Path Following and Obstacle Avoidance for Autonomous Ground Vehicles Using Nonlinear Model Predictive Control

by

Robert Brothers

A thesis submitted to the Graduate Faculty of
Auburn University
in partial fulfillment of the
requirements for the Degree of
Master of Science

Auburn, Alabama
May 2, 2020

Keywords: model, predictive, control, avoidance

Approved by

David Bevly, Chair, Professor of Mechanical Engineering
George Flowers, Professor of Mechanical Engineering
John Hung, Professor of Electrical and Computer Engineering

Abstract

This thesis presents a nonlinear model predictive controller (NMPC) for path following and obstacle avoidance in automated driving systems. Automated safety control systems have been increasingly effective at reducing the number of traffic fatalities in the United States. Many of the commercially available safety systems are still only classified as SAE level 1 and level 2 autonomy features. To progress towards SAE level 3 and level 4 automated driving systems, obstacle avoidance control must be added to the vehicle's dynamic driving task, removing human drivers from the control loop. Many current level 2 automated driving systems, such as Auburn University's heavy truck platooning system, could progress towards full autonomy by incorporating obstacle avoidance control into their existing control architectures. The NMPC control module developed in this work is designed to take advantage of an automated vehicle's existing software stack to provide enhanced path tracking and obstacle avoidance maneuvering.

Two simple vehicle models, a kinematic model and a dynamic bicycle model, are developed identified and implemented in a flexible NMPC software library which is feasible for real-time control of an autonomous vehicle. In a series of simulation and real-time experiments, a detailed tuning procedure and performance evaluation for both NMPC implementations are given. The kinematic model implementation is also shown to work as a replacement controller in Auburn University's existing software architecture for long distance, non-line-of-sight following of a manually driven leader vehicle. Obstacle avoidance is added to each controller implementation through a set of hard constraints. The feasibility of this constraint method is demonstrated with two simulated obstacle avoidance scenarios. Future improvements to both the obstacle avoidance method and path tracking accuracy are discussed.

Acknowledgments

First, I have to thank my family for loving, supporting, and encouraging me through my graduate education. Without my parents, Robert and Polly, I never would have made it to this point in my life. I attribute my love for learning and my work ethic to them. Special thanks must also be given to my wonderful girlfriend, Ashley. She has been patient and supportive throughout all the ups and downs of my graduate work.

I also have to thank Dr. Bevly for giving me this opportunity to work on interesting and fullfilling projects during my graduate education. He originally inspired me during my undergraduate studies to continue learning about the control of dynamic systems and to push my skills as a mechanical engineer. I have learned so much from him and Dr. Scott Martin during my three years in the GAVLab. I hope to get a chance to work with both of them again and to see them again out on the slopes in Utah.

There are so many other fellow GAVLab members that I have to thank. This thesis certainly would not have been possible without the mentorship and prior work that Dan Pierce and Grant Apperson provided. Thank you both for your generosity, guidance, and friendship. Thanks must also be given to Tanner Ray and Stephen Geiger for always keeping things lighthearted and fun in our office, even during some of our second-shift work hours. Special thanks to my friend and office-mate Patrick Smith for always being available to bounce ideas off of. He has provided immeasurable ammounts of help towards finishing this thesis. Thank you to all my other past and present office-mates: Troupe Tabb, Luke Kamrath, Dan Kamrath, Nate Kamrath, Houston Cleveland, and Jake Ward. I really enjoyed celebrating festivus with you all every Thursday, because one airing of grievances on December 23rd is just not enough.

Table of Contents

Abstract ii

Acknowledgments iii

List of Abbreviations xiii

1 Introduction 1

1.1 Background and Motivation 1

1.2 Prior Research 4

1.3 Contributions 10

1.4 Thesis Outline 10

2 Ground Vehicle Modeling and Simulation 12

2.1 Vehicle Coordinate Frames 12

2.2 Kinematic Modeling 14

2.3 Yaw Dynamic Modeling 20

2.4 Simulator Modeling 21

3 Model Predictive Control 26

3.1 The Receding Horizon Control Principle 26

3.2 Optimization Problem Setup 28

3.3 Path Following and Obstacle Avoidance MPC 34

3.4 MPC Implementation 39

4 Automated Vehicle System Overview: Auxiliary MPC Interfaces 45
4.1 DRTK/TDCP Path Following 45
4.2 Current System Hardware and Software Architecture 48
4.3 Obstacle Detection and Tracking 51
5 Simulation and Experimentation 54
5.1 Experimental Vehicle Setup 54
5.2 NMPC Performance Evaluation 57
5.2.1 Experiment Procedures 57
5.2.2 Horizon Sensitivity and Tuning 60
5.2.3 Performance Metrics 75
5.3 Obstacle Avoidance Application 80
5.3.1 Simulation Procedures 81
5.3.2 Obstacle Avoidance Results 83
5.4 Non-line-of-sight Path Following Application 95
5.4.1 Experiment Procedures 95
5.4.2 Non-line-of-sight following results 97
6 Conclusions and Future Work 100
6.1 Conclusions 100
6.2 Future Work 101
References 103
Appendices 113
A MKZ System Identification 114
B MPC With IPOPT: An Example Sparsity Encoding Problem 125

C NMPC Horizon Tuning Procedure and Results 140

List of Figures

1.1 Lives Saved from Driving Safety Technology 1960 to 2012 1

1.2 Path Planning Stack 5

1.3 MPC Path Planning Hierarchy Control Architecture 8

1.4 MPC Trajectory Control Architecture 9

2.1 Vehicle-fixed Coordinate System 13

2.2 Earth-fixed Coordinate Systems 14

2.3 4 Wheel Vehicle in Planar Motion 15

2.4 Bicycle Model in Steady-state Turning 17

2.5 Bicycle Model Free-body Diagram 19

2.6 Tire Force Curve for Pacejka's Magic Tire Model 20

2.7 Real Drive-by-wire Lincoln MKZ (left) and Simulated Drive-by-wire Lincoln MKZ (right) 23

2.8 Gazebo MKZ Model Kinematic Chain 24

2.9 Gazebo MKZ Model Kinematic Chain with Suspension Elements 25

3.1 Vehicle With State $x(0)$ and a Number of Way-points x_{des_i} in the State-space . 37

3.2 Circle-to-circle Collision Distance 38

3.3 NMPC Software Library UML Class Diagram 44

4.1 46

4.2 High Precision Change in Position Between Measurement Epochs from TDCP . 47

4.3 Virtual Leader Following Using DRTK/TDCP Measurements 48

4.4 Auburn's Current Hardware Suite for CACC Truck Platooning 49

4.5 Subset of Auburn's Current Software Architecture for CACC Truck Platooning that Relates to Way-point Path Generation 50

4.6 New Software Architecture Including MPC for Obstacle Avoidance and Way-point Following 53

5.1 Gazebo MKZ Drive-by-wire Software Interface 55

5.2 Real MKZ Drive-by-wire Software Interface 55

5.3 MKZ Drive-by-wire Software Interface: Real-time Controller Testing 56

5.4 MKZ Base-line Hardware Setup 57

5.5 Reference Paths for Performance Evaluation and Tuning Experimental Procedures 58

5.6 Gazebo Simulation Two-lane Road 58

5.7 Gazebo Plugin Published Path Visualized in RVIZ 59

5.8 NCAT Skid Pad Area 59

5.9 Skid Pad Reference Paths (1 cm Resolution) Created From RTK GPS 60

5.10 Kinematic Model NMPC Controller Horizon Tuning Results 62

5.11 Bicycle Model NMPC Controller Horizon Tuning Results 63

5.12 Kinematic Model NMPC Controlled Single Lane Change in Gazebo With Horizon Tuning $(N = 60, T = 0.75s)$ 64

5.13 Kinematic Model NMPC Controlled Single Lane Change in Real-time With Horizon Tuning $(N = 60, T = 0.75s)$ 65

5.14 Bicycle Model NMPC Controlled Single Lane Change in Gazebo With Horizon Tuning $(N = 100, T = 0.03s)$ 66

5.15 Bicycle Model NMPC Controlled Single Lane Change in Real-time With Horizon Tuning $(N = 100, T = 0.03s)$ 67

5.16 Kinematic Model NMPC Steering Control Output with Various Input Weights . 69

5.17 Kinematic Model NMPC Velocity Control Output with Various Input Weights . 70

5.18 Kinematic Model NMPC Path Variation due to Steering Input Weights 70

5.19 Kinematic Model NMPC Path Variation due to Velocity Input Weights 71

5.20 20 m/s Step Steer – Bicycle Model NMPC Velocity Control Output with Various Input Weights 72

5.21 20 m/s Step Steer – Bicycle Model NMPC Steering Control Output with Various Input Weights 72

5.22 10 m/s Step Steer – Bicycle Model NMPC Steering Control Output with Various Input Weights 73

5.23 10 m/s Step Steer – Bicycle Model NMPC Velocity Control Output with Various Input Weights 74

5.24 Bicycle Model NMPC Path Variation due to Steering Input Weight 74

5.25 Bicycle Model NMPC Path Variation due to Velocity Input Weights 74

5.26 Step Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed 76

5.27 Single Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed 77

5.28 Double Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed 77

5.29 Step Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed 78

5.30 Single Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed 79

5.31 Double Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed 79

5.32 RVIZ Visualization of an Empty Simulation World for Single Target Tracking and Obstacle Avoidance Control 81

5.33 RVIZ Visualization of the Published Path and Obstacle Position (left) and the Gazebo Simulation World with a Walking Pedestrian Model Acting as an Obstacle (right) 82

5.34 Single Target Avoidance Test: Target Placed Directly Ahead of the Vehicle and No Obstacle in the Path 83

5.35 Single Target Avoidance Test: Target Placed Offset From the Vehicle's Initial Heading and No Obstacle in the Path 84

5.36 Single Target Avoidance Test: Target Placed Directly Ahead of the Vehicle with an Obstacle in the Path 85

5.37 Kinematic Model NMPC Pedestrian Obstacle Avoidance at 1 m/s 85

5.38 Kinematic Model NMPC Pedestrian Obstacle Avoidance at 5 m/s 86

5.39 Kinematic Model NMPC Pedestrian Obstacle Avoidance at 10 m/s 86

5.40 Bicycle Model NMPC Pedestrian Obstacle Avoidance at 1 m/s 87

5.41 Bicycle Model NMPC Pedestrian Obstacle Avoidance at 5 m/s 88

5.42 Bicycle Model NMPC Pedestrian Obstacle Avoidance at 10 m/s 88

5.43 Prediction Horizon and Avoidance Constraint Lagrange Multipliers at Select Instants of the 10 m/s Pedestrian Avoidance Test Using the Kinematic Model NMPC 89

5.44 Prediction Horizon and Avoidance Constraint Lagrange Multipliers at Select Instants of the 10 m/s Pedestrian Avoidance Test Using the Bicycle Model NMPC 90

5.45 Kinematic Model NMPC Pedestrian Obstacle Avoidance at 5 m/s with Path Pre-processing 92

5.46 Kinematic Model NMPC Pedestrian Obstacle Avoidance at 10 m/s with Path Pre-processing 92

5.47 Bicycle Model NMPC Pedestrian Obstacle Avoidance at 5 m/s with Path Pre-processing 93

5.48 Bicycle Model NMPC Pedestrian Obstacle Avoidance at 10 m/s with Path Pre-processing 94

5.49 MKZ Test Vehicle Setup as an Autonomous Follower to the Manually Driven Kia Optima Leader Vehicle 96

5.50 Kia Optima Leader Vehicle Hardware Setup 96

5.51 Kinematic Model NMPC Following a Circular Path Generated from DRTK/TDCP 97

5.52 Kinematic Model NMPC Non-line-of-sight Following a Leader Vehicle 98

5.53 Sample Control Iterations During Non-line-of-sight Following 99

A.1 Gazebo Simulation Constant Radius (50 m) Test Set-up 117

A.2 MKZ Constant Radius (25 m) Test Track Path 117

A.3 Gazebo MKZ Constant Radius Test Results and Fit of Understeer Gradient . . . 118

A.4 MKZ Test Vehicle Constant Radius Test Results and Fit of Understeer Gradient 118

A.5 Gazebo MKZ Step Steer Test Model Comparison: 1 m/s 119

A.6 Gazebo MKZ Step Steer Test Model Comparison: 5 m/s 119

A.7 Gazebo MKZ Step Steer Test Model Comparison: 10 m/s 120
A.8 MKZ Vehicle Step Steer Test Model Comparison: 1 m/s 120
A.9 MKZ Vehicle Step Steer Test Model Comparison: 5 m/s 121
A.10 MKZ Vehicle Step Steer Test Model Comparison: 10 m/s 121
A.11 Bicycle Model Frequency Response for Real MKZ Vehicle 122
A.12 MKZ Model Comparisons: Step Steer Response at Various Speeds 124
B.1 GAVLab ATRV Differential Drive Robot 126
B.2 NMPC Software Library UML Class Diagram 132
C.1 Kinematic Model NMPC Horizon Tuning Map: 1 m/s 141
C.2 Kinematic Model NMPC Horizon Tuning Map: 5 m/s 142
C.3 Kinematic Model NMPC Horizon Tuning Map: 10 m/s 142
C.4 Kinematic Model NMPC Horizon Tuning Map: 15 m/s 143
C.5 Kinematic Model NMPC Horizon Tuning Map: 20 m/s 143
C.6 Bicycle Model NMPC Horizon Tuning Map: 1 m/s 144
C.7 Bicycle Model NMPC Horizon Tuning Map: 5 m/s 144
C.8 Bicycle Model NMPC Horizon Tuning Map: 10 m/s 145
C.9 Bicycle Model NMPC Horizon Tuning Map: 15 m/s 145
C.10 Bicycle Model NMPC Horizon Tuning Map: 20 m/s 145

List of Tables

2.1 MKZ Modeling Properties 23

5.1 2-D Grid of Horizon Tuning Parameters for the Kinematic Model NMPC Implementation 61

5.2 2-D Grid of Horizon Tuning Parameters for the Bicycle Model NMPC Implementation 61

5.3 NMPC Controller Implementations Final Tuning Parameters 75

A.1 MKZ Modeling Properties 121

A.2 MKZ Bicycle Model Bandwidths at Varying Forward Speeds 121

B.1 Sparsity Encoding for the Jacobian of the Constraint Equations 130

B.2 Sparsity Encoding for the Hessian of the Lagrangian 131

C.1 2-D Grid of Horizon Tuning Parameters for the Kinematic Model NMPC Implementation 140

C.2 2-D Grid of Horizon Tuning Parameters for the Bicycle Model NMPC Implementation 140

C.3 Kinematic Model NMPC Optimal Horizon Tuning Parameters 141

C.4 Bicycle Model NMPC Optimal Horizon Tuning Parameters 146

List of Abbreviations

ABS Anti-lock Braking System

ACC Adaptive Cruise Control

ADS Automated Driving System

API Application Programmer Interface

CACC Cooperative Adaptive Cruise Control

CAN Controller Area Network

CC Cruise Control

DARPA Defense Advanced Research Projects Agency

DDT Dynamic Driving Task

DRTK Dynamic base Real Time Kinematic

DSRC Dedicated Short Range Communication

ECEF Earth-Centered, Earth-Fixed

ECU Electronic Control Unit

ENU East-North-Up

ESC Electronic Stability Control

FMVSS Federal Motor Vehicle Safety Standards

GNSS/INS Global Navigation Satellite System/Inertial Navigation System

GPS Global Positioning System

IMU Inertial Measurement Unit

IP Interior-point

KKT Karush-Kuhn-Tucker Conditions

LiDAR Light Detection and Ranging

LQR Linear Quadratic Regulator

LTV Linear Time Varying

MDP Markov Descision Process

MIMO Multiple-input Multiple-output

MPC Model Predictive Control

NHTSA National Highway Traffic Safety Administration

NMPC Nonlinear Model Predictive Control

ODD Operational Design Domain

ODE Open Dynamics Engine

OEDR Object and Event Detection and Response

OO Object-oriented

OSRF The Open Source Robotics Foundation

PID Proportional Integral Derivative

RADAR RAdio Detection And Ranging

RK4 Runge-Kutta 4th Order

ROS Robot Operating System

RRT Rapidly Exploring Random Tree

RTK Real Time Kinematic

SAE Society of Automotive Engineers

SIL Software in the Loop

SQP Sequential Quadratic Programming

TDCP Time Differenced Carrier Phase

Chapter 1

Introduction

1.1 Background and Motivation

Many of the advances in automobile technology have been in the areas of safety and efficiency. Auto manufacturers put thousands of hours into testing safety critical software and hardware to improve the overall safety rating of a new vehicle. The National Highway Traffic Safety Administration (NHTSA) estimates that 613,501 lives were saved between 1960 to 2012 through the use of new safety technologies [1]. The steady increase in lives saved from 1988

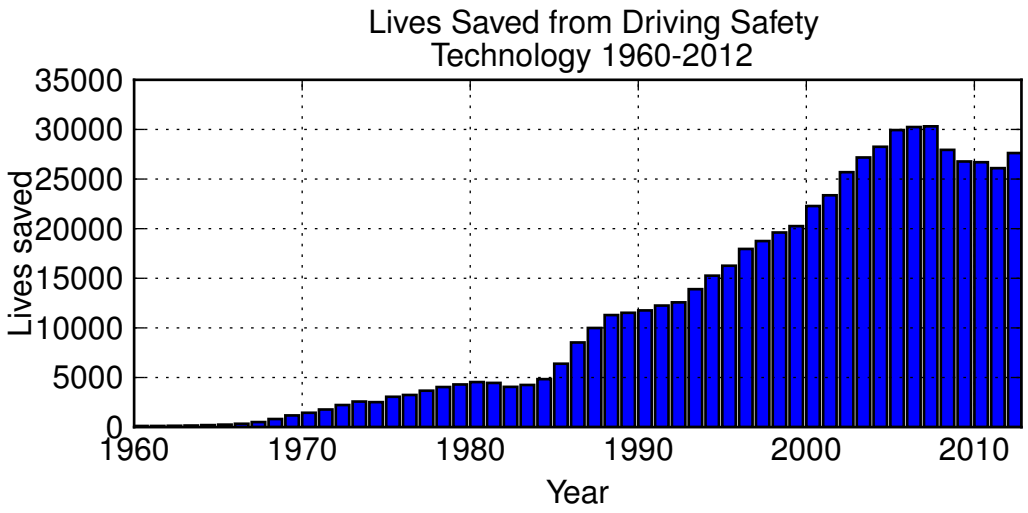


Figure 1.1: Lives Saved from Driving Safety Technology 1960 to 2012

to 2007, seen in Figure 1.1, is attributed to laws and Federal Motor Vehicle Safety Standards (FMVSS) that mandate the use of proven safety features in all new vehicles. Features such as the anti-lock braking system (ABS), electronic stability control (ESC), and traction control are all examples of early vehicle automation systems that contributed to the growth in lives saved. Electronic stability control, for example, reduced the number of single vehicle crashes by 36% and fatal vehicle rollovers by 70% from 1997 to 2004 [2]. Even with all these safety

improvements, NHTSA's National Center for Statistics and Analysis still reported that 37,133 people lost their lives in traffic-related accidents in 2017 [3]. More advanced ground vehicle autonomy is one proposed solution for driving this number to zero.

Some commercial vehicles sold today come with driving automation system features, but true vehicle autonomy, or an *automated driving system* (ADS) as defined by the Society of Automotive Engineers (SAE), is still on the horizon. To effectively communicate about ADS, the SAE Standard J3016 [4] defines six levels of autonomy. An international group of leading automotive manufacturers in the Automated Vehicle Research Consortium concluded that the SAE Standard J3016 aligned most closely with their understanding of ADS. Other automotive standards organizations around the world, such as the International Association of Automotive Associations and the German Automotive Manufacturers Association, are also conforming to the SAE taxonomy [5]. Before classifying the levels of autonomy, SAE defines the Dynamic Driving Task (DDT) and the sub-tasks that must be completed by a human and/or a machine to successfully operate a vehicle. The DDT sub-tasks are listed below.

1. Lateral vehicle motion control
2. Longitudinal vehicle motion control
3. Monitoring the driving environment via object and event detection, recognition, classification, and response preparation with maneuver planning

In sub-tasks 1 and 2 and throughout the rest of this work, lateral control will refer to control of the vehicle motion via the steering wheel and longitudinal control will refer to control of the vehicle through the accelerator and brake. The third sub-task is often referred to as *object and event detection and response* (OEDR). Vehicle autonomy features also may be constrained by the operational design domain (ODD). An ODD defines the set of conditions under which an ADS can be effectively operated. With these definitions, the J3016 levels of autonomy can be summarized as follows:

Level 0: No sustained lateral or longitudinal control from the machine driver. The human driver participates in all sub-tasks of the DDT.

Level 1: Sustained machine driver control of either lateral *or* longitudinal motion, with the machine performing some part of the OEDR sub-task and the human driver performing the rest of the control and monitoring.

Level 2: Sustained machine driver control of both lateral *and* longitudinal motion. The human driver performs continuous supervision of the machine (sharing the OEDR sub-task), ready to retake control at any time.

Level 3: Machine driver performs all of the DDT sub-tasks and alerts the human driver to retake control when a failure occurs or the vehicle exits the ODD.

Level 4: Machine driver performs all of the DDT sub-tasks when in the ODD and can safely control the vehicle to a minimized risk state if a failure occurs or the vehicle exits the ODD.

Level 5: Machine driver can perform all of the DDT sub-tasks in any environment (i.e. no bounds on the ODD).

Previously mentioned active safety systems such as ESC and ABS, as well as driver comfort systems such as conventional cruise control (CC), would be considered level 0 technologies by SAE. However, new level-1 ADS features such as adaptive cruise control (ACC) and lane centering are now becoming available to consumers. Like classic cruise control, an ACC system allows the driver to set a desired speed, but it can also modify speed when it senses other slower-moving vehicles ahead in order to maintain a safe following distance; in other words, the vehicle can sustain longitudinal control on its own. This is an example of level 1 autonomy because while the machine exercises full longitudinal control, there is human/machine cooperation in the OEDR sub-task. Similarly, lane-centering is a level 1 feature, in which the machine driver exercises full lateral control while the human and machine share the OEDR sub-task by monitoring the roadway for hazards and lane boundaries respectively. The combination of these two systems would result in a level 2 driving automation system, in which both longitudinal and lateral control are achieved through the machine driver, and the human driver must be ready to

regain control immediately when a hazard is detected. Several fatal incidents from on-road testing of level 2 technologies [6, 7] raise an important question: are human and machine drivers capable of *continuous cooperation*? In both crashes, the lack of reaction from the driver was a factor. Statistics researchers in Australia evaluated public autonomous vehicle failure data from multiple companies and found a strong correlation between the number of miles driven autonomously and the time required for a human driver to react to a failure situation. Their results indicate that as human drivers become more comfortable with an autonomous system, they are more likely to neglect parts of their full supervisory role [8]. This insight motivates the need for level 3 ADS that completely removes human perception from the control loop.

The technological jump from a level 2 to level 3 ADS requires significant improvements in the machine driver's perception and planning capabilities. This thesis will focus primarily on the *response* aspect of the OEDR sub-task as it relates to local trajectory planning. In particular, it will address the goal of a system that can autonomously handling obstacle avoidance scenarios as part of the continuous lateral and longitudinal control of the vehicle. Because perception subsystems are outside the scope of this work, the algorithms developed will assume the presence of sufficient obstacle and hazard detection.

1.2 Prior Research

Obstacle avoidance in an automated driving system is typically handled in the layers of a motion planning stack. A motion planning stack is a group of algorithms, or layers, each working on a specific sub-problem of the overall motion planning objective. Each layer, from bottom to top, works at an increasing level of abstraction from the actual vehicle. Surveys of motion planning literature generally agree on four layers of the motion planning stack [9, 10]. At the bottom layer, feedback controllers combine the reference path and sensor information to produce commands for vehicle actuators that will execute the motion plan. The reference path comes from the local motion planner (one layer above), that considers the local environment, including any obstacles and closely spaced way-points. Another level up is the behavioral layer, which decides the style of driving (e.g., highway driving vs. parking-lot driving) or switches the ODD based on the given location and high-level navigation objectives. Many examples

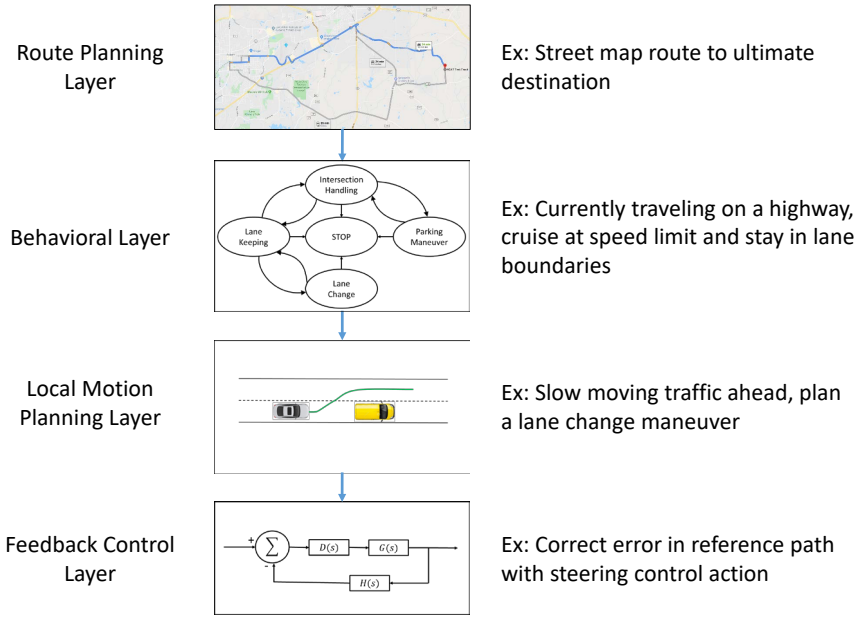


Figure 1.2: Path Planning Stack

of behavior planners are implemented with finite state machines [11, 12]. At the top layer is a route planner that does long distance way-point planning, sometimes at continental scales [13, 14]. This modular architecture, shown in Figure 1.2, allows algorithms with different strengths to work together to accomplish the entire DDT, though it is important to consider how the layers cooperate with each other. This thesis will consider only the local motion planning and feedback control layers and the tightly coupled interface between them.

A wide variety of techniques have been developed over the past few decades to solve the problem of local path planning with both static and dynamic obstacles. This problem is sometimes referred to as the Mover's Problem or the Piano Mover's Problem and is considered P-space hard, meaning the computational complexity in higher dimensional problem spaces increases exponentially to determine the optimal solution within a certain desired accuracy [15]. Graph-based methods, which relied on graph-based optimization algorithms such as Dijkstra's algorithm [16] and A* [17], were the first viable methods to gain popularity in the robotics community [18]. Despite the popularity of these methods, their computational complexity made real-time implementations difficult. To solve this problem, artificial potential fields were

introduced [19]. However, these algorithms suffered from becoming trapped in local minima of the configurations space. This issue was solved with harmonic potential fields [20], but in the meantime much of the community had moved to probabilistic and sampling-based methods because of their simplicity and relative ease of implementation. Sampling-based path planners apply random sampling to the configuration space and use heuristics to drive the sampling towards the optimal path. Probabilistic road maps [21] and rapidly exploring random trees [22] (RRT) present a way to combine the vehicle or robot kinematic constraints with graph-based collision checking and avoid the local minima problem; however initial implementations were too slow for true real-time applications. RRT has emerged as one of the state-of-the-art algorithms for local path planning in obstacle-dense environments because of improvements such as the use of A* and other graph optimization techniques. Some real-time implementations of RRT have been achieved by modifying the iterative replanning behavior of the algorithm [23, 24]. For a full review of incremental sampling-based algorithms, see [25].

Model predictive control (MPC) is another state-of-the-art technique that is being applied to the local path planning problem. MPC takes current feedback information about the state of the system to be controlled and predicts the future state trajectory over a short time window by applying a sequence of suggested control inputs to a dynamic model of the system. The output of the model prediction is optimized to meet some control objective by iteratively adjusting the sequence of control inputs. The first control input in the sequence is then applied to the system and the process is repeated so the controller can react quickly to external disturbances. Model predictive control first emerged in the late 1970s [26] and applications in large industrial processes emerged in the 1980s. Some of these large systems, such as chemical and food processing, took advantage of MPC's ability to handle multiple-input multiple-output (MIMO) systems and apply constraints on the state and control variables directly. MPC requires significantly higher computational burden than a classical control scheme like a proportional integral derivative (PID) controller, but the early industrial applications were for slow moving systems where the extra computation time was insignificant compared to the system bandwidth [27, 28]. Applications of MPC using linear system models with linear and convex constraints are typically more computationally efficient than nonlinear system models with linear/nonlinear and

convex/non-convex constraint sets. However, recent advances in computer hardware and numerical optimization make it possible to do nonlinear model predictive control (NMPC) in fast-moving systems today. An overview of the generalized formulation of NMPC and many of the supporting theorems proving its stability and robustness can be found in [29].

Applications of MPC for vehicle stability and steering control began appearing in the early 2000s. One of the earliest examples is a traction control system that utilized an efficient lookup table of offline-optimized solutions based on a discretized solution space [30]. NMPC was explored for trajectory tracking through active steering control as early as 2005; however the implementation was only tested in simulation and was not yet feasible for a real-time application. Some sub-optimal modifications for a possible online application are discussed in [31]. This work by Borrelli et al. was extended in 2007 with real-time applications of NMPC and linear time varying (LTV) MPC [32]. Other researchers improved the steering control capabilities of linear MPC by formulating the yaw stability objective and dynamic vehicle model constraints as a convex optimization problem, which was easily real-time capable at 100 Hz [33]. This implementation used lateral acceleration bound constraints to model the saturation limits of the tires; however it was later shown that the tire nonlinearity could be abstracted away from the MPC problem with an affine force-input model that preserved convexity for the optimization problem, while the controller still continued to output steering commands based on a more accurate vehicle model [34].

The ability to form constraints on the feasible states of a system through MPC makes it a very popular control strategy for obstacle avoidance. Control designers have incorporated these safety constraints in many different forms to ensure that the controlled trajectory does not include any vehicle state where a collision is possible. Many implementations use the MPC to accomplish the local motion planning task by using a simple vehicle model to generate a collision-free trajectory that can be passed to a lower-level, trajectory-tracking controller that is designed from a much higher fidelity vehicle model. This hierarchical control scheme is shown in Figure 1.3. An early example of this architecture is given in [35], where a simple kinematic vehicle model is used to plan the safe trajectory and the longitudinal and lateral control are individually handled by PID and linear quadratic regulator (LQR) type controllers respectively.

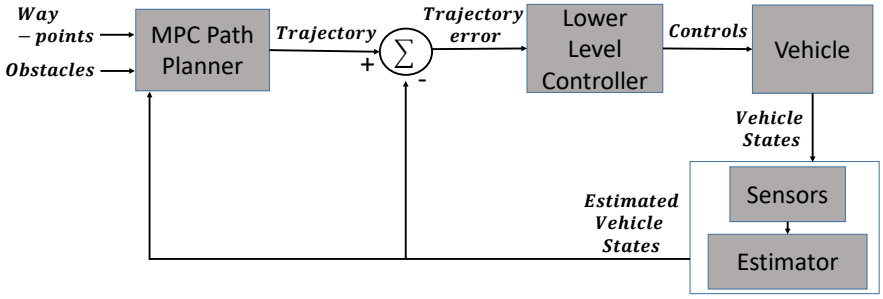


Figure 1.3: MPC Path Planning Hierarchy Control Architecture

In another heirarchical path-planning MPC, Yoon et al. use observations of the parallax effect from obstacles to generate the avoidance constraints, which is shown to be effective for both static and dynamic obstacles [36, 37]. Yet another hierarchical implementation uses a low-fidelity vehicle model for trajectory generation and a robust invariant set technique for accurately tracking the reference trajectory under disturbances (including model error from the trajectory generation). This work by Gao et al. uses a distance-based constraint for collision avoidance, and represents the obstacles as ellipses in free space [38]. The trajectory generation MPC, even using low order models, can still be computationally inefficient. Bevan et al. were some of the first researchers to apply convex optimization to the obstacle avoidance case, although only bound constraints on the longitudinal and lateral directions of the vehicle could be applied and their approach required three optimization passes [39]. These hierarchical schemes provide partial coupling between the trajectory generation and feedback control layers of the motion planning stack; other forms of MPC can be used to effectively combine the two layers.

When MPC is formulated such that it completes the driving objective and directly outputs the control signals without an intermediate trajectory generation it can be thought of as comprising both the local motion planning and feedback control layers. An example of this architecture is given in Figure 1.4, where the control signals are the steering, throttle, and brakes and the driving objective is to take the vehicle through some set of way-points while avoiding obstacles. The convex optimization approach to vehicle stability control, found in [34], was extended in [40] and [41] to include obstacle avoidance constraints. These real-time capable

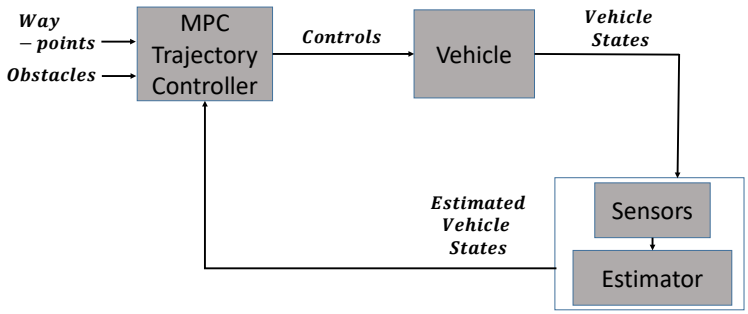


Figure 1.4: MPC Trajectory Control Architecture

implementations demonstrate a trade-off between avoidance and stability at the vehicle's handling limits, which can be exploited for use in highly dynamic situations. Other researchers have used non-gradient-based optimization schemes such as pattern search optimization [42] and particle swarm optimization [43] with soft constraints on both avoidance and target seeking. While these methods are often intuitive, the computational complexity quickly exceeds that of an established gradient-based method. Some recent work [44, 45] evaluates MPC control structures that switch between a target tracking and obstacle avoidance mode. Proportional navigation was used in cooperation with a NMPC in avoidance mode of one implementation and could handle both static and dynamic obstacles. Another more recent implementation ran two efficient MPC controllers concurrently and chose the best control solution based on the desired trade-off of path tracking and obstacle avoidance [46]. These MPC control schemes lay the ground work for the implementation of a level 3 capable automated driving system.

One of the primary applications pushing innovation in automated driving is platooning, or autonomous following, of heavy duty vehicles. Platooning has generated a lot of interest for military applications, specifically with the U.S. Army's leader- follower program [47]. The commercial sector also has significant interest in this technology because of the possible fuel saving benefits associated with the close following distances that cooperative adaptive cruise control (CACC) vehicles can achieve [48, 49]. Many of the research and commercial systems currently available are capable of level 2 autonomy, while only a few systems claim level 3 capabilities [50]. MPC-based obstacle avoidance has been proposed in a number of works, but all of them have focused on testing only in simulation [44, 51, 52]. In this thesis, some of the

technologies that are currently used in a level 2 platooning system will be incorporated into an MPC controller that is capable of both following and obstacle avoidance, in progress towards a level 3 automated driving system.

1.3 Contributions

Nonlinear model predictive control is not a new solution to obstacle avoidance control. Similarly, there exists applications of level-3 and level-4 vehicle platooning. In this work, an application of NMPC for obstacle avoidance and path-following is tailored to an existing platooning software framework as an extension to its current level-2 capability. Specifically, the work presented in this thesis makes the following contributions:

- A NMPC library for quickly developing real-time control applications
- An application of NMPC for obstacle avoidance and trajectory tracking in platooning and non-platooning scenarios
- A NMPC controller tuning procedure and path tracking performance analysis
- Software-in-the-loop and vehicle testing of the proposed NMPC algorithm

1.4 Thesis Outline

This thesis has five remaining chapters. Chapter 2 details the vehicle modeling used in this thesis. It begins with simple kinematic vehicle modeling, advances to linear and nonlinear dynamic vehicle modeling, and ends with a discussion of the simulation modeling used for preliminary testing of control algorithms. Chapter 3 discusses the nonlinear model predictive control algorithms developed for this thesis, including reference inputs, model discretization, and the formulation of the cost function and constraints. Chapter 4 gives an overview of Auburn University's existing software stack for level-2 path following in automated truck platoons and discusses how an NMPC software module fits into this stack. Chapter 4 also presents a brief overview of obstacle detection and tracking methods found in the literature, and how software

modules for these algorithms can be incorperated into the system or simulated to test the controller in an obstacle avoidance scenario. Chapter 5 presents simulated and experimental test results for the proposed NMPC controller. Chapter 6 concludes with ideas for future improvement of the NMPC algorithm's performance.

Chapter 2

Ground Vehicle Modeling and Simulation

Mathematical models of a vehicle's motion are essential to the design and simulation of a model predictive controller. These models come in varying degrees of complexity. There is an important trade-off between model complexity and computational efficiency that must be made in both the design and testing of an MPC. MPC designers typically use simpler models that can make the optimization routine computationally efficient and still describe the driving objective. On the other hand, simulation testing of an MPC design typically sacrifices computational efficiency to more accurately model the vehicle's dynamic response to both control inputs and disturbances. This chapter will introduce the vehicle modeling necessary for the proposed NMPC controller, starting with the common coordinate frames used to develop all of the vehicle models. Simple kinematic vehicle models will be presented, followed by a common low-order dynamic model. Finally, the advanced simulation modeling used as a software in the loop (SIL) testing framework will be discussed.

2.1 Vehicle Coordinate Frames

Many different coordinate systems are used to describe vehicle motion. All axis-systems in a particular coordinate system are considered right-handed such that two of the three axes, $\hat{i}$ and $\hat{j}$ in an arbitrary frame, are perpendicular and form a plane while the third axis, $\hat{k}$, is orthogonal to that plane.

$$\hat{k} = \hat{i} \times \hat{j} \tag{2.1}$$

The vehicle models used in this thesis are developed using two primary coordinate systems: a vehicle-fixed system and an Earth-fixed system.

The vehicle-fixed coordinate system, $\{x_v, y_v, z_v\}$, is a moving reference frame that is attached to the vehicle as shown in Figure 2.1. The origin of this system is commonly placed at

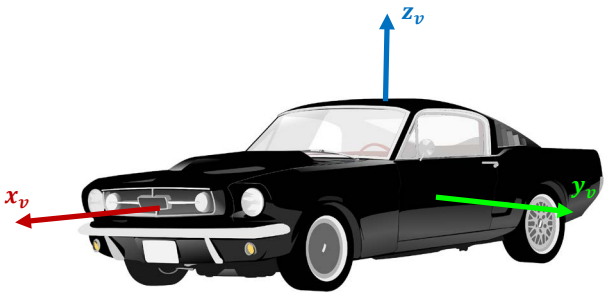


Figure 2.1: Vehicle-fixed Coordinate System
Figure adapted from [53]

the vehicle's center of gravity (CG) or at the center of the front or rear axle. In this work the vehicle-fixed origin is assumed to be at the vehicle CG unless otherwise specified. The axes in Figure 2.1 are defined in accordance with the z-up system defined in SAE Standard J670 [54], which defines x_v pointing out the front of the vehicle (the longitudinal axis), y_v pointing out the left side of the vehicle (the lateral axis), and z_v pointing out the roof of the vehicle. This convention is used in place of the traditional z-down system that is typically used in aeronautical navigation and control applications, because it aligns with the convention adopted by the mobile robotics community [55]. The vehicle kinematics are developed in the vehicle-fixed frame and the vehicle's linear/angular velocities and accelerations are most often expressed in this frame. Vehicle position information is often expressed in the vehicle-fixed frame for the purposes of low-level throttle, steering and braking control. Position vectors in the vehicle-fixed frame are relative to a position in the earth-fixed frame, which is often used as the common frame to describe high-level control objectives.

An inertial reference frame is necessary for defining relative velocities and accelerations in a vehicle's kinematic modeling. The WGS84 XYZ coordinate system, an earth-centered, earth-fixed (ECEF) system, is the *de facto* standard inertial frame for navigation systems that rely on the Global Positioning System (GPS) [56]. WGS84 XYZ coordinates are very useful for describing motion over very long ranges; however, ground vehicle motion is more intuitively described in an intermediate fixed frame that is defined with a reference position on the surface of the Earth. The east-north-up (ENU) convention, depicted in Figure 2.2, is standard in the

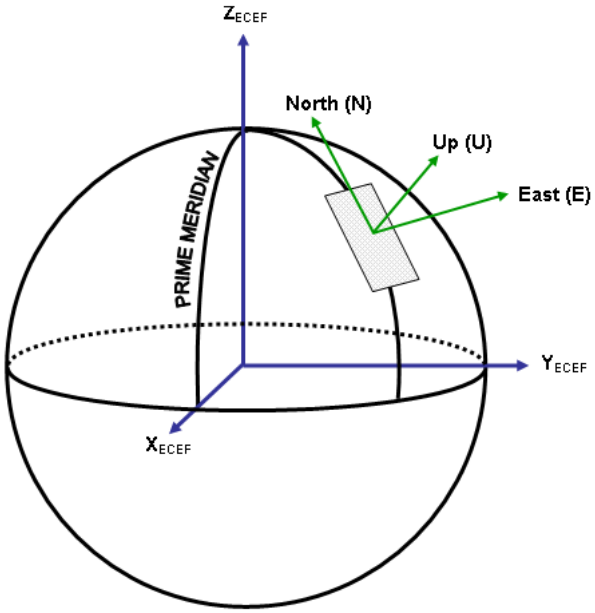


Figure 2.2: Earth-fixed Coordinate Systems[57]

mobile robotics community [58]. This convention is used because a zero translation and rotation offset aligns exactly with the vehicle-fixed frame defined in Figure 2.1 if the vehicle origin is at the ENU origin. Although the ENU frame is not technically an inertial reference frame, the rotation of the Earth about its own axis is neglected in typical vehicle dynamics modeling. With the Earth-fixed and vehicle-fixed coordinate systems defined, the vehicle kinematics can be formulated.

2.2 Kinematic Modeling

Ground vehicles are complex multi-body systems that can be represented with high degree-of-freedom dynamic models, but many control designers use simplified low order models that can still represent the vehicle's kinematic constraints and dynamic response to a certain level of accuracy. For lateral control of the vehicle, it is common to assume planar motion. The simplest planar motion models also do not consider weight transfer of the vehicle thus neglecting the pitch and roll dynamics. Figure 2.3 shows a typical 4-wheeled (non-articulated) vehicle traveling in the east–north plane at a velocity $\vec{v}$. The vehicle's yaw angle, or heading, is the angle ψ between the positive east axis and the longitudinal axis of the vehicle. The yaw-rate is the rotation rate around the vehicle's vertical axis and the time derivative of the vehicle yaw

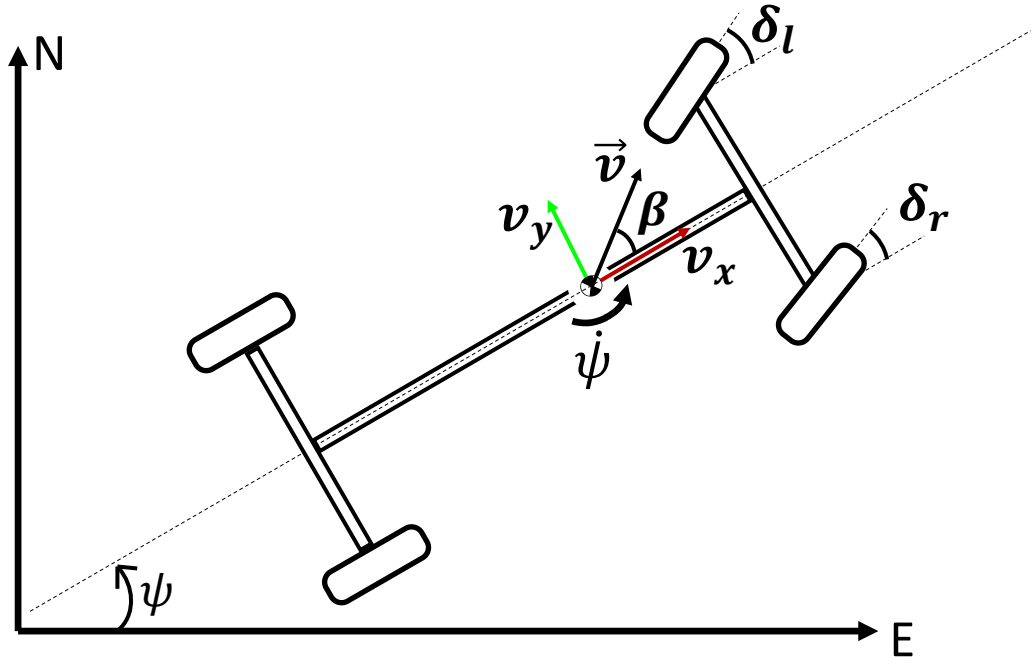


Figure 2.3: 4 Wheel Vehicle in Planar Motion

angle. The angle β, known as the side-slip angle, is described in terms of the vehicle-fixed frame velocity components.

$$\beta = \arctan(\frac{v_y}{v_x}) \tag{2.2}$$

The vehicle's direction of travel in the east–north plane is defined by the course angle, given in Equation (2.3).

$$\nu = \psi + \beta \tag{2.3}$$

The vehicle in Figure 2.3 is considered front-steering with the left tire steer angle δ_l and right tire steer angle δ_r.

The vehicle's position dynamics can be described with a simple rotation of the vehicle-fixed velocity into the earth-fixed frame. The component velocity form is shown below in Equations (2.4–2.5).

$$\dot{E} = v_x \cos(\psi) - v_y \sin(\psi) \tag{2.4}$$

$$\dot{N} = v_x \sin(\psi) + v_y \cos(\psi) \tag{2.5}$$

The equivalent vehicle course form is shown below in Equations (2.6–2.7).

$$\dot{E} = |\vec{v}| \cos(\nu) \tag{2.6}$$

$$\dot{N} = |\vec{v}| \sin(\nu) \tag{2.7}$$

If the vehicle velocity and yaw-rate are independently controllable, the planar motion model given in Equation (2.8) may be suitable.

$$\frac{d}{dt} \begin{pmatrix} E \\ N \\ \psi \end{pmatrix} = \begin{pmatrix} |\vec{v}| \cos(\nu) \\ |\vec{v}| \sin(\nu) \\ \dot{\psi} \end{pmatrix} \tag{2.8}$$

This model is common for high-level motion control of differential-drive robots; however, it allows for a non-zero yaw-rate command even when $|\vec{v}| = 0$, which does not accurately represent the kinematic constraints of a front-steering vehicle.

The diagram in Figure 2.3 is commonly simplified into a single track model, also known as the bicycle model, for developing a planar dynamic vehicle model. Through *collapsing* the width of the vehicle, the steering is simplified to a single input, δ. The lateral velocities at the front and rear axles remain unaffected by this assumption and the imaginary *center tire* retains Ackermann steering geometry if it is the cotangent average of the true left and right tire steer angles [59]. Figure 2.4 shows the bicycle model in a steady-state turn around a circle of constant radius, R. The lengths a and b in Figure 2.4 are the distance from the front axle to the CG and the distance from the CG to the rear axle respectively. These distances sum to the total wheel base length of the vehicle.

$$L = a + b \tag{2.9}$$

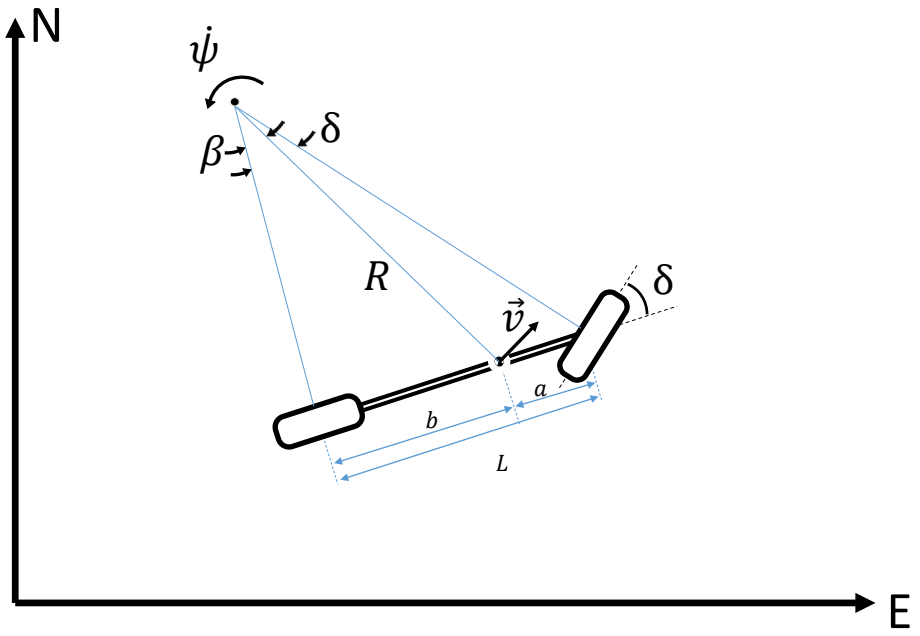


Figure 2.4: Bicycle Model in Steady-state Turning

Analysis of this steady-state turning scenario results in a modified version of the model in Equation (2.8), which is often referred to as the kinematic bicycle model. Note that at steady-state, the yaw-rate is a function of the velocity and the turning radius.

$$\dot{\psi} = \frac{|\vec{v}|}{R} \tag{2.10}$$

The vehicle side-slip angle is also related to the turning radius.

$$\sin(\beta) = \frac{b}{R} \tag{2.11}$$

The turning radius can be eliminated from Equation (2.10) by substituting in Equation (2.11).

$$\dot{\psi} = \frac{|\vec{v}|\sin(\beta)}{b} \tag{2.12}$$

Through the turning geometry, the side-slip angle can be put in terms of the steering angle input.

$$\beta = \arctan\left(\frac{b\tan(\delta)}{L}\right) \tag{2.13}$$

Equation (2.13) can be substituted into Equation (2.12) resulting in a relationship between the steady-state yaw-rate output and the steering input that is proportional to the magnitude of the vehicle's velocity.

$$\dot{\psi} = \frac{|\vec{v}| \sin\left(\arctan\left(\frac{b\tan(\delta)}{L}\right)\right)}{b} \tag{2.14}$$

If the steering angle is assumed to remain within the small angle approximation, which is a reasonable assumption for highway driving scenarios, then the nonlinear trigonometric functions can be removed from Equation (2.14).

$$\dot{\psi} = \frac{|\vec{v}|}{L}\delta \tag{2.15}$$

The full planar kinematic bicycle model is given in Equation (2.16).

$$\frac{d}{dt}\begin{pmatrix} E \\ N \\ \psi \end{pmatrix} = \begin{pmatrix} |\vec{v}|\cos(\nu) \\ |\vec{v}|\sin(\nu) \\ \frac{|\vec{v}|\sin\left(\arctan\left(\frac{b\tan(\delta)}{L}\right)\right)}{b} \end{pmatrix} \tag{2.16}$$

The steady-state model given Equation (2.16) breaks down under steering transients and can be inaccurate even at steady-state for high velocities if the vehicle has significant understeer. For an evaluation of these discrepancies, see Appendix A. In order to develop equations of motion for the bicycle model, equations for the tire side-slip angles and acceleration at the CG are needed. The tire side-slip angles, α_f and α_r for the front and rear respectively, are shown in Figure 2.5. The slip angle at the tire is simply the angle formed by the velocity components in the body frame, less any steer angle. The front and rear tire slip angles are given in Equation (2.17) and Equation (2.18), respectively.

$$\alpha_f = \arctan\left(\frac{v_y + a\dot{\psi}}{v_x}\right) - \delta \tag{2.17}$$

$$\alpha_r = \arctan\left(\frac{v_y - b\dot{\psi}}{v_x}\right) \tag{2.18}$$

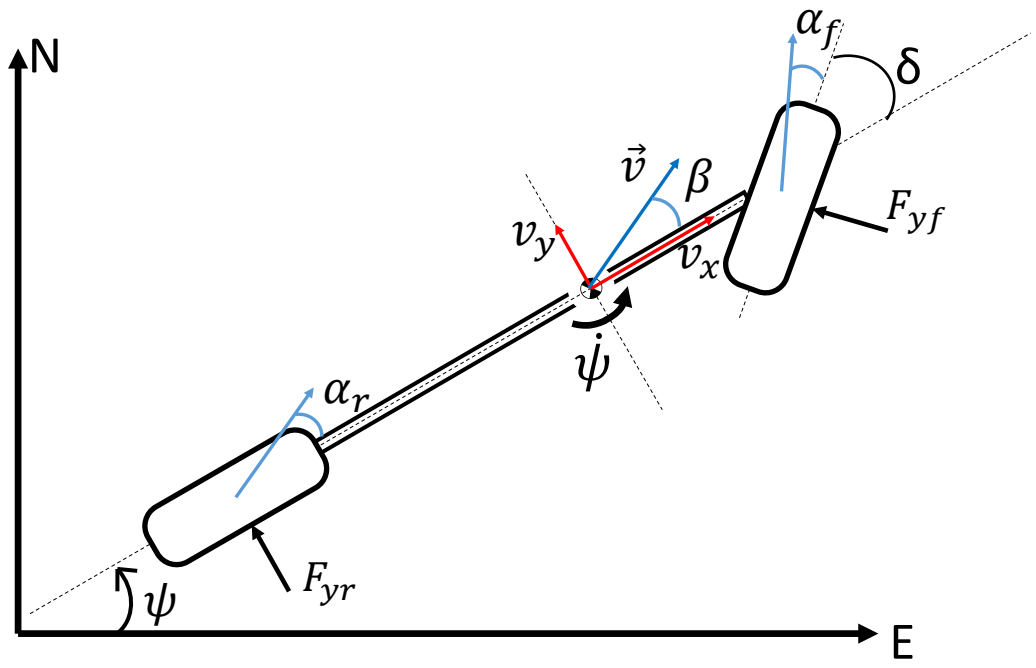


Figure 2.5: Bicycle Model Free-body Diagram

The operating range of tire slip-angles is generally within the small angle approximation, therefore Equation (2.17) and Equation (2.18) can be simplified to the following approximations.

$$\alpha_f = \left(\frac{v_y + a\dot{\psi}}{v_x}\right) - \delta \tag{2.19}$$

$$\alpha_r = \left(\frac{v_y - b\dot{\psi}}{v_x}\right) \tag{2.20}$$

The acceleration at the CG is found by differentiating the velocity vector at the CG.

$$\vec{a} = \dot{\vec{v}} + \vec{v} \times \vec{\omega} \tag{2.21}$$

The general rotation-rate vector is $\vec{\omega} = \dot{\psi}$, assuming no rolling or pitching. The resulting acceleration components in the vehicle-fixed frame are given in Equations (2.22–2.24).

$$a_x = \dot{v_x} + v_y\dot{\psi} \tag{2.22}$$

$$a_y = \dot{v_y} - v_x\dot{\psi} \tag{2.23}$$

$$a_z = 0 \tag{2.24}$$

2.3 Yaw Dynamic Modeling

The planar yaw dynamic bicycle model is the basis for linear analysis and control of the vehicle's lateral transients. Given the free-body diagram in Figure 2.5 and assuming no longitudinal tire forces, the sum of forces in the vehicle's lateral direction is given in Equation (2.25) and the sum of moments about the vehicle-fixed z-axis is given in Equation (2.26).

$$m(\dot{v_y} - v_x\dot{\psi}) = F_{yf}\cos(\delta) + F_{yr} \tag{2.25}$$

$$I_{zz}\ddot{\psi} = aF_{yf}\cos(\delta) - bF_{yr} \tag{2.26}$$

The vehicle's total mass is denoted by m in Equation (2.25) and I_{zz} is the principle mass moment of inertia about the vehicle-fixed z-axis in Equation (2.26). The lateral tire forces are a nonlinear function of tire slip-angles. There are many complex nonlinear tire models such as Pacejka's magic tire model that capture the peak tire force and saturation effect at the limits of handling [60]. An example tire force curve for various different tire loading conditions (and assuming no tire-inclination or longitudinal slip) is given in Figure 2.6. In the typical operating

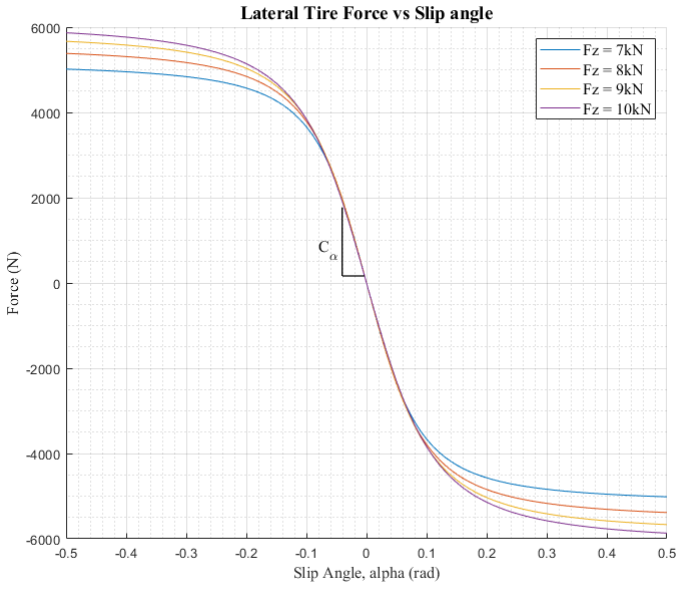


Figure 2.6: Tire Force Curve for Pacejka's Magic Tire Model

range of tire slip-angles the tire force relationship can be approximated as linear.

$$F_{yf} = -C_{\alpha_f}\alpha_f \tag{2.27}$$

$$F_{yr} = -C_{\alpha_r}\alpha_r \tag{2.28}$$

In Equations (2.27–2.28) the C_α term is known as the tire stiffness coefficient. This coefficient is the slope in the linear region of the tire force curves in Figure 2.6. The linear equations of motion are expressed in Equations (2.29–2.30) by combining Equations (2.27–2.28) with Equations (2.17–2.18) and substituting the result into Equations (2.25–2.26).

$$\dot{v}_y = \frac{-\left(C_{\alpha_f} + C_{\alpha_r}\right)}{mv_x}v_y + \frac{-\left(aC_{\alpha_f} - bC_{\alpha_r}\right) - mv_x^2}{mv_x}\dot{\psi} + \frac{C_{\alpha_f}}{m}\delta \tag{2.29}$$

$$\ddot{\psi} = \frac{-\left(aC_{\alpha_f} - bC_{\alpha_r}\right)}{I_{zz}v_x}v_y + \frac{-\left(a^2C_{\alpha_f} + b^2C_{\alpha_r}\right)}{I_{zz}v_x}\dot{\psi} + \frac{aC_{\alpha_f}}{I_{zz}}\delta \tag{2.30}$$

Note that this model is only linear and time-invariant if the small steering angle approximation is valid ($\cos(\delta) \approx 1$) and the vehicle is not accelerating longitudinally ($F_x = 0$ and v_x is constant). The dynamic bicycle model has significantly more parameters to identify than the kinematic bicycle model. The tire stiffness coefficients can be particularly hard to estimate; however, when correct model parameters are used, the dynamic bicycle model is much more accurate than the kinematic bicycle model. More information on the system identification methods used in this thesis and a comparison of the kinematic and dynamic bicycle models are given in Appendix A.

2.4 Simulator Modeling

Over the past decade, a large number of simulators have been developed to provide robotics engineers a SIL interface to accelerate development and testing of real-time software components. The power of SIL development is the ability to transfer the exact software component being developed in a safe, virtual sandbox to the target platform for field testing. In this work,

the Gazebo simulator [61] from the Open Source Robotics Foundation (OSRF) is the SIL platform of choice. OSRF also maintains the Robot Operating System (ROS) libraries [62] and provides tight ROS integration for Gazebo users. If a simulated Gazebo vehicle provides the same ROS interface as the real vehicle then any component that adheres to that interface is agnostic to its test platform. Gazebo provides realistic multi-body physics simulation by using the Open Dynamics Engine (ODE) [63]. Robots of any form-factor can be composed from generic ODE link and joint types, but Gazebo also gives users the option to customize almost any aspect of a simulation component through a piece of plugin code that is dynamically loaded into the simulation at run-time. All of these features are utilized in SIL testing of the algorithms developed in this thesis, but it is important to understand the simulation model structure and its limitations.

Gazebo gives users the ability to create links in a model from primitive shapes, like boxes, cylinders, and spheres, or more complex shapes from custom tessellated mesh files. Each link in a model must define its inertial properties and can optionally define visual and collision properties. Model links are chained together with joints that limit the model's degrees of freedom and provide a way to control and measure the state of the links they are attached to. Gazebo does not require that a model have a single kinematic chain of links or restrict the user from defining closed kinematic chains (i.e., a child link may have multiple parents). Vehicle model definitions typically contain a single tree-link kinematic chain that is a simplification of a real vehicle's chassis.

The primary vehicle platform used in this thesis is a 2017 Lincoln MKZ with full drive-by-wire capability. The real MKZ and its Gazebo companion, shown in Figure 2.7, provide the same ROS application programming interface (API) for interacting with the drive-by-wire kit. The Gazebo MKZ model is implemented with the minimum set of components that result in a 4-wheel Ackermann-steering vehicle that accurately represents the vehicle's geometry and gross weight distribution. Important model parameters for both the Gazebo MKZ model and MKZ test vehicle are summarized in Table 2.1. The simulated model's simple structure contains 12 degrees of freedom. Most of the vehicle's mass and inertia terms are represented by a single box style link, the chassis link. All the joints are a revolute type joints that only allows rotation

Figure 2.7: Real Drive-by-wire Lincoln MKZ (left) and Simulated Drive-by-wire Lincoln MKZ (right)

Table 2.1: MKZ Modeling Properties

property	Gazebo MKZ	Real MKZ
$a(m)$	1.428	1.257
$b(m)$	1.423	1.593
$L(m)$	2.850	2.850
track width (m)	1.594	1.594
mass (Kg)	1542	1857
$I_{zz}(Kg \cdot m^2)$	1000	4292
$C_{\alpha_f}(N/rad)$	31240	120000
$C_{\alpha_r}(N/rad)$	31240	184600

about a single axis. The joints between the chassis and the steering links provide realistic limits on total steer angle and a method for steering control. All four wheels are connected through revolute joints that allow continuous rotation. The default kinematic chain for the Gazebo MKZ is shown in Figure 2.8. Note that there are no suspension elements in this kinematic chain that would allow for significant pitching or rolling of the vehicle body frame relative to the ground plane; in other words Gazebo restricts the pitch and roll of this model. All joints in Gazebo do have some compliance, as the physics engine will slightly relax constraints to produce a numerically stable simulation of the motion of the overall system, but this compliance is not easily tuned.

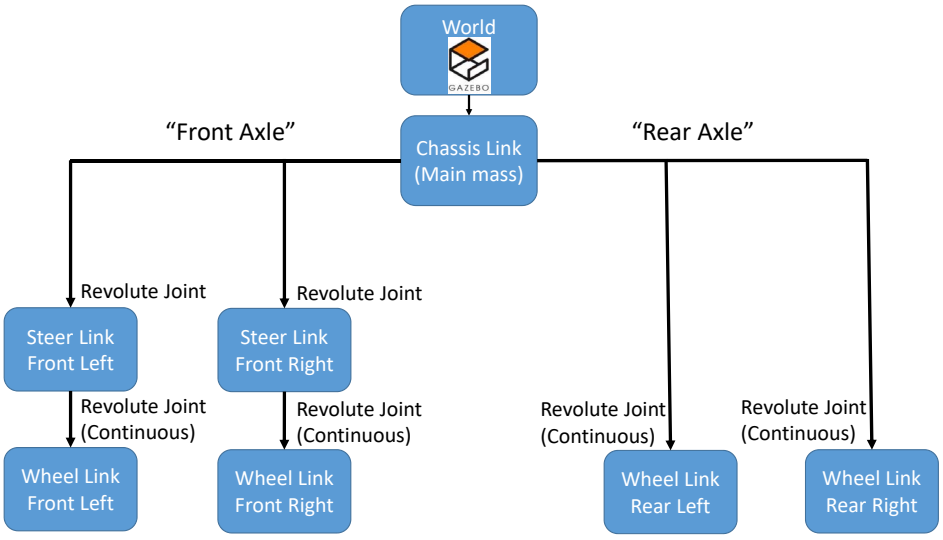


Figure 2.8: Gazebo MKZ Model Kinematic Chain

If modeling the vehicle's suspension is critical to the simulation performance of the application under test, this limitation must be addressed through an adjustment to the kinematic chain shown in Figure 2.8. An intermediate suspension link must be added between the chassis and each wheel or steering link/wheel subsystem that is connected through a prismatic joint. This prismatic joint constrains the suspension link's motion to move linearly along a single axis. The joint's dynamics can be modified to emulate the proper suspension spring and damping rates. The 16 DOF modified kinematic chain with the additional suspension elements is shown in Figure 2.9. This modified suspension model allows the vehicle to realistically transfer weight over individual tires during cornering and braking scenarios. For advanced tire

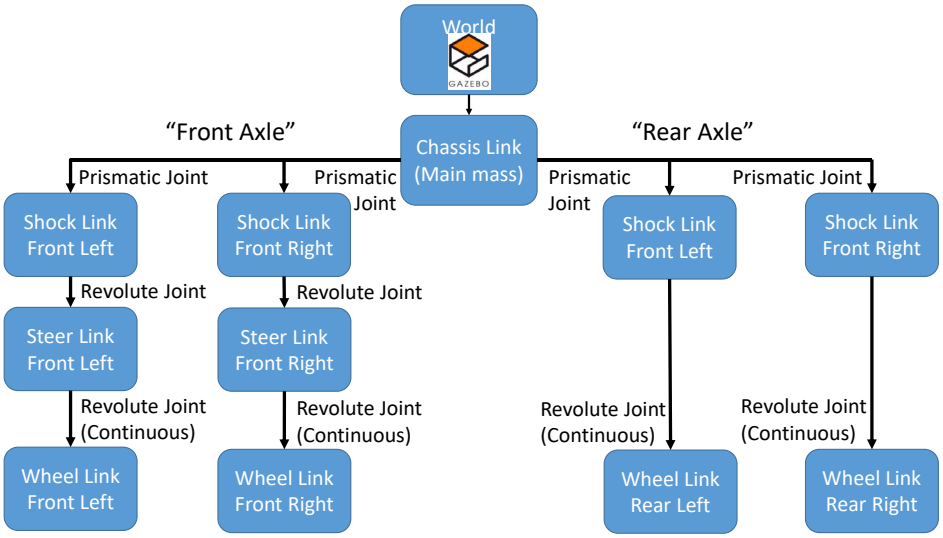


Figure 2.9: Gazebo MKZ Model Kinematic Chain with Suspension Elements

models such as the Pacejka tire model shown in Figure 2.6 the change in normal force on a tire (induced by road changes or weight transfer) is a direct input to the model that affects the maximum amount of force the tire is capable of generating. Gazebo does not provide any explicit tire models by default, but the force generated from the wheel/ground collision works on a similar principle [63].

Ultimately, the simpler simulation model structure, given in Figure 2.8, was chosen for simulation testing of the proposed MPC algorithms because it provides model fidelity that is adequate to verify the software implementations. SIL frameworks other than Gazebo should be evaluated in the future if higher simulation fidelity is desired. Some prior research has shown that Gazebo can adequately simulate a vehicle's yaw dynamics, but other simulators that were developed specifically for ground vehicles can provide better model fidelity [64]. The two models presented in this chapter are even further simplifications of the multi-body physics model that Gazebo provides however, so testing and comparing the MPC implementations using these simple planar yaw models in Gazebo and on the real vehicle provides an indication of the controllers' robustness to model inaccuracies.

Chapter 3

Model Predictive Control

Model predictive control is a growing sub-field of optimal control that has many different implementations. This chapter will begin by outlining a general MPC algorithm and introducing the nomenclature used to describe the elements of MPC developed for this thesis. The following sections will develop the control objective and constraints that are specific to the path following and obstacle avoidance problem domain. The chapter will conclude with some specifics on the software used to realize the proposed MPC implementations.

3.1 The Receding Horizon Control Principle

Receding horizon control is sometimes used in the literature as an alternate name for MPC. The receding horizon refers to the fact that the control objective is not optimized with an infinite time horizon like the analytical solution of LQR for linear state-feedback control, but rather on a finite time horizon, H. This type of control makes use of a dynamic model of the system to simulate the system behavior on the receding horizon as the value of the control input is optimized. Equation (3.1) refers to a generic dynamic model that is a function of the system state $x \in \mathbb{R}^n$, control input $u \in \mathbb{R}^m$, and time t.

$$\dot{x} = f(t, x, u) \tag{3.1}$$

Generally, the model is simulated on a discrete time step, T, with a number of simulation steps, N. Equation (3.2) defines the time horizon, H, given a number of simulation steps that are uniformly spaced in time, and accounts for the current state, $x(t)$.

$$H = T(N - 1) \tag{3.2}$$

The optimization routine iteratively seeks the optimal set of control inputs, $\mathbf{u}^*(\mathbf{t}) = (u^*(t), u^*(t+T), \ldots, u^*(t+H-T))$, such that the simulated state trajectory, $\mathbf{x} = (x(t), x(t+T), \ldots, x(t+H))$, results in the optimal trajectory $\mathbf{x}^*(\mathbf{t})$. The set of optimized control inputs, $\mathbf{u}^*(\mathbf{t})$, is an approximate *value* of the control law at the current state. Because MPC does not seek to find the *definition* of the control law for all states, the control must be re-planned as new observations of the states become available. No model is perfect and the actual system will not respond exactly as the forward simulation predicts at each iteration of the MPC controller. To minimize the effect of model error and take into account any new measurements of the state, only the first control value of the optimized set of inputs, $u^*(t)$, is actually applied to the system on each iteration of the MPC controller. This is the crux of the receding horizon control principle. The general MPC algorithm, following the receding horizon control principle, is summarized below in Algorithm 1. Although the MPC algorithm is naturally expressed in a discrete manner, before presenting it, the next sections will first set up the optimization problem using the generic continuous model given in Equation (3.1). The model discretization is an important step in transforming the MPC expression of the optimal control problem into the generic form that an optimization routine can handle. An explanation of the discretization method used in this thesis will precede the discussion of the proposed implementations for automated vehicle path following and obstacle avoidance.

Algorithm 1: Basic MPC Algorithm

Choose number of prediction steps, N;
Choose model time-step, T;
Discretize model with state, x, and input, u, by time-step, T, over N steps;
while *control running* **do**
  Get the current state, $x(t)$;
  find the optimal set of inputs, $\mathbf{u}^* = (u^*(t), ..., u^*(t+H-T))$, that gives the optimal state trajectory, $\mathbf{x}^*$;
  apply first control sample, $u^*(t_k)$;
end

3.2 Optimization Problem Setup

As previously mentioned, MPC is in essence solving a numerical optimization problem. All optimization problems seek to minimize or maximize some *objective* function. An unconstrained optimization problem with the objective function $\phi(z)$ to be minimized is given in Equation (3.3), where z is the variable to be optimized, otherwise known as the decision variable.

$$\min_{z} \quad \phi(z) \tag{3.3}$$

Choosing the objective function to appropriately model the desired outcome is very important in order to receive a feasible solution from the optimization routine. Many optimization problems start with a cost function that is convex, because convex optimization problems are in general much easier to solve than non-convex problems. Convexity is also a desirable property because any local minimum solution to the optimization problem is also the global minimum solution, which is not guaranteed to be true of a local solution to a non-convex problem. For the definition of convexity and a full review of convex optimization methods, see the text from Boyd and Vandenberghe [65].

An MPC regulator may be set up with exactly the same convex and quadratic objective function as the LQR formulation. This objective function, sometimes referred to as a cost function because finding the minimum cost is desirable, is given in Equation (3.4).

$$\phi(x,u) = \int_{t=0}^{t=\infty} \left(x^T(t)Qx(t) + u^T(t)Ru(t)\right) dt \tag{3.4}$$

In Equation (3.4), the positive semi-definite matrix Q penalizes the distances of the states from the origin, and the positive semi-definite matrix R penalizes the control energy applied to the system. For an MPC, this form of the cost function must be adjusted from the infinite time interval $[0, \infty)$ to the finite time interval $[0, H]$. This modification, including a terminal cost, or the *cost-to-go* that approximates the remaining cost at infinite time, is given in Equation (3.5).

$$\phi(x,u) = \int_{t=0}^{t=H} \left(x^T(t)Qx(t) + u^T(t)Ru(t)\right) dt + V_f(x(H)) \tag{3.5}$$

In this form of the cost function, the decision variable is some combination of the continuous variables u and x, leading to an infinite dimension problem that is not feasible to solve with numerical optimization. Equation (3.6) converts the continuous time addition of cost to the discrete summation of individual stage costs over the prediction horizon.

$$\phi(x,u) \approx \sum_{k=0}^{k=N-1} T\left[x^T(k)Qx(k) + u^T(k)Ru(k)\right] + V_f(x(N)) \tag{3.6}$$

The discrete time index k is given in Equation (3.7) and hereafter will be used for discrete formulations in place of the continuous time variable t.

$$t = 0, T, ..., H, \qquad \forall k = 0, 1, ..., N \tag{3.7}$$

Also note that all the models used for this thesis are time invariant and the time $t = 0$ describes the current time and the start of the prediction horizon as it is given in Equation (3.7).

The cost function given in Equation (3.6) covers the MPC regulator case. In the MPC reference tracking case, this equation is modified to penalize the squared error from the desired state trajectory, x_{des}, as shown in Equation (3.8).

$$\phi(x,u) \approx \sum_{k=0}^{k=N-1} T\left[(x(k) - x_{des}(k))^T Q(x(k) - x_{des}(k)) + u^T(k)Ru(k)\right] + V_f(x(N)) \tag{3.8}$$

This reference tracking form of the cost function is the method used in this thesis to accomplish way-point following control. The reference trajectory x_{des} can be made a single target way-point if it is the only feasible point along the prediction horizon. If there are multiple feasible way-points within the prediction horizon, they may be included in a changing set of desired states, $\mathbf{x_{des}} = (x_{des}(0), ..., x_{des}(N))$. Even with this selection of the cost or objective function, the model prediction must be included as a part of the optimization routine.

One of the great features of numerical optimization is the ability to include constraints on the feasible solution space. MPC takes advantage of this feature by imposing the model dynamics as a set of constraints on the optimal control problem. Combining the continuous

objective function in Equation (3.5) with an initial condition constraint and the prediction constraint using the model in Equation (3.1) results in the continuous optimal control problem given in Equation (3.9).

$$\begin{aligned} \min_{x,u} \quad & \int_{t=0}^{t=H} \left(x^T(t)Qx(t) + u^T(t)Ru(t)\right) dt + V_f(x(H)) \\ & s.t. \\ & x(t) = x(t), \\ & \dot{x} = f(t, x, u). \end{aligned} \tag{3.9}$$

The initial condition does not immediately appear useful, but is necessary in order to include state feedback information in the MPC. The left-hand side $x(t)$ is the representation of the model's initial condition in terms of the decision variable, while the right-hand side $x(t)$ is the actual value obtained from measuring and/or estimating the system state. Similarly, the implementation of the model constraint equations is not immediately apparent in their continuous form. The first step toward a practical application of these prediction constraints is to reformulate Equation (3.9) into its discrete form, given in Equation (3.10).

$$\begin{aligned} \min_{x,u} \quad & \sum_{k=0}^{k=N-1} T\left[x^T(k)Qx(k) + u^T(k)Ru(k)\right] + V_f(x(N)) \\ & s.t. \\ & x(0) = x(0), \\ & x(k+1) = F(t(k), x(k), u(k)) \qquad \forall k = 0, 1, ..., N-1. \end{aligned} \tag{3.10}$$

The mapping $F(t, x, u)$ depends on the method of discretization, which will be discussed in detail later for nonlinear system models. If the system is linear and time invariant, this mapping can be obtained from a number of well-known discretization methods including the zero-order-hold method and Tustin's method. Also, if the system is linear and time invariant, convexity of the optimization problem is preserved. However, nonlinear system models generally result in loss of convexity in the constraint equations and thus add additional complexity to solving the optimization problem online.

MPC designers are not limited to the constraints given in Equation (3.10). The most common additional constraints place bounds on the feasible system states and feasible control actions. If the feasible set of system states is represented by $\mathbb{X}$ and the feasible control action set by $\mathbb{U}$, Equation (3.10) can be rewritten as Equation (3.11).

$$\begin{aligned}
&\min_{x,u} \quad \sum_{k=0}^{k=N-1} T\left[x^T(k)Qx(k) + u^T(k)Ru(k)\right] + V_f(x(N)) \\
&\qquad\qquad s.t. \\
&\qquad\qquad x(0) = x(0), \\
&x(k+1) = F(t(k), x(k), u(k)) \qquad \forall k = 0, 1, ..., N-1, \\
&\qquad\qquad x(k) \in \mathbb{X} \qquad \forall k = 0, 1, ..., N, \\
&\qquad\qquad u(k) \in \mathbb{U} \qquad \forall k = 0, 1, ..., N-1.
\end{aligned} \tag{3.11}$$

The bound set $\mathbb{U}$ is especially useful for incorporating the effect of control saturation due to actuator limits into the prediction horizon. Likewise, if there is a practical reason for limiting the desired states to the bounded set $\mathbb{X}$ (e.g., keeping a robot's position within the bounds of a room), a successful optimization iteration guarantees that the model will respect these limits. Other common constraint types include linear matrix inequality constraints such as in Equation (3.12) and Equation (3.13), or general linear/nonlinear inequality and equality constraints as in Equation (3.14) and Equation (3.15) respectively.

$$Ax \geq b \tag{3.12}$$

$$Cu \geq d \tag{3.13}$$

$$g(x, u) \geq 0 \tag{3.14}$$

$$h(x, u) = 0 \tag{3.15}$$

Additional constraints must be added with careful consideration of the desired optimization solver. Linear constraints are generally acceptable when using a convex optimization solver,

such as CVX [66]. Nonlinear constraints in the form of Equations (3.14–3.15) require the use of a solver that can handle nonlinear/non-convex optimization problems. Because this thesis focuses on the use of nonlinear model predictive constraints, additional nonlinear constraints do not add much more complexity from the optimization solver's perspective. Whatever solver is selected, the next step in the optimization problem setup is choosing the form of the decision variable as well as the method of discretization for the model prediction constraints such that the problem definition can be transformed from the form given in Equation (3.11) to a form the solver can understand.

The class methods used in this thesis to solve the optimal control problem are known as *direct* methods. Direct solver methods can be summarized as "first discretize, then optimize" as opposed to an indirect method which can be summarized as "first optimize, then discretize" [67]. The particular variant presented in this thesis is known as the direct multiple-shooting method. In multiple-shooting, the model prediction equality constraints are approximated using a numerical integration technique. The simplest numerical integration technique to demonstrate is Euler's method, shown in Equation (3.16).

$$x(k+1) = F(t(k), x(k), u(k)) \approx x(k) + Tf(t(k), x(k), u(k)) \tag{3.16}$$

This method uses only a single finite difference to approximate the mapping $F(t(k), x(k), u(k)$ seen in Equations (3.10–3.11). The accuracy of Euler integration relies on a sufficiently small discretization time step, T. To increase the model prediction accuracy, typically a higher order Runge-Kutta integration method is employed. In this thesis, the model prediction constraints

are implemented using the popular Runge-Kutta 4th order (RK4) approximation unless otherwise noted. The RK4 algorithm is summarized in Equation (3.17).

$$K_1 = Tf\left(t(k), x(k), u(k)\right) \tag{3.17a}$$

$$K_2 = Tf\left(t(k) + \frac{T}{2}, x(k) + \frac{K_1}{2}, u(k)\right) \tag{3.17b}$$

$$K_3 = Tf\left(t(k) + \frac{T}{2}, x(k) + \frac{K_2}{2}, u(k)\right) \tag{3.17c}$$

$$K_4 = Tf\left(t(k) + T, x(k) + K_3, u(k)\right) \tag{3.17d}$$

$$x(k+1) = F(t(k), x(k), u(k)) \approx x(k) + \frac{1}{6}\left(K_1 + 2K_2 + 2K_3 + K_4\right) \tag{3.17e}$$

Note that a constant input, $u(k)$, is assumed in each step of Equation (3.17). Euler's method, Runge-Kutta, and a variety of more complex techniques, such as the polynomial fitting collocation methods, are discussed in detail in [67].

To implement the selected numerical integration technique, the multiple-shooting method also defines the mapping of the MPC problem variables to the optimization problem decision variable, x,$u \rightarrow z$. In this mapping, both the discretized state, $x(k)$, and discretized control, $u(k)$, for the entire control horizon are included in the optimization variable, shown in Equation (3.18).

$$z = \left(x(0), x(1), ..., x(N), u(0), u(1), ..., u(N-1)\right)^T \tag{3.18}$$

The inclusion of the state trajectory in the optimization variable makes the dimension of the problem large compared to other mappings that only include the control variables; however, it results in many simple equality constraints that can be solved easily for long prediction horizon lengths. This larger optimization variable dimension is shown in Equation (3.19).

$$z \in \mathbb{R}^{n\cdot(N+1)+m\cdot N} \tag{3.19}$$

The optimal control problem in Equation (3.11) can now be restated in terms of the general decision variable, z, as in the unconstrained optimization problem given in Equation (3.3).

This reformulated constrained optimization problem is given below in Equation (3.20).

$$\begin{aligned} \min_{z} \quad & \phi(z) \\ & s.t. \\ c_i(z) &= 0 \qquad i \in \mathcal{E}, \\ c_i(z) &\geq 0 \qquad i \in \mathcal{I}. \end{aligned} \tag{3.20}$$

The constraints, $c_i(z)$, are now expressed in terms of the generic decision variable z and divided into the set $\mathcal{E}$ of equality constraints and set $\mathcal{I}$ of inequality constraints. Letting $z(x_k)$ and $z(u_k)$ denote the indexing of the optimization variable in Equation (3.18) for the state $x(k)$ and input $u(k)$ respectively, the model prediction constraints in the set $\mathcal{E}$ are defined in Equation (3.21).

$$\begin{aligned} z(x_0) - x(0) &= 0 \\ z(x_{k+1}) - F(t(k), z(x_k), z(u_k)) &= 0 \qquad \forall k = 0, 1, ..., N-1 \end{aligned} \tag{3.21}$$

The remainder of the constraints in the sets $\mathcal{E}$ and $\mathcal{I}$ from Equation (3.20) must also be translated from the constraints given in Equations (3.11–3.15).

3.3 Path Following and Obstacle Avoidance MPC

The major goal of this thesis is to develop and test applications of MPC for trajectory following integrated with obstacle avoidance. The models given in Chapter 2 will be restated (in order of increasing complexity) such that their commonalities can be exploited to create a similar reference trajectory and obstacle avoidance constraint set for each different model implementation. The simplest model to implement is the planar kinematic bicycle model, first given in Equation (2.16) and shown again in Equation (3.22).

$$\frac{d}{dt}\begin{pmatrix} E \\ N \\ \psi \end{pmatrix} = \begin{pmatrix} |\vec{v}|\cos(\nu) \\ |\vec{v}|\sin(\nu) \\ \frac{|\vec{v}|\sin\left(\arctan\left(\frac{b\tan(\delta)}{L}\right)\right)}{b} \end{pmatrix} \tag{3.22}$$

This model only requires the identification of the vehicle's wheel-base length, L, and the longitudinal weight split for the parameter b; however, the downside of the kinematic model for MPC is that the steady-state handling assumption does not hold for control applications that require a dynamic steering response, such as a quick evasive maneuver. To more accurately capture the vehicle's lateral transients, the more complex dynamic bicycle model presented in Equations (2.29–2.30) is combined with the vehicle's planar motion model to yield the planar dynamic model given in Equation (3.23).

$$\frac{d}{dt}\begin{pmatrix} E \\ N \\ \psi \\ v_y \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} v_x\cos(\psi) - v_y\sin(\psi) \\ v_x\sin(\psi) + v_y\cos(\psi) \\ \dot{\psi} \\ \frac{-\left(C_{\alpha_f}+C_{\alpha_r}\right)}{mv_x}v_y + \frac{-\left(aC_{\alpha_f}-bC_{\alpha_r}\right)-mv_x^2}{mv_x}\dot{\psi} + \frac{C_{\alpha_f}}{m}\delta \\ \frac{-\left(aC_{\alpha_f}-bC_{\alpha_r}\right)}{I_{zz}v_x}v_y + \frac{-\left(a^2C_{\alpha_f}+b^2C_{\alpha_r}\right)}{I_{zz}v_x}\dot{\psi} + \frac{aC_{\alpha_f}}{I_{zz}}\delta \end{pmatrix} \tag{3.23}$$

This higher order model gives a more feasible steering trajectory compared to the model in Equation (3.22). On the other hand, the dynamic model has many more complex terms to identify. The tire stiffness values, C_{α_f} and C_{α_r}, can be particularly hard to identify correctly because of the true non-linear behavior of the tires shown previously in Figure 2.6. The dynamic model also requires a much lower discretization time-step, T, when compared to the kinematic model. Also note that the vehicle's longitudinal speed, v_x, appears in the denominator of the coefficients of the states v_y and $\dot{\psi}$. Therefore, at low speeds, these terms can create numerical instability in the model; see Equation (3.23). All of these model differences must be considered along with the controller's other design requirements, including the intended operational environment, to effectively use them in an MPC application, but it is the models' similarities that make it possible to create a common trajectory tracking and obstacle avoidance interface.

The models given in Equations (3.22–3.23) both output a two-dimensional position and orientation in the fixed east–north plane. The common planar motion states, $x_p \in \mathbb{R}^3$, are given in Equation (3.24).

$$x_p = (E, N, \psi)^T \tag{3.24}$$

If the desired state trajectory, x_{des}, is formulated in terms of just the states of x_p, then the same reference trajectory form can be used with either the model in Equation (3.22) or Equation (3.23). Likewise, if known obstacle information can be expressed in terms of the states in x_p then either model can accommodate the same obstacle avoidance constraints. Also note that both models have inputs of the vehicle's speed in some form and the vehicle's front steer angle, such that the common input vector, $u \in \mathbb{R}^2$, is defined in Equation (3.25).

$$u = (v, \delta)^T \tag{3.25}$$

The models in Equations (3.22–3.23) can define the same bound constraints in the set $\mathbb{U}$ for the steering angle based on the physical limits of steering, δ_{max}.

$$\{u \in \mathbb{U} \mid -\delta_{max} \leq \delta \leq \delta_{max}\} \tag{3.26}$$

However, the bound constraints set up for velocity, given in Equation (3.27), have a less clear definition.

$$\{u \in \mathbb{U} \mid v_{min} \leq v \leq v_{max}\} \tag{3.27}$$

While it is possible to set the lower bound as $v_{min} = 0$ in the kinematic model in Equation (3.22), it may be desirable to change both the upper and lower bounds dynamically based on the maximum acceleration/deceleration allowed over the prediction horizon. When using the dynamic model in Equation (3.23), the lower bound, v_{min}, can guarantee that the model prediction will remain numerically feasible at low speed (a good rule-of-thumb for this is $v_{min} = 1\ mph$). In a hybrid control approach, activating the lower velocity bound could even be used to trigger a switch to a controller that is designed only to brake to a smooth stop.

To accomplish trajectory following, the desired trajectory is incorporated into an MPC objective function in the form of Equation (3.8). Consider a vehicle with the current state $x(0)$ and a number of discrete way-points, x_{des_i}, in the east–north plane as shown in Figure 3.1 These discrete way-points can come from a number of sources, including sampling of a map-based path, the projection of a pre-planned trajectory, or even a leader vehicle's previous state. As

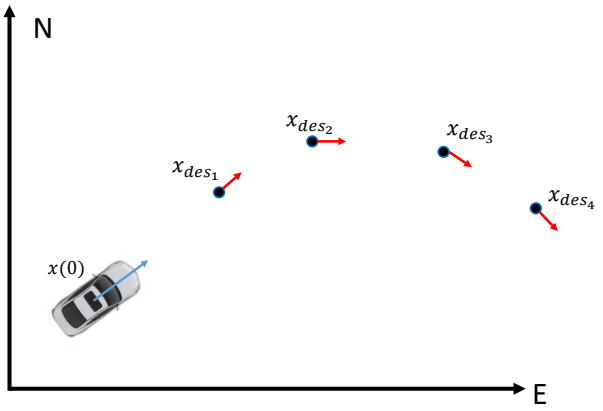


Figure 3.1: Vehicle With State $x(0)$ and a Number of Way-points x_{des_i} in the State-space

previously mentioned, this discrete trajectory is usually defined in terms of the states in x_p and can be used for both models in Equations (3.22–3.23). For the kinematic model in Equation (3.22), any desired state $x_{des} = (X_{des}, Y_{des}, \psi_{des})^T$ defines the reference for the entire state-space. For the dynamic model in Equation (3.23), any cost due to error in the lateral dynamic states can be ignored by setting the diagonals of the state weighting matrix Q associated with v_y and $\dot{\psi}$ to zero. If the way-points are spaced closely, the curvature of the discrete path could also be used to generate the references $\dot{\psi}_{des}$ and $v_{y_{des}}$ to augment each reference point. In addition to the definition of all the desired trajectory way-points, some consideration must be given to their spacing along the prediction horizon of each control iteration. If only the first way-point, x_{des_1}, in Figure 3.1 is considered reachable, then it should be held constant over the entire prediction horizon. Conversely, if all way-points, x_{des_1} through x_{des_4}, are reachable within a single prediction horizon, they should be included as a varying reference with a spacing that corresponds to the frequency at which the way-points were generated. Conditioning the reference to best accomplish the trajectory-following objective depends heavily on the method in which the trajectory is generated and provided to the MPC. The methods used to generate trajectories for the applications in this work will be discussed further in the next two chapters.

Although obstacle avoidance is certainly an objective for application of MPC in this thesis, it is not modeled in the objective function. Obstacle avoidance is handled via a set of inequality constraints that can guarantee that the states of a feasible solution are not in collision with a known obstacle. Additional to the guarantees provided by using hard constraints, this method

was selected because it is a simple way to incorporate collision checking to every way-point along the horizon. To relate an obstacle to the vehicle states, known obstacle positions are expressed in the fixed east–north reference frame as the coordinate pair (X_{obst}, Y_{obst}). The type of collision constraint used in this thesis is a circle-to-circle collision constraint, which checks that the distance between the vehicle and any known obstacle must be greater than the estimated radius of the obstacle, r_{obst}, and a safety radius around the vehicle, r_{safe}. The collision distance and avoidance radii are shown in Figure 3.2. A set of inequality constraints for avoiding a

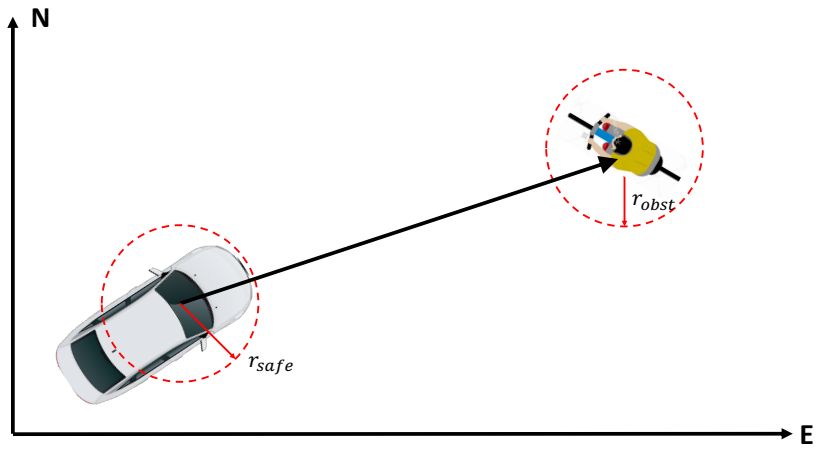


Figure 3.2: Circle-to-circle Collision Distance

single obstacle over the entire prediction horizon is given in Equation (3.28).

$$\sqrt{(X_{obst} - X(k))^2 + (Y_{obst} - Y(k))^2} \geq (r_{obst} + r_{safe}) \qquad \forall k = 0, 1, ..., N \tag{3.28}$$

The constraints in Equation (3.28) are easily expressed in the general form of inequality constraints in the set $\mathcal{I}$ from Equation (3.20).

$$\sqrt{(X_{obst} - X(k))^2 + (Y_{obst} - Y(k))^2} - (r_{obst} + r_{safe}) \geq 0 \qquad \forall k = 0, 1, ..., N \tag{3.29}$$

One of the major advantages of the circle-to-circle avoidance constraint is its simplicity. Each collision constraint reduces to a single dimension so many constraints can be added without exceeding the feasible problem size. The drawback of this simple representation of collision is

that a circular safety boundary around the vehicle may be overly conservative. Given the prototype nature of the implementations presented in this thesis, both simplicity and conservatism were prioritized.

3.4 MPC Implementation

In the previous sections, the general form of the optimal control problem has been formulated and the control objective and constraints for trajectory following and obstacle avoidance have been presented, but no consideration has been given to how the optimization will be solved in an online control application. A majority of the state-of-the-art optimization techniques that make MPC real-time feasible are gradient based. A gradient-based method uses derivatives of the optimization objective to iterate towards the minimizing solution. Quadratic objectives like those present in Equations (3.4–3.11) are taken advantage of by a whole class of gradient-based methods known as *quadratic programming*. In this section, the first and second order optimality conditions will be presented to explain the requirements of the modern quadratic programming software routines used extensively in nonlinear optimization software. The software packages used for formulating and solving the optimization problem for the proposed MPC implementations will also be presented. The section will then conclude with an overview of the software abstraction layer created to encapsulate the different MPC implementations such that they can be easily be swapped in and out of the larger autonomy system.

In constrained optimization, it is not sufficient to evaluate only the change in the objective function when looking for a local minimizer/maximizer. To include information about the constraint equations, a Lagrange function is constructed for the optimization problem. The Lagrangian of the optimization problem in Equation (3.20) with the vector of Lagrange multipliers λ is defined in Equation (3.30).

$$\mathcal{L}(z, \lambda) = \phi(z) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i c_i(z) \tag{3.30}$$

For any point (z^*, λ^*) to be considered optimal, a set of first-order and second-order necessary conditions must be met. The first order Karush-Kuhn-Tucker (KKT) conditions [68] are defined

below in Equation (3.31).

$$\nabla_z \mathcal{L}(z^*, \lambda^*) = 0 \tag{3.31a}$$

$$c_i(z^*) = 0 \qquad \forall i \in \mathcal{E} \tag{3.31b}$$

$$c_i(z^*) \geq 0 \qquad \forall i \in \mathcal{I} \tag{3.31c}$$

$$\lambda_i^* \geq 0 \qquad \forall i \in \mathcal{I} \tag{3.31d}$$

$$\lambda_i^* c_i(z^*) = 0 \qquad \forall i \in \mathcal{E} \cup \mathcal{I} \tag{3.31e}$$

Equations (3.31a–3.31c) simply state that the Lagrangian is not changing at the optimal point (i.e., it is a stationary point) and that the constraint equations are satisfied. Equations (3.31d–3.31e) state that all Lagrange multipliers associated with active constraints must be greater than or equal to zero so that Equation (3.31a) may be satisfied even if the gradient of the object, $\nabla_z \phi(z)$, is non-zero. In general, a point that satisfies Equation (3.31) may be a local minimizer or local maximizer. The second-order optimality condition defined in Equation (3.32) is used to determine if a KKT point is in a convex region of the feasible solution set, and is therefore a local minimizer.

$$w^T \nabla_{zz}^2 \mathcal{L}(z^*, \lambda^*) w \geq 0 \tag{3.32}$$

The direction w in Equation (3.32) is any direction that either increases the first-order approximation to the objective function or keeps it the same (i.e. $w^T f(x^*) \geq 0$). To state Equation (3.32) in a different way, the Hessian of the Lagrangian, $\nabla_{zz}^2 \mathcal{L}(z^*, \lambda^*)$, must be a positive semi-definite matrix at the stationary KKT point to be a local minimizer. These optimality conditions provide a stopping condition for iterative search schemes.

There are many different algorithms for constrained nonlinear optimization that all iterate on a quadratic approximation to the Lagrange function in Equation (3.30) by solving some

variant of the following quadratic programming sub-problem.

$$\begin{aligned} \min_{d} \quad & \frac{1}{2} d^T \nabla^2_{zz} \mathcal{L}(z_k, \lambda_k) d + \nabla \phi(z_k)^T d \\ & s.t. \\ & \nabla c_i(z_k)^T d + c_i(z_k) = 0 \qquad i \in \mathcal{E} \\ & \nabla c_i(z_k)^T d + c_i(z_k) \geq 0 \qquad i \in \mathcal{I} \end{aligned} \tag{3.33}$$

The solution to Equation (3.33) yields a new search direction, d, that is used to update the current iterative guess at the optimal solution of the original optimization problem, z_k, as shown in Equation (3.34).

$$z_{k+1} = z_k + d \tag{3.34}$$

The iterative scheme is repeated until the optimality conditions in Equation (3.31–3.32) are met to within a prescribed tolerance. For a comprehensive description of constrained nonlinear optimization algorithms, see [69]. Optimization software packages provide users with many different choices of the algorithm used in the solver. For MPC designers, it is important to understand that the definition of the optimization problem's Lagrangian and its derivatives are required by most every optimization package, regardless of how the underlying algorithmic details are changed.

The solver software used to implement the MPC designs in this thesis is IPOPT, an open-source nonlinear optimization library written in C++. IPOPT uses an interior-point (IP) method that is well suited to large-scale nonlinear problems. A basic interior-point formulation of the constrained optimization problem in Equation (3.20) is given in Equation (3.35).

$$\begin{aligned} \min_{z,s} \quad & \phi(z) - \mu \sum_{i \in \mathcal{I}} ln(s_i) \\ & s.t. \\ & c_i(z) = 0 \qquad i \in \mathcal{E} \\ & c_i(z) - s_i = 0 \qquad i \in \mathcal{I} \end{aligned} \tag{3.35}$$

The interior-point formulation's primary difference from other well-known techniques like Sequential Quadratic Programming (SQP) is that all inequality constraints are turned into equality constraints with the introduction of slack variables, s_i, and a logarithmic barrier term is added to the objective function such that any constraint violation forces the objective function value toward infinity. These features ensure that the optimal solution remains in the bounds created by the logarithmic barrier, such that as the barrier parameter goes to zero, $\mu \to 0$, the solution converges on the solution of the original problem in Equation (3.20). For more detail on IPOPT's implementation of interior-point optimization, see [70]. IPOPT also scales well for very large problems (as MPCs with large prediction horizons become) because the representations of the objective function, gradient of the Lagrangian, and Hessian of the Lagrangian provided by the user are encoded with their sparsity structures. IPOPT takes advantage of sparse matrix math libraries to avoid unnecessary computations (i.e., the necessity of multiplying and accumulating with zero) and memory overhead associated with the sparse matrices that are typical of large optimization problems. While this sparsity encoding has run-time speed advantages, it also presents an interesting challenge for the design of a general MPC software framework.

In a customizable software framework for MPC, it is desirable to be able to change components and re-use components between different control designs. Any change to an MPC's problem structure, such as an increase to the prediction horizon length or the addition of a constraint set, can completely alter the dimension and sparsity structure of the vectors and matrices that must be provided to IPOPT. An example of how the sparsity structure changes drastically with an increase to the prediction horizon is presented in Appendix B. To solve this problem, many software approaches for algorithmic differentiation have been created. Any such approach could be provided as an abstraction to solvers like IPOPT. In this thesis, CasADi was chosen as the algorithmic differentiation library upon which to build the NMPC framework [71]. CasADi provides the ability to model the objective function and constraints symbolically so that complex functions, gradients, Hessians, and all of their associated sparsity structures can be generated at run-time. CasADi also directly provides an interface to IPOPT and other optimization solvers to prototype implementations easier. In order to implement many different

variations of MPC in this thesis, a custom software abstraction from CasADi was written to facilitate the process of reusing components and integrating them into the test vehicle's software suite.

The NMPC software developed in this thesis was built with modular components from a custom library written in C++. The library was written in an object-oriented (OO) fashion to achieve modularity while abstracting the more general CasADi framework. A simple UML class diagram for the major library components is given in Figure 3.3. This diagram is given in full resolution with a code example in Appendix B. General NMPC constructs, such as a model, cost function, or optimizer, have common and well-defined interfaces, shown in blue in Figure 3.3. The library contains some specific implementations of these interfaces, such as the Euler and RK4 integrator types. Some of the other concrete components, shown in orange in Figure 3.3, allow users to instantiate and use them directly without having to create a custom or inherited type. For example, almost any kind of constraint can be constructed through creating a *CasadiSXConstraint* type. An implementation of the optimizer interface aggregates all the required components and uses them to set up the optimal control problem and abstract the actual IPOPT solver interface. A user may choose to create one or many optimizer implementations and include them in a wrapper object, shown in green in Figure 3.3, that serves as an adapter to communicate with other systems and as a controller that executes the optimizer methods. In this thesis, the wrapper object contains the ROS communication primitives that receive current state information and update control parameters from other nodes in the autonomy system. This wrapper object also sends out, or publishes, the optimized control output to actuator interfaces. This collection of modular NMPC components facilitated creating the different controllers tested in this thesis.

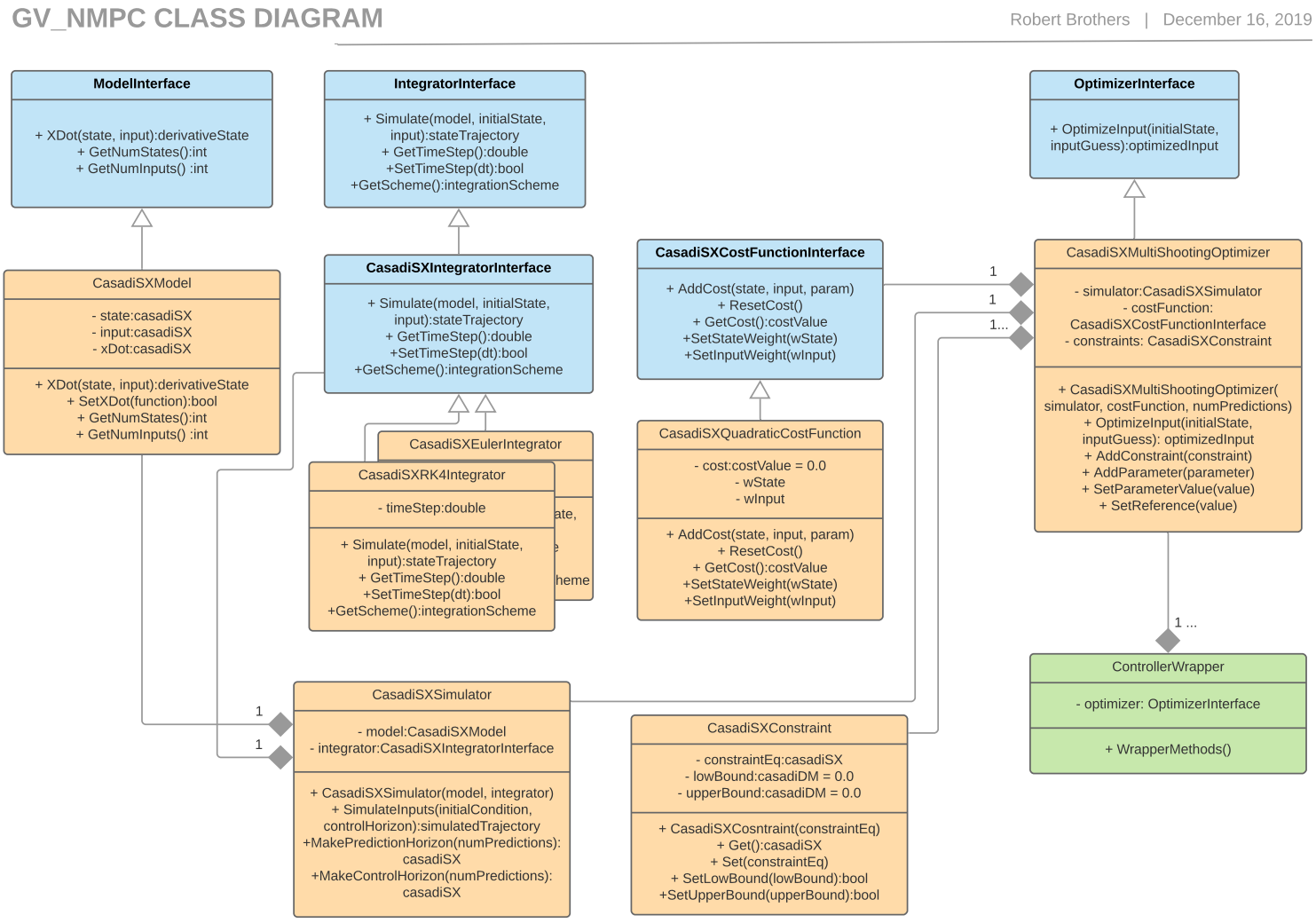


Figure 3.3: NMPC Software Library UML Class Diagram

Chapter 4

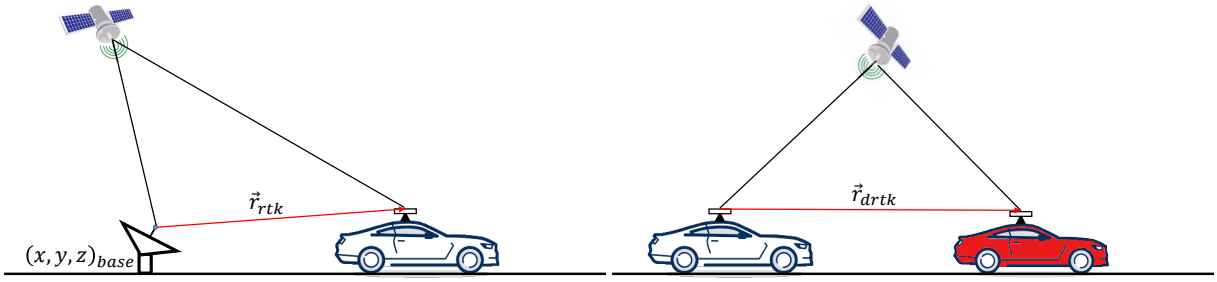
Automated Vehicle System Overview: Auxiliary MPC Interfaces

To fit MPC into the greater path planning system for a level-3 autonomous vehicle, the inputs to the MPC module must be evaluated. In both the common architectures for MPC path planning and obstacle avoidance control, shown in Figures 1.3–1.4, some form of way-point reference position and obstacle position information need to be fed to the MPC controller. This chapter will present Auburn University's current system for non-line-of-sight (long-distance) following and autonomous vehicle platooning applications and its modification to provide a way-point reference similar to the one previously shown in Figure 3.1. The current system hardware and software architecture as it relates to path following and way-point generation will also be discussed. Although this thesis does not focus on the obstacle detection and tracking technologies that provide the obstacle position feedback, a short survey of the current techniques and the necessary additional hardware will be given.

4.1 DRTK/TDCP Path Following

The current non-line-of-sight or long-distance following system developed at Auburn University allows a manually or autonomously driven leader vehicle to communicate with an autonomous follower vehicle to replicate the leader's path, with centimeter-level precision, at distances greater than most conventional line-of-sight following systems ($>$200 m) are capable of handling. This system is highly dependent on the Dynamic base Real Time Kinematic (DRTK) and Time Differenced Carrier Phase (TDCP) differential GPS techniques. Both techniques rely on the highly precise but ambiguous carrier phase measurements from the GPS receiver but only provide relative position information rather than a global ECEF position. The relative position information derived from both methods is combined to provide a way-point reference for the follower vehicle that has a much higher accuracy than traditional GPS way-points that function like a "bread-crumb" trail dropped by the leader vehicle.

DRTK is an extension of Real Time Kinematic (RTK) differential GPS technique. In RTK, a static GPS base station with a well-known global position shares code and carrier phase measurements with a nearby mobile receiver (<20 km), typically through wireless communication. The code and carrier measurements from each respective receiver are differenced to cancel common mode errors and the differential measurements are used to estimate a relative position vector from the base station to the mobile receiver, accurate to the centimeter-millimeter level. This relative position vector can be added to the known base station position to create a high-accuracy global position for the mobile receiver. In DRTK, the base receiver is also mobile and therefore only the relative position between the two receivers is known, but it does not rely on stationary infrastructure. The relative position outputs from RTK and DRTK are depicted in Figure 4.1.



(a) RTK Relative Positioning With a Stationary Receiver

(b) DRTK Relative Positioning With a Mobile Receiver

Figure 4.1

The TDCP algorithm is a differential GPS technique that can be implemented with a single receiver. The GPS carrier phase measurements from a single receiver are differenced between successive measurements epochs, from t_{k-1} to t_k, to cancel common mode errors, including the satellite clock error and atmospheric interference, and also to cancel the integer ambiguity (number of whole cycles of the carrier wave between the satellite and receiver). The resulting differenced measurement can be used to estimate a high-precision change in receiver position between measurement epochs, depicted in Figure 4.2. This method assumes that the measurements are closely spaced in time, such that the errors in the successive measurements are highly correlated. For an in-depth description of both the DRTK and TDCP algorithms, see

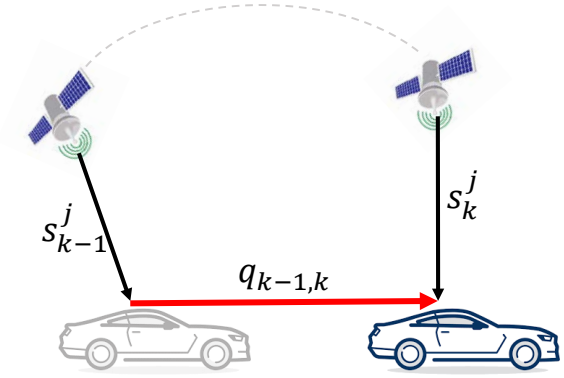


Figure 4.2: High Precision Change in Position Between Measurement Epochs from TDCP

[72, 73, 74].

To combine both the TDCP and DRTK updates into a useful control reference for platooning, the follower vehicle keeps a short history of its own position changes from TDCP and relative positions to the leader from DRTK. This time history can be used to reconstruct a new relative position vector from the follower vehicle to the position of the leader vehicle at a previous moment in time, t_{k-n}, which is significantly closer to the current vehicle position. When following this "virtual leader" position, the follower vehicle does not cut corners in the leader's path like it would if it was following based only on the current relative position to the leader. The relative position to the virtual leader, $\vec{r}_{k-n,k}$, is derived in Equation (4.1) from subtracting a previous DRTK relative position vector, $\vec{d}_{k-n}$, from all the previous TDCP updates, $\vec{q}_{k-n,k-n+1}$ to $\vec{q}_{k-1,k}$.

$$\vec{r}_{k-n,k} = \vec{d}_{k-n} - \sum_{j=k-n}^{k} \vec{q}_{j-1,j} \tag{4.1}$$

The "virtual leader" calculation can be visualized in Figure 4.3. Virtual leader following has been used previously with a simple PD controller for lateral path following control of a single vehicle [72, 73, 74] and with an advanced control design for lateral string stability in a platoon of laterally controlled vehicles [75]. These techniques are currently being used in a class 8 heavy truck platooning application that is level 2 capable [49]. In this thesis, the virtual leader method will be used to recursively reconstruct a path of way-points to the leader vehicle. This varying reference for the proposed MPC controller will include multiple way-points into the

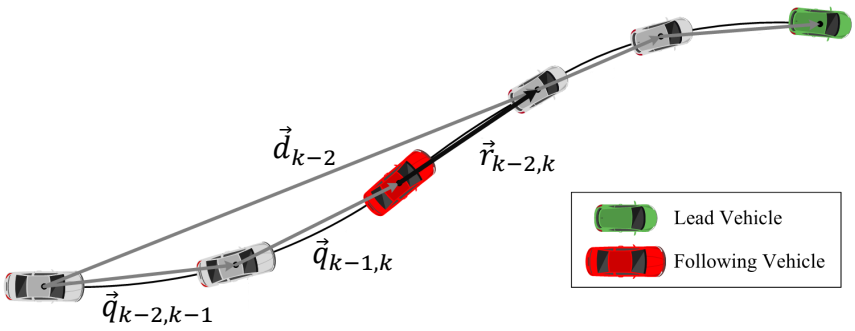


Figure 4.3: Virtual Leader Following Using DRTK/TDCP Measurements
Figure adapted from [75]

prediction horizon, giving the MPC the ability to anticipate dynamic maneuvers conducted by the leader vehicle.

4.2 Current System Hardware and Software Architecture

A minimal hardware suite is currently used in Auburn's truck platooning applications to accomplish the previously mentioned DRTK/TDCP path following and other platooning control objectives. A diagram of the hardware components and its connections is given below in Figure 4.4. All connections are centered around the rugged Linux PC that provides a platform for running the custom software. One of the primary connections to the PC is the Controller Area Network (CAN) bus. The CAN bus, originally developed by Robert Bosch GmbH, is a low-level communication protocol that is widely used in the automotive industry as a way of communicating between individual electronic control units (ECUs) in a vehicle [76]. The CAN bus connection allows low-level control of actuators like the throttle/engine, brakes, and steering, and provides measurements from stock sensors on the vehicle. The GPS unit is a dual frequency (L1 and L2) receiver that provides raw pseudorange and carrier phase measurements

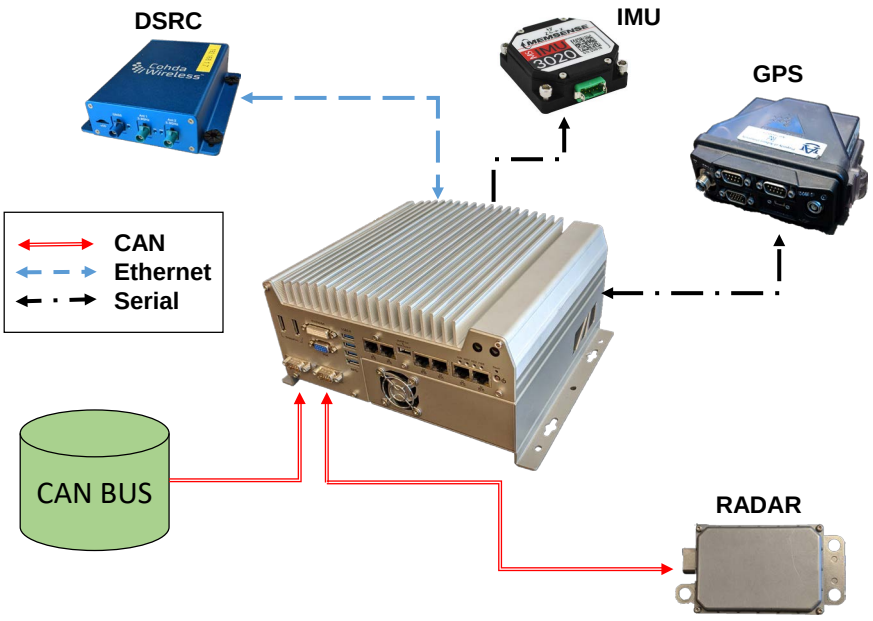


Figure 4.4: Auburn's Current Hardware Suite for CACC Truck Platooning

(at rates of 1, 2, 5, and 10 Hz) used in the DRTK and TDCP algorithms. A dedicated short range communication (DSRC) radio, operating in the 5.9 GHz band, is used to communicate the raw GPS measurements from a leader vehicle for DRTK. These sensors form the basis for doing differential GPS navigation, but the system includes other measurements for increased performance and additional features. The inertial measurement unit (IMU) provides accelerations and rotation rates at a very high update rate (100–800 Hz) and can be fused with GPS for a standalone navigation solution. Fusing the IMU with TDCP provides a higher information update rate for the output path. The RADAR sensor is used in the platooning application primarily for a higher update rate measurement of the range to the leader vehicle for more precise longitudinal control, but it can also be used to detect neighboring vehicles and obstacles. For an in-depth discussion of the design methodology for this system for Cooperative Adaptive Cruise Control (CACC) truck platooning, see [77].

All software modules in the current system are written such that they could work together with any other module, and this is accomplished with the previously mentioned middle-ware framework ROS. Each piece of the system is written as a standalone process, or ROS node, that can communicate with other processes, or nodes. ROS nodes typically communicate over ROS

topics with standardized messages in a publisher–subscriber fashion. Any node may publish data to a topic and any node that is listening to that topic will receive the data in a callback for processing. This distributed software framework has many advantages for complex system designs. It enforces standardized communication between nodes and allows each node to be modular and specific to its allocated task as well as interchangeable with other nodes that have the same input/output interface. The ROS software architecture that will be described below is only a piece of a complete truck platooning system. It will be briefly outlined in terms of its relation to the way-point path generation and hardware described above.

A given ROS node in the automated driving system can be classified generally as either a hardware driver, estimator/processing algorithm, or controller node. The hardware drivers and estimator nodes from the current truck platooning system are shown in Figure 4.5. The proposed MPC controller will be added as the primary controller node for the system in this thesis. Each node in the diagram is represented as an oval and is named according to its function. The

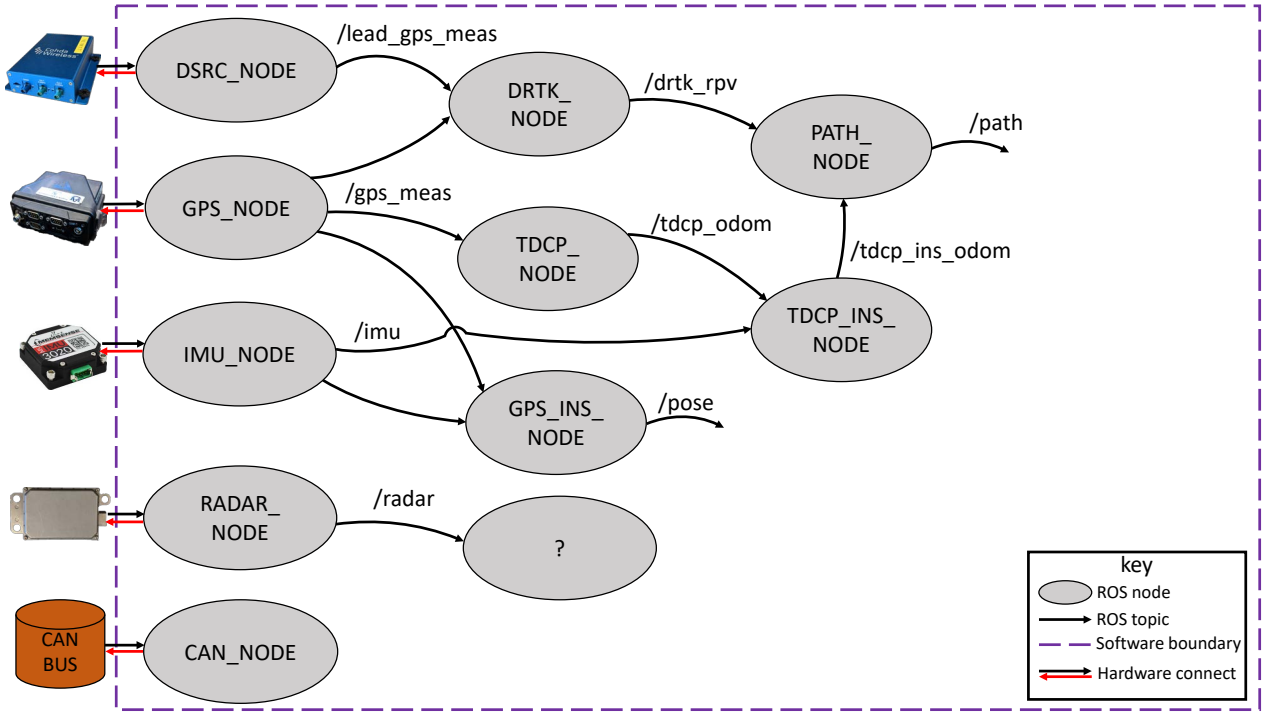


Figure 4.5: Subset of Auburn's Current Software Architecture for CACC Truck Platooning that Relates to Way-point Path Generation

ROS topics (streams of data communication) are represented by the arrows connecting each node. The hardware drivers (e.g., *gps_node* and *dsrc_node*) are grouped in the left of Figure

4.5. Their primary function is to implement the communication protocol specified by the sensor and output the sensor data in a standard ROS message format that other nodes can consume. The estimation and data processing algorithm nodes are shown in the right of Figure 4.5. The desired way-point data that will provide input to the MPC node comes from a high-level path generation algorithm in the *path_node*, which in turn relies on the *drtk_node* and *tdcp_ins_node* estimation processes, which in turn rely on the sensor data from the *gps_node*, *imu_node*, and *dsrc_node*. Notice that the RADAR hardware driver is connected to an unknown node. In the current system, it is connected to yet another estimator process, but it could be connected to an obstacle detection and tracking node. In fact, an obstacle detection and tracking node could take a variety of sensor inputs, which will be discussed in the next section.

4.3 Obstacle Detection and Tracking

Detecting and tracking obstacles for automated driving applications is a large field of ongoing research; however, significant advancements have been made in the past couple decades due to the decreasing cost of "vision" sensor technology and the increase in available computing power for mobile platforms. The three primary sensors used in automated driving applications are RAdio Detection And Ranging (RADAR), Light Detection And Ranging (LiDAR), and cameras [78, 79]. Automotive RADARs typically have a small field of view but a long maximum range ($>$100 m) and are robust in varying weather conditions. To take advantage of these strengths, researchers have tailored model-based filtering techniques [80] and new signal processing techniques [81] to ACC applications. ACC technology has matured to the point that many modern, commercially-available vehicles come equipped with one or more RADAR units. LiDAR, a light-based ranging sensor that is similar to RADAR, offers less range but a much larger field of view (up to full 360° horizontal resolution with multiple vertical channels). The Defense Advanced Research Projects Agency's (DARPA) Grand and Urban Challenges, held in 2005 and 2007 respectively, pushed LiDAR's popularity in autonomous vehicle research to the forefront with the new obstacle detection and navigation algorithms it enabled. The winner of the 2007 Urban Challenge, the BOSS vehicle, used a fusion of multiple LiDAR units and a RADAR unit for its novel obstacle detection and tracking algorithm [82]. Since the DARPA

Challenges, many other researchers have proposed improved LiDAR-only algorithms for both obstacle detection [83] and tracking [84]. Camera-based obstacle detection and tracking algorithms can generally be divided into two categories: monocular and stereo camera processing. Monocular approaches rely on frame-to-frame differences to estimate a pixel's or group of pixels' motion. In one example, surrounding vehicles are extracted as rectangular features and a Kalman Filter is updated with the frame-to-frame differences to track each feature [85]. Stereo camera setups can measure the depth of an object in the frame from the disparity between the two synchronized images. An example stereo camera system used the disparity image to update a Kalman Filter that tracked in-scene objects and computed the "free" or navigable space ahead of a vehicle [86]. Any of these three perception-based sensors can be used to produce a meaningful estimate of obstacle positions in an automated driving system, but the robustness of the system may be increased by combining the measurements of several sensors.

Many state-of-the-art obstacle detection and tracking methods combine individual sensor processing techniques with a multi-modal sensor fusion algorithm that is robust to noise and takes advantage of the measurement characteristics of each sensor. In 2016, Asvadi et al. presented an algorithm that accomplished detection, tracking, and prediction (for a single time epoch in the future) of an object's location in world coordinates by fusing RGB camera images, LiDAR measurements, and the vehicle's global navigation satellite system/intertial navigation system (GNSS/INS) navigation solution in a Kalman tracking filter [87]. A more recent algorithm has the ability to fuse many different configurations of LiDAR units and cameras to detect and track in-scene objects. This algorithm relies on a Markov Decision Process (MDP), a form of machine learning, to manage the individual vehicle trackers [88]. Both of these state-of-the-art algorithms have been tested with the KITTI data-set such that they can be compared against other solutions. The KITTI data-set is a collection of data-sets, which include LiDAR, camera, GPS, IMU, and RTK-corrected positions, and are publicly available for vision researchers [89, 90]. The KITTI authors also provide standard comparison metrics and a leader board for published results. The competitive nature of this direct algorithm comparison has inspired researchers to produce faster and more robust algorithms. Many of these algorithms are open-sourced for other researchers to validate [91].

Most of these obstacle detection and tracking algorithms could be implemented as a single ROS node (excluding any additional hardware drivers for additional measurement devices) in the system architecture shown in Figure 4.5. If the output of this obstacle detection and tracking node is assumed to be an obstacle position or set of obstacle positions, then that output could also be provided by a "mock" node. In other words, obstacle positions, in the same form as the output of a real obstacle detection and tracking algorithm, could be simulated in real-time for testing the proposed MPC implementations. A new system architecture with the MPC ROS node and the obstacle simulation node is shown in Figure 4.6. This obstacle simulation method

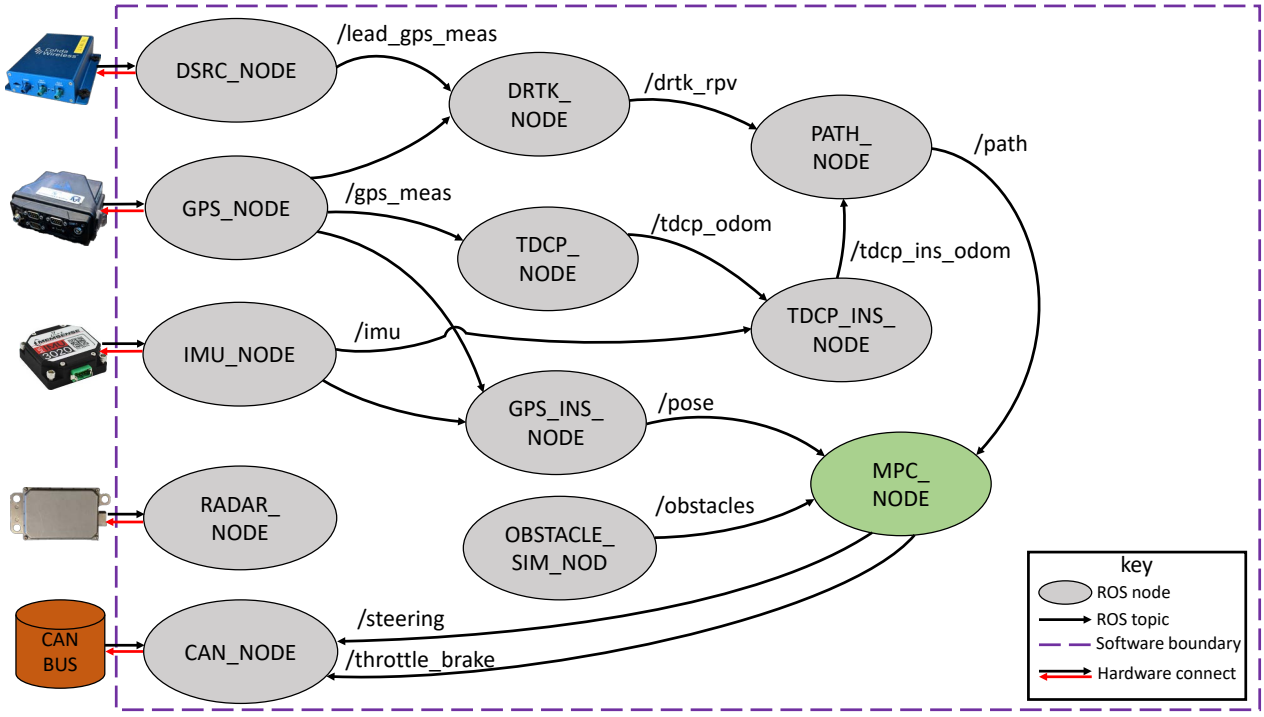


Figure 4.6: New Software Architecture Including MPC for Obstacle Avoidance and Way-point Following

has a couple advantages in testing the obstacle avoidance control features of the proposed MPC. First, it provides obstacle positions without uncertainty for a repeatable test scenario. It is also significantly safer to run in real-time as there are no actual objects the vehicle could collide with in the case of a control system failure. Figure 4.6 shows all the required inputs and outputs for the MPC controller to be implemented and tested. The next chapter presents all the simulated and real-time testing scenarios for this new system architecture.

Chapter 5

Simulation and Experimentation

This chapter presents results from Software-in-the-loop (SIL) and real-time testing of the proposed NMPC software module, implemented with the previously presented library architecture. Tuning and performance characteristics for two different NMPC implementations will be presented with a set of short simulation and real-world tests. The relevant impacts of important tuning parameters will be discussed. Obstacle avoidance feature testing is presented with the results of two experiments conducted in simulation. Advantages and improvements to the current avoidance constraint method will be discussed. Finally, a practical path following application, resulting from integrating the NMPC module into an existing automated vehicle software stack, will be presented.

5.1 Experimental Vehicle Setup

As previously mentioned in Chapter 2, the two primary test platforms for this thesis are a 2017 Lincoln MKZ with a drive-by-wire interface and a Gazebo simulation model of this vehicle. This section will explains the common vehicle setup used in all experiments on both the real and simulated vehicles. Any differences from this base-line vehicle setup for a particular experiment will be discussed in the procedure for that experiment.

The drive-by-wire ROS software interface is nearly identical for both the real and simulated vehicle. Graphs of the ROS nodes that make up this drive-by-wire interface for the Gazebo simulation model and for the real vehicle are shown in Figure 5.1 and Figure 5.2, respectively. The */vehicle/dbw_node* in both vehicles provides transmitted and received CAN bus messages. The received CAN messages include data from on-board vehicle sensors such as GPS, IMU, and wheel-encoder data as well as actuator feedback such as steering position/velocity and pedal positions. The transmitted CAN messages include actuator commands to directly control the steering, throttle, and braking actuators. The */vehicle/ulc_node* in both vehicles is

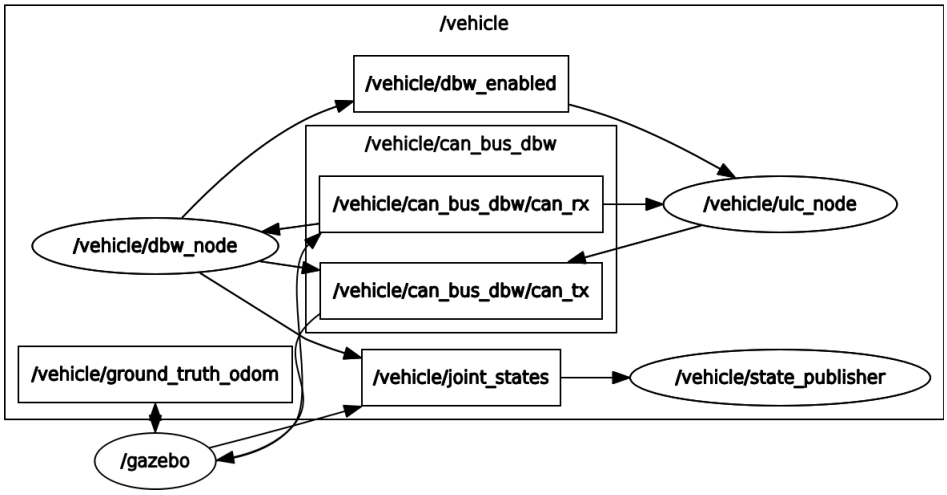


Figure 5.1: Gazebo MKZ Drive-by-wire Software Interface

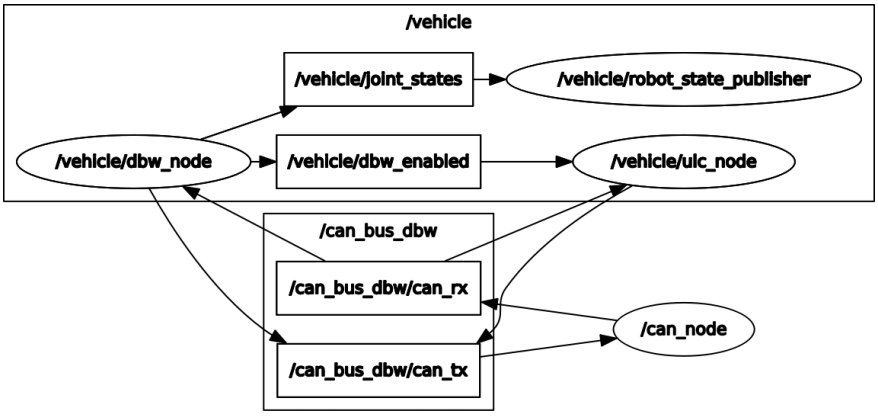


Figure 5.2: Real MKZ Drive-by-wire Software Interface

a low-level controller interface that handles steering, throttle, and braking commands and allows users to command either a desired vehicle yaw-rate or speed, or both simultaneously. The NMPC controllers in this thesis provide the direct steer angle command but utilize the */vehicle/ulc_node*'s speed controller by providing only the desired velocity command output. The Gazebo vehicle has the ability to provide a true position and velocity measurement with a high update rate (100 Hz) on the */vehicle/ground_truth_odom* topic, published directly from the Gazebo simulator. Simulation runs use these truth measurements as a source of vehicle position and velocity feedback. The same format of feedback is obtained on the real vehicle as the output of an additional estimator ROS node that takes input from a basic sensor suite. The additional ROS nodes used for real-time operation, including the sensor drivers, vehicle state estimator, and a reference path generator, are shown in Figure 5.3.

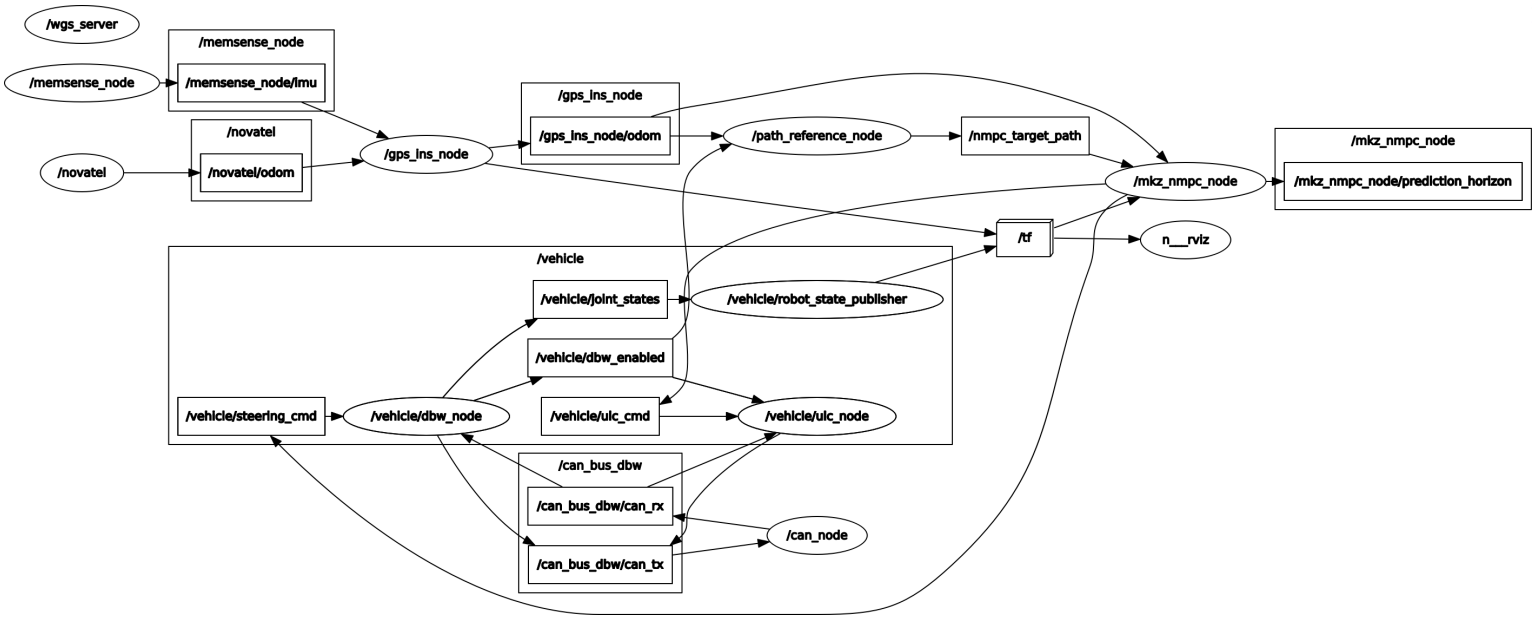


Figure 5.3: MKZ Drive-by-wire Software Interface: Real-time Controller Testing

The base-line hardware setup for the MKZ is similar to the hardware suite used for Auburn's CACC truck platooning system, previously shown in Figure 4.4. Most of the additional (non-stock) hardware components are located near the vehicle's Linux PC in the trunk, as shown in Figure 5.4. The Memsense 3020 IMU and Novatel GPS receiver with single antenna setup are shown. Although it is not shown in Figure 5.4 the PC is directly connected to an embedded CAN bus adapter module that communicates with the vehicle CAN bus and other

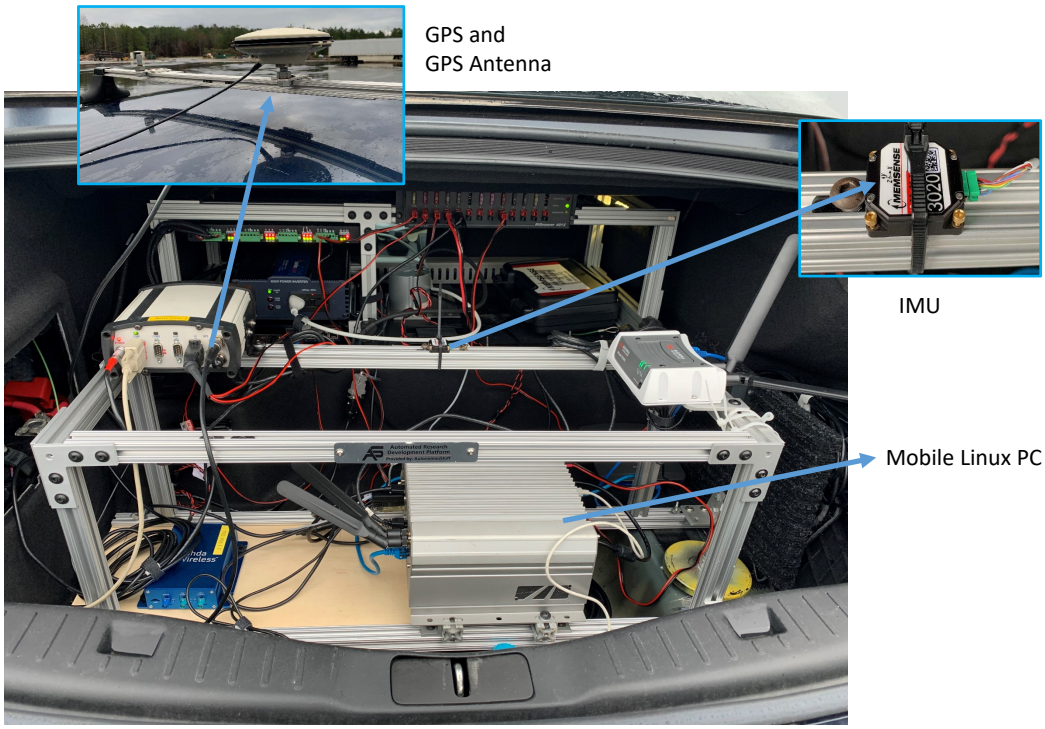


Figure 5.4: MKZ Base-line Hardware Setup

modules in the drive-by-wire kit. A Cohda wireless DSRC radio can be seen in the figure, but is unconnected for single vehicle tests.

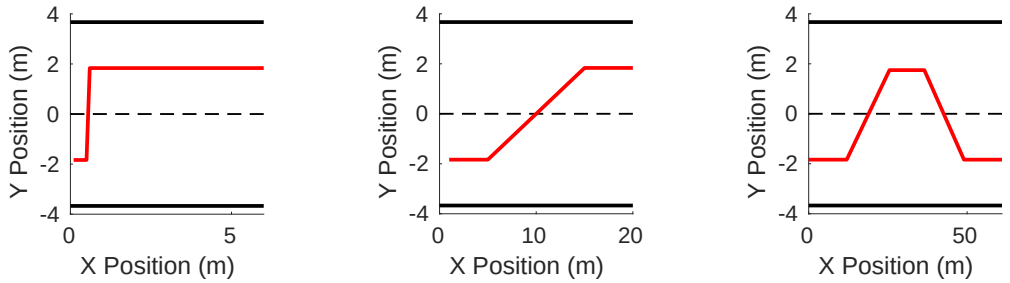
5.2 NMPC Performance Evaluation

The performance of each variant of the NMPC is characterized with a small number of case-studies using simulation and real-world experiments. This section will define each of these experimental procedures, explain the tuning procedure and performance evaluation, and present results from the tuned controllers. To conclude this section, the final tuning parameters for each controller will be presented. Prediction horizon length, prediction time step, and control effort weights will be used in subsequent sections to demonstrate applications of the model predictive controller implementations.

5.2.1 Experiment Procedures

Three experimental procedures were used to quantify and compare the performance of each NMPC implementation. The first is a step change in lateral position by a single lane's

width in the reference path the controller is tracking. The second procedure is a more gradual single lane change, where the reference position changes laterally by a single lane's width over 10 meters of travel in the vehicle's longitudinal direction. The final performance evaluation is an ISO-standard double lane change procedure that is often used to quantify a vehicle's transient handling abilities. Each of these procedures are run at multiple desired longitudinal speeds as it was previously shown in Chapter 2 that the vehicle's lateral dynamics vary with longitudinal speed. The reference paths for each procedure are shown in Figure 5.5.



(a) Step Lane Change (b) Single Lane Change (c) Double Lane Change

Figure 5.5: Reference Paths for Performance Evaluation and Tuning Experimental Procedures

Each of these experiments were run with the Gazebo simulation vehicle in the same test environment, a Gazebo world with a single 5 km long, two-lane road. This environment is shown in Figure 5.6, where the road in this environment is aligned with the Gazebo world's globally fixed X-axis. A custom Gazebo plugin was created for these simulations to generate

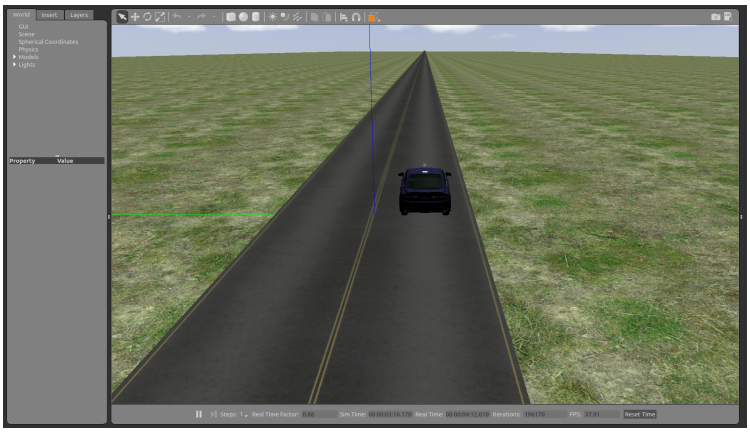
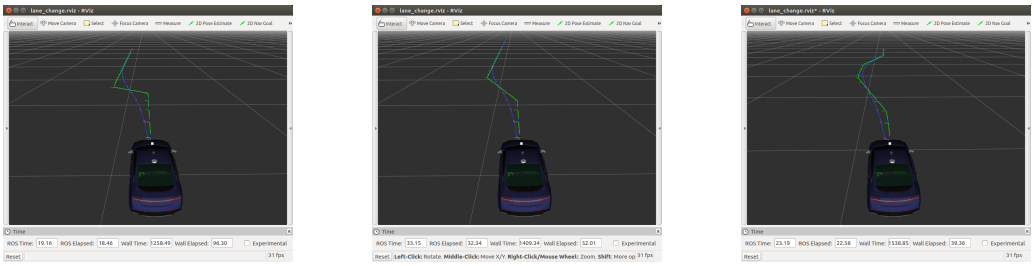


Figure 5.6: Gazebo Simulation Two-lane Road

the reference paths shown in Figure 5.5. The Gazebo published paths are shown in the ROS visualization tool RVIZ in Figure 5.7. Before a desired trajectory is published, the plugin



(a) Step Lane Change (b) Single Lane Change (c) Double Lane Change

Figure 5.7: Gazebo Plugin Published Path Visualized in RVIZ

handles ramping the vehicle up to the desired speed for the test. A similar principal was used to create a ROS node to publish the reference paths for the real-time MKZ vehicle experiments; however, the online ROS node could not rely on the assumption that the real road lies along a globally fixed axis.

The real-time test environment for these maneuvers was at the NCAT test track in an area known as the "skid pad", which is a small, unstructured area of pavement. The skid pad area is shown below in Figure 5.8. Unlike the simulation environment, this test area does not have

Figure 5.8: NCAT Skid Pad Area

5 Kilometers of open space to get the vehicle safely up to speed and execute a maneuver. The maximum testing speed that could be achieved by most maneuvers was 10 m/s due to the limited space. To generate repeatable reference paths in this small test area, it was mapped

with highly accurate RTK GPS data. One centimeter resolution reference paths were created from this map such that each maneuver would start in approximately the middle of the skid pad area. An aerial view of these reference paths on the skid pad is shown in Figure 5.9. The

(a) Step Lane Change (b) Single Lane Change (c) Double Lane Change

Figure 5.9: Skid Pad Reference Paths (1 cm Resolution) Created From RTK GPS

reference path generation ROS node, used to publish the reference paths in Figure 5.9 in real-time, was set up such that the path was translated to the vehicle's estimated position at the beginning of each test, correcting for any translation errors in the vehicle's position estimation. The reference path generation node also commanded the vehicle to a stopping position at the end of the desired reference path.

5.2.2 Horizon Sensitivity and Tuning

The NMPC's time horizon, H, has a large impact on lateral control performance. The important tuning parameters that make up the time horizon are N, the number of prediction steps, and T, the prediction time step, as shown previously in Equation (3.2). The full horizon tuning procedure and results from simulation are shown in Appendix C. A short summary of the results are presented in this section.

In this thesis, the single lane change maneuver was chosen to perform the horizon tuning procedure because it represents "normal" driving behavior; however any maneuver or a combination of maneuvers could be used to generate more data for tuning. 2-D grids of tuning parameters are shown in Table 5.1 and Table 5.2 for the kinematic model NMPC and bicycle model NMPC, respectively. The grid discretizations were picked from a range based around initial hand-tuned parameters that kept the vehicle stable through the maneuver. Each grid of

tests was run at varying desired speeds of 1, 5, 10, 15, and 20 m/s. The standard deviation on

Table 5.1: 2-D Grid of Horizon Tuning Parameters for the Kinematic Model NMPC Implementation

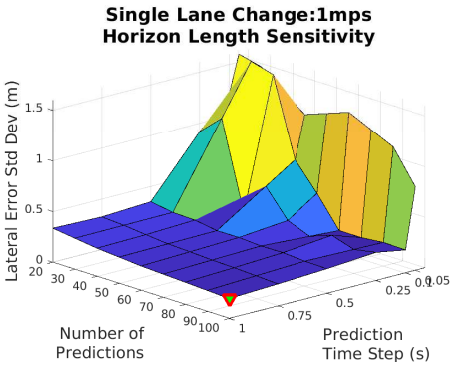
Number of Predictions (N)	Prediction Time Steps (T)				
20	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
40	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
60	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
80	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
100	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s

Table 5.2: 2-D Grid of Horizon Tuning Parameters for the Bicycle Model NMPC Implementation

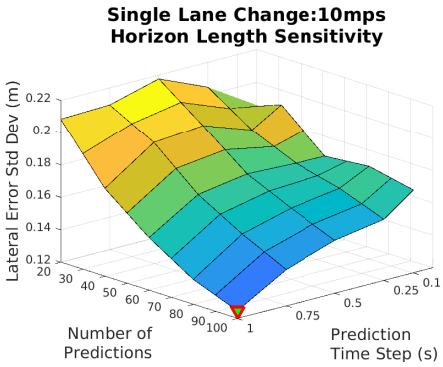
Number of Predictions (N)	Prediction Time Steps (T)			
75	0.01 s	0.02 s	0.03 s	0.04 s
100	0.01 s	0.02 s	0.03 s	0.04 s
125	0.01 s	0.02 s	0.03 s	0.04 s
150	0.01 s	0.02 s	0.03 s	0.04 s

the lateral path error is used as the performance metric for each of the horizon grid points. A 3-D surface plot of the standard deviation on the lateral path error as a function of number of predictions and the prediction time step is used to visualize and compare tuning parameters for different desired path speeds. The low-, medium-, and high-speed (1, 10, and 20 m/s respectively) results of the tuning procedure for the kinematic model NMPC controller are shown in Figure 5.10 and for the bicycle model NMPC in Figure 5.11.

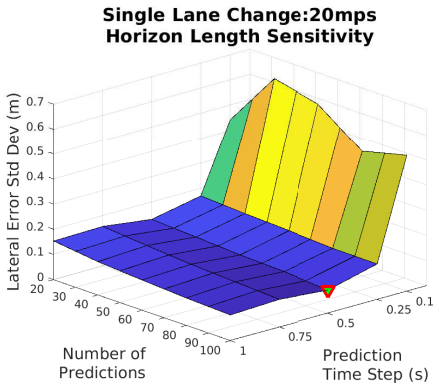
The tuning procedure reveals that the kinematic model based NMPC controller performs best in simulation with the combination that results in a long horizon time, H. Although the optimal combination across all parameters was determined to be $(N = 100, T = 0.75s)$, the alternate parameters $(N = 60, T = 0.75s)$ were selected as a compromise to reduce the run-time computation of each control iteration. Although increasing the number of predictions quickly results in a longer horizon time, the performance gain diminishes and the run-time computational costs increase. The bicycle model based NMPC controller requires a significantly smaller prediction time step compared to the kinematic model NMPC. This smaller prediction time step is required to achieve accurate predictions of the lateral handling transients included in the more detailed dynamic bicycle model. While the horizon tuning results for this implementation show



(a) 1 m/s

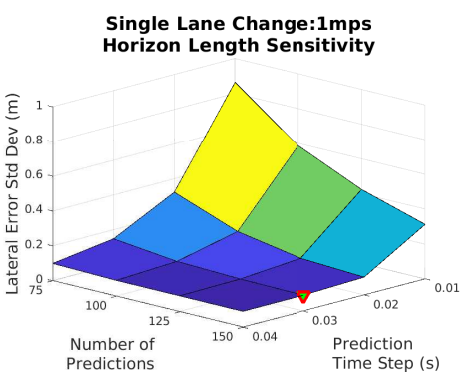


(b) 10 m/s

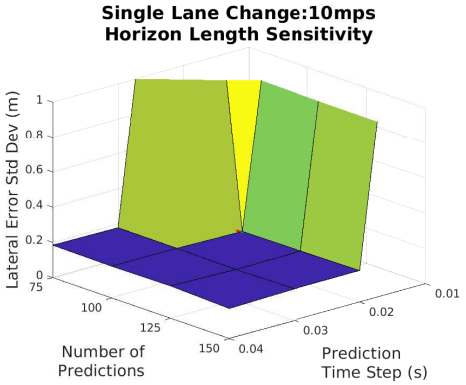


(c) 20 m/s

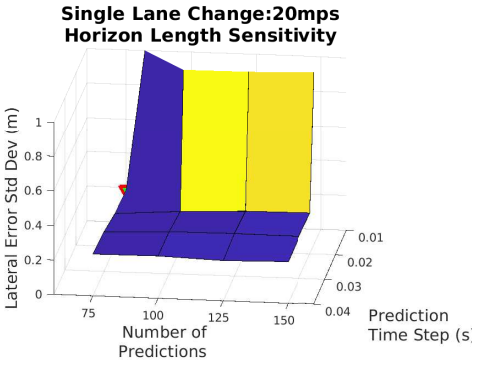
Figure 5.10: Kinematic Model NMPC Controller Horizon Tuning Results



(a) 1 m/s



(b) 10 m/s

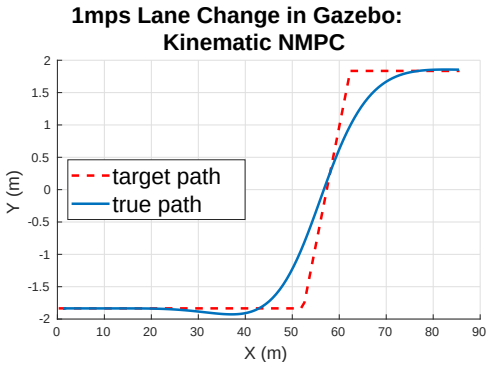


(c) 20 m/s

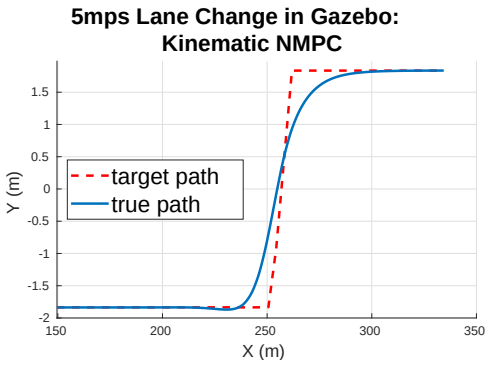
Figure 5.11: Bicycle Model NMPC Controller Horizon Tuning Results

that some of the best performance came from runs using the smallest prediction time step, the performance was only slightly degraded and more consistent across all speeds with a slightly higher prediction time step and reasonable number of predictions. The horizon tuning selected for the bicycle model NMPC is $(N = 100, T = 0.03s)$. Note that the total horizon times for the selected tuning parameters of each implementation vary widely: the kinematic model NMPC horizon time is $H = 45s$ while the bicycle model NMPC horizon time is $H = 3s$. This difference may affect the suitability of one implementation over the other depending on the desired application. For example, the kinematic model NMPC has a significant advantage in planning paths around far away detected obstacles in an avoidance scenario, while a well tuned bicycle model implementation has the advantage of tracking more aggressive short-term paths.

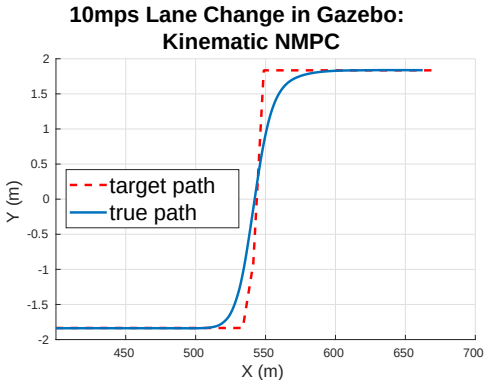
The paths of the kinematic model NMPC with $(N = 60, T = 0.75s)$ at 1, 5, and 10 m/s are shown in Figure 5.12. Similarly, the paths of the kinematic model NMPC run with the same tuning parameters in real-time (i.e. on the real MKZ vehicle) are shown in Figure 5.13. The



(a) 1 m/s

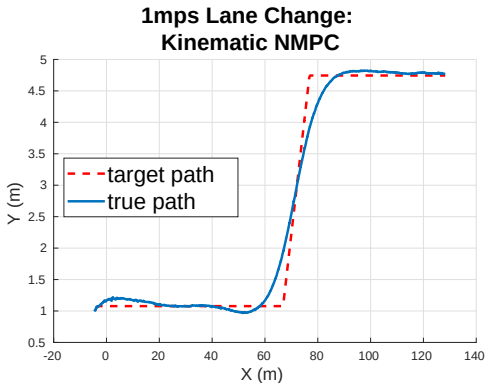


(b) 5 m/s

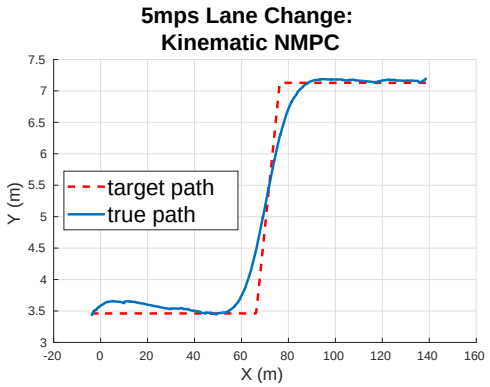


(c) 10 m/s

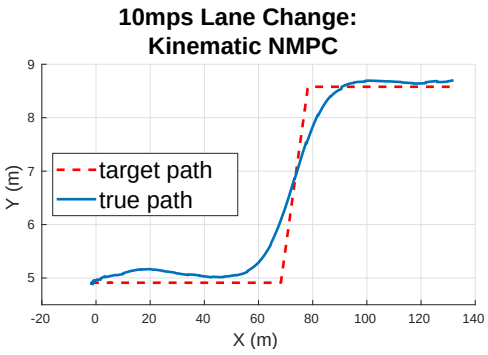
Figure 5.12: Kinematic Model NMPC Controlled Single Lane Change in Gazebo With Horizon Tuning $(N = 60, T = 0.75s)$



(a) 1 m/s



(b) 5 m/s

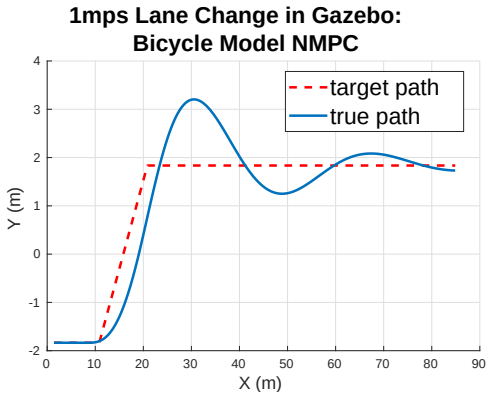


(c) 10 m/s

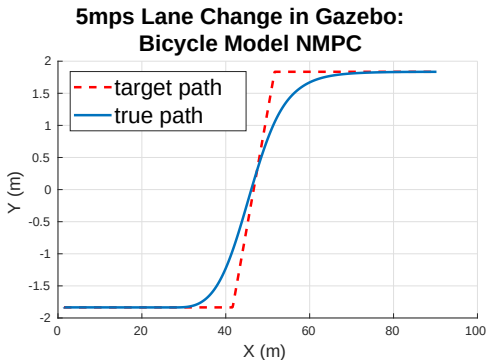
Figure 5.13: Kinematic Model NMPC Controlled Single Lane Change in Real-time With Horizon Tuning $(N = 60, T = 0.75s)$

path-tracking behavior predicted in the Gazebo simulation is very similar to the real-time results shown in Figure 5.13. Both the simulation and real-time results show the NMPC's ability to anticipate the change in the reference path's lateral position. This anticipation causes the NMPC controller to begin steering towards the final lateral position offset before a classical controller, which acts only on current and previous errors, would. Note the only major discrepancy between the simulation and real-time path tracking: in each test case shown in Figure 5.13 the controller corrects for an initial orientation offset from the path, while the simulation tests in Figure 5.12 starts each run aligned with the path.

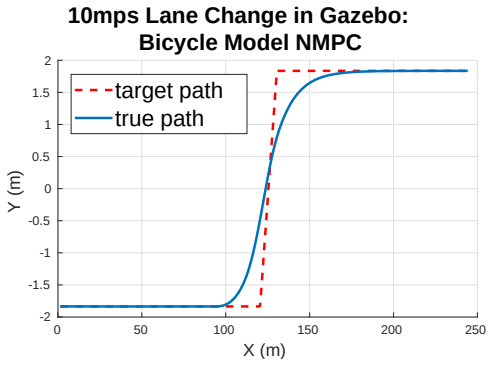
The paths of the dynamic bicycle model NMPC implementation with the horizon tuning $(N = 100, T = 0.03s)$ simulated at 1, 5, and 10 m/s are shown in 5.14. The corresponding paths from real-time testing of the same NMPC implementation are shown in Figure 5.15.



(a) 1 m/s



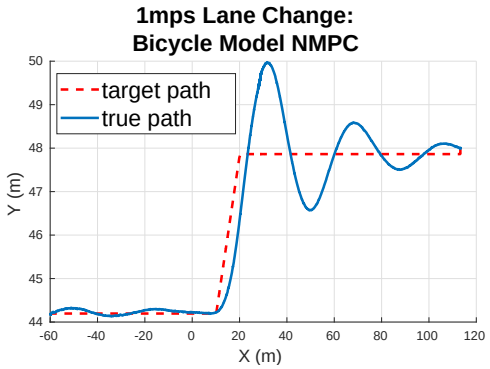
(b) 5 m/s



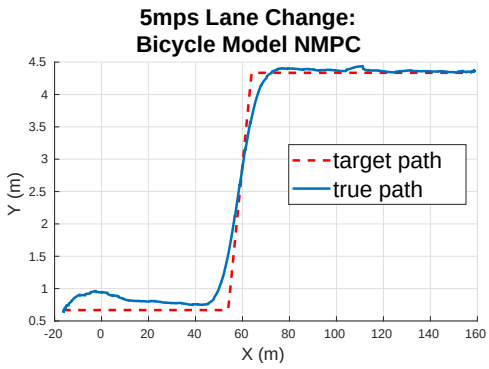
(c) 10 m/s

Figure 5.14: Bicycle Model NMPC Controlled Single Lane Change in Gazebo With Horizon Tuning $(N = 100, T = 0.03s)$

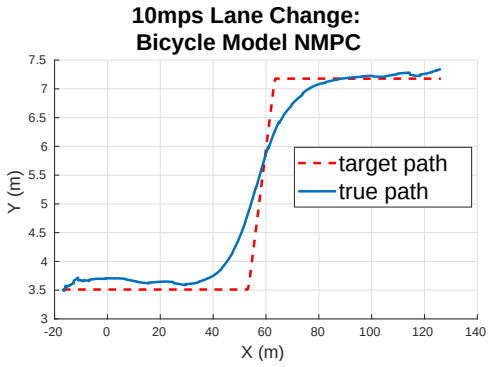
Again, the real-time tests show similar results to the Gazebo simulations. In the lowest speed



(a) 1 m/s



(b) 5 m/s



(c) 10 m/s

Figure 5.15: Bicycle Model NMPC Controlled Single Lane Change in Real-time With Horizon Tuning $(N = 100, T = 0.03s)$

runs for both the simulated and real MKZ vehicle, the bicycle model NMPC overshoots the desired path significantly. This overshoot is likely not present in the kinematic model implementation because of its much larger prediction time step. The bicycle model NMPC cannot "see" beyond the maneuver on its prediction horizon until the vehicle is almost half way through the lane change portion of the path, unlike the kinematic model NMPC, which receives way-points throughout the maneuver well before it begins to turn. During the higher-speed runs in Figure 5.14b–5.14c and Figure 5.15b–5.15c the controller can "see" through the entire maneuver before reaching the actual lane change. Much like the kinematic model NMPC, the controller anticipates the maneuver and begins to turn before reaching the actual reference change when operating at these higher speeds.

Fine tuning the NMPC's control output is achieved through modifications to the state and input weighting matrices in the cost function. Tuning these weighting matrices, given in Equation (3.8), is similar to tuning an LQR controller. Exact values on the diagonals of the state weight, Q, and control weight, R, are less important than the ratio of the values between the different matrices and between the individual diagonals. Take the state weighting matrix used in this thesis, given in Equation (5.1), as an example.

$$Q = \begin{bmatrix} 1.0 & & \emptyset \\ & 1.0 & \\ \emptyset & & 0.1 \end{bmatrix} \tag{5.1}$$

This matrix gives equal weighting to errors in the position states X and Y, and errors in the yaw state, ψ, are $1/10$ as important as errors in either position state. If the following input weighting matrix is used in the NMPC, large inputs in velocity and steer angle will be penalized with equal and 100 times the importance, respectively, given to errors in the position states.

$$R = \begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 100.0 \end{bmatrix} \tag{5.2}$$

In this thesis the input weighting matrix was tuned to achieve smoother control output and balance tracking performance and robustness.

A series of simulation experiments were conducted to gain an intuition for the impacts of tuning the input weighting matrix. The step steer maneuver test, shown in Figure 5.5a, was conducted at 20 m/s in Gazebo. This maneuver was chosen because it is the most difficult in terms of keeping the vehicle within its stability limits. Individual test runs varied one of the input weight diagonals (either the steering input weight or velocity input weight) while the other was held at 1.0. The results of these experiments show that the controller using the kinematic model is far more sensitive to changes in the velocity input weight.

The control signal outputs over each run are shown in Figure 5.16 and Figure 5.17 for desired steering and velocity, respectively. Figure 5.16 presents steering commands from four different orders of magnitude tuning values. These command histories do not vary appreciably,

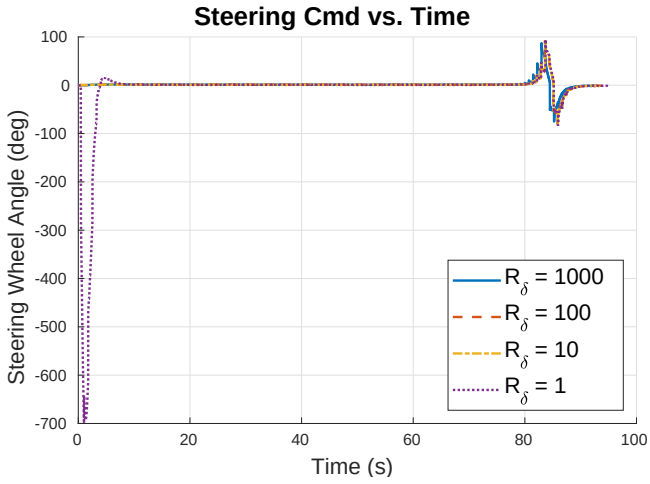


Figure 5.16: Kinematic Model NMPC Steering Control Output with Various Input Weights

except for the $R_\delta = 1$ case where the maximum steer angle is commanded and quickly corrected as the vehicle is accelerating from rest. However, Figure 5.17 shows the tuning on the velocity input has a large effect on the amplitude of a noisy command output. This noisy output is primarily due to the lack of a longitudinal dynamic model in the current control implementations. The current architecture, presented in Figure 5.3, passes this desired velocity output to a lower level controller that acts as a filter for the actual longitudinal actuator commands. While the velocity command output noise is undesirable, a higher velocity tuning value also

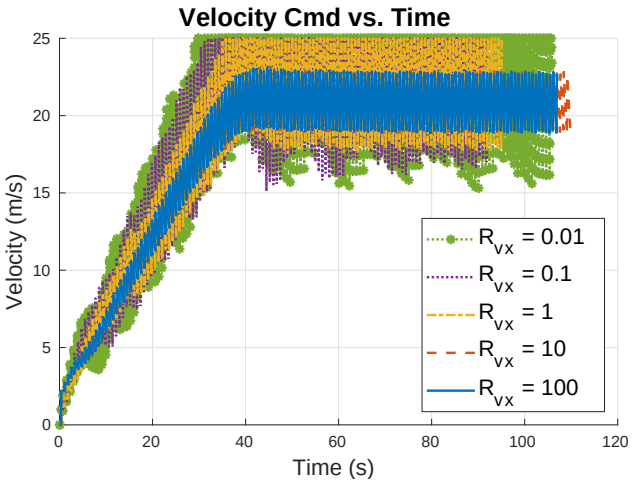


Figure 5.17: Kinematic Model NMPC Velocity Control Output with Various Input Weights

drastically affects the controller's tracking performance. The effects of changing both tuning parameters on the path tracking performance is shown in Figure 5.18 and 5.19. Again, the

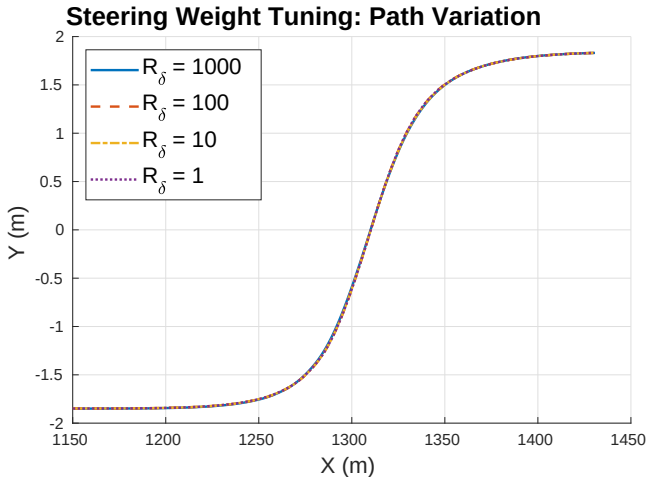


Figure 5.18: Kinematic Model NMPC Path Variation due to Steering Input Weights

steering tuning parameter has little to no effect on the variation of the controlled path. The velocity tuning parameter changes the path tracking drastically over the 5 orders of magnitude shown in Figure 5.19. In the most drastic case, $R_{v_x} = 0.01$, the optimizer failed to converge on a correct solution at the beginning of the test and appears to be "stuck in" or converged to a local minima solution that is offset from the desired path. In general, increasing the tuning value decreases the controller's settle time. In the applications shown in this thesis, achieving better path tracking is worth the trade-off for a noisier velocity command output signal since

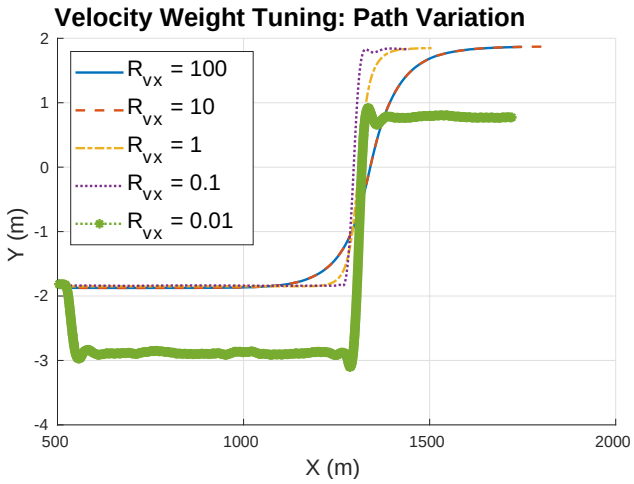


Figure 5.19: Kinematic Model NMPC Path Variation due to Velocity Input Weights

it is not directly connected to physical actuators and passes through multiple lower levels of software filtering.

The results of the input weight tuning procedure for the bicycle model NMPC implementation are similar to the kinematic model NMPC. The weighting on the velocity input has a large effect on the tracking performance and stability of the controlled vehicle. The velocity command over time for various tunings during the 20 m/s step steer maneuver is shown in Figure 5.20. At the extremely low tuning value $R_{v_x} = 0.01$ command velocity quickly saturates and bounces between the limits set on the velocity input, but this output does not destabilize the vehicle. The highest input tuning value shown, $R_{v_x} = 10$, does cause the vehicle to go unstable when it reaches the speed of the desired maneuver. There does, therefore, appear to be an optimum velocity tuning between these extremes at around $R_{v_x} = 1$. Unlike the kinematic model NMPC, the bicycle model implementation is also sensitive to the input steering weight. The steering command over time for various tunings during the 20 m/s step steer maneuver is shown in Figure 5.21. None of the tunings shown in Figure 5.21 destabilize the vehicle; however, when initially hand-tuning the controller to get the initial set of weights to test with, there was a minimum weight that stabilized the vehicle through the maneuver. This minimum weight appeared to change with the velocity (and therefore, the model) that the controller is operating on. The tuning procedure for the bicycle model NMPC was also run at 10 m/s with the same maneuver to demonstrate the change in range of appropriate tuning values for the

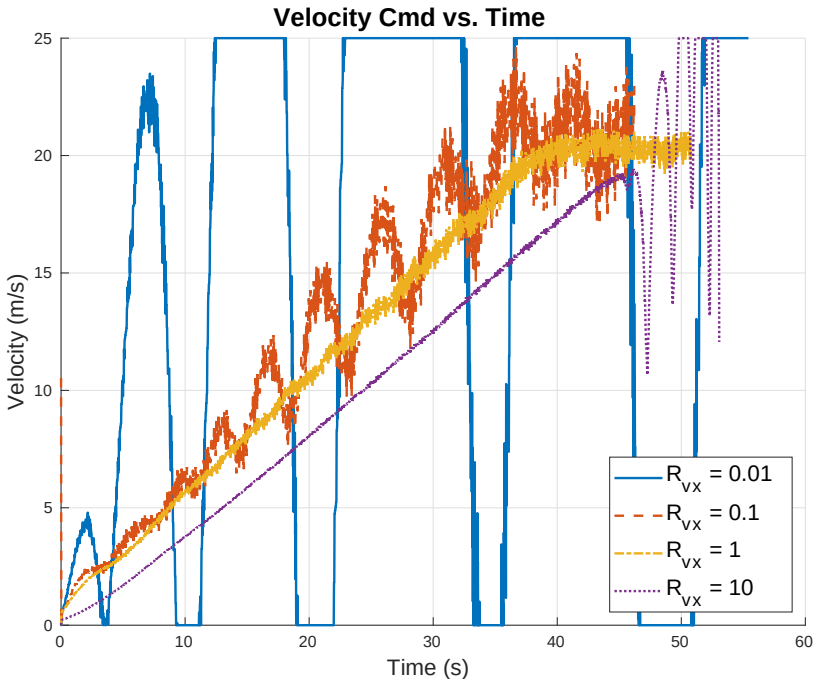


Figure 5.20: 20 m/s Step Steer – Bicycle Model NMPC Velocity Control Output with Various Input Weights

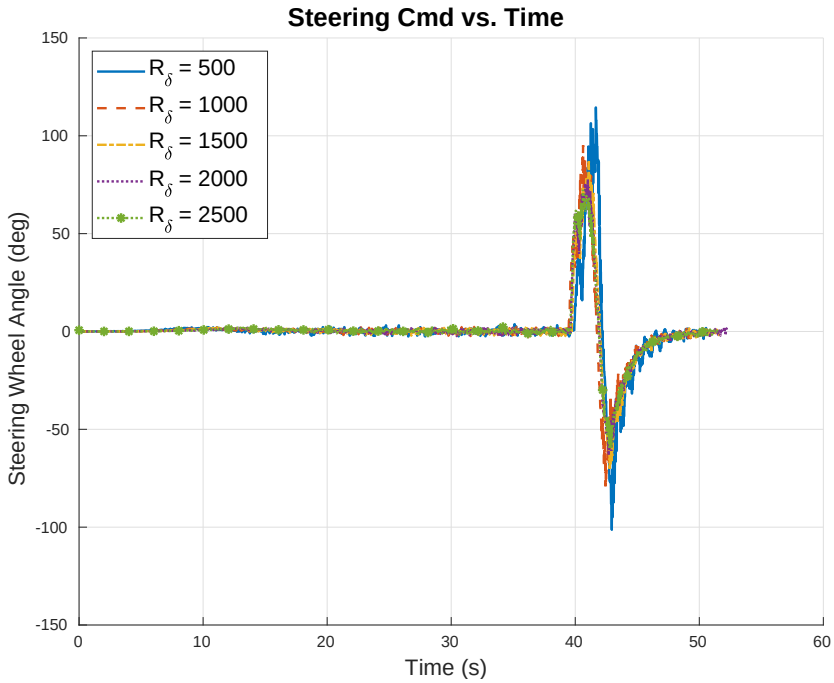


Figure 5.21: 20 m/s Step Steer – Bicycle Model NMPC Steering Control Output with Various Input Weights

steering input weight. The steering and velocity command outputs over time for this 10 m/s scenario are shown in Figure 5.22 and 5.23 respectively. Figure 5.23 shows generally the same

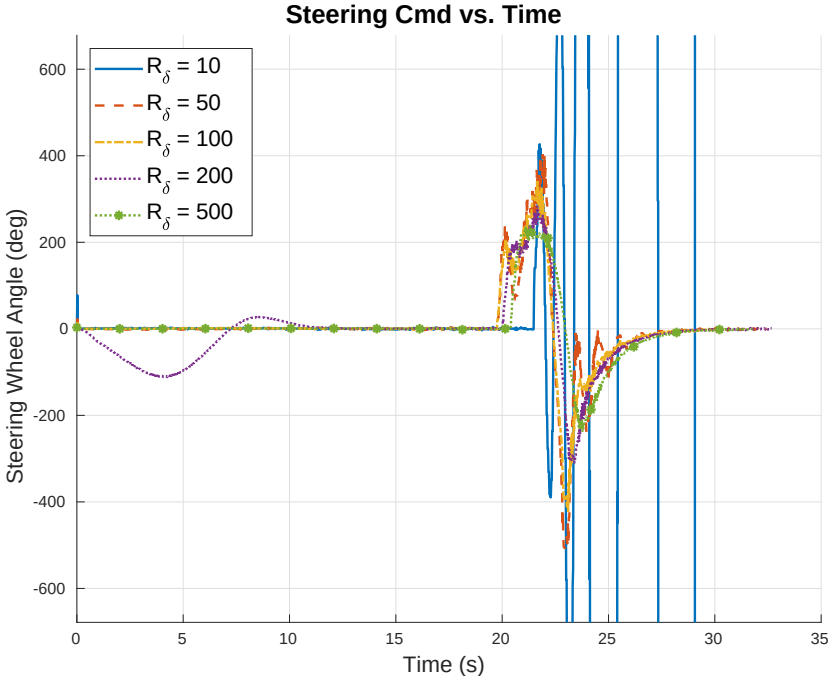


Figure 5.22: 10 m/s Step Steer – Bicycle Model NMPC Steering Control Output with Various Input Weights

trend for the velocity tuning parameter as Figure 5.20. However, Figure 5.22 shows a much different range of steering weight input values compared to the 20 m/s case. While the lowest feasible steering input weight in the 20 m/s case was given at $R_\delta = 500$, in the 10 m/s case the lowest feasible input weight is given at $R_\delta = 100$. An additional simulation with a weight below the stability threshold, $R_\delta = 10$, is also shown in Figure 5.22.

The effects of changing the steering input weight on the vehicle's trajectory at both 10 and 20 m/s is shown in Figure 5.24. Similarly, the effects of changing the velocity input weight on the vehicle's trajectory at both 10 and 20 m/s is shown in Figure 5.25. A reasonable tuning for the bicycle model NMPC implementation might be selecting the minimum steering input weight at the maximum expected operating velocity for the vehicle. However, this tuning is overly conservative for lower-speed operations, compromising on path tracking performance. For the best possible path tracking performance from this implementation, a variable or scheduled steering input weight should be selected based on the vehicle's current velocity.

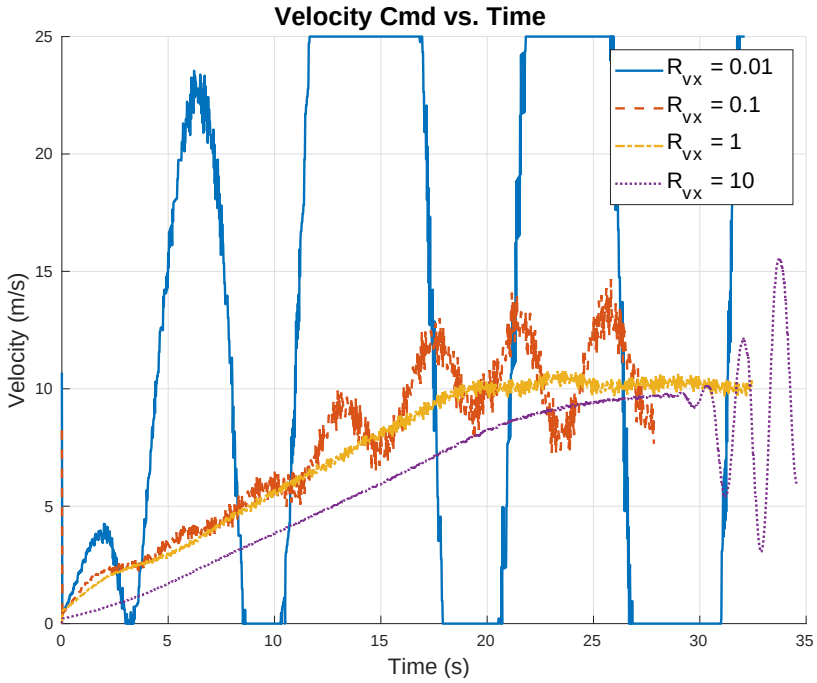
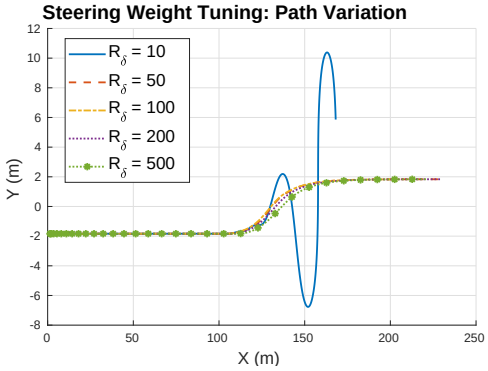
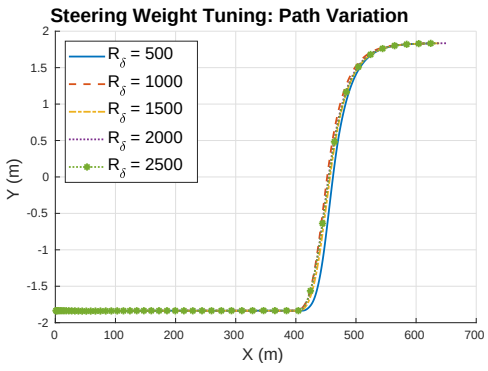


Figure 5.23: 10 m/s Step Steer – Bicycle Model NMPC Velocity Control Output with Various Input Weights

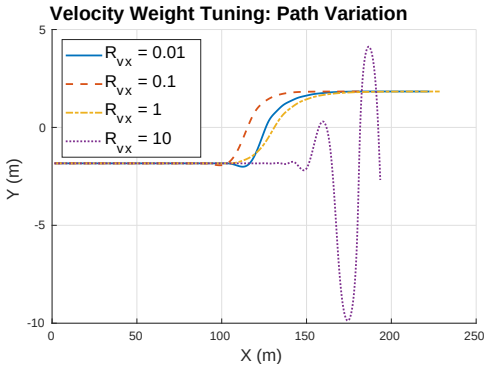


(a) 10 m/s

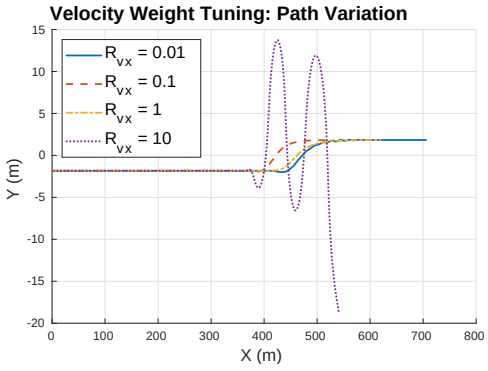


(b) 20 m/s

Figure 5.24: Bicycle Model NMPC Path Variation due to Steering Input Weight



(a) 10 m/s



(b) 20 m/s

Figure 5.25: Bicycle Model NMPC Path Variation due to Velocity Input Weights

The final tuning parameters for both NMPC controller implementations are given in Table 5.3. It should be noted that the analysis presented in this section and the horizon tuning procedure in Appendix C can be performed iteratively to further optimize control performance; however only a single tuning iteration was performed for this thesis as it gave acceptable results to prove feasibility in the presented applications.

Table 5.3: NMPC Controller Implementations Final Tuning Parameters

	Kinematic Model NMPC	Bicycle Model NMPC
N	60	100
T	0.75 s	0.03 s
H	45 s	3 s
Q_X	1.0	1.0
Q_Y	1.0	1.0
Q_ψ	0.1	0.1
R_{v_x}	0.5	0.8–1.0
R_δ	10	50–500

5.2.3 Performance Metrics

The performance of each maneuver is evaluated by analyzing how well the vehicle controls to the path laterally and longitudinally. For each run, a single test maneuver was executed at a desired speed and the lateral path error is calculated. The mean and standard deviation of the path error is plotted as a function of the desired vehicle speed to characterize the performance of the controller over the expected operating range of speeds. Similarly, for each run, the error in the controlled velocity is calculated. The mean and standard deviation of the velocity error is also plotted as a function of the desired vehicle speed. The control errors at each speed are expected to be zero mean with low standard deviation for good control performance.

Tests with both the simulation and the real MKZ vehicle were conducted from 1 to 10 m/s in 1 m/s increments for each maneuver in Figure 5.5. Additional tests were run in simulation at higher speeds of 15 and 20 m/s. While it was not possible to run real-time experiments on the MKZ vehicle for these test cases, these simulation data points show the general trend of the error results at higher speeds. The lateral path error results for the step lane change, single lane change and double lane change maneuvers are shown in Figure 5.26, Figure 5.27,

and Figure 5.28 respectively. All three error figures contain a dashed black line through zero

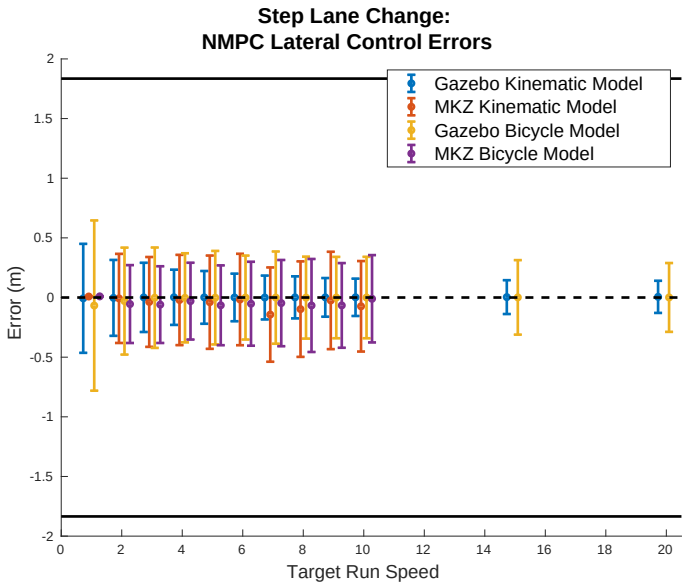


Figure 5.26: Step Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed

error and straight black lines at $\pm 1.835m$; these represent the lane center-line and outer lane boundaries (for a single lane), respectively. The kinematic model and bicycle model NMPC implementations follow the path with approximately zero mean error across all maneuvers. The simulations of the kinematic model NMPC show a convergence to a small (± 0.2–$0.25m$) path error standard deviation with increasing speed, while the real-time results for the same implementation show a consistent error standard deviation (excluding a few outlier tests) with a slight increase at the higher speeds across all maneuvers. These results suggest that the performance in real-time would only slightly degrade at higher speeds. Real-time results for the bicycle model NMPC are equivalent or slightly better than the real-time performance of the kinematic model NMPC. This implementation is also expected to have consistent path tracking in the higher speed range.

The desired speed of each test run is set as a constant (implemented by correctly spacing the reference path way-points). Therefore, the velocity error is calculated by subtracting the vehicle's recorded planar velocity from the constant desired velocity for each run. Because each test starts from rest, a velocity ramp-up period is included in the beginning of each test. These ramp-up periods are excluded in the velocity error calculation so as to not offset the

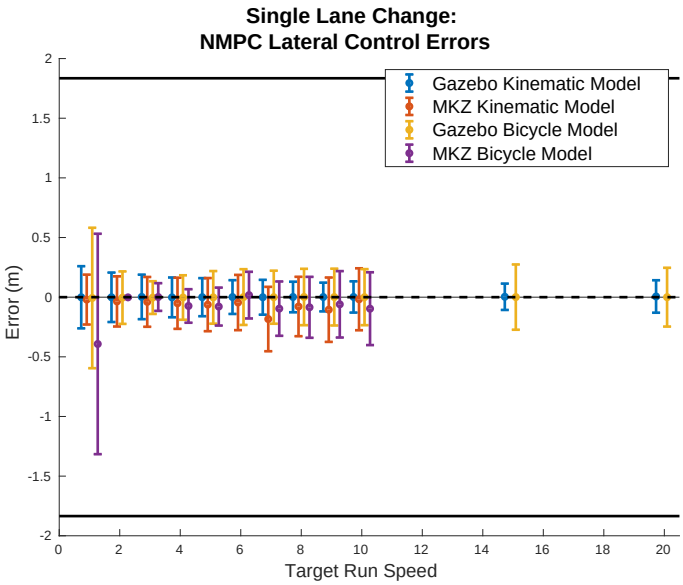


Figure 5.27: Single Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed

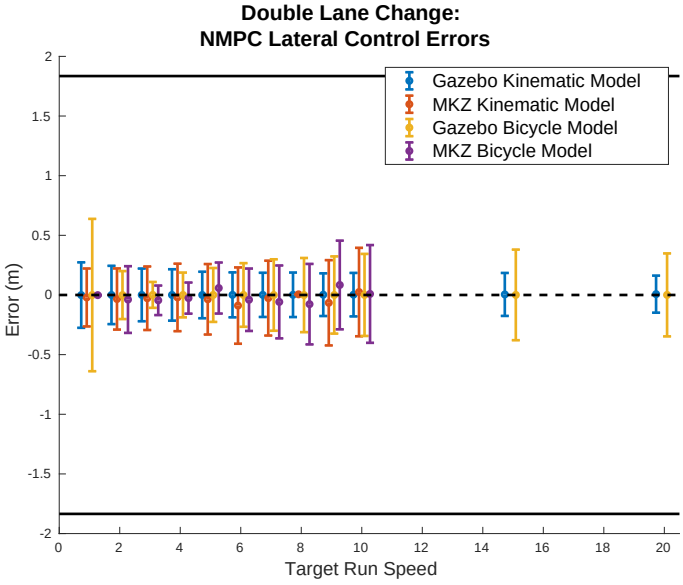


Figure 5.28: Double Lane Change Maneuver Lateral Path Error Mean and Standard Deviation with Increasing Path Speed

control error means for the region of interest in each test. The velocity control error mean and standard deviation results for each of the same tests discussed above are given in Figures 5.29–5.31. The simulated kinematic model NMPC again shows approximately zero mean error with

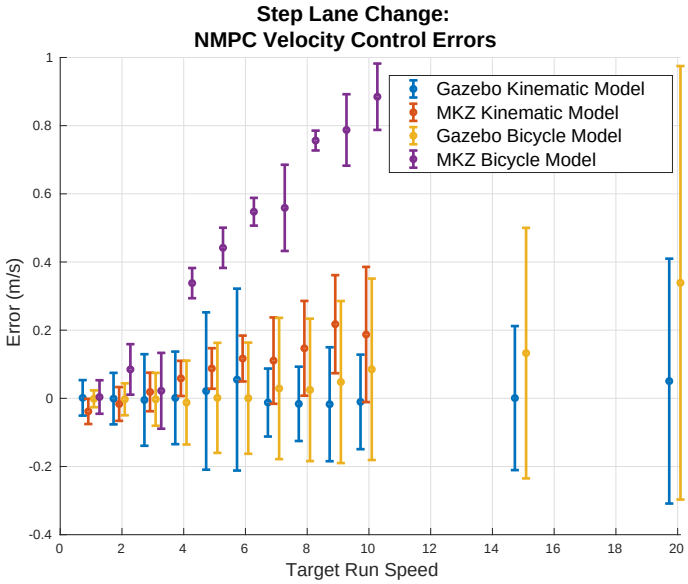


Figure 5.29: Step Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed

a standard deviation of $\pm 0.35 m/s$ in the worst case. The real-time kinematic model NMPC appears to develop significant error at steady-state that is contributing to the increase in mean velocity error with desired speed. Both the simulated and real-time kinematic model NMPC error standard deviations appear to grow with desired speed, but remain within a tolerance of $\pm 0.5 m/s$ that would be acceptable for many applications. The bicycle model NMPC's velocity tracking performance degrades with increasing run speed much more quickly than the kinematic model implementation.

Interestingly, there is a large discrepancy in the mean errors between the simulations and real-time tests of the bicycle model NMPC. While the real-time tests show that the mean error grows with run speed, the simulation results show a growth in oscillations around an approximately zero mean error. This result suggests that there are modeling errors in the simulation of the lower-level velocity control. The real vehicle likely filters much more of the high-frequency content from the longitudinal control signals than the simulation vehicle. The performance of

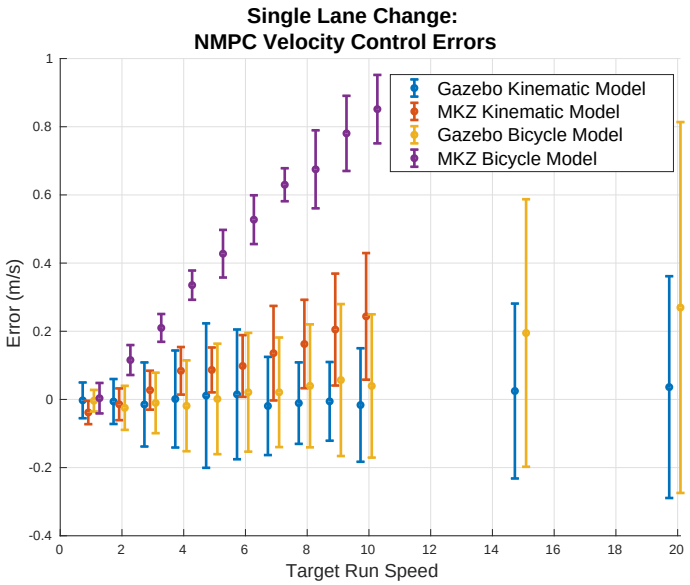


Figure 5.30: Single Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed

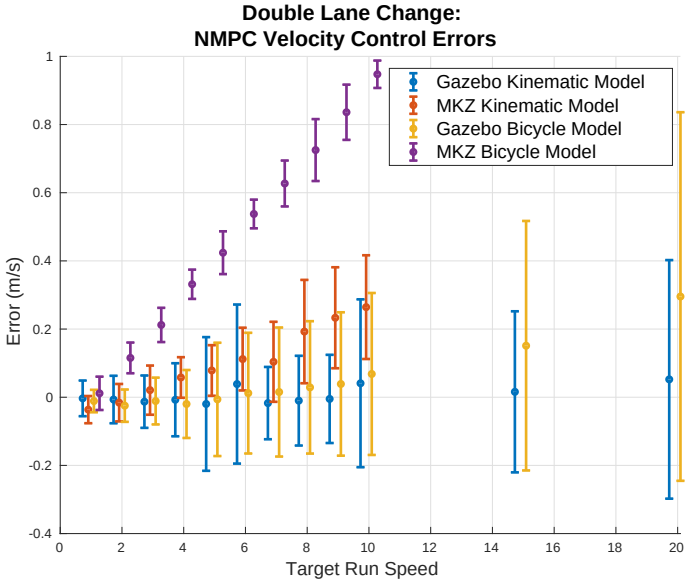


Figure 5.31: Double Lane Change Maneuver Velocity Control Error Mean and Standard Deviation with Increasing Path Speed

both implementations could possibly be improved by incorporating actuator models that include the delay from a throttle or brake petal input to the vehicle's velocity output.

Both NMPC implementations achieve similar path tracking performance with very different tunings. These differences have a couple important implications that should be taken into account when applying the NMPC in a larger system. As previously mentioned, the bicycle model implementation performs best when the tuning is optimized for each operating speed. This makes the bicycle model implementation significantly harder to tune than the kinematic model NMPC. A controller that is easier to tune and requires less detailed parameter identification may be desirable for applications that are used in a wide variety of vehicles. The kinematic model NMPC is also run with significantly longer prediction horizons and has less computational cost than the bicycle model NMPC. The update rates and fidelity of reference information and measurements used to update the NMPC impact the suitability of one implementation over the other. For example, the bicycle model implementation is well suited to applications with a detailed, high-update-rate reference while the kinematic model implementation may be better applied in a system that receives sparse, long-term reference information. These differences will be taken into consideration for the applications explored in the following sections.

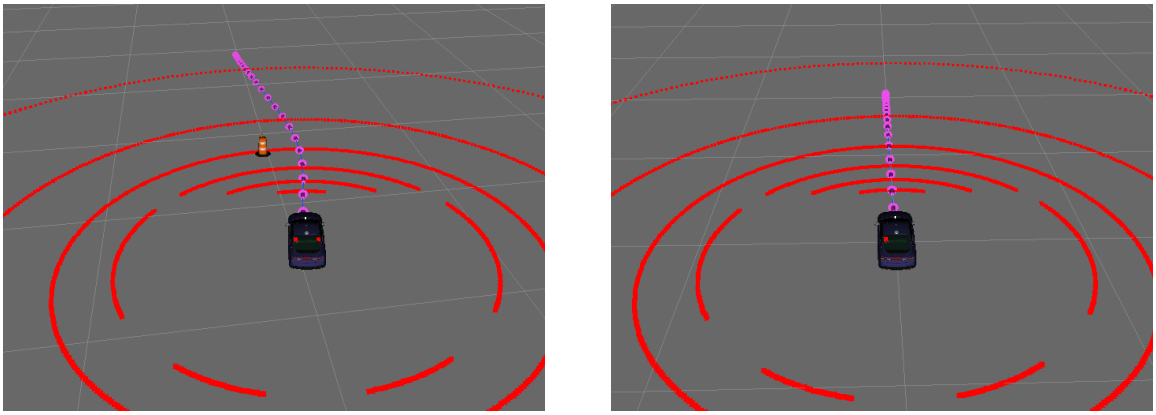
5.3 Obstacle Avoidance Application

One of the primary motivations for developing the presented NMPC implementations is the ability to anticipate and produce evasive maneuvers when an obstacle enters the vehicle's path and to set a hard constraint on collision avoidance. This section presents results from two simulation experiments that demonstrate the advantages and disadvantages of using the position-based obstacle constraint given in Equation (3.29). In the first experiment, the controller attempts to reach only a single position reference with a virtual obstacle is placed in its path. Additional simulation results without the obstacle are shown to observe the normal behavior of the controller when tracking to a single reference point. The second experiment uses the multiple way-point path references used in Section 5.2 and Section 5.4 to control around a pedestrian obstacle that steps directly into the desired path.

5.3.1 Simulation Procedures

Both obstacle avoidance experiments were run in the Gazebo simulation environment for a variety of reasons. The primary concern for all avoidance tests is safety. Until the controller's avoidance behavior is proven with perfect knowledge of obstacle positions, real-time experimental testing is impractical. The simulation environment also allows for exact and repeatable placement of a virtual obstacle. Because the tests in Section 5.2 show good correlation between simulation and real-time controller performance, both obstacle avoidance simulations were designed to understand the normal behavior of the controller when the avoidance constraints are active and to explore any weaknesses in the proposed constraint method.

The simulation environment for the first experiment is a completely flat and empty world. A single reference position and orientation, or target, can be arbitrarily selected for the controller. A tool for placing a virtual obstacle into the simulation environment was created to publish an arbitrary obstacle position for the controller to react to. An RVIZ visualization of the empty simulation environment with and without an obstacle is shown in Figure 5.32. In the



(a) Obstacle Placed in Vehicle Path (b) No Obstacle in Path, Tracking to a Single Target

Figure 5.32: RVIZ Visualization of an Empty Simulation World for Single Target Tracking and Obstacle Avoidance Control

simplest test, a single reference point was placed directly in front of the vehicle (with no obstacle in the path) to verify that the controller takes the obvious straight line path to the target and stops. A second test was conducted with a single reference point that was not directly ahead of the vehicle and again no obstacle was placed in the path. This test shows the normal steering behavior of the controller when tracking to a single reference position. The final test procedure

for this experiment places the reference position directly ahead of the vehicle and an obstacle directly in the middle of the straight line path from the vehicle's starting position to the desired reference position. The control inputs and controlled vehicle positions from these tests will be discussed in the next section.

The second experiment utilizes the straight road simulation environment shown in Figure 5.6. In these experiments a multiple way-point reference path is generated at a high rate (100 Hz) and published to the NMPC as the control reference. This path attempts to keep the vehicle in the right lane and at a constant forward velocity. The path is not altered based on any obstacle positions. A simulated pedestrian is then placed in the environment with the ability to walk into and out of the road. The pedestrian begins each test by standing on the road shoulder and eventually walks directly into the center of the right lane, intersecting with the vehicle's desired path. This pedestrian obstacle publishes its position as an obstacle via a Gazebo plugin, simulating the feedback that could come from an obstacle tracking module in the system. The simulation setup for this experiment is shown below in Figure 5.33. Tests were conducted with

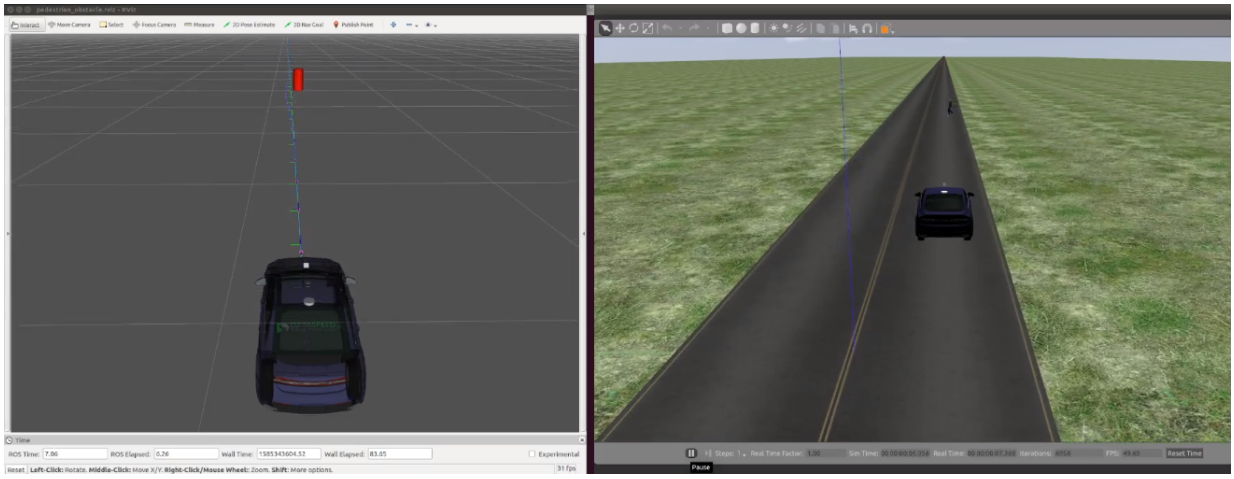
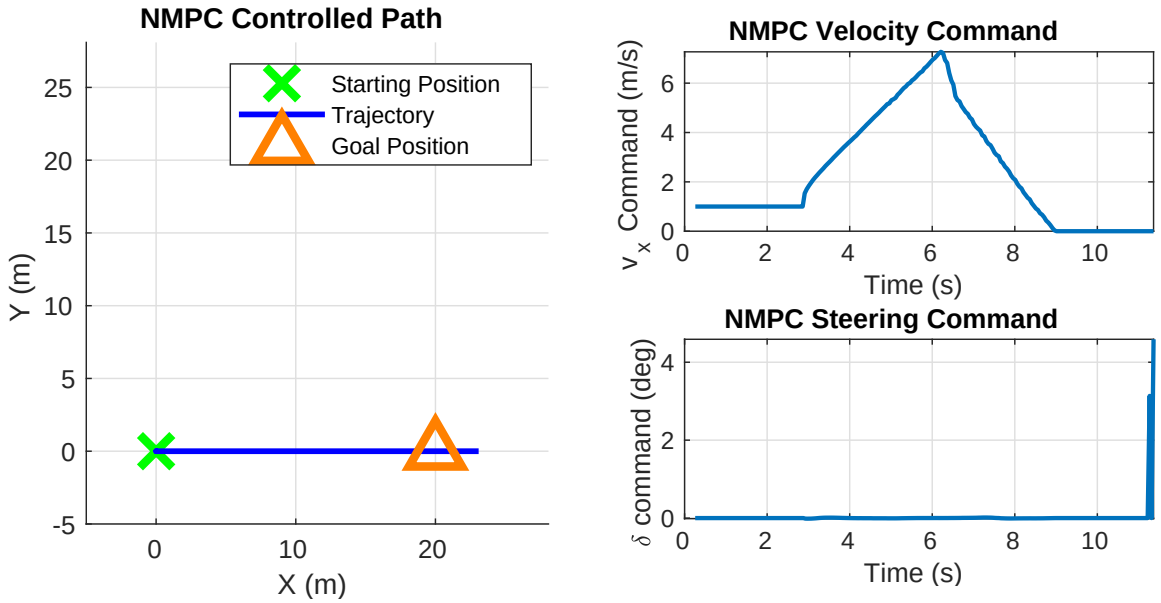


Figure 5.33: RVIZ Visualization of the Published Path and Obstacle Position (left) and the Gazebo Simulation World with a Walking Pedestrian Model Acting as an Obstacle (right)

both the kinematic and dynamic bicycle model-based NMPC implementations at 1, 5, and 10 m/s.

5.3.2 Obstacle Avoidance Results

For each test in the first experiment, the controlled vehicle trajectory and the control inputs generated by the NMPC are plotted and evaluated. The results from the first simple test, a single target placed directly ahead of the vehicle and no obstacle in the path, are shown in Figure 5.34. The vehicle drives a straight line trajectory exactly as expected. Because there is only a single



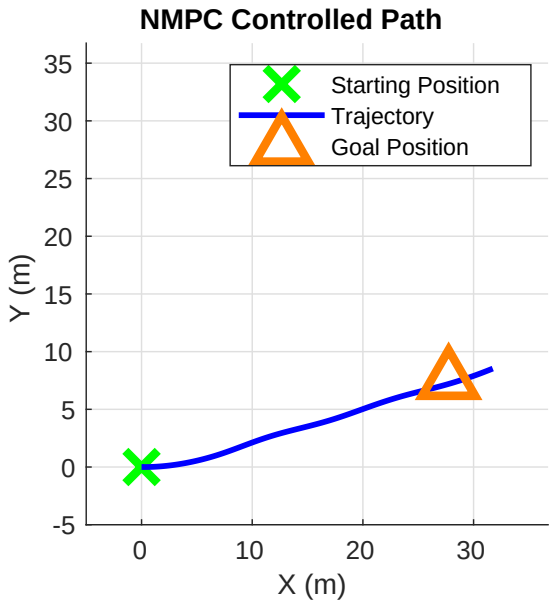
(a) Vehicle Trajectory

(b) NMPC Controller Output

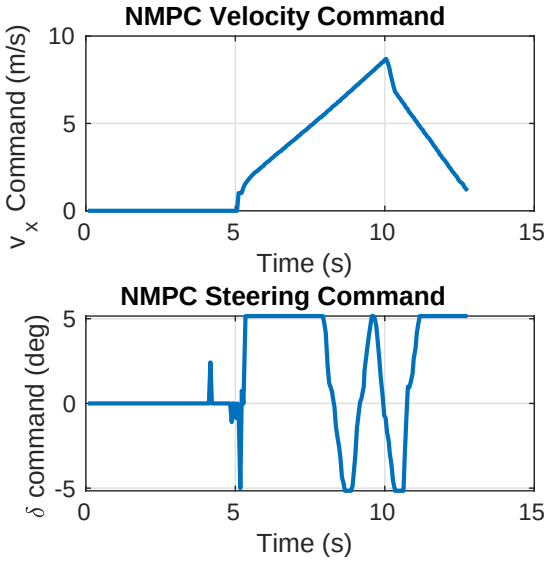
Figure 5.34: Single Target Avoidance Test: Target Placed Directly Ahead of the Vehicle and No Obstacle in the Path

reference point, the optimizer attempts to reach it as quickly as possible by linearly ramping up the velocity command and then decelerating quickly as the vehicle reaches the target. The vehicle slightly overshoots the desired position because the braking dynamics are not captured in the simple kinematic controller model. The steering command for this test remains straight, as expected, until the vehicle overshoots the target.

The results for the second test, a reference placed offset from the vehicle's initial heading, are shown in Figure 5.35. The desired velocity profile is similar to the previous test as it ramps up towards its upper bound (set at 10 m/s in these tests) and decelerates hard to stop at the target position. This test shows an unexpected low-frequency oscillation in the steering control output as the vehicle commands a velocity greater than 5 m/s. Although the vehicle ultimately reaches its target position, the steering profile generated from the controller is undesirable.



(a) Vehicle Trajectory

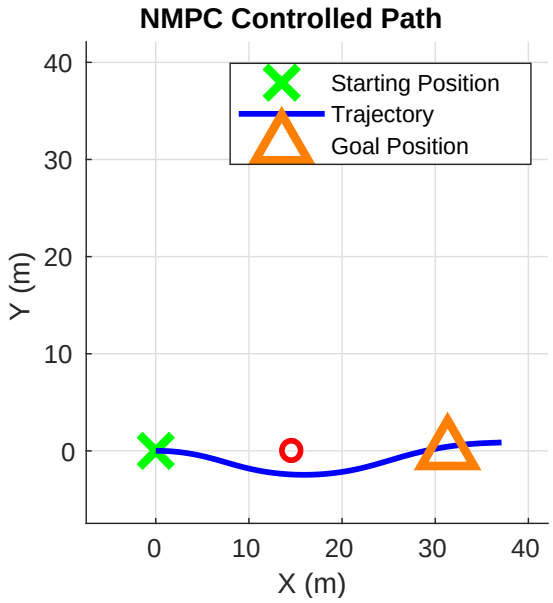


(b) NMPC Controller Output

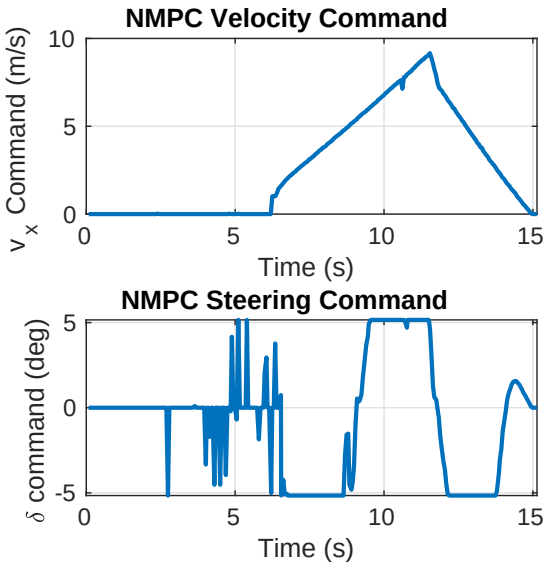
Figure 5.35: Single Target Avoidance Test: Target Placed Offset From the Vehicle's Initial Heading and No Obstacle in the Path

The results of the final test in the first experiment, a single target placed directly ahead of the vehicle with an obstacle placed directly in the center of the path, are shown in Figure 5.36. In this test the vehicle successfully generates a trajectory that avoids the obstacle and only deviates slightly from the expected straight line path. The velocity and steering profile are similar to the previous test. To achieve the same control performance shown in Section 5.2 while avoiding obstacles, the avoidance constraints must be tested with a multiple way-point reference path.

In the second simulation experiment, the controlled vehicle trajectory and control inputs from the NMPC controller are also plotted for each test. The results for the kinematic model at 1, 5, and 10 m/s are shown in Figure 5.37, Figure 5.38, and Figure 5.39 respectively. In each test, the kinematic model NMPC fails to avoid the obstacle. In the low-speed test, with a desired path speed of 1 m/s, the vehicle reaches its target speed before reaching the obstacle and begins to steer around the obstacle but stops directly in collision with it. In the 5 and 10 m/s tests, the controller attempts to stop when it reaches the obstacle. Because the vehicle's longitudinal dynamics are not modeled in the controller, the vehicle does not stop immediately when the commanded velocity is zero, and its momentum carries the vehicle through the collision with

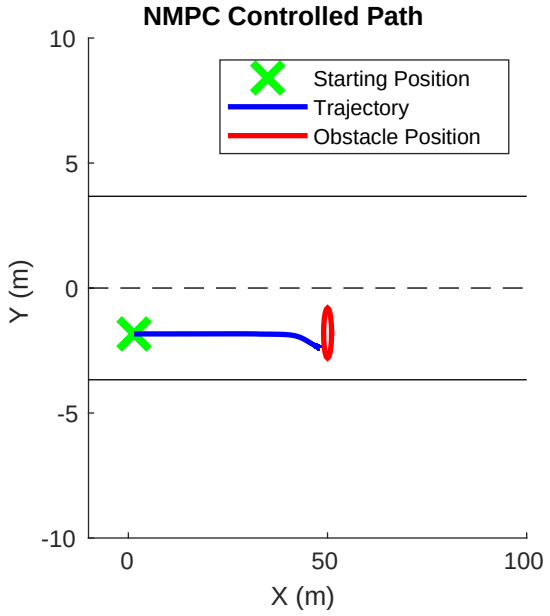


(a) Vehicle Trajectory

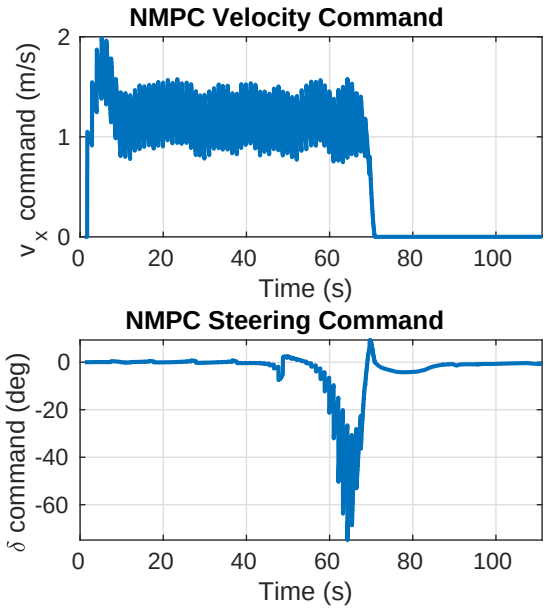


(b) NMPC Controller Output

Figure 5.36: Single Target Avoidance Test: Target Placed Directly Ahead of the Vehicle with an Obstacle in the Path

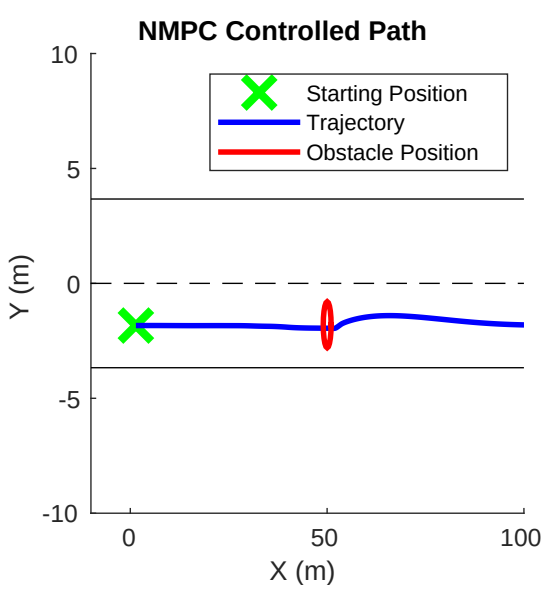


(a) Vehicle Trajectory

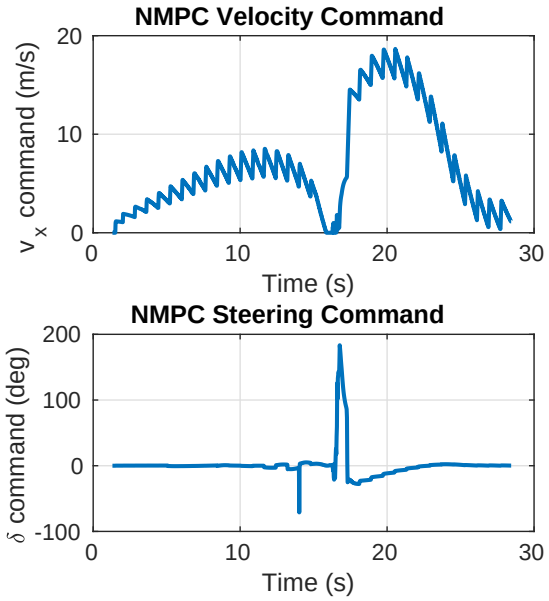


(b) NMPC Controller Output

Figure 5.37: Kinematic Model NMPC Pedestrian Obstacle Avoidance at 1 m/s

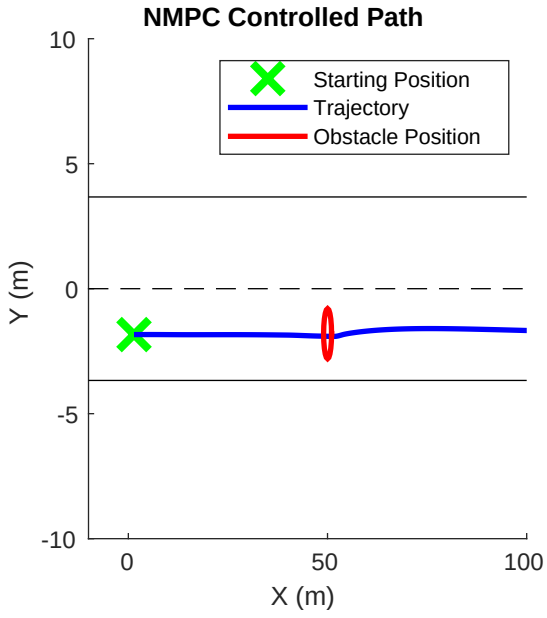


(a) Vehicle Trajectory

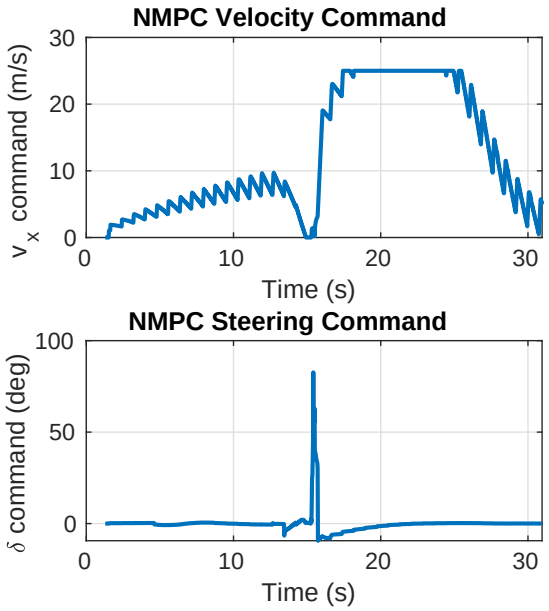


(b) NMPC Controller Output

Figure 5.38: Kinematic Model NMPC Pedestrian Obstacle Avoidance at 5 m/s



(a) Vehicle Trajectory

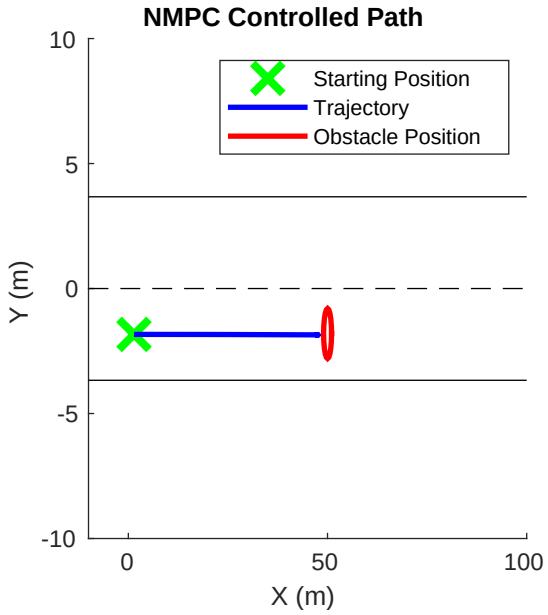


(b) NMPC Controller Output

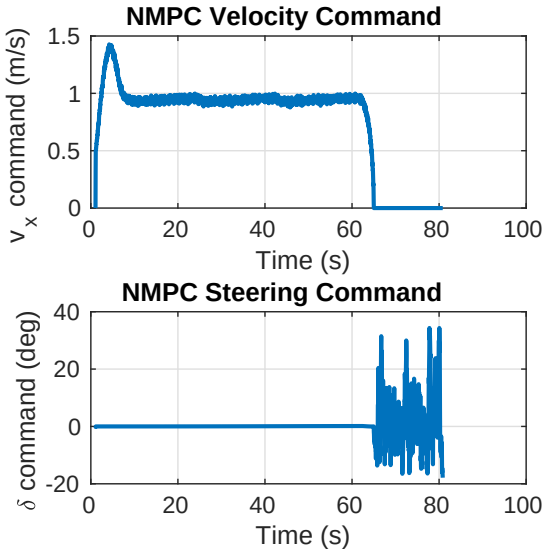
Figure 5.39: Kinematic Model NMPC Pedestrian Obstacle Avoidance at 10 m/s

the obstacle. The controller then ramps up the desired velocity again to catch up with the desired path. Note that the obstacle model in the simulation environment did not have any physical collision properties and therefore it did not impede or stop the vehicle in any way.

The results for the bicycle model NMPC implementation at 1, 5, and 10 m/s are shown in Figure 5.40, Figure 5.41, and Figure 5.42 respectively. Similar to the kinematic model



(a) Vehicle Trajectory

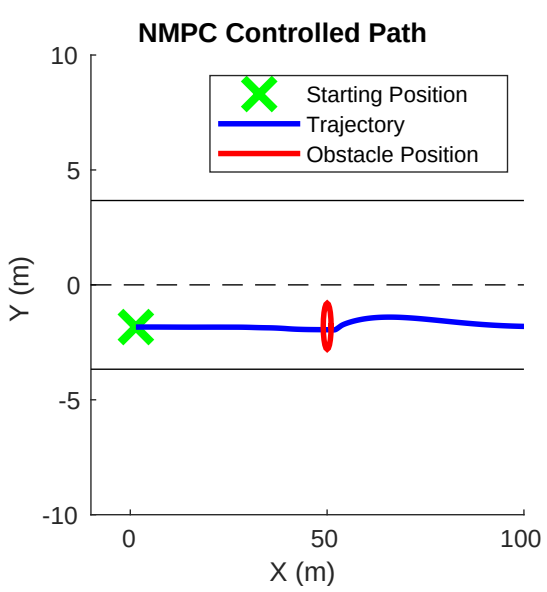


(b) NMPC Controller Output

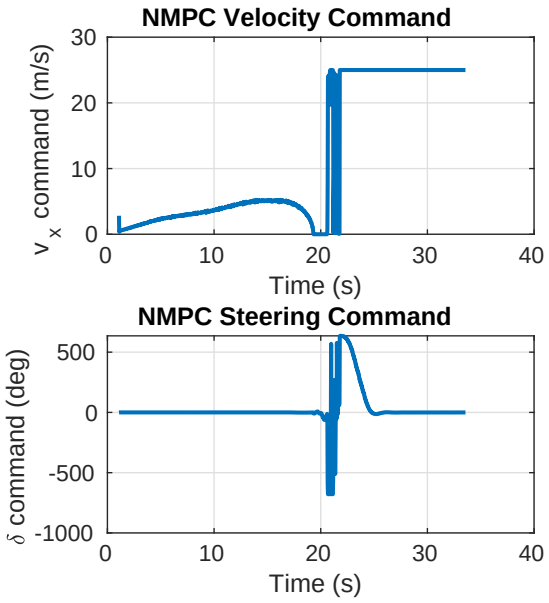
Figure 5.40: Bicycle Model NMPC Pedestrian Obstacle Avoidance at 1 m/s

NMPC implementation, the bicycle-model-based controller failed to avoid the obstacle in each of these tests. The controller stops the vehicle in collision with the obstacle at low speed and at higher speeds runs directly through the collision and continues back to the desired path. In each test, when the vehicle is in collision with the obstacle, the optimizer in the controller fails to converge and produces erratic steering commands. This behavior is easily seen in Figure 5.40b after approximately 62 seconds. The higher-speed tests shown in Figure 5.41b and Figure 5.42b demonstrate that the controller operates normally after the vehicle is clear of the obstacle's collision radius.

To further analyze this failure mode that is present in both NMPC implementations, the prediction horizons before, during, and after the obstacle intersects with the desired reference path must be evaluated. During the pedestrian avoidance simulations, there are short periods of

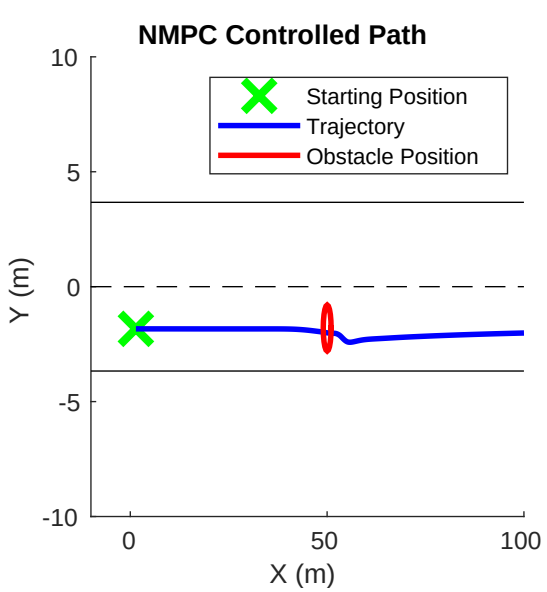


(a) Vehicle Trajectory

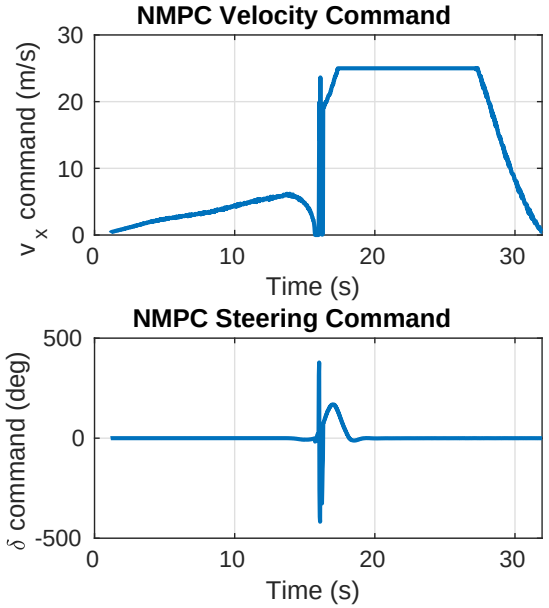


(b) NMPC Controller Output

Figure 5.41: Bicycle Model NMPC Pedestrian Obstacle Avoidance at 5 m/s



(a) Vehicle Trajectory



(b) NMPC Controller Output

Figure 5.42: Bicycle Model NMPC Pedestrian Obstacle Avoidance at 10 m/s

time in which the obstacle is moving into the controller's reference path, and in some simulations there is some time after the vehicle has passed the obstacle and can no longer see it on the horizon. The positions of the prediction horizon and the Lagrange multipliers associated with the collision constraints are plotted at multiple instants during these transition points to give some indication of where the failure mode originates and how it can be detected. Select plots for the 10 m/s test of the kinematic model NMPC implementation and the dynamic bicycle model NMPC implementation are shown in Figure 5.43 and Figure 5.44, respectively.

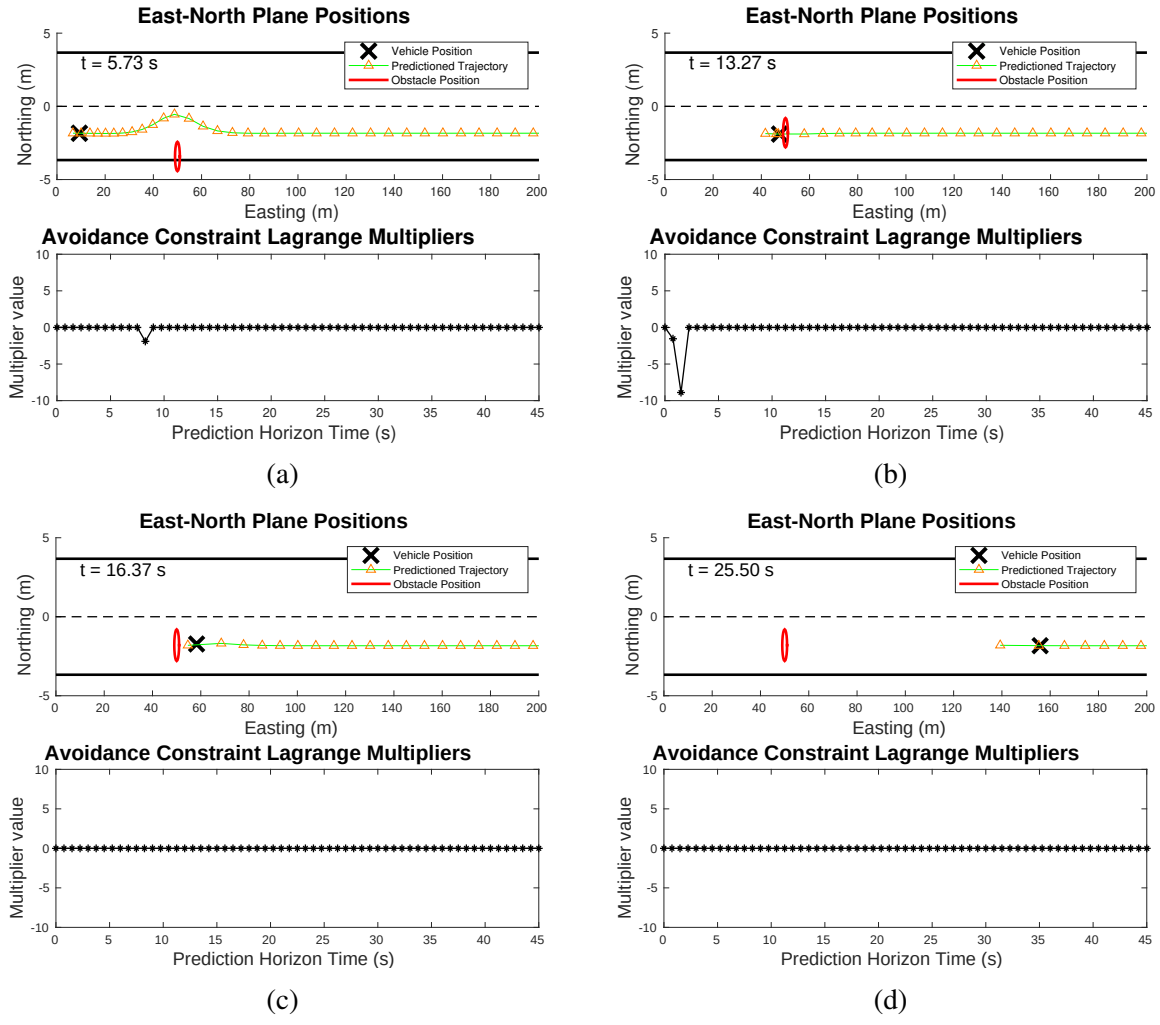


Figure 5.43: Prediction Horizon and Avoidance Constraint Lagrange Multipliers at Select Instants of the 10 m/s Pedestrian Avoidance Test Using the Kinematic Model NMPC

Interestingly, the prediction horizon in Figure 5.43a shows the expected evasive maneuver as the obstacle is moving into the desired path. There is also a spike in the constraint Lagrange multiplier values between 5 and 10 seconds on the prediction horizon. A similar spike in the

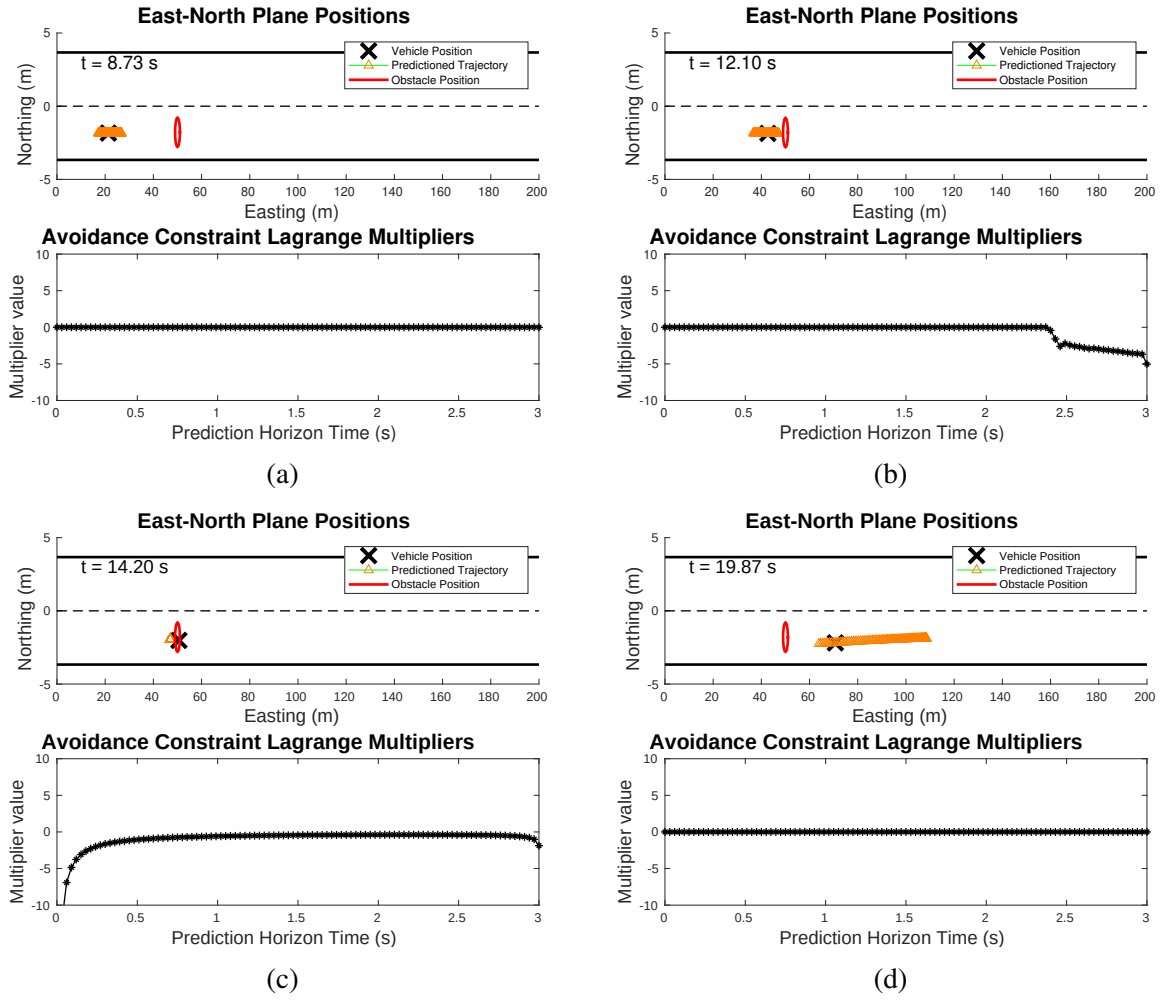


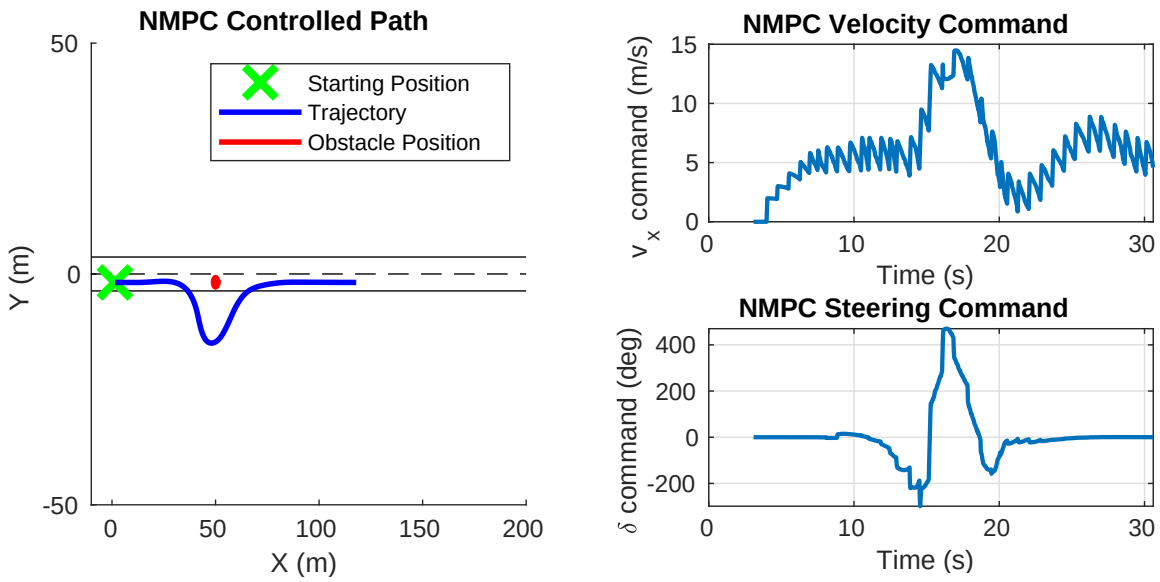
Figure 5.44: Prediction Horizon and Avoidance Constraint Lagrange Multipliers at Select Instants of the 10 m/s Pedestrian Avoidance Test Using the Bicycle Model NMPC

Lagrange multiplier values is shown in Figure 5.43b, much closer to $T = 0$ on the horizon as the vehicle's CG is almost in collision with the obstacle. After the vehicle has cleared the obstacle, the Lagrange multiplier values return to approximately zero. This detectable spike in Lagrange multiplier value along the horizon gives a strong indication of where along the prediction horizon the collision avoidance constraint is active.

The Lagrange multiplier values show similar behavior in the test of the bicycle model NMPC implementation. Figure 5.44b and Figure 5.44c show a significant spike in the multiplier values as the closely spaced predictions collide with the obstacle. Because the prediction horizon length for this controller implementation is tuned to be significantly shorter (3 s), the controller cannot "see" the obstacle until it is directly in the desired path. An evasive maneuver trajectory is never generated on the prediction horizon, but the failure mode appears to be the same in both controller implementations: reference states that directly violate the collision constraint cause the optimizer to generate an invalid solution.

There are a couple of possible solutions to correct the failure of this proposed collision constraint method in a way-point reference path-following scenario. The simplest solution is to pre-process the reference path before each control iteration. This pre-processing step would check each reference way-point in the path for a collision constraint violation and move problem way-points to the constraint boundary before each call to the optimization routine in the controller. This solution method would require an algorithm for *where* to move problem way-points along the boundary.

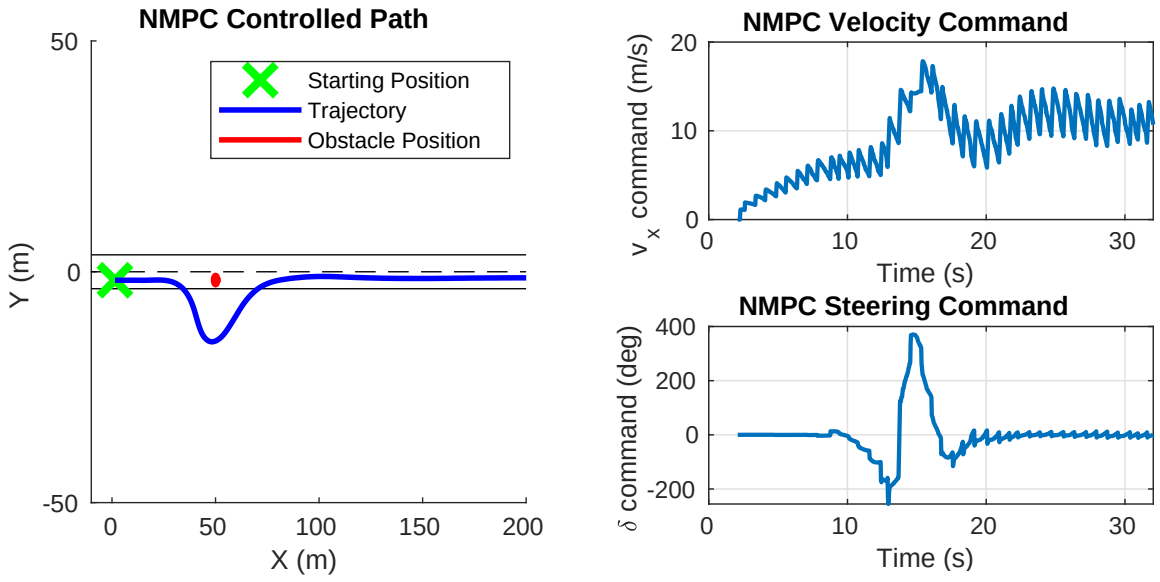
To demonstrate this simple solution method and any possible side-effects, path pre-processing was added to the NMPC controller. Any way-point that was found to be in collision with a known obstacle position was simply moved to the constraint boundary by projecting the way-point both ahead and behind the obstacle along the distance vector to the obstacle position and selecting the modified way-point that is closest to the original path. The same pedestrian avoidance scenarios were re-run in simulation with both the kinematic and bicycle model NMPC implementations at 5 m/s and 10 m/s. The results from the two simulations with the kinematic model implementation are shown in Figure 5.45 and Figure 5.46. Similarly, the results



(a) Vehicle Trajectory

(b) NMPC Controller Output

Figure 5.45: Kinematic Model NMPC Pedestrian Obstacle Avoidance at 5 m/s with Path Pre-processing

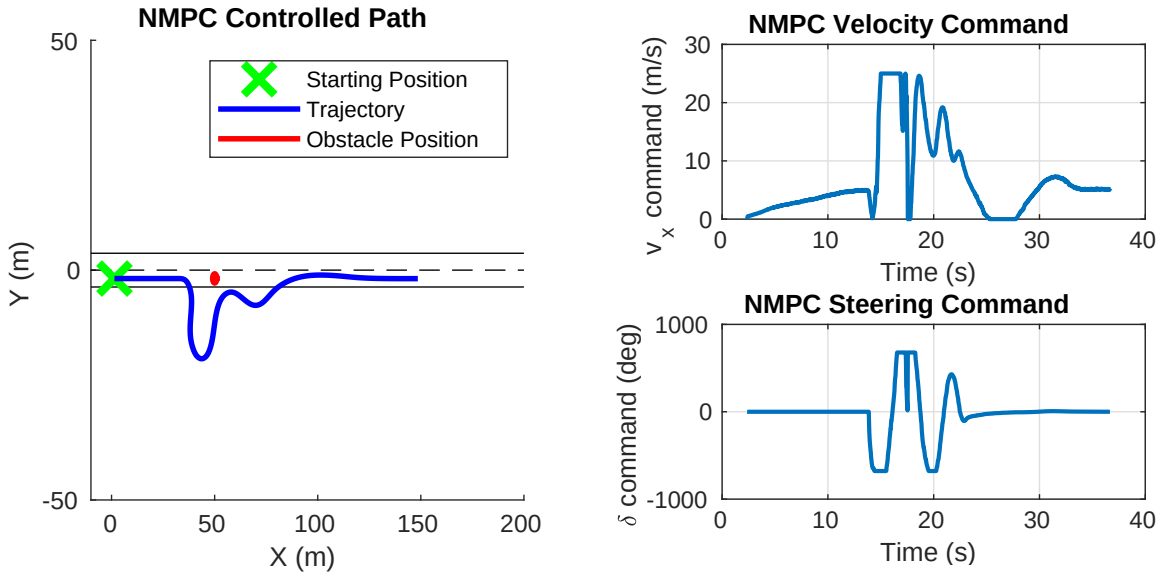


(a) Vehicle Trajectory

(b) NMPC Controller Output

Figure 5.46: Kinematic Model NMPC Pedestrian Obstacle Avoidance at 10 m/s with Path Pre-processing

from the two simulations with the bicycle model implementation are shown in Figure 5.47 and Figure 5.48.



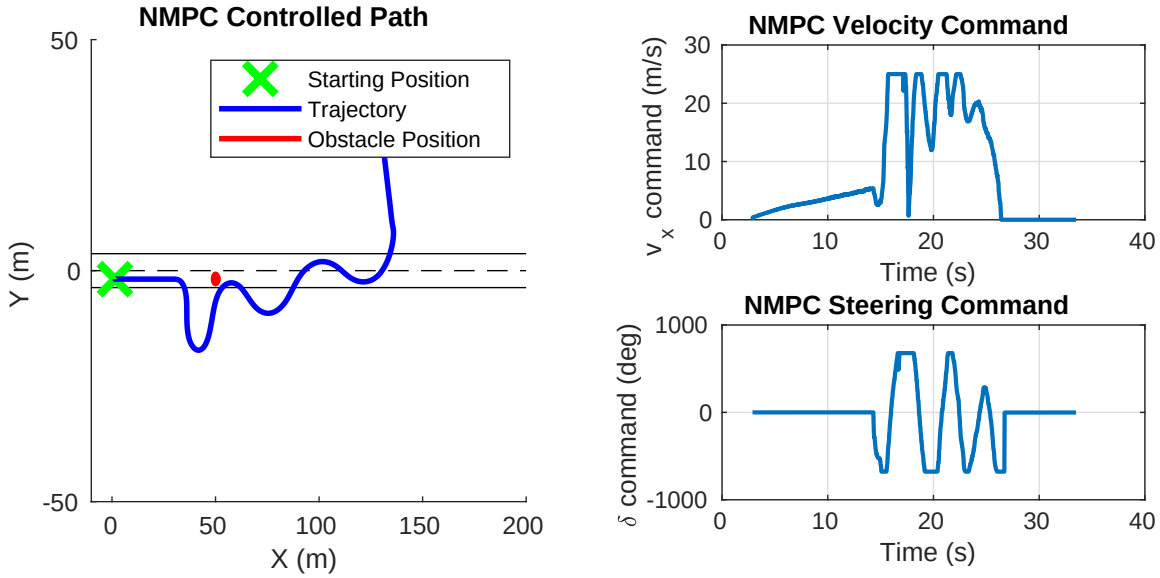
(a) Vehicle Trajectory

(b) NMPC Controller Output

Figure 5.47: Bicycle Model NMPC Pedestrian Obstacle Avoidance at 5 m/s with Path Pre-processing

Note that there is a major trade-off between the path-tracking robustness and avoidance that is made when this pre-processing modification is added. Even in the lower speed 5 m/s simulations, the trajectory that is generated deviates significantly from the original path (multiple road lanes). The control actions generated when the obstacle comes into view is much more aggressive than during the normal path tracking operation in all cases. In the 10 m/s case with the bicycle model NMPC implementation, the erratic steering output hits its limits and ultimately results in unstable behavior. These results suggest that additional solutions should be explored to better understand and handle the avoidance control mode.

A second solution might be to use the Lagrange multiplier values as feedback to indicate to the controller which states along the prediction horizon are potentially in collision. This feedback information could be used to de-weight the reference path errors in the NMPC's cost function at these collision points on the horizon. This solution would allow the optimizer to generate a continuous trajectory around the remaining valid way-points in the reference path. There are other possible solution methods that extend beyond the capabilities of the current constraint implementation.



(a) Vehicle Trajectory

(b) NMPC Controller Output

Figure 5.48: Bicycle Model NMPC Pedestrian Obstacle Avoidance at 10 m/s with Path Pre-processing

Changing the form of the avoidance constraint equations may be necessary to get the best path tracking performance during obstacle avoidance. Note that the current avoidance constraints are discrete in nature. Although each point along the horizon is constrained to be outside the collision radius, there is no guarantee that the continuous trajectory between two successive collision free predictions does not cross the collision boundary. This is especially apparent in the un-modified pedestrian avoidance simulation case with the kinematic model NMPC implementation. In Figure 5.43b the second and third predicted positions are just ahead of the obstacle and then directly behind it, respectfully. The straight-line trajectory between those two points goes directly through the collision with the obstacle. Using the current state prediction models, there may be a constraint equation that can describe and prevent collision between the circular obstacle boundary and the line segment of two sequential predictions. These solutions and others need to be verified extensively in simulation before this obstacle avoidance method can be applied to a way-point reference following application.

5.4 Non-line-of-sight Path Following Application

As previously mentioned, a possible application of the NMPC controller is in Auburn University's platooning system that is capable of doing non-line-of-sight path following. The following sections will discuss the experimental test setup that was modified from the original heavy truck platforms to the available passenger vehicle test platforms. The experiments for real-time testing of the controller in the following scenarios will be described. Finally, the results of these experiments will be shown to prove the feasibility of NMPC control for use in platooning or following applications.

5.4.1 Experiment Procedures

Two experiments were performed to test the controller. The first experiment was a simple circular maneuver in which a leader vehicle drove a wide circle around the NCAT skid pad area and the controlled vehicle followed approximately 20 seconds behind. In the second experiment the leader vehicle drove a more complex looped route that turned into and out of the NCAT skid pad area via a small access road. In the second experiment, the leader vehicle got much further ahead of the controlled follower vehicle and lost line-of-sight through the turns onto the access road. Both of these experiments were run only in real-time with the MKZ test vehicle. The current Gazebo simulation environment is not capable of simulating the DRTK/TDCP algorithms that are used to create the reference path for the follower.

In both experiment procedures, the MKZ test vehicle acted as the follower that was running the NMPC. Another passenger vehicle was used as the manually driven leader vehicle. Both vehicles in a leader–follower configuration are shown below in Figure 5.49. DSRC radio antennas and GPS antennas can be seen atop both test vehicles. The leader vehicle contained only a mobile computer, GPS receiver, and DSRC radio as shown in Figure 5.50. The MKZ follower vehicle did not require any hardware beyond what was shown previously in Figure 5.4.

Only the kinematic model NMPC was used in these experiments. This choice was primarily due to the low frequency of path updates (2 Hz) from the lead vehicle, which matched

Figure 5.49: MKZ Test Vehicle Setup as an Autonomous Follower to the Manually Driven Kia Optima Leader Vehicle

Figure 5.50: Kia Optima Leader Vehicle Hardware Setup

an acceptable prediction time step for the kinematic model NMPC. The number of path points published from the path generation software module could also vary at any given path update. While the NMPC software module is set up to handle paths of varying sizes, the bicycle model NMPC implementation is not robust enough to handle a sporadic and undersampled path. To use the bicycle model implementation, the path would need to be upsampled or resampled; however, the kinematic model implementation is robust enough to be used without any pre-processing of the path.

5.4.2 Non-line-of-sight following results

Results from the first experiment, following the leader vehicle driving in a circular path, are shown below in Figure 5.51. The solid path line in Figure 5.51 is the GPS/INS position

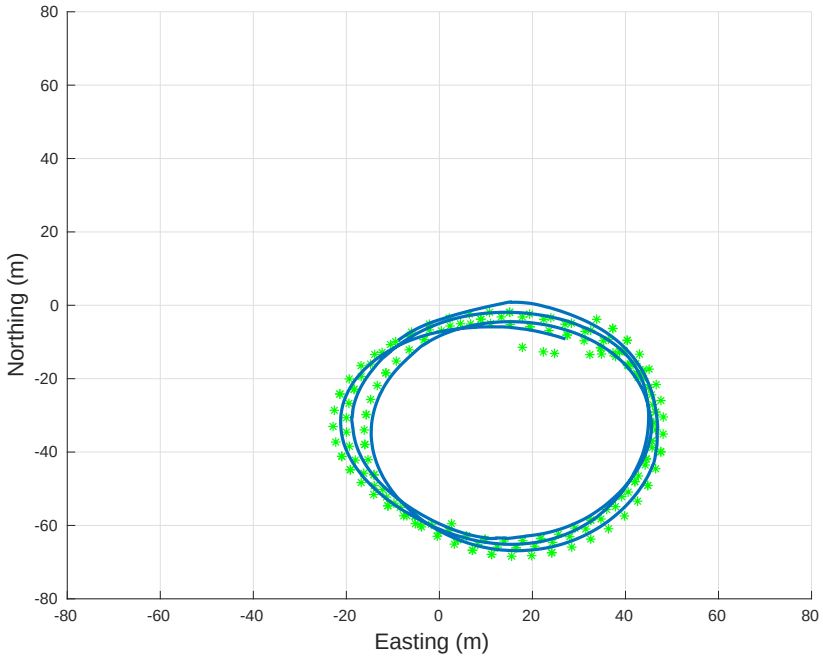


Figure 5.51: Kinematic Model NMPC Following a Circular Path Generated from DRTK/TDCP

solution for the follower vehicle. The leader vehicle's positions from DRTK are represented as green stars. The vehicle successfully follows the leader's path for approximately 3 laps before control was manually overridden by the driver. During this test the following vehicle did not ever lose sight of the leader.

The first experiment did not exercise the non-line-of-sight feature of the DRTK/TDCP path generation method. To test it, the second experiment was designed to include a significantly longer following distance. The results of following this longer looped path are shown below in Figure 5.52. In this path, the leader vehicle began by making a large sweeping turn around the

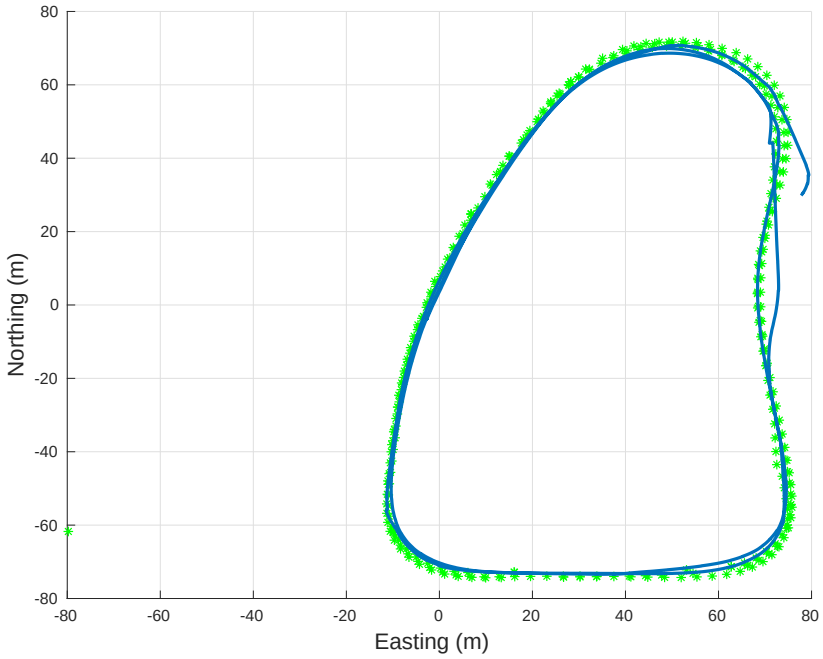


Figure 5.52: Kinematic Model NMPC Non-line-of-sight Following a Leader Vehicle

NCAT skid pad area. The leader proceeded straight down to a right turn onto a small access road, traveled straight down this road, and made another right turn back up towards the skid pad area. During this portion of the path the follower vehicle completely lost sight of the leader. The controller successfully followed the path during each part of this experiment; however, a few interesting parts of the path should be highlighted.

Select instants in time during the non-line-of-sight path following are shown in Figure 5.53. During the sweeping turn, the model predictions match almost exactly with the reference path, resulting in good path tracking. After a sharper turn, the vehicle's true position overshot the predicted trajectory and had to correct back to the desired path. This overshoot is likely due to the model inaccuracies of the low-fidelity kinematic model used in the controller. During both right turns, onto and off of the access road, the controller cuts corners in the path. This problem may have been less apparent if this sharp turn had contained more way-points in the controller's prediction horizon, because the NMPC's cost function would be dominated by the

path errors in the turn. Possible solutions to these edge cases and other general improvement suggestions will be discussed further in the following chapter.

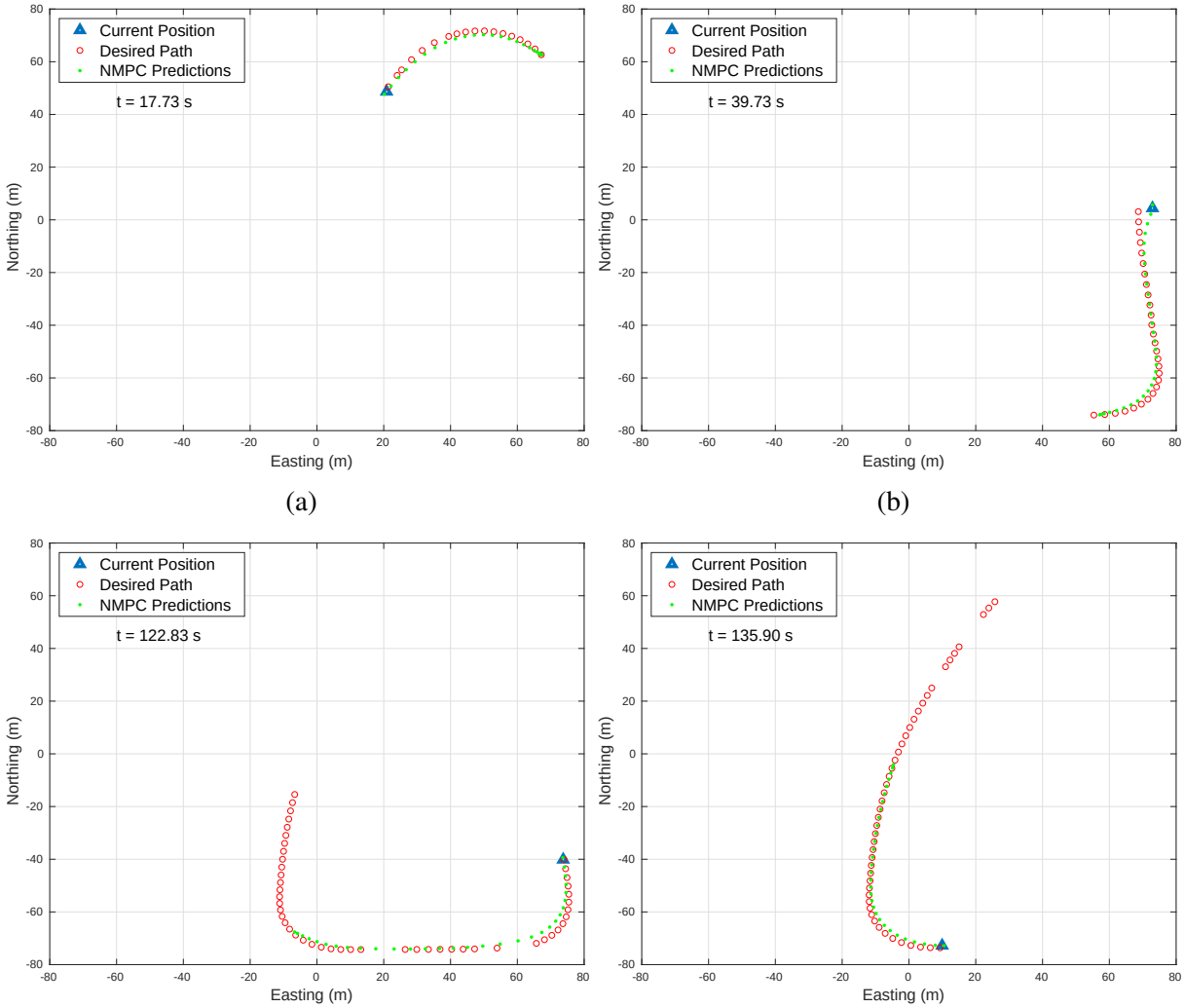


Figure 5.53: Sample Control Iterations During Non-line-of-sight Following

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The safety of automated driving systems is paramount to their progression from generally capable SAE level 2 systems to fully featured level 3 and level 4 systems that do not require a human in the control loop. This thesis has presented an NMPC approach to add obstacle avoidance capabilities to a current SAE level 2 automated driving system and future automated ground vehicles. The model predictive control algorithm, including the formulation of the cost function, constraints, and underlying optimization algorithms, were detailed in this thesis. A flexible NMPC software library was developed and proven to be feasible for real-time control of an autonomous vehicle platform. Two simple vehicle models, a kinematic model and a dynamic bicycle model, were developed, identified, and implemented in NMPC controllers for a drive-by-wire capable Lincoln MKZ. These controllers were demonstrated with a series of simulation and real-time experiments.

Both controller implementations were shown to be capable of tracking a path of way-points when the path is used as the control reference. A set of highly accurate global position-based paths with 1 cm resolution were used to tune and test each controller's performance in autonomous following applications. The detailed tuning procedure revealed that the tuning of the prediction horizon parameters is equally important to the stability of the controller as the tuning of the control effort weights in the cost function. When using a static tuning, the control performance varies with the vehicle's desired speed; however, tuning values for each controller implementation were found and proven to work at speeds from 1 to 20 m/s. The tuned kinematic model-based NMPC implementation was also used in Auburn's non-line-of-sight following system to successfully duplicate a leader vehicle's path autonomously. This

application proved that the NMPC controller can be integrated into an existing level 2 platooning software stack to replace the original lateral controller module.

Obstacle avoidance was integrated into the controller through a set of hard constraints. Simulation testing proved these constraints work when the controller drove to a target position that was not in collision with an obstacle. The obstacle constraints failed in the reference path following tests when one or multiple of the reference path way-points were in collision with the obstacle. One proposed solution to this problem is to pre-filter the path for collisions between reference positions and obstacle positions. A second proposed solution is an adaptive weighting scheme that de-weights the tracking errors in the cost function for only the states on the horizon that are in collision with the obstacle. These improvements must be incorperated into the controller before it can be tested with the added complexity and uncertainty of a real-time obstacle detection and tracking system.

6.2 Future Work

In addition to the proposed improvements in the obstacle avoidance control, there are many other aspects of the controller that can be improved for faster, higher performance, and more robust control. These improvements begin with continued improvements to the software implementation. Currently, the controller uses CasADi's interal interface to the IPOPT solver. This internal interface evaluates the optimization equations with their symbolic math representations. These function evaluations are far slower than evaluating the problem definition Jacobians and Hessians once, generating C/C++ function code for the function evaluations the optimization library requires, and compiling these C/C++ representations. CasADi offers code generation tools to create these compiled functions for almost all of its solver interfaces. These code generation tools should be explored to increase the computational efficiency of the controller, allowing longer prediction horizons and faster control loops.

Control performance, especially the longitudinal control performance, can be improved by including dynamic models of the vehicle's actuators. For the longitudinal control, the model from throttle and brake pedal inputs to velocity output could significantly reduce overshoot and control chatter. Similarly, the delay from steering command to actual steering output could

be modeled and included in the controller for a more accurate prediction of the lateral motion of the vehicle over the horizon. Many other more complex models could also be explored, including articulated tractor-trailer models for control of large trucks.

Many other weigthing schemes and forms of the optimization cost function should also be explored. One popular addition to the cost function is a term that penalizes changes in control input over the horizon. This is similar to adding derivative control in a classical PID controller. This additional term adds another tuning matrix that can be used to further tune the control performance. Although the current NMPC has quite a few tuning parameters, they could all be adjusted to different values along the horizon or adjusted online based on the desired maneuver on the horizon. A self-tuning or adaptive tuning controller could possibly be accomplished through including machine learning into the NMPC.

References

[1] C. J. Kahane, "Lives saved by vehicle safety technologies and associated federal motor vehicle safety standards, 1960 to 2012 – passenger cars and LTVs," *Report No. DOT HS 812 069*, 2015.

[2] J. N. Dang, "Statistical analysis of the effectiveness of electronic stability control (ESC) systems," *Report No. DOT HS 810 794*, July 2007.

[3] National Center for Statistics and Analysis, "Speeding: 2017 data (Traffic Safety Facts. DOT HS 812 687)," tech. rep., National Highway Traffic Safety Administration, Washington, DC, May 2019.

[4] SAE International Surface Vehicle Recommended Practice, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," *SAE Standard J3016, Rev. Jun. 2018*.

[5] Crash Avoidance Metrics Partnership on behalf of the Automated Vehicle Research Consortium, "Automated vehicle research for enhanced safety final report," tech. rep., National Highway Traffic Safety Administration, Mar. 2016.

[6] Office of Highway Safety, "Preliminary report highway: HWY18MH010," tech. rep., National Transportation Safety Board, 490 L'Enfant Plaza East, S.W. Washington, DC 20594, Mar. 2018.

[7] Tesla, Inc., "A tragic loss." `https://www.tesla.com/blog/tragic-loss`, June 2016. Accessed: 2019-10-08.

[8] V. V. Dixit, S. Chand, and D. J. Nair, "Autonomous vehicles: Disengagements, accidents and reaction times," *PLOS ONE*, vol. 11, pp. 1–14, 12 2016.

[9] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on Intelligent Vehicles*, vol. 1, 04 2016.

[10] C. Katrakazas, M. Quddus, W.-H. Chen, and L. Deka, "Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions," *Transportation Research Part C: Emerging Technologies*, vol. 60, pp. 416 – 442, 2015.

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

[12] T. Gindele, D. Jagszent, B. Pitzer, and R. Dillmann, "Design of the planner of team AnnieWAY's autonomous vehicle used in the DARPA urban challenge 2007," in *2008 IEEE Intelligent Vehicles Symposium*, pp. 1131–1136, June 2008.

[13] R. Geisberger, P. Sanders, D. Schultes, and C. Vetter, "Exact routing in large road networks using contraction hierarchies," *Transportation Science*, vol. 46, pp. 388–404, Aug. 2012.

[14] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck, "Route planning in transportation networks," in *Algorithm engineering*, pp. 19–80, Springer, 2016.

[15] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 421–427, Oct 1979.

[16] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, Dec 1959.

[17] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, pp. 100–107, July 1968.

[18] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, pp. 560–570, Oct. 1979.

[19] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, March 1985.

[20] R. Daily, *Stream Function Path Planning and Control for Unmanned Ground Vehicles*. phdthesis, Auburn University, Aug. 2008.

[21] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, pp. 566–580, Aug 1996.

[22] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., Iowa State University, 1998.

[23] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, vol. 3, pp. 2383–2388 vol.3, Sep. 2002.

[24] K. Naderi, J. Rajamäki, and P. Hämäläinen, "RT-RRT*: A real-time path planning algorithm based on RRT*," in *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, MIG '15, (New York, NY, USA), pp. 113–118, ACM, 2015.

[25] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *arXiv preprint arXiv:1005.0416*, 2010.

[26] J. Richalet, A. Rault, J. Testud, and J. Papon, "Model predictive heuristic control: applications to industrial processes," *Automatica*, vol. 14, pp. 413–428, sep 1978.

[27] E. F. Camacho and C. Bordons, *Model Predictive Control*. Springer-Verlag GmbH, second ed., 2007.

[28] L. Wang, *Model Predictive Control System Design and Implementation Using MATLAB*. Advances in Industrial Control, Springer-Verlag GmbH, 2009.

[29] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*. Communcations and Control Engineering, Springer International Publishing, second ed., 2017.

[30] F. Borrelli, A. Bemporad, M. Fodor, and D. Hrovat, "A hybrid approach to traction control," in *Hybrid Systems: Computation and Control* (M. D. Di Benedetto and A. Sangiovanni-Vincentelli, eds.), (Berlin, Heidelberg), pp. 162–174, Springer Berlin Heidelberg, 2001.

[31] F. Borrelli, P. Falcone, T. Keviczky, J. Asgari, and D. Hrovat, "MPC–based approach to active steering for autonomous vehicle systems," *International Journal of Vehicle Autonomous Systems*, vol. 3, pp. 265–291, 01 2005.

[32] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on Control Systems Technology*, vol. 15, pp. 566–580, May 2007.

[33] C. E. Beal and J. C. Gerdes, "Enhancing Vehicle Stability Through Model Predictive Control," vol. ASME 2009 Dynamic Systems and Control Conference, Volume 1 of *Dynamic Systems and Control Conference*, pp. 197–204, 10 2009.

[34] C. E. Beal and J. C. Gerdes, "Model predictive control for vehicle stabilization at the limits of handling," *IEEE Transactions on Control Systems Technology*, vol. 21, pp. 1258–1269, July 2013.

[35] J. M. Park, D. W. Kim, Y. Yoon, and K. S. Yi, "Obstacle avoidance of autonomous vehicles based on model predictive control," *Proceedings of The Institution of Mechanical Engineers Part D-journal of Automobile Engineering*, vol. 223, pp. 1499–1516, Dec. 2009.

[36] Y. Yoon, T. Choe, Y. Park, and H. J. Kim, "Obstacle avoidance for wheeled robots in unknown environments using model predictive control," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 6792 – 6797, 2008. 17th IFAC World Congress.

[37] Y. Yoon, J. Shin, H. J. Kim, Y. Park, and S. Sastry, "Model-predictive active steering and obstacle avoidance for autonomous ground vehicles," *Control Engineering Practice*, vol. 17, no. 7, pp. 741 – 750, 2009.

[38] Y. Gao, A. Gray, H. E. Tseng, and F. Borrelli, "A tube-based robust nonlinear predictive control approach to semiautonomous ground vehicles," *Vehicle System Dynamics*, vol. 52, no. 6, pp. 802–823, 2014.

[39] G. P. Bevan, H. Gollee, and J. O'Reilly, "Trajectory generation for road vehicle obstacle avoidance using convex optimization," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, vol. 224, no. 4, pp. 455–473, 2010.

[40] S. M. Erlien, S. Fujita, and J. C. Gerdes, "Safe driving envelopes for shared control of ground vehicles," *IFAC Proceedings Volumes*, vol. 46, no. 21, pp. 831 – 836, 2013. 7th IFAC Symposium on Advances in Automotive Control.

[41] J. Funke, M. Brown, S. M. Erlien, and J. C. Gerdes, "Prioritizing collision avoidance and vehicle stabilization for autonomous vehicles," in *2015 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1134–1139, June 2015.

[42] A. Eick, "A nonlinear model predictive control algorithm for an unmanned ground vehicle on variable terrain," Master's thesis, Auburn University, 2016.

[43] V. Stamenov, S. Geiger, D. Bevly, and C. Balas, "Robust vehicle stability based on nonlinear model predictive control and environmental characterization," in *Proceedings of the NDIA Ground Vehicle Systems Engineering and Technology Symposium*, vol. Autonomous Ground Systems (AGS) Technical Session, (Novi, Michigan), Aug. 2017.

[44] R. P. Shaw and D. M. Bevly, "Proportional navigation and model predictive control of an unmanned autonomous ground vehicle for obstacle avoidance," in *Proceedings of the*

ASME 2018 Dynamic Systems and Control Conference, vol. DSCC2018, ASME, Oct. 2018.

[45] R. Shaw, "Obstacle avoidance of an unmanned ground vehicle using a combined approach of model predictive control and proportional navigation," mathesis, Auburn University, Dec. 2018.

[46] J. P. Alsterda, M. Brown, and J. C. Gerdes, "Contingency model predictive control for automated vehicles," *2019 American Control Conference (ACC)*, pp. 717–722, 2019.

[47] C. Lee, "Autonomous convoy tech moves toward official program." `https://www.nationaldefensemagazine.org/articles/2019/2/22/autonomous-convoy-tech-moves-toward-official-program`, Feb. 2019. Accessed: 29 Oct. 2019.

[48] R. Bishop, D. Bevly, L. Humphreys, S. Boyd, and D. Murray, "Evaluation and testing of driver-assistive truck platooning: Phase 2 final results," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2615, pp. 11–18, 01 2017.

[49] P. Smith and D. Bevly, "Analysis of on-road highway testing for a two truck cooperative adaptive cruise control (CACC) platoon," *SAE Technical Paper*, 2019.

[50] P. Slowik and B. Sharpe, "Automation in the long haul: challenges and opportuniteis of autonomous heavy-duty trucking in the United States," tech. rep., The International Council on Clean Transportation, Mar. 2018.

[51] M. Graf Plessen, D. Bernardini, H. Esen, and A. Bemporad, "Spatial-based predictive control and geometric corridor planning for adaptive cruise control coupled with obstacle avoidance," *IEEE Transactions on Control Systems Technology*, vol. 26, pp. 38–50, Jan 2018.

[52] W. D. Herrera and N. Lidander, "Lateral control of heavy duty vehicles in platooning using model predictive control," mathesis, Chalmers University of Technology, Gothenburg, Sweden, 2016.

[53] "Racing car automobile race." https://free-images.com/display/racing_car_automobile_race.html, 2017. [Online] Accessed: 5 Nov. 2019.

[54] SAE International Surface Vehicle Recommended Practice, "Vehicle dynamics terminology," *SAE Standard J670, Rev. Jan. 2008*.

[55] T. Foote and M. Purvis, "REP 103 – Standard Units of Measure and Coordinate Conventions." https://www.ros.org/reps/rep-0103.html, Dec. 2014. Accessed: 5 Nov. 2019.

[56] P. Misra and P. Enge, *Global positioning system: signals, measurements, and performance*. Lincoln, MA 01733: Ganga-Jamuna Press, revised 2nd ed. ed., 2011. pp. 102-138.

[57] R. T. Austin, "Earth tangential plane." https://en.wikipedia.org/wiki/File:EarthTangentialPlane.png, May 2007. [Online] Accessed: 11 Nov. 2019.

[58] W. Meeussen, "REP 105 – Coordinate Frames for Mobile Platforms." https://www.ros.org/reps/rep-0103.html, Oct. 2010. Accessed: 5 Nov. 2019.

[59] R. N. Jazar, *Vehicle Dynamics: Theory and Application*. New York, NY: Springer, 2009.

[60] H. B. Pacejka, *Tyre and Vehicle Dynamics*. Oxford, UK: Butterworth-Heinemann, 2nd ed., 2006.

[61] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, pp. 2149–2154 vol.3, Sep. 2004.

[62] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[63] R. Smith, "Open dynamics engine ODE. multibody dynamics simulation software." `http://www.ode.org`, 2019. [Online] Accessed: 15 Nov. 2019.

[64] R. Brothers and D. Bevly, "A comparison of vehicle handling fidelity between the Gazebo and ANVEL simulators," in *Proceedings of the Ground Vehicle Systems Engineering and Technology Symposium (GVSETS)*, (Novi, MI), NDIA, Aug. 2019.

[65] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2009.

[66] CVX Research, Inc., "CVX: Matlab software for disciplined convex programming, version 2.0." `http://cvxr.com/cvx`, Aug. 2012.

[67] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl, *Model predictive control: theory, computation, and design*. Nob Hill Publishing, second ed., Feb. 2019.

[68] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, (Berkeley, Calif.), pp. 481–492, University of California Press, 1951.

[69] J. Nocedal and S. J. Wright, *Numerical Optimization*. Springer-Verlag GmbH, 2006.

[70] A. Wächter, "Short tutorial: Getting started with ipopt in 90 minutes," in *Combinatorial Scientific Computing*, vol. 09061, Combinatorial Scientific Computing, 2009.

[71] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, In Press, 2018.

[72] W. E. Travis, *Path duplication using GPS carrier based relative position for automated ground vehicle convoys*. phdthesis, Auburn University, May 2010.

[73] W. Travis, S. M. Martin, D. W. Hodo, and D. M. Bevly, "Non-line-of-sight automated vehicle following using a dynamic base RTK system," *NAVIGATION, Journal of the Institute of Navigation*, vol. 58, no. 3, pp. 241–255, 2011.

[74] S. M. Martin, "Closely coupled GPS/INS relative positioning for automated vehicle convoys," mathesis, Auburn University, May 2011.

[75] S. A. Geiger, "Laterally string stable control at large following distances using DRTK and TDCP," mathesis, Auburn University, Aug. 2018.

[76] K. H. Johansson, M. Törngren, and L. Nielsen, "Vehicle applications of controller area network," in *Handbook of Networked and Embedded Control Systems*, 2005.

[77] W. G. Apperson, "Design and evaluation of cooperative adaptive cruise control system for heavy freight vehicles," Master's thesis, Auburn University, Dec. 2019.

[78] Z. Sun, G. Bebis, and R. Miller, "On-road vehicle detection: a review," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, pp. 694–711, May 2006.

[79] S. Sivaraman and M. M. Trivedi, "Looking at vehicles on the road: A survey of vision-based vehicle detection, tracking, and behavior analysis," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, pp. 1773–1795, Dec 2013.

[80] R. Mobus and U. Kolbe, "Multi-target multi-object tracking, sensor fusion of radar and infrared," in *IEEE Intelligent Vehicles Symposium, 2004*, pp. 732–737, June 2004.

[81] X. Mao, D. Inoue, S. Kato, and M. Kagami, "Amplitude-modulated laser radar for range and speed measurement in car applications," *IEEE Transactions on Intelligent Transportation Systems*, vol. 13, pp. 408–413, March 2012.

[82] M. Darms, P. Rybski, C. Baker, and C. Urmson, "Obstacle detection and tracking for the urban challenge," *Intelligent Transportation Systems, IEEE Transactions on*, vol. 10, pp. 475 – 485, 10 2009.

[83] I. Bogoslavskyi and C. Stachniss, "Efficient online segmentation for sparse 3d laser scans," *PFG – Journal of Photogrammetry, Remote Sensing and Geoinformation Science*, vol. 85, pp. 41–52, Feb 2017.

[84] T. Miyasaka, Y. Ohama, and Y. Ninomiya, "Ego-motion estimation and moving object tracking using multi-layer LIDAR," in *2009 IEEE Intelligent Vehicles Symposium*, pp. 151–156, June 2009.

[85] J. Arrospide, L. Salgado, M. Nieto, and F. Jaureguizar, "On-board robust vehicle detection and tracking using adaptive quality evaluation," in *2008 15th IEEE International Conference on Image Processing*, pp. 2008–2011, Oct 2008.

[86] H. Badino, U. Franke, and R. Mester, "Free space computation using stochastic occupancy grids and dynamic programming," 03 2012.

[87] A. Asvadi, P. Girão, P. Peixoto, and U. Nunes, "3D object tracking using RGB and LIDAR data," in *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 1255–1260, Nov 2016.

[88] A. Rangesh and M. Trivedi, "No blind spots: full-surround multi-object tracking for autonomous vehicles using cameras & LiDARs," *IEEE Transactions on Intelligent Vehicles*, vol. PP, 02 2018.

[89] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The KITTI vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, June 2012.

[90] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: the KITTI dataset," *The International Journal of Robotics Research*, vol. 32, pp. 1231–1237, Aug. 2013.

[91] "Papers with code: 3D object detection." `https://paperswithcode.com/task/3d-object-detection`, 2020. [Online] Accessed: 13 Jan. 2020.

[92] T. D. Gillespie, *Fundamentals of Vehicle Dynamics*. 1992.

[93] Y. Kawajir, C. Laird, S. Vigerske, and A. Wächter, "Introduction to IPOPT: a tutorial for downloading, installing, and using IPOPT." `https://coin-or.github.io/Ipopt/`, 2015. accessed: 5 April 2020.

Appendices

Appendix A

MKZ System Identification

Identifying parameters for vehicle dynamic modeling is made challenging by the model's dependence on vehicle speed and the non-linear relationships between tire slip and tire force generation. The bicycle model significantly simplifies the number of parameters to identify. Most physical parameters in the bicycle model (mass, inertia, weight split) can be measured or approximated with measurements of the stationary vehicle. The linear tire stiffness parameters, however, must be extracted from an analysis of dynamic driving data. Many techniques have been developed to extract dynamic modeling information from evaluation of the vehicle's steady-state response. The dynamic bicycle model's steady-state behavior is studied through the DC gain of its transfer functions. The bicycle model equations of motion, given in Equations (2.25–2.26), can be expressed in the form of the linear transfer functions in Equations (A.1–A.2).

$$\frac{v_y(s)}{\delta(s)} = \frac{\frac{C_{\alpha_f}}{mv_x}s + \frac{C_2C_{\alpha_f} - aC_1C_{\alpha_f} - amC_{\alpha_f}v_x^2}{I_{zz}mv_x^2}}{s^2 + \frac{I_{zz}C_0 + mC_2}{I_{zz}mv_x}s + \frac{C_0C_2 - C_1^2 - C_1mv_x^2}{I_{zz}mv_x^2}} \tag{A.1}$$

$$\frac{\dot{\psi}(s)}{\delta(s)} = \frac{\frac{aC_{\alpha_f}}{I_{zz}}s + \frac{aC_{\alpha_f}C_0 - C_1C_{\alpha_f}}{I_{zz}mv_x}}{s^2 + \frac{I_{zz}C_0 + mC_2}{I_{zz}mv_x}s + \frac{C_0C_2 - C_1^2 - C_1mv_x^2}{I_{zz}mv_x^2}} \tag{A.2}$$

The simplification terms C_0, C_1, and C_2 are given in Equations (A.3–A.5).

$$C_0 = C_{\alpha_f} + C_{\alpha_r} \tag{A.3}$$

$$C_0 = aC_{\alpha_f} - bC_{\alpha_r} \tag{A.4}$$

$$C_0 = a^2C_{\alpha_f} + b^2C_{\alpha_r} \tag{A.5}$$

Equation (A.1) can also be put in terms of the side-slip dynamic, as shown in Equation (A.6). The side-slip formulation will be used for the remainder of the system identification analysis

since the steady-state side-slip can be obtained from a minimal set of GPS heading and course measurements (see Equation (2.3)).

$$\frac{\beta(s)}{\delta(s)} = \frac{\frac{C_{\alpha_f}}{mv_x}s + \frac{C_2C_{\alpha_f} - aC_1C_{\alpha_f} - amC_{\alpha_f}v_x^2}{I_{zz}mv_x^2}}{s^2 + \frac{I_{zz}C_0 + mC_2}{I_{zz}mv_x}s + \frac{C_0C_2 - C_1^2 - C_1mv_x^2}{I_{zz}mv_x^2}} \tag{A.6}$$

The DC gain relationships for Equation (A.6) and Equation (A.2) are given in Equation (A.7) and Equation (A.8) respectively.

$$\frac{\beta(s=0)}{\delta(s=0)} = \frac{\frac{C_2C_{\alpha_f} - aC_1C_{\alpha_f} - amC_{\alpha_f}v_x^2}{I_{zz}mv_x^2}}{\frac{C_0C_2 - C_1^2 - C_1mv_x^2}{I_{zz}mv_x^2}} \tag{A.7}$$

$$\frac{\dot{\psi}(s=0)}{\delta(s=0)} = \frac{\frac{aC_{\alpha_f}C_0 - C_1C_{\alpha_f}}{I_{zz}mv_x}}{\frac{C_0C_2 - C_1^2 - C_1mv_x^2}{I_{zz}mv_x^2}} \tag{A.8}$$

While these two equations can be used to back out the tire stiffness values (assuming the velocity, mass, inertia and DC gain values are known), a simple closed form solution is realized by using an additional steady-state handling relationship.

Vehicle handling is commonly characterized using only the steady-state steering response. Gillespie claims that the understeer gradient is the most common measure for open-loop handling [92]. The understeer gradient, K_{us}, is a constant that relates the amount of steering input required to hold a steady-state turn give the vehicle's lateral acceleration at the CG, a_y. This relationship is given in Equation (A.9)

$$\delta = \frac{L}{R} + K_{us} \cdot a_y \tag{A.9}$$

In the equation above, L is the vehicle wheel-base length and R is the turning radius. The understeer gradient can also be put in terms of the linear tire stiffness values, as shown in Equation (A.10)

$$K_{us} = \frac{mb}{LC_{\alpha_f}} - \frac{ma}{LC_{\alpha_r}} \tag{A.10}$$

Equation (A.10) can be re-arranged to solve for the combined tire stiffness term in Equation (A.4), shown in Equation (A.11).

$$C_1 = -\frac{C_{\alpha_f} C_{\alpha_r} L K_{us}}{m} \tag{A.11}$$

Equation (A.7) and Equation (A.8) can be significantly simplified by substituting Equation (A.11). These simplifications are shown in Equation (A.12) and Equation (A.13) respectively.

$$\frac{\beta(s=0)}{\delta(s=0)} = \frac{b - \frac{m a v_x^2}{L C_{\alpha_r}}}{L + K_{us} v_x^2} \tag{A.12}$$

$$\frac{\dot{\psi}(s=0)}{\delta(s=0)} = \frac{v_x}{L + K_{us} v_x^2} \tag{A.13}$$

The rear tire stiffness value can be found directly, as shown in Equation (A.14), by substituting Equation (A.13) into Equation (A.12) and Equation (A.10) can be rearranged to solve for the front tire stiffness, given in Equation (A.15).

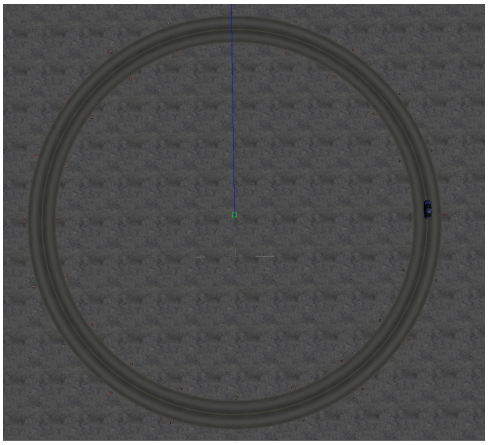
$$C_{\alpha_r} = \frac{m a v_x \dot{\psi}(s=0)}{L\left(\frac{b}{v_x}\dot{\psi}(s=0) - \beta(s=0)\right)} \tag{A.14}$$

$$C_{\alpha_f} = \frac{mb}{\frac{ma}{C_{\alpha_r}} + L K_{us}} \tag{A.15}$$

Equations (A.14–A.15) can be solved assuming the DC gains and the understeer gradient are known. In comparison to directly measuring tire stiffness values, measuring and obtaining accurate estimates of these parameters is easy.

Four common test procedures can be performed to estimate the understeer gradient for steady-state parameter identification. Any two of the three independent variables (δ, a_y, R) in Equation (A.9) can be varied while the other is held constant. The constant radius test, the procedure used in this work, holds the turning radius, R, constant along a circular track. The vehicle speed is slowly incremented such that the steady-state steering angle is developed at each step up in speed. The constant radius test was performed both in simulation and on the MKZ test vehicle. The Gazebo simulation constant radius test was conducted with a 50

meter radius track. The simulation test was manually steered with a steering joystick while a ROS node handled slowly increasing the velocity and holding it constant for an entire lap. The Gazebo simulation test track and driving set-up is shown in Figure A.1. The constant radius



(a) Gazebo 50 m Radius Test Track

(b) Gazebo Manual Driving Set-up

Figure A.1: Gazebo Simulation Constant Radius (50 m) Test Set-up

tests performed with the MKZ test vehicle were conducted at the NCAT test facility using a 25 meter radius track. The vehicle was manually driven (both steering and velocity) by slowly increasing speed each lap around the track and maintaining radial position until the vehicle reached its lateral grip limits. The test path recorded from GPS is shown in Figure A.2

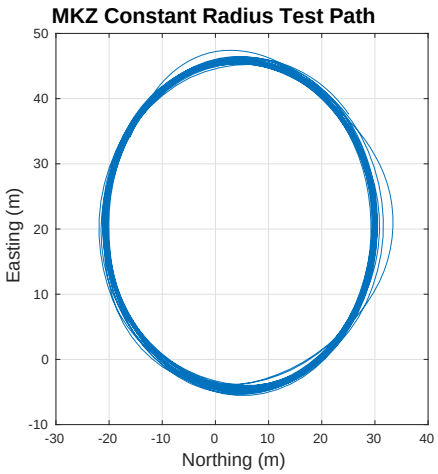


Figure A.2: MKZ Constant Radius (25 m) Test Track Path

A simple linear least squares estimation technique is used with the understeer model in Equation (A.9) to fit the understeer gradient to the constant radius test data. The understeer

gradient estimate, $\hat{K}_{us}$, from least squares is shown in Equation (A.16).

$$\hat{K}_{us} = \left(H^T H\right)^{-1} H^T y \tag{A.16}$$

The measurement in Equation (A.16) is $y = \delta - \frac{L}{R}$ and the model is simply $H = a_y$, where a_y is assumed to be the vehicle's centripetal acceleration at steady state, given in Equation (A.17)

$$a_y = \frac{|\vec{v}|^2}{R} \tag{A.17}$$

The results of the Gazebo constant radius test and the fit of the understeer gradient for the simulated MKZ vehicle are shown in Figure A.3. Similarly, the results of the MKZ test vehicle's constant radius tests and the fit of its understeer gradient are shown in Figure A.4. Inter-

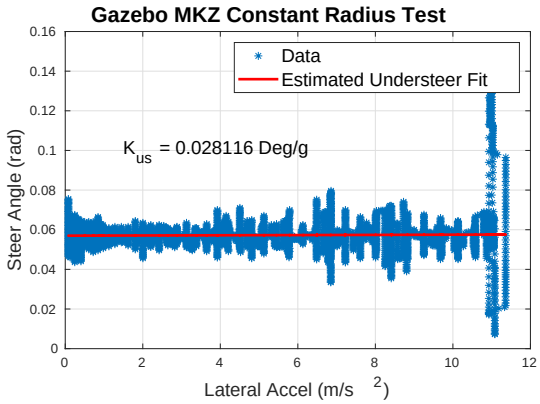


Figure A.3: Gazebo MKZ Constant Radius Test Results and Fit of Understeer Gradient

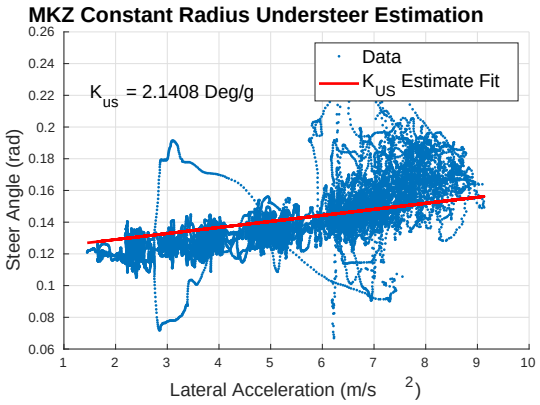


Figure A.4: MKZ Test Vehicle Constant Radius Test Results and Fit of Understeer Gradient

estingly, the simulated MKZ vehicle exhibits neutral steer ($K_{us} \approx 0$) behavior, allowing the

vehicle to maintain grip at lateral accelerations $> 1g$; however, the real MKZ vehicle has an understeer gradient that is within the typical range for a normal sedan, $K_{us} = 2.1408 \frac{deg}{g}$. The MKZ test vehicle also hit its grip limits at $< 1g$. This discrepancy results in a difference in tire stiffness values (both magnitude and split) between the simulated and real MKZ vehicles.

After determining the understeer gradient values, steady state yaw rates and side slips for both the simulated and real MKZ were collected with a series of step steer input tests. In the Gazebo simulation each test used a longitudinal controller to maintain a desired speed. After reaching and settling on the desired speed, a constant steering angle was commanded. With velocity data collected at the simulation vehicle's CG, the tire stiffness values were calculated for each run. The final set of tire stiffness values is the result of an average across all test speeds. Yaw rates and side-slip truth data from the simulation vehicle and a simulation with the extracted bicycle model are shown in Figure A.5, Figure A.6, and Figure A.7 at low, medium, and high speed respectively. Similarly, the model predicted yaw rate and side-slip for the MKZ

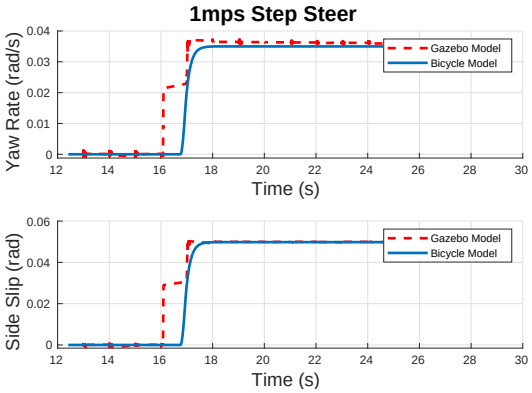


Figure A.5: Gazebo MKZ Step Steer Test Model Comparison: 1 m/s

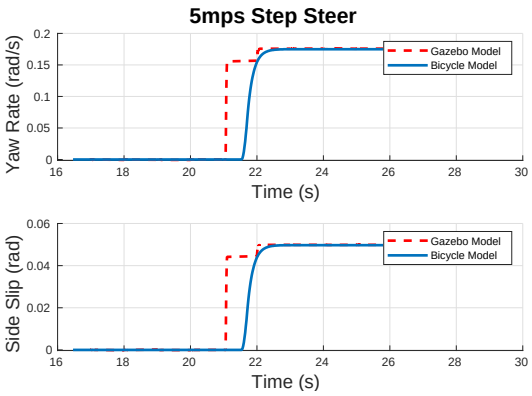


Figure A.6: Gazebo MKZ Step Steer Test Model Comparison: 5 m/s

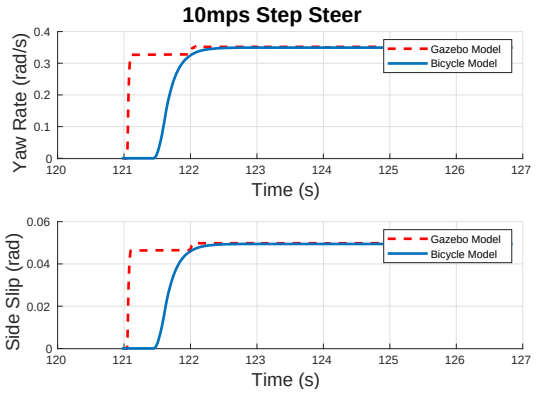


Figure A.7: Gazebo MKZ Step Steer Test Model Comparison: 10 m/s

test vehicle are plotted over the estimated yaw-rate and side slip data from the experimental step steer procedures in Figure A.8, Figure A.9, and Figure A.10 for low, medium, and high speed respectively. The tire stiffness values and other important model parameters of both the

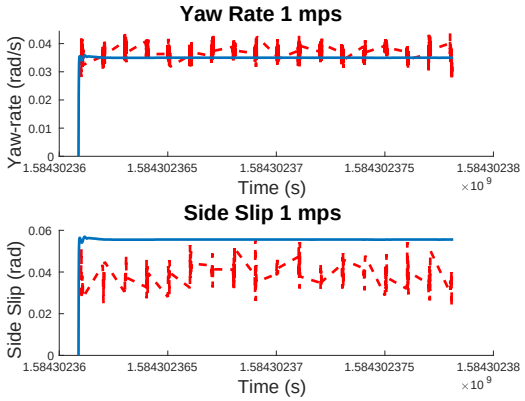


Figure A.8: MKZ Vehicle Step Steer Test Model Comparison: 1 m/s

simulated MKZ and MKZ test vehicle are summarized in Table 2.1 and below in Table A.1.

The open-loop bandwidth of the identified dynamic bicycle model changes with increasing forward velocity, but remains low by some standards. In fact, the lateral handling bandwidth tends to decrease at increasing speeds. To show this, the frequency response of the real MKZ's identified model is plotted on a bode diagram in Figure A.11 at multiple speeds. The bandwidth, in this case the system's cutoff frequency, is also given for each model with its given forward speed, v_x, in Table A.2. This low open-loop bandwidth can explain why using the bicycle model with a model predictive controller might not provide as much of an advantage as expected over the kinematic model for trajectory tracking.

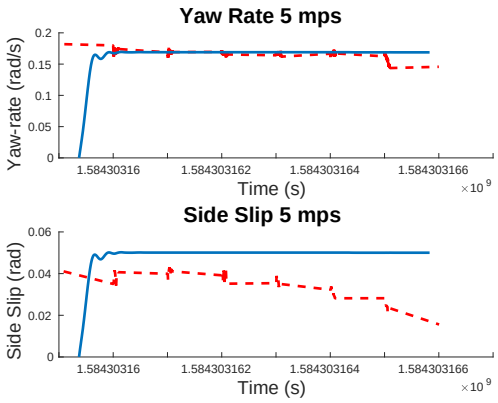
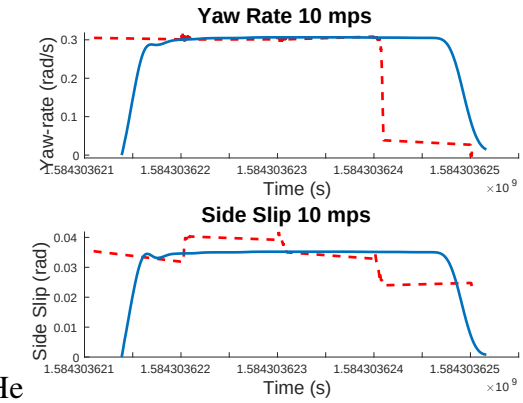


Figure A.9: MKZ Vehicle Step Steer Test Model Comparison: 5 m/s



He

Figure A.10: MKZ Vehicle Step Steer Test Model Comparison: 10 m/s

Table A.1: MKZ Modeling Properties

property	Gazebo MKZ	Real MKZ
$a(m)$	1.428	1.257
$b(m)$	1.423	1.593
$L(m)$	2.850	2.850
track width (m)	1.594	1.594
mass (Kg)	1542	1857
$I_{zz}(Kg \cdot m^2)$	1000	4292
$C_{\alpha_f}(N/rad)$	31240	120000
$C_{\alpha_r}(N/rad)$	31240	184600

Table A.2: MKZ Bicycle Model Bandwidths at Varying Forward Speeds

Speed (m/s)	Bandwidth (Hz)
5	3.71
10	2.56
15	1.84
20	1.51

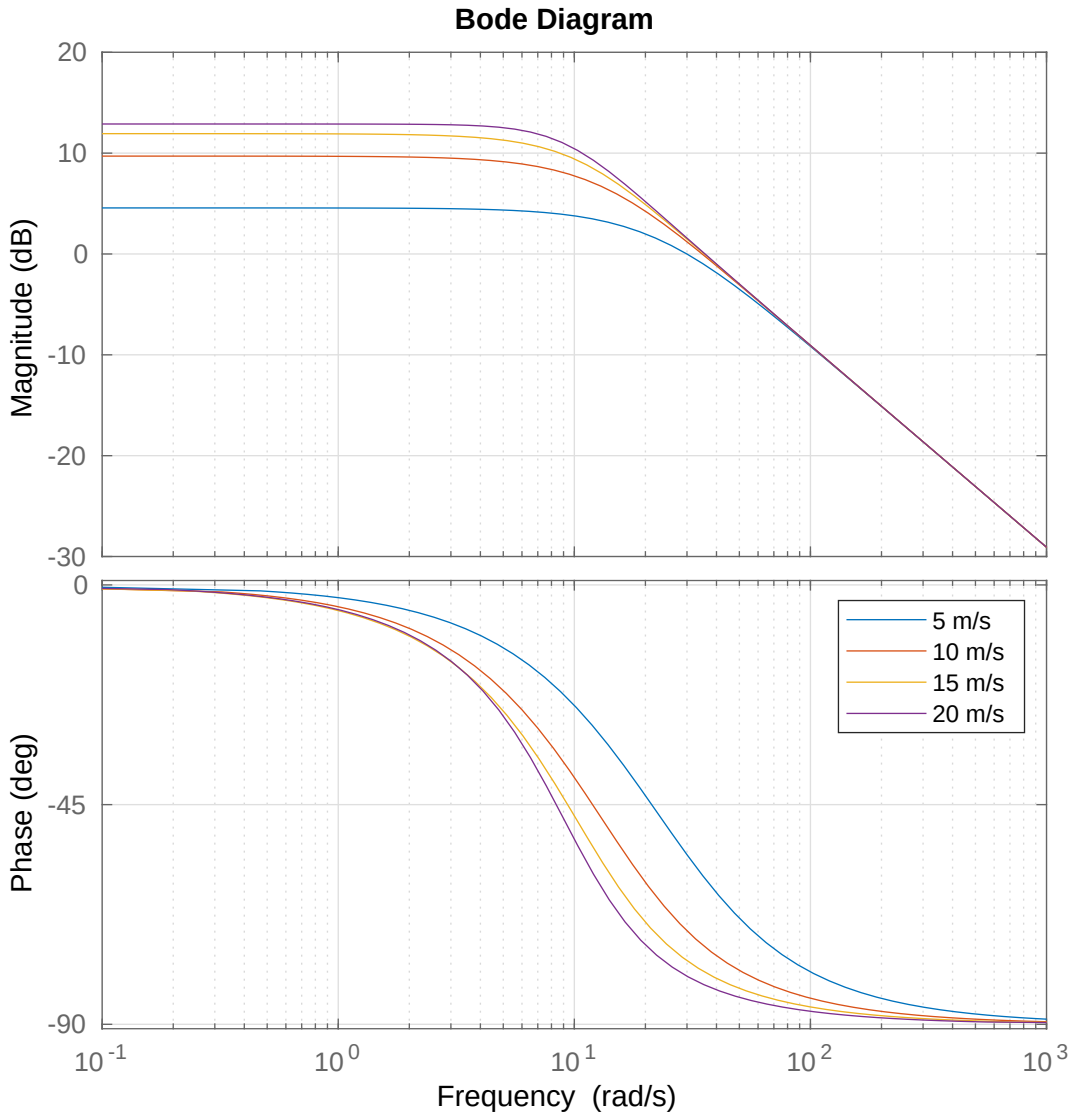


Figure A.11: Bicycle Model Frequency Response for Real MKZ Vehicle

To compare the kinematic and bicycle models, the bicycle model's DC and transient response are plotted with the kinematic model's response for various speeds. All models are simulated with a step-steer input of $\delta = 0.1\ rad$ ($\approx 90^\circ$ at the hand wheel) at 5, 10, 15, and 20 m/s and the results are shown in Figure A.12. Notice that the kinematic model, given in Equation (2.14), has a similar steady-state response to the bicycle model at low speeds, but diverges at higher speeds. This discrepancy is the presence of the handling property understeer. Notice also that the simplified version of the kinematic model, given in Equation (2.15), is very similar in form to the simplified version of the bicycle model's yaw-rate DC gain, given in Equation (A.13). The difference is the addition of the understeer term, $K_{us}v_x^2$, to the denominator of the equation derived from the dynamic bicycle model. In future NMPC implementations it may be desirable to develop a controller based on this yaw-rate DC gain relationship to replace the current kinematic model. This would require only identifying the vehicle's understeer gradient and may provide a good balance of the benefits of control with each model.

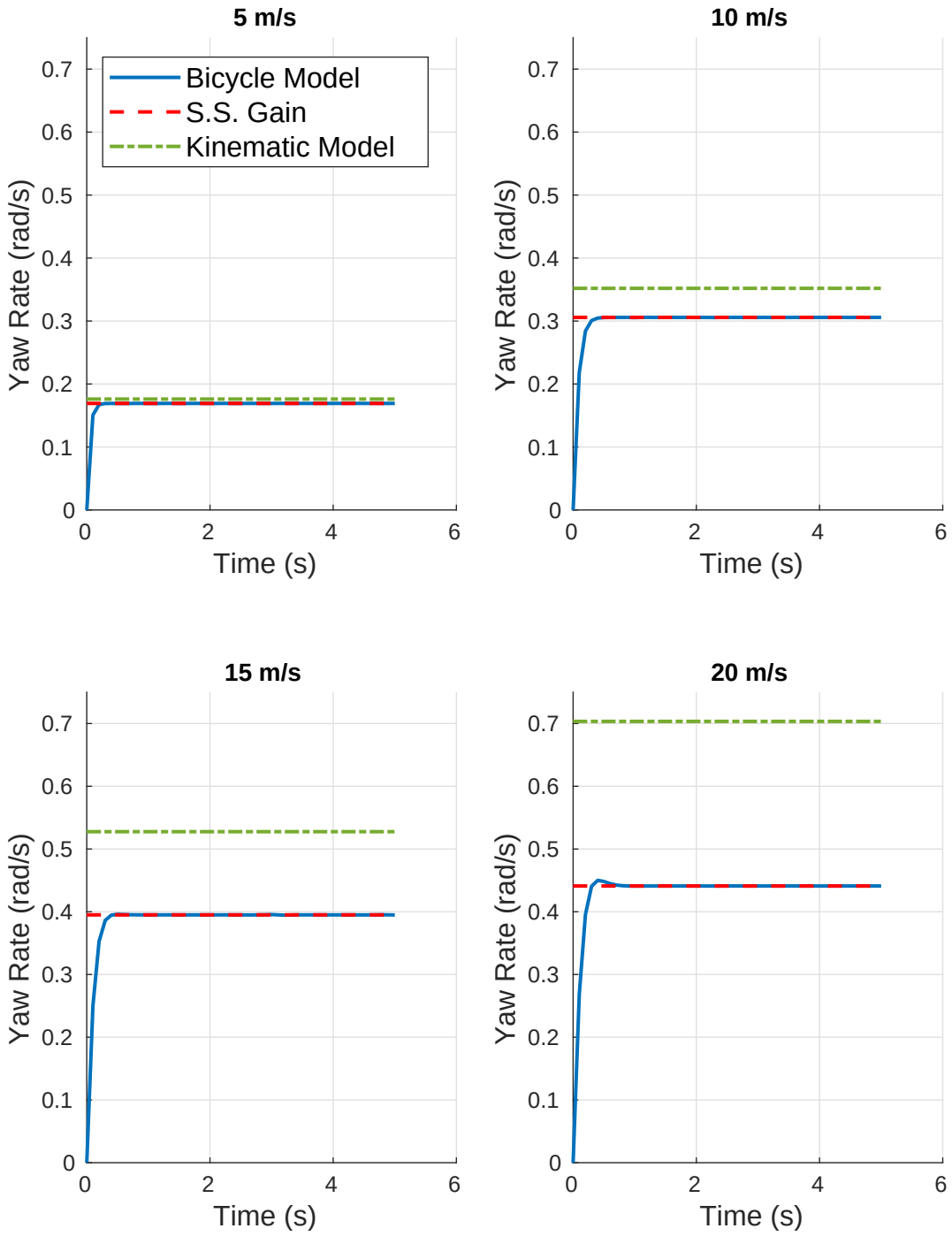


Figure A.12: MKZ Model Comparisons: Step Steer Response at Various Speeds

Appendix B

MPC With IPOPT: An Example Sparsity Encoding Problem

The IPOPT optimization solver software utilizes sparse matrix representations of the optimization problem's constraint gradients and the Hessian of the Lagrangian to reduce the required memory and run-time computational cost of each optimization iteration. This appendix will demonstrate the requirements of implementing a simple optimal control problem in IPOPT with an example system that is very similar to the kinematic bicycle model presented in chapter 2. The computed Jacobian and Hessian matrices for this problem will be presented with their sparsity structures. For brevity, only a single multi-shooting model prediction constraint set will be considered.

Differential drive robots can be controlled with a very simple model that uses a desired yaw-rate and linear velocity as the model inputs. Similar to the kinematic bicycle model, these desired velocity commands can be passed to lower level controllers that are tuned for the robot's actuator interfaces. An applicable robot platform for this kind of differential drive controller is the GAVLab's ATRV robot, shown in Figure B.1. The simplest model of this vehicle for use in a model predictive controller is given in Equation (B.1).

$$\frac{d}{dt}\begin{pmatrix} X \\ Y \\ \psi \end{pmatrix} = \begin{pmatrix} |\vec{v}|\cos(\psi) \\ |\vec{v}|\sin(\psi) \\ \dot{\psi} \end{pmatrix} \tag{B.1}$$

The system state, x, and control input u are given in Equations (B.2–B.3).

$$x = \begin{pmatrix} X \\ Y \\ \psi \end{pmatrix} \tag{B.2}$$

Figure B.1: GAVLab ATRV Differential Drive Robot

$$u = \begin{pmatrix} |\vec{v}| \\ \dot{\psi} \end{pmatrix} \tag{B.3}$$

The position states, X and Y, define a plane in a generic fixed coordinate system. The heading angle, ψ, is defined as the vehicle's positive rotation from the X axis. The robot's linear velocity, $|\vec{v}|$, and yaw-rate, $\dot{\psi}$, are considered independently controllable through some lower-level control interface.

The optimal control problem formulation given in Equation (3.11) will be used with only a single prediction step, $N = 1$; however, it must be put in the generic form of equation 3.20. The generic optimization variable, z, is given in Equation (B.4) in terms of each state.

$$z = \left[X(0), Y(0), \psi(0), X(1), Y(1), \psi(1), |\vec{v}|(0), \dot{\psi}(0)\right]^T \tag{B.4}$$

The cost function, $\phi(z)$, is expanded in terms of Equation (B.4) and given in Equation (B.5).

$$\begin{aligned}\phi(z) = Q_X X(0)^2 + Q_Y Y(0)^2 + Q_\psi \psi(0)^2 + Q_X X(1)^2 + \\ Q_Y Y(1)^2 + Q_\psi \psi(1)^2 + R_{|\vec{v}|}|\vec{v}|(0) + R_{\dot{\psi}}\dot{\psi}(0)\end{aligned} \tag{B.5}$$

The initial condition constraints are defined in Equations (B.6a–B.6c).

$$X(0) - X(0)_{meas} = 0 \tag{B.6a}$$

$$Y(0) - Y(0)_{meas} = 0 \tag{B.6b}$$

$$\psi(0) - \psi(0)_{meas} = 0 \tag{B.6c}$$

The measured states in Equation (B.6), $X(0)_{meas}$, $Y(0)_{meas}$, and $\psi(0)_{meas}$, are the most recent measurements of the robot state and are considered constant parameters in each optimization iteration. The model prediction constraints are given in Equation (B.7) using Euler's method to propogate the states over the prediction time step, T.

$$X(1) - (X(0) + |\vec{v}|(0)\cos(\psi(0))T) \tag{B.7a}$$

$$Y(1) - (Y(0) + |\vec{v}|(0)\sin(\psi(0))T) \tag{B.7b}$$

$$\psi(1) - \Big(\psi(0) + \dot{\psi}(0)T\Big) \tag{B.7c}$$

These equations will be used to define and derive the requirements for the IPOPT optimization interface.

IPOPT requires the following individual function evaluations in a generic optimizer interface:

- The objective function, $\phi(z)$
- The gradient of the objective function, $\nabla\phi(z)$
- Constraint equations, $c_i(z) = 0 \qquad \forall i = 1, ..., m$
- Jacobian of the constraint equations, $\nabla c(z)$

- The Hessian of the Lagrangian, $\nabla^2\phi(z) + \sum_{i=1}^{i=m} \lambda_i \nabla^2 c_i(z)$

The objective function and constraint equations have already been defined in Equations (B.5–B.7), but the remaining functions evaluations still must be derived. The gradient of the objective function is given below in Equation (B.8).

$$\nabla\phi(z) = [2Q_X X(0), 2Q_Y Y(0), 2Q_\psi \psi(0), 2Q_X X(1), \\ 2Q_Y Y(1), 2Q_\psi \psi(1), 2R_{|\vec{v}|}|\vec{v}|(0), 2R_{\dot{\psi}}\dot{\psi}(0)]^T \quad \text{(B.8)}$$

Notice that this function returns only an $n \times 1$ dense matrix, where n is the size of the optimization variable; it is therefore not returned with a sparsity encoding. The Jacobian of the constraint equations and the Hessian of the Lagrangian are given in Equation (B.9) and Equation (B.10) respectively.

$$\nabla c(z) = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & |\vec{v}|(0)\sin(\psi(0))T & 1 & 0 & 0 & -\cos(\psi(0))T & 0 \\ 0 & -1 & -|\vec{v}|(0)\cos(\psi(0))T & 0 & 1 & 0 & -\sin(\psi(0))T & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 & 0 & -T \end{bmatrix} \quad \text{(B.9)}$$

$$\nabla^2\phi(z)+\sum_{i=1}^{i=m}\lambda_i\nabla^2 c_i(z)=\begin{bmatrix}2Q_X & & & & & & & \\ & 2Q_Y & & & & \emptyset & & \\ & & 2Q_\psi & & & & & \\ & & & 2Q_X & & & & \\ & & & & 2Q_Y & & & \\ & \emptyset & & & & 2Q_\psi & & \\ & & & & & & 2R_{|\vec{v}|} & \\ & & & & & & & 2R_{\dot{\psi}}\end{bmatrix}+$$

$$\lambda_4\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&|\vec{v}|(0)\cos(\psi(0))T&0&0&0&\sin(\psi(0))T&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&\sin(\psi(0))T&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix}+$$

$$\lambda_5\begin{bmatrix}0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&|\vec{v}|(0)\sin(\psi(0))T&0&0&0&-\cos(\psi(0))T&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&0&0&0&0&0&0\\0&0&-\cos(\psi(0))T&0&0&0&0&0\\0&0&0&0&0&0&0&0\end{bmatrix} \tag{B.10}$$

The matrix for the Jacobian of the constraints is very sparse with only 14 non-zero elements and the Hessian of the Lagrangian is even sparser with only 10 non-zero elements. The sparsity, which can be thought of as the ratio of zero elements to non-zero elements, grows with additional prediction constraints and generally grows with the addition of many other kinds of

practical constraints. Operations in IPOPT with these large, sparse matrices can benefit from the compression and speed gained by using sparse matrix math libraries.

The function evaluations for the Jacobian of the constraint equations and the Hessian of the Lagrangian return 3 arrays, each the size of number of non-zero elements in the respective matrix. The first array contains only the values of the non-zero elements. The matrix position of the non-zero element value is recorded at the corresponding index of the other two arrays, the row and column position arrays. This encoding scheme is known as the triplet format for sparse matrices. More information on this encoding scheme and other example optimization problems can be found in the IPOPT documentation [93]. The sparsity encoding for the matrices given in Equation (B.9) and Equation (B.10) are displayed in Table B.1 and Table B.2 respectively. Assuming 8 byte double precision floating point numbers are used in each element of these

Table B.1: Sparsity Encoding for the Jacobian of the Constraint Equations

Row Array	Column Array	Non-zero Value Array
row[0] = 0	col[0] = 0	value[0] = 1
row[1] = 1	col[1] = 1	value[1] = 1
row[2] = 2	col[2] = 2	value[2] = 1
row[3] = 3	col[3] = 0	value[3] = -1
row[4] = 3	col[4] = 2	value[4] = $\|\vec{v}\|(0)\sin(\psi(0))T$
row[5] = 3	col[5] = 3	value[5] = 1
row[6] = 3	col[6] = 6	value[6] = $-\cos(\psi(0))T$
row[7] = 4	col[7] = 1	value[7] = -1
row[8] = 4	col[8] = 2	value[8] = $-\|\vec{v}\|(0)\cos(\psi(0))T$
row[9] = 4	col[9] = 4	value[9] = 1
row[10] = 4	col[10] = 6	value[10] = $-\sin(\psi(0))T$
row[11] = 5	col[11] = 2	value[11] = -1
row[12] = 5	col[12] = 5	value[12] = 1
row[13] = 5	col[13] = 7	value[13] = -T

matrices and 4 byte integer values are used in the matrix position arrays, the sparsity encoded Jacobian matrix uses a total of 224 bytes where the dense matrix representation requires 384 bytes, yielding a memory savings of 42%. The sparse Hessian representation given in Table B.2 requires only 160 bytes where its dense matrix counterpart would use 512 bytes, a 69% memory savings. The run-time computational savings from using the sparse matrix representations is even greater than the memory savings, and these advantages scale favorably with the size of the optimization problem. These computational efficiencies make IPOPT an attractive solver

Table B.2: Sparsity Encoding for the Hessian of the Lagrangian

Row Array	Column Array	Non-zero Value Array
row[0] = 0	col[0] = 0	value[0] = $2Q_X$
row[1] = 1	col[1] = 1	value[1] = $2Q_Y$
row[2] = 2	col[2] = 1	value[2] = $2Q_\psi + \|\vec{v}\|(0)T\left(\sin(\psi(0)) + \cos(\psi(0))\right)$
row[3] = 2	col[3] = 6	value[3] = $T\left(\sin(\psi(0)) - \cos(\psi(0))\right)$
row[4] = 3	col[4] = 3	value[4] = $2Q_X$
row[5] = 4	col[5] = 4	value[5] = $2Q_Y$
row[6] = 5	col[6] = 5	value[6] = $2Q_\psi$
row[7] = 6	col[7] = 2	value[7] = $T\left(\sin(\psi(0)) - \cos(\psi(0))\right)$
row[8] = 6	col[8] = 6	value[8] = $2R_{\|\vec{v}\|}$
row[9] = 7	col[9] = 7	value[9] = $2R_{\dot{\psi}}$

to use in a real-time model predictive control application, but the implementation is difficult to generalize to any system.

When developing a general library for flexible MPC controllers, using a symbolic math framework with automatic differentiation that can handle creating these Jacobian and Hessian matrix representations is the only tractable solution. Any change to the prediction horizon length, a cost function term, or any constraint equation can drastically alter these sparsity structures. Unless the exact structure of the controller is settled upon for a deeply embedded application, programming these sparsity encodings manually is impractical. These realizations motivate the use of the symbolic math framework CasADi in the NMPC software library presented in this work.

The custom software abstraction layer created for this thesis was originally presented in Chapter 3. The UML class diagram for this library of higher level NMPC components was previously presented in Figure 3.3, but it is shown again in Figure B.2 with higher resolution.

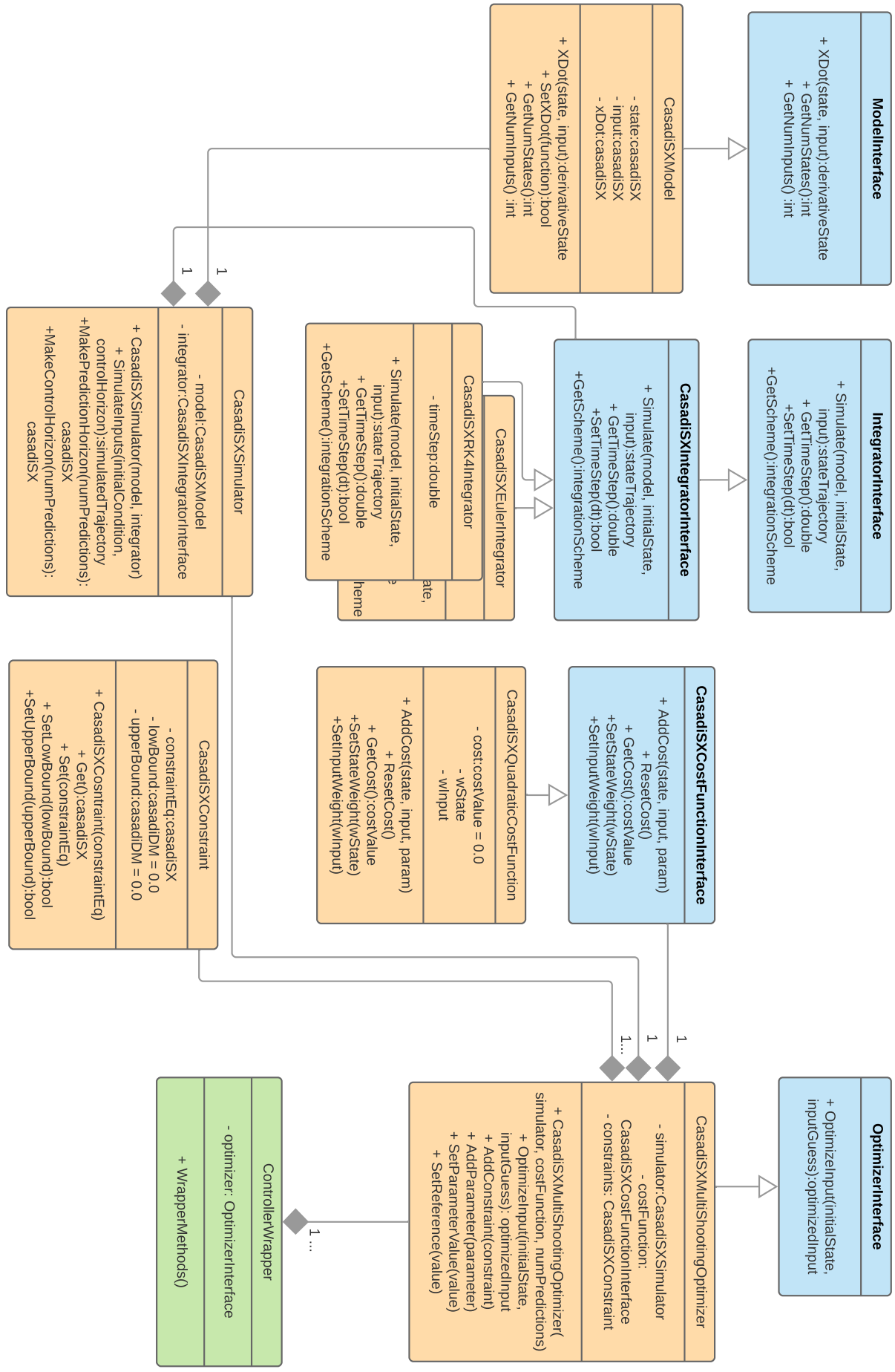


Figure B.2: NMPC Software Library UML Class Diagram

To further the example given in this appendix, a *CasadiSXMultiShootingOptimizer* class will be extended to create an NMPC controller for the differential-drive robot in the following code example. In the current version of the library (version 2.1.0 as of this writing), there is a script that creates the required files for a new controller class that inherits from the *CasadiSX-MultiShootingOptimizer* class and fills in most of the boiler-plate code that is not model specific. The C++ code required to implement the model specific setup for this example controller class, *ExampleDiffDriveController*, is shown in the listing below.

```
bool ExampleDiffDriveController::Setup()
  {
    /* Model setup */
    // model states
    casadi::SX x = casadi::SX::sym("x");
    casadi::SX y = casadi::SX::sym("y");
    casadi::SX psi = casadi::SX::sym("psi");
    // model inputs
    casadi::SX v = casadi::SX::sym("v");
    casadi::SX psiDot = casadi::SX::sym("psiDot");
    // symbolic model
    casadi::SX xDot = casadi::SX(3,1);
    xDot(0,0) = v*cos(psi);
    xDot(1,0) = v*sin(psi);
    xDot(2,0) = psiDot;
    modelPtr_->Setup(vertcat(x,y,psi), vertcat(v,psiDot), xDot);

    /* Setup additional parameters (this->AddParam()) */
    casadi::SX x0 = casadi::SX::sym("x0", GetNumStates(), 1);
    AddParam("initialCondition", x0);
    casadi::SX ref = casadi::SX::sym("ref", GetPredictionHorizonSize(), 1);
    AddParam("ref", ref);

    /* Setup control and prediciton horizon variables */
    CreateHorizons();
```

```
/* Set state and input weights */
SetStateWeight(opts_.stateWeight);
SetInputWeight(opts_.inputWeight);

/* Add up cost with costFunPtr_ */
costFunPtr_->ResetCost();
// Add up cost of initial condition
costFunPtr_->AddCost(predictionHorizon_(casadi::Slice(0,GetNumStates()),0),
  casadi::SX::zeros(GetNumInputs(),1), ref(casadi::Slice(0, GetNumStates()),0));
// Add up cost over the prediction horizon
for (auto i=0; i < GetNumPredictions(); ++i)
{
  const auto &state = predictionHorizon_(casadi::Slice((i+1)*GetNumStates(),
    (i+2)*GetNumStates()),0);
  const auto &input = controlHorizon_(casadi::Slice(i*GetNumInputs(),
    (i+1)*GetNumInputs()),0);
  const auto &target = ref(casadi::Slice((i+1)*GetNumStates(),
    (i+2)*GetNumStates()),0);
  // Check for terminal cost setup
  if (i == (GetNumPredictions()-1))
  {
    costFunPtr_->SetStateWeight(opts_.terminalCostWeight*
      std::dynamic_pointer_cast<CasadiSXQuadraticCostFunction>
        (costFunPtr_)->GetStateWeight()
    );
  }
  costFunPtr_->AddCost(state,input,target);
}

/* Setup constraints */
// Model prediction constraint
casadi::SX predictionVec = simulatorPtr_->SimulateInputsMS(predictionHorizon_,
  controlHorizon_);
predictionVec(casadi::Slice(0, GetNumStates()),0) = x0;
CasadiSXConstraint multiShootingConstraint(predictionHorizon_ - predictionVec);
```

```
  /* Concatenate constraints */
  constraints_={multiShootingConstraint /* ,... And any other constraints created*/};

  /* Setup NLP */
  casadi::SXDict nlpStructure{{"x",vertcat(predictionHorizon_,controlHorizon_)},
    {"p", params_}, {"f", costFunPtr_->GetCost()}, {"g",constraints_.Get()}};
  casadi::Dict nlpOpts;
  nlpOpts["ipopt.print_level"] = 0;
  nlpOpts["print_time"] = 0;
  nlpOpts["ipopt.acceptable_tol"] = 1e-8;
  nlpOpts["ipopt.max_iter"] = 2000;
  nlpOpts["ipopt.constr_viol_tol"] = 1e-6;
  nlpOpts["ipopt.acceptable_obj_change_tol"] = 1e-6;
  nlpOpts["ipopt.fixed_variable_treatment"] = "relax_bounds";
  try
  {
    optimFun_ = casadi::nlpsol("solver", "ipopt", nlpStructure, nlpOpts);
  }
  catch(const std::exception &e)
  {
    isOptimizerSetup_=false;
    std::stringstream ss;
    ss << "CasadiSXMultiShootingOptimizer::Setup() threw and exception when trying to
    create the optimization function: " << e.what();
    throw GvNmpcException(ss.str());
  }

  /* Setup optimizer argument map */
  CreateDefaultBounds();
  /* Make sure optimizer is marked as setup */
  isOptimizerSetup_ = true;
  return isOptimizerSetup_;
}
```

The *Setup* method is the only method the user has to implement in this example to have a fully working NMPC controller that can be interacted with in a wrapper object (typically a ROS node) using the *CasadiSXMultiShootingOptimizer* APIs. The method begins by setting up the model equations using the CasADi symbolic math primitive types. The code block below creates individual symbolics for the model states, inputs, and equations of motion. These symbolic variables are passed to the controller's encapsulated *CasadiSXModel*.

```
/* Model setup */
    // model states
    casadi::SX x = casadi::SX::sym("x");
    casadi::SX y = casadi::SX::sym("y");
    casadi::SX psi = casadi::SX::sym("psi");
    // model inputs
    casadi::SX v = casadi::SX::sym("v");
    casadi::SX psiDot = casadi::SX::sym("psiDot");
    // symbolic model
    casadi::SX xDot = casadi::SX(3,1);
    xDot(0,0) = v*cos(psi);
    xDot(1,0) = v*sin(psi);
    xDot(2,0) = psiDot;
    modelPtr_->Setup(vertcat(x,y,psi), vertcat(v,psiDot), xDot);
```

The next portion of the setup method creates some additional parameters that will be used in the process of each control iteration. The two parameters created in this example are the model's initial condition and the control reference. The initial condition parameter is used update the controller and start all the predictions with the most recent state feedback. The control reference parameter can be updated at any time to give the controller a trajectory to follow. Both parameters can be modified at run-time through the *CasadiSXMultiShootingOptimizer::SetParamValue* API.

```
/* Setup additional parameters (this->AddParam()) */
    casadi::SX x0 = casadi::SX::sym("x0", GetNumStates(), 1);
    AddParam("initialCondition", x0);
    casadi::SX ref = casadi::SX::sym("ref", GetPredictionHorizonSize(), 1);
```

```
    AddParam("ref", ref);
```

The next step of the controller setup is to create the prediction and control horizon variables and initialize the weighting matrices used in the cost function.

```
/* Setup control and prediciton horizon variables */
    CreateHorizons();

    /* Set state and input weights */
    SetStateWeight(opts_.stateWeight);
    SetInputWeight(opts_.inputWeight);
```

Because the horizons variables have been setup, the symbolic representation of the cost function can be constructed. This is achieved by iterating through each prediction on the horizon and using the encapsulated *CasadiSXCostFunction* object to add up each discrete cost. In the last iteration of the loop, a terminal cost is multiplied with the final cost value on the horizon.

```
/* Add up cost with costFunPtr_ */
    costFunPtr_->ResetCost();
    // Add up cost of initial condition
    costFunPtr_->AddCost(predictionHorizon_(casadi::Slice(0,GetNumStates()),0),
      casadi::SX::zeros(GetNumInputs(),1), ref(casadi::Slice(0, GetNumStates()),0));
    // Add up cost over the prediction horizon
    for (auto i=0; i < GetNumPredictions(); ++i)
    {
      const auto &state = predictionHorizon_(casadi::Slice((i+1)*GetNumStates(),
        (i+2)*GetNumStates()),0);
      const auto &input = controlHorizon_(casadi::Slice(i*GetNumInputs(),
        (i+1)*GetNumInputs()),0);
      const auto &target = ref(casadi::Slice((i+1)*GetNumStates(),
        (i+2)*GetNumStates()),0);
      // Check for terminal cost setup
      if (i == (GetNumPredictions()-1))
      {
```

```
        costFunPtr_->SetStateWeight(opts_.terminalCostWeight*
          std::dynamic_pointer_cast<CasadiSXQuadraticCostFunction>
            (costFunPtr_)->GetStateWeight()
        );
      }
      costFunPtr_->AddCost(state,input,target);
    }
```

The constraint equations are constructed and aggregated in the following step of the setup process. In this example, the only constraints to be implemented are the multi-shooting model-prediction constraints.

```
/* Setup constraints */
    // Model prediction constraint
    casadi::SX predictionVec = simulatorPtr_->SimulateInputsMS(predictionHorizon_,
      controlHorizon_);
    predictionVec(casadi::Slice(0, GetNumStates()),0) = x0;
    CasadiSXConstraint multiShootingConstraint(predictionHorizon_ - predictionVec);

    /* Concatenate constraints */
    constraints_={multiShootingConstraint /*,... And any other constraints created*/};
```

The final portion of the setup code required for the controller creates the handle to the IPOPT optimizer with the now defined optimization problem. This block also checks for errors in the solver setup process and sets some flags to ensure the setup process can be verified externally.

```
/* Setup NLP */
    casadi::SXDict nlpStructure{{"x",vertcat(predictionHorizon_,controlHorizon_)},
      {"p", params_}, {"f", costFunPtr_->GetCost()}, {"g",constraints_.Get()}};
    casadi::Dict nlpOpts;
    nlpOpts["ipopt.print_level"] = 0;
    nlpOpts["print_time"] = 0;
    nlpOpts["ipopt.acceptable_tol"] = 1e-8;
    nlpOpts["ipopt.max_iter"] = 2000;
```

```
nlpOpts["ipopt.constr_viol_tol"] = 1e-6;
nlpOpts["ipopt.acceptable_obj_change_tol"] = 1e-6;
nlpOpts["ipopt.fixed_variable_treatment"] = "relax_bounds";
try
{
  optimFun_ = casadi::nlpsol("solver", "ipopt", nlpStructure, nlpOpts);
}
catch(const std::exception &e)
{
  isOptimizerSetup_=false;
  std::stringstream ss;
  ss << "CasadiSXMultiShootingOptimizer::Setup() threw and exception when trying to
  create the optimization function: " << e.what();
  throw GvNmpcException(ss.str());
}

/* Setup optimizer argument map */
CreateDefaultBounds();
/* Make sure optimizer is marked as setup */
isOptimizerSetup_ = true;
return isOptimizerSetup_;
```

As previously mentioned, this completed controller can now be incorporated into a larger system. The controller can be constructed with different tunings (prediction horizon length and cost function tunings) without having to recompile or rework this code. The implemented setup method is only required to be called once and then the controller can be run at each iteration through the *CasadiSXMultiShootingOptimizer::Optimize* API or other similar public interfaces. These are the significant advantages of the NMPC software library demonstrated in this thesis.

Appendix C

NMPC Horizon Tuning Procedure and Results

Tuning for the NMPC Horizon begins with discretizing the number of predictions, prediction time steps, and speeds to generate a matrix of required tests. The starting 2-D grid of predictions vs. prediction time steps for the initial tuning of the kinematic model NMPC, presented in this work, is given below in Table C.1 The 2-D grid used to tune the bicycle model

Table C.1: 2-D Grid of Horizon Tuning Parameters for the Kinematic Model NMPC Implementation

Number of Predictions (N)	Prediction Time Steps (T)				
20	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
40	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
60	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
80	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s
100	0.10 s	0.25 s	0.50 s	0.75 s	1.00 s

NMPC is given in Table C.2. Each combination (N, T) results in a test that is run with a desired speed and desired reference path. A single test provides a lateral path error mean and standard deviation measurement. If the controller remains stable and controls to the path during the test, then the lateral path error mean result is relatively close to zero. If all test runs are zero mean, the average lateral path error does not provide any sensitivity information for the horizon tuning parameters. On the other hand, the standard deviation values for a well-tuned horizon will be significantly lower than an ill-tuned horizon. Therefore, searching for the minimum in a map of the lateral path error standard deviation results in the best candidate tuning parameters for a

Table C.2: 2-D Grid of Horizon Tuning Parameters for the Bicycle Model NMPC Implementation

Number of Predictions (N)	Prediction Time Steps (T)			
75	0.01 s	0.02 s	0.03 s	0.04 s
100	0.01 s	0.02 s	0.03 s	0.04 s
125	0.01 s	0.02 s	0.03 s	0.04 s
150	0.01 s	0.02 s	0.03 s	0.04 s

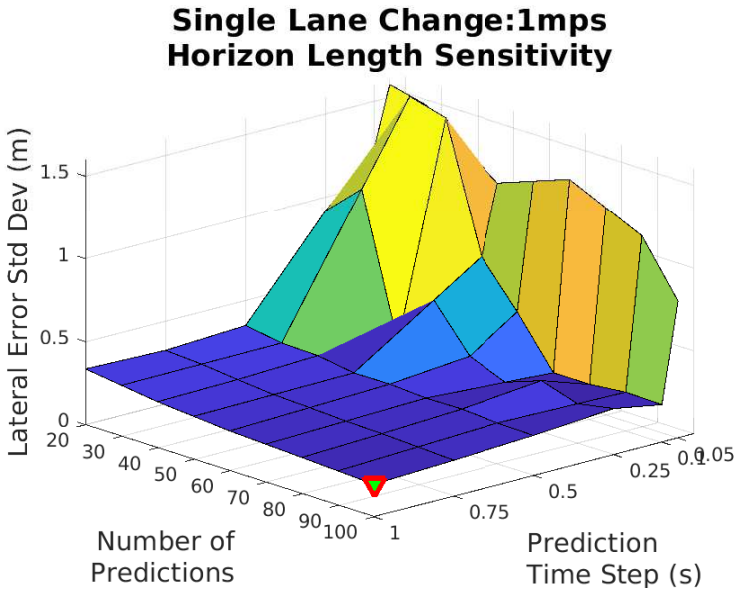


Figure C.1: Kinematic Model NMPC Horizon Tuning Map: 1 m/s

specific path and desired speed. If multiple reference paths are used, their standard deviation results may be combined on the tuning grid points.

The kinematic model NMPC implementation was tested in simulation using the single lane change reference path shown in Figure 5.5b at 1, 5, 10, 15, and 20 m/s. The results at each respective desired speed are shown in Figures C.1 – C.5 and the optimal horizon parameters are summarized in Table C.3.

Table C.3: Kinematic Model NMPC Optimal Horizon Tuning Parameters

Speed	Number of Predictions (N)	Prediction Time Step (T)
1 m/s	100	1.00
5 m/s	100	0.75
10 m/s	100	1.00
15 m/s	100	0.75
20 m/s	100	0.5

Each sensitivity map for the Kinematic NMPC shows the optimal horizon tuning with a downward red/green triangle over the corresponding (N, T) grid point. Not surprisingly, the results favor a larger number of predictions, but somewhat against first intuition, they do not favor short prediction time steps. Although the prediction fidelity degrades with larger prediction time steps, a longer horizon, H, seems to have greater impact on the control performance. It can also be seen in Figures C.1 – C.5 that the performance gain from increasing the number

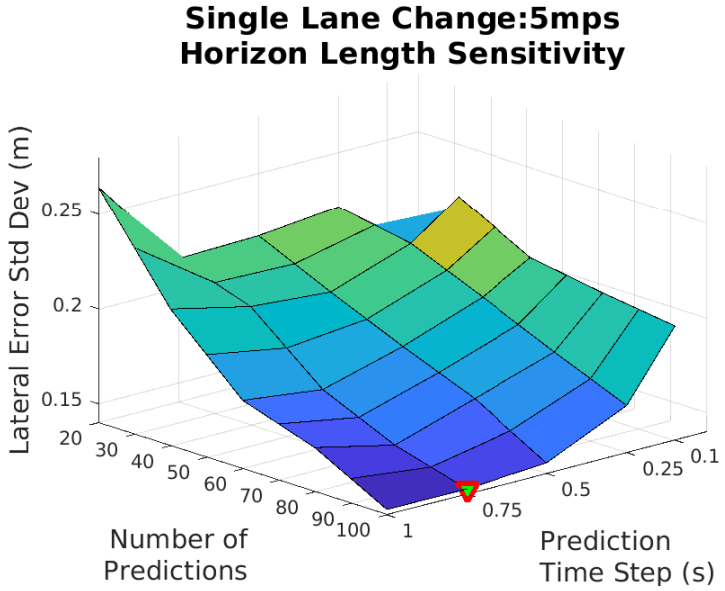


Figure C.2: Kinematic Model NMPC Horizon Tuning Map: 5 m/s

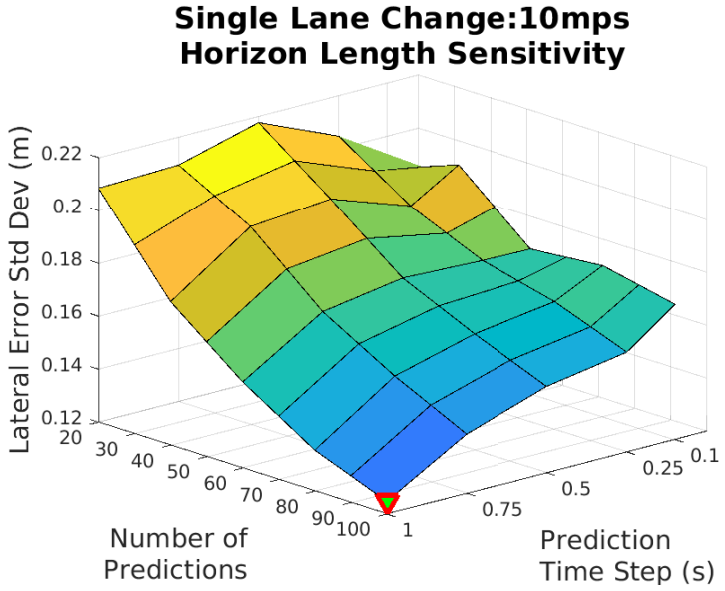


Figure C.3: Kinematic Model NMPC Horizon Tuning Map: 10 m/s

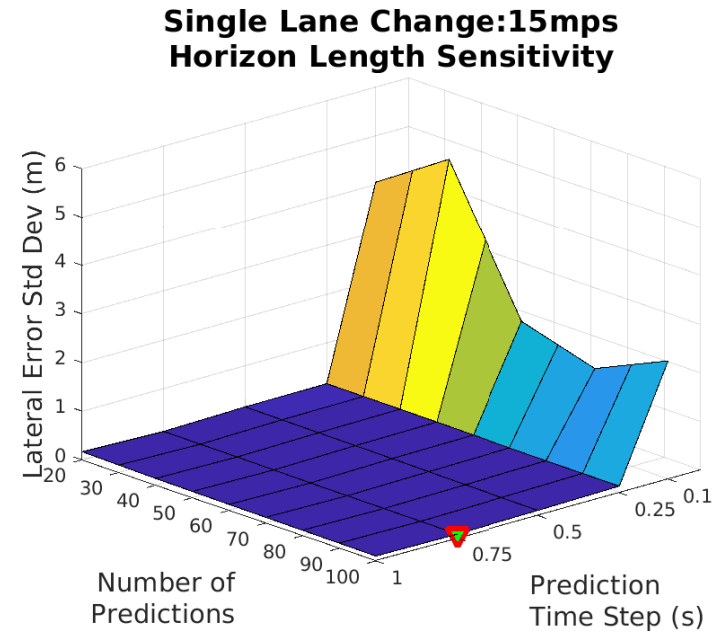


Figure C.4: Kinematic Model NMPC Horizon Tuning Map: 15 m/s

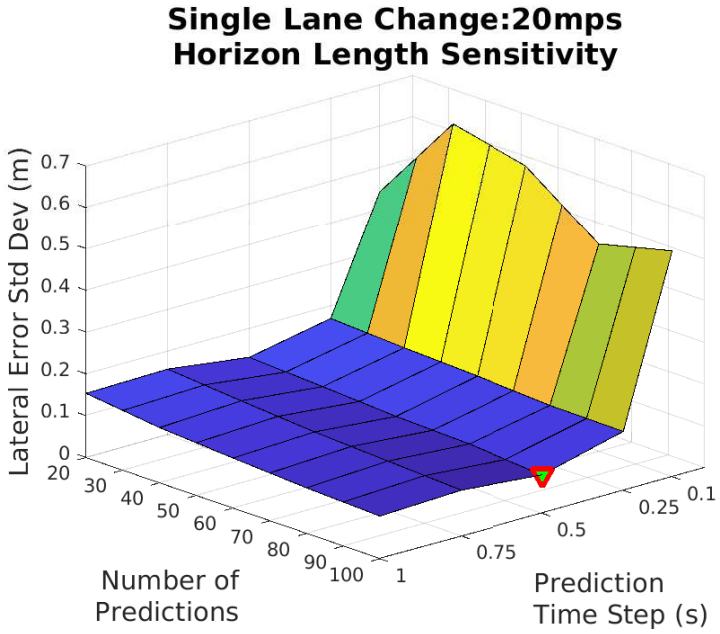


Figure C.5: Kinematic Model NMPC Horizon Tuning Map: 20 m/s

of predictions is minimal after approximately 60 predictions at each test speed. Conversely, the results show a very large performance gain when increasing the prediction time step. It must be noted that the drastic flattening in the graphs of the 15 and 20 m/s cases when increasing the time step from 0.1 s to 0.25 s was due to controller instability, further justifying the choice of a larger prediction time step. Given the summarized results in Table C.3, the optimal choice of horizon should be $(N = 100, T = 0.75s)$. However, given the constrained testing area for MKZ test vehicle, a suitable choice of parameters is $(N = 60, T = 0.75s)$.

The bicycle model NMPC implementation was also tested in simulation using the single lane change reference path at 1, 5, 10, 15, and 20 m/s. The results at each respective desired speed are shown in Figures C.6 – C.10 and the optimal horizon parameters are summarized in Table C.4. Each horizon tuning map for the bicycle model NMPC is plotted with a

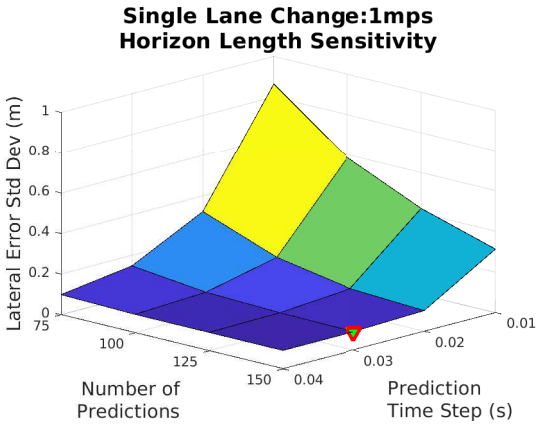


Figure C.6: Bicycle Model NMPC Horizon Tuning Map: 1 m/s

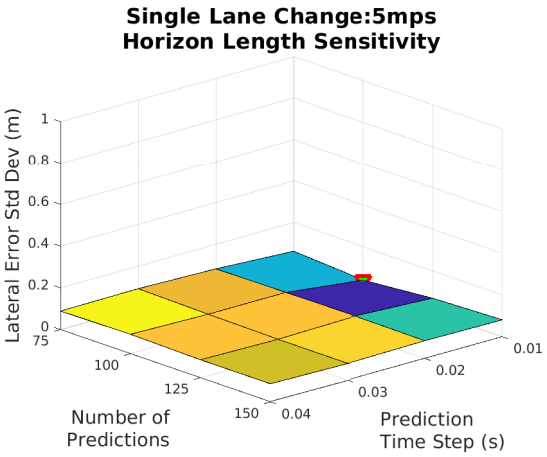


Figure C.7: Bicycle Model NMPC Horizon Tuning Map: 5 m/s

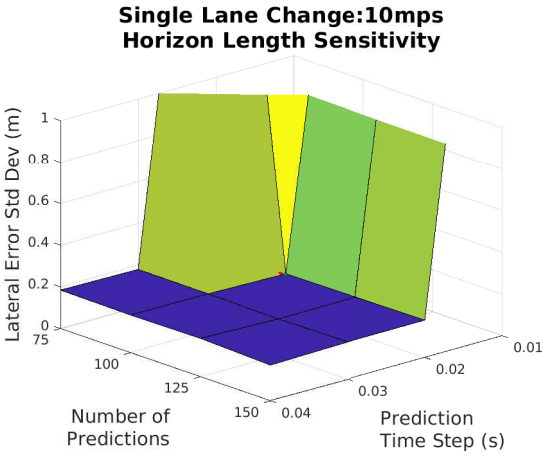


Figure C.8: Bicycle Model NMPC Horizon Tuning Map: 10 m/s

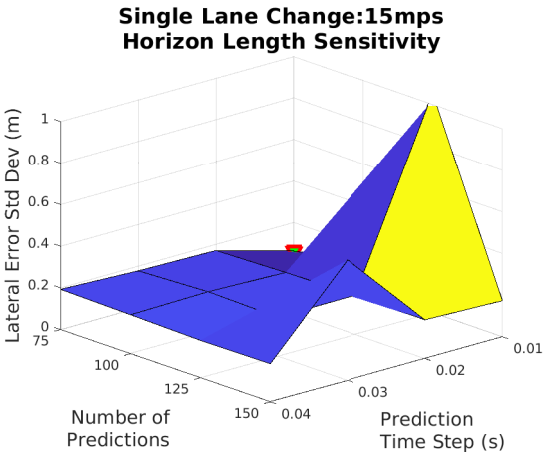


Figure C.9: Bicycle Model NMPC Horizon Tuning Map: 15 m/s

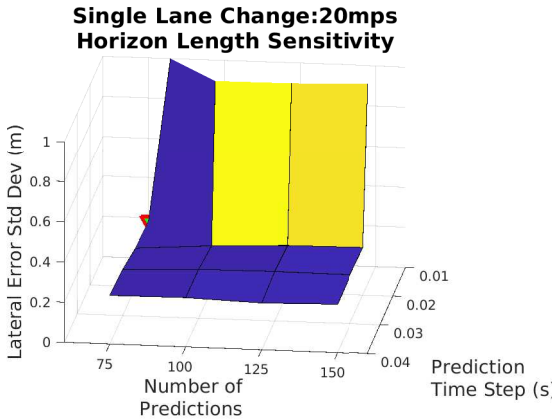


Figure C.10: Bicycle Model NMPC Horizon Tuning Map: 20 m/s

Table C.4: Bicycle Model NMPC Optimal Horizon Tuning Parameters

Speed	Number of Predictions (N)	Prediction Time Step (T)
1 m/s	150	0.03
5 m/s	100	0.01
10 m/s	100	0.02
15 m/s	75	0.01
20 m/s	75	0.02

limited range of $[0, 1]m$ on the standard deviation axis, so the results for different speeds can be easily compared. If the path error standard deviation value for a run on the map is above $1m$, the controller was not able to stabilize the vehicle's yaw through the maneuver. While each optimum tuning point suggests that a low prediction time step gives the best performance, the larger prediction time steps have more consistent performance at each horizon length. Given that the consistent performance of the grid point $(N = 100, T = 0.03)$ at each test speed, it seems a reasonable tuning selection that has a favorable run-time computational cost (compared to the other tuning grid points).