Title
Development and Testing of a Scale Autonomous Vehicle for Control Research

Author
Martin, George

Publication Date
2023

Peer reviewed|Thesis/dissertation

Development and Testing of a Scale Autonomous Vehicle for Control Research

By

GEORGE MARTIN
THESIS

Submitted in Partial Satisfaction of the Requirements for the Degree of

MASTER OF SCIENCE

in

Mechanical and Aerospace Engineering

in the

OFFICE OF GRADUATE STUDIES

of the

University Of California, Davis

Approved:

Francis Assadian, Chair

Shima Nazari

Iman Soltani

Committee in Charge

2023

ABSTRACT

Development and Testing of a Scale Autonomous Vehicle for Control Research

GEORGE MARTIN

This thesis will present the development and construction of a scale autonomous vehicle, along with its software development. The platform will serve as a means to test various lateral controllers and compare their results. Using Buckingham Pi Theorem, the platform will be scaled to achieve dynamic similitude with a standard electric vehicle, while utilizing common sensors found in modern autonomous vehicle systems. The software framework will be created using modern software packages, allowing for easy integration with a wide range of useful robotics tool kits and providing the ability to modify the car for various future applications. The study will examine the performance of a variety of control methodologies, with consideration for both their simulated and actual performance on the scaled car.

Acknowledgements

Funding

This project was generously funded through the Future Mobility Lab, in return the scale test bed has been provided as a platform for future rapid prototyping and research.

General Acknowledgements

I am extremely grateful for the support and advice of Dr. Francis Assadian in the development of controllers for this car as well as their practical implementations, as well as the advice of Ehsan Arasteh, who helped guide the development of the concept for this project. Additionally I would like to thank Trevor Vidano, Jonathan Dorsey, and Tristan Pham for their contributions to the code base of the Future Mobility Lab Autonomous Car.

Table of Contents

List of Tables

List of Figures

Dedication

To Jennifer, whose unwavering support made this work possible.

Chapter 1

Introduction and Literature Review

1.1 Introduction

The development of autonomous vehicles has been the subject of intensive research and development in recent years, with the aim of creating safer and more efficient transportation systems. One of the critical challenges in the design of AVs is the development of effective lateral control systems that can ensure their stable and safe operation in various driving scenarios. While simulations and computer modeling have been used extensively to evaluate lateral control strategies, physical experimentation is also essential to verify the effectiveness of these systems. However, it can be difficult to bridge the gap between simulation and testing on a full-size vehicle, which presents an urgent need for an efficient method to rapid-prototype controls before they are implemented at scale. This master's thesis focuses on the design, development, and testing of a scale car test bed to evaluate the lateral control strategies of autonomous vehicles. The test bed will be equipped with sensors, actuators, and a control system that can mimic the lateral behavior of an actual vehicle. Through a series of experiments, the test bed will be used to evaluate the performance of different lateral control strategies. The results of these experiments will be analyzed to determine the effectiveness of various lateral control strategies and identify the factors that impact their performance. The insights gained from these experiments can help inform the design and development of more robust and effective lateral control systems for autonomous vehicles. By leveraging physical experimentation on a scale car test bed, this thesis aims to contribute to the advancement of AV technology and provide a safe intermediary step before their implementation in real-world scenarios.

Modern control design generally begins with design of a controller, then simulation of this controller against a system model before testing on a physical system. These simulations can vary in fidelity from simple linear models to high fidelity, multi-physics simulations. Though there is a large difference between simulation and reality, there are many options to help bridge the gap, such as hardware-in-the-loop which provides real responses of full size components and subsystems during simulation. A scale test bed takes

this a step further, providing a realistic response of a full system reduced in scale, though it is necessary to ensure dynamic similitude between the scale and full-size models to guarantee that results from testing are meaningful. An approach for achieving this comparison will be elaborated upon within this thesis. An added benefit of both hardware-in-the-loop and scaled testbeds is that both configurations are also able to reveal the effect of real-time processing on the controller, as computational delay can significantly impact performance.

The development of a scaled test bed for controls testing within this thesis provides several long-term benefits. First, it provides a physical test bed which allows for testing on a system with computational delay. Additionally, this system uses real sensors with sensor noise, drift, and acquisition delay. Furthermore, the scale test car is able to, at minimum, behave in similar fashion to a full-size vehicle below a lateral acceleration of 3-4 $\frac{m}{s^2}$. Finally, this method allows the integration of a platform which is similar to a real vehicle control system, with localization, sensor fusion, safety protocols and other such features which allows for testing of the controller's true behavior when used in production. For the purposes of this thesis and for future research in the Future Mobility Lab, this vehicle is structured in such a way that controller algorithms can be integrated modularly, to compare their performance against a common computational infrastructure.

Control of a standard vehicle can be presented as several separate but related problem statements, from the control of subsystems such as brake actuation, to full chassis control. Regardless of formulation, one key feature of these systems is that they are underactuated for the task of autonomous trajectory following, in that the vehicle cannot move directly laterally, which complicates control design. For this thesis, the control problem for autonomous vehicles will be separated into longitudinal and lateral controllers. Longitudinal control encompasses speed, braking, cruise control, and other similar features of a standard vehicle. Lateral control then concerns itself with the steering of the vehicle. Within this work, testing of controllers will focus on the development of lateral controllers. As longitudinal and lateral control cannot be decoupled, and the performance of a longitudinal controller would affect the output of the lateral controller, testing will be performed at constant speed. However, the structure of the code base allows for the use of an integrated controller or separate longitudinal controller, enabling testing of any control configuration on the test bed.

Within this thesis, the development, construction and software development of a scale autonomous vehicle will be presented. This platform will be used to test a variety of lateral controllers, whose results will be compared. The platform will be scaled using Buckingham Pi Theorem to achieve dynamic similitude with a standard electric vehicle, and use common sensors found on modern autonomous vehicle systems such as an IMU, wheel speed sensor, and LIDAR. The software package will be built using Robot Operating System 2 (ROS2), which provides easy integration with a wide range of useful robotics software packages and provides the ability to modify the car to a large set of applications in the future. Controllers including pure pursuit,

Stanley, Youla, and MPC will be tested and results compared with consideration to their simulated and actual performance on the scaled car.

1.2 Literature Review

1.2.1 Autonomous Vehicles

The technologies supporting autonomous vehicles are rapidly advancing, and at an accelerating rate. Several degrees of autonomy have been classified into six levels by the Society of Automotive Engineers (10). The first three levels, zero to three, encompass technologies which already exist in the majority of new cars on the market, from cruise control to automatic braking and lane keep assist. The higher three levels, three to five, are considered degrees of autonomy. SAE definitions of the levels of autonomy are provided in Table 1.1. It is predicted that within and almost certainly by the end of this century, the highest level of autonomy will be reached (11). As these technologies grow in popularity, autonomous vehicles will become an increasingly common and fundamental part of many people's lives. Therefore, it is critical that any new advances are rigorously studied, to ensure their performance and the safety of both passengers and pedestrians.

Level 0	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Warnings and Momentary Assistance	Steering **OR** Brake Assistance	Steering **AND** Brake Assistance	Autonomy, Driver Takeover Required	Autonomy, No Driver Takeover	Full Autonomy
Automatic Emergency Braking	Adaptive Cruise Control or Lane Centering	Adaptive Cruise Control and Lane Centering	Traffic Chauffeur	Local Driverless Taxi	Global Driverless Taxi

Table 1.1: SAE Levels of Autonomy

The system architecture of an autonomous vehicle is often split into a few key subsystems. From the environment, data is captured using a variety of sensors, and a map of this environment is generated as well as the vehicle's relative position within it. This is called 'perception.' After this, the software must decide a course of action to navigate through the environment, which is called 'planning.' Finally, commands are given to the vehicle with the intent of following this course of action. These commands are then translated and sent to actuators, which maintain motion through the environment, which encompasses the 'control' of the vehicle. A diagram of this architecture is shown in Figure 1.1. For the purposes of this thesis, vehicle-to-vehicle communication will not be included but is possible to integrate in future research. The primary focus of the scale car development here will be the control subsystem, as it provides a basis for further development of higher level systems.

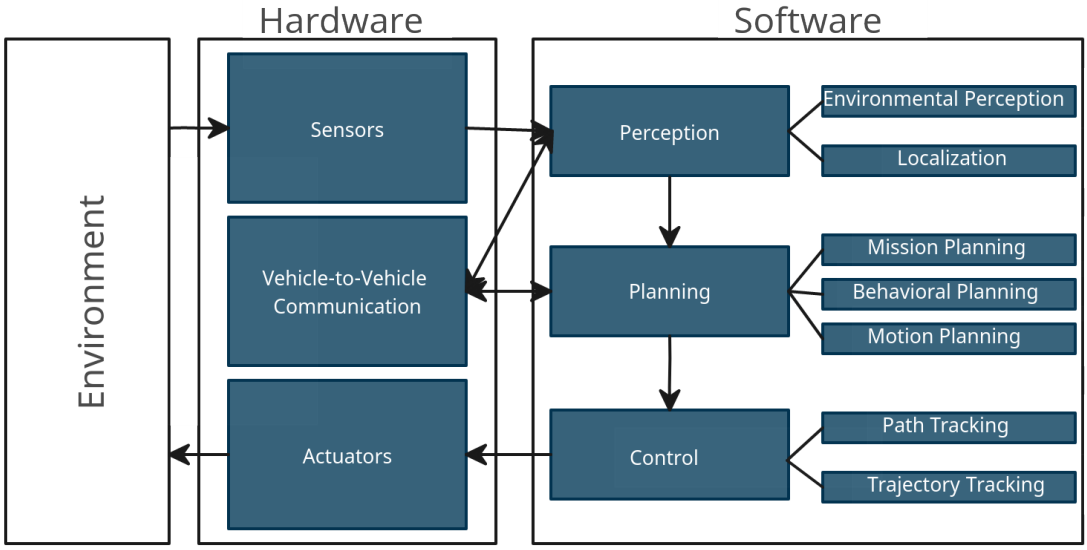


Figure 1.1: System Architecture of an Autonomous Vehicle (2)

Given that the emphasis of this project will be controls, perception and path planning will be integrated from external sources. Perception will be largely handled by a set of popular robotics tool kits, and path planning developed in the Future Mobility Lab prior to the development of the scale car (12). Therefore, beyond the physical construction and dynamic analysis of the test bed, the remainder of this thesis will focus on path following methodologies. Further, these control methodologies will be focused solely on the steering control of the vehicle, with speed control using a common PID control scheme widely used in industry.

1.2.2 Scale Car Dynamics

Several studies have been performed to validate the use of a scaled vehicle test bed in research applications. First, it was proven that dimensionless analysis could be used to develop a scale vehicle whose properties could be related to a full-size car (13). Then, this theory was applied to an off-the-shelf remote controlled car, and shown that the principles of dimensionless analysis held (14). Furthermore, the dimensionless analysis has been expanded and validations performed to show that trajectories observed by a scale test bed can behave in similar fashion to a full-size car (15). However, generally control development on a dynamically accurate model has been limited to studies of widely used control methodologies, with limited research involving implementations of modern control theory. Additionally, the structural design used to acquire data from these systems relies heavily on external localization systems such as dGPS and remote camera localization, which limits testing to a confined test location.

1.2.3 Scale Car Controls

Though controls testing on dynamically scaled test beds is limited, there are several developed architectures for software testing on a scaled vehicle. F1Tenth provides a platform for artificial intelligence and reinforcement learning-based autonomous vehicle development, and incorporates GPU acceleration to enable computationally demanding tasks (16). However, the design of this scale car is purely for the adaptation of an RC car, without consideration of the dynamic properties of the vehicle. A similar structure has been developed at the University of Washington (17), though with the same focus on software development and less interest given to scaled dynamics. Additionally, both structures require the use of a more expensive single board computer, sold by Nvidia. Given the rise in price of single board computers as a result of the Coronavirus Pandemic, the expense of a GPU-accelerated SBC has presented an additional hurdle to implementing these designs. Additionally, both architectures use Robot Operating System 1 (ROS1), the precursor to ROS2 which has released its final version and will no longer be supported after 2025. Therefore, it was decided to develop a new system architecture in ROS2, which will see continuous releases and be supported for the foreseeable future.

Given the breadth of research into scale car dynamics, and software development of autonomous RC cars, a clear possibility presents itself: the development of a software stack for autonomous research on a dynamically scaled vehicle. Furthermore, many systems require expensive sensor and compute units, which adds a roadblock to the adoption of scaled test bed research. To mitigate this, one of the primary design philosophies for the Future Mobility Lab Autonomous Car was the implementation of affordable components to develop a system which could perform all necessary functions for controls research in autonomous vehicles.

Chapter 2

Vehicle Modeling

2.1 Modeling Convention of Passenger Vehicles

System models used for control design attempt to use a minimal number of states required to accurately describe the behavior of the system. To this end, several key assumptions can be made to reduce this number of states for a vehicle model. A passenger vehicle is a complex system, with many moving parts and component dynamics. Each tire behaves differently in a turn, and inertial properties can be shifted through the suspension (3). Additionally, the behavior of a car turning at a high lateral acceleration can be very nonlinear, and system dynamics change depending on bank and pitch angle. However, for most controllers many of these effects can be ignored in modelling, and treated as a system disturbance.

It can be assumed that the car is moving in a two-dimensional space, which is generally valid for most use-cases. Additionally, for a lateral and longitudinal controller, suspension dynamics are also unnecessary except in extreme handling scenarios. This results in a car with no vertical motion, which can be completely described by its motion in the x-y plane, as shown in Figure 2.1.

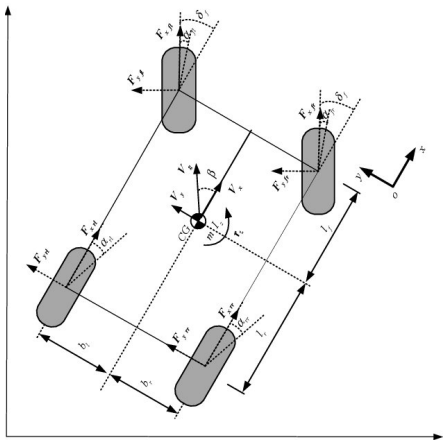


Figure 2.1: Two Track Vehicle Model (3)

This model, while reduced enough for some control applications, is still relatively complex, as forces on each tire deviate when turning. Another practical assumption is that front and rear tires can be modeled

as singular tires, generating the bicycle model. This assumption is valid for low lateral acceleration and for less aggressive maneuvers.

2.2 Kinematic Bicycle Model

For low lateral acceleration maneuvers, it is reasonable to assume that the car will move directly towards the direction of steer. In this case, tire slip angles are very near to zero and tire forces can be neglected. Therefore, when using the bicycle model of a vehicle, it can be assumed that the front wheel moves exactly in the direction in which it is pointed. The rear wheel simply moves forward and is strictly guided by the steering of the front wheel. A diagram of this model in a global coordinate frame is provided in Figure 2.2. It should be noted that for a standard passenger vehicle, the rear steering angle δ_r is always zero.

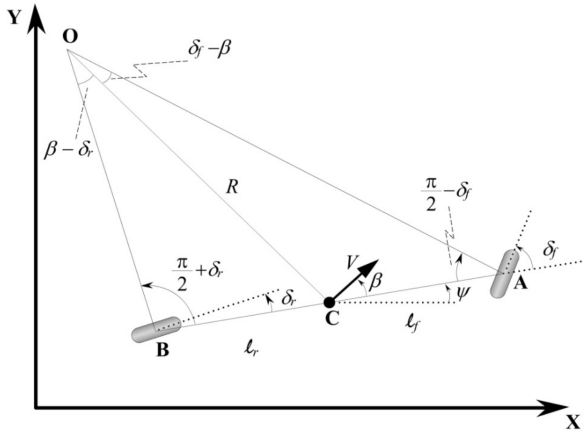


Figure 2.2: Kinematic Bicycle Model Schematic (4)

Using the kinematic relations of this model, the equations of motion in a global frame can be written as the following

$$\dot{X} = v\cos(\psi + \beta) \tag{2.1}$$

$$\dot{Y} = v\sin(\psi + \beta) \tag{2.2}$$

$$\dot{\psi} = \frac{v\cos(\beta)}{l}\tan(\delta_f) \tag{2.3}$$

$$\beta = \arctan(\frac{l_r}{l}tan(\delta_f)) \tag{2.4}$$

$$\tag{2.5}$$

where: X = x Position in Inertial Frame

Y = y Position in Inertial Frame

ψ = Inertial Yaw

v = Longitudinal Velocity

l_r = Rear Axle Distance to Center Mass

l = Wheelbase

δ_f = Steer Angle

This model is still nonlinear, as steering angle has a trigonometric relation to the motion of the car, but can easily be linearized about a neutral trim state. Though this model is reasonable for the design of a controller at low speed, it is completely unable to account for tire forces, and as such cannot consider road friction, drift, understeer or oversteer. These dynamics require a model of the tire, which is used to estimate slip angles of both front and rear tires. As this limits the design capability of a controller, it was decided to use a model which could account for the dynamic behavior exhibited by the tires.

2.3 Tire Dynamics

Before describing the model used for control development on the scale car, it is useful to describe a model of tire dynamics. At low speeds and in standard driving conditions, tires can be assumed to be rigid with perfect traction. However, tires in reality deflect, deform, and interface with the road in ways which are heavily dependant on conditions such as road surface, driving speed, steer angle, the mass distribution of the car and the characteristics of the tire itself. The source of tire force is the contact patch, which is the compressed section of the tire in direct contact with the road surface, as is illustrated in Figure 2.3. The friction interface of the tire on the road provides resistance to force applied by the mass momentum of the car, which can be decomposed into a longitudinal and lateral tire force.

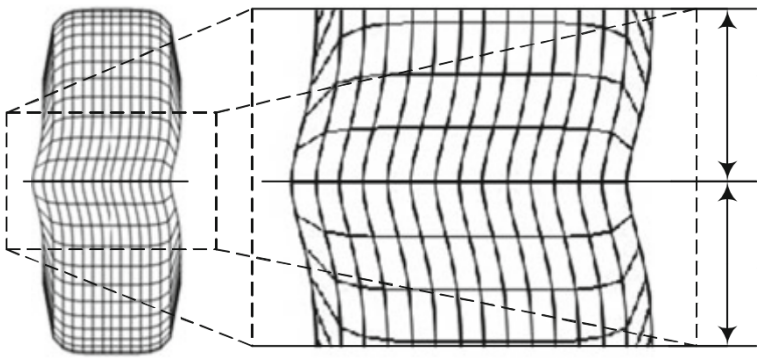


Figure 2.3: Dynamic Bicycle Model Schematic (5)

In the rigid kinematic model, the direction of tire motion is simply forward with respect to steer angle, with no lateral motion. However, with the deflection of the contact patch considered, the motion is actually in the direction of the vector sum of longitudinal and lateral tire forces. The difference between the kinematic and true direction of motion is called slip angle, and represents the direction of tire travel with respect to steering direction. With tire modeling and slip angles considered, understeer and oversteer behavior can be modeled. Also, the contact patch exhibits nearly linear lateral force behavior up to a certain lateral acceleration, and beyond this will begin to behave in a nonlinear manner and later saturate as the tire loses traction on the road. This saturation is exhibited in such phenomena as drifting.

For the purposes of this project, as control design is focused on steering at constant velocity, only lateral tire forces are considered. Various models of tire forces have been thoroughly researched and validated, with various advantages and disadvantages, such as the Dugoff and Fiala tire models (18). One of the most popular models in recent years is the Pacejka Magic Tire Formula, which semi-empirically derives tire forces using several key tire parameters(19). The equation for lateral tire forces for pure slip angles is presented in the following form

$$F_y = D\sin[C\arctan(B\Phi_{F_y})] + \Delta S_v \tag{2.6}$$

$$\Phi_{F_y} = (1-E)(\alpha + \Delta S_h) + (\frac{E}{B})\arctan[B(\alpha + \Delta S_h)] \tag{2.7}$$

where: B, C, D, E = Semi-Empirical Constants

α = Slip Angle

$\Delta S_v, \Delta S_h$ = Tire Characteristic Shifts

The semi-empirical constants as well as tire characteristic shifts are determined through a series of experiments, and do not have a strict analytical relation, but form a highly accurate model of tire behavior in steady state conditions. Though this model is quite complex, it can be linearized to the following form

$$C_\alpha = BCD \tag{2.8}$$

where: C_α = Cornering Stiffness

Thus, a simple model of tire lateral forces can be used to augment the bicycle model and develop a higher fidelity dynamic model. This final model will be implemented in the design of controllers for the car. The parameters for the scale car itself are discussed in a later section, as tire behavior is critical to understanding the dynamics of a car beyond the range acceptable for kinematic modeling.

2.4 Dynamic Bicycle Model

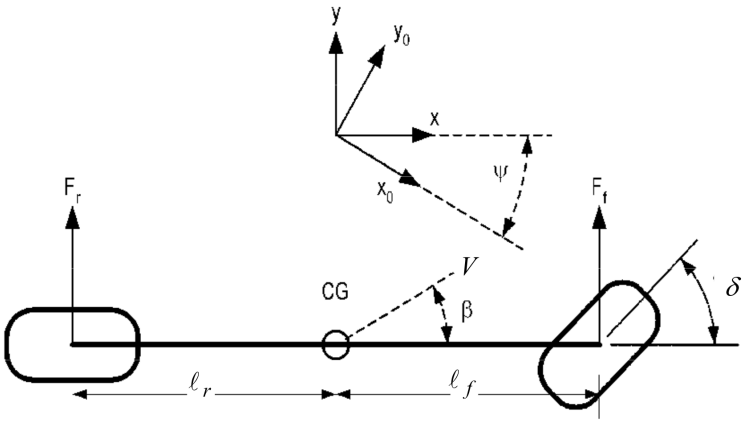


Figure 2.4: Dynamic Bicycle Model Schematic (4)

With tire forces modeled, the kinematic bicycle model can be modified as shown in Figure 2.4, in which lateral tire forces are included. Additionally, the model can be linearized and presented in error coordinates, which is useful for control development and is presented in state-space form as follows

$$\frac{d}{dt}\begin{bmatrix} e_{lat} \\ \dot{e}_{lat} \\ e_{\psi} \\ \dot{e}_{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -\frac{2C_{\alpha f}+2C_{\alpha r}}{mV_x} & \frac{2C_{\alpha f}+2C_{\alpha r}}{m} & \frac{-2C_{\alpha f}l_f+2C_{\alpha r}l_r}{mV_x} \\ 0 & 0 & 0 & 1 \\ 0 & -\frac{2C_{\alpha f}l_f+2C_{\alpha r}l_r}{I_zV_x} & \frac{2C_{\alpha f}l_f+2C_{\alpha r}l_r}{Iz} & \frac{-2C_{\alpha f}l_f^2+2C_{\alpha r}l_r^2}{I_zV_x} \end{bmatrix} \begin{bmatrix} e_{lat} \\ \dot{e}_{lat} \\ e_{\psi} \\ \dot{e}_{\psi} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{2C_{\alpha f}}{m} \\ 0 \\ \frac{2C_{\alpha f}l_f}{I_z} \end{bmatrix} \delta \tag{2.9}$$

$$\dot{y} = Cx + Du \tag{2.10}$$

where: $C_{\alpha r}, C_{\alpha r}$ = Front and Rear Cornering Stiffness

l_f, l_r = Front and Rear Axle Distances to Center Mass

V_x = Longitudinal Velocity

m = Mass of Vehicle

I_z = Mass Moment of Inertia of Vehicle

C, D = Matrices which can be selected depending on the control application

In this formulation, the C matrix can be formed by selecting zero for unregulated error, and one for controlled error states while D should be held as zero. For example, a controller which considers only lateral error would have a C matrix of $[\,1\ 0\ 0\ 0\,]$ and a D Matrix of 0.

Chapter 3

Scale Car Build

3.1 Dimensionless Analysis for Scale Test Bed Development

Scale testing requires the development of a vehicle model, with a given aspect ratio to the full-size model. This vehicle must be dynamically similar to its larger counterpart. However, it is not as simple as reducing each dimension by the same aspect ratio. Certain characteristics are warped at different rates comparatively, and it is necessary to keep these proportionate characteristics identical, more so than maintaining the specific static ratio. Buckingham Pi Theorem (20) provides a meaningful way to analyze scaled dynamics, with an added benefit of reducing the number of comparative parameters. This theorem states that for a given linear system, there exists a set number of dimensionless parameters n which fully describe the dynamics of a system. These parameters, being dimensionless, therefore can be used to make a scale model with identical parameters, that achieved dynamic similarity with the original model. The number of parameters is reduced by the total number of base units included in the formulation. As these parameters are dimensionless, they hold true for any ratio or scaling of the model vehicle. In order to generate these dimensionless parameters, it is critical to deconstruct parameters to their fundamental units. For example, an acceleration should be written in component units such as $\frac{m}{s^2}$, which have base units of $[\mathrm{L}][\mathrm{T}]^{-2}$. Though any set of base units can be used in the process of dimensionless parameter generation, the two most common are MLT and FLT, which correspond to mass-length-time and force-length-time, respectively. The goal when generating these parameters is to incorporate all parameters in such a way that the base units cancel, leaving only dimensionless key parameters known as Pi groups.

Using the linear dynamic bicycle model, these parameters become the front axle distance to center of gravity l_f, rear axle distance to center of gravity l_r, wheelbase l, front cornering stiffness C_f, rear cornering stiffness C_r, longitudinal speed v_x, mass m, and rotational inertia I_z. The units and base units for these parameters are shown in Table 3.1. Using Buckingham Pi Theorem, these parameters are sufficient to develop Pi groups (13). With these eight parameters, the number of Pi groups needed is reduced by three, as all three base units of the MLT formulation are included, leaving a total of five groups.

Parameter	Units	Base Units
l_f	m	[L]
l_r	m	[L]
l	m	[L]
C_f	$\frac{kgm}{s^2}$	[M][L][T]$^{-2}$
C_f	$\frac{kgm}{s^2}$	[M][L][T]$^{-2}$
v_x	$\frac{m}{s}$	[L][T]$^{-1}$
m	kg	[M]
I_z	kgm^2	[M][L]^2

Table 3.1: Key Parameters of Bicycle Model, with Units and Base Units

$$n_{pi} = n_{parameters} - n_{unique\ base\ units} = 8 - 3 = 5 \tag{3.1}$$

The Pi groups are then determined as the following.

$$\Pi_1 = \frac{l_f}{l} \tag{3.2}$$

$$\Pi_2 = \frac{l_r}{l} \tag{3.3}$$

$$\Pi_3 = \frac{C_f l}{m v^2} \tag{3.4}$$

$$\Pi_4 = \frac{C_r l}{m v^2} \tag{3.5}$$

$$\Pi_5 = \frac{I_z}{m l^2} \tag{3.6}$$

However, this set of Pi groups only concerns itself with the dynamic behavior of the vehicle. In order to compare the error of the lateral controller at scale, it is necessary to expand the Pi groups to include positional scaling. This study has been previously performed, and the additional groups identified using additional parameters yaw ψ, steering angle δ, x-position x, and y-position y (15).

$$\Pi_6 = \psi \tag{3.7}$$

$$\Pi_7 = \delta \tag{3.8}$$

$$\Pi_8 = \frac{x}{l} \tag{3.9}$$

$$\Pi_9 = \frac{y}{l} \tag{3.10}$$

Though seemingly trivial, the inherent lack of dimension in yaw angle and position show that the steering commands of cars and car-like robots are comparable at any scaling, and that the x and y position of the

vehicle is directly scaled by the aspect ratio, which reaffirms the idea that analysis at scale will provide meaningful results to a controller meant for implementation on larger vehicle.

Pi groups can then be calculated for a full size vehicle using the parameters listed in Table 3.3. Then, a scale vehicle can be modified in order to align its own Pi group values to the larger vehicle, thereby guaranteeing dynamic similarity.

3.2 Scale Dynamics

As described in the previous section, there are several key parameters to be identified in order to correlate the scale vehicle dynamics to a full-size car. While certain parameters are static or difficult to modify and align with the dimensionless parameters of a car, others are relatively easy to manipulate in order to ensure a good agreement of the car's dynamic properties. The center of gravity, and by extension ratio of distances from the center of gravity to axles, can be manipulated by shifting component positions on the scale car. The inertia can also be altered in similar fashion by varying the mass distributions of the components. Furthermore, it is entirely feasible to add small weights to the car as necessary to vary both these parameters as well as the total mass directly. This leaves velocity and cornering stiffness to be determined. Conveniently, these numbers are correlated with Pi groups three and four, which means that test speed of the lateral controller can be varied, given a prescribed cornering stiffness to produce useful results.

One key feature of an electric vehicle is that its center of gravity is much closer to the midpoint between each axle than a conventional vehicle with an internal combustion engine. With this known, it becomes much easier to make the first four groups align. The first two groups essentially describe the position of the center of mass, in relation to the wheelbase, which is easy to modify to achieve similarity. The third and fourth groups become trivial as well, as the cornering stiffnesses of an idealized car, with center of gravity precisely at this midpoint between axles, become identical. This means that, for a given physical model, forward speed can be varied and provide sufficient flexibility to achieve similitude. The fifth and final group would be the most difficult in this case to guarantee, and may require some adjustments of component positions, while maintaining the position of the center of gravity, to achieve similarity. Therefore, a practical method to achieve total similarity of the five Pi groups is to modify the dimensionless parameters of the scale car in ascending order.

3.2.1 Weight, Center of Mass, and Axle Distances

Wheelbase is strictly provided as a parameter of the 1:7 scale car purchased from Traxxas. This parameter has a value of 0.404 m. The center of mass was then calculated by force balance, by measuring the weights

over each axle using four scales with a measuring precision of 0.1 g. The measured mass of the car is 5.568 kg. Similarity was achieved by adjusting the position of the battery packs used to power the computation units of the car, which required minimal effort.

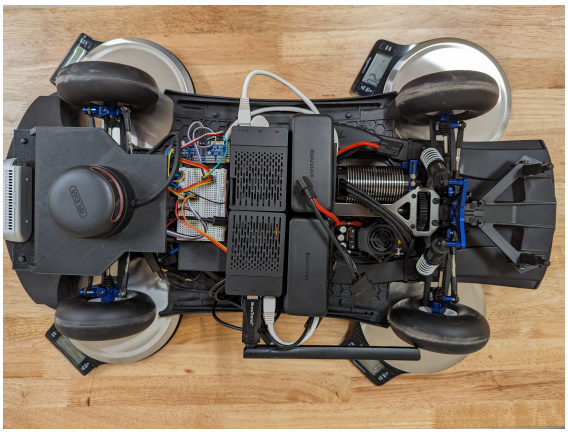


Figure 3.1: Scale Car on Four Weight Scales, for Force Balance Calculations

The force and moment balance equations are relatively straightforward, using static equations and knowledge of the wheelbase l, car total mass m_{car}, and normal force at each axle, $F_{rearaxle}$ and $F_{frontaxle}$ to determine the front and rear axle distances, l_f and l_r.

$$\Sigma F = F_{rearaxle} + F_{frontaxle} - m_{car}g \tag{3.11}$$

$$\Sigma M_{rearaxle} = F_{frontaxle}l - l_r m_{car}g \tag{3.12}$$

$$l = l_f + l_r \tag{3.13}$$

Using this system of equations, the values of l_f and l_r can be trivially derived as 0.205 m and .0199 m, respectively.

3.2.2 Cornering Stiffness

Cornering stiffnesses present a unique challenge in the design of a scale car. First, it must be shown that the behavior of a scaled tire itself is similar to that of a standard tire, in addition to the requirement to achieve similarity in the third and fourth Pi groups. As this would be a large undertaking, requiring the creation of a test apparatus, it was decided that a preferable alternative would be to use a tire which had already been thoroughly analyzed. Fortunately, this work has already been done for a select line of DuBro pneumatic tires. Intended for use with remote controlled aircraft, these tires have been shown to have good agreement with a Pacejka tire model (1). Additionally, published test data showing the Pacejka constants for these tires

have been provided, as shown for the 4 $\frac{1}{2}$" tires in Table 3.2. In order to calculate the cornering stiffnesses, the model was linearized about a neutral trim state, which is shown in Equation 2.8. Since the results of tire testing are presented as a table in incremental speeds and tire loads, the value of the cornering stiffness was interpolated between cornering stiffnesses at each setpoint, to determine the parameters for both the front and rear tires.

Pacejka Constants for 4 $\frac{1}{2}$" Du-Bro Tires at Varying Loads						
F_N	B	C	D	E	ΔS_v	ΔS_h
20	0.132	1.30	21.30	0.04	0.06	-0.59
30	0.112	1.23	31.14	-0.62	1.44	-0.67
40	0.104	1.15	38.77	-0.99	3.15	-0.90
50	0.079	1.18	50.98	-1.71	3.64	-0.99
60	0.065	1.35	57.45	-1.05	3.66	-0.92
70	0.057	1.43	62.86	-1.17	3.06	-0.82

Table 3.2: Tire Parameters for Du-Bro 450TV Pneumatic Tires, Courtesy of UIUC (1)

Using these parameters, the front and rear cornering stiffness can be interpolated. Given the axle loads, it can be assumed that each tire carries half the weight, so the cornering stiffness of each tire can be directly interpolated, then doubled to determine a total front and rear cornering stiffness, resulting in a $C_{\alpha f}$ of 6.932 $\frac{N}{rad}$ and $C_{\alpha r}$ of 6.918 $\frac{N}{rad}$.

3.2.3 Moment of Inertia

The rotational moment of inertia for the scale car requires more effort than the previous parameters. One simple approximation would be to assume that weight is centered above each axle, and could be calculated using the measurements taken when determining the center of mass. This assumption can be used to trivially simplify the mass moment of inertia equation to the following equation.

$$I_z = m l_f l_r \tag{3.14}$$

where: m = Mass of the Vehicle

l_f = Front Axle Distance to Center of Mass

l_r = Rear Axle Distance to Center of Mass

However, this is not a very precise measurement, and in comparison to the precision of the other identified parameters, is severely lacking in both accuracy and verifiability. One method that would provide a higher accuracy estimate would be the modeling of each component and assembly in a 3D CAD software, using measured masses of each component and allowing the software to identify the center of mass through a brute force calculation. This would be reasonable, but leaves room for error in modeling, compounding error due to

calibration, and requires a significant time investment. Ultimately, it was decided to measure the rotational inertia of the car using a bifilar pendulum. This pendulum is used widely in industry to find the rotational inertia of irregular objects, and provides a practical method to derive the parameter for the scale car. The pendulum is anchored at two points, and the test specimen mounted to a rigid structure connecting these anchors. The pendulum is then moved out of equilibrium, by twisting it slightly. The period of oscillation for this pendulum is then related to the combined moment of inertia of the rigid structure and test specimen. Using a structure with simple geometry allows for easy calibration and validation of the pendulum itself. This inertia can then be subtracted from the total measured inertia to isolate the scale car's rotational inertia.

$$I_z = \frac{d^2 m g T^2}{16\pi^2 l} \tag{3.15}$$

where: d = Distance Between Anchors

m = Mass of the Rotating System

g = Gravitational Constant

T = Period of Oscillation

l = Length of the Tension Cables

A pendulum was made using two pieces of planed two-by-four lumber which were selected to have no aberrations or knots in the wood, with the assumption that this would maximize the homogeneity of the wood and greatly simplify inertia calculations. The lumber sections were connected by two equal lengths of nylon cord, and one section of lumber clamped rigidly to a sturdy table.

Figure 3.2: Bifilar Pendulum with Scale Car Mounted, for Moment of Inertia Calculations

After assembly, the pendulum was oscillated and its period calculated. The resultant moment of inertia

measurement was in good agreement with the theoretical calculation. The measured value was chosen as the true value, as the calculation did not account for the added mass of the mounting hardware or any other marginal inconsistencies in the lumber. At this point, the scale car was mounted to the rigid structure, taking care to position the center of masses of both objects directly in line with the midpoint between the anchor points. Oscillation period was then recorded. Using the parameters of this test setup, the inertia is calculated as follows, subtracting the predetermined inertia of the wooden mounting block.

$$I_z = \frac{0.864^2(5.568 + 1.660)g(0.82)^2}{16\pi^2(0.80)} - 0.1146kgm^2 = 0.167kgm^2 \tag{3.16}$$

3.2.4 Dimensionless Parameters of the Scale Car

Using the parameters calculated in the previous section, dimensionless parameters can be formed and compared to a full-size car. To provide a comparison, vehicle parameters used in the study of an in-wheel electric vehicle are used as reference (21). These parameters provide a generic model for an electric vehicle, and although no specific model is provided gives a good estimate of a standard model for a car of this type. Furthermore, if provided a specific model, the scale car is capable of being modified by adjusting weight distribution and tuning vehicle speed to match a given vehicle's properties, allowing for additional flexibility in future research.

Parameter	Standard EV	Scale Car
l_f $[m]$	1.000	0.205
l_r $[m]$	1.000	0.199
w $[m]$	2.000	0.404
C_f $[\frac{N}{rad}]$	22000	6.932
C_r $[\frac{N}{rad}]$	25000	6.918
v $[\frac{m}{s}]$	7.000	1.000
m $[kg]$	900.0	5.568
I_z $[kgm^2]$	1200.0	0.167

Table 3.3: Comparison of Full-Size and Scale Car Parameters

Using these parameters, Pi groups can be generated and compared, as shown in Table 3.4. The Pi Groups show very good agreement, with a maximum deviation of 13.4% of the fourth group, while the first three groups remain within 1% tolerance. For a standard vehicle model, this shows sufficient agreement, given the variance in possible parameters shown in the NHTSA database far exceeds the deviation of the scale car in some vehicle models (22).

Π Group	Standard EV	Scale Car
Π_1	0.500	0.508
Π_2	0.500	0.492
Π_3	0.998	1.003
Π_4	1.134	1.001
Π_5	0.333	0.366

Table 3.4: Comparison of Scale and Full-Size Pi Groups

3.3 Configuration

The car was designed with a goal of providing an affordable, open-source test platform that could still utilize modern technologies and be capable of performing any necessary testing maneuver and provide dynamic similarity with a standard electric vehicle. To this effect, it was decided that the vehicle would be built upon a standard remote-controlled 1:7 scale car, the Traxxas XO-1. This vehicle comes equipped with a drive motor, electronic speed controller, and steering servo with an Ackermann steering design. Additionally, the vehicle has both a front and rear differential, as well as spring and damper suspension system which allows for a more realistic behavior when driving. In the future, this also provides an opportunity to generate a higher fidelity model of the car for testing of more complex control designs.

The car is also equipped with a set of sensors which are commonly found in autonomous vehicles: a wheel-speed sensor, inertial measurement unit (IMU), and light detecting and ranging unit (LIDAR). Additionally, a depth camera has been mounted to the car, and while it is out of scope and unused in this thesis, there are existing software suites that can perform sensor fusion between the LIDAR and depth camera. The wheels-speed sensor is an AS5600 magnetic encoder, which uses the Hall effect to determine the angular position of a magnet mounted on the front differential of the car. The IMU is a standard MPU6050, an affordable sensor with six degrees of freedom, providing linear accelerations and angular velocities. The LIDAR is an RPLIDAR A2M8, which refreshes a two-dimensional map at a rate of ten full scans per second, and scans pixels at a rate of eight thousand per second. The included depth camera is an Intel Realsense D435i. Mounting brackets were 3D printed for each sensor, and will be provided on the Future Mobility Lab GitHub repository. To control the ESC and servo motor, a PWM I2C breakout board was added, which contains a real-time clock and permits precise control of both motors mounted on the car.

One unique aspect of this project is the inclusion of two separate compute units, which handle localization and control separately. This allows for future flexibility in the project, leaving additional computational headroom for more advanced software and the inclusion of additional features such as obstacle detection and avoidance. Two Raspberry Pi 4b single board computers were mounted on the car, each equipped with an individual battery. Tests were performed with both the 4 GB and 8 GB models, and in this case

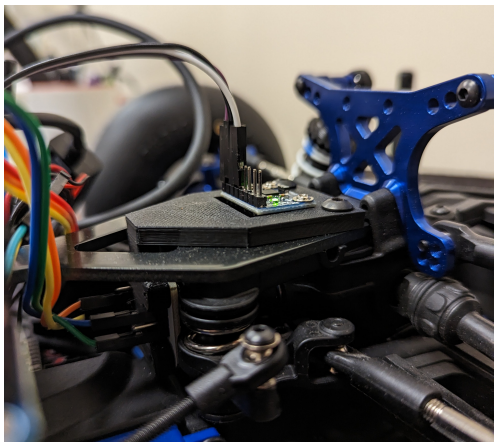


Figure 3.3: IMU and Encoder Mounted on the Scale Car

both possessed adequate memory to perform the required tasks. An overview of the sensor integration and interconnection of compute units is shown in Figure 3.4.

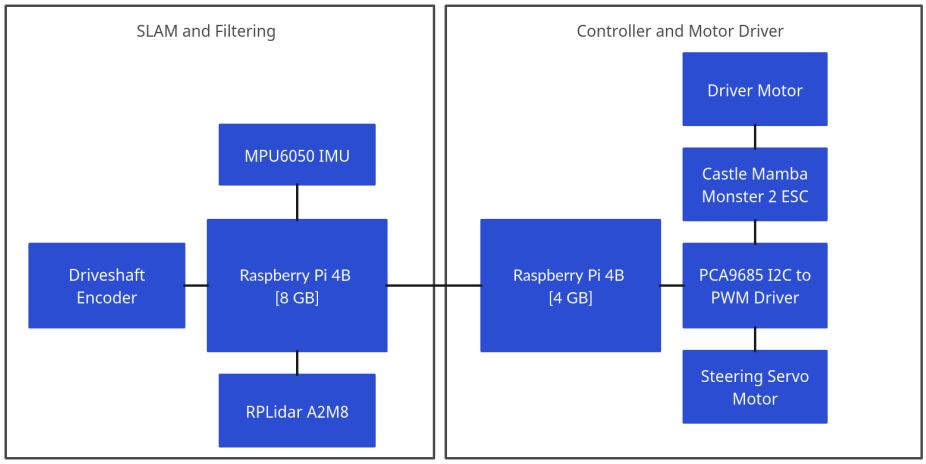


Figure 3.4: Physical Component Configuration, Sensors and Compute Units

Communication from the scale car to a ground station is also critical. To enable remote control, start up and safe takeover during failure, a network connection was established between the localization unit and a ground station, using a local area network. Additionally, an alternate configuration was made for outside use in which the ground station serves as the router, generating a Wi-Fi hotspot for the localization unit to connect to. A visual representation of this network configuration is shown in Figure 3.5. With this configuration, latency between the onboard units is an average 0.33 ms, while the latency from the ground station to car is 164 ms. Speed is only critical between the compute units, as messages are rapidly transferred to provide pose estimates to the controller, and as such this configuration was deemed sufficient.

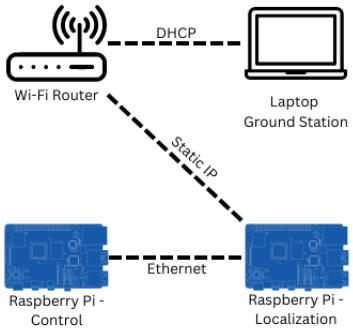


Figure 3.5: Computer Networking Between Scale Car and Ground Station

3.3.1 Software Stack

The codebase for this project was developed using Robot Operating System 2 - Humble Hawksbill (23). ROS2 is an open-source software development kit that provides a set of libraries, interfaces, and other utilities for robotics development. It was chosen as the platform for the development of the scale car due to its ease of implementation, modular functionality, and popularity within the robotics community. The version, Humble Hawksbill, was selected as it is the first version of ROS2 with a long-term support (LTS) of five years. This means that the platform will be supported, provided critical updates, and likely see active development until mid-2027. This will prove incredibly useful to future research using this test bed, as it enables the implementation of state-of-the-art software packages without excessive effort. Packages such as Robot Localization, SLAM Toolbox and Nav2 are consistently updated and optimized for performance on embedded systems, providing simple APIs and configurations for tuning on the scale car.

ROS2 uses a publisher-subscriber protocol, in which modular programs can run independently, subscribing to topics with pertinent information, while publishing their own outputs asynchronously. this means that sensor messages, localization, sensor fusion, and control can all be handled in separate programs while communicating through a unified protocol. Additionally, this format allows for the exchange of information across multiple devices over a network. The two onboard Raspberry Pi single board computers communicate via an Ethernet cable, while topics are transmitted over Wi-Fi for viewing on a remote computer.

The distributed structure of ROS2 means that programs can be organized into nodes, which communicate through the previously described protocol and together perform the sum of a robot's functions. These nodes are further organized into packages which contain sets of interrelated nodes. For example, the control architecture of the scale car is split into two packages - one low level and one high level. The high level package has customizeable nodes for each controller type, and the low level package contains a servo motor and drive motor control node. The distributed nature of this system lends itself to the use of callbacks. The

high level controller has a callback which is triggered whenever the robot's position is updated, the low level controller has a steering callback which is triggered whenever a new control command is published, and a driver motor control loop has a direct signal callback which is triggered whenever a new velocity reading or command is provided to it.

One package of considerable influence to this project is Nav2 (24). This package contains a complete toolbox for use in the navigation of mobile robots, from behavior trees, to localization, to control. However, the control structure is more heavily tailored to either omni-wheel or dual-motor controlled robots, which are structurally different from a car-like robot with Ackermann steering. Therefore, the entire path planning, control, and sensor integration libraries were written exclusively for the scale car, but Nav2 message types and localization was incorporated to provide further flexibility in future research, such as obstacle avoidance.

An important consideration in robotics design is the relationship of the various coordinate systems of the robot. Each sensor has its own respective coordinate system which must be respected, and which relates to the coordinate system of the robot to be controlled. For example, an IMU may be mounted at an angle or offset from the center of the robot, and as such its recorded accelerations will not immediately align with the robots own accelerations. To account for this, transforms are published between each sensor's coordinate frame and a base link frame of the robot. For the scale car, this is shown in Figure 3.6. Using these transforms, all sensor values can be merged into a unified model of the behavior of the robot with respect to any pertinent reference frame. All sensors except for the wheel speed sensor are provided coordinate transforms, as the wheel speed sensor's output is assumed to share longitudinal velocity with the base link of the car.

Figure 3.6: Coordinate Frames Provided in ROS, Compared to Physical Car

In addition to providing transforms between the coordinate frames of the robot, ROS2 also expects a transform between the base link of the robot and its odometry estimate, as well as the odometry estimate and the map in which the robot is navigating. To achieve these requirements, the raw sensor values are fused with an extended Kalman filter, and the transform of the robots pose to the map is provided by either a graphical SLAM package or Adaptive Monte Carlo Localization (AMCL) package. All transforms

are connected in a transformation tree, which enables all data published by ROS2 to be provided to each sensor with a complete transform from point to point. For example, the IMU reading can be transformed to mathematically relate it to the acceleration of the robot with respect to the map coordinate frame. The bi-directional transformation tree for the scale car is shown in Figure 3.7, with arrow indicating forward transformation.

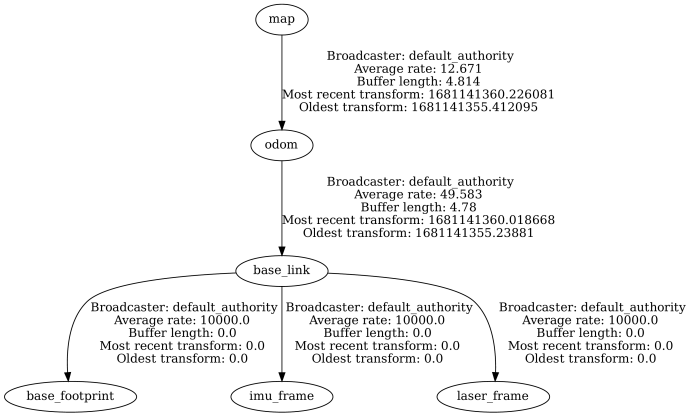


Figure 3.7: Transformation Tree of the Scale Car

The distributed structure, transform tree, and modular nature of ROS2 enables the separation of subsystems to handle the various functions of the scale car during operation. The following subsections will elaborate on the differentiation of functionality and responsibility of each package, from the basic system functions to high level controls.

3.3.1.1 Low Level Drivers

In order to maintain the modularity of the software stack, and enable the use of new sensor or low-level control interfaces, sensors and motor controllers were subdivided into independent packages, each interfacing from an I2C (Inter-Integrated Circuit) protocol to ROS2 topics. The IMU and wheel encoder packages are publishers which read in raw data and publish inertial frame accelerations and wheel velocities, which are the provided to other nodes as needed. The motor and servo controller were merged, as both use a single breakout board over the I2C interface. This low level controller sends direct servo steering angle commands, translating from desired steering angle to motor angle. The drive motor is regulated by a PI loop which takes raw velocity readings and outputs a throttle value. Additionally, a feed forward component was added for steering angle to prevent excessive velocity losses when turning. The control design is expanded upon in Section 4.2.

3.3.1.2 Robot Localization

In order to fuse the IMU and wheel encoder values, and publish a unified pose estimate, an extended Kalman Filter is implemented using the Robot Localization package of ROS2 (25). This package provides the utility to use a standard, extended, or unscented Kalman Filter, and further provides a configuration file format for adjusting filter parameters. The filter comes with models for an omni-drive and dual-motor drive robot structure, and while it is not an exact match the dual drive configuration encompasses the behavior of the scale car accurately enough to be implemented. The extended Kalman filter was selected as it more accurately encompasses the motion of the scale car without adding the quantity of parameters which are used in an unscented filter. However, switching between filter implementations is as simple as revising the configuration file, and a study of performance between the filters could easily be performed in the future. This package, in addition to performing sensor fusion, also provides a smoothing between the map to car transform and the car to sensor transforms.

3.3.1.3 SLAM Toolbox

In order to map the environment, a graphical SLAM implementation is used. The SLAM toolbox for ROS2 provides another easily implemented package of SLAM nodes, which fuse LIDAR scans into a pose graph, which can be tuned and refined to improve performance. This map is then voxelized, and exported for future testing to remove the computational load of mapping while navigating during testing.

3.3.1.4 Adaptive Monte Carlo Localization

While the SLAM toolbox is highly optimized and does not use excessive resources in localization mode, a more optimal localization method is Adaptive Monte Carlo Localization (AMCL). This method uses a particle filter, generating between one and two thousand particles and forward simulating the motion of the vehicle. The particles are then averaged to identify the pose estimate and covariance of the car. The algorithm for this method is elaborated in Probabilistic Robotics, and application on the scale car further described in Section 5.1.3(26).

3.3.1.5 Controls

The remaining functionalities are implemented in the control packages, which encompass planning as well as control. The path planner was written previously for the Future Mobility Lab and has functionalities for Djikstra's search algorith, A*, RRT, and PRM path planners. As this work has been performed, compared and written about, it will not be expanded on in the scope of this thesis other than functional testing in

Section 5.2.

The controls are built using an abstract controller class, which is initialized with a configuration file that allows a variety of controllers to be constructed and used. For this thesis, pure-pursuit, Stanley, Youla, H_∞, and MPC controllers have been written and tested. These controls send a constant velocity command to the low-level drive motor controller, and a steering command to the servo. The entirety of Chapter 4 is dedicated to the development of these control methodologies. The ability to develop an independent longitudinal controller or integrated lateral and longitudinal controller was heavily considered in the development of the controller class, and can easily be added using the convention of the other controllers in the Python source code.

Chapter 4

Controller Design and Simulation

4.1 Control Problem

Passenger vehicles are inherently under-actuated for the purposes of trajectory following. This means that a car cannot follow an arbitrary provided path. For example, a car is generally incapable of pure lateral translation during standard operation. The actuators available to a car are throttle, brake and steering. The throttle and brake control longitudinal velocity, while steering controls the heading angle of the front wheels. For the purposes of this thesis, longitudinal control will be managed by a simple low level controller to achieve constant velocity, while several lateral controllers are tested and compared.

Lateral control of an autonomous vehicle is intended to allow the vehicle to follow a given trajectory or path. However, since the system is under-actuated, the dynamics of the car make this design non-trivial. A controller cannot simply control for cross-track error without consideration of vehicle yaw, as it would be incredibly difficult to tune and would likely result in erratic behavior about the target path. A variety of strategies exist to perform this control, which can be described as geometric control, model-based control, and model predictive control methods. Geometric controllers use kinematic relations between the car and path to determine a steering angle. Internal model controllers use a system model to derive the control law, which incorporates knowledge of the system to be controlled with the goal of improving performance. Model predictive controllers forward-simulate the vehicle's motion in discrete space, and use an optimization scheme to select steering angles at a set frequency.

4.2 Longitudinal Controller

Longitudinal control is limited within the scope of this thesis, and as such the scale car relies on a reliable, well-tested control scheme of a PI controller. During testing, it was noticed that the vehicle would slow down at high steering angles, and as such a feed-forward component was added to help account for the additional throttle required to maintain velocity into a turn.

$$P = K_p e + K_\delta \delta + K_i \int e \, dt \tag{4.1}$$

where: P = Throttle Command

K_p = Proportional Gain

e = Velocity Error

K_i = Integral

K_δ = Steering Gain

δ = Steering Angle

Due to the nature of this controller, and fact that it is implemented inline with an electronic speed controller that cannot be easily modeled, performance can only be gauged from real-world data. Figure 4.1 shows the step response of the vehicle from idle to a 1 $\frac{m}{s}$ setpoint. In addition to extensive tuning, the quality of the controller is gauged by its ability to maintain constant velocity during test maneuvers.

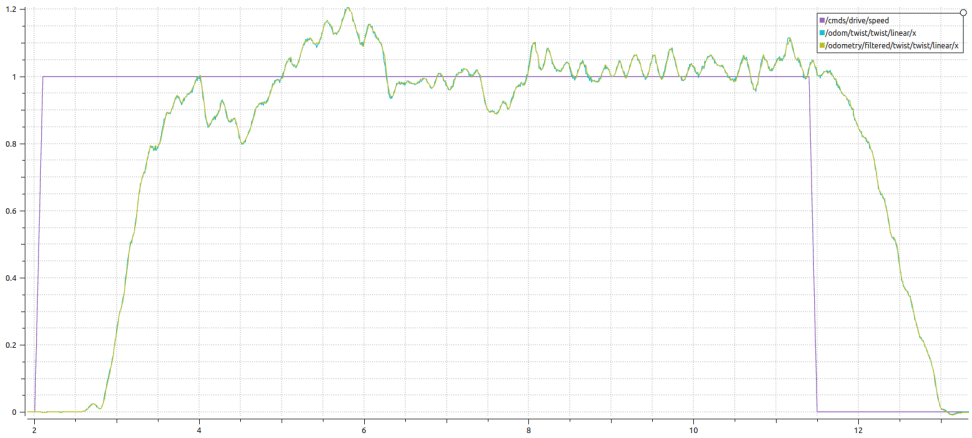


Figure 4.1: Behavior of Longitudinal Controller During Startup (Speed [$\frac{m}{s}$] v. Time [s])

4.3 Lateral Controllers

The lateral controllers are design to follow a predetermined trajectory at constant velocity, as speed is regulated by the longitudinal controller. In this section, the theory and control methodologies will be explained for controllers of increasing complexity. Geometric controllers for steering provide a simple control formulation, but rely on simple geometric relations between the vehicle pose and path which are not always held, and generally respond poorly to disturbances and sensor noise. Model-based controllers use mathematical models of the vehicle to be controlled in order to develop a controller which considers system dynamics. This enables the development of optimal or robust control strategies and can consider lateral as well as yaw

error. Finally, model predictive control forward-simulates the dynamics of the car and uses an optimization technique to select the next steering input. This method is computationally expensive compared to previous controllers, but is able to account for state and steering input limitations using its predictive model.

To be considered for use in a real car, a controller must be able to follow a given path with reasonable accuracy, and be sufficiently robust to common disturbances and sensor noise. Some common examples of antagonist interactions are GPS position jumping, wind gusts, poor sensor calibration, and for non-geometric controllers, modelling inaccuracies. Additionally, the controller must be able to operate with real steering, in which there is delay between control signal and actuation, as well as a limit to the rate of change of steering angle and maximum steer angle. Controllers must be able to operate with these physical limitations without reducing performance. In this chapter, theory and control formulation will be discussed, as well as the practical implementation of these strategies for testing on the scale car.

Before discussing controllers, there are a few common elements used in application to be presented. Controllers require a reference signal to operate, and for path following this signal is generally a reference point or set of reference points on the given path. For the most part, these reference points can be found by identifying either the nearest point to the car within the path, or to a point at some distance ahead of the car on the path. Algorithm 1 presents a basic method for finding reference points using both methods, as the lookahead calculation first involves finding the nearest reference point. It is assumed that all waypoints and pose data are provided in an ordered format.

Algorithm 1 Pseudo-Code for the Identification of Reference Points, Nearest and Look-Ahead

$l \leftarrow$ Look-Ahead Distance
waypoints $\leftarrow$ Ordered Waypoint Array
pose $\leftarrow$ Pose of Vehicle
procedure GET NEAREST REFERENCE(*pose*, *waypoints*)
    Distance Array $\leftarrow$ []
    for i in length(*waypoints*) **do**
        Reference Distance $\leftarrow$ norm($pose_{x,y} - waypoints(i)_{x,y}$)
        Distance Array.append(*Reference Distance*)
    Nearest Reference Index $\leftarrow$ minindex(*Distance Array*)
    return*Nearest Reference Index*, *Distance Array*
procedure GET LOOK-AHEAD REFERENCE($l, waypoints, pose$)
    Nearest Reference, *Distance Array* $\leftarrow$ GETNEARESTREFERENCE($pose, waypoints$)
    Look-Ahead Index $\leftarrow$ *Nearest Reference*
    Look-Ahead Reference $\leftarrow$ *Distance Array*(*Look-Ahead Index*)
    while *Look-Ahead Reference* $<= l$ **do**
        Look-Ahead Reference $\leftarrow$ *Distance Array*(*Look-Ahead Index*)
        Look-Ahead Index $\leftarrow$ *Look-Ahead Index* $+ 1$
    return*Look-Ahead Index*

These algorithms will be referenced when generating inputs to all controllers in the following sections.

4.3.1 Pure Pursuit

4.3.1.1 Theory

The pure pursuit method is an intuitive, simple control scheme, which looks ahead of the car at a given distance, and steers toward the path's location at this distance. Figure 4.2 depicts the geometric model used for the pure pursuit algorithm.

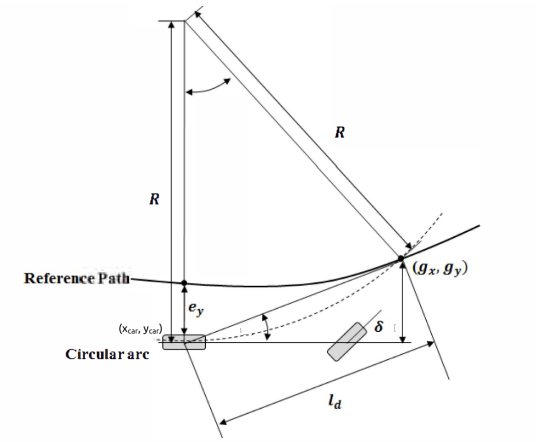


Figure 4.2: Pure Pursuit Geometric Relations (6)

One important consideration is the definition of the car's pose. In this model, as well as many in industry, the position of the vehicle is defined by the pose of its rear axle. For this controller, a point is found on the reference path at distance l_d in front of the rear axle, and steering angle is defined as δ. The full equation used in the controller, considering global path coordinates, is the following

$$\delta = \arctan(\frac{g_y - y_{car}}{g_x - x_{car}}) - \psi_g \tag{4.2}$$

where:
- δ = Steer Command
- (g_x, g_y) = Look-Ahead Reference
- (x_{car}, y_{car}) = Rear Axle Position
- l_d = Look-Ahead Distance
- ψ_g = Current Vehicle Global Yaw Heading (From $+x$ direction)

4.3.1.2 Controller Design

The implementation of this controller is relatively straightforward. After identifying the look-ahead reference point, the pure pursuit function is directly applied to determine the steering angle of the car, as shown in Algorithm 2.

Algorithm 2 Pseudocode for Pure Pursuit Controller

$l \leftarrow$ Look-Ahead Distance
$waypoints \leftarrow$ Ordered Waypoint Array
$pose \leftarrow$ Pose of Vehicle
procedure GET STEERING ANGLE($l, waypoints, pose$)
    Look-Ahead Reference Index $\leftarrow$ GET LOOK-AHEAD REFERENCE($l, waypoints, pose$)
    Y-Vector $\leftarrow$ *Look-Ahead Reference.y* $-$ *pose.y*
    X-Vector $\leftarrow$ *Look-Ahead Reference.x* $-$ *pose.x*
    SteerAngle $\leftarrow \arctan(\frac{\textit{Y-Vector}}{\textit{X-Vector}}) - \textit{pose.}\psi$
    return *SteerAngle*

4.3.2 Stanley

4.3.2.1 Theory

The Stanley controller was originally designed at Stanford during the DARPA Grand Challenge in 2005 (7). This geometric controller identifies a stabilizing function which combines both the yaw heading of the path and cross-track error. The equation was derived such that the system would have a globally asymptotic and stable equilibrium of zero error. For use in the Grand Challenge, additional terms were added to account for behavioral effects of the car, but these are not included in this controller as the base formulation is sufficient to develop a reference controller on the scale car.

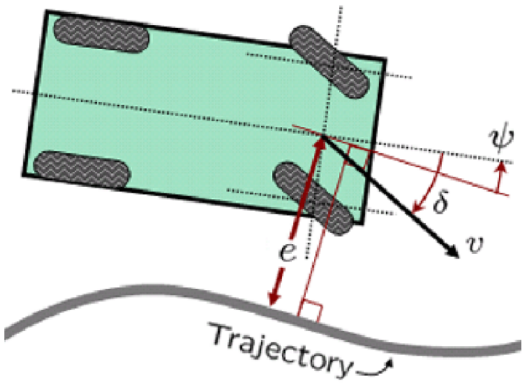


Figure 4.3: Stanley Controller Geometric Relations (7)

The stabilizing equation using this geometric model is as follows

$$\delta = \psi + \arctan(\frac{ke}{v}) \tag{4.3}$$

where: δ = Steer Command

ψ = Vehicle Yaw Heading Relative to Path Yaw

k = Stanley Gain Parameter

e = Cross-Track Error

v = Longitudinal Velocity

4.3.2.2 Controller Design

The implementation of this controller is similar to pure pursuit, in that reference points are generated, and control law directly applied. Pseudo-code for the Stanley controller implementation is shown in Algorithm 3.

Algorithm 3 Pseudocode for Stanley Controller

$k \leftarrow$ Stanley Controller Gain
$v \leftarrow$ Longitudinal Velocity
$waypoints \leftarrow$ Ordered Waypoint Array
$pose \leftarrow$ Pose of Vehicle
procedure GET STEERING ANGLE($v, waypoints, pose$)
    Nearest Reference Index, Distance Array $\leftarrow$ GET NEAREST REFERENCE($waypoints, pose$)
    $\psi_{ref} \leftarrow pose.\psi - waypoints(Nearest\ Reference\ Index).\psi$
    $e_{vector} = \text{cross}(pose_{x,y,z}, waypoints(Nearest\ Reference\ Index_{x,y,z})$
    $e \leftarrow Distance\ Array\ (Nearest\ Reference\ Index) * \text{sign}(e_{vector})$
    $SteerAngle \leftarrow \psi_{ref} + \arctan(\frac{ke}{v})$
    return *SteerAngle*

4.3.3 Youla Parameterization

4.3.3.1 Theory

The simplest model of a plant with a feedback controller is presented in Figure 4.4. In this system, error state e is provided as a reference signal to the controller, which then generates a control output. feedback gain is modeled by $H(s)$, and for the remainder of this discussion will be considered as Unity. For this control scheme, the return ratio or open-loop transfer function then becomes $L = G_c(s)G_p(s)$. This return ratio can be used to derive three transfer functions that convey an immense amount of information about the system's behavior. These transfer functions are sensitivity S, complementary sensitivity T, and Youla Y.

$$S = \frac{1}{1+L} \tag{4.4}$$

$$T = \frac{L}{1+L} \tag{4.5}$$

$$Y = \frac{G_c}{1+L} = G_c S = \frac{T}{G_p} \tag{4.6}$$

First, it is necessary to understand the relationship between sensitivity and complementary sensitivity. It can be shown simply that $S + T = 1$, and although this is apparently trivial, it is in face a profound statement. A well-designed controller should seek a T which has a gain of 0 dB at low frequencies, to ensure target tracking. T should then drop off at higher frequencies in order to reject noise, as well as high frequency disturbances. However, given the relationship of T and S, this implies that sensitivity must rise as complementary sensitivity drops. This reveals a key consideration of control design, that there is a necessary balance between optimally and robustness. That is to say that it is impossible to design a controller which perfectly tracks a setpoint at all frequency ranges.

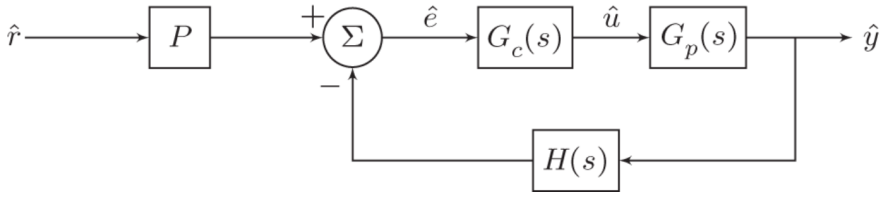


Figure 4.4: Simple Block Diagram

The Youla Parameterization Approach is a linear frequency-based control design technique. If a stable Youla parameter is found that meets a set of interpolation conditions, then it can be used to generate a controller that is guaranteed to be stabilizing in a linear sense, and whose robustness to noise and disturbances can easily be identified. The guiding principle of this method which enables its flexibility is the relation of sensitivity, complementary sensitivity and the Youla parameter, whose Bode magnitudes are shown in Figure 4.4.

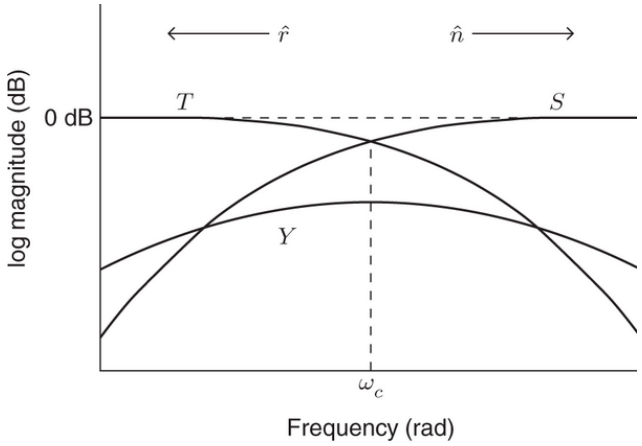


Figure 4.5: Sensitivity S, Complementary Sensitivity T, and Youla Y (8)

Using Youla Parameterization, it is possible to describe the set of *all* stabilizing linear controllers, so

long as certain conditions can be met. These conditions and their proofs are thoroughly explained in *Robust Control, Youla Parameterization Approach* (8). The necessary and sufficient conditions are as follows

1. Y must be stable.
2. G_pY must be stable.
3. $G_p(1 - G_pY)$ must be stable.

The first and second conditions are easy to achieve, but interpolation conditions are presented which can be used to confirm that the third requirement has been met. In complete form, a list of requirements is as follows

1. $T(p) = 1$ for any unstable poles p
2. $T(z) = 0$ for any unstable zeroes z
3. $\frac{d^k}{ds^k}T(p) = 0$ for $1 \leq k \leq a_p - 1$, a_p no. of repeated unstable poles in G_p
4. $\frac{d^k}{ds^k}T(z) = 0$ for $1 \leq k \leq a_z - 1$, a_z no. of repeated unstable zeroes in G_p

Although this appears to be more tedious to achieve, these interpolation conditions provide a straightforward method to designing a Y that is internally stabilizing. This method provides a great degree of flexibility in design.

4.3.3.2 Controller Design

To perform the requisite analysis and identify a Youla parameter for this model, Equations 2.9 and 2.10 were converted from state-space to transfer function form.

$$G_p = C(SI - A)^{-1}B + D = \frac{17.735(s + 4.073)(s + 1.049)}{s^2(s + 7.032)(s + 4.712)(s + 0.9091)} \tag{4.7}$$

where: $C = [1\ 0\ d_s\ 0]$

$d_s =$ Yaw Error Weight Relative to Cross-Track Error

$D = 0$

With two unstable poles at the origin, the additional interpolation conditions of repeated poles (as discussed in the previous section) must be met. This was achieved by first shaping T, and using the relationship of T and G_p to generate Y. T, S, and L for this system are presented in Figure 4.6. It is relevant

to note that the crossover frequency is between 1 and 2 Hz, which is sufficient to capture most lateral vehicle dynamic behavior in the linear range.

$$T = \frac{k(\tau_1 s + 1)}{(s^2 + 2\zeta_1\omega_1 s + \omega_1^2)(s^2 + 2\zeta_2\omega_2 s + \omega_2^2)} \tag{4.8}$$

where: $k, \zeta_1, \omega_1, \zeta_2, \omega_2 =$ Tuning Parameters

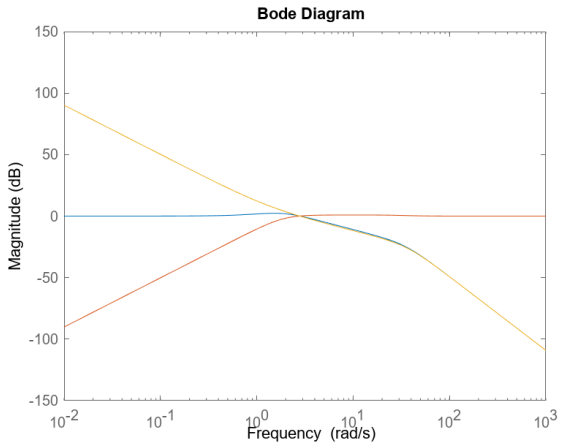


Figure 4.6: T (blue), S (red), and L (yellow) for the Scale Car

Using two second-order Butterworth Low Pass filters, in addition to a static gain k permits a great deal of design flexibility. After some tuning, the final Youla Parameter was selected. It can be quickly confirmed through examination that this selection meets all required interpolation conditions. Y was then used to generate the controller transfer function G_c using the relation of Y, S, and G_c. Bode magnitude plots of the relevant functions are presented in Figure 4.7.

$$Y = \frac{3146s^2(s + 7.032)(s + 4.712)(s + 1.15)(s + 0.9091)}{(s + 11.11)(s + 4.073)(s + 1.049)(s^2 + 2.687s + 3.61)(s^2 + 56.56s + 1600)} \tag{4.9}$$

After selecting G_c, the controller can be implemented by converting its transfer function to a discrete state-space formulation. Then, this digital filter can be used to implement a controller as shown in Algorithm 4. This requires a strict control frequency, which is currently 20 H_z for all controllers on the scale car. This frequency was selected as it captures all relevant dynamics necessary for the controller, with a Nyquist frequency of 10 Hz.

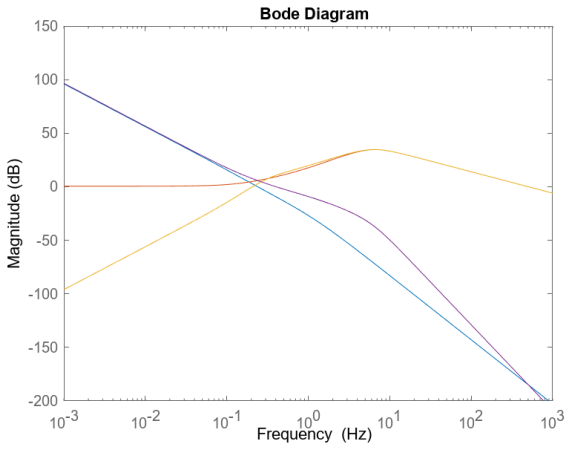


Figure 4.7: G_c (red), G_p (blue), Y (yellow), and L (purple) for the Scale Car

Algorithm 4 Pseudocode for LTI Controller

$A, B, C, D \leftarrow$ Discrete State-Space Form of G_c
$l \leftarrow$ Look-Ahead Distance
$waypoints \leftarrow$ Ordered Waypoint Array
$pose \leftarrow$ Pose of Vehicle
$G_cStates \leftarrow$ Array of Zeros, Size $m - by - 1$ for G_c of size $m - by - m$
procedure GET STEERING ANGLE($l, waypoints, pose$)
    $Look\text{-}Ahead\ Reference\ Index \leftarrow$ GET LOOK-AHEAD REFERENCE($l, waypoints, pose$)
    $error = \text{norm}(waypoints(Look\text{-}Ahead\ Reference\ Index)_{x,y} - pose_{x,y})$
    $G_cStates = \text{cross}(A, G_cStates) + \text{cross}(B, error)$
    $SteerAngle = \text{cross}(C, G_cStates) + \text{cross}(D, error)$
    return *SteerAngle*

4.3.4 H_∞

4.3.4.1 Theory

The H_∞ method is an optimization technique, which shapes the closed-loop transfer functions T, S, and Y using a set of weighing filters in order to generate a controller which minimizes an infinity norm of these weighted transfer functions. This method generates a form of optimal robust controller, in that it is robust to the worst-case disturbance response of the system and optimal for the presented cost function.

The mathematical proof of the validity for this optimization technique is again shown in *Robust Control, Youla Parameterization Approach* (8), but a cursory explanation of the method will be presented here. Figure 4.8 shows a set of weighing filters, which are used to derive a mapping transfer function T_{zw} which represents a transfer function of the response of all controlled signals and tracked errors z to all exogenous inputs w such as signal noise, disturbances, and control commands.

This function is shown in Equation 4.10. The weighing transfer functions W_p, W_u, and W_Δ can then be tuned to shape T, S, and Y and achieve good reference tracking, disturbance rejection, and controller actuation effort respectively. Optimization is achieved when this function is minimized by a controller K

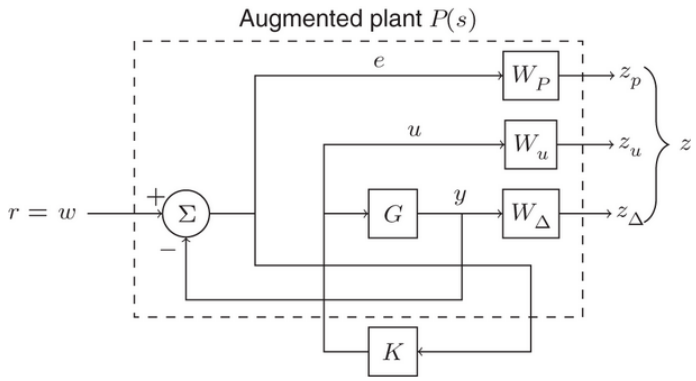


Figure 4.8: Weighing Matrices Used for H_∞ Optimization(8)

and the nominal performance is less than one, as shown in equation 4.11.

$$T_{zw} = [W_p S \quad W_u Y \quad W_\Delta T]^T \tag{4.10}$$

$$\underset{k}{\text{minimize}} ||T_{zw}||_\infty < 1 \tag{4.11}$$

Weighing filters can be selected given the following constraints.

$$|W_p(s)|^{-1} > |S(s)| \quad \forall \quad s > 0 \tag{4.12}$$

$$|W_\Delta(s)|^{-1} > |T(s)| \quad \forall \quad s > 0 \tag{4.13}$$

$$|W_u(s)|^{-1} > |Y(s)| \quad \forall \quad s > 0 \tag{4.14}$$

4.3.4.2 Controller Design

Transfer functions for W_p,W_u, and W_Δ were generated with respect to the conditions provided in the previous section, and are presented in Figure 4.9. Using these filters, H_∞ synthesis was achieved using a MATLAB optimization package, which generated a transfer function of the form

$$G_c = \frac{0.0075619(s + 2.234e07)(s + 13.69)(s^2 + 0.009107s + 4.145e - 05)}{(s + 2234)(s + 21.15)(s + 16.34)(s + 0.4355)(s + 1.056e - 05)} \tag{4.15}$$

The computed G_c can then be converted to a discrete states-space filter, much like the Youla approach. This means that the implementation of this method also uses Algorithm 4. A Comparison of G_c, G_p, Y, and L is shown in Figure 4.10.

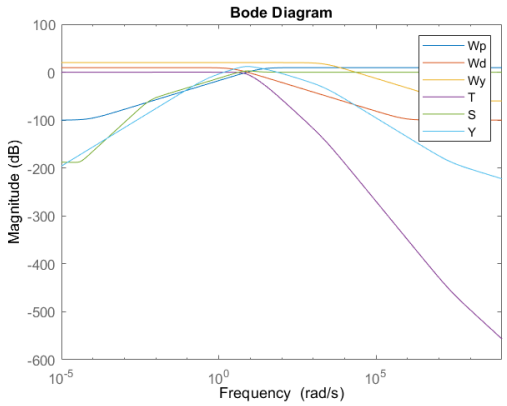


Figure 4.9: Weighing filters compared to T, S, and Y

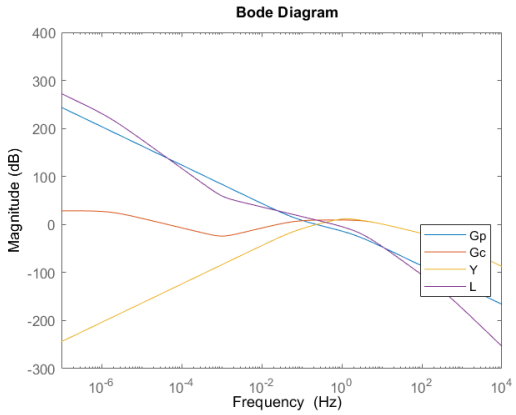


Figure 4.10: Weighing filters compared to T, S, and Y

4.3.5 Model Predictive Control

4.3.5.1 Theory

Model Predictive Control (MPC) is an optimal control strategy which forward-simulates a dynamic model, and optimizes the control inputs based on a given cost function. It is very computationally expensive relative to other control methodologies, with computing power only recently increasing to the level required for real-time operation on complex, fast dynamic systems. MPC is performed by discretizing n time steps forward to a time horizon T_d (in this case assuming number of input time steps N_u and state time steps N_y are equal), and optimizing within each time step using the assumption that a constant command signal for each time steps' duration. An example of the control discretization is shown in Figure 4.11. The fundamental assumption that enables MPC is that the system model is deterministic; if the system is determined only by its current states and inputs, and inputs are held constant, then the states at the end of each time step are completely determined by the inputs at the beginning of each time step. Therefore, an optimization scheme

only needs to vary the inputs to the dynamic system in order to find a solution for a given time step.

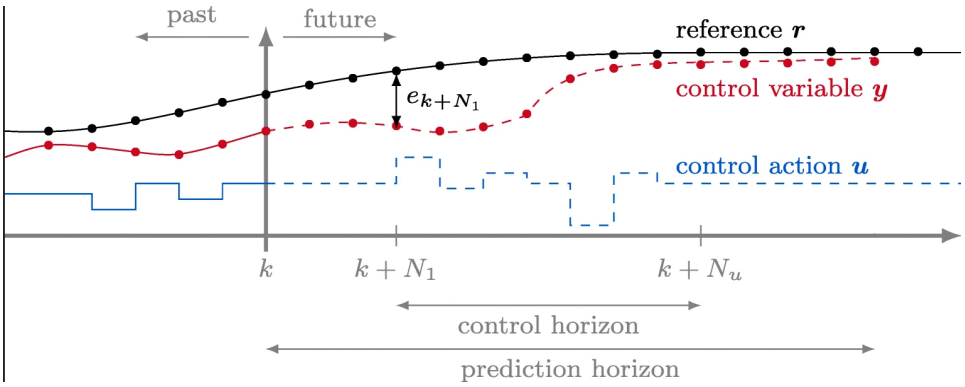


Figure 4.11: Discretized Control Output of an MPC Controller (9)

Now, knowing that optimizing inputs at the beginnings of each time step, and holding these inputs for the duration of each time step produces a theoretically optimal control output (in a discrete sense), the method of optimization becomes the primary question.

Optimization strategies seek to find the global minimum or maximum of some function. For linear systems, this is a solved problem, as linear convex systems must possess one global minima. For nonlinear systems, which can contain both local and global minima, the problem becomes more complex. As an optimization algorithm runs, it may descend into a local minima, in which there are currently no guaranteed methods to determine whether or not the given minimum is local or global, without probabilistic assumptions or brute force for nonlinear systems. However, so long as the system is convex, linear or nonlinear, it by definition must contain a single global minimum (27). In these situations, it is possible to use convex, constrained optimization techniques.

Many such techniques exist and are implemented in control applications, but for illustrative purposes the Newton-Raphson constrained optimization method will be presented here (28). This method uses a form of gradient descent, in which a Hessian approximation of a function about a given point is calculated, and its minimum determined. The formula for determining the next iteration is as follows

$$x_{k+1} = x_k - (\nabla^2 f(x_k))^{-1} \nabla f(x_k) \tag{4.16}$$

This process is iterated several times, and usually terminated when a maximum number of iterations is achieved, or the change of estimate from one iteration to the next becomes sufficiently small. This technique is applied at each time step, where in MPC applications the states x of this technique are the inputs, and $f(x_k)$ is the cost function of both states and inputs. Generally, cost functions are selected to be quadratic as functions of this sort are inherently convex. An example cost function would be

$$J = x^T Q x + u^T R u + x_f^T Q_T x_f \tag{4.17}$$

where: J = Cost

x = Vector of Error States

Q = Error Weighing Matrix

u = Vector of Inputs

R = Input Weighing Matrix

Q_T = Terminal State Weights

x_f = Final State

If the dynamic model is linear, and cost function is convex, then convex optimization techniques will be effective in determining a global minimum. With both a method of discretization and optimization defined, the final concept in MPC formulation can be elucidated: implementation.

The most straightforward way of implementing MPC is referred to as direct single shooting. This method is sequential, and is performed by optimizing over the first time step, and using the final state calculations as initial parameters in the next time step. The process is then repeated, solving step-by-step in time order until the time horizon is reached. This method is practical in many applications, but too slow in limited computation environments or for fast dynamic systems. A more advanced approach is called direct multiple shooting (29). This method optimizes over each time step *in parallel*, an effective strategy on most modern computing platforms. However, this method presents an issue of determining initial states. As opposed to single shooting, in which the initial states of each time step are calculated prior to optimizing, multiple shooting attempts to solve all inputs at once. This means that in multiple shooting, each time step must first guess its initial states, and optimize based on this guess. To remedy this, slack variables are introduced. Slack variables are a measure of error between one time step's final calculation and the next time step's initial guess, which is also frequently referred to as a defect. To generate a working solution, the optimization process attempts to derive inputs which minimize the cost function, and also reduce each defect to zero. These defects can have their own quadratic costs, such that the optimization scheme seeks to optimize the input to the system to both meet the control cost function and minimize error in initial state estimation. This increases the size of the state-space significantly, but in practice greatly reduces computational time.

4.3.5.2 Controller Design

In order to use MPC with a set of prescribed waypoints, it is necessary to relate the global and local frame by some means. Either the waypoints can be translated in realtime using a coordinate transform, or the

vehicle's position can be described with respect to the global frame. The latter method was chosen for this implementation, as it proved to be more intuitive in implementation (30). The nonlinear dynamic bicycle model with linear tire parameters, in global coordinates is as follows

$$\ddot{x} = \dot{\psi}\dot{y} + a_x \tag{4.18}$$

$$\ddot{y} = -\dot{\psi}\dot{x} + \frac{2}{m}(F_{c,f}\cos\delta_f + F_{c,r}) \tag{4.19}$$

$$\ddot{\psi} = \frac{2}{I_z}(l_f F_{c,f} - l_r F_{c,r}) \tag{4.20}$$

$$\dot{X} = \dot{x}\cos\psi - \dot{y}\sin\psi \tag{4.21}$$

$$\dot{Y} = \dot{x}\sin\psi + \dot{y}\cos\psi \tag{4.22}$$

where:
- x, y = Local Frame Position Coordinates
- X, Y = Global Frame Position Coordinates
- ψ = Yaw Heading
- m = Mass of Vehicle
- I_z = Mass Moment of Inertia of Vehicle
- $F_{c,f}$ = Front Cornering Force
- $F_{c,r}$ = Rear Cornering Force
- δ = Steering Angle

The cost function used in this formulation is modeled akin to Equation 4.17, in which error states and inputs have transient costs and the final error states are given a terminal cost. Additionally, slack parameters are given a cost between shooting nodes. This optimization problem formulation is inspired by work done in vehicular MPC for race cars at the University of Freiburg (31), and adapted to the previously described dynamic bicycle model. The cost function to be implemented then can be shown as follows

$$J = x^T Q x + u^T R u + x_f^T Q_T x_f + \sum_{n=1}^{n} |x_{n+1} - x_n|^T L |x_{n+1} - x_n| \tag{4.23}$$

where: x = [X,Y,ψ] Errors

$u = \delta$

Q = Error Weights

Q_T = Terminal Weights

R = Input Weights

L = Slack Weights Between Nodes

n = Number of Shooting Nodes

This MPC controller design is implemented using Acados, a highly optimized C code generator for nonlinear optimal control (32). Acados uses a symbolic interpreter to take a given system model and cost function, and generates C code that can be targeted directly to a computer chip architecture. Additionally, Acados has a Python interface which enables a previously generated code to be called directly in Python, making it easy to integrate with the scale car software stack. The pseudo-code for this implementation is presented in Algorithm 5. This implementation runs at an average 200 Hz using single-threading on the Raspberry Pi's processor, and easily meets the 20 Hz requirement of the controller on the scale car.

Algorithm 5 Pseudocode for MPC Controller

```
v ← Longitudinal Velocity
T_d ← Time Horizon
waypoints ← Ordered Waypoint Array
pose ← Pose of Vehicle
AcadosSolver ← Acados Solver Object
procedure GET STEERING ANGLE(v, T_d, waypoints, pose)
    Nearest Reference Index, Distance Array ← GET NEAREST REFERENCE(waypoints, pose)
    MPC_ref ← []
    for i in length(Distance Array) do
        if Distance Array(i) ≤ v * T_d and i ≥ Nearest Reference Index then
            MPC_ref.append(waypoints(i))                ▷ Collect All Reachable References
    AcadosSolver.pose ← pose
    AcadosSolver.references ← MPC_ref
    AcadosSolver.solve()                                ▷ Run Optimization Process
    SteerAngle ← AcadosSolver.u_0                       ▷ Take First Optimized Input
    return SteerAngle
```

4.4 Simulations

Before testing on the scale car, all controllers were simulated to confirm that their implementation is sound and perform initial tuning. Simulations for all controllers except for MPC were performed in MATLAB, and MPC performed in Python using the Acados Python interface. Test runs were performed on a path identical to the real test path. The simulation was performed by initializing the vehicle at the beginning of the path,

with a slight offset moving at constant velocity. The simulated model is a dynamic bicycle model, with constant cornering stiffnesses and nonlinearity introduced by coordinate transforms and steering angles. All control parameters were held constant during testing. For the test manuever, the car must first perform a right-hand turn and then chicane (quick right then left) at constant speed.

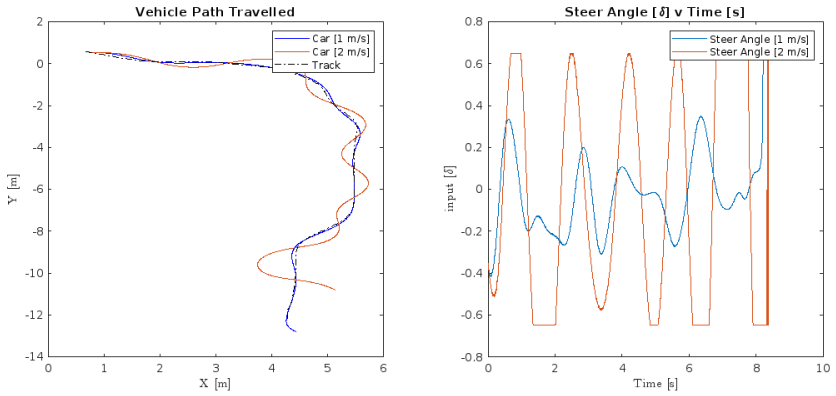


Figure 4.12: Simulation of Pure Pursuit Controller on Test Route

The Pure Pursuit controller performed the best at 1 $\frac{m}{s}$, and the worst at 2 $\frac{m}{s}$. The look-ahead distance selected was 0.5 m, which most likely contributed to the poor performance during the higher speed run. Furthermore, the controller experiences nearly bang-bang phenomena at higher speeds, which is not remotely close to optimal. However, if the look-ahead distance was increased then tracking performance was reduced to a degree that the car would likely have experienced a collision were the experiment to take place on the scale car. Performance is roughly as expected, as Pure Pursuit is documented as having near-optimal performance at very low speeds with non-aggressive maneuvers, but suffer greatly when either phenomena is introduced.

The Stanley controller shows good performance at both speeds, with slightly increased lateral error and higher overshoot during the chicane maneuver at the end of the route. Furthermore, the inputs steering angle does not saturate at any point in the run. This consistent performance aligns with expectations of the Stanley controller, as it was originally designed with constant velocity operation in mind.

The Youla controller performs roughly the same as the Stanley controller at 1 $\frac{m}{s}$, but shows much better performance at 2 $\frac{m}{s}$. This is likely due to the consideration of vehicle dynamics in the formulation controller, and good disturbance rejection of the Youla approach. However, the input steering angle has some jitter, which may affect performance during testing on the scale car as the servo has a limited input speed that it can achieve.

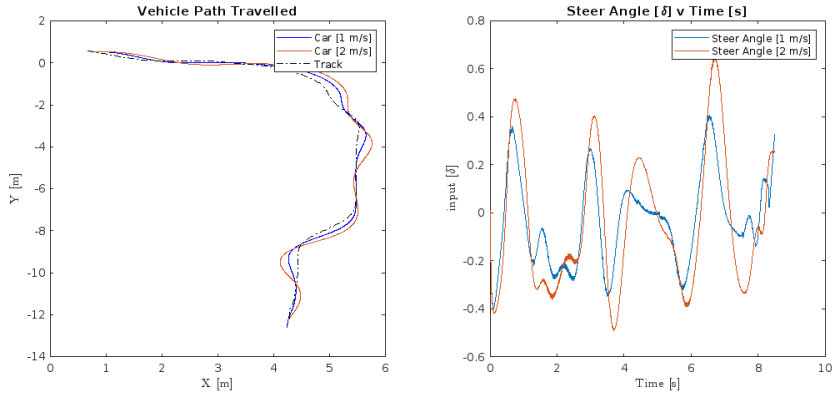


Figure 4.13: Simulation of Stanley Controller on Test Route

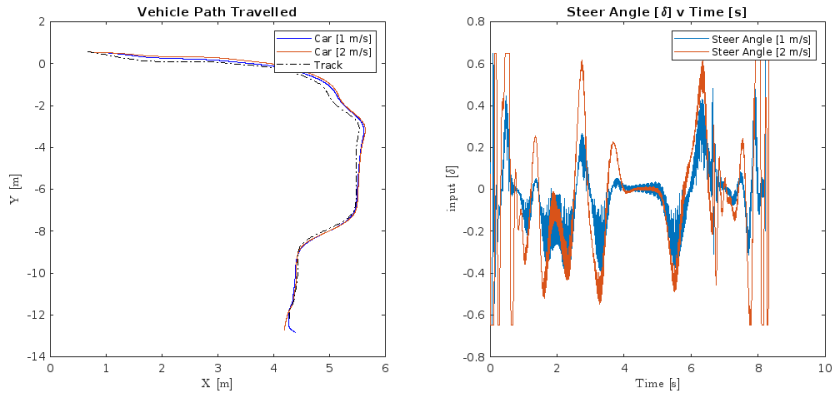


Figure 4.14: Simulation of Youla Controller on Test Route

The H_∞ controller performs as well as Youla at 1 $\frac{m}{s}$, but much worse at 2 $\frac{m}{s}$. In the future, additional investigation must be performed with regards to internal stability as well as the weighing matrices of this controller, as it should at minimum perform as well as the Youla controller. Other than the pure pursuit implementation, this is the only controller which saturates the steering input at the higher speed.

The MPC controller performs nearly as well as the Pure Pursuit controller at low speed, and the lowest root-mean-square lateral error at the higher speed. This controller was implemented in Python instead of MATLAB, as Acados was natively developed in Python. However, the dynamic model and test implementation are identical and as such can be compared meaningfully. An interesting phenomena is that the steering angles generated by the MPC controller are lower at high speeds, which is the inverse of the behavior exhibited by all other controllers. This is potentially due to the predictive feature of MPC, which calculates an optimal trajectory with low or no overshoot and provides a steering input which is expected to achieve this.

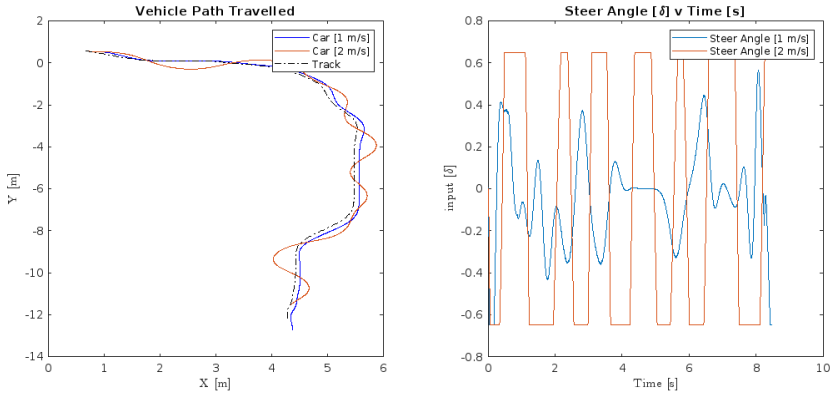


Figure 4.15: Simulation of H_∞ Controller on Test Route

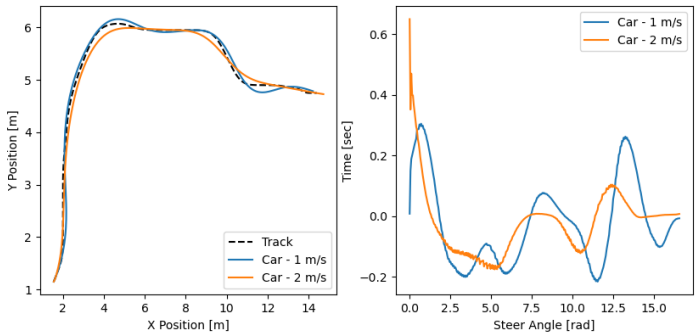


Figure 4.16: Simulation of Model Predictive Controller on Test Route

At higher speeds, with a constant time horizon, the controller is additionally optimizing over a larger span, possibly resulting in smoother performance at the higher speed.

From the simulations, root-mean-square lateral error measurements were taken for each controller at two speed set-points. These are tabulated in Table 6.1. Surprisingly, pure pursuit is shown to have the minimum error at low speeds. However, at higher speeds, the error was the largest by a significant margin. All other controllers have similar performance at 1 $\frac{m}{s}$, with their performance both quantitatively and qualitatively diverging at 2 $\frac{m}{s}$. This provides a good basis for testing on the scale car, as both data-driven and observational performance can be compared to these simulations. It should be noted that results are generated without any robustness analysis, in that model discrepancies, sensor noise, or additional disturbances were not considered.

Lateral Error	1 m/s	2 m/s
Pure Pursuit	0.0699	0.2697
Stanley	0.1003	0.1809
Youla	0.1017	0.1286
H_∞	0.1012	0.2138
MPC	0.0730	0.0887

Table 4.1: RMS of Lateral Error for All Controllers in Simulation

Chapter 5

Test Procedure

5.1 Functional Testing

After construction of the scale car and during development of its code base, functional testing was performed to ensure the system operated as expected. First, low level functions were validated, then mapping and localization packages following this. The validation process was performed to ensure that path planning and control algorithms were tested on a fundamentally sound system architecture.

5.1.1 Low Level Controls

First, the most basic components were validated, to ensure that the mounting, connecting, calibration, and filtering methodologies were acceptable. Motor drivers were tested by placing the car on a block, such that all wheels were free to rotate. Then, a set of steering commands were sequentially sent to the servo and visually inspected to ensure a match. A sine wave input was then sent to ensure steady turning could be performed. The motor electronic speed controller (ESC) contains an internal throttle regular, not specifically tied to velocity. Before testing full drive capabilities, throttle commands were sent to test forward, reverse, and idle ranges as well as the relative speed of the unloaded wheels. The operating range was then limited, as full throttle was observed to be excessively fast for any reasonable operation within the lab or any indoor test facility.

After determining that motor controls functioned acceptably, sensor tests were performed. The encoder and IMU were and tested in tandem with the Kalman filter implemented by the Robot Localization package. Figures 5.1 and 5.2 show raw data captured during a constant velocity maneuver. Results indicate that while velocity is not regulated strongly by the Kalman filter, with only the sharpest spikes being filtered out, the acceleration noise is highly filtered and produces a very smooth output near zero, which is reasonable given the maneuver performed.

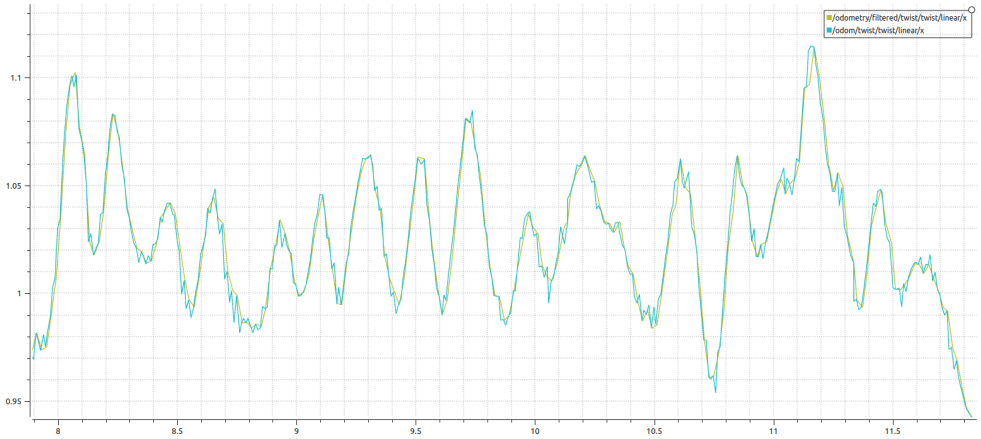


Figure 5.1: Encoder Raw Output, vs. Kalman Filtered

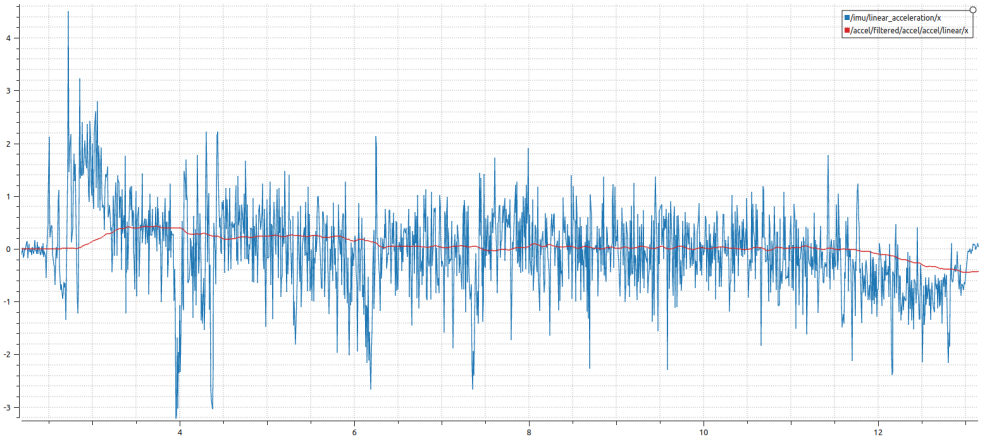


Figure 5.2: IMU Raw Output, vs. Kalman Filtered

5.1.2 SLAM Toolbox

Mapping was performed using SLAM Toolbox, and the generated pose graph was exported as an image file, an industry standard for two-dimensional maps. Initially, this map was very sparse, and had a significant issue of drift over the course of the mapping run. This was due to a low rate of scan samples being added to the pose graph, as well as a high driving speed of the car. This toolbox comes with an asynchronous mode, which stores all scans captured during a test run, and adds them to the pose graph whenever the CPU has additional capacity. After enabling this mode, and adjusting settings to add as many poses as reasonably possible to the graph, map quality was greatly improved 5.3. Slight drift is still observed, but is considered minimal given the amount of drift per unit length of the test run map. Additionally, the pose graph can be manually adjusted in future runs if drift is considered excessive using the ROS2 Visualization Tool (RVIZ).

The largest source of error in this map is observed between the two rooms of the lab, as it is difficult to perform loop closure in a way that aligns the rooms to each other in a satisfactory manner, despite each room itself being mapped accurately. To remedy this, the mapping session was greatly extended in time,

Figure 5.3: Comparison of Three Maps Taken During Tuning Process

and room re-mapped several dozen times within the same session. This provided ample opportunities for loop closure to the algorithm. Using the floor plan of the lab as a reference, the improvement in map quality becomes very clear. The final map and floor plan overlay are presented in Figure 5.4. Minor adjustments to the map allow it to essentially become a ground truth, and maintain the relative locations of furniture within the room.

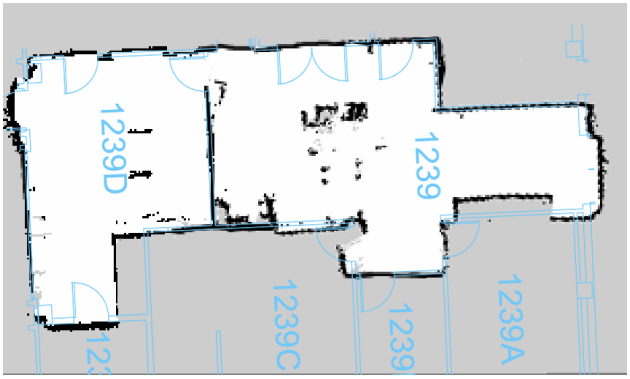
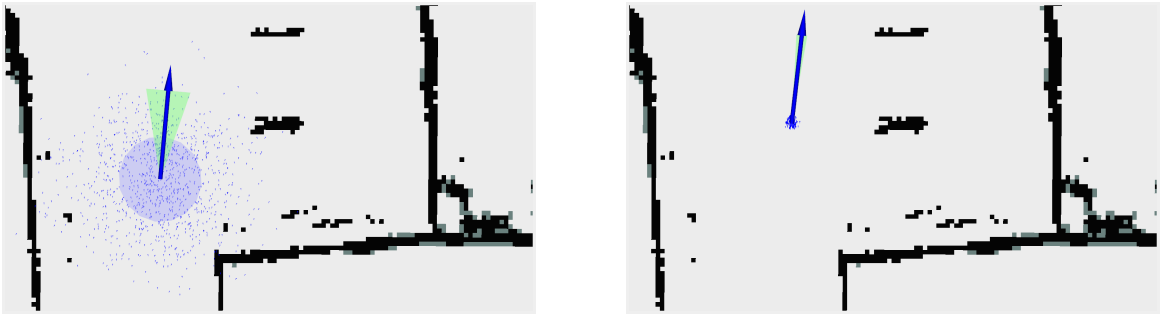


Figure 5.4: Final Map, Overlaid with Reference Floor Plan Geometry

5.1.3 AMCL

The final functional testing to be performed was the implementation and tuning of the AMCL node. This method uses a particle filter, which generates various pose estimates for the vehicle with a stochastic variation, averaging the existing estimates to provide a single pose of the robot. If estimates show significant error when matching the current LIDAR scan to the given map, these estimates are trimmed and new estimates generated. Initially, variance is large and estimates highly random, but if working properly they converge with a relatively small covariance. When first tested, this filter had very poor performance, especially when turning. The primary issue was discovered to be the stochastic variation of yaw, which was increased. To



(a) AMCL Visualization Prior to Convergence **(b)** AMCL Visualization After Convergence

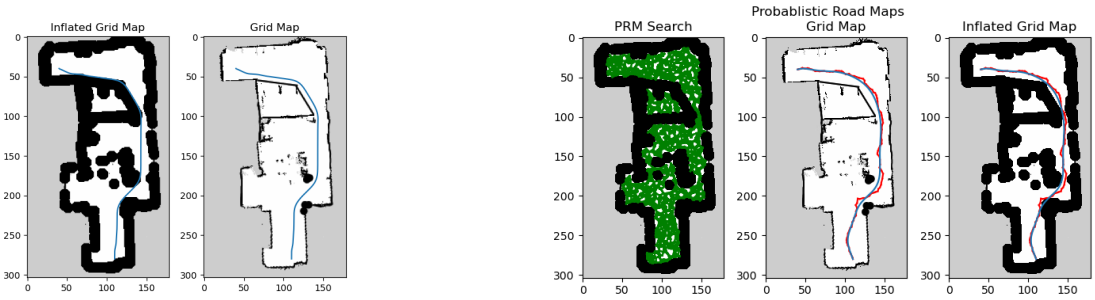
Figure 5.5: Tuned AMCL Performance Visualization

further improve performance, the number of generated particles was increased and tolerance for matching decreased to maximize the performance of the AMCL node, using as much of the available computational power as was deemed reasonable.

5.2 Path Planning

To generate a path for testing applications, it was decided that a single, well constructed global path should be generated and repeatedly used to validate controller performance against a universal test route. Tested algorithms include Dijkstra's, A*, Rapidly-Exploring Random Trees (RRT), and Probabilistic Road Map (PRM). A* was selected as the algorithm to be used to generate the standard route, due to its deterministic nature and relative performance improvement due to its implementation of a heuristic. Although performance was not an immediate concern, A* would most likely see use in a real scenario as opposed to Dijkstra's search algorithm and as such was decided to be the most realistic generated route. Additionally of note, the search algorithms were implemented with a standard form, not considering vehicle dynamics. From initial testing, it was decided to add a keep-out zone as well as add artificial search barriers around furniture corners to enable the planner to consistently generate a path which was dynamically feasible and possessed lower risk of a collision failure. These features are included in both examples of Figure 5.6.

In the future, it is highly recommended to incorporate a more advanced planner to reduce user input. One issue present is that the door is very narrow, and a binary cost map requires an inflation layer no larger than half of the door's width. A more ideal implementation would be the development of a modern cost map, with gradients of cost which would permit the planner to search for an optimal path even with tight dimensional constraints, and allow the search algorithm to intrinsically provide safer wall margins within the cost map gradient.



(a) A* Path Planner Results

(b) PRM Path Planner Results

Figure 5.6: Samples of Path Planner Results

5.3 Control Testing

Control testing was performed by running a test route with each controller, initialized at the same position with the same planned path. For initial testing, lower cost rubber tires were used to validate functionality. After this, tires were replaced with the Du-Bro tires modeled in Chapter 3. The final iteration of the scale car is shown in Figure 5.7. To prevent any damage to the vehicle in the event of a collision, a foam lining was installed as a front bumper, which proved effective in mitigating damage from slow speed collisions.

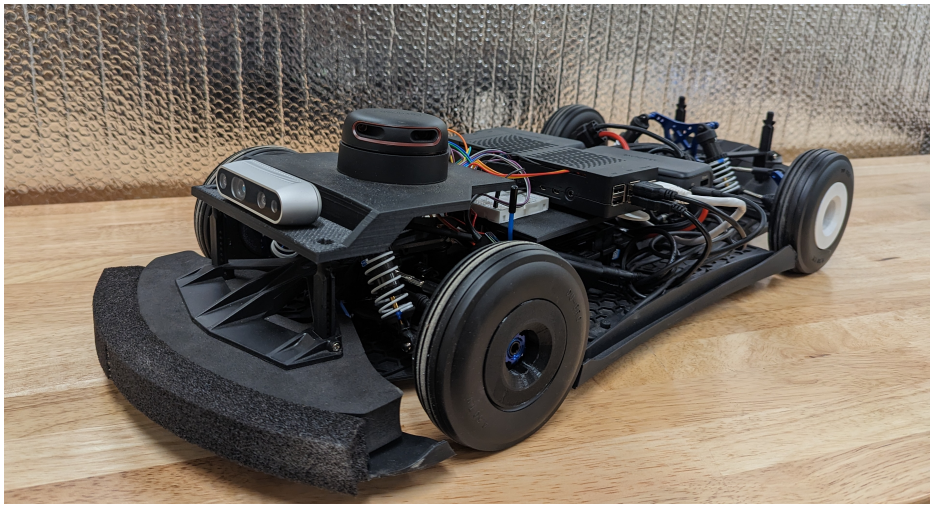


Figure 5.7: Final Iteration of Car Used in Testing

During testing, it was noticed that when passing from one room to another, the localization would experience sudden drift, which caused each controller to respond in a fashion similar to a small step input. Though the source of this jump has not been isolated, it is likely either due to the lack of features in the narrow doorway causing AMCL to converge more slowly, or the LIDAR reaching a minimum threshold distance in which scans become unreliable. To reduce the effect of this disturbance, the path was modified to ensure that the vehicle steered straight during passage between rooms, as turning through the door

introduced erratic behavior. Since the current yaw estimate relies heavily on AMCL, this problem could be mitigated by the introduction of additional yaw and yaw rate sensors onboard the scale car, to improve odometry in the inertial reference frame.

Four of the five controllers were implemented and tested with varying degrees of success. Unfortunately, at the time of writing, the H_∞ controller failed to complete the route. When implemented on the car, this controller saturates the steering input immediately, and even with anti-windup added to the controller, it's performance is akin to a bang-bang controller. This may be due to inconsistent velocity measurements, as this controller is optimized to an exact speed setpoint and the speed necessarily varies in a real environment. The remaining four controllers were able to complete the course, and generate data for comparison.

Chapter 6

Discussion & Conclusion

6.1 Results

Each successful controller test produced behavior similar, though not identical to simulation. The Pure Pursuit controller, whose results are shown in Figure 6.1, exhibited very similar behavior. Notably, it cut the corner in the first right-hand turn, which is an expected behavior of the controller. Furthermore, the car did not overshoot the chicane maneuver at all, but took additional time to converge to the path.

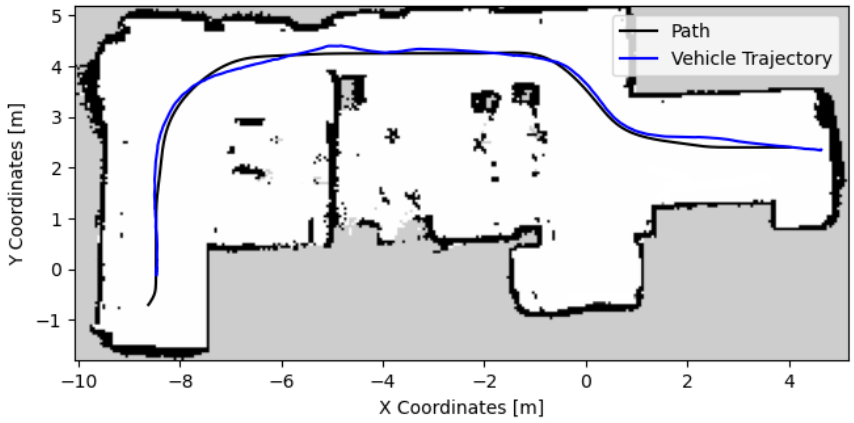


Figure 6.1: Pure Pursuit Testing on Scale Car - 1 m/s

The Stanley controller also performed similarly to simulation, following the path as illustrated in Figure 6.2. This controller was able to track the right hand turn very well, but exhibits oscillatory behavior after entry into the new room, likely caused by poor disturbance rejection. The controller also overshoots the chicane maneuver as expected from simulation, but the controller does course-correct and track the line for the remainder of the test run.

The Youla controller performed in similar manner to the simulation, except at the beginning of the chicane. This could likely be due to to the bandwidth limiting behavior of the controller, as well as discrep-

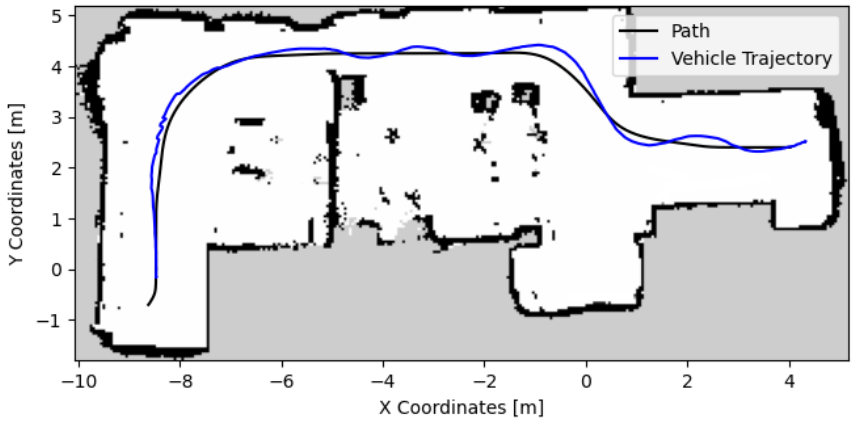


Figure 6.2: Stanley Testing on Scale Car - 1 m/s

ancies between the model and physical car. However, compared to the other controllers, the Youla controller was able to handle the localization disturbance through the doorway, and maintain course along the path. Results of this controller are shown in Figure 6.3.

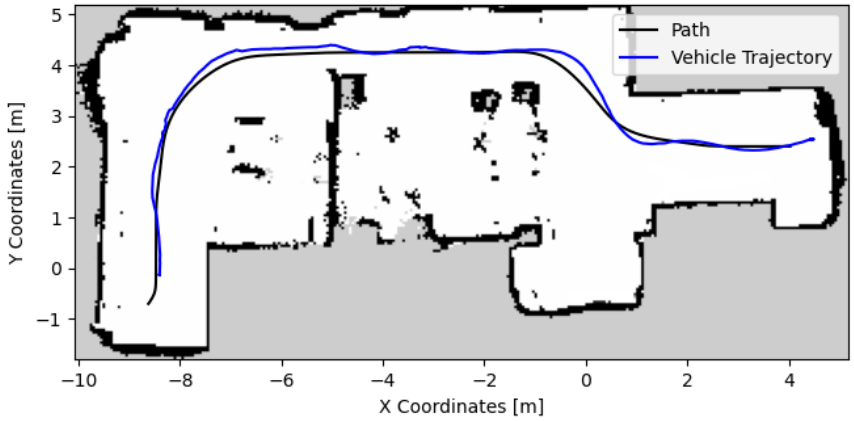


Figure 6.3: Youla Testing on Scale Car - 1 m/s

The Model Predictive Controller showed perhaps the largest discrepancy between simulation and reality, not reacting quickly to the first turn as well as overshooting the chicane. Additionally, it has the largest difference in average lateral error between reality and simulation. This is most likely due to discrepancies between the design model and actual car. In simulation, MPC is tested against its own exact model, but any modelling discrepancies between the design model and true controller may impact controller performance on a real vehicle. One additional consideration is that tuning the simulation is relatively easy, but tuning MPC on the car is much more difficult. This process requires re-compilation of the code each time the cost function

is modified. Though in a lab setting this can be mitigated, this would present a challenge for adoption of this control methodology in industry as it would greatly lengthen the tuning process.

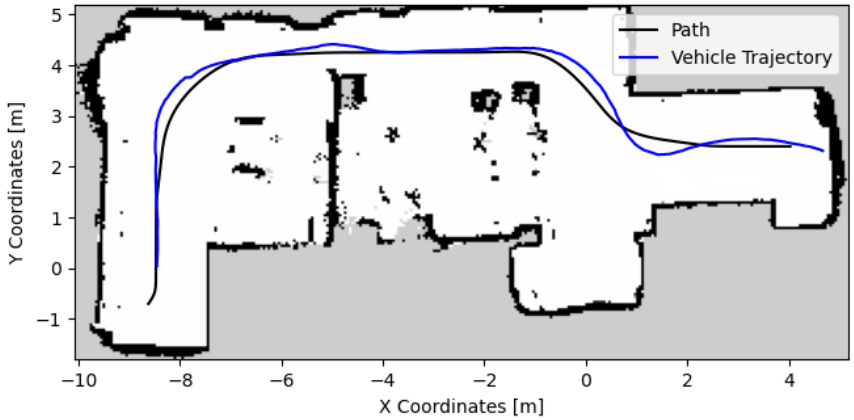


Figure 6.4: MPC Testing on Scale Car - 1 m/s

Lateral Error	1 m/s
Pure Pursuit	0.0846
Stanley	0.1046
Youla	0.0999
MPC	0.1367

Table 6.1: RMS of Lateral Error for All Controllers in Simulation

The purpose of this work was to develop a scaled vehicle test bed for use in the research of controls for autonomous vehicle applications. A vehicle was constructed which possesses dynamic similitude with a standard battery electric vehicle, and maintains the ability to be retrofitted and align with other parameters if so desired. On this test bed, a control architecture was developed using ROS2, providing a modular software stack that enables isolated research into the many subsystems of an autonomous car, as well as full system research. Five controllers were designed, implemented, and compared within this framework, validating the practicality of this application.

The development of the scaled test bed was successful, and resulted in the construction of a functional, dynamically similar model. However, only minimal validation has been done with respect to this car, and much more work can be done to confirm both the acquired model parameters, as well as correlate them to a full-size vehicle. The implementation of ROS2 was successful, and provides an avenue for continued development of the test bed. As new technologies are developed and implemented in ROS2, such as advanced localization and sensor fusion packages, they can be added to and tested on the scale car with minimal effort.

The controllers implemented present a meaningful comparison not only between the respective methods,

but also between simulated and real performance. The work done in this thesis reinforces the necessity of robust control design, and confirms the essential characteristics of each tested control methodology.

6.2 Future Work

As this project was built from the ground up, and therefore relatively new, there is no lack of work to be performed. Dynamics within the linear range have been validated within literature, but more work can be done to validate the nonlinear dynamics of the vehicle. Additionally, higher order models can be developed. An ultimate goal of this validation would be the tuning of scale parameters to a full-size vehicle, then the testing of a full control implementation pipeline. This would require simulation, implementation on the scale car, and finally implementation on a full-scale car. One important aspect to investigate is the tuning of controllers between both vehicles. This is performed easily using Youla or H_∞ methods, as they are model-based controllers with a constant G_c, but other controllers have parameters that may scale in a fashion that has not yet been investigated.

With this car, tuning has been performed to improve vehicle function as much as possible within the project's time frame, but more work can always be done. Filtering, localization, and controller tuning has been done individually, with the goal of improving controller performance, but all three configurations interrelate and should be investigated further. Additionally, only one method of filtering and localization has been thoroughly tested, and different methodologies could be investigated in these areas as has been performed for the controller. Furthermore, path planning is limited currently to the generation of a global path, which could be improved in the future to enable re-planning and obstacle avoidance. From experiments performed, there is enough computational space left in the compute units to enable this and other more advanced autonomous navigation concepts.

In the future, lessons learned from this scale car build could be used to construct a more advanced scale car, such as one which enables torque vectoring with a dual motor configuration. One possibility is to design this build to have inertial masses which can be re-positioned to match several different vehicle models, to enable rapid tuning and development of controllers for several vehicles at once. There are many such options for improvement, but one universal improvement that should be made is the use of a more complex IMU and encoders on each wheel as well as an average wheel speed encoder. State estimation is limited and heavily relies on the tuning of the AMCL package, which introduces a covariance and calculation delay which could be improved with the implementation of more sensors and more robust sensor fusion. Additionally, it was determined through this build that the driver motor of the Traxxas XO-1 has significantly more power than is required for research at virtually and speed. With the current scaling, the car would be performing 100

mile per hour equivalent maneuvers at only a speed of 7 meters per second. In fact, using a motor with lower torque or gearing may improve startup performance, as this motor was primarily designed for race track operation. Finally, given the abundance of power available in the RC car batteries, it would be reasonable to install a Buck converter, taking power from the car's batteries. The current portable batteries powering each Raspberry Pi are sufficient, but with access to higher voltages with a higher amperage tolerance than the current configuration, next generation single board computers such as the Rock Pi 4 could easily be implemented to upgrade the car's computational stack.

Overall, this project has accomplished what it set out to achieve: the development of a dynamically scaled vehicle that can be used for controls research. Beyond this, the work has enabled a variety of future endeavors into scaled car autonomous vehicle development, and provided an affordable means with which to create new test beds which have access to leading-edge technologies.

References

[1] M. Polley and A. Alleyne, "Dimensionless analysis of tire characteristics for vehicle dynamics studies," *Proceedings of the 2004 American Control Conference*, 2004.

[2] S. Pendleton, H. Andersen, X. Du, X. Shen, M. Meghjani, Y. Eng, D. Rus, and M. Ang, "Perception, planning, control, and coordination for autonomous vehicles," *Machines*, vol. 5, no. 1, p. 6, 2017.

[3] Jin, Yin, and Chen, "Advanced estimation techniques for vehicle system dynamic state: A survey," *Sensors*, vol. 19, no. 19, p. 4289, 2019.

[4] R. Rajamani, *Vehicle Dynamics and control*. Springer, 2012.

[5] R. N. Jazar, *Vehicle dynamics: Theory and application*. Springer, 2018.

[6] M.-W. Park, S.-W. Lee, and W.-Y. Han, "Development of lateral control system for autonomous vehicle based on adaptive pure pursuit algorithm," *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, 2014.

[7] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun, "Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing," *2007 American Control Conference*, 2007.

[8] F. Assadian and K. R. Mallon, *Robust control: Youla parameterization approach*. Wiley, 2022.

[9] M. Schwenzer, M. Ay, T. Bergs, and D. Abel, "Review on model predictive control: An engineering perspective," *The International Journal of Advanced Manufacturing Technology*, vol. 117, no. 5-6, p. 1327–1349, 2021.

[10] S. International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," 2014.

[11] M. Martínez-Díaz and F. Soriguera, "Autonomous vehicles: Theoretical and practical challenges," *Transportation Research Procedia*, vol. 33, p. 275–282, 2018.

[12] J. Dorsey, "Path planning for autonomous vehicles," 2023.

[13] S. Brennan and A. Alleyne, "The illinois roadway simulator: A mechatronic testbed for vehicle dynamics and control," *IEEE/ASME Transactions on Mechatronics*, vol. 5, no. 4, p. 349–359, 2000.

[14] R. O'Brien, J. Piepmeier, P. Hoblet, S. Burns, and C. George, "Scale-model vehicle analysis using an off-the-shelf scale-model testing apparatus," *Proceedings of the 2004 American Control Conference*, 2004.

[15] A. Liburdi, "Development of a scale vehicle dynamics test bed," 2010.

[16] V. S. Babu and M. Behl, "f1tenth.dev - an open-source ros based f1/10 autonomous racing simulator," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)*, pp. 1614–1620, 2020.

[17] S. S. Srinivasa, P. Lancaster, J. Michalove, M. Schmittle, C. Summers, M. Rockett, J. R. Smith, S. Chouhury, C. Mavrogiannis, and F. Sadeghi, "MuSHR: A low-cost, open-source robotic racecar for education and research," *CoRR*, vol. abs/1908.08031, 2019.

[18] D. L. Edwards and D. M. Bevly, "A method to estimate critical tire properties using non-linear tire models," *Volume 9: Mechanical Systems and Control, Parts A, B, and C*, 2007.

[19] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle System Dynamics*, vol. 21, no. sup001, p. 1–18, 1992.

[20] A. A. Sonin, "A generalization of the pi-theorem and dimensional analysis," *Proceedings of the National Academy of Sciences*, vol. 101, no. 23, p. 8525–8526, 2004.

[21] R. Wang, H. Zhang, J. Wang, F. Yan, and N. Chen, "Robust lateral motion control of four-wheel independently actuated electric vehicles with tire force saturation consideration," *Journal of the Franklin Institute*, vol. 352, no. 2, p. 645–668, 2015.

[22] S. Zagorski, G. Heydinger, J. Coyle, and M. Jebode, "Measured vehicle inertial parameters - nhtsa's data through august 2020," *SAE Technical Paper Series*, 2021.

[23] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[24] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The marathon 2: A navigation system," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[25] T. Moore and D. Stouch, "A generalized extended kalman filter implementation for the robot operating system," in *Proceedings of the 13th International Conference on Intelligent Autonomous Systems (IAS-13)*, Springer, July 2014.

[26] T. Sebastian, B. Wolfram, and F. Dieter, *Probabilistic Robotics*. The MIT Press, 2005.

[27] S. P. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge University Press, 2021.

[28] B. Polyak, "Newton's method and its use in optimization," *European Journal of Operational Research*, vol. 181, no. 3, p. 1086–1096, 2007.

[29] J. B. Rawlings, D. Q. Mayne, and M. Diehl, *Model predictive control: Theory, computation, and design*. Nob Hill Publishing, 2020.

[30] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," *2015 IEEE Intelligent Vehicles Symposium (IV)*, 2015.

[31] D. Kloeser, T. Schoels, T. Sartor, A. Zanelli, G. Prison, and M. Diehl, "Nmpc for racing using a singularity-free path-parametric model with obstacle avoidance," *IFAC-PapersOnLine*, vol. 53, no. 2, p. 14324–14329, 2020.

[32] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021.