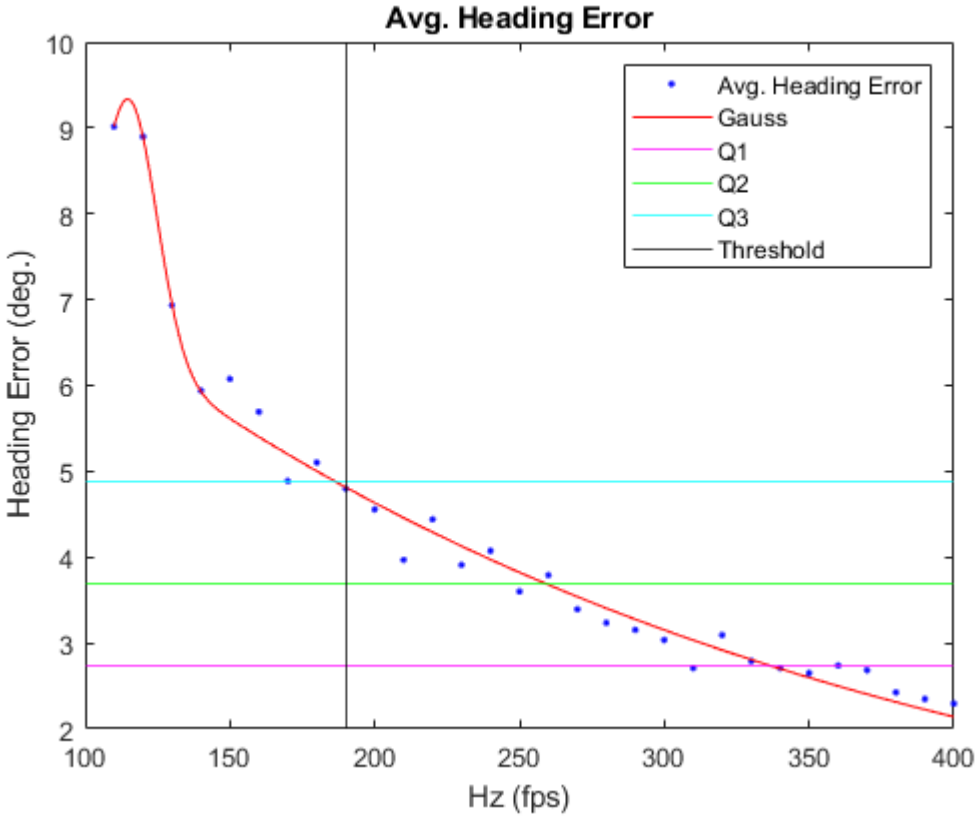

Figure 4.2: Overall Average Heading Error

4.2 Calibration Tests

Table 4.3 displays the collected data of the simulations at specified frequencies for the calibration tests. Fig. 4.3 and Fig. 4.4 plot the $\overline{CTE}$ and $\overline{Head.Err.}$ of each test on the same graphs for analysis.

Table 4.3
Calibration Simulation Results

	St5		***Left10***		***Left15***	
Hz	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$
60	1.891522	0.922805	3.082390	15.697267	6.819039	21.225049
70	1.842817	0.757049	2.805024	13.525814	6.196473	18.390512
80	1.833223	0.672022	2.708854	11.881769	5.979504	16.327520
90	1.967582	0.621831	2.673124	10.599547	5.763679	14.669610
100	1.931992	0.562873	2.624452	9.559972	5.62052	13.288782
110	1.787793	0.459340	2.584373	8.712740	5.499453	12.167102
120	1.889159	0.441234	2.723644	8.007371	5.423508	11.216049
130	1.685567	0.400268	2.682468	7.399635	5.331185	10.405371
140	1.816073	0.364401	3.023863	6.876754	5.226642	9.700805
150	1.78716	0.326872	3.079309	6.434316	5.183404	9.088313
160	1.655485	0.316272	3.013240	6.030094	5.125658	8.547498
170	1.646901	0.299599	3.131273	5.676292	5.060905	8.053036
180	1.713229	0.277426	3.143653	5.365062	5.049087	7.638043
190	1.831148	0.262586	3.119689	5.086318	4.988932	7.253359
200	1.696292	0.248950	2.379170	4.822986	4.962521	6.891421
210	1.711245	0.234282	2.432057	4.599132	4.951152	6.588155
220	1.676641	0.223797	2.328067	4.389601	4.897463	6.290790
230	1.663754	0.210848	2.315582	4.199207	4.890135	6.031354
240	1.880598	0.205233	2.316239	4.026346	4.864323	5.787045
250	1.888980	0.204431	2.391243	3.865517	4.835505	5.553622
260	1.772259	0.193888	2.384717	3.718839	4.845091	5.357040
270	1.801274	0.186497	2.405052	3.583335	4.827863	5.163151
280	1.797556	0.191196	2.321159	3.347964	4.779015	4.977857
290	1.741477	0.185695	2.281089	3.236352	4.800795	4.815137
300	1.737188	0.174694	2.344374	3.034957	4.772164	4.657860
310	1.740281	0.171027	2.323503	3.132990	4.762870	4.534330
320	1.680158	0.162433	2.344374	3.034957	4.748046	4.369146
330	1.895515	0.162490	2.414352	2.947065	4.752296	4.248080
340	1.962175	0.148854	2.300683	2.865992	4.731562	4.122030
350	1.774689	0.150172	2.294771	2.784460	4.727151	4.008068
360	1.842076	0.146906	2.404804	2.712324	4.714672	3.898805
370	1.729449	0.142609	2.385316	2.636866	4.707031	3.813778
380	1.707039	0.143239	2.370392	2.567366	4.685610	3.693285
390	1.770111	0.142322	2.319529	2.505773	4.672859	3.601498
400	1.920164	0.137796	2.312064	2.441258	4.679238	3.515382

Looking at Fig. 4.3, a low $\overline{CTE}$ beneath Q1 for test speeds of 5, 10, and 15mph is demonstrated. A slight exponential growth can be seen developing along with a higher overall $\overline{CTE}$ within Test *Left15* moving at the highest speed of 15mph compared to the low $\overline{CTE}$ of 2-3cm from tests *St5* and *Left10*.

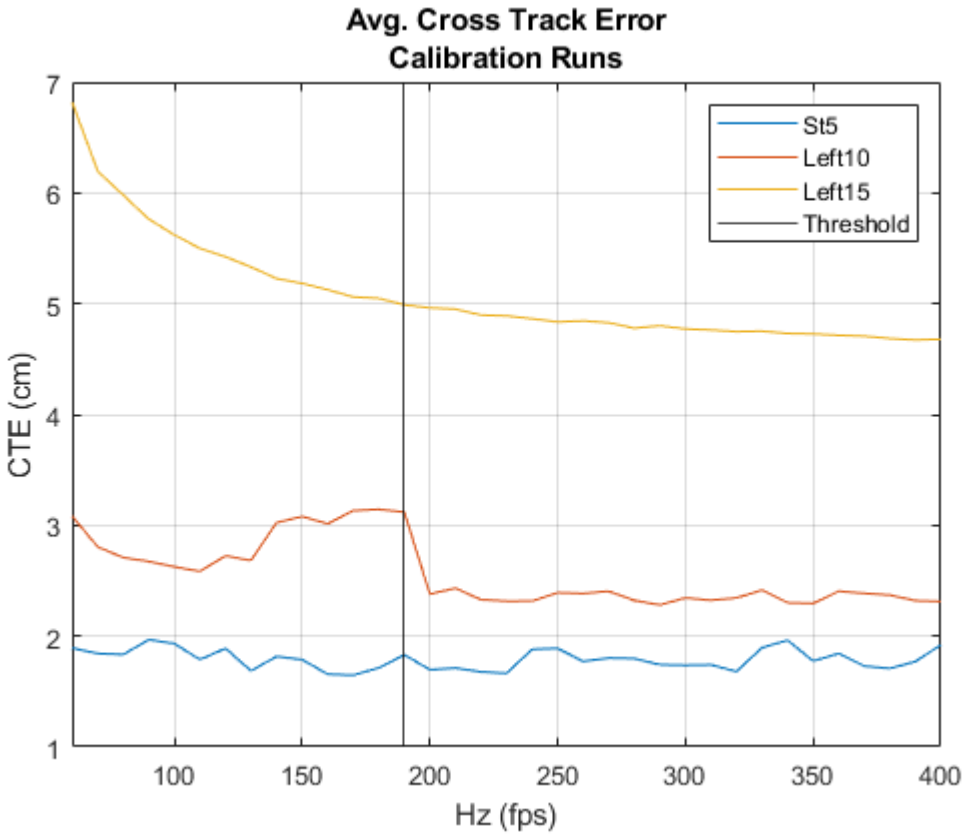

Figure 4.3: Calibration Paths CTE

A larger difference is viewed between the straight run $\overline{Head.Err.}$ of Test *St5* and the two tests with a sharp left hand turn seen in Fig. 4.4. A more pronounced exponential increase is viewed in test *Left10* and *Left15* running at 10 and 15mph as the frequency is pushed lower.

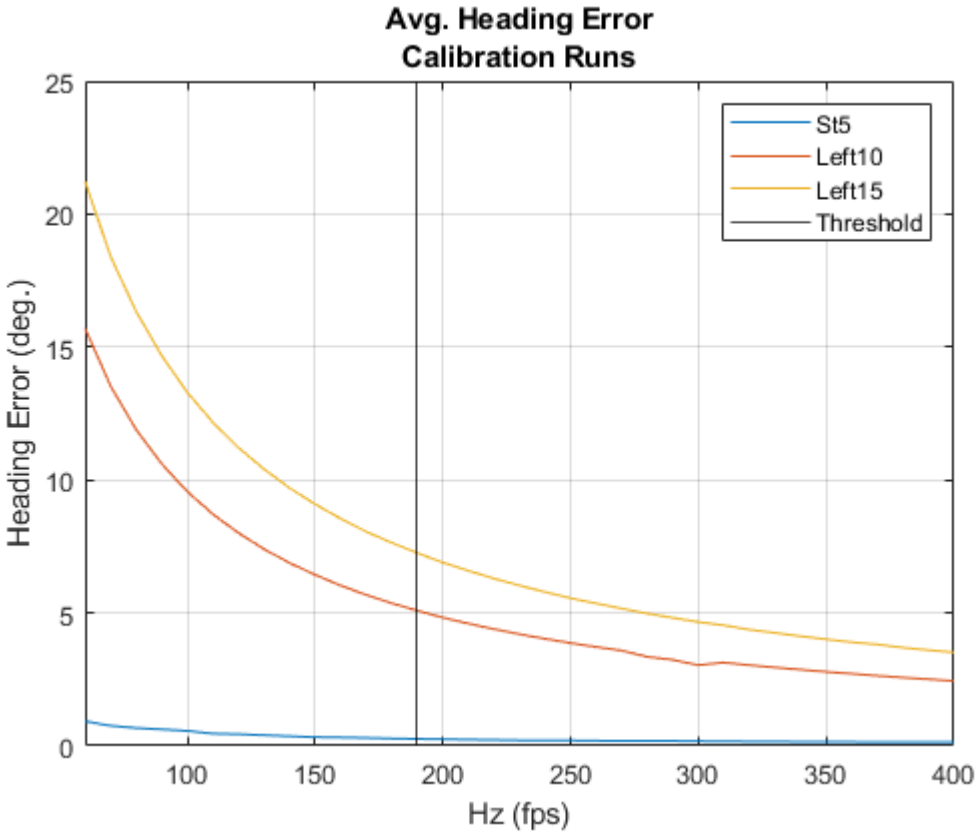

Figure 4.4: Calibration Paths Heading Error

4.3 SS Tests

Table 4.4 displays the collected data of the simulations at specified frequencies for the SS tests. Fig. 4.5 and Fig. 4.6 plots the $\overline{CTE}$ and $\overline{Head.Err.}$ of each test on the same graphs for analysis.

Table 4.4
SS Test Simulation Results

	SS20		***SS25***		***SS30***	
Hz	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$
60	DNF	DNF	DNF	DNF	DNF	DNF
70	8.34685	7.843849	DNF	DNF	DNF	DNF
80	7.378466	13.74342	9.773618	7.962279	16.653856	18.60262
90	7.295697	6.098677	9.417886	7.223737	13.762456	16.23756
100	7.376283	11.35264	9.301751	6.643273	12.876025	14.48718
110	7.389835	5.219473	9.196547	12.30031	12.694913	13.28202
120	7.424166	4.856619	9.180739	11.35585	12.620791	12.28828
130	7.371074	4.548482	9.215352	5.322033	12.533267	11.39464
140	7.454406	4.264181	9.232253	4.997223	12.619330	10.65231
150	7.471124	4.022392	9.247021	4.709713	12.585999	9.996108
160	7.461627	3.800314	9.260652	4.453314	12.602511	9.414670
170	7.508097	3.613873	9.275335	4.223100	12.525677	8.893851
180	7.512874	3.434652	9.243983	4.009673	12.522070	8.429641
190	7.496139	3.260702	9.242389	3.821170	12.529353	8.006111
200	7.505121	3.114713	9.261789	3.649053	12.585373	7.633975
210	7.559954	2.974796	9.271124	3.491662	12.566146	7.290773
220	7.582916	5.461032	9.284304	3.347276	12.581700	6.969916
230	7.581328	5.236662	9.290617	3.208219	12.612000	6.679885
240	7.602193	5.028506	9.309110	3.091221	12.613731	6.415007
250	7.60614	4.835649	9.300417	2.971416	12.579530	6.169781
260	7.611008	4.655969	9.316853	2.866622	12.583597	5.941400
270	7.617917	2.366029	9.331010	2.768016	12.583583	5.727286
280	7.571183	2.287132	9.319286	2.675541	12.628793	5.535001
290	7.644691	2.213507	9.369827	2.586961	12.632623	5.351826
300	7.651090	2.144752	9.371148	2.506231	12.604430	5.177705
310	7.627004	2.073992	9.384303	2.431059	12.678157	5.008052
320	7.652097	3.805184	9.402393	2.358695	12.663909	4.863437
330	7.668487	1.960833	9.393157	2.292518	12.695947	4.720198
340	7.634447	1.903365	9.388501	2.227659	12.681464	4.587787
350	7.625379	1.850309	9.396739	2.167384	12.676575	4.459846
360	7.644919	3.393800	9.401565	2.111635	12.669837	4.339754
370	7.664490	3.307112	9.409889	2.056173	12.734031	4.226251
380	7.644102	1.710966	9.399457	2.004550	12.675023	4.117733
390	7.666570	3.140668	9.418030	1.957395	12.747469	4.011392
400	7.687033	3.064693	9.395208	1.910126	12.665688	3.917197

A higher $\overline{CTE}$ is observed again in Fig. 4.5 when the vehicle moves at higher velocities, this time when performing a more gradual SS turn. Test *SS30* moving at 30mph shows a sharp increase in $\overline{CTE}$ when simulating at less than 100Hz. The sharp increase follows the simulation failing at 70Hz for tests *SS25* and *SS30*, and at 60Hz for Test *SS20*. These DNFs are caused by the vehicle missing the 1m padding waypoint area P.

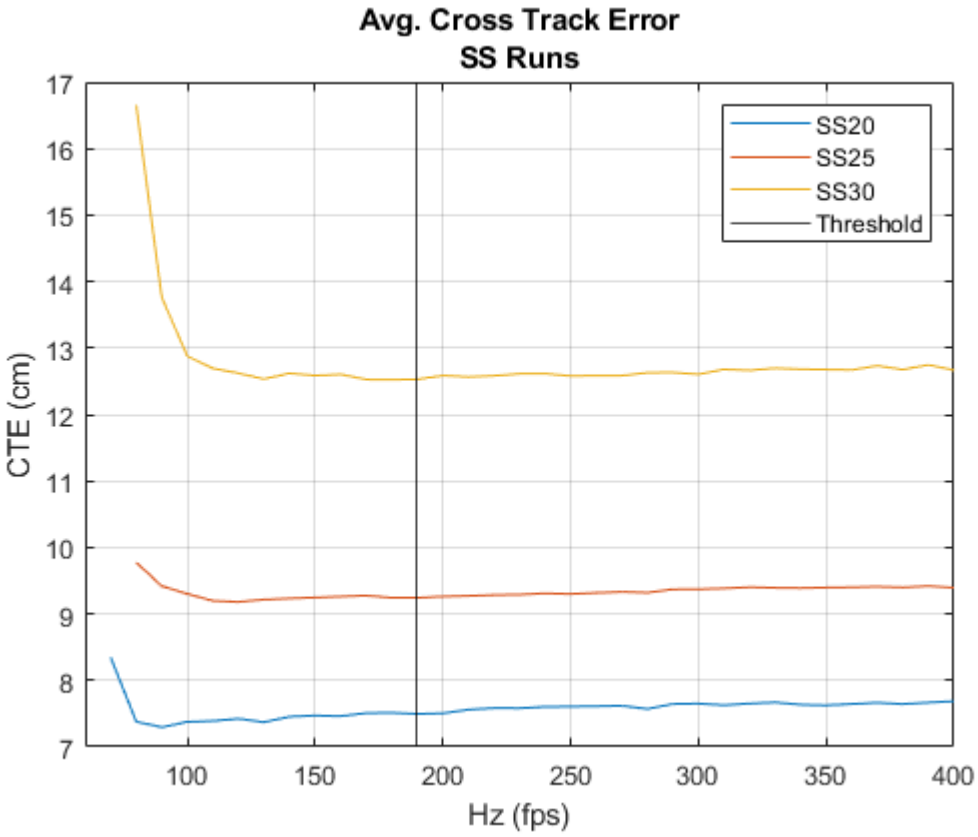

Figure 4.5: SS CTE

The simulation AV model shows greater deviation from the trace path at lower frequencies, which can again be seen by the greater $\overline{Head.Err.}$ data shown in Fig. 4.6 for tests *SS20*, *SS25*, and *SS30*. As the frequency of the simulation is lowered, the vehicle starts to deviate the original trace path. This is similar to the increase in $\overline{Head.Err.}$ seen at higher speeds with low simulation frequency in Tests *Left10* and *Left15*.

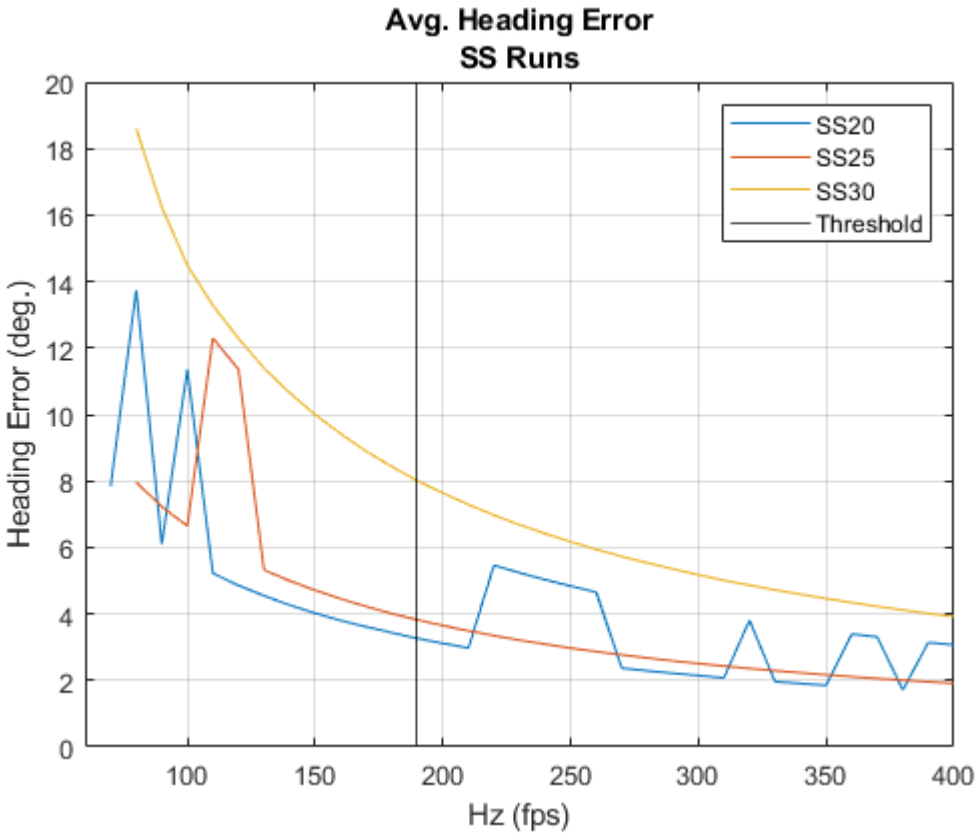

Figure 4.6: SS Heading Error

This deviation from the trace path is further evidenced by comparing the $\overline{CTE}$ during Test *SS30* running at 400Hz and 80Hz seen in Fig. 4.7 and Fig. 4.8. The 80Hz $\overline{CTE}$ shows the vehicle straying farther from the input path during the SS turn between 15 and 30 seconds compared to the more controlled 400Hz $\overline{CTE}$ during execution[1].

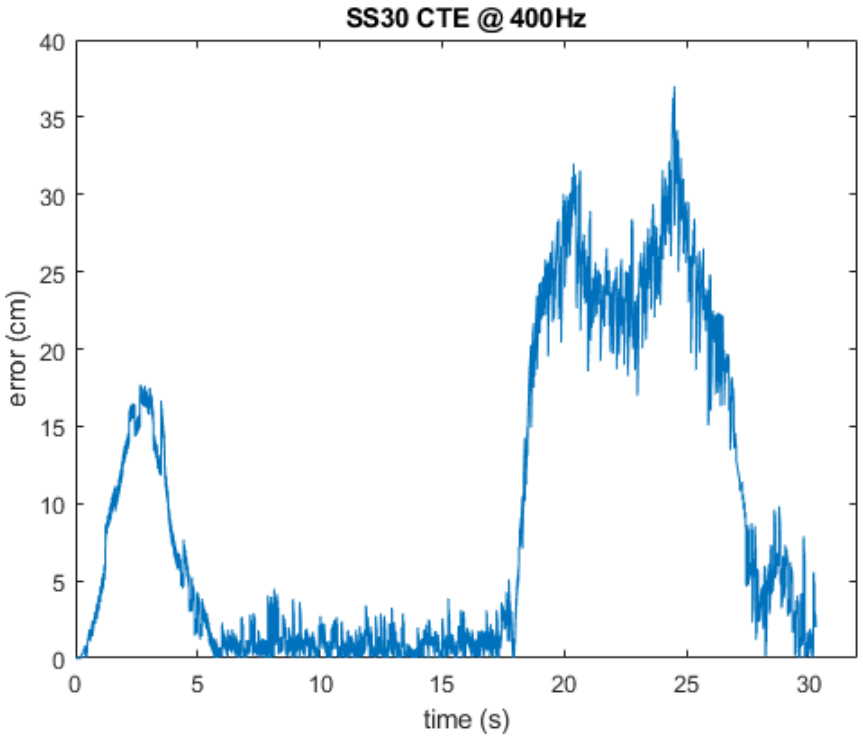

Figure 4.7: SS30 CTE Error 400Hz

[1]CTE oscillation comparing higher and lower simulation frequencies are further shown for each additional maneuver in Appendix A.2.

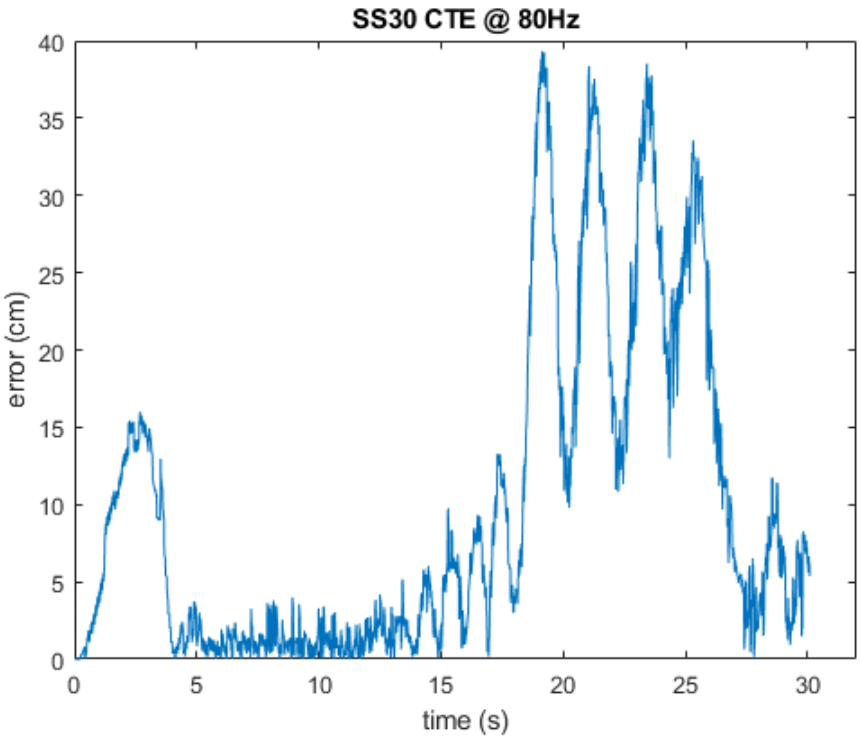

Figure 4.8: SS30 CTE Error 80Hz

4.4 DLC Tests

Table 4.5 displays the collected data of the simulations at specified frequencies for the DLC tests. Fig. 4.9 and Fig. 4.10 plots the $\overline{CTE}$ and $\overline{Head.Err.}$ of each test on the same graphs for analysis.

Table 4.5
DLC Test Simulation Results

	DLC30		***DLC35***		***DLC40***	
Hz	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$	$\overline{CTE}(cm)$	$\overline{Head.Err.}(deg)$
60	DNF	DNF	DNF	DNF	DNF	DNF
70	DNF	DNF	DNF	DNF	DNF	DNF
80	DNF	DNF	DNF	DNF	DNF	DNF
90	DNF	DNF	DNF	DNF	DNF	DNF
100	27.341143	8.431303	12.463499	6.006316	DNF	DNF
110	18.052394	10.59267	11.885003	5.451120	20.190624	12.94265
120	11.667213	9.619273	13.020851	5.309141	15.062654	16.99220
130	6.243541	3.328311	9.422495	4.120425	12.93372	15.46785
140	4.946065	3.070423	9.422495	4.120425	13.165816	9.410889
150	4.271237	2.712554	9.254976	3.880700	13.129571	13.50725
160	4.100219	2.501189	8.582768	3.600180	12.226453	12.56399
170	3.751231	2.366888	7.627353	3.316967	11.090973	7.532504
180	7.328052	2.495918	7.655718	3.178082	12.316035	11.09435
190	3.836295	2.088717	7.239554	2.972963	11.570381	10.43986
200	3.57391	1.992919	7.011474	2.810644	10.456394	9.839060
210	4.112967	1.897349	7.005286	2.647695	9.931593	5.989013
220	3.516976	1.794904	6.785181	2.531900	10.438969	8.948512
230	3.51787	1.726952	6.631455	2.412152	9.239214	5.476960
240	6.866305	1.776742	6.377403	2.203824	8.370936	8.126489
250	3.988504	1.594770	6.377403	2.203824	8.757278	5.003067
260	3.569377	1.529568	6.356443	2.117250	8.262035	7.727653
270	3.865507	1.490205	6.361691	2.035432	8.409946	7.207064
280	3.849455	1.425977	6.285842	1.962036	8.225969	6.706585
290	3.923436	1.387073	6.161731	1.893797	8.225969	6.706585
300	4.516296	1.336997	6.140598	1.827735	8.189398	6.466630
310	4.561139	1.294941	6.135383	1.716638	8.004916	4.004172
320	4.111466	1.259303	6.135383	1.716638	7.672652	6.260767
330	4.13219	1.216847	6.144427	1.667364	7.863027	5.889834
340	4.075147	1.180006	6.096241	1.615683	7.831302	5.716629
350	4.193705	1.144196	6.066429	1.565091	7.636952	5.715196
360	4.850157	1.118069	6.101322	1.527448	8.169249	5.386204
370	4.912846	1.087531	5.975787	1.481038	7.630288	5.399783
380	5.00752	1.059971	5.919995	1.444025	7.689152	5.087406
390	5.014129	1.034016	5.915088	1.403861	7.526209	3.320977
400	7.003259	1.054528	5.874571	1.368681	7.506525	3.237211

Test *DLC40*, as with the previous tests performing their selected maneuvers at the highest speed, shows a greater $\overline{CTE}$ for each frequency displayed in Fig. 4.9. Due to the more complex DLC maneuver and the higher speeds of each test, it can be seen in Table 4.5 that the simulation fails at all frequencies lower than 100 Hz, whereas the Calibration and SS Tests successfully complete simulations at frequencies lower than 100 Hz. Test *DLC40* is shown to fail even sooner at 110 Hz. Even though Test *DLC30* appears to show a higher $\overline{CTE}$ at 100Hz, this is due to Test *DLC40* failing at that frequency.

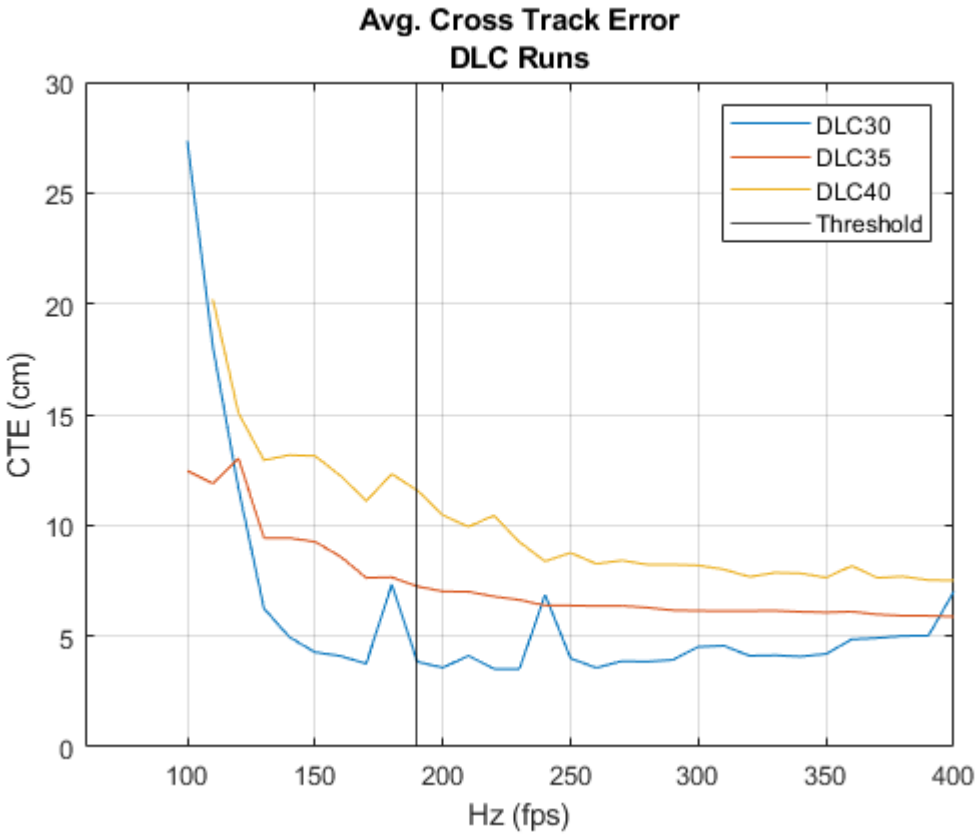

Figure 4.9: DLC CTE

The simulation AV model deviates greater when traveling at higher speeds and performing a more complex driving maneuver, which is further evidenced observing the Test *DLC30* $\overline{Head.Err.}$ in Fig. 4.10. Tests *DLC30* and *DLC35* show a steady increase in $\overline{Head.Err.}$ while descending to lower frequencies, while Test *DLC40* is greater and less predictable deviating from the trace path at the highest speed.

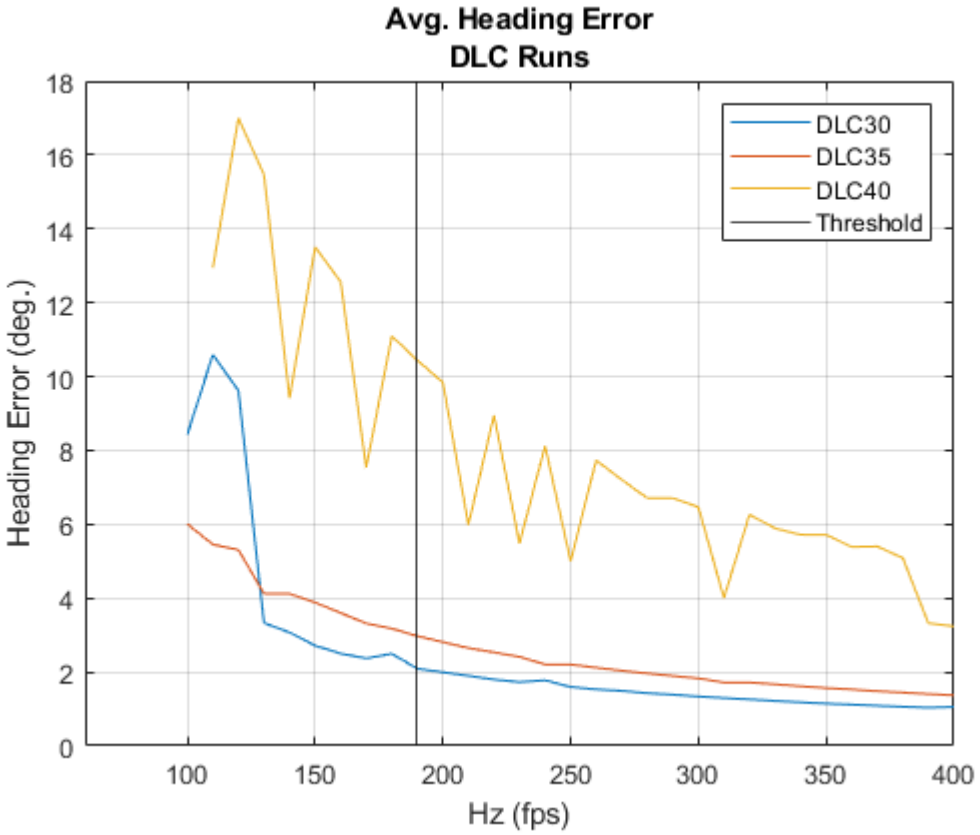

Figure 4.10: DLC Heading Error

Chapter 5

Discussion

This chapter sets to draw conclusions from observations of the $\overline{CTE}$ and $\overline{Head.Err.}$ data from each set of off road simulation tests. A determination is made about whether or not a baseline low frequency can be successfully demonstrated when repeating and modeling real world off road AV scenarios. Additional simulation improvements outlined in this experiment are discussed, along with future research moving forward.

5.1 Conclusions

This thesis has successfully demonstrated that a low simulation frequency (<1000Hz) can be used for repeatable $\overline{CTE}$ and $\overline{Head.Err.}$ performance when analyzing the recorded path telemetry of a real vehicle platform and its virtual counterpart. Integrated C++ Pure Pursuit and PID control algorithms running at the set simulation frequency were used to drive a HMMWV military vehicle modeled within UE4 (a single simulation platform) through multiple different off road driving maneuvers performed by a real driver at varying low to original input velocities within a virtual representation of the KRC test course.

A trade off between the set simulation frequency and the virtual AV's deviation from the real input trace path telemetry can be seen when observing each set of tests. The $\overline{CTE}$ and $\overline{Head.Err.}$ for each test remained lowest when running at or near the original input trace path sampled frequency, and slowly increased linearly until hitting a point in which the simulation started deviating from the supplied trace path exponentially. This was seen in the $\overline{CTE}$ for *SS30* running at 400Hz [Fig. 4.7] compared to running at 80Hz [Fig. 4.8]. The relationship between $\overline{CTE}$ and maximum AV speed was positive, as maximum speed increased $\overline{CTE}$ increased. This was also true for the relationship between $\overline{Head.Err.}$ and AV speed.

Based on the overall average results for both $\overline{CTE}$ and $\overline{Head.Err.}$, a lower boundary simulation frequency of 190Hz is recommended for path following when modeling and repeating the stated straight, left, SS, and DLC real world off road driving maneuvers up to the tested maximum speeds on the modeled terrain. Frequencies lower than 190Hz introduce noticeable path deviation, increasing the risk of simulation failure especially when the driving maneuver was performed at higher velocities. The greater deviation is also shown in the exponential increase found in the Gaussian fit line's slope when applied to the overall average results for both the $\overline{CTE}$ and $\overline{Head.Err.}$ below 190Hz viewed in Fig. 4.1 and Fig. 4.2.

5.2 Future Research

More complex autonomous control methods besides Pure Pursuit could be implemented and improved upon to control the vehicle model using the created UE4 simulation environment. An example of a more complex controller is the Stanley control method, which includes lateral acceleration, yaw rate, and current CTE in its steering angle calculations. A similar experiment would be performed for each additional control method at the same lower frequencies presented in this thesis to see if greater stability can be achieved below 190Hz.

Once a path planning control method is chosen, an open-loop control system within

the virtual world theoretically could be constructed between the simulation and a real world AV platform to test the viability of future AV control using the simulated environment. Data compared between the simulated path and actual path executed on the real AV could be analyzed to see if a low $\overline{CTE}$ holds at a simulation frequency of 190Hz. Any instability caused by introducing systems outside of an ideal virtually simulated environment could also be observed (e.g., communication over a wired or wireless network medium such as Ethernet or 802.11p).

A basic default set of virtual off road vehicle dynamics and characteristics (e.g. suspension system, brake torques, lateral slip coefficients, etc.) were chosen to achieve path following. Characteristics of the vehicle, such as the slip and yaw rate, still need to be researched and evaluated within the simulation using UE4 and PhysX. Outside of the default physical interactions between the PhysX vehicle and UE4 landscape environment using the integrated Pure Pursuit and PID controllers, definite approximations of what is possible between real and virtual world vehicle dynamics is still unknown.

Even though the research in this thesis has demonstrated that low simulation frequency AV modeling and repeatability using real world telemetry is possible, more research is needed before any definitive statements can be made regarding the full scope of testing real time control or path planning AV systems using a single visual and physics simulation platform like UE4.

References

[1] J. Brummelen, D. Gruyer, H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow", Transportation Research Part C: Emerging Technologies, vol. 89, Apr. 2018, pp. 384–406.

[2] G. Chance, A. Ghobrial, K. McAreavey, S. Lemaignan, T. Pip, K. Eder, "On Determinism of Game Engines used for Simulation-based Autonomous Vehicle Verification", arxiv, Apr. 7, 2021.

[3] M. Fields, R. Brewer, H. Edge, J. Pusey, E. Weller, et al., "simulation tools for robotics research and assessment", Proc. SPIE 9837, Unmanned Systems Technology XVIII, May 13, 2016.

[4] T. Erez, Y. Tassa and E. Todorov, "Simulation tools for model-based robotics: Comparison of Bullet, Havok, MuJoCo, ODE and PhysX," 2015 IEEE International Conference on Robotics and Automation (ICRA), 2015, pp. 4397-4404, doi: 10.1109/ICRA.2015.7139807.

[5] "High Mobility Multi-Purpose Wheeled Vehicle (HMMWV)", Military.com, [Online] Available: https://www.military.com/equipment/high-mobility-multipurpose-wheeled-vehicle-hmmwv, [Accessed: Nov. 04, 2021].

[6] "Unreal Engine", Epic Games, 2021, [Online] Available: https://www.unrealengine.com/en-US/, [Accessed: Nov. 04, 2021].

[7] "Evaluation", Keweenaw Research Center, Michigan Technological University, 2021, [Online] Available: https://www.mtu.edu/krc/evaluation/, [Accessed: Nov. 04, 2021].

[8] D. P. Brutzman, Y. Kanayama and M. J. Zyda, "Integrated simulation for rapid development of autonomous underwater vehicles," Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology, 1992, pp. 3-10, doi: 10.1109/AUV.1992.225199.

[9] S. Shah, D. Dey, C. Lovett, A. Kapoor, "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles", in Field and Service Robotics, Nov. 3, 2017, pp. 621–635

[10] T. Pollok, L, Junglas, B. Ruf, A. Schumann, "UnrealGT: Using Unreal Engine to Generate Ground Truth Datasets", in Advances in Visual Computing, Lecture Notes in Computer Science, vol. 11844, pp. 670-682, Oct. 21, 2019.

[11] T. T. M. Tran, C. Parker and M. Tomitsch, "A Review of Virtual Reality Studies on Autonomous Vehicle–Pedestrian Interaction," in IEEE Transactions on Human-Machine Systems, vol. 51, no. 6, pp. 641-652, Dec. 2021, doi: 10.1109/THMS.2021.3107517.

[12] J. P. Espineira, J. Robinson, J. Groenewald, P. H. Chan and V. Donzella, "Realistic LiDAR With Noise Model for Real-Time Testing of Automated Vehicles in a Virtual Environment," in IEEE Sensors Journal, vol. 21, no. 8, pp. 9919-9926, Apr. 15, 2021.

[13] "Unreal Engine Forms", Epic Games, 2021, [Online] Available: https://forums.unrealengine.com/, [Accessed: Nov. 04, 2021]

[14] "Unreal Engine 4 C++ Source Code via Github", Epic Games, 2021, [Online] Available: https://www.unrealengine.com/en-US/ue4-on-github, [Accessed: Nov. 04, 2021]

[15] Z. Jeffries, C. Majhor, J. Carter, S. Kysar, J. P. Bos, "MUONS path planning performance for a vehicle with complex suspension in Unreal", in Proc. SPIE 11748, Autonomous systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2021, Apr. 12, 2021.

[16] P. Young, S. Kysar, J.P. Bos, "Unreal as a simulation environment for off-road autonomy", in Proc. SPIE 11415, Autonomous Systems: Sensors, Processing, and Security for Vehicles and Infrastructure 2020, May 19, 2020.

[17] R.C. Coulter, "Implementation of the Pure Pursuit Path Tracking Algorithm", The Robotics Institute, Carnegie Mellon University, Jan. 1992.

[18] S. Wang, S. Fu, B. Li and S. Wang, "Path Tracking Control of Tracked Paver Based on Improved Pure Pursuit Algorithm," 2021 40th Chinese Control Conference (CCC), 2021, pp. 4187-4192, doi: 10.23919/CCC52363.2021.9550395.

[19] "Blender", The Blender Foundation, 2021, [Online] https://www.blender.org/, [Accessed: Nov. 04, 2021]

[20] "PROJ", OS Geo Project, 2021, [Online] https://proj.org/index.html, [Accessed: Nov. 04, 2021]

[21] J. Snider, "Automatic Steering Methods for Autonomous Automobile Tracking", Robotics Institute, Carnegie Mellon University, Feb., 2009.

[22] Z. Lu, B. Shyrokau, B. Boulkroune, S. van Aalst and R. Happee, "Performance benchmark of state-of-the-art lateral path-following controllers", 2018 IEEE 15th International Workshop on Advanced Motion Control (AMC), 2018, pp. 541-546, doi: 10.1109/AMC.2019.8371151

[23] H. Niu, Y. Lu, A. Savvaris and A. Tsourdos, "Efficient path following algorithm for unmanned surface vehicle", OCEANS 2016 - Shanghai, 2016, pp. 1-7, doi: 10.1109/OCEANSAP.2016.7485430.

[24] NVIDIA, "NVIDIA PhysX SDK 4.1 Documentation", NVIDIA PhysX, Apr. 18, 2021, [Online] Available: https://gameworksdocs.nvidia.com/PhysX/4.1/documentation/physxguide/Index.html, [Accessed: Feb. 08, 2022].

[25] D. Talwar, S. Guruswamy, N. Ravipati and M. Eirinaki, "Evaluating Validity of Synthetic Data in Perception Tasks for Autonomous Vehicles", 2020 IEEE International Conference On Artificial Intelligence Testing (AITest), 2020, pp. 73-80, doi: 10.1109/AITEST49225.2020.00018.

[26] S. Loze, "Autonomous vehicle development and training meets user experience at VI-grade", Epic Games, Nov. 11, 2021, [Online] Available: https://www.unrealengine.com/en-US/spotlights/autonomous-vehicle-development-and-training-meets-user-experience-at-vi-grade, [Accessed: Feb. 08, 2022].

[27] P. Schmitt, et al., "nuReality: A VR environment for research of pedestrian and autonomous vehicle interactions", Cornell University, arXiv preprint arXiv:2201.04742, Jan. 12, 2022.

[28] H. Wang, B. Liu, X. Ping and Q. An, "Path Tracking Control for Autonomous Vehicles Based on an Improved MPC," in IEEE Access, vol. 7, pp. 161064-161073, 2019, doi: 10.1109/ACCESS.2019.2944894.

[29] "Performance and Profiling", Epic Games, Unreal Engine, 2022, [Online] Available: https://docs.unrealengine.com/4.27/en-US/TestingAndOptimization/PerformanceAndProfiling/, [Accessed: Feb. 10, 2022]

[30] Y. Chen, S. Chen, T. Zhang, S. Zhang and N. Zheng, "Autonomous Vehicle Testing and Validation Platform: Integrated Simulation System with Hardware in the Loop*", 2018 IEEE Intelligent Vehicles Symposium (IV), 2018, pp. 949-956, doi: 10.1109/IVS.2018.8500461.

[31] B. Mashadi, M. Majidi, "Global optimal path planning of an autonomous vehicle for overtaking a moving obstacle", Latin American Journal of Solids and Structures, Dec. 12, 2014, doi:10.1590/S1679-78252014001400002.

[32] A. Peirt, E. Karpman, et. al., "Modelling of off-road wheeled vehicles for real-time dynamic simulation", Journal of Terramechanics, vol. 97, pp. 45-58, Oct. 2021, https://doi.org/10.1016/j.jterra.2021.04.001.

[33] A. Singh, D. Holtz, "Modeling Off-Road Rollover Using Terramechanics For Real Time Driving Simulator", 2014 NDIA Ground Vehicle Systems Engineering and Technology Symposium, Novi, MI, Aug. 12-14, 2014.

[34] A. Aquilio, "A Framework for Dynamic Terrain with Application

in Off-road Ground Vehicle Simulations", Computer Science Dissertations, Georgia State University, Feb. 08, 2006. [Online] Available: https://scholarworks.gsu.edu/cs_diss/11/ [Accessed: Feb. 10, 2022].

[35] X. Yuan, G. Huang and K. Shi, "Improved Adaptive Path Following Control System for Autonomous Vehicle in Different Velocities", in IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 8, pp. 3247-3256, Aug. 2020, doi: 10.1109/TITS.2019.2925026.

[36] X. Dai, C. Ke, Q. Quan and K. -Y. Cai, "Simulation Credibility Assessment Methodology With FPGA-based Hardware-in-the-Loop Platform", in IEEE Transactions on Industrial Electronics, vol. 68, no. 4, pp. 3282-3291, April 2021, doi: 10.1109/TIE.2020.2982122.

[37] Y. Gao, Z. Xu, X. Zhao, G. Wang and Q. Yuan, "Hardware-in-the-Loop Simulation Platform for Autonomous Vehicle AEB Prototyping and Validation", 2020 IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC), 2020, pp. 1-6, doi: 10.1109/ITSC45102.2020.9294704.

[38] X. Che, C. Li and Z. Zhang, "A Test Method for Self-driving Vehicle Based on Mixed Reality", 2021 IEEE International Conference on Smart Internet of Things (SmartIoT), 2021, pp. 401-405, doi: 10.1109/SmartIoT52359.2021.00075.

[39] N. Mowerin, "Scale and Measurement Inside Unreal Engine 4", techarthub,

2022, [Online] Available: https://www.techarthub.com/scale-and-measurement-inside-unreal-engine-4/, [Accessed: Feb. 16, 2022].

[40] NVIDIA, "User's Guide, Vehicles", NVIDIA PhysX SDK 3.4.0 Documentation, 2022, [Online] Available: https://docs.nvidia.com/gameworks/content/gameworkslibrary/physx/guide/Manual/Vehicles.html, [Accessed: Feb. 18, 2022]

[41] MathWorks, "Pure Pursuit Controller", MATLAB, 2021b, 2022, [Online] Available: https://www.mathworks.com/help/nav/ref/controllerpurepursuit-system-object.html, [Accessed: Feb. 18, 2022].

[42] A. Overvoorde, "Vulkan Tutorial", Vulkan, Apr. 2020, [Online] Available: https://github.com/Overv/VulkanTutorial, [Accessed: Mar. 20, 2022].

[43] "OpenGL Headline News", Khronos Group, Jun. 4, 2021, [Online] Available: https://www.opengl.org/, [Accessed: Mar. 20, 2022].

[44] "Programming in C++", Epic Games, Unreal Engine v4.27, 2022, [Online] Available: https://docs.unrealengine.com/4.27/en-US/ProgrammingAndScripting/ProgrammingWithCPP/, [Accessed: Mar. 20, 2022]

[45] O. Swierad, "Measuring Performance", Unreal Art Optimization, 2019, [Online] Available: https://unrealartoptimization.github.io/book/process/measuring-performance/, [Accessed: Mar. 3, 2022].

[46] F. Messaoudi, A. Ksentini, G. Simon, P. Bertin, "Performance Analysis of Game Engines on Mobile and Fixed Devices", ACM Transactions on Multimedia Computing, Communications, and Applications, vol. 13, no. 4, Nov. 2017, [Online] Available: https://dl-acm-org.services.lib.mtu.edu/doi/abs/10.1145/3115934, [Accessed: Mar. 20, 2022]

[47] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo and S. Thrun, "Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing", 2007 American Control Conference, 2007, pp. 2296-2301, doi: 10.1109/ACC.2007.4282788.

[48] S. Rathinam, et al., "Autonomous Searching and Tracking of a River using an UAV", 2007 American Control Conference, 2007, pp. 359-364, doi: 10.1109/ACC.2007.4282475.

[49] D. Kamran, J. Zhu and M. Lauer, "Learning Path Tracking for Real Car-like Mobile Robots From Simulation", 2019 European Conference on Mobile Robots (ECMR), 2019, pp. 1-6, doi: 10.1109/ECMR.2019.8870947.

[50] C. Hu, R. Wang, F. Yan and N. Chen, "Should the Desired Heading in Path Following of Autonomous Vehicles be the Tangent Direction of the Desired Path?", in IEEE Transactions on Intelligent Transportation Systems, vol. 16, no. 6, pp. 3084-3094, Dec. 2015, doi: 10.1109/TITS.2015.2435016.

Appendix A

Additional Figures

Appendix Fig. A.1 - Fig. A.9 displays the PID controller velocity control for each different maneuver performed during the experiment. The velocity of the UE4 model is plotted against the trace velocity of the input data.

Appendix Fig. A.10 - Fig. A.15 displays the additional CTE during simulation execution not given during Chapter 4 for each omitted maneuver besides the SS. These figures further display the oscillation of the vehicle model when following a set path at lower simulation frequencies.

A.1 PID Velocity Control

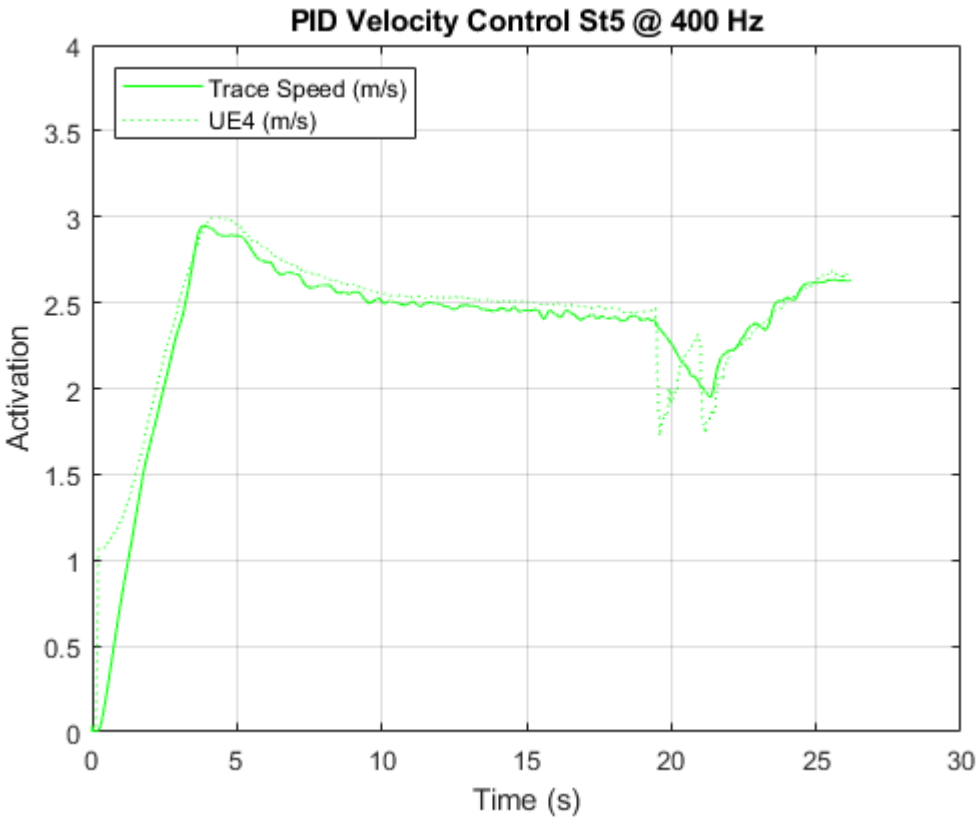

Figure A.1: PID Control St5 400Hz

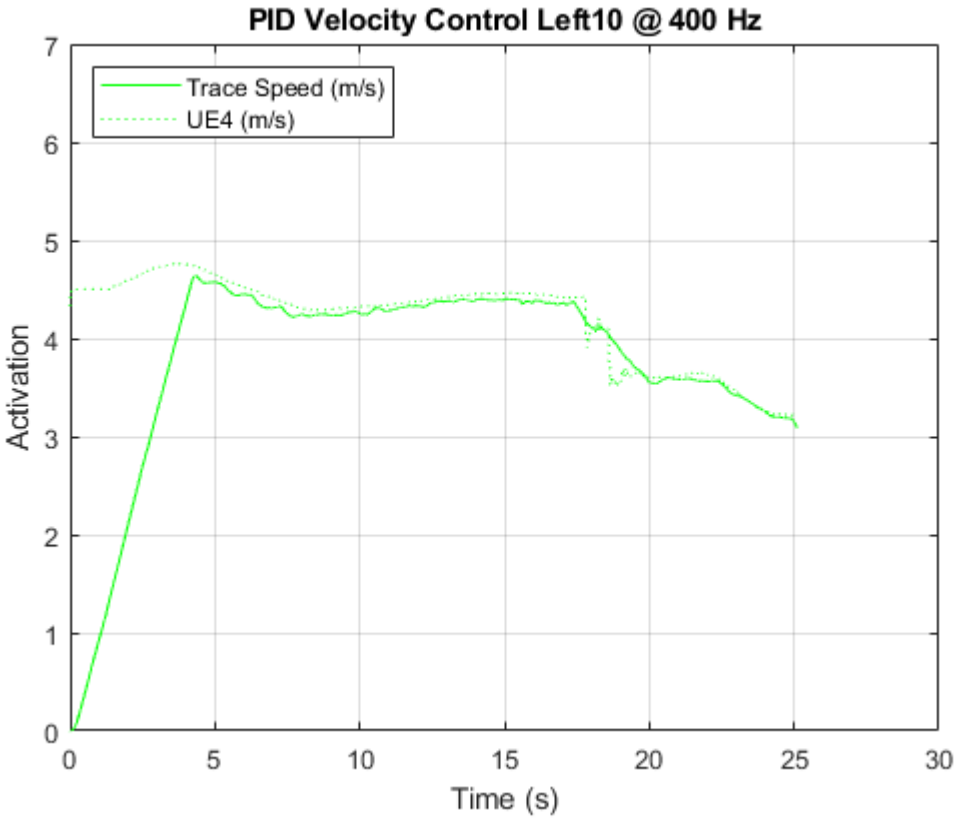

Figure A.2: PID Control Left10 400Hz

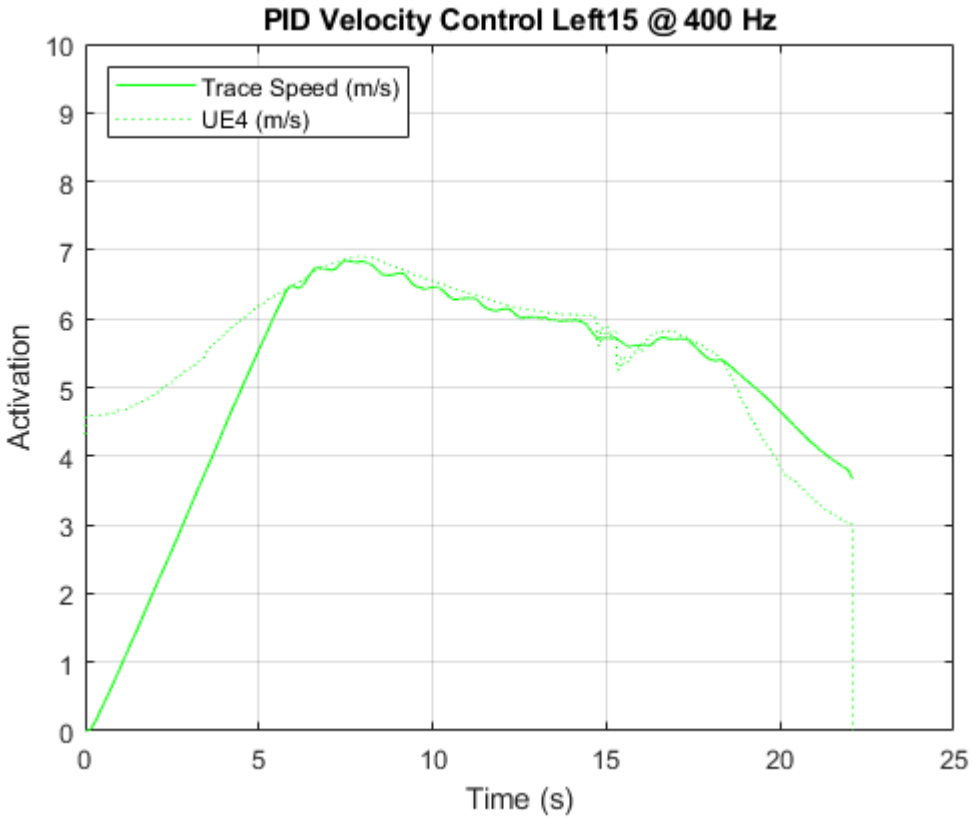

Figure A.3: PID Control Left15 400Hz

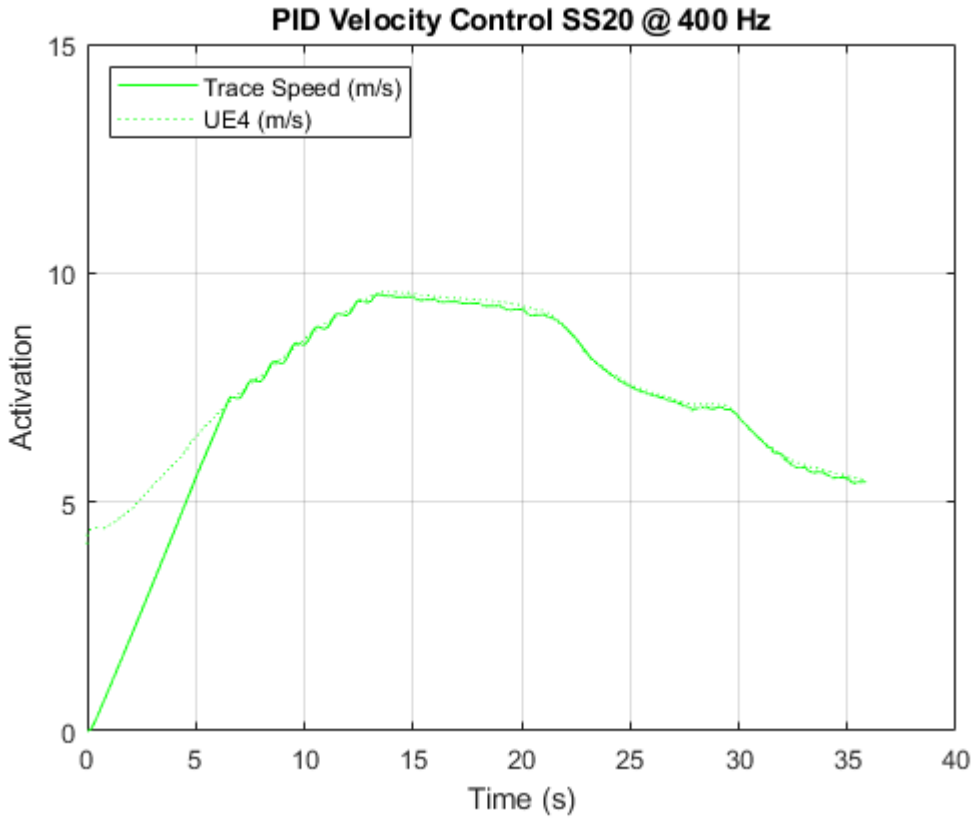

Figure A.4: PID Control SS20 400Hz

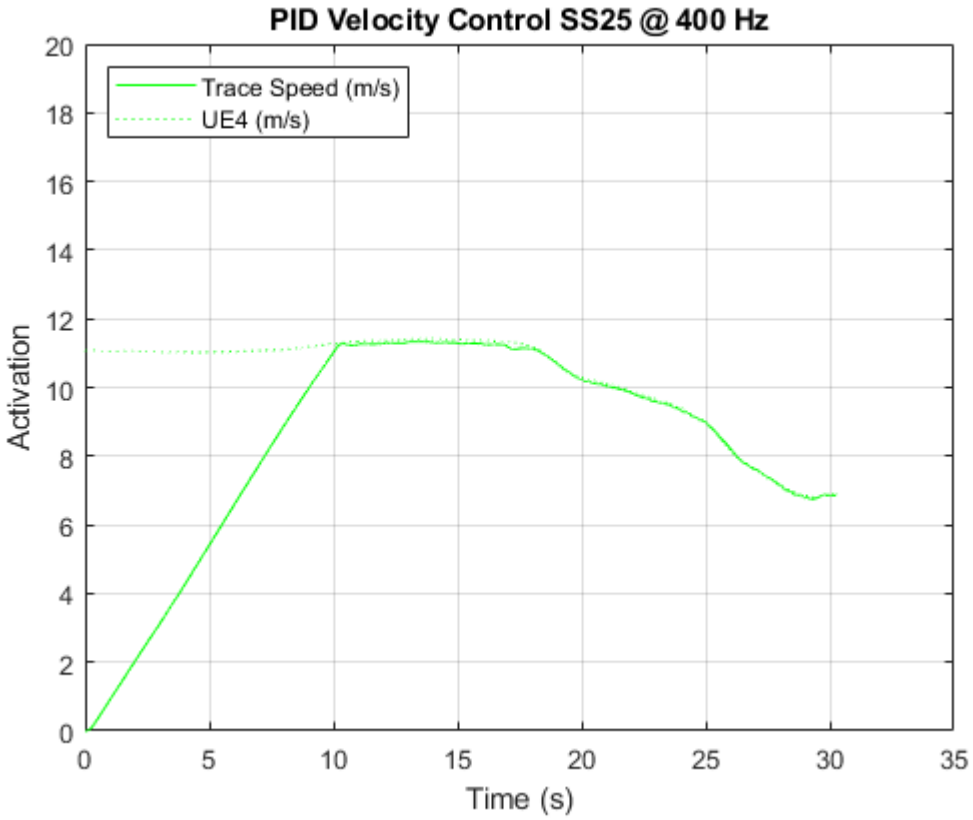

Figure A.5: PID Control SS25 400Hz

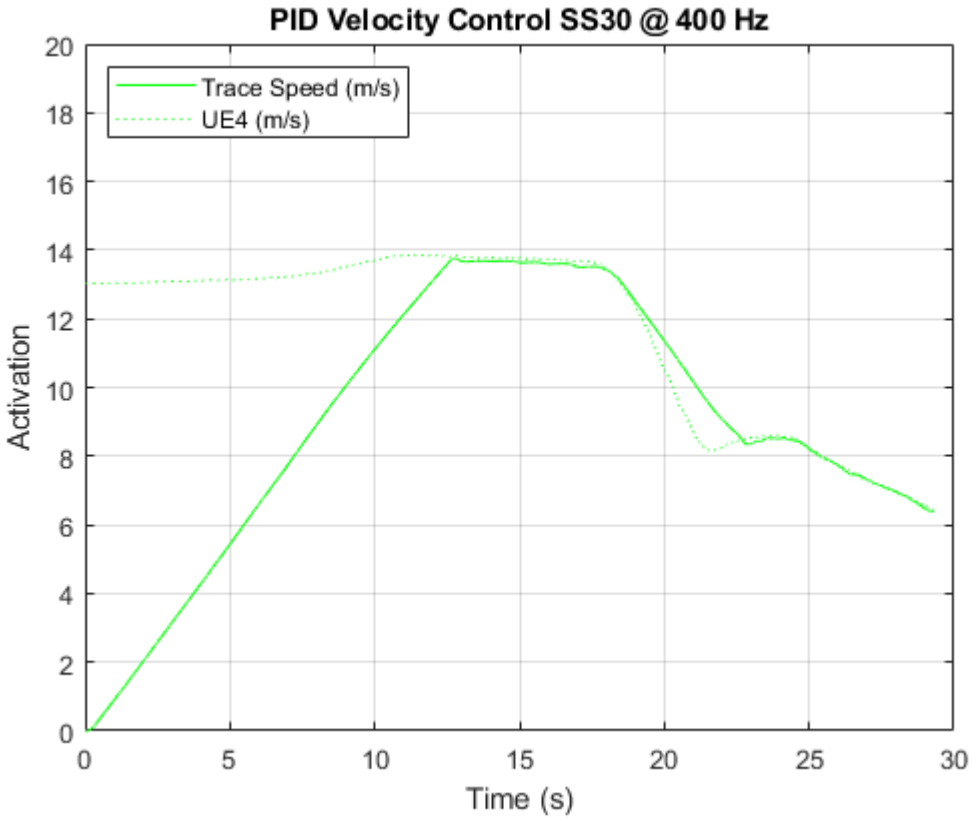

Figure A.6: PID Control SS30 400Hz

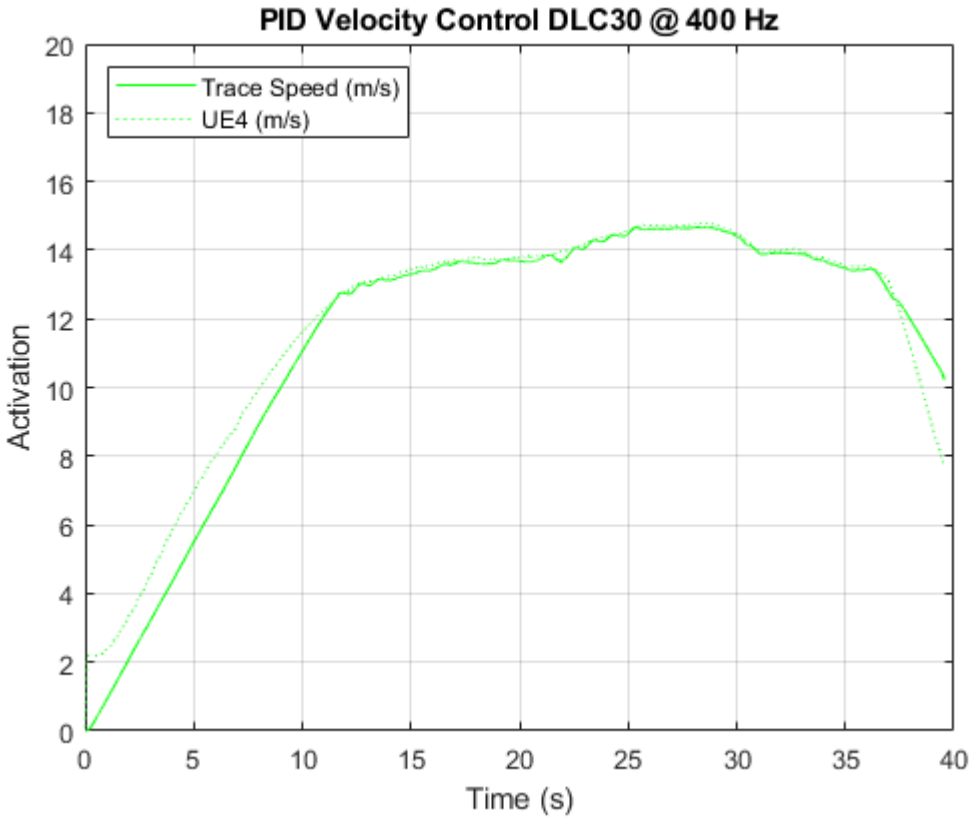

Figure A.7: PID Control DLC30 400Hz

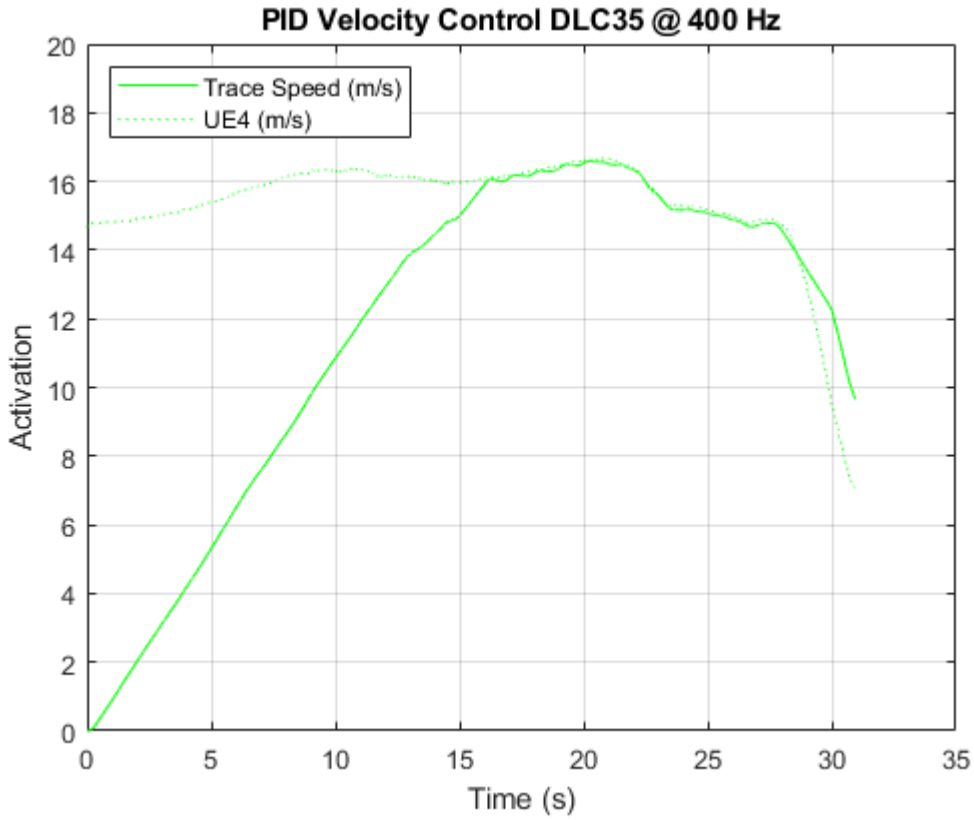

Figure A.8: PID Control DLC35 400Hz

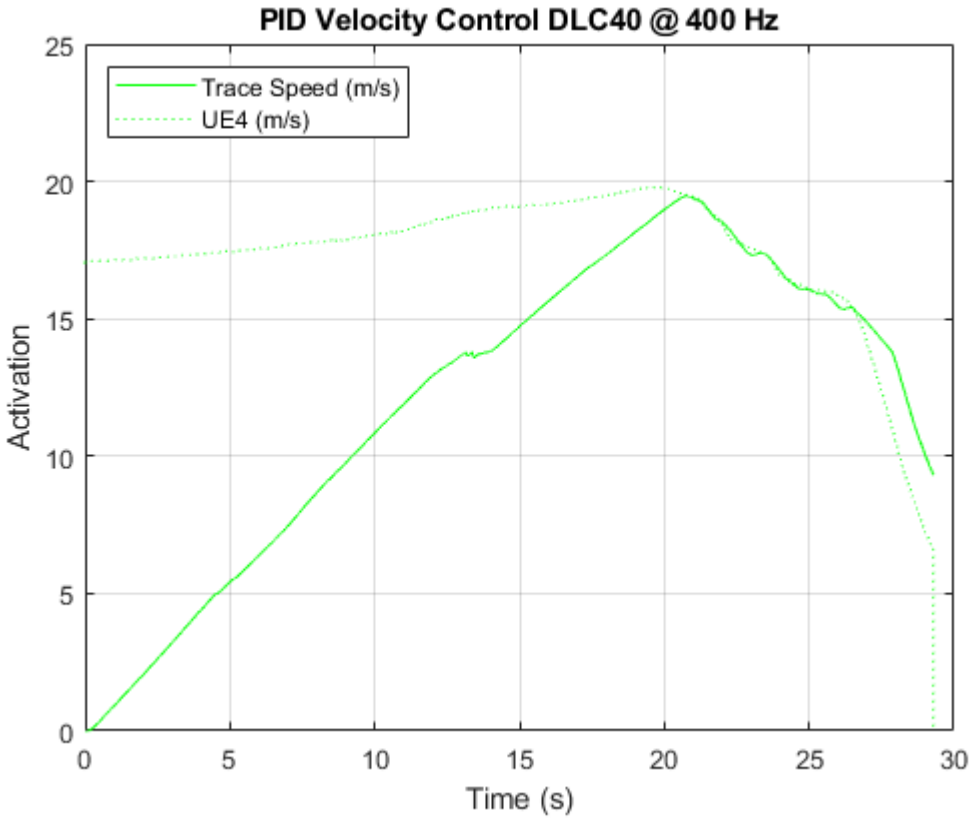

Figure A.9: PID Control DLC40 400Hz

A.2 CTE Error

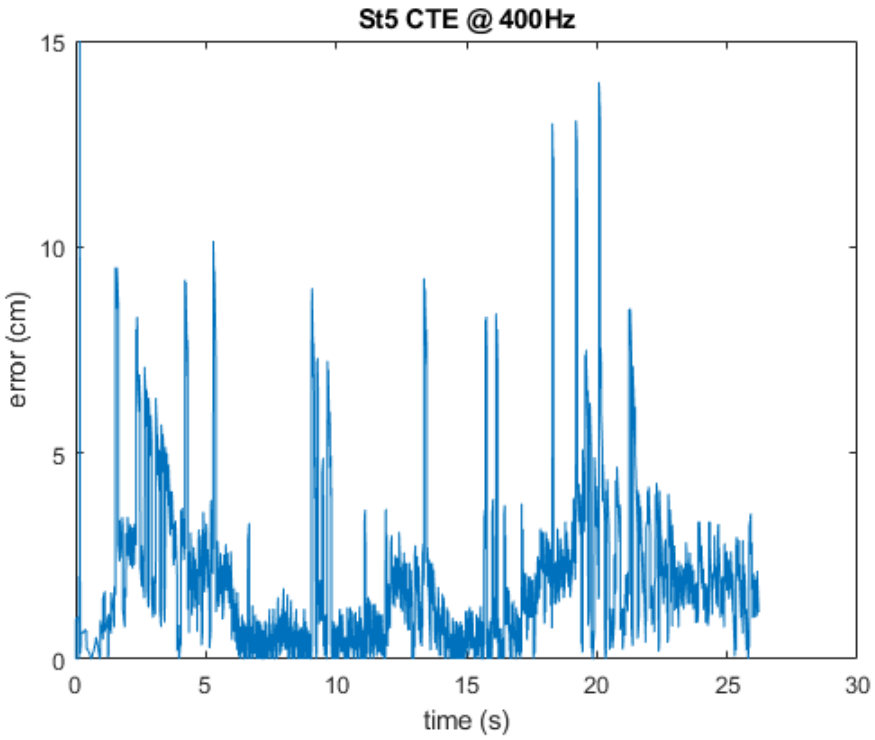

Figure A.10: CTE Error St5 400Hz

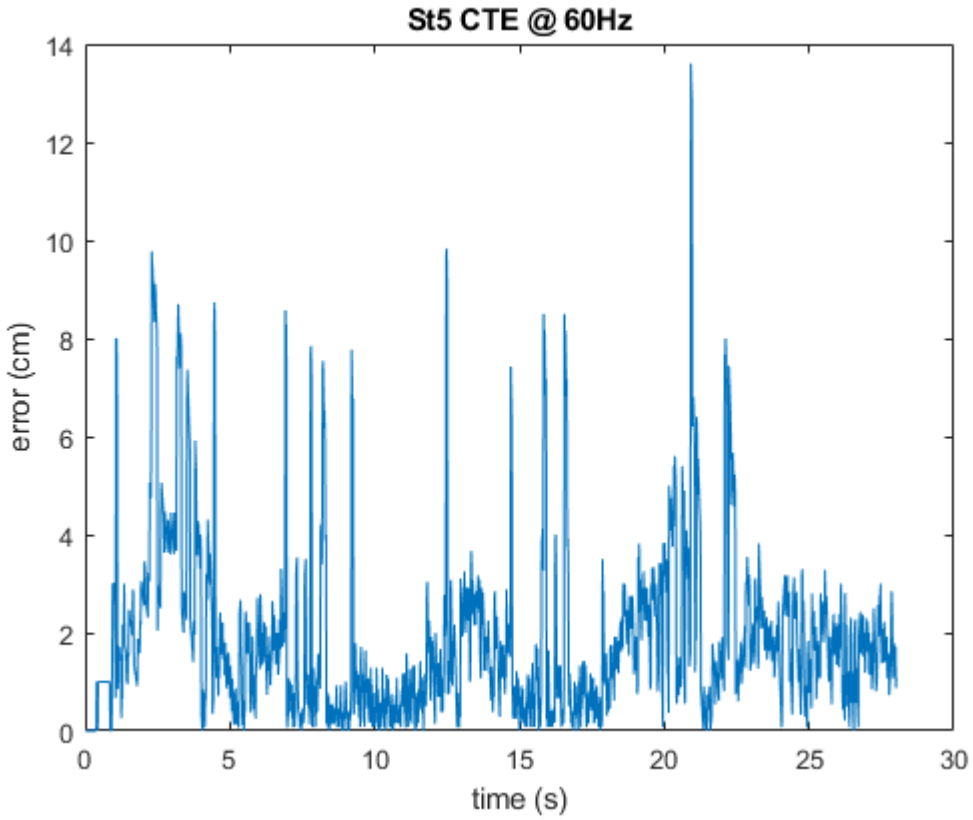

Figure A.11: CTE Error St5 60Hz

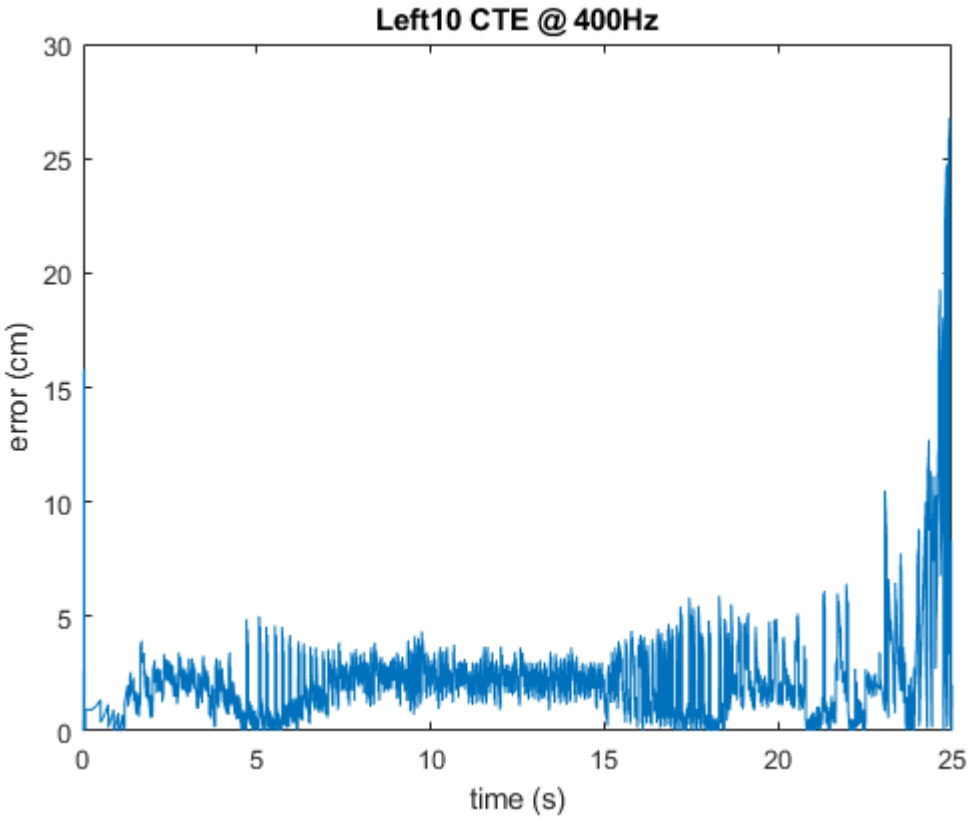

Figure A.12: CTE Error Left10 400Hz

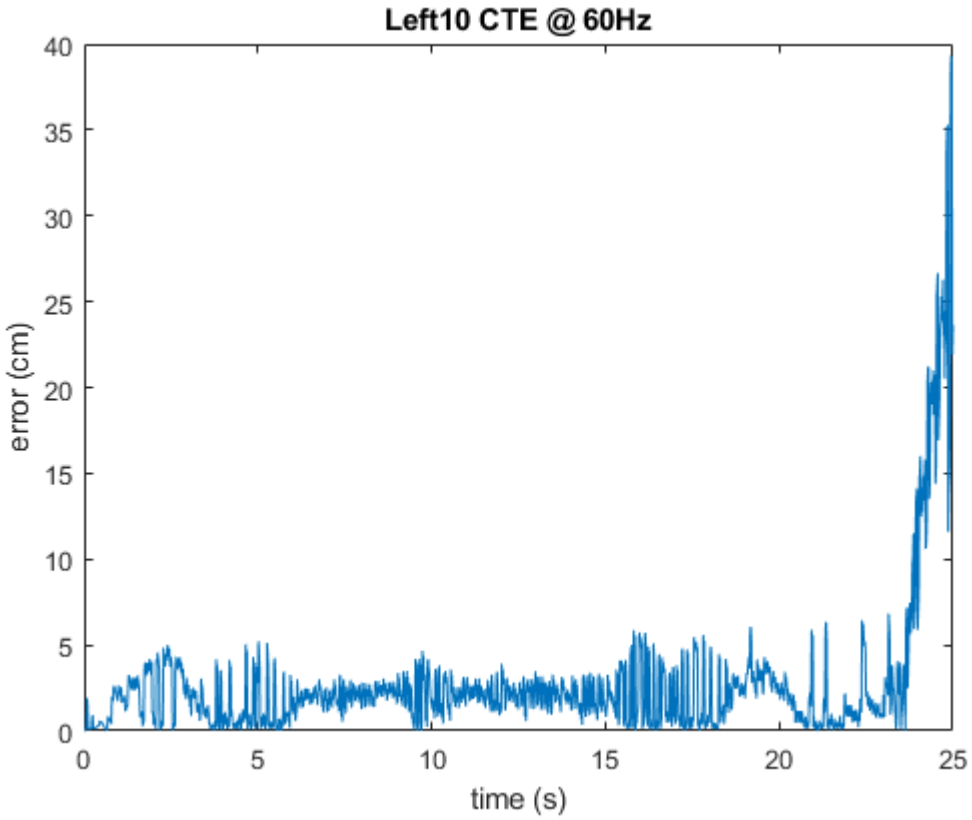

Figure A.13: CTE Error Left10 60Hz

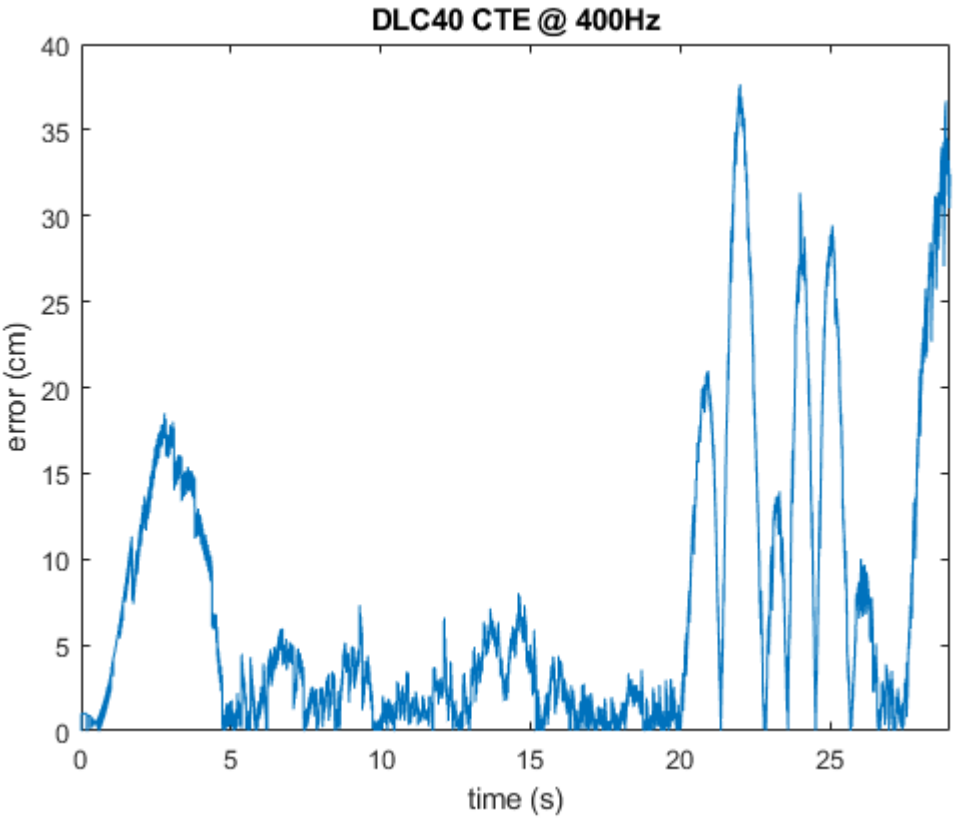

Figure A.14: CTE Error DLC40 400Hz

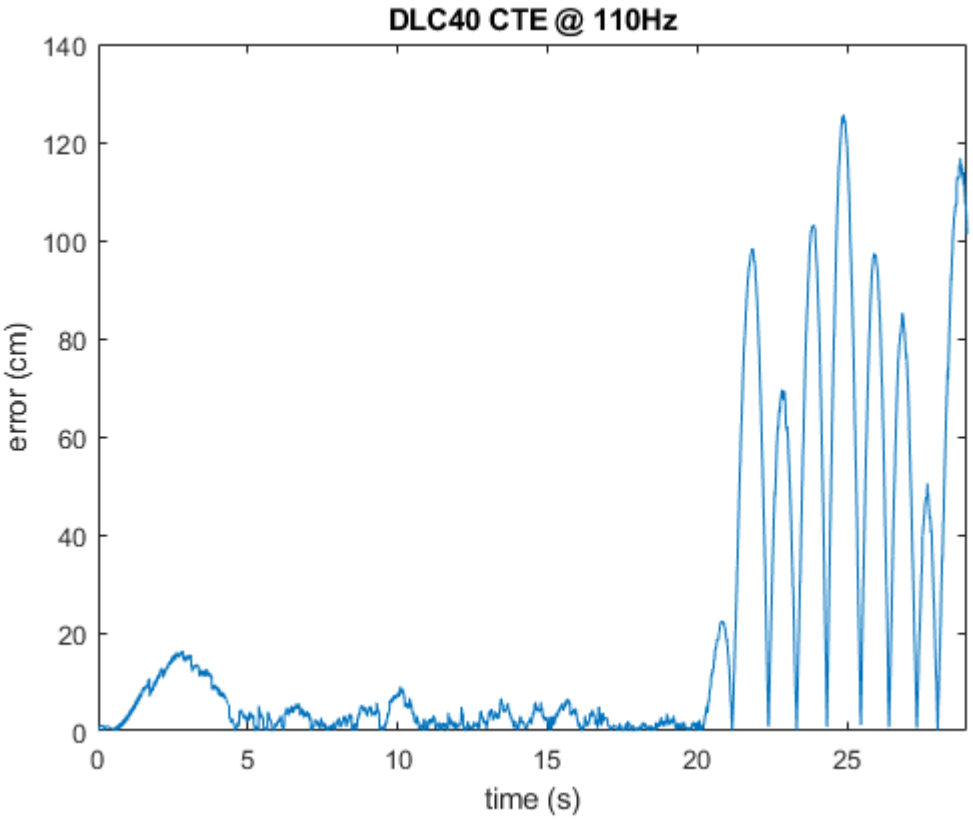

Figure A.15: CTE Error DLC40 110Hz

Appendix B

C++ Functions

Sections B.1 and B.2 contain the implementation of the .h and .cpp Pure Pursuit and look-ahead algorithms. The following code is embedded within the UE4 main function that runs during simulation execution.

B.1 PurePursuit_Controller.h

```
#pragma once
#include "CoreMinimal.h"
#include "Kismet/KismetMathLibrary.h"
#include "Kismet/KismetSystemLibrary.h"
#include <math.h>
#include "PID.h"

class PurePursuit_Controller
{
public:
  float GetSteeringAngle(int k, int currStep);
  float FindLookAhead(int currStep);
    float GetAcceleration();

  // velocity pid variables
  float pv = 6.0;
  float iv = 0.25;
  float dv = 0;

  // Other class variables
  float ceilingSpeed = 0;
  float prevMaxSpeed = 0;
  int preK = 0;
  int i = 0;
  int K = 10;
  float steerMem[10] = {};

  // Initialize pid controller
  PID pidVelocity = PID(pv, iv, dv);
```

```
private:
  FVector position;
  FVector destination;
  FRotator rotation;
};
```

B.2 PurePursuit_Controller.cpp

```
#include "PurePursuit_Controller.h"

/*** Calculate the value to turn the front tires ***/
float PurePursuit_Controller::GetSteeringAngle(int K, ↩
   int currStep)
{
  // Init var and save previous time step place
  lookAhead = FindLookAhead(K);
  float avgResult = 0;

  // Get Steering Angles
  for (int i = 0, i < K, i++)
  {
    // Init positions and rotations for steering ↩
       calculation
    position = FVector(inputData.currX, inputData.currY,↩
        inputData.currZ);
    destination = trace.GetDestination(currStep + i + ↩
       lookAhead);
    rotation.Roll = inputData.currR;
    rotation.Pitch = inputData.currP;
    rotation.Yaw = inputData.currH;

    // Use engine functions to find yaw angle for ↩
       timestep
    FRotator findRotation = UKismetMathLibrary::↩
       FindLookAtRotation(position, destination);
    FRotator newRotation = UKismetMathLibrary::↩
       NormalizedDeltaRotator(findRotation, rotation);
```

```
    float yawRotation = FMath::↩
       GetMappedRangeValueUnclamped(FVector2D(-45.0, ↩
       45.0), FVector2D(-1.0, 1.0), newRotation.Yaw);
    float preRotation = FMath::↩
       GetMappedRangeValueUnclamped(FVector2D(-45.0, ↩
       45.0), FVector2D(-1.0, 1.0), inputData.steerAngle)↩
       ;
    steerMem[i] = yawRotation;

  }

    for (int j = 0; j < K; j++)
    {
        avgResult = steerMem[j] + avgResult;
    }
    avgResult = avgResult / K;
  }

  // Return steering angle rotation
  return avgResult;
}

/*** Determine if the vehicle is traveling in a straight↩
    line, if so then move the look ahead distance
forward, if a turn is approaching then move it closer. ↩
   Max = 200, Min = 10 ***/
float PurePursuit_Controller::FindLookAhead(int currStep↩
   )
{
  // Init variables
  int phi       = currStep;
  int i       = currStep;
  int j       = 10;
  int ex        = 0;
```

```
int max_phi     = 200; // max look ahead is 0.5 second
float steerAngle  = 0;
float yawRotation = 0;
float steerThresh = 2.0;
FVector destLook;

// Look until either max look ahead is reached or a ↩
   turn is approaching
while (ex != 1)
{
  // Current Look ahead destination
  phi = i + j;

  // Init positions and rotations
  destLook = GetDestination(phi);

  // Init positions and rotations for steering ↩
     calculation
  position = FVector(inputData.currX, inputData.currY,↩
      inputData.currZ);
  rotation.Roll = inputData.currR;
  rotation.Pitch = inputData.currP;
  rotation.Yaw = inputData.currH;

  // Use engine functions to find yaw angle
  FRotator findRotation = UKismetMathLibrary::↩
     FindLookAtRotation(position, destLook);
  FRotator newRotation = UKismetMathLibrary::↩
     NormalizedDeltaRotator(findRotation, rotation);
  yawRotation = FMath::GetMappedRangeValueUnclamped(↩
     FVector2D(-45.0, 45.0), FVector2D(-1.0, 1.0), ↩
     newRotation.Yaw);
```

```
    steerAngle = FMath::GetMappedRangeValueUnclamped(↩
       FVector2D(-1.0, 1.0), FVector2D(-45.0, 45.0), ↩
       yawRotation);

    // Determin if the path is straight or a turn ↩
       approaches. Set look ahead accordingly
    if ((steerAngle > (steerMax * steerThresh) && ↩
       steerAngle < steerThresh) && (j != max_k))
    {
      j++;
    }
    else
    {
      ex = 1;
    }
  }

  phi = j;

  // Return look ahead value
  return phi;
}

/*** Use the PID control library to return an ↩
   acceleration to reach the desired velocity ***/
float PurePursuit_Controller::GetAcceleration()
{
  float pidResult = pidVelocity.calculate(inputData.↩
     maxSpeed, inputData.currSpeed);
  return pidResult;
}
```