1-1-2017

Optimized Trajectory Generation for Car-Like Robots on a Closed Loop Track

Tyler Friedl
University of Denver

Optimized Trajectory Generation for Car-Like Robots on a Closed Loop Track

A Thesis

Presented to

the Faculty of the Daniel Felix Ritchie School of Engineering and Computer Science

University of Denver

in Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Tyler Friedl

November 2017

Advisor: Kyoung-Dae Kim

Author: Tyler Friedl
Title: Optimized Trajectory Generation for Car-Like Robots on a Closed Loop Track
Advisor: Kyoung-Dae Kim
Degree Date: November 2017

Abstract

This thesis presents a method for generating an optimized path through a given track. The path is generated by choosing waypoints throughout the track then iteratively optimizing the position of these waypoints. The waypoints are then connected by optimized paths represented by curvature polynomials. The end result is a path through the track represented as a spline of curvature polynomials. This method is applied to multiple simulated tracks and the results are presented. By generating and representing the paths in the continuous domain, the method has improved computational efficiency from many of the discrete methods used to generate an optimal path through a track. Also, when using a path to guide an autonomous vehicle, paths represented in the continuous domain can allow for better tracking and control than discrete counterparts. As autonomous systems become more integral to our society, increased computational efficiency, tracking, and control are important areas of improvement.

Acknowledgements

Thanks to Dr. Kyoung-Dae Kim for guiding me through the process of developing the methods in this thesis and writing the thesis.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 General Background

As improvements in autonomous systems are being made, the need for efficient path generation is becoming increasingly important. The problem of path generation for car-like robots is one of particular interest and difficulty. The interest stems from a desire to create fully autonomous cars and the difficulty from the motion constraints of car-like robots.

Most current methods for generating the optimal path through a track are discrete and can be computationally expensive. Paths generated in the continuous domain are often more computationally efficient and allow for better tracking and control than their discrete counterparts.

1.2 Previous Work

There has been relatively little work completed on optimized path generation for car-like robots traveling through a track. S. Omidshafiei presents an approach in [19]. Omidshafiei uses both kinematic and dynamic car models to formulate an optimal control problem then uses the GPOPS MATLAB toolbox to solve this problem. The problem is formulated in discrete phases that represent sections of the track. While Omidshafiei was able to produce good results for less complicated tracks (smaller number of phases), when this method was applied to a more complicated track (80+ phases) the problem was determined to be infeasible to solve in MATLAB.

In [29], S. Koutrik also presented an approach to solve for the optimized path for a race car traveling through a track. Like Omidshafiei, Koutrik used a dynamic model of a car to formulate an optimal control problem. He then discretized and translated the problem into a non-linear control problem. Reference [11] presents the fuzzy controller methods which were used in the 2007 CIG Simulated Car Racing Competition, by the winning entry. The race objective is to reach as many randomly generated waypoints as possible within a given time frame. In [3] the authors outline a method for computing the time-optimal race line through a track using a series of connected Bézier curves. The authors of [7] find the optimal race line through a track by dividing the track into sections and use genetic algorithms to search for the optimal path, in the sense of both path length and curvature. Reference [22] addresses finding an optimal path through a track while avoiding collision with other cars on the track.

As mentioned above much of the work in this area is discretized and becomes increasingly computationally expensive as the tracks become more complicated. This is due to having a very large number of decision variables, parameters, in a single optimization problem. The method developed in this thesis has no more than twelve parameters for each optimization problem iteratively solved.

Closely related to path planning through a track is general path planning for non-holomonic or car-like robots. A. Scheuer and Th. Fraichard, in [24], create continuous curvature paths using line segments, circular arcs or clothoid arcs. In [16] the problem of finding smooth and collision free trajectories is presented and studied. The authors of [15] use a lower level geometric planner to find a path and then optimize the path so that it is the shortest distance which fits the non-holomonic constraints of the robot. A method of using sinusoidal curves to compute non-holomonic paths is presented in [17]. In [1], N. Achour, A. Lakhdari and F. Ferguene use Hierarchical Genetic Algorithms to solve for the optimal paths, balancing between path length and collision avoidance. In [9] and [27] the problem of motion planning and control of multiple car-like robots is addressed.

A. Kelly and B. Nagy present a method for rapidly generating optimized paths using curvature polynomials in [13]. This serves as the basis for the path generation method in this thesis and will be discussed in more detail later on.

The problem of general collision free motion planning is also closely related to the problem of optimal path generation through a track. In [21] S. Quinlan and O. Khatib present the method of elastic bands. This method forms a connection between global path planning and real time collision avoidance.

The elastic band method can both smooth a globally planned path to fit the constraints of a given robot and adjust the path to avoid obstacles as the robot detects them. In [20] a kinematic model is used to determine the set of feasible trajectories. Obstacle conditions, for moving obstacles, are then used to determine the feasible collision free trajectories from this set. The authors of [18] provide an off-line trajectory planner for a robot with a single spherical wheel and propose a feedback controller to track and control the robot along the computed trajectory. Genetics algorithms are used to plan for the motion of mobile robots in [26].

Cubic splines are used to parameterize the state-space trajectory and solve for smooth and time-optimal trajectories, of robot manipulators, in [8]. Reference [28] uses the RRT (Rapidly-exploring Random Tree) algorithm to generate a set of possible paths for a mobile robot then develops an algorithm to select the best path, taking into account information such as the sensors and controllers of the robot.

In [31] and [30] E. Velenis and P. Tsiotras formulate a method for the generation of a time optimal velocity profile for a car traveling through a given path. The velocity profiles generated in this paper are done so using the methods presented in [31] and [30]. These methods will be discussed in more detail in a later Chapter. Related to velocity profile generation is tracking and control of car-like robots. L. Caracciolo, A. Luca and S. Iannitti use a dymanic car model and design a non-linear controller to solve this problem in [6].

This thesis also uses non-linear programming and optimal control techniques such as the ones described in [13]. Much work has been done in this

area. There are a variety of algorithms and methods that can be used to solve optimal control and non-linear programming problems such as in [4], [12], [5], [25] and [10]. Reference [14] provides an overview of the history of non-linear programming developments. In [23] R. Rockafellar extends the traditional formulation of non-linear programming problems.

1.3 Problem Introduction

This thesis addresses the problem of computing an optimized path through a closed loop track. The method presented optimizes a series of waypoints selected around the track,

$$\mathbf{w}_i = [x_i \; y_i \; \theta_i \; \kappa_i]^T , \tag{1.3.1}$$

such that the path connecting these waypoints is the optimal path through the track. In (1.3.1), (x_i, y_i) is the position coordinate of the ith waypoint and θ_i and κ_i are the heading and curvature of the ith waypoint, respectively. Technical contributions that were made in the process of developing this method are: (i) Generalizing the optimized path generation method from an arbitrary starting state. (ii) Formulating a constrained optimization problem to solve for the optimal position of a waypoint. The optimality is with respect to the path through the waypoint from given start to end states. (iii) Presenting an approach, using iterative algorithms, to optimize the position of a series of waypoints throughout a track.

1.4 Thesis Overview

This thesis is divided into six Chapters. Chapter 2 formulates the methods to: (i) Generate an optimal path given fixed but arbitrary start and end states and (ii) optimize a single waypoint, with respect to the path, from a beginning state, through the waypoint, to an end state. Chapter 3 develops the solution, using the methods formulated in Chapter 2, to compute the optimized path through a track. Chapter 4 presents the method used to build the time-optimal velocity profile for a car traveling on the optimized paths. Chapter 5 presents the results of the solution on multiple tracks and analyzes the performance of the solution. Chapter 6 is the conclusion.

Chapter 2

Point-to-Point Path Generation

2.1 Formulation of Curvature Polynomial Approach

As previously mentioned, the path generation methods used in this thesis are derived by the methods presented in [13]. This section briefly describes those methods as they apply to this thesis.

The state of a vehicle can be modeled as:

$$\begin{aligned} x(s) &= \int_0^s \cos\theta(z)dz \\ y(s) &= \int_0^s \sin\theta(z)dz \\ \theta(s) &= \int_0^s \kappa(z)dz \\ \kappa(s) &= u(s). \end{aligned} \tag{2.1.1}$$

The curvature, $\kappa(s)$, is considered the input. The state vector is then made

up of position coordinates, heading and curvature,

$$\mathbf{x} = [x \ y \ \theta \ \kappa]^T . \tag{2.1.2}$$

The problem of path generation can then be stated as follows: Given initial and final states, $\mathbf{x}_0$ and $\mathbf{x}_f$, find the input, $u(s)$, such that the resulting path satisfies $\mathbf{x}_0$ and $\mathbf{x}_f$, as well as any constraints on the path's shape. A sample generated path is shown in Figure 2.1.

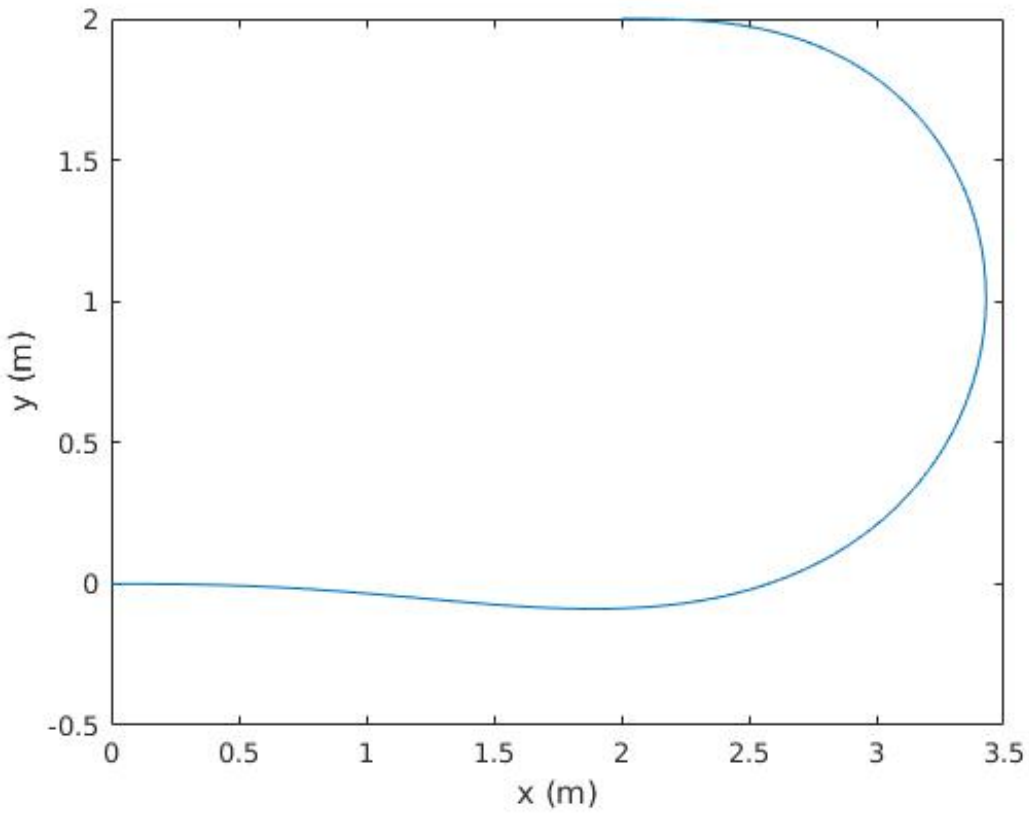

Figure 2.1: A path generated from $\mathbf{x}_0 = [0 \ \ 0 \ \ 0 \ \ 0]^T$ to $\mathbf{x}_f = [2 \ \ 2 \ \ \pi \ \ 0]^T$.

If we choose polynomial spirals of arbitrary order as the form of the input,

$$u(s) = a + bs + cs^2 + ds^3 + \cdots, \tag{2.1.3}$$

then the state of the vehicle can now be parameterized as follows:

$$\begin{aligned} x(\mathbf{p}, s) &= \int_0^s \cos\theta(\mathbf{p}, z)dz \\ y(\mathbf{p}, s) &= \int_0^s \sin\theta(\mathbf{p}, z)dz \\ \theta(\mathbf{p}, s) &= as + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \cdots \\ \kappa(\mathbf{p}, s) &= a + bs + cs^2 + ds^3 + \cdots \end{aligned} \tag{2.1.4}$$

where $\mathbf{p} = [a\ b\ c\ d\ \cdots]^T$.

The challenge is then to calculate the parameters $\mathbf{p}$ such that $\mathbf{x}_0$ and $\mathbf{x}_f$ are satisfied. This can be accomplished by formulating the problem as a constrained optimization problem. Let s_f be the length of path that satisfies boundary conditions $\mathbf{x}_0$ and $\mathbf{x}_f$. Then, a new extended parameter vector including s_f is defined as:

$$\mathbf{q} = \left[\mathbf{p}^T\ s_f\right]^T. \tag{2.1.5}$$

The general form of an optimization problem with equality boundary conditions is:

$$\begin{aligned} \text{minimize } J(\mathbf{q}) &= \phi(\mathbf{q}) + \int_0^{s_f} L(\mathbf{q})ds \\ \text{s.t. } \mathbf{g}(\mathbf{q}) &= \mathbf{0}, \quad s_f \text{ is free.} \end{aligned} \tag{2.1.6}$$

It is straightforward to see that $\kappa(0) = a$ and if we assume $x_0 = 0,\ y_0 = 0,\ \theta_0 = 0$ then we only need to consder $\mathbf{x}_f = [x_f\ y_f\ \theta_f\ \kappa_f]^T$ as the boundary condition. Hence, the constraint equation, $\mathbf{g}(\mathbf{q})$, is given by:

$$\mathbf{g}(\mathbf{q}) = \mathbf{x}(\mathbf{q}) - \mathbf{x}_f \tag{2.1.7}$$

where $\mathbf{x}(\mathbf{q}) = [x(\mathbf{q})\ y(\mathbf{q})\ \theta(\mathbf{q})\ \kappa(\mathbf{q})]^T$. Since $\kappa(0) = a$ parameter a, this parameter a does not need to be considered moving forward and the parameter vector $\mathbf{q}$ can be redefined as $\mathbf{q} = [b\ \ c\ \ d\ \ \cdots\ \ s_f]^T$.

2.2 Solution Method for Curvature Polynomial Problem

The solution can be obtained by using the MATLAB 'fmincon' solver. This solver directly solves problems with the form of (2.1.6). For the purposes of this thesis the 'Newtonian Interior Point Algorithm' was the chosen 'fmincon' algorithm. In its default setting, this algorithm will approximate the necessary first and second order conditions. This can lead to inaccurate solutions especially with a complex $\mathbf{g}(\mathbf{q})$, as is the case in this thesis, due to the integrals in $x(\mathbf{q})$ and $y(\mathbf{q})$. This problem can be avoided by directly supplying the gradients of $J(\mathbf{q})$ and $\mathbf{g}(\mathbf{q})$ to the 'fmincon' solver. Additionally, setting the Hessian approximation to use finite-differences was found to produce the most rapid and accurate results.

2.3 Generalized Curvature Polynomial Approach

The method of path generation through a track presented in this thesis involves connecting multiple optimized paths. However, as discussed in Section 2.1, the methods of generating optimized paths presented in [13] assumes that each path is generated from the origin of its own coordinate system. This requires several steps and presents complications when connecting the paths. Specifically let τ_k be the kth path segment that connects two boundary conditions, $\mathbf{x}_{f_{k-1}} = \begin{bmatrix} x_{f_{k-1}} & y_{f_{k-1}} & \theta_{f_{k-1}} & \kappa_{f_{k-1}} \end{bmatrix}^T$ and $\mathbf{x}_{f_k} = [x_{f_k} \; y_{f_k} \; \theta_{f_k} \; \kappa_{f_k}]^T$, represented in a fixed world coordinate system, C_w. Also let C_k be the coordinate system for τ_k such that $(x_{f_{k-1}}, y_{f_{k-1}}, \theta_{f_{k-1}})$ is the origin. To generate path τ_k, the coordinate frame C_k needs to be constructed through translation by $(x_{f_{k-1}}, y_{f_{k-1}})$ and rotation by $\theta_{f_{k-1}}$, from origin of the world coordinate system C_w. Next, the representation of $\mathbf{x}_{f_k}$ must be transformed into the C_k coordinate system from the world coordinate system, C_w. And then finally, we apply the methods presented in Section 2.1 to generate the path τ_k that starts from $\mathbf{x}_{f_{k-1}}$, $(0, 0, 0)$ in C_k, and ends at $\mathbf{x}_{f_k}$, represented in C_k. This method of connecting paths was successfully implemented but was clearly more complicated and messy than needed.

A much more convenient approach to connecting the paths is to generalize each path so that it can start from an arbitrary initial condition. If the state

of a vehicle is modeled as:

$$\begin{aligned} x(\mathbf{p}, s) &= x_0 + \int_0^s \cos\theta(\mathbf{p}, z)dz \\ y(\mathbf{p}, s) &= y_0 + \int_0^s \sin\theta(\mathbf{p}, z)dz \\ \theta(\mathbf{p}, s) &= \theta_0 + \kappa_0 s + \frac{bs^2}{2} + \frac{cs^3}{3} + \frac{ds^4}{4} + \cdots \\ \kappa(\mathbf{p}, s) &= \kappa_0 + bs + cs^2 + ds^3 + \cdots \end{aligned} \tag{2.3.1}$$

where $(x_0, y_0, \theta_0, \kappa_0)$ are the initial position coordinates, heading, and curvature, respectively. Then paths of arbitrary initial conditions can be generated using the same methods as described in Section 2.1. Although the generalized paths can be generated using the same methods, the derivatives required for optimization problem differ due to the changes in the equations modeling the state of the vehicle. These necessary derivatives are derived in the appendix.

2.4 Path with an Optimized Waypoint

Consider the following problem: Optimize the position of a single waypoint with respect to a cost function. The cost function is evaluated on the path from an initial state, $\mathbf{x}_0$, to a final state, $\mathbf{x}_f$, through the waypoint, $\mathbf{w} = [x_w\ y_w\ \theta_w\ \kappa_w]^T$ where $\mathbf{w}$ has the constraints:

$$\begin{aligned} &x_{lb} \leq x_w \leq x_{ub} \\ &y_{lb} \leq y_w \leq y_{ub} \\ &\theta_w, \kappa_w \text{ are free.} \end{aligned} \tag{2.4.1}$$

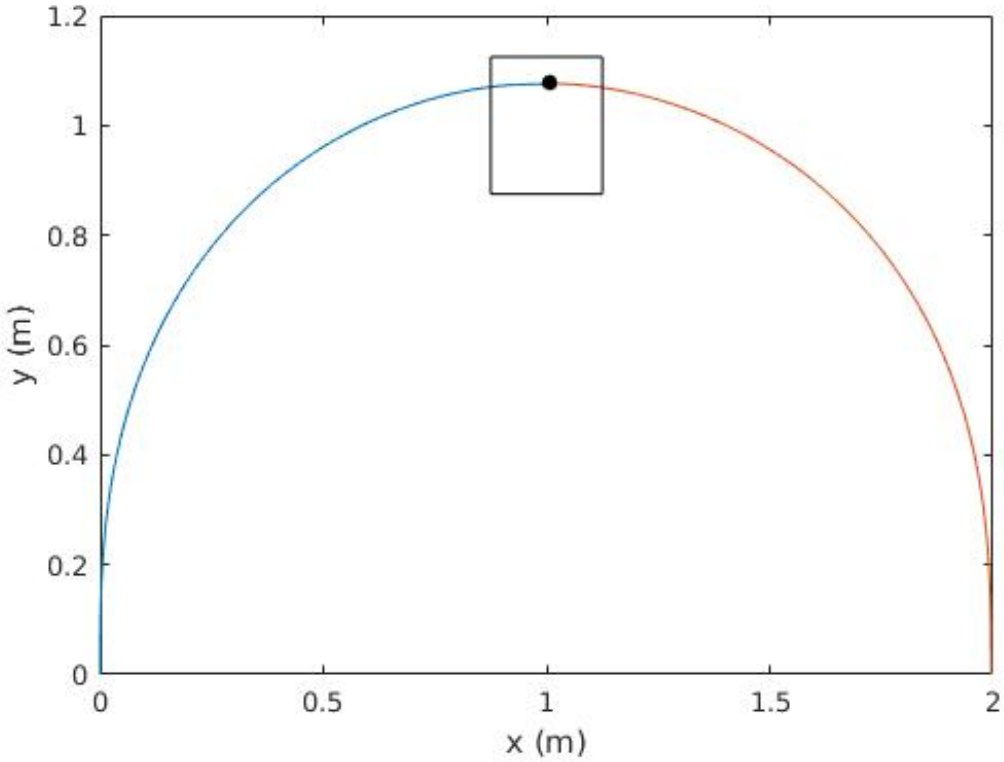

Figure 2.2: A path generated from $\mathbf{x}_0 = [2 \;\; 0 \;\; \pi/2 \;\; 0]$ to $\mathbf{x}_f = [0 \;\; 0 \;\; 3\pi/2 \;\; 0]^T$ passing through the waypoint with boundaries described by the square.

Let τ be the path that connects $\mathbf{x}_0$ and $\mathbf{x}_f$ passing through a given waypoint $\mathbf{w}$. Then τ can be decomposed into two subpath segments as follows:

$$\tau = \tau_1 \oplus \tau_2 \tag{2.4.2}$$

where τ_1 is the path from $\mathbf{x}_0$ to $\mathbf{w}$, τ_2 is the path from $\mathbf{w}$ to $\mathbf{x}_f$, and $\oplus$ denotes a concatenation operator that connects two paths. For each path τ_i $i = 1, 2$, τ_i is defined by the state equation (2.3.1) with parameters $\mathbf{q}_i = [b_i \;\; c_i \;\; d_i \;\; \cdots \;\; s_{fi}]^T$. Note that for τ_1 the initial conditions are simply $\mathbf{x}_0$ and for τ_2 the initial conditions are $\mathbf{w}$. An example of such a path τ is shown in Figure 2.2.

To optimize the location of a waypoint $\mathbf{w}$, within the constraints (2.4.1) and with respect to some cost function, the waypoint $\mathbf{w}$ must also be considered a parameter:

$$\mathbf{q}_3 = [x_w \; y_w \; \theta_w \; \kappa_w]^T . \tag{2.4.3}$$

Then the whole parameter vector for optimized path generation problem is:

$$\mathbf{q} = \left[\mathbf{q}_1^T \; \mathbf{q}_2^T \; \mathbf{q}_3^T\right]^T . \tag{2.4.4}$$

It is now possible to formulate this problem as a constrained optimization problem using the form given in (2.1.6) with a constraint equation of the form:

$$\mathbf{g}(\mathbf{q}) = \begin{bmatrix} \mathbf{g}_1(\mathbf{q}) \\ \mathbf{g}_2(\mathbf{q}) \end{bmatrix} \tag{2.4.5}$$

where $\mathbf{g}_1(\mathbf{q}) = [g_{1x} \; g_{1y} \; g_{1\theta} \; g_{1\kappa}]^T$ is the constraint equation corresponding to τ_1 and $\mathbf{g}_2(\mathbf{q}) = [g_{2x} \; g_{2y} \; g_{2\theta} \; g_{2\kappa}]^T$ is the constraint equation corresponding to τ_2.

If we let $\mathbf{x}_1(\mathbf{q})$ and $\mathbf{x}_2(\mathbf{q})$ be the state vectors for τ_1 and τ_2, respectively, then

$$\begin{aligned} \mathbf{g}_1(\mathbf{q}) &= \mathbf{x}_1(\mathbf{q}) - \mathbf{q}_3 \\ \mathbf{g}_2(\mathbf{q}) &= \mathbf{x}_2(\mathbf{q}) - \mathbf{x}_f. \end{aligned} \tag{2.4.6}$$

This problem can then be solved in the same way as the path generation problem in the previous section. Derivatives of the constraints equation with respect to the parameters are

$$\frac{\partial \mathbf{g}}{\partial \mathbf{q}} = \begin{bmatrix} \dfrac{\partial \mathbf{g}_1}{\partial \mathbf{q}} \\ \\ \dfrac{\partial \mathbf{g}_2}{\partial \mathbf{q}} \end{bmatrix} = \begin{bmatrix} \dfrac{\partial \mathbf{g}_1}{\partial \mathbf{q}_1} & \dfrac{\partial \mathbf{g}_1}{\partial \mathbf{q}_2} & \dfrac{\partial \mathbf{g}_1}{\partial \mathbf{q}_3} \\ \\ \dfrac{\partial \mathbf{g}_2}{\partial \mathbf{q}_1} & \dfrac{\partial \mathbf{g}_2}{\partial \mathbf{q}_2} & \dfrac{\partial \mathbf{g}_2}{\partial \mathbf{q}_3} \end{bmatrix} \tag{2.4.7}$$

where, for $i, j = 1, 2$, $\frac{\partial \mathbf{g}_i}{\partial \mathbf{q}_j}$ are defined as

$$\frac{\partial \mathbf{g}_i}{\partial \mathbf{q}_j} = \begin{bmatrix} \dfrac{\partial g_{ix}}{\partial b_j} & \dfrac{\partial g_{ix}}{\partial c_j} & \dfrac{\partial g_{ix}}{\partial d_j} & \cdots & \dfrac{\partial g_{ix}}{\partial s_{f_j}} \\ \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \\ \dfrac{\partial g_{i\kappa}}{\partial b_j} & \dfrac{\partial g_{i\kappa}}{\partial c_j} & \dfrac{\partial g_{i\kappa}}{\partial d_j} & \cdots & \dfrac{\partial g_{i\kappa}}{\partial s_{f_j}} \end{bmatrix} \tag{2.4.8}$$

and, for $j = 3$,

$$\frac{\partial \mathbf{g}_i}{\partial \mathbf{q}_3} = \begin{bmatrix} \frac{\partial g_{ix}}{\partial x_w} & \frac{\partial g_{ix}}{\partial y_w} & \frac{\partial g_{ix}}{\partial \theta_w} & \frac{\partial g_{ix}}{\partial \kappa_w} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial g_{i\kappa}}{\partial x_w} & \frac{\partial g_{i\kappa}}{\partial y_w} & \frac{\partial g_{i\kappa}}{\partial \theta_w} & \frac{\partial g_{i\kappa}}{\partial \kappa_w} \end{bmatrix}. \tag{2.4.9}$$

A detailed derivation of these partial derivatives is in the appendix.

2.5 Choice of Cost Function and Number of Parameters

This section describes the selection of the cost function. The cost function used is:

$$J(\mathbf{q}) = \frac{1}{2} \int_0^{s_f} \kappa^2(\mathbf{p}, s) ds. \tag{2.5.1}$$

This cost function prioritizes a smooth path. But this cost function does not necessarily lead to the most optimal path in the sense of travel time, as the increased path smoothness may lead to a longer path. The derivatives associated with this cost function are detailed in in the appendix.

The optimization problem of generating a single path segment has four constraints, that is $\mathbf{g}(\mathbf{q})$ contains four equations. As a result it is necessary to include at least four parameters in $\mathbf{q}$ to obtain a solution which satisfies the constraints. Considering s_f as one of the parameters, the assumed form of $k(s)$ must be at least a second order polynomial. One additional parameter

is used to allow for optimality in the solution. Therefore $k(s)$ is a third order polynomial and the full parameter vector is $\mathbf{q} = [a\ b\ c\ d\ s_f]^T$.

Chapter 3

Closed-Loop Path Generation

3.1 An Iterative Approach for Loop Waypoints Optimization

Consider a car traveling through a track that is defined by a series of waypoints, each defined as in (2.4.1), and no boundaries. This is demonstrated in Figure 3.1.

In order to solve for the optimal path through all of these waypoints, the position of each waypoint (within its bounds) is optimized as in Section 2.4 and then the waypoints are connected with optimal paths. To give more detailed discussion on this process, let us first define

$$wp\_opt(\mathbf{x}_0, \mathbf{x}_f, \mathbf{w}, \mathbf{b}) \tag{3.1.1}$$

as a function that uses the methods of Section 2.4 to optimize the position of a waypoint, $\mathbf{w}$, such that the path from the initial state, $\mathbf{x}_0$, through the

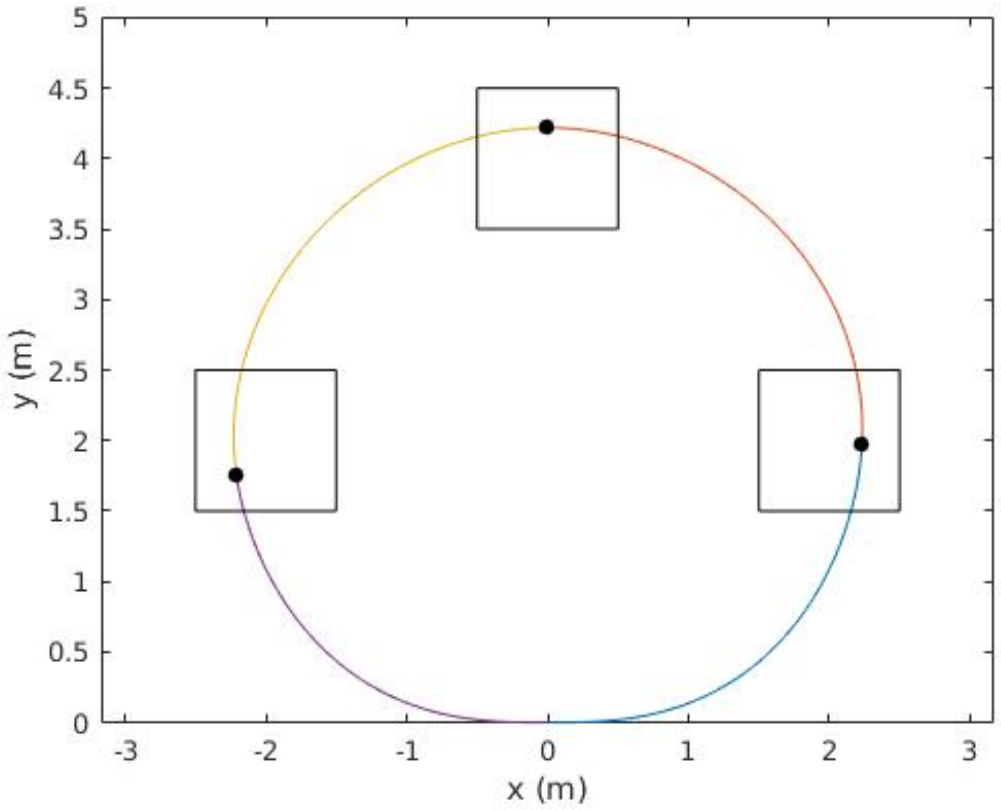

Figure 3.1: A path generated from $\mathbf{x}_0 = [0 \;\; 0 \;\; 0 \;\; 0]^T$ to $\mathbf{x}_f = [0 \;\; 0 \;\; 2\pi \;\; 0]^T$, passing through the waypoints with boundaries described by the squares. The dots denote the optimized waypoint positions.

waypoint, $\mathbf{w}$, and to the final state, $\mathbf{x}_f$, is optimal in the sense of some cost function, such as the one described in Section 2.5. The function's inputs are the initial and final states, the initial waypoint position $\mathbf{w}$, and a vector,

$$\mathbf{b} = [x_{lb} \;\; x_{ub} \;\; y_{lb} \;\; y_{ub}]^T, \tag{3.1.2}$$

which denotes the bounds on the waypoint as defined in (2.4.1). The function outputs the optimized waypoint position.

Additionally, we define

$$path\_opt(\mathbf{x}_0, \mathbf{x}_f) \tag{3.1.3}$$

as a function that uses the methods of Section 2.3 to optimize a path, in the sense of the same cost function as above. This function takes initial and final states, $\mathbf{x}_0$ and $\mathbf{x}_f$, as inputs and outputs an optimized path. The output is in the form of a vector containing the parameters which describes the path, thus for path i, $\tau_i = \mathbf{q}_i = [a_i \; b_i \; c_i \; d_i \; s_{fi}]^T$.

The final function defined is

$$cost\_function(\tau). \tag{3.1.4}$$

This function takes a matrix of path descriptions, τ, as an input and outputs the value of the sum of the same cost function used in (3.1.1) and (3.1.3) for all the inputted paths. In the input each column vector contains the parameters describing a single path.

Algorithm 1 Loop Waypoint Optimization

Input: Input: A vector $\mathbf{x}_0 = [x_0 \quad y_0 \quad \theta_0 \quad \kappa_0]^T$ which denotes the initial and find states, and a matrix $\mathbf{B} = [\mathbf{b}_1 \quad \mathbf{b}_2 \quad \cdots \quad \mathbf{b}_n]$ where each vector $\mathbf{b}_i$ contains the bounds corresponding to the ith waypoint as defined in (3.1.2).
Output: The matrix $\mathbf{W} = [\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_n]$ where each vector $\mathbf{w}_i$ is the optimized waypoint, and a matrix $\tau = [\tau_1 \quad \tau_2 \quad \cdots \quad \tau_{n+1}]$ that contains the paths connecting the waypoints.
Initialization:
$\Delta J = 1$
$j = 0$
Choose $\mathbf{w}_i$ within $\mathbf{b}_i$, $\forall i \in 1, \cdots, n$
$\mathbf{X} = [\mathbf{x}_0 \quad \mathbf{w}_1 \quad \mathbf{w}_2 \quad \ldots \quad \mathbf{w}_n \quad \mathbf{x}_0]$
$\tau = [\tau_1 \quad \tau_2 \quad \cdots \quad \tau_n]$
LOOP Process:
while $\Delta J >$ Tolerance **do**
  $j = j + 1$
  for $i = 2 \cdots n + 1$ **do**
    $\mathbf{w}_i = wp\_opt(\mathbf{X}_{i-1}, \mathbf{X}_{i+1}, \mathbf{X}_i, \mathbf{B}_{i-1})$
    $\mathbf{X}_i = \mathbf{w}_i$
  end for
  for $i = 1 \cdots n + 1$ **do**
    $\tau_i = path\_opt(\mathbf{X}_i, \mathbf{X}_{i+1})$
  end for
  $J(j) = cost\_function(\tau)$
  if $j > 1$ **then**
    $\Delta J = |J(j) - J(j-1)|$
  end if
end while
$\mathbf{W} = [\mathbf{X}_2 \quad \mathbf{X}_3 \quad \cdots \quad \mathbf{X}_n]$
return τ, $J(j)$, $\mathbf{W}$

The pseudo-code for the waypoint optimization algorithm is presented in Algorithm 1. Note that, in Algorithm 1, the cost function is built in to (3.1.1) and (3.1.3). The notation $\mathbf{X}_i$ denotes the ith element in $\mathbf{X}$. τ_i is used in the same manner, and $\mathbf{w}_i$ is a local variable. The while loop in Algorithm 1 is stopped when the difference in the cost function from the $(j-1)$th to

jth iteration is below a tolerance. Figure 3.2 demonstrates the asymptotic behavior of Algorithm 1 for the generation of the path shown in Figure 3.1. Figures 3.4 and 3.6 also show this asymptotic behavior for the paths in 3.3 and 3.5. This asymptotic behavior, along with the condition described above, ensures that an optimal solution is reached. Note that the solution is for the given initial conditions i.e. it could be a local minimum. The tolerance used in this paper is .001.

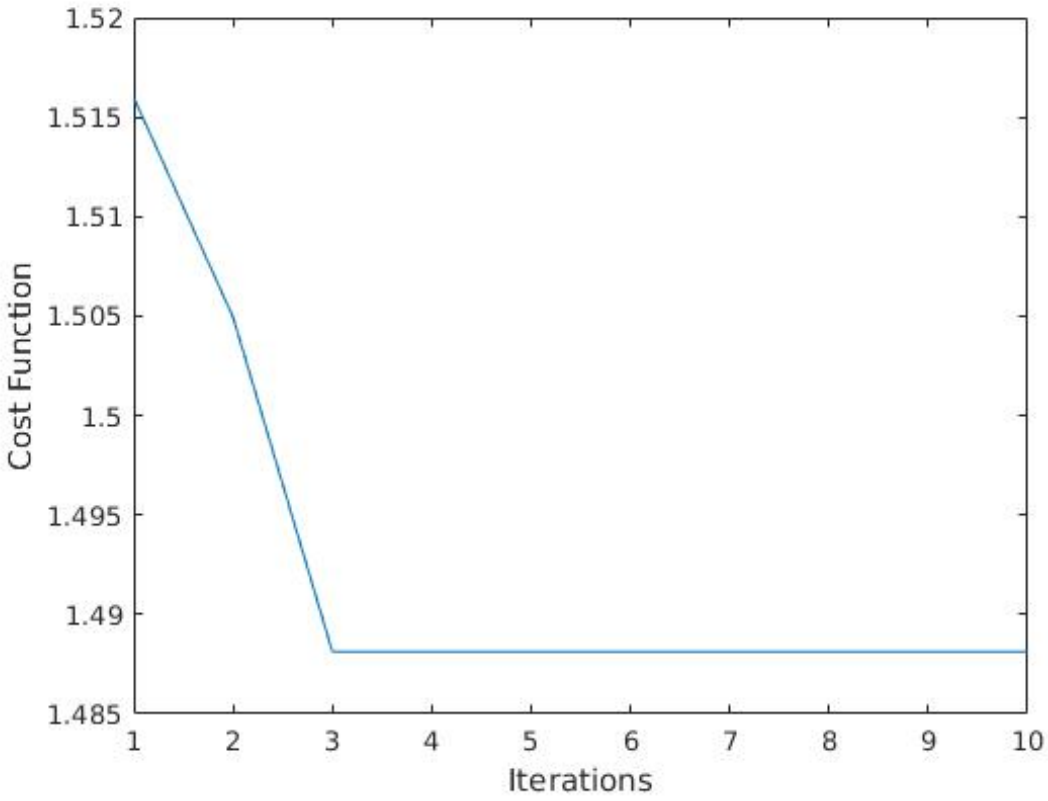

Figure 3.2: Cost function versus iterations of Algorithm 1 for the path generation shown in Figure 3.1.

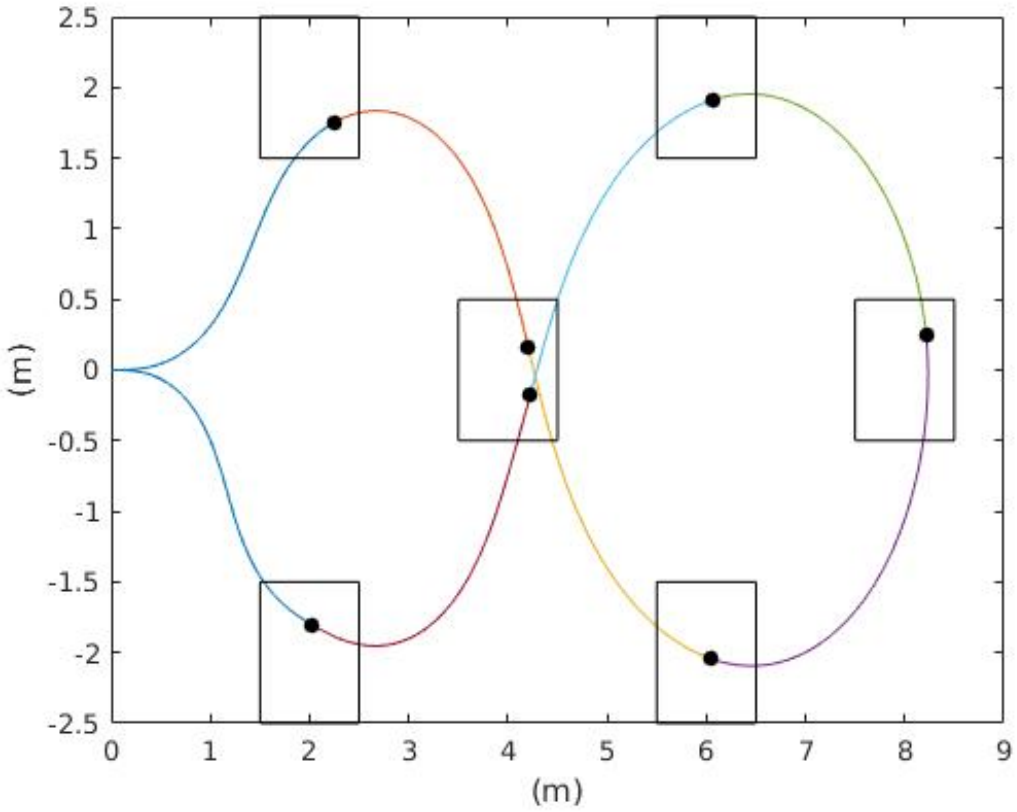

Figure 3.3: A path generated from $\mathbf{x}_0 = [0 \quad 0 \quad 0 \quad 0]^T$ to $\mathbf{x}_f = [0 \quad 0 \quad \pi \quad 0]^T$ passing through the waypoints with boundaries described by the squares. The dots denote the optimized waypoint positions.

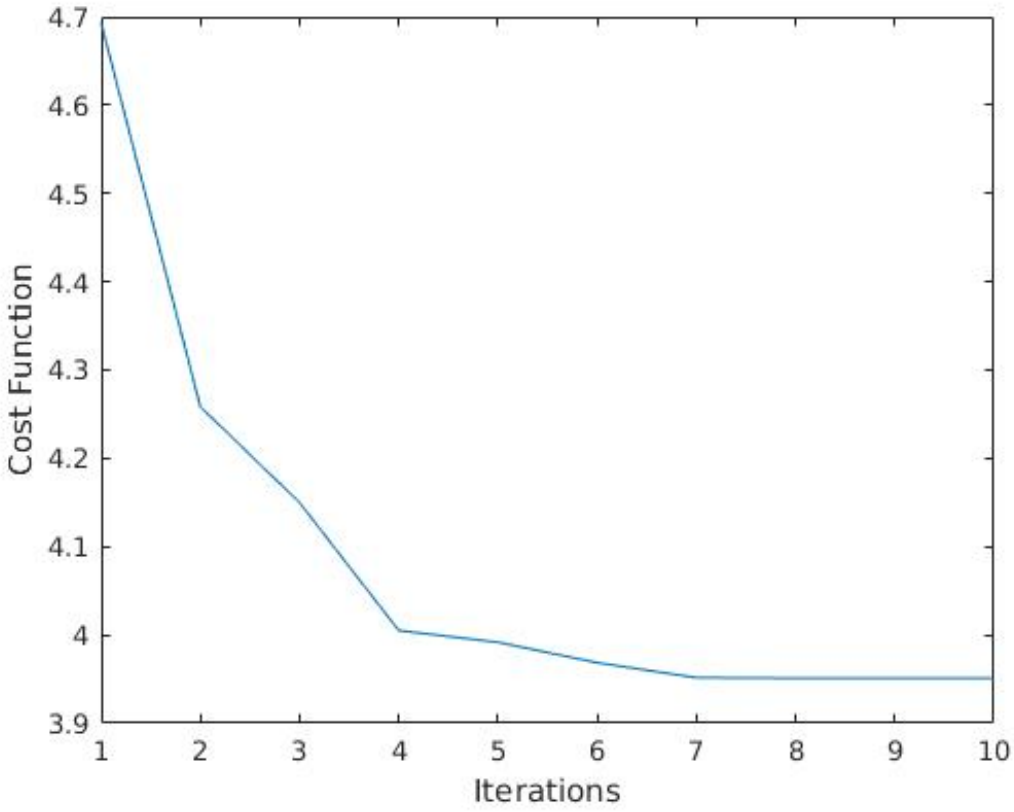

Figure 3.4: Cost function versus iterations of Algorithm 1 for the path generation shown in Figure 3.3.

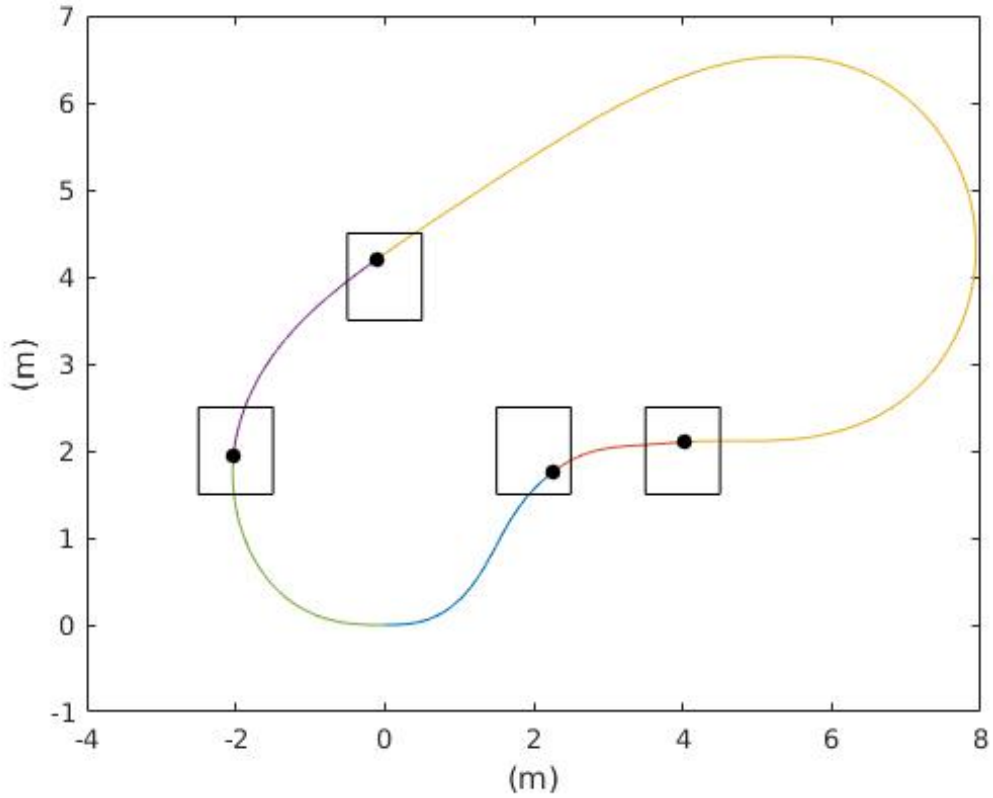

Figure 3.5: A path generated from $\mathbf{x}_0 = [0 \quad 0 \quad 0 \quad 0]^T$ to $\mathbf{x}_f = [0 \quad 0 \quad \pi \quad 0]^T$ passing through the waypoints with boundaries described by the squares. The dots denote the optimized waypoint positions.

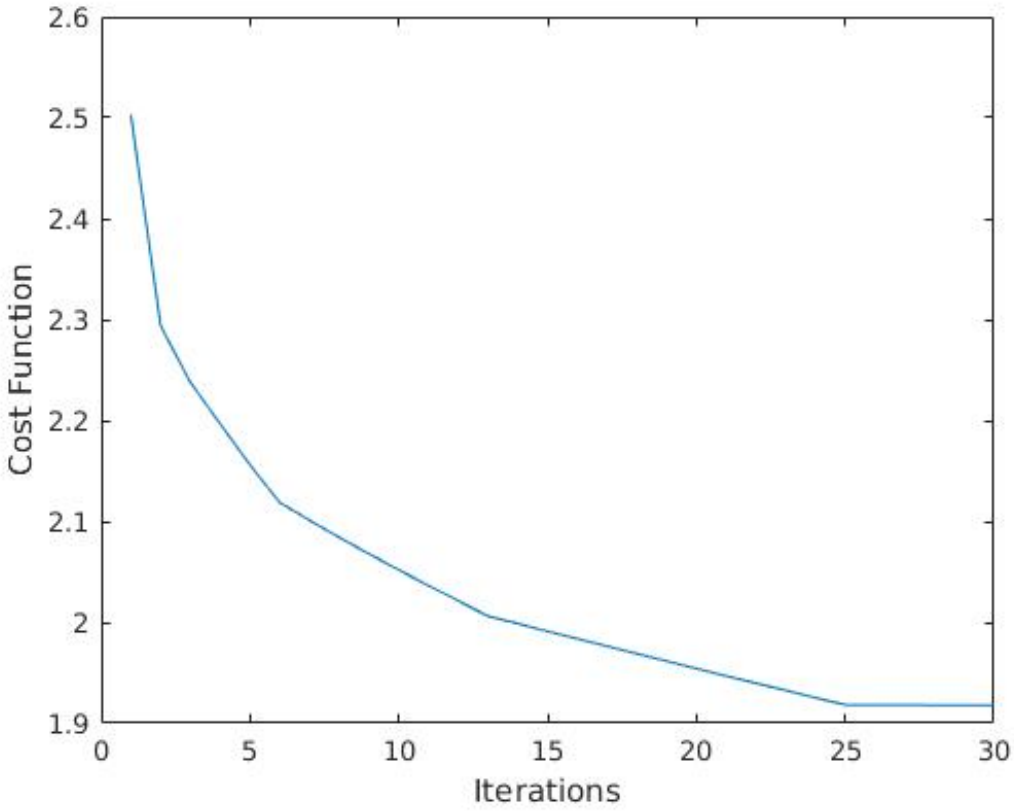

Figure 3.6: Cost function versus iterations of Algorithm 1 for the path generation shown in Figure 3.5.

It is important to notice that in new iterations each waypoint is optimized with updated starting and ending states. This is the key factor that allows this iterative process to find an optimal solution.

3.2 Obstacle Avoidance

The same general process described in the above section can be used to solve for an optimal path through a track with boundaries. Waypoints are chosen throughout the track, optimized, and then connected by optimal paths. This general process integrated with track boundary collision avoidance obstacles is shown in Algorithm 2.

Algorithm 2 **T**rack **P**ath **O**ptimization (TPO) Algorithm

Input: The description of a track, denoted by `Track`
Output: The optimal path through the track
Initialization:
$J = 0$
in_track = false
LOOP Process:
while true **do**
  Run Algorithm 1 $\rightarrow \tau_{new}, J_{new}$
  $\Delta J = J_{new} - J$
  if in_track & $\Delta J > 0$ **then**
    break
  else
    $J = J_{new}$
    $\tau = \tau_{new}$
    in_track = test_in_track(τ, `Track`)
    Update waypoints selections
  end if
end while

The function *test_in_track*(τ, `Track`) takes a path, τ, and a description of a track, `Track`, as inputs and outputs a Boolean indicating if the path is entirely

within the track. Remembering that Algorithm 1 outputs an optimized path connecting a series of waypoints, this algorithm is fairly straight forward. The implementation of $test\_in\_track(\tau, \texttt{Track})$ was completed by representing the track as a polygon and the path as a set of (x, y) points. The MATLAB 'inpolygon' function was then used to check for collision of the path and track. The 'inpolygon' function takes vectors representing the x and y points of the inner and outer boundaries of a polygon and checks if all points in an inputed set are within this polygon. Details of this track representation are in the next section.

3.2.1 Track Representation

How the track is represented can allow for easier selection, initialization, and updating of waypoints. The methods of representing the track were formulated with this in mind and the reasons for this formulation will become more clear through the discussion of updating waypoint selection.

Two vectors represent the inner track: one representing x values and one representing y values. The inner line of the track is composed by connecting the corresponding set of x and y points. This set of points is used to construct the center and outer lines of the track. Figure 3.7 shows an example representation of a track inner line.

Constructing the vectors representing the center and outer lines is completed through the use of transformation matrices. Consider the problem of transforming a point in a coordinate system, C_1, to the world coordinate system, C_w. If (x, y) is the coordinates of the base of C_1, represented in C_w, and θ represents the angle of rotation from C_w to C_1 then the following transfor-

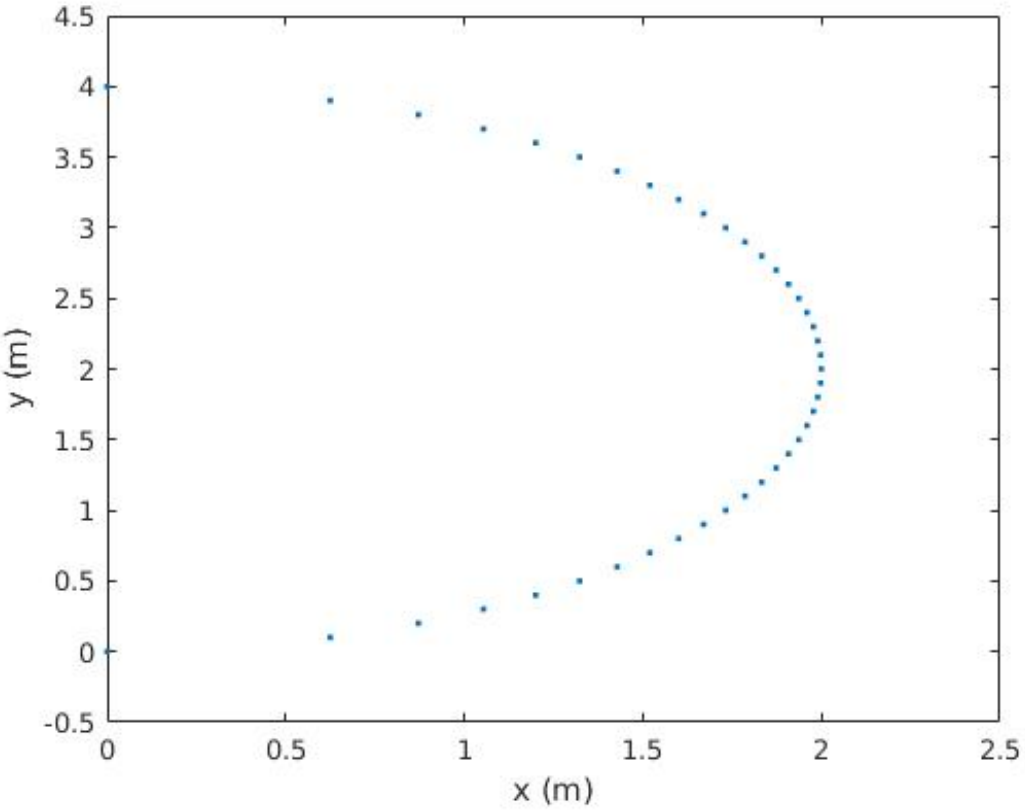

Figure 3.7: Example track inner line represented by two vectors, $V1$ and $V2$. $V1 = [0 : 0.1 : 4]$ (a vector from 0 to 4 with increment size 0.1) and $V2 = \sqrt{4 - (V1 - 2)^2}$. $V1$ represents the y values of the inner line and $V2$ represents the x values of the inner line.

mation matrix transforms a point from C_1 to C_w

$$\mathbf{T} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & x \\ \sin(\theta) & \cos(\theta) & y \\ 0 & 0 & 1 \end{bmatrix}. \tag{3.2.1}$$

The point $\mathbf{p} = [p_x \ p_y]^T$, in C_1 coordinates, can be transformed to C_w by

$$\begin{bmatrix} \tilde{\mathbf{p}} \\ 1 \end{bmatrix} = \mathbf{T} \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} \tag{3.2.2}$$

where $\tilde{\mathbf{p}}$ is the point $\mathbf{p}$ in C_w. p_x is the x coordinate of $\mathbf{p}$ and p_y is the y coordinate of $\mathbf{p}$. In order to construct the center and outer track lines, define $\mathbf{p}_i = [x_i \ y_i]^T$ to be a point along the inner track line, $\mathbf{p}_c = [x_c \ y_c]^T$ and $\mathbf{p}_o = [x_o \ y_o]^T$ to be the points along the center and outer track lines corresponding to $\mathbf{p}_i$, w to be the track width, and θ to be the angle of the tangent to the inner track line at $\mathbf{p}_i$. Also define C_i to be a coordinate system where $\mathbf{p}_i$, represented in C_w, is the base and θ is the angle of rotation from C_w to C_i. Then $\mathbf{p}_c$ and $\mathbf{p}_o$, in C_i, are $[0 \ -w/2]^T$ and $[0 \ -w]^T$, respectively. Using Equations (3.2.1) and (3.2.2), $\mathbf{p}_c$ and $\mathbf{p}_o$ can be transformed to C_w giving the following relations:

$$\begin{aligned} x_c &= \frac{w}{2}\sin(\theta) + x_i \\ y_c &= -\frac{w}{2}\cos(\theta) + y_i \\ x_o &= w\sin(\theta) + x_i \\ y_o &= -w\cos(\theta) + y_i. \end{aligned} \tag{3.2.3}$$

θ is calculated by:

$$\theta = \arctan\left(\frac{y_i - y_{i-1}}{x_i - x_{i-1}}\right) \tag{3.2.4}$$

where $[x_{i-1}\ y_{i-1}]^T$ denotes the point before $\mathbf{p}_i$ along the inner line. The center and outer line vectors are calculated by iteratively using this process on all points in the inner line. Figure 3.8 shows an example of this method. With this formulation the track can be represented as a polygon and used to check if a path is within its boundaries. As mentioned earlier the MATLAB function 'inpolygon' is used to do this testing.

3.2.2 Selecting Initial Waypoints and Waypoint Boundaries

Initial waypoints are selected so that they are evenly spaced along the center line. The number of initial waypoints depends on the shape of the track with more complicated tracks requiring more waypoints. Note that the choice of the amount of initial waypoints only affect computational efficiency, i.e. if the choice is close to the optimal number of waypoints it will take less iterations to find this optimal number of waypoints. If n waypoints are selected and s_f is the length of the track center line, then the waypoints are initially positioned at intervals of $\frac{s_f}{n}$. Given the representation of the track from the previous sub-section, the distance along the center line from the start to the nth point on the center line is:

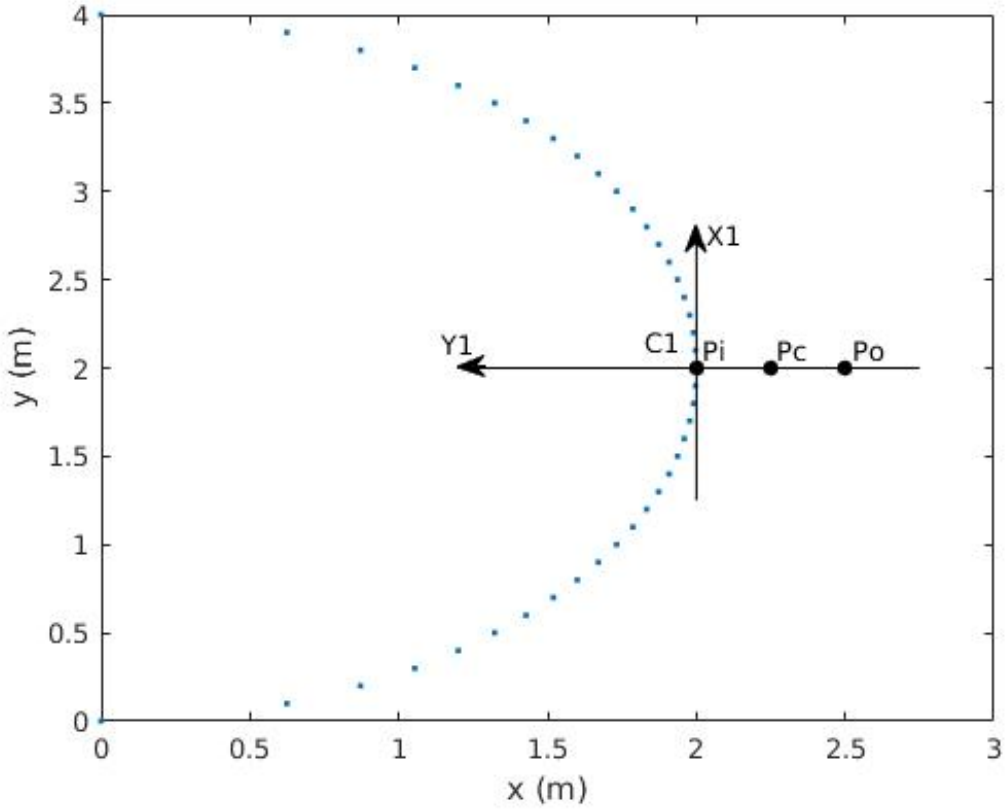

Figure 3.8: Example of calculating the points on the center and outer track lines, $\mathbf{p}_c$ and $\mathbf{p}_o$, corresponding to a point on the inner track line, $\mathbf{p}_i$. $\mathbf{p}_i = [2\ 2]^T$ and θ, the angle of the tangent at $\mathbf{p}_i$, is π. The base of C_i is translated by $(2, 2)$ and rotated by π, from C_w. The track width is .5 giving $\mathbf{p}_c = [0\ -.25]^T$ and $\mathbf{p}_o = [0\ -.5]^T$, in C_i. Using Equation (3.2.3), $\mathbf{p}_c$ and $\mathbf{p}_o$ can be transformed to C_w and are [2.25 2] and [2.5 2], respectively.

$$s_n = \sum_{i=2}^{n} \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \tag{3.2.5}$$

where (x_i, y_i) is point i on the center line and (x_{i-1}, y_{i-1}) is point $i-1$ on the center line. The initial θ for the waypoints is selected to be the angle of the tangent to the center line at the waypoint and can be calculated with (3.2.4). The curvature, k, values are initially all set to 0.

The waypoint boundaries must be defined so that they are fully within the track. The waypoint boundaries are described as follows:

$$\begin{aligned} x_{lb} &= x_{wc} - \frac{w}{2\sqrt{2}} \\ x_{ub} &= x_{wc} + \frac{w}{2\sqrt{2}} \\ y_{lb} &= y_{wc} - \frac{y}{2\sqrt{2}} \\ y_{ub} &= y_{wc} + \frac{w}{2\sqrt{2}} \end{aligned} \tag{3.2.6}$$

where w is the track width and $x_{ub},\ y_{ub},\ x_{lb},\ y_{lb}$ are the upper and lower bounds on the waypoint. The point (x_{wc}, y_{wc}) is defined as the center point of the waypoint and is always a point on the center line of the track. Given (3.2.6) the boundaries of any waypoint can be described by the center point and track width. Equation (3.2.6) is derived by inscribing a square within a circle of radius $\frac{w}{2}$ centered on (x_{wc}, y_{wc}). The resulting square has a width of $\frac{w}{\sqrt{2}}$. Since the track is defined to have a constant width at all points, these boundary conditions guarantee that the waypoint will always be within the track.

3.2.3 Updating the Waypoints

Two simple methods for updating the number of waypoints were explored: (i) Consecutively multiplying the number of waypoints by some number at the end of each iteration, and (ii) Consecutively adding some number to the number of waypoints at the end of each iteration. Once the number of waypoints is updated they are positioned along the track center line as described in Section 3.2.2.

The first method was generally faster at solving for a trajectory but could lead to a less optimal solution. Since Algorithm 2 iteratively optimizes each waypoint, using the previous and next waypoints as initial and final states, if the waypoints are too close together than the outputted path will follow closely along the track centerline. The second method led to more optimal solutions but can be at the cost of computational efficiency, if the number of starting waypoints is far off from the optimal number. In that case many iterations must be run to find a solution.

Figures 3.9 and 3.10 show paths through a simple track and demonstrate the two different waypoint update methods. Figure 3.9 shows the waypoints updated by adding one waypoint each iteration and Figure 3.10 shows the waypoints updated by doubling the number of waypoints each iteration. These Figures demonstrate that adding one waypoint each iteration led to a more optimal solution. Results on more complicated tracks are contained in the Chapter 5.

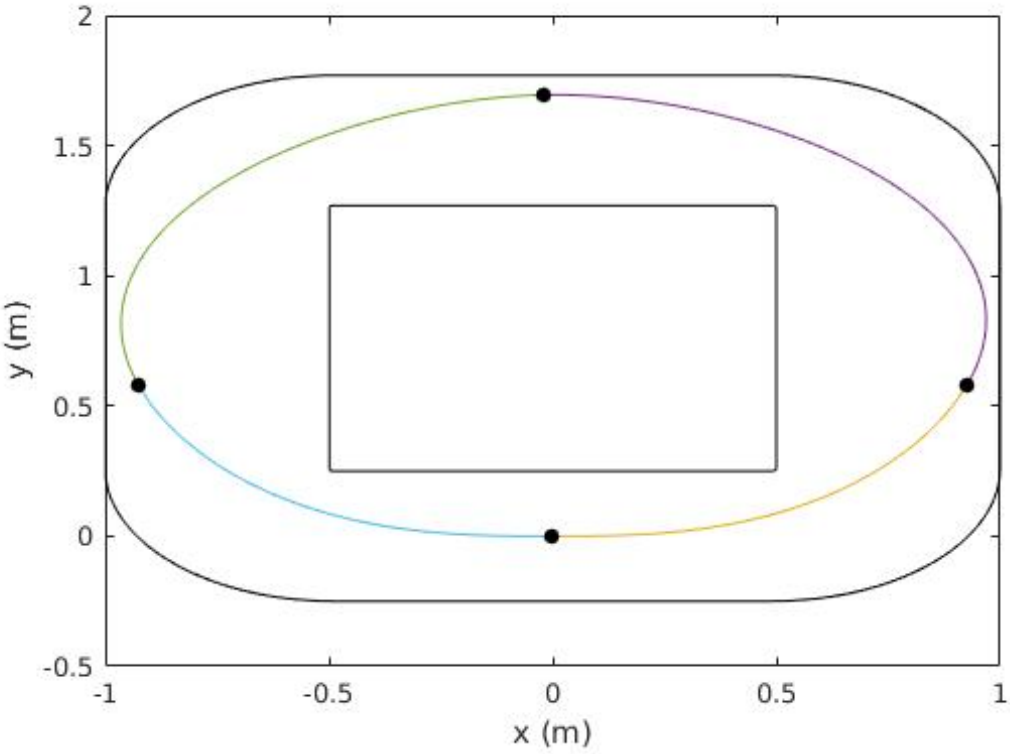

Figure 3.9: Path through track 1. Start and end point is $(0, 0)$. Number of waypoints: 3, marked by the black dots. Waypoint update method: adding one waypoint each iteration. Ending cost function value: 3.614.

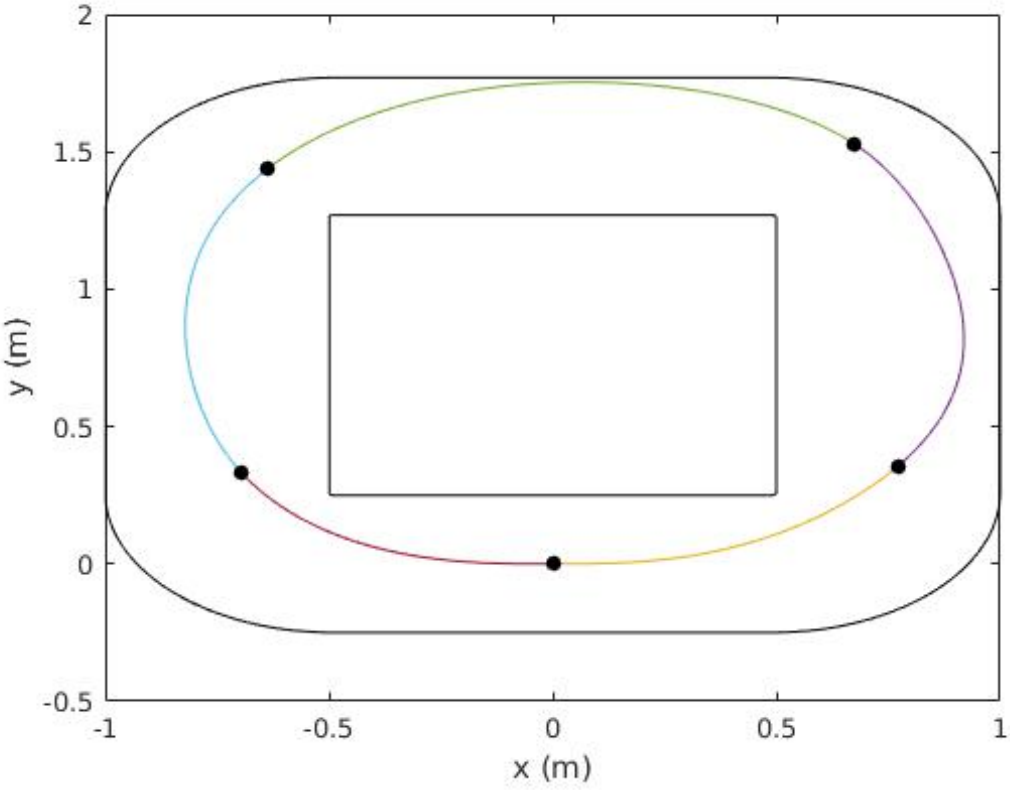

Figure 3.10: Path through track 1. Start and end point is $(0, 0)$. Number of waypoints: 4, marked by the black dots. Waypoint update method: multiplying the number of waypoints by two each iteration. Ending cost function value: 3.775.

Chapter 4

Velocity Profile Generation Method

In order to create trajectories from the optimized paths a velocity profile was created using the methods described in [31] and [30]. Profiles were created for both point mass and a dynamic models. This chapter directly outlines the methods from [31] and [30] used to create the velocity profiles in this thesis.

4.1 Point Mass Velocity Profile Formulation

The problem, as given in [31], for a point mass traveling through a path is as follows: Given acceleration limits and fixed boundary conditions (initial and final position and velocity), find the velocity profile for minimum travel time. The path is described by its curvature as a function of path length, s. The equations of motion, for the point mass, are given by:

$$\begin{aligned} m\frac{d^2s}{dt^2} &= f_t \\ m\left(\frac{ds}{dt}\right)^2 k(s) &= f_n \end{aligned} \tag{4.1.1}$$

where $\kappa(s)$ is the curvature function describing the path, m is the mass, f_t is the force tangential to the path, and f_n is the force normal to the path. Considering given acceleration limits on the tangential force, f_t^{max}, and normal force, f_n^{max}, equation (4.1.1) leads to the following relation:

$$\left(\frac{f_t}{f_t^{max}}\right)^2 + \left(\frac{f_n}{f_n^{max}}\right)^2 - 1 \leq 0. \tag{4.1.2}$$

In [31] it is formally proven that in order to achieve minimum travel time through a path the maximum available force must be used at all times. The following control law allows this to be achieved:

$$f_t = u\sqrt{(f_t^{max})^2 - m\kappa(s)\left(\frac{f_t^{max}}{f_n^{max}}\right)^2\left(\frac{ds}{dt}\right)^2}. \tag{4.1.3}$$

Both u and f_t are considered inputs with $u \in [-1, +1]$. Maximum force is achieved by setting $u = -1$ when the vehicle is decelerating and $u = +1$ when the vehicle is accelerating. There also exist a critical velocity,

$$\left(\frac{ds}{dt}\right)_{critical} = v_{critical} = f_n^{max}\sqrt{\frac{1}{m\kappa(s)}}, \tag{4.1.4}$$

where $f_t = 0$. At this $v_{critical}$ there is a loss of controllability since $f_t = 0$. Therefore $v_{critical}$ is the maximum allowable velocity along the trajectory. The method of constructing a velocity profile from the above control laws and

relations is as follows:

- Find the local maximum points of the absolute value of the path curvature. Figure 4.1 shows the absolute value curvature profile of a sample path.

- For each of the local maximum a velocity profile is constructed so that at the local maximum $v = v_{critical}$. $u = -1$ is used before the maximum and $u = +1$ after the maximum. Figure 4.2 shows the velocity profiles constructed from each of the points of local maximum shown in Figure 4.1.

- The profile from the given initial condition is constructed with $u = +1$. For the final condition a velocity profile is constructed with $u = -1$, ending at the final condition. Figure 4.2 also shows the profiles from the initial condition and leading into the final condition.

- The optimal velocity profile is then the minimum of all of the above profiles at each point along the trajectory. The optimal velocity profile for the trajectory described by Figure 4.1 is shown in Figure 4.3.

Given an initial velocity, v_0, acceleration, a, and distance traveled, Δs, the following equation is used to calculate the velocity after traveling Δs:

$$v = \sqrt{2a\Delta s + {v_0}^2}. \tag{4.1.5}$$

Equations (4.1.3) and (4.1.5) can be combined to give the discrete integration relationship:

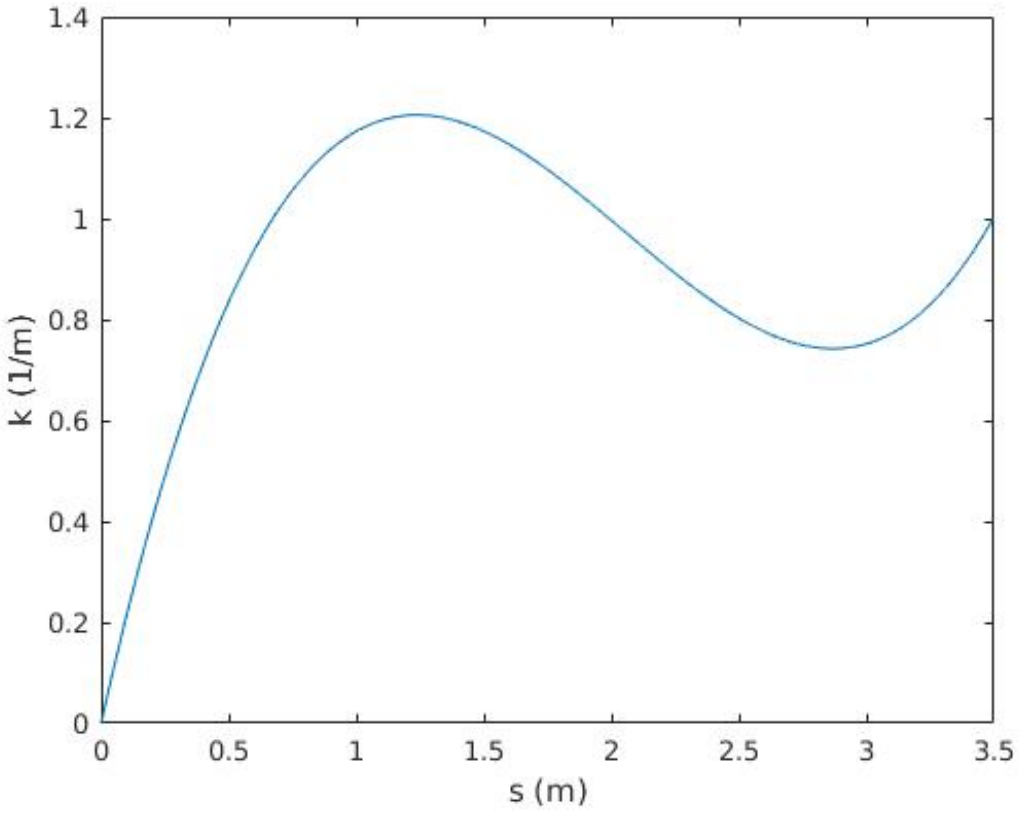

Figure 4.1: Curvature of a path with respect to distance traveled.

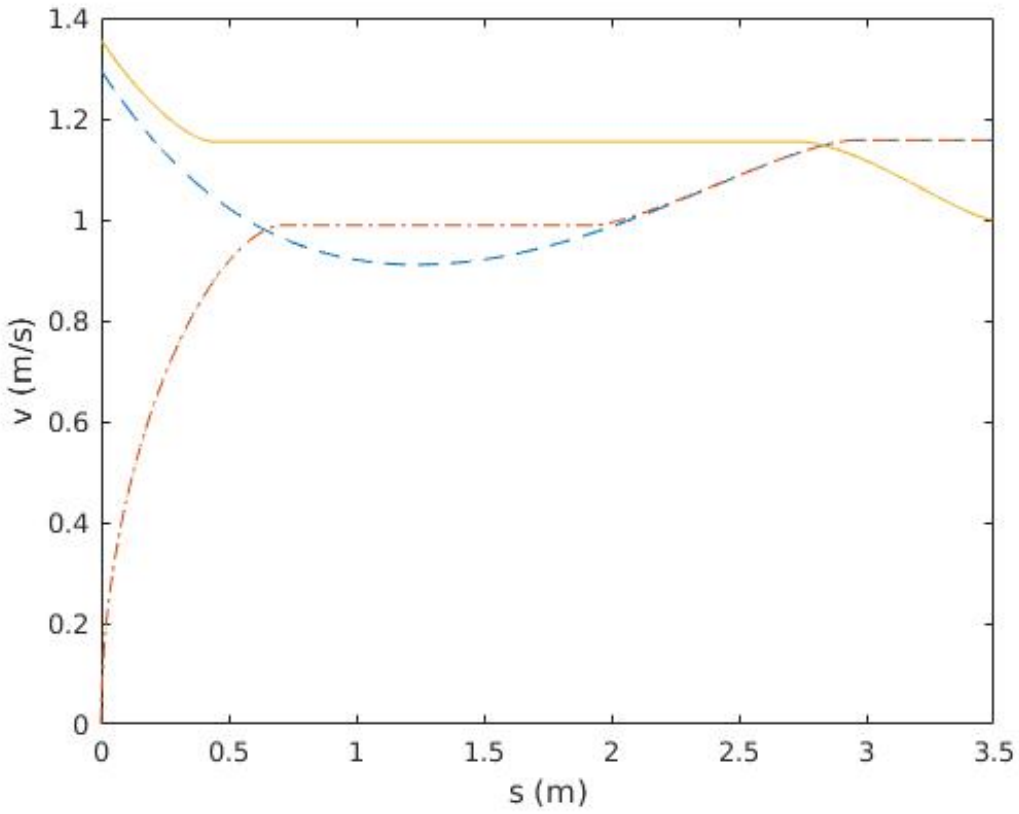

Figure 4.2: Constructed velocity profiles from the path described by Figure 4.1. The solid line is the profile constructed leading into the final condition, the dashed line is constructed from the curvature maximum, and the dashed and dotted line is constructed from the initial condition.

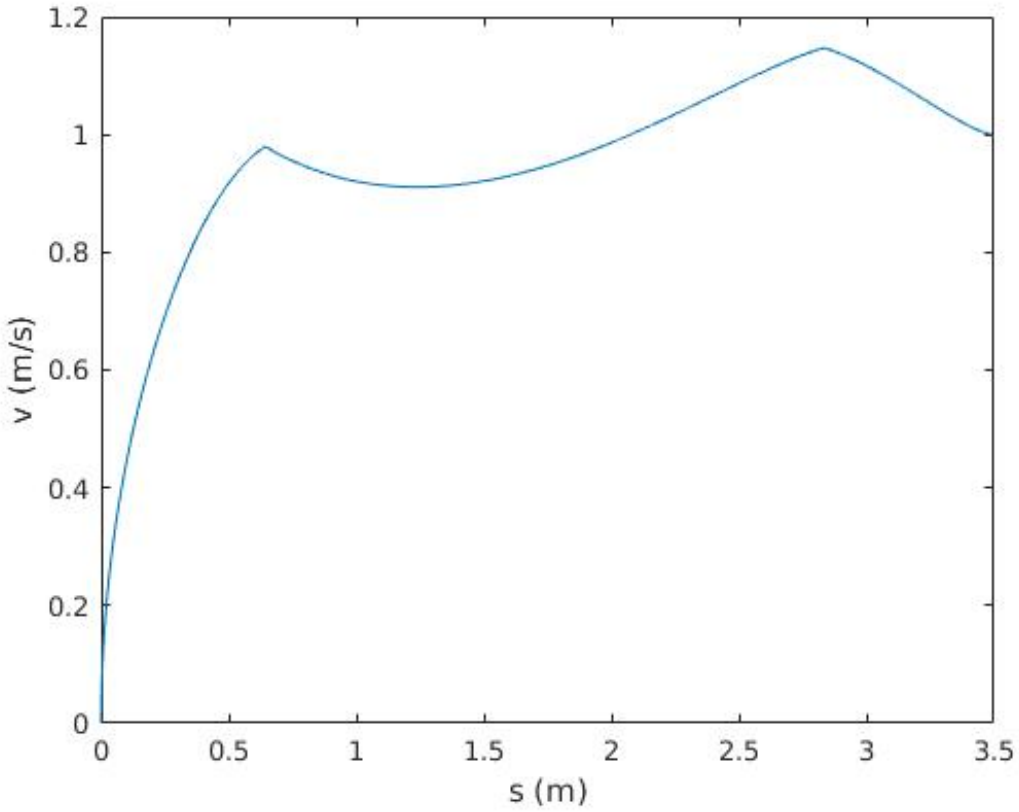

Figure 4.3: The time optimal velocity profile constructed as the minimum of the velocity profiles in Figure 4.2.

$$v_i = \sqrt{\frac{2u\sqrt{(f_t^{max})^2 - m\kappa(s_{i-1})\left(\frac{f_t^{max}}{f_n^{max}}\right)^2 v_{i-1}^2}\Delta s}{m} + v_{i-1}^2} \tag{4.1.6}$$

where v_{i-1} is the velocity when the distance s_{i-1} has been traveled on the path described by $\kappa(s)$.

The velocity profile can be constructed as described above along with equation (4.1.6). In the case of an accelerating profile ($u = +1$) equation (4.1.6) can be used as is, but in the case of a decelerating profile ($u = -1$) the subscripts i and $i-1$ are switched in order to reverse integrate.

4.2 Dynamic Model Velocity Profile

As mentioned previously, the methods in this section directly outline the methods in [30] used to generate a time optimal velocity profile for a half car model. This section begins by presenting and explaining the half car model and all associated equations. Then the method to generate an optimal velocity profile is presented and explained.

The half car model as used in this thesis is shown in Figure 4.4. The motion of a half-car model can be described by the following equations:

$$\begin{aligned} m\ddot{y} &= (f_{Fx} + f_{Rx})\sin\psi - (f_{Fy} + f_{Ry})\cos\psi \\ m\ddot{x} &= (f_{Fx} + f_{Rx})\cos\psi - (f_{Fy} + f_{Ry})\sin\psi \\ I_z &= f_{Fy}\ell_F - f_{Ry}\ell_r \end{aligned} \tag{4.2.1}$$

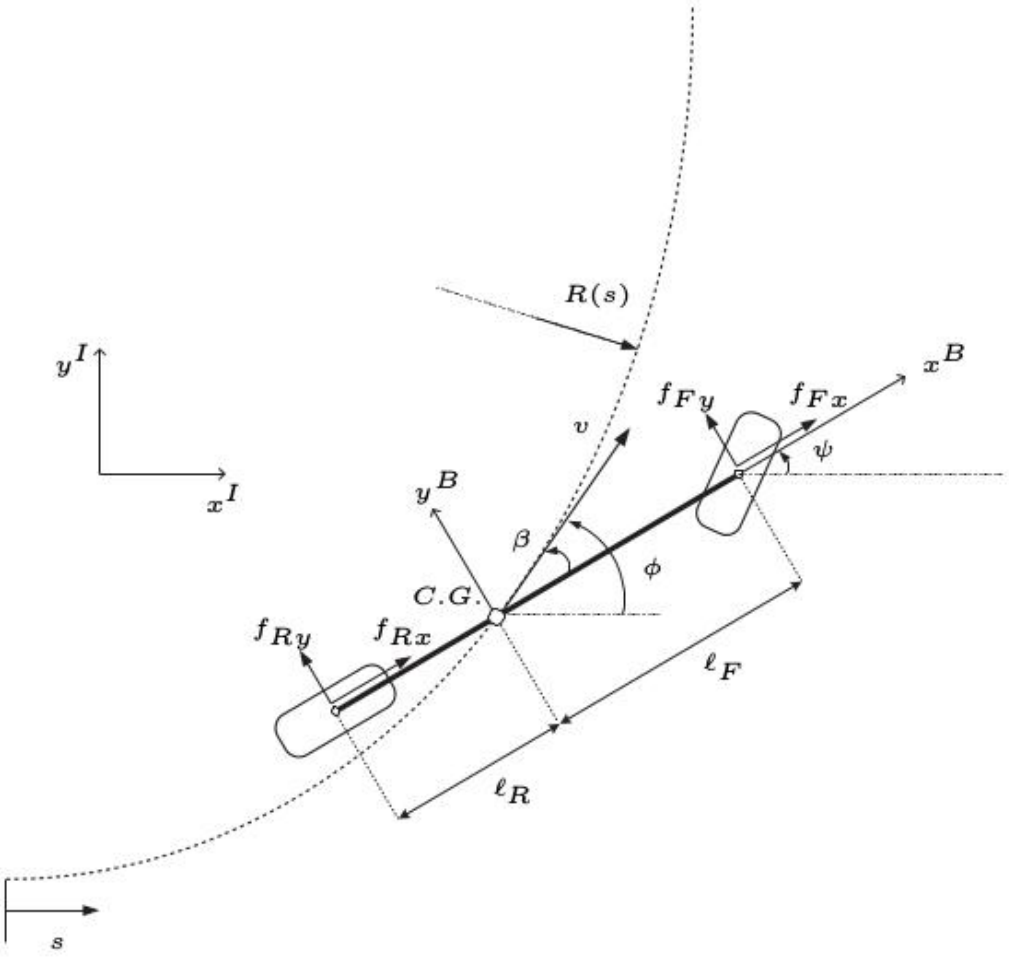

Figure 4.4: The half car model for a car traveling on a path described by radius, $R(s)$, or alternatively curvature, $\kappa(s)$, through the relationship $\kappa(s) = \frac{1}{R(s)}$. Diagram from [30] Figure 6.

where m is the vehicle mass, I_z is the polar moment of inertia of the vehicle, f_{ij} $(i = F, R,\ j = x, y)$ denotes the friction forces on the front and rear tires (F, R) along the longitudinal and lateral body axes (x, y), and ψ is the yaw angle of the vehicle. The coordinates of the center of gravity are denoted by (x, y).

The path angle (angle of the tangent to the path), ϕ, and the slip angle, β, are defined as:

$$\begin{aligned} \phi &= \arctan\left(\frac{\dot{x}}{\dot{y}}\right) \\ \beta &= \phi - \psi. \end{aligned} \tag{4.2.2}$$

The total friction forces for the front and rear tires are given by:

$$F_i = F_{iz} D \sin(C \arctan(B s_i)) \tag{4.2.3}$$

where $i = F, R$, F_{iz} is the vertical load at the front and rear axles, and $B,\ C,\ D$ are constants defined by the type of surface, tire, ect. Reference [19] gives the following equations to calculate F_{iz}:

$$\begin{aligned} F_{Fz} &= mg\frac{\ell_R}{\ell_F + \ell_R} \\ F_{Rz} &= mg\frac{l_F}{\ell_F + \ell_R} \end{aligned} \tag{4.2.4}$$

where ℓ_R and ℓ_F are the lengths from the center of gravity to the rear and

front axles, respectively. The total slip, s_i $(i = F, R)$, is given by:

$$s_i = \sqrt{s_{ix}^2 + s_{iy}^2} \tag{4.2.5}$$

where s_{ij} $(i = F, R,\ j = x, y)$ is the slip along the longitudinal and lateral axes of the ith wheel.

The lateral slip of the rear tire is:

$$s_{Ry} = \frac{v \sin\beta - \dot{\psi} l_R}{v \cos\beta}. \tag{4.2.6}$$

Pacejka's Magic Formula [2] uses equations (4.2.3), (4.2.5), and (4.2.6) and gives the rear tire friction force by:

$$f_{Rj} = -\frac{s_{Rj}}{s_R} F_R \tag{4.2.7}$$

where $j = x, y$.

The front and rear longitudinal slip are considered the control inputs and are chosen in the range $s_{ix} = [-1, +1]$ [2].

After establishing these equations and control inputs the next step is to construct front, rear, and resultant force diagrams that will be used in determining the optimal velocity profile. We will begin by constructing the rear force diagram (also called the rear force characteristic).

For a given state, $(v,\ \beta,\ \dot{\psi})$, it can be seen from Equation (4.2.6) that the rear lateral slip, s_{Ry}, is fixed. Then, for that same given state, the lateral and longitudinal rear tire forces, f_{Rj} $(j = x, y)$, can be plotted against rear longitudinal slip, s_{Rx}, using Equations (4.2.3), (4.2.5), (4.2.7), and the fixed

rear lateral slip, s_{Ry}. Figure 4.5 shows an example of these plots. The values of f_{Rx} and f_{Ry} can be plotted against each other to create the rear friction force diagram (rear force characteristic). An example of this diagram is shown in Figure 4.6.

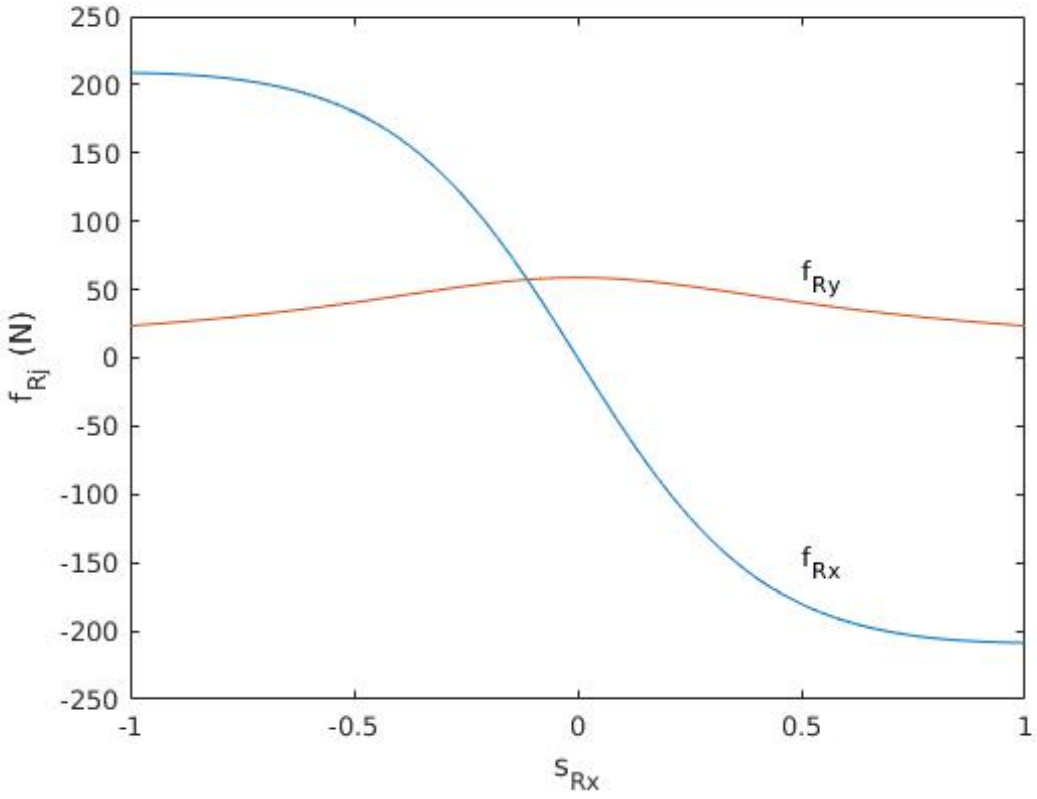

Figure 4.5: Plots of f_{Rj} $(j = x, y)$ against $s_{Rx} \in [-1, +1]$. Computed with the values of $F_{Fz} = 300\ N$, $D = .7$, $C = 1.5$, $B = 7$, $l_R = .4\ m$, $\beta = \frac{\pi}{32}\ rad$, $\dot{\psi} = .35\ \frac{rad}{s}$, $v = 1\ \frac{m}{s}$.

Next the front friction force diagram (front friction characteristic) is constructed. Equation (4.2.8) shows that the front tire lateral slip, s_{Fy}, is dependent on the steering angle, which is a control input, so the front friction characteristic cannot be constructed directly as with the rear friction charac-

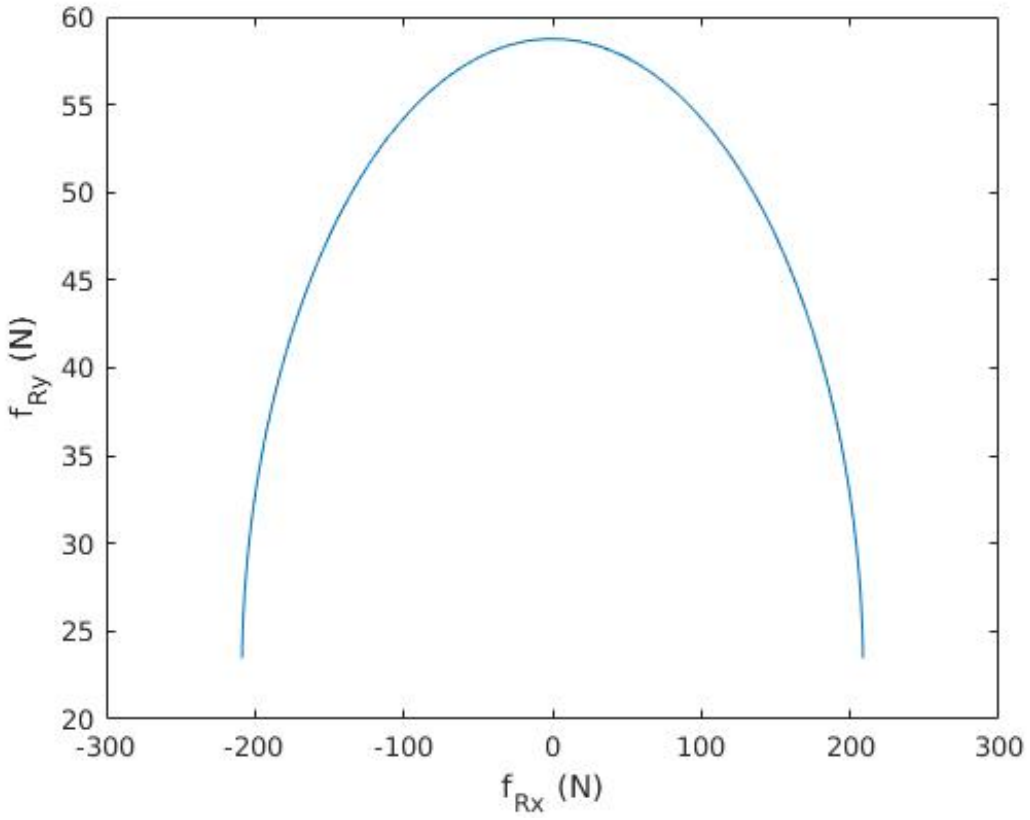

Figure 4.6: The rear friction characteristic from the vehicle state described in Figure 4.5.

teristic.

$$s_{Fy} = -\frac{v\sin(\beta-\delta) + \dot{\psi} l_F \cos\delta}{v\cos(\beta-\delta) + \dot{\psi} l_F \sin\delta} \tag{4.2.8}$$

where δ is the steering angle. The steering angle is defined as the angle between the longitudinal front tire axes and the longitudinal body axis.

It can be seen from Equation (4.2.8) that $s_{Fy} \in [-1, +1]$. Combining this with Equation (4.2.5), remembering that the front lateral slip is a control input chosen such that $s_{Fx} \in [-1, +1]$, it is clear that the total front slip is between zero and one i.e. $s_F \in [0, +1]$. Using Equation (4.2.3) the maximum combined front tire force, F_F^{max}, can be calculated on the interval $s_F \in [0, +1]$, this is demonstrated in Figure 4.7.

The total front tire force, F_F, must then lie within a circle of radius F_F^{max}. Such a friction circle is shown in Figure 4.8.

Reference [30] demonstrates that, assuming we can control the front lateral slip and steering angle, the front friction forces, f_{Fx} and f_{Fy}, can be chosen anyway such that the total front tire friction force, F_F, is within the friction circle. This friction circle is the front friction characteristic (front friction force diagram).

The front friction circle can be combined with the rear friction characteristic to create a resultant force diagram ('GG-diagram'). Note that the above mentioned diagrams are unique for each state of the vehicle. An example 'GG-diagram' is shown in Figure 4.9. The unique F_i $(i = F, R)$ values are where f_{tot} intersects with the front friction circle and rear wheel friction characteristic, respectively. f_{ij} $(i = F, R,\ j = x, y)$ can then be found by breaking

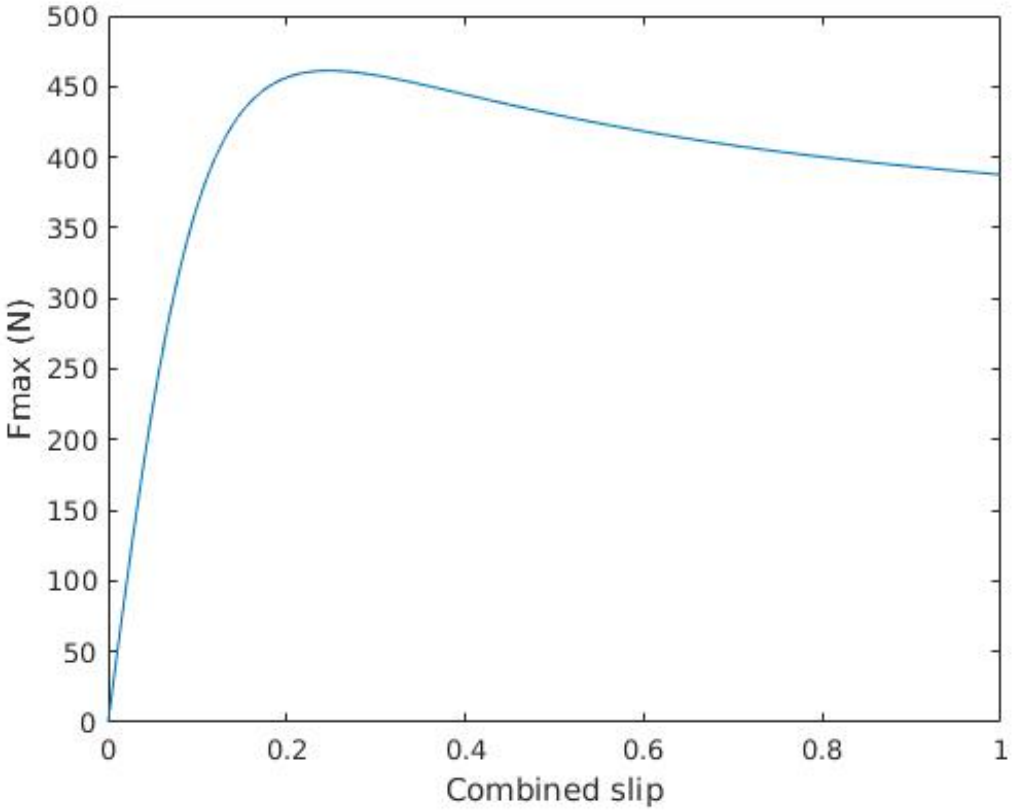

Figure 4.7: The front tire force curve calculated from (4.2.3) with the values of $F_{Rz} = 392.4\ N\ ,\ D = .7\ ,\ C = 1.5\ ,\ B = 7$. The maximum of this graph is F_F^{max}.

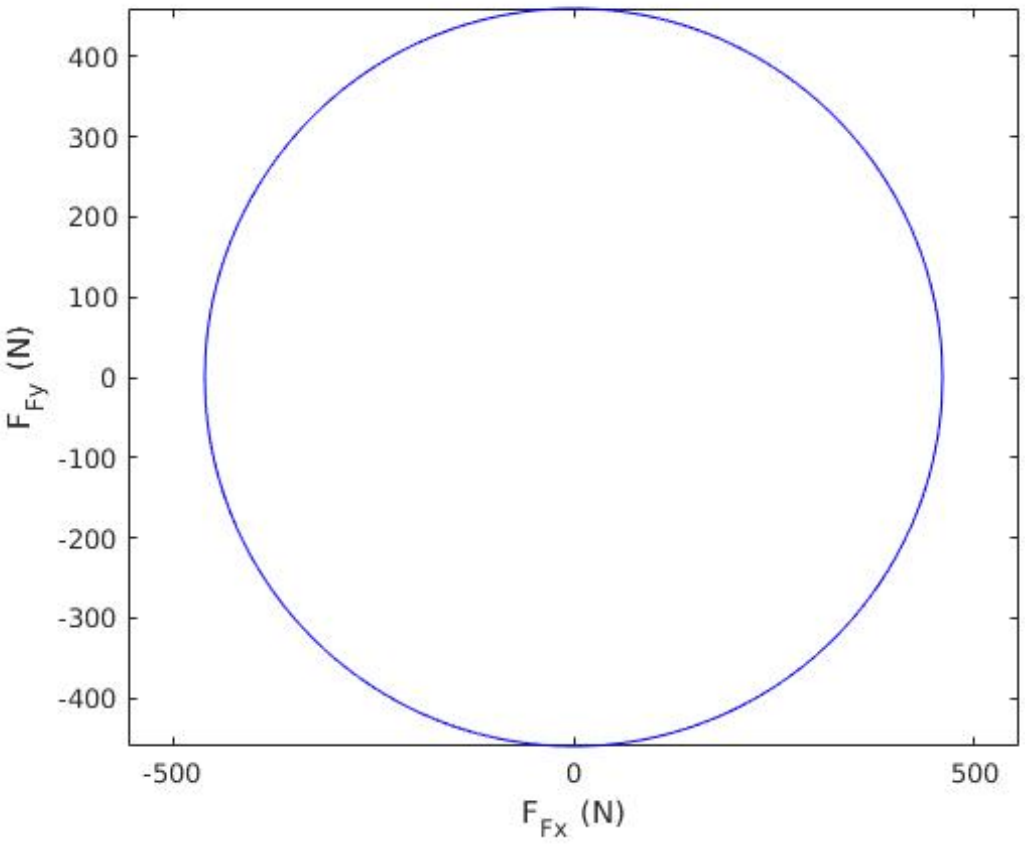

Figure 4.8: The friction circle corresponding to the max shown by Figure 4.7 .

F_i $(i = F, R)$ into its components along the x^β and y^β directions.

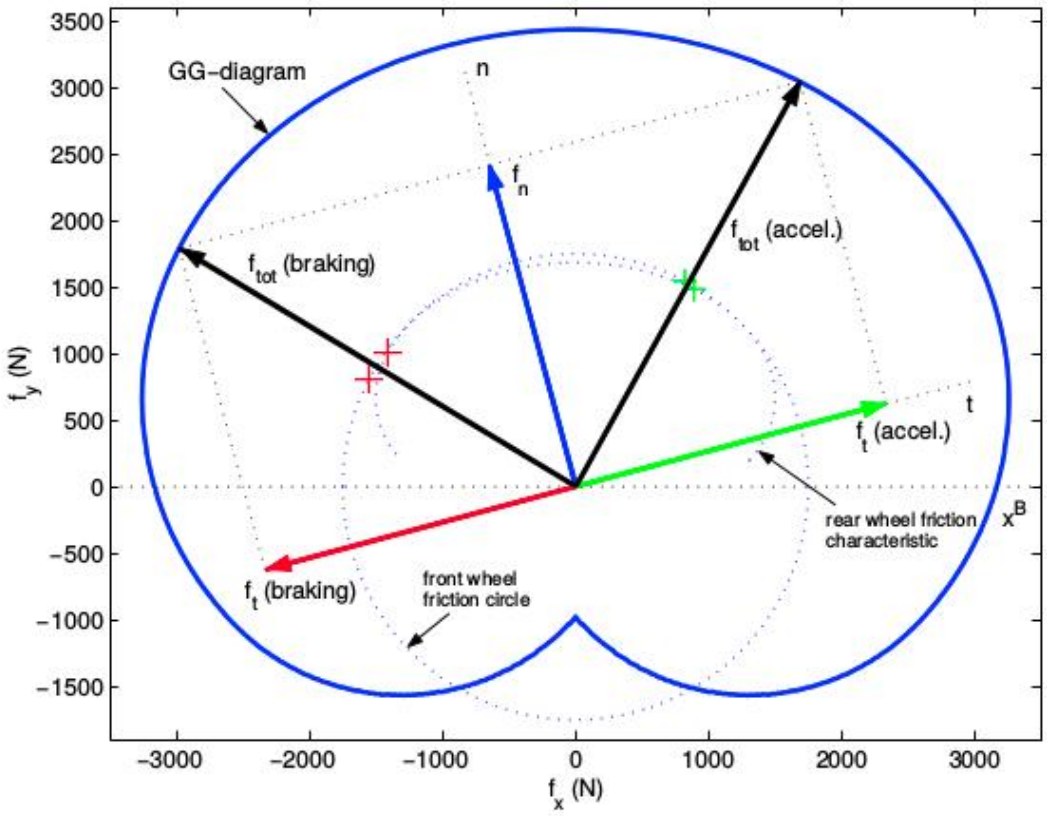

Figure 4.9: A sample 'GG-Diagram'. f_n, f_{tot}, and f_t refer to the normal, total, and tangential forces on the center of gravity of the car. The angle between x^β and t is β. This figure can be best understood when looking at the corresponding half car diagram, Figure 4.4. Diagram from [30] Figure 10.

.

It can be seen that for any given f_{tot} there are unique front and rear longitudinal and lateral tire forces. These forces are such that the total front tire friction force, F_F, and total rear tire friction force, F_R, are along the direction of f_{tot} and lie on the front wheel friction circle and rear wheel friction characteristic, respectively. f_{tot} is also associated with a unique normal and

tangential force, f_n and f_t.

Velocity profiles, for the dynamic case, are constructed as follows:

- First a 'GG-Diagram' for the initial state is constructed.
- Using this unique 'GG-Diagram' the initial values of f_{ij} $(i = F, R,\ j = x, y)$ are found or calculated as will be described.
- Equation (4.2.1) can then be directly integrated and gives, along with Equation (4.2.2), the state of the car $(v,\ \beta,\ \dot{\psi})$, at the next time step.
- This new state is then used to construct its unique 'GG-digram'.
- The required normal force, f_n, then is calculated using (4.1.1) and the tangential force, f_t, is computed such that the total force, f_{tot}, is on the edge of the 'GG-Diagram'.
- The new values of f_{ij} $(i = F, R,\ j = x, y)$ are calculated. From here Equation (4.2.1) can be integrated again to get the values for the next state of the car.
- This process is then iteratively repeated to construct a velocity profile.

This method is followed until $\beta \geq \beta_{critical}$. Reference [30] shows that incorporating this $\beta_{critical}$ increases stability in the yaw dynamics and suggests a value of 10 degrees for $\beta_{critical}$, which is used in this thesis. At this point the method changes so that $F_F + F_R = f_n$. The total rear friction force, F_R, is chosen to be along the normal direction. F_F is then calculated such that $F_F + F_R = f_n$ holds. This method is continued at each iteration until $\beta < \beta_{critical}$.

The strategy for calculating the optimal velocity profile is generally the same as the strategy in the previous section, with a few changes, and can be summarized as follows:

- Find the local maximum points of the absolute value of the path curvature.

- For each of the local maximum a velocity profile is constructed so that at the local maximum $v = v_{critical},\ \beta = 0$, and $\dot{\psi} = 0$. $v_{critical}$ is calculated using Equation 4.1.4. The initial values of f_{ij} $(i = F, R,\ j = x, y)$ are computed such that the total force of the 'GG-Diagram', f_{tot}, is along the normal direction i.e. $f_{tot} = f_n$. The velocity profile leading into the maximum is then constructed, as described above, with reverse integration of Equations (4.2.1) and negative tangential force, f_n. The profile leading away from the maximum is constructed with a positive tangential force, f_t, and forward integration of Equations (4.2.1). Figure 4.10 shows an example of these profiles.

- The profile from the given initial condition is constructed with a positive f_t. For the final condition a velocity profile is constructed with a negative f_t, ending at the final condition. Figure 4.10 also shows the profiles from the initial condition and leading into the final condition.

- The optimal velocity profile is then the minimum of all of the above profiles at each point along the trajectory. The optimal velocity profile for the trajectory described by Figure 4.1 is shown in Figure 4.11.

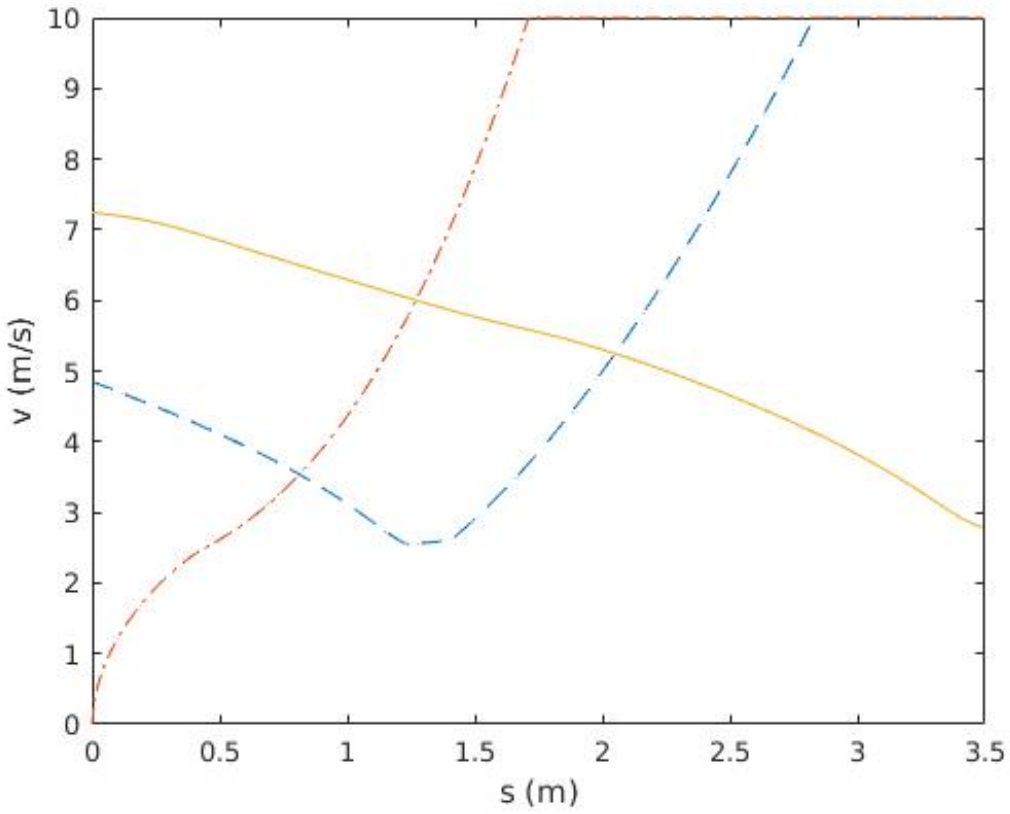

Figure 4.10: Constructed velocity profiles with the dynamic model from the path described by Figure 4.1. The solid line is the profile constructed from the final condition, the dashed line is constructed from the curvature maximum, and the dashed and dotted line is constructed from the initial condition.

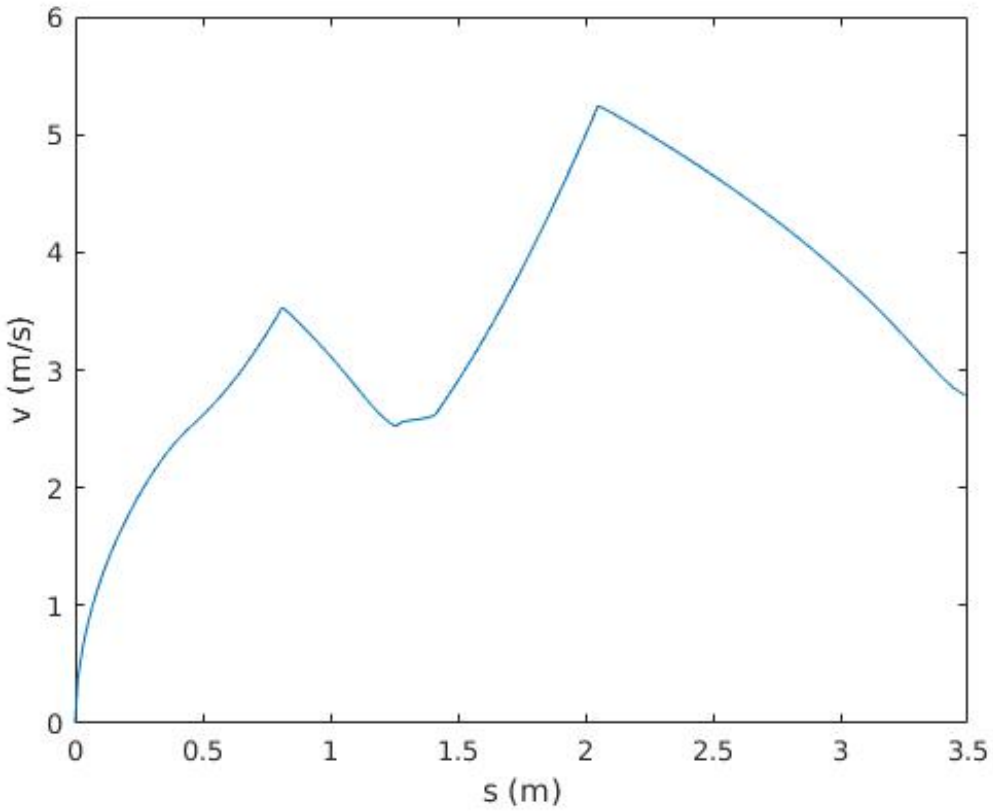

Figure 4.11: The time optimal velocity profile constructed as the minimum of the velocity profiles in Figure 4.10

As this section was an outline of the methods from references [31] and [30] used in this thesis the reader is referred to these references for more detail.

Chapter 5

Results

5.1 Optimized Paths Through a Track

This section contains the results for optimized paths through tracks. Algorithm 2 was tested on several different tracks. In every case the waypoint update method was to add one to the number of waypoints.

First, Figures 5.1 and 5.2 demonstrate the behavior of Algorithm 2 as the number of waypoints increase. Figure 5.1 shows the paths with 5 and 75 waypoints. Figure 5.2 shows the cost function values of path calculated with increasing waypoints. This demonstrates that increasing the number of waypoints around the track causes the path to closely follow the center line and thus the cost function approaches the value equal to cost function evaluation of the track center line. When running Algorithm 2 the first path found that is within the track boundaries may not always be the most optimal path. Therefore running the algorithm until the cost function begins to increase will help ensure that the optimal path is found. It is important to note here that

the solution is optimal for the waypoint update and initial placement methods used. As will be mentioned in Chapter 6, new methods of waypoint placement and updating are being explored. These new methods could lead to solutions with lower cost function values.

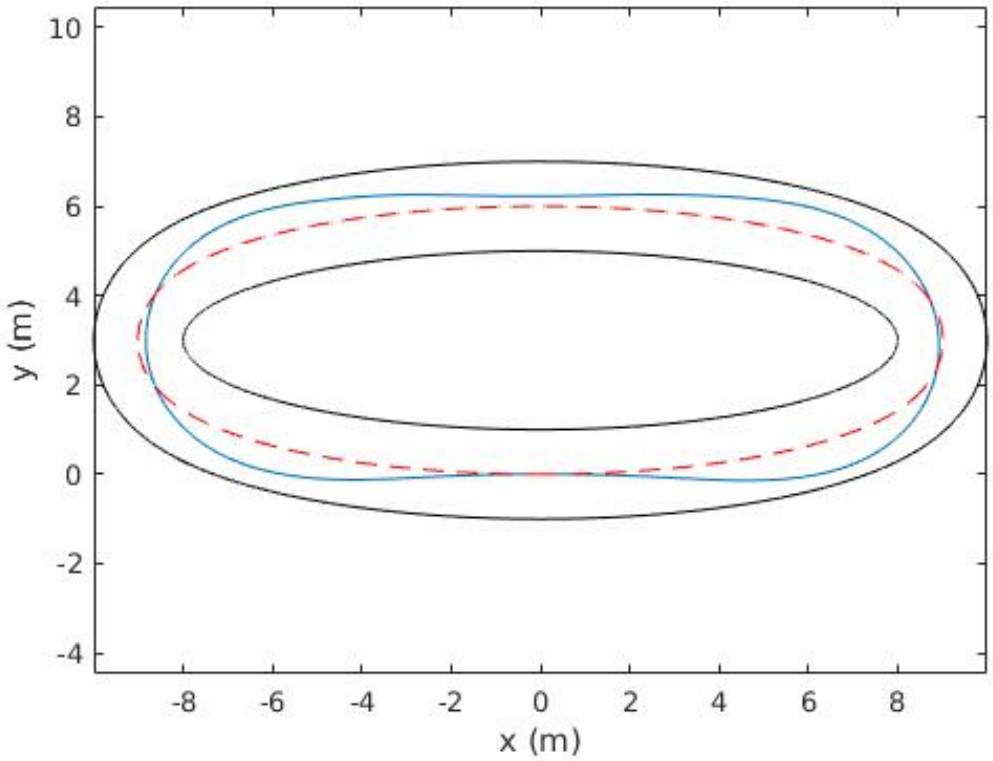

Figure 5.1: Paths through the shown oval track starting and ending at $(0, 0)$. The solid path is calculated with 5 optimized waypoints and the dashed path is calculated with 75 optimized waypoints.

Results for the optimized closed-loop path generation through tracks of varying complexity are shown in Figures 3.9, 5.3, 5.4, and 5.5.

In each case the first path found that fit within the track was the most optimal path. That is, using the minimum number of waypoints led to the

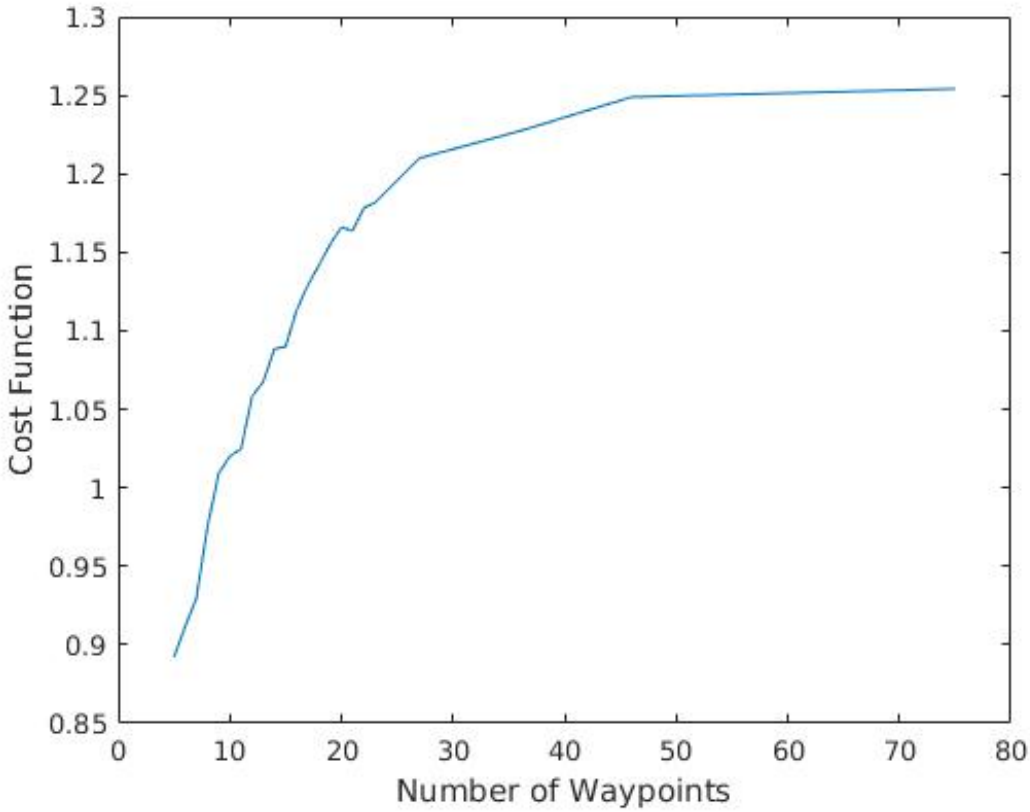

Figure 5.2: The cost function value of the optimized path through the track in Figure 5.1 for a given number of waypoints.

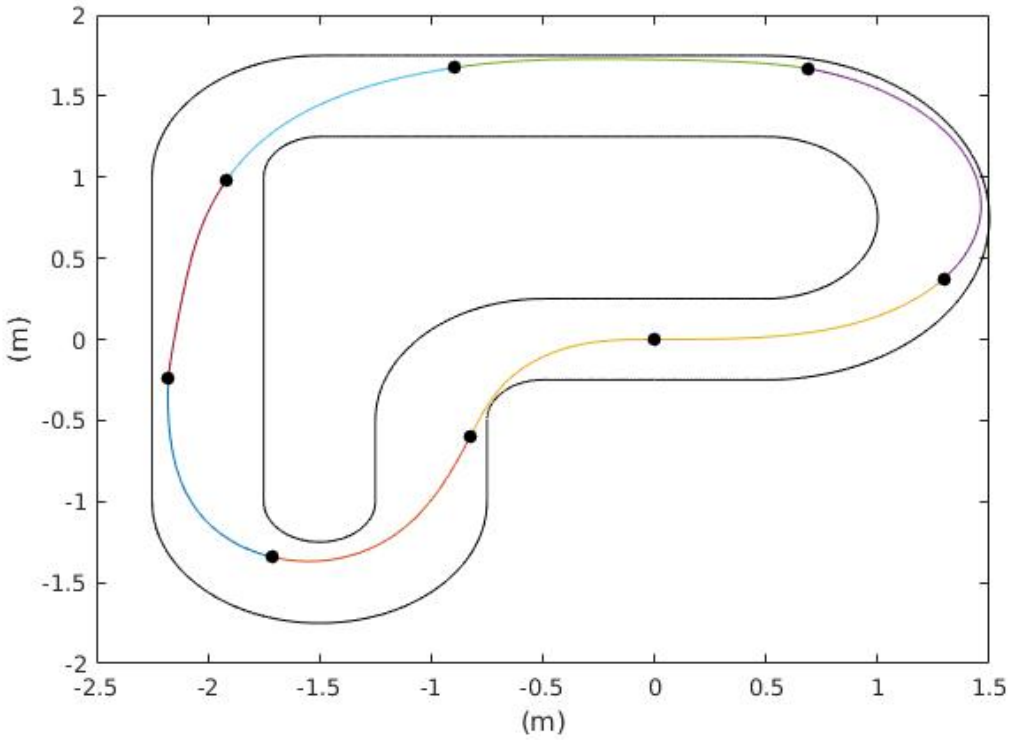

Figure 5.3: Path through track 2. Start and end point is $(0, 0)$. Number of waypoints: 7, marked by the black dots.

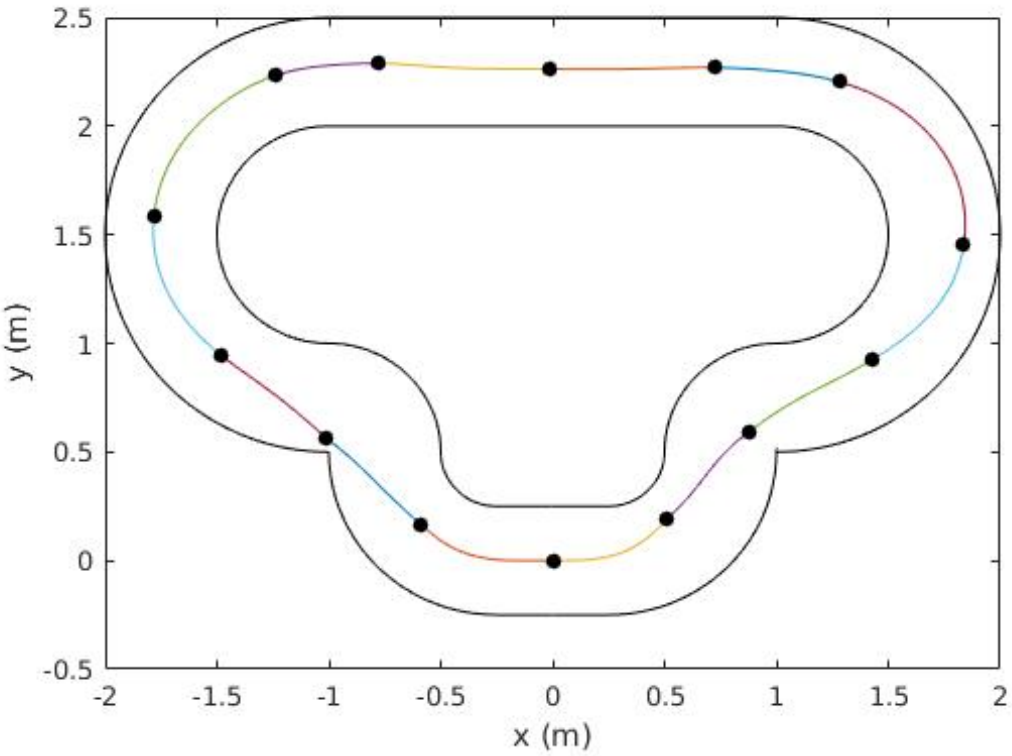

Figure 5.4: Path through track 3. Start and end point is $(0, 0)$. Number of waypoints: 13, marked by the black dots.

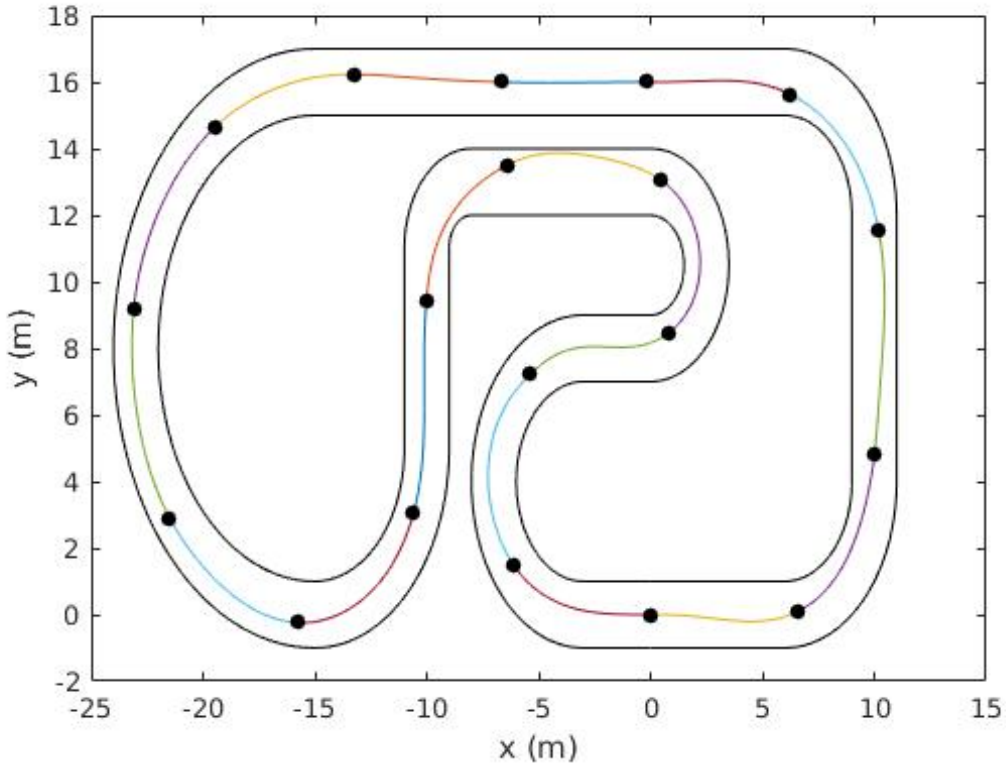

Figure 5.5: Path through track 4. Start and end point is $(0, 0)$. Number of waypoints: 18, marked by the black dots.

most optimal path through the track. This is expected but, as mentioned above, may not always be the case. It is also noted that scaling down the track size leads to a much more rapid solution. Figures 3.9, 5.3, and 5.4 are on a much smaller scale than Figure 5.5 and Algorithm 2 was able solve for a solution much faster.

5.2 Velocity Profiles on Tested Tracks for Point Mass

The results of creating a velocity profile, for a point mass of 1 kg with maximum normal and tangential forces of 1 N, on each of the four tracks in Figures 3.9, 5.3, 5.4 and 5.5 are shown here.

The average velocity and total travel time for each of the above Figures are summarized in Table 5.1.

Table 5.1: Comparison of Average Velocities and Total Travel Times.

Corresponding Figure	Average Velocity (m/s)	Total Travel Time (s)
5.6	0.892	5.923
5.7	1.039	6.451
5.8	0.987	5.469
5.9	2.680	20.967

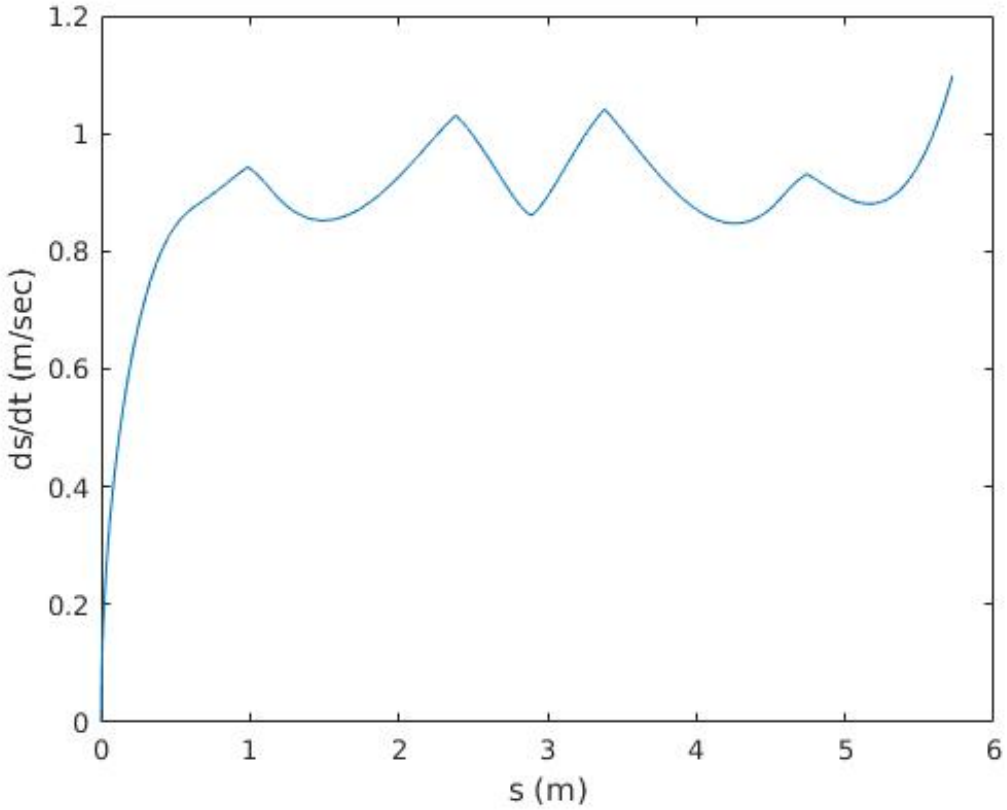

Figure 5.6: Velocity vs. distance traveled for the trajectory shown in Figure 3.9.

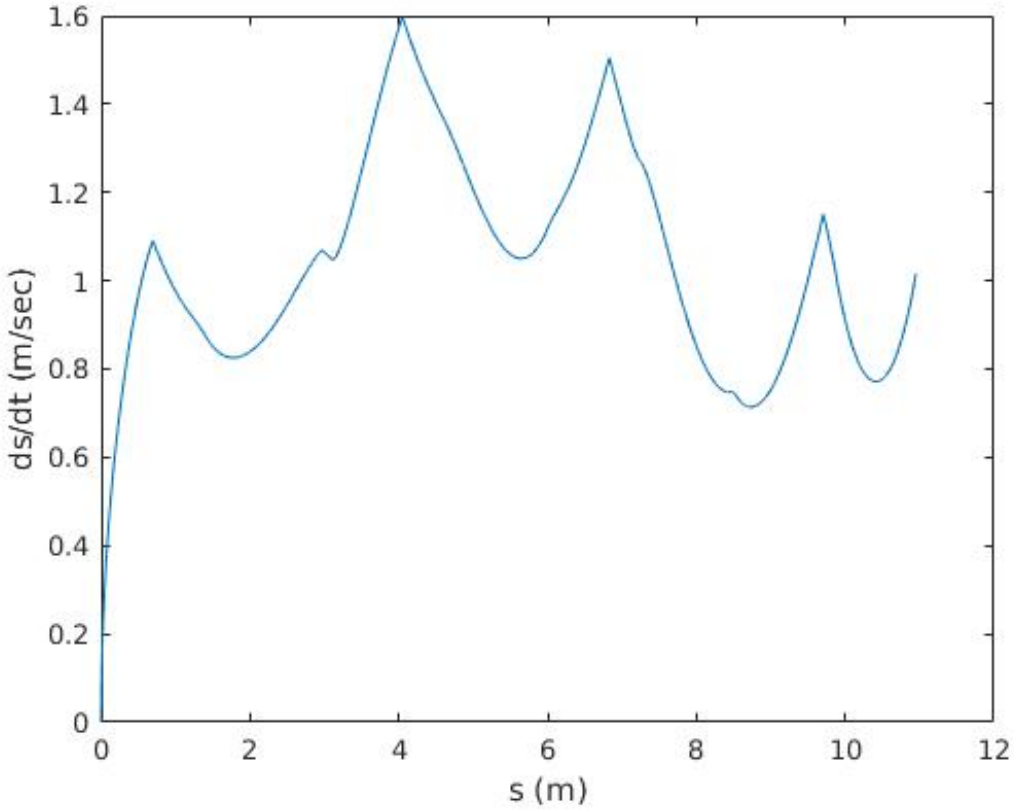

Figure 5.7: Velocity vs. distance traveled for the trajectory shown in Figure 5.3.

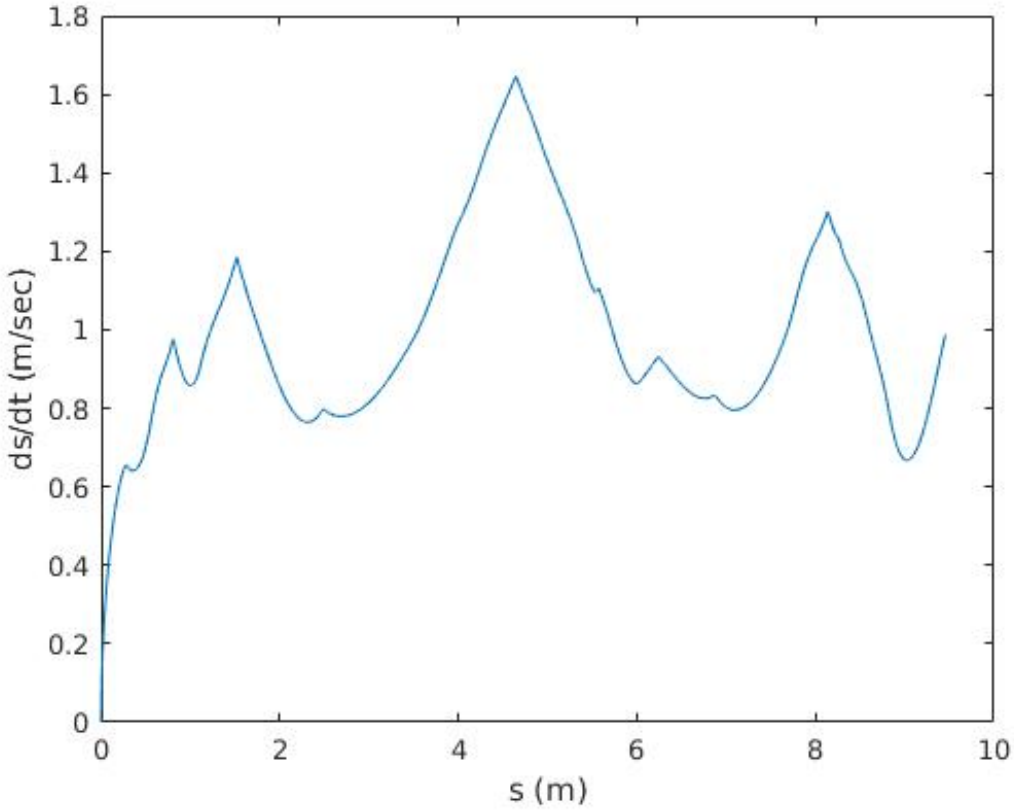

Figure 5.8: Velocity vs. distance traveled for the trajectory shown in Figure 5.4.

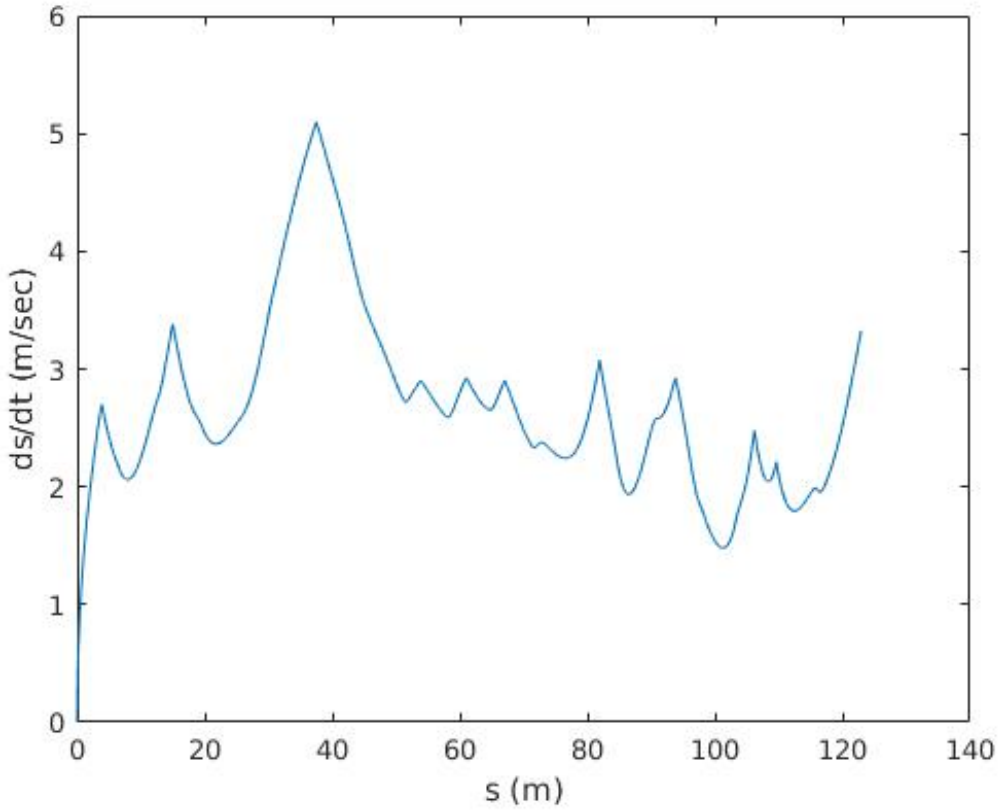

Figure 5.9: Velocity vs. distance traveled for the trajectory shown in Figure 5.5.

5.3 Velocity Profiles on Tested Tracks for Dynamic Model

The results of creating a velocity profile based on a dynamic model with the properties summarized in Table 5.2 are shown in this section. Note that due to the tested tracks being scaled in size the properties of the dynamic model were also scaled.

Table 5.2: Properties of the Dynamic Half-Car Model.

B	7
C	1.5
D	.7
ℓ_f (m)	.0065
ℓ_r (m)	.0055
I_z (kgm^2)	10
m (kg)	6
Max Velocity (m/s)	15

The average velocity and total travel time for each of the above Figures are summarized in Table 5.3.

Table 5.3: Comparison of Average Velocities and Total Travel Times.

Corresponding Figure	Average Velocity (m/s)	Total Travel Time (s)
5.10	2.946	1.899
5.11	3.271	3.349
5.12	3.088	3.063
5.13	8.643	14.224

Figures 5.10, 5.11, 5.12, and 5.13 show examples of velocity profiles that can be constructed to complete the creation of a trajectory given an optimal

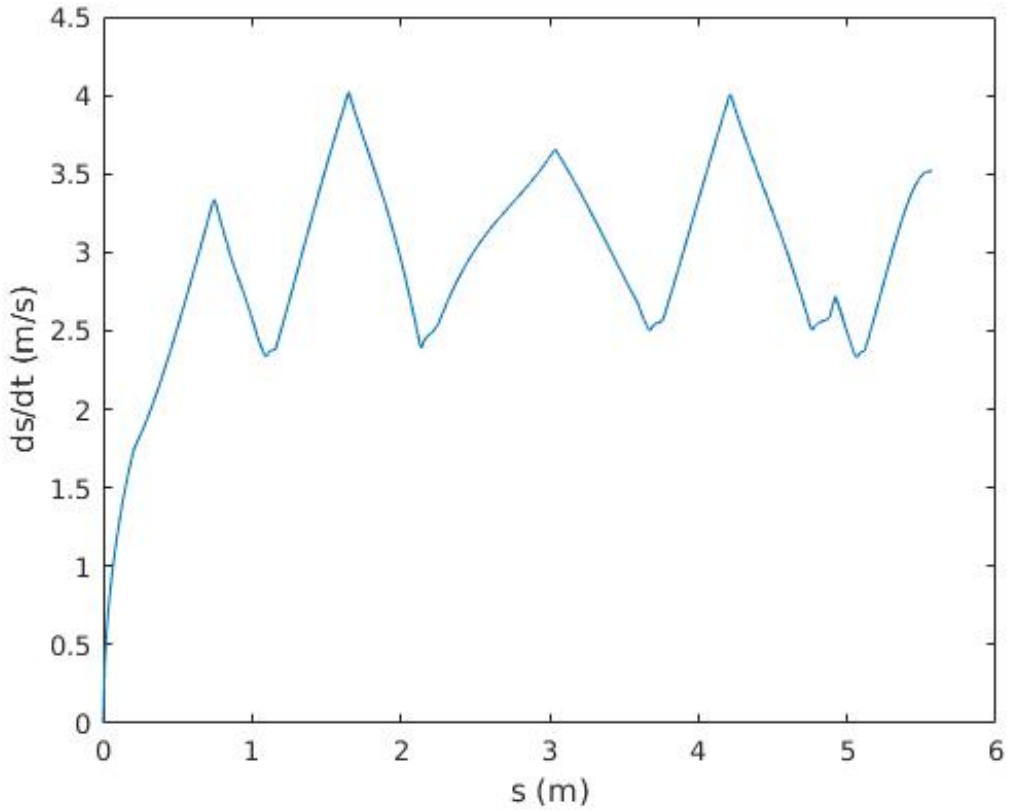

Figure 5.10: Velocity vs. distance traveled for the path shown in Figure 3.9 for the dynamic model.

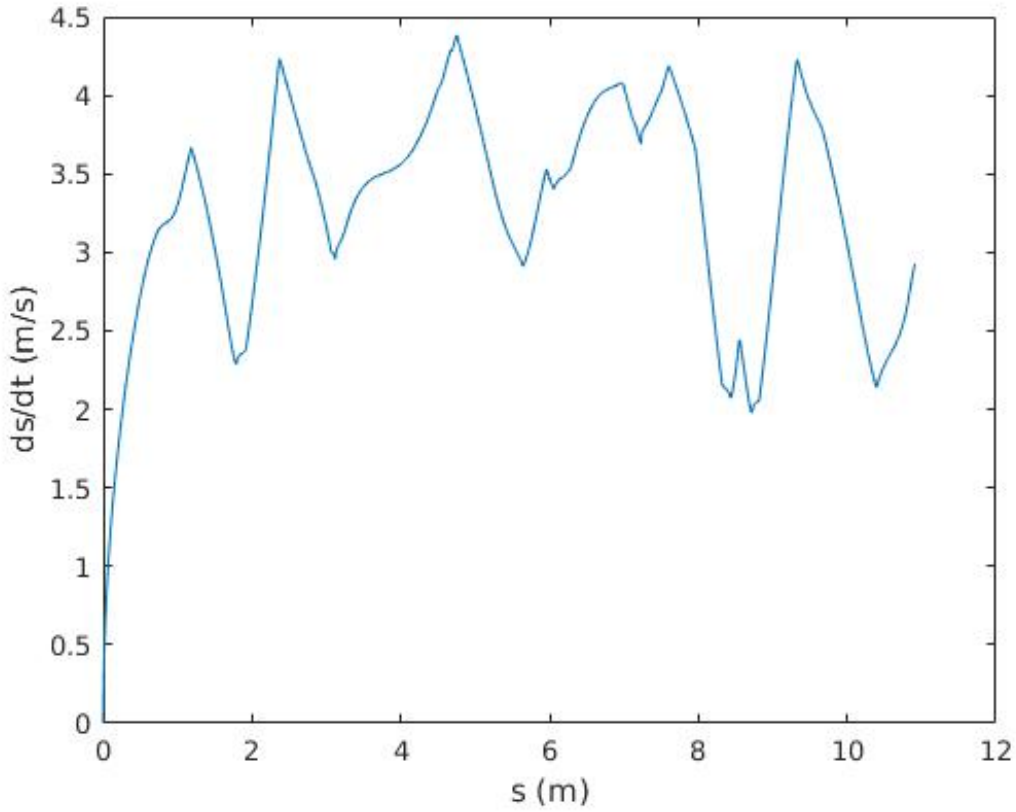

Figure 5.11: Velocity vs. distance traveled for the path shown in Figure 5.3 for the dynamic model.

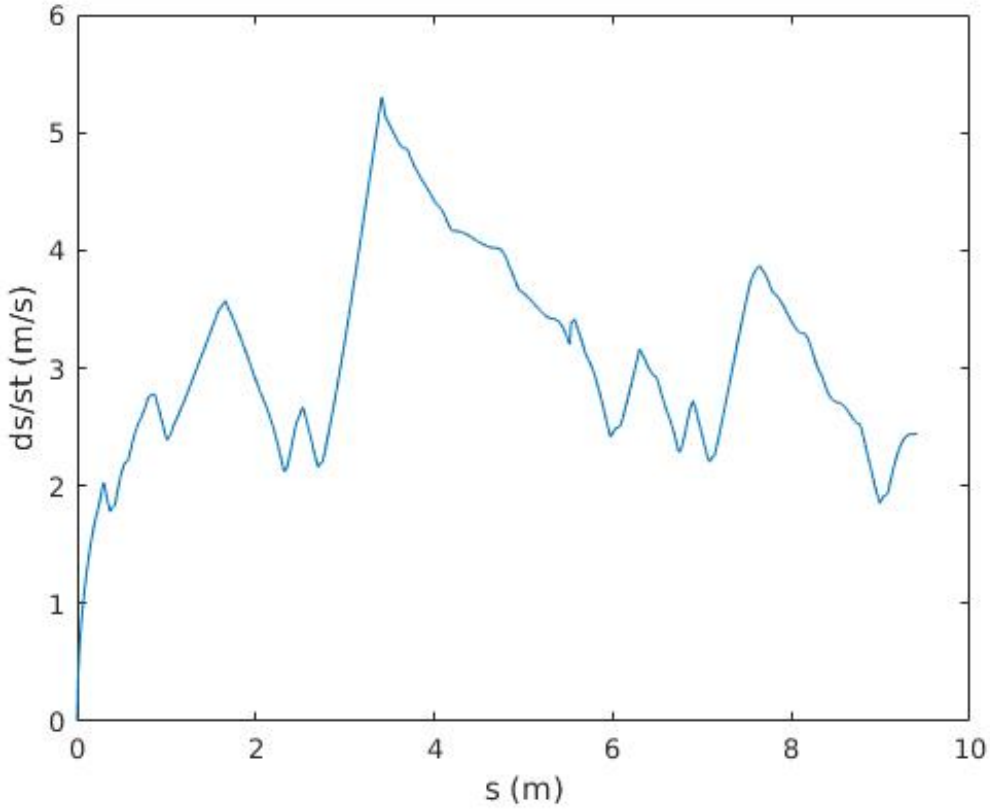

Figure 5.12: Velocity vs. distance traveled for the path shown in Figure 5.4 for the dynamic model.

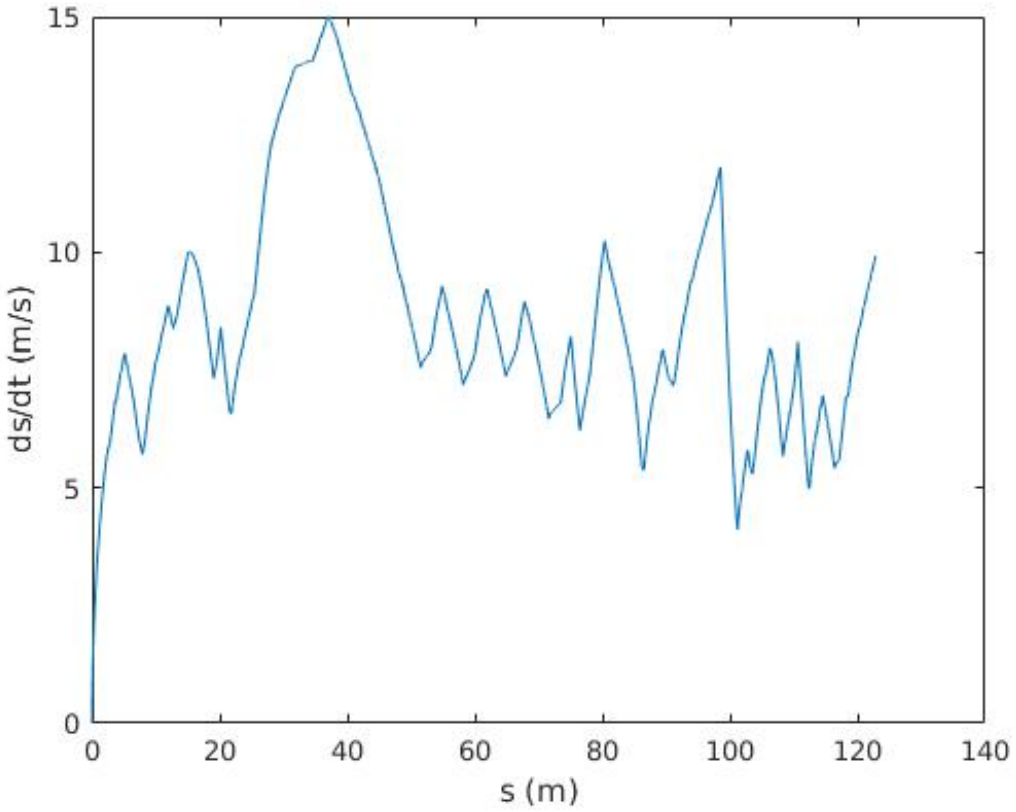

Figure 5.13: Velocity vs. distance traveled for the path shown in Figure 5.5 for the dynamic model.

path calculated as outlined in this thesis. In order to evaluate the cost function used in this paper, the average normal forces were compared for a car traveling on the center line of the track (using the same car properties as described in Table 5.2) and on the optimized path, with the average velocities computed from the optimized dynamic velocity profile. These results are summarized in Table 5.4. Table 5.4 shows that in each case the optimized path has a much smaller average normal force. This would lead to a much more comfortable ride as passengers would experience less forces acting on them throughout the ride.

Table 5.4: Comparison of Average Normal Force Traveling through the Track Center Line and Optimized Path.

Corresponding Track Figure	Average Center Line Normal Force (N)	Average Optimized Path Normal Force (N)
3.9	17.076	0.885
5.3	12.853	0.708
5.4	14.421	0.618
5.5	11.316	0.699

Chapter 6

Conclusion

In this thesis a method of calculating optimized paths through a track has been presented.

First, a constrained optimization problem was formulated to calculate an optimized path between arbitrary initial and terminal states. Next, the constrained optimization problem was extended to incorporate a waypoint optimization problem so that the path from an initial state, through the waypoint, to a final state is optimal in some sense. Finally, the algorithms to calculate the optimal closed-loop path of a car through a track were presented. These algorithms were tested on multiple tracks and the results were shown. These paths were also used to construct velocity profiles in order to calculate complete trajectories.

Further investigation into the efficient placement and updating of waypoints could lead to more optimal and efficient paths. A method of using the track's curvature to choose the placement of the waypoints is currently under investigation.

Bibliography

[1] N Achour, A Lakhdari, and F Ferguene. Motion planning for car-like robots using hierarchical genetic algorithms. In *Proceedings of the World Congress on Engineering*, volume 2, 2013.

[2] Pacejka H Bakker E, Nyborg L. Tyre modelling for use in vehicle dynamics studies. *SAE*, (870421), 1987.

[3] Matteo Botta, Vincenzo Gautieri, Daniele Loiacono, and Pier Luca Lanzi. Evolving the optimal racing line in a high-end racing game. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 108–115. IEEE, 2012.

[4] MJ Box. A new method of constrained optimization and a comparison with other methods. *The Computer Journal*, 8(1):42–52, 1965.

[5] Richard H Byrd, Peihuang Lu, Jorge Nocedal, and Ciyou Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[6] Luca Caracciolo, Alessandro De Luca, and Stefano Iannitti. Trajectory tracking control of a four-wheel differentially driven mobile robot. In

Robotics and Automation, 1999. Proceedings. 1999 IEEE International Conference on, volume 4, pages 2632–2638. IEEE, 1999.

[7] Luigi Cardamone, Daniele Loiacono, Pier Luca Lanzi, and Alessandro Pietro Bardelli. Searching for the optimal racing line using genetic algorithms. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 388–394. IEEE, 2010.

[8] D Costantinescu and EA Croft. Smooth and time-optimal trajectory planning for industrial manipulators along specified paths. *Journal of robotic systems*, 17(5):233–249, 2000.

[9] Jaydev P Desai, James P Ostrowski, and Vijay Kumar. Modeling and control of formations of nonholonomic mobile robots. *IEEE transactions on Robotics and Automation*, 17(6):905–908, 2001.

[10] Philip E Gill, Walter Murray, and Michael A Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM review*, 47(1):99–131, 2005.

[11] Duc Thang Ho and Jonathan M Garibaldi. A fuzzy approach for the 2007 cig simulated car racing competition. In *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On*, pages 127–134. IEEE, 2008.

[12] Dervis Karaboga and Bahriye Basturk. Artificial bee colony (abc) optimization algorithm for solving constrained optimization problems. In *International Fuzzy Systems Association World Congress*, pages 789–798. Springer, 2007.

[13] Alonzo Kelly and Bryan Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *The International Journal of Robotics Research*, 22(7-8):583–601, 2003.

[14] Harold W Kuhn. Nonlinear programming: a historical view. In *Traces and Emergence of Nonlinear Programming*, pages 393–414. Springer, 2014.

[15] J-P Laumond, Paul E Jacobs, Michel Taix, and Richard M Murray. A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593, 1994.

[16] Jean-Paul Laumond. Finding collision-free smooth trajectories for a nonholonomic mobile robot. In *IJCAI*, volume 10, pages 1120–1123, 1987.

[17] Richard M Murray and Sosale Shankara Sastry. Nonholonomic motion planning: Steering using sinusoids. *IEEE Transactions on Automatic Control*, 38(5):700–716, 1993.

[18] Umashankar Nagarajan, George Kantor, and Ralph L Hollis. Trajectory planning and control of an underactuated dynamically stable single spherical wheeled mobile robot. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3743–3748. IEEE, 2009.

[19] Shayegan Omidshafiei. Optimal racing line control. 2014.

[20] Zhihua Qu, Jing Wang, and Clinton E Plaisted. A new analytical solution to mobile robot trajectory generation in the presence of moving obstacles. *IEEE transactions on robotics*, 20(6):978–993, 2004.

[21] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, May 1993.

[22] Tizar Rizano, Daniele Fontanelli, Luigi Palopoli, Lucia Pallottino, and Paolo Salaris. Global path planning for competitive robotic cars. In *Decision and Control (CDC), 2013 IEEE 52nd Annual Conference on*, pages 4510–4516. IEEE, 2013.

[23] R Tyrrell Rockafellar. Extended nonlinear programming. In *Nonlinear optimization and related topics*, pages 381–399. Springer, 2000.

[24] Alexis Scheuer and Th Fraichard. Continuous-curvature path planning for car-like vehicles. In *Intelligent Robots and Systems, 1997. IROS'97., Proceedings of the 1997 IEEE/RSJ International Conference on*, volume 2, pages 997–1003. IEEE, 1997.

[25] Klaus Schittkowski. Nlpql: A fortran subroutine solving constrained nonlinear programming problems. *Annals of operations research*, 5(1):485–500, 1986.

[26] Kazuo Sugihara and John Smith. Genetic algorithms for adaptive motion planning of an autonomous mobile robot. In *Computational Intelligence in Robotics and Automation, 1997. CIRA'97., Proceedings., 1997 IEEE International Symposium on*, pages 138–143. IEEE, 1997.

[27] Petr Svestka and Mark H Overmars. Coordinated motion planning for multiple car-like robots using probabilistic roadmaps. In *Robotics and*

Automation, 1995. Proceedings., 1995 IEEE International Conference on, volume 2, pages 1631–1636. IEEE, 1995.

[28] Jur Van Den Berg, Pieter Abbeel, and Ken Goldberg. Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information. *The International Journal of Robotics Research*, 30(7):895–913, 2011.

[29] S Van Koutrik. Optimal control for race car minimum time maneuvering. Master's thesis, Delft University of Technology, 2015.

[30] E Velenis and P Tsiotras. Optimal velocity profile generation for given acceleration limits; the half-car model case. In *EEE International Symposium on Industrial Electronics*, pages 355–360, 2005.

[31] Efstathios Velenis and Panagiotis Tsiotras. Optimal velocity profile generation for given acceleration limits: Theoretical analysis. In *American Control Conference, Proceedings of the 2005*, pages 1478–1483. IEEE, 2005.

Appendix A

Derivation of Derivatives from Section 2.3

The derivatives in this section are calculated using the methods presented in [13], with the addition of an arbitrary starting position. In (2.1.7) the constraint equation was defined as:

$$\mathbf{g}(\mathbf{q}) = \mathbf{x}(\mathbf{q}) - \mathbf{x}_f, \tag{A.0.1}$$

where $\mathbf{q} = [\mathbf{p}^T \ s_f]^T$, $\mathbf{p} = [b \ c \ d \ \cdots \ ]^T$, $\mathbf{x}(\mathbf{q}) = [x(\mathbf{q}) \ y(\mathbf{q}) \ \theta(\mathbf{q}) \ \kappa(\mathbf{q})]^T$, and $\mathbf{x}_f = [x_f \ y_f \ \theta_f \ \kappa_f]^T$. The state equations are:

$$\begin{aligned} x(\mathbf{q}) &= x_0 + \int_0^{s_f} \cos\theta(\mathbf{p}, s)ds \\ y(\mathbf{q}) &= y_0 + \int_0^{s} \sin\theta(\mathbf{p}, s)ds \\ \theta(\mathbf{q}) &= \theta_0 + \kappa_0 s_f + \frac{bs_f^2}{2} + \frac{cs_f^3}{3} + \frac{ds_f^4}{4} + \cdots \\ \kappa(\mathbf{q}) &= \kappa_0 + bs_f + cs_f^2 + ds_f^3 + \cdots \end{aligned} \tag{A.0.2}$$

where $\mathbf{q} = [b\ c\ d\ \cdots\ s_f]^T$. Note that the parameter a is not considered in $\mathbf{p}$ since $\kappa(\mathbf{p}, s)|_{s=0} = a = \kappa_0$. The derivatives of $\theta(\mathbf{q})$ and $\kappa(\mathbf{q})$ with respect to $\mathbf{q}$ are readily apparent:

$$\begin{aligned} \frac{\partial \theta}{\partial \mathbf{q}} &= \left[\frac{s_f^2}{2}\ \frac{s_f^3}{3}\ \frac{s_f^4}{4}\ \cdots\ \kappa(\mathbf{q})\right] \\ \frac{\partial \kappa}{\partial \mathbf{q}} &= \left[s_f\ s_f^2\ s_f^3\ \cdots\ \kappa'(\mathbf{q})\right] \end{aligned} \tag{A.0.3}$$

where $\kappa'(\mathbf{q}) = b + 2cs_f + 3ds_f^2 + \cdots$.

The calculations of the derivatives of $x(\mathbf{q})$ and $y(\mathbf{q})$ are computed using Leibniz's Rule, which is as follows:

$$\begin{aligned} \frac{\partial}{\partial x}\left(\int_{a(x)}^{b(x)} f(x,t)dt\right) = \\ f(x,b(x))\frac{\partial}{\partial x}b(x) - f(x,a(x))\frac{\partial}{\partial x}a(x) + \\ \int_{a(x)}^{b(x)} \frac{\partial}{\partial x} f(x,t)dt. \end{aligned} \tag{A.0.4}$$

For simplicity of notations, it is useful to define the following equations:

$$\begin{aligned} f_y(\mathbf{p}) &= \sin\theta(\mathbf{p}, s) \\ f_x(\mathbf{p}) &= \cos\theta(\mathbf{p}, s). \end{aligned} \tag{A.0.5}$$

$x(\mathbf{q})$ and $y(\mathbf{q})$ can now be written as:

$$\begin{aligned} x(\mathbf{q}) &= x_0 + \int_0^{s_f} f_x(\mathbf{p}, s)ds \\ y(\mathbf{q}) &= y_0 + \int_0^{s_f} f_y(\mathbf{p}, s)ds. \end{aligned} \tag{A.0.6}$$

Now a direct application of Leibnitz's Rule can be used to solve for $\frac{\partial x}{\partial \mathbf{q}}$ and $\frac{\partial y}{\partial \mathbf{q}}$ as follows:

$$\begin{aligned}
\frac{\partial z}{\partial b} &= -\frac{1}{2}\int_0^{s_f} s^2 f_i(\mathbf{p}, s)ds \\
\frac{\partial z}{\partial c} &= -\frac{1}{3}\int_0^{s_f} s^3 f_i(\mathbf{p}, s)ds \\
\frac{\partial z}{\partial d} &= -\frac{1}{4}\int_0^{s_f} s^4 f_i(\mathbf{p}, s)ds \\
&\vdots \\
\frac{\partial z}{\partial s_f} &= f_j(\mathbf{p}, s_f)
\end{aligned} \tag{A.0.7}$$

where $f_i(\cdot) = f_y(\cdot), f_j(\cdot) = f_x(\cdot)$ for $z = x$ and $f_i(\cdot) = f_x(\cdot), f_j(\cdot) = f_y(\cdot)$ for $z = y$.

Appendix B

Derivation of Derivatives from Section 2.4

The state equations defined in Section 2.4 are defined as:

$$\begin{aligned}
x_1(\mathbf{q}) &= x_0 + \int_0^{s_{f1}} \cos\theta_1(\mathbf{p}_1, s)ds \\
y_1(\mathbf{q}) &= y_0 + \int_0^{s_{f1}} \sin\theta_1(\mathbf{p}_1, s)ds \\
\theta_1(\mathbf{q}) &= \theta_0 + \kappa_0 s_{f1} + \frac{b_1 s_{f1}^2}{2} + \frac{c_1 s_{f1}^3}{3} + \frac{d_1 s_{f1}^4}{4} + \cdots \\
\kappa_1(\mathbf{q}) &= k_0 + b_1 s_{f1} + c_1 s_{f1}^2 + d_1 s_{f1}^3 + \cdots \\
x_2(\mathbf{q}) &= x_w + \int_0^{s_{f2}} \cos\theta_2(\mathbf{p}_2, s)ds \\
y_2(\mathbf{q}) &= y_w + \int_0^{s_{f2}} \sin\theta_2(\mathbf{p}_2, s)ds \\
\theta_2(\mathbf{q}) &= \theta_w + \kappa_w s_{f2} + \frac{b_2 s_{f2}^2}{2} + \frac{c_2 s_{f2}^3}{3} + \frac{d_2 s_{f2}^4}{4} + \cdots \\
\kappa_2(\mathbf{q}) &= k_w + b_2 s_{f2} + c_2 s_{f2}^2 + d_2 s_{f2}^3 + \cdots
\end{aligned} \tag{B.0.1}$$

where $\mathbf{q} = [\mathbf{q}_1\ \mathbf{q}_2\ \mathbf{q}_3]^T$, $\mathbf{q}_i = [\mathbf{p}_i^T\ s_{fi}]$ and $\mathbf{p}_i = [b_i\ c_i\ d_i \cdots\ ]^T$ for $i = 1, 2$.

Equation (2.4.5) gives the constraint:

$$\mathbf{g}(\mathbf{q}) = \begin{bmatrix} \mathbf{g}_1(\mathbf{q}) \\ \mathbf{g}_2(\mathbf{q}) \end{bmatrix} \tag{B.0.2}$$

where $\mathbf{g}_1(\mathbf{q}) = [g_{1x}\ g_{1y}\ g_{1\theta}\ g_{1\kappa}]^T$ is the constraint equation corresponding to τ_1 and $\mathbf{g}_2(\mathbf{q}) = [g_{2x}\ g_{2y}\ g_{2\theta}\ g_{2\kappa}]^T$ is the constraint equation corresponding to τ_2. We begin by deriving the derivatives for g_{1k} and g_{2k} with respect to $\mathbf{q}$. These derivatives are straight forward:

$$\begin{aligned}
\frac{\partial g_{1k}}{\partial \mathbf{q}_1} &= \left[s_{f1}\ s_{f1}^2\ s_{f1}^3\ \cdots\ \kappa_1'(\mathbf{q})\right] \\
\frac{\partial g_{1k}}{\partial \mathbf{q}_2} &= [0\ 0\ 0\ \cdots\ 0] \\
\frac{\partial g_{1k}}{\partial \mathbf{q}_3} &= [0\ 0\ 0\ -1] \\
\frac{\partial g_{2k}}{\partial \mathbf{q}_1} &= [0\ 0\ 0\ \cdots\ 0] \\
\frac{\partial g_{2k}}{\partial \mathbf{q}_2} &= \left[s_{f2}\ s_{f2}^2\ s_{f2}^3\ \cdots\ \kappa_2'(\mathbf{q})\right] \\
\frac{\partial g_{2k}}{\partial \mathbf{q}_3} &= [0\ 0\ 0\ 1]
\end{aligned} \tag{B.0.3}$$

where

$$\begin{aligned}
\kappa_1'(\mathbf{q}) &= b_1 + 2c_1 s_{f1} + 3d_1 s_{f1}^2 + \cdots \\
\kappa_2'(\mathbf{q}) &= b_2 + 2c_2 s_{f2} + 3d_2 s_{f2}^2 + \cdots .
\end{aligned}$$

The derivatives of $g_{\theta 1}$ and $g_{\theta 2}$ are also straight forward:

$$\begin{aligned}
\frac{\partial g_{\theta 1}}{\partial \mathbf{q}_1} &= \left[\frac{s_{f1}^2}{2} \; \frac{s_{f1}^3}{3} \; \frac{s_{f1}^4}{4} \; \cdots \; \kappa_1(\mathbf{q})\right] \\
\frac{\partial g_{\theta 1}}{\partial \mathbf{q}_2} &= [0 \; 0 \; 0 \; \cdots \; 0] \\
\frac{\partial g_{\theta 1}}{\partial \mathbf{q}_3} &= [0 \; 0 \; -1 \; 0] \\
\frac{\partial g_{\theta 2}}{\partial \mathbf{q}_1} &= [0 \; 0 \; 0 \; \cdots \; 0] \\
\frac{\partial g_{\theta 2}}{\partial \mathbf{q}_2} &= \left[\frac{s_{f2}^2}{2} \; \frac{s_{f2}^3}{3} \; \frac{s_{f2}^4}{4} \; \cdots \; \kappa_2(\mathbf{q})\right] \\
\frac{\partial g_{\theta 2}}{\partial \mathbf{q}_3} &= [0 \; 0 \; 1 \; s_{f2}] \, .
\end{aligned} \tag{B.0.4}$$

The derivatives of g_{x1} and g_{y1} can be derived by the same method as in the previous appendix. Also note that g_{x1} and g_{y1} do not depend at all on $\mathbf{q}_2$.

The derivatives are:

$$\begin{aligned}
\frac{\partial g_{x1}}{\partial \mathbf{q}_1} &= \begin{bmatrix} -\frac{1}{2}\int_0^{s_{f1}} s^2 f_{y1}(\mathbf{p}_1, s)ds \\ -\frac{1}{3}\int_0^{s_{f1}} s^3 f_{y1}(\mathbf{p}_1, s)ds \\ -\frac{1}{4}\int_0^{s_{f1}} s^4 f_{y1}(\mathbf{p}_1, s)ds \\ \vdots \\ f_{x1}(\mathbf{p}_1, s_{f1}) \end{bmatrix}^T \\
\frac{\partial g_{x1}}{\partial \mathbf{q}_2} &= [0\ 0\ 0\ \cdots\ 0] \\
\frac{\partial g_{x1}}{\partial \mathbf{q}_3} &= [-1\ 0\ 0\ 0] \qquad \text{(B.0.5)} \\
\frac{\partial g_{y1}}{\partial \mathbf{q}_1} &= \begin{bmatrix} \frac{1}{2}\int_0^{s_{f1}} s^2 f_{x1}(\mathbf{p}_1, s)ds \\ \frac{1}{3}\int_0^{s_{f1}} s^3 f_{x1}(\mathbf{p}_1, s)ds \\ \frac{1}{4}\int_0^{s_{f1}} s^4 f_{x1}(\mathbf{p}_1, s)ds \\ \vdots \\ f_{y1}(\mathbf{p}_1, s_{f1}) \end{bmatrix}^T \\
\frac{\partial g_{y1}}{\partial \mathbf{q}_2} &= [0\ 0\ 0\ \cdots\ 0] \\
\frac{\partial g_{y1}}{\partial \mathbf{q}_3} &= [0\ -1\ 0\ 0] \qquad \text{(B.0.6)}
\end{aligned}$$

where

$$\begin{aligned}
f_{x1}(\mathbf{p}_1, s) &= \cos\theta_1(\mathbf{p}_1, s) \qquad \text{(B.0.7)} \\
f_{y1}(\mathbf{p}_1, s) &= \sin\theta_1(\mathbf{p}_1, s).
\end{aligned}$$

The only remaining derivatives are the derivatives of g_{x2} and g_{y2} with respect to $\mathbf{q}$. Noting that g_{x2} and g_{y2} do not depend at all on $\mathbf{q}_1$, the derivatives with respect to $\mathbf{q}_1$ are:

$$\begin{aligned} \frac{\partial g_{x2}}{\partial \mathbf{q}_1} &= [0\ 0\ 0\ \cdots\ 0] \\ \frac{\partial g_{y2}}{\partial \mathbf{q}_1} &= [0\ 0\ 0\ \cdots\ 0]\,. \end{aligned} \tag{B.0.8}$$

The derivatives of g_{x2} and g_{y2} with respect to $\mathbf{q}_2$ can be derived by the same method in the previous appendix. These derivatives are:

$$\begin{aligned} \frac{\partial g_{x2}}{\partial \mathbf{q}_2} &= \begin{bmatrix} -\frac{1}{2}\int_0^{s_{f2}} s^2 f_{y2}(\mathbf{p}_2, s)ds \\ -\frac{1}{3}\int_0^{s_{f2}} s^3 f_{y2}(\mathbf{p}_2, s)ds \\ -\frac{1}{4}\int_0^{s_{f2}} s^4 f_{y2}(\mathbf{p}_2, s)ds \\ \vdots \\ f_{x2}(\mathbf{p}_2, s_{f2}) \end{bmatrix}^T \\ \frac{\partial g_{y2}}{\partial \mathbf{q}_2} &= \begin{bmatrix} \frac{1}{2}\int_0^{s_{f2}} s^2 f_{x2}(\mathbf{p}_2, s)ds \\ \frac{1}{3}\int_0^{s_{f2}} s^3 f_{x2}(\mathbf{p}_2, s)ds \\ \frac{1}{4}\int_0^{s_{f2}} s^4 f_{x2}(\mathbf{p}_2, s)ds \\ \vdots \\ f_{y2}(\mathbf{p}_2, s_{f2}) \end{bmatrix}^T \end{aligned} \tag{B.0.9}$$

where

$$f_{x2}(\mathbf{p}_2, s) = \cos\theta_2(\mathbf{p}_2, s) \tag{B.0.10}$$
$$f_{y2}(\mathbf{p}_2, s) = \sin\theta_2(\mathbf{p}_2, s).$$

Lastly, we define the derivatives of g_{x2} and g_{y2} with respect to $\mathbf{q}_3$. The derivatives with respect to x_w and y_w are:

$$\begin{aligned} \frac{\partial g_{x2}}{\partial x_w} &= 1 \\ \frac{\partial g_{y2}}{\partial y_w} &= 1 \\ \frac{\partial g_{x2}}{\partial y_w} &= 0 \\ \frac{\partial g_{y2}}{\partial x_w} &= 0. \end{aligned} \tag{B.0.11}$$

The Leibnitz's rule must be used in order to calculate the derivatives with respect to θ_w and κ_w. Rewriting g_{x2} and g_{y2} as

$$\begin{aligned} g_{x2} &= x_w - x_f + \int_0^{s_{f2}} f_{x2}(\mathbf{p}_2, s)ds \\ g_{y2} &= y_w - y_f + \int_0^{s_{f2}} f_{y2}(\mathbf{p}_2, s)ds \end{aligned} \tag{B.0.12}$$

gives way to a straightforward use of Leibnitz' rule. The final derivatives are:

$$\begin{aligned}
\frac{\partial g_{x2}}{\partial \theta_w} &= -\int_0^{s_{f2}} f_{y2}(\mathbf{p}_2, s)ds \\
\frac{\partial g_{x2}}{\partial \kappa_w} &= -\int_0^{s_{f2}} s f_{y2}(\mathbf{p}_2, s)ds \\
\frac{\partial g_{y2}}{\partial \theta_w} &= \int_0^{s_{f2}} f_{x2}(\mathbf{p}_2, s)ds \\
\frac{\partial g_{y2}}{\partial \kappa_w} &= \int_0^{s_{f2}} s f_{x2}(\mathbf{p}_2, s)ds.
\end{aligned} \tag{B.0.13}$$

Appendix C

Calculation of Cost Function Derivatives

The derivatives of the cost function needed for the single path generation problem from section 2.3 directly follow the formulations in [13]. These formulations are outlined here.

The cost function used in this thesis, (2.5.1), is:

$$J(\mathbf{q}) = \frac{1}{2}\int_0^{s_f} \kappa^2(\mathbf{p}, s)ds. \tag{C.0.1}$$

The Leibnitz's rule is used to calculate the following relation for the the gradient with respect to $\mathbf{q}$:

$$\frac{\partial J}{\partial \mathbf{q}} = \int_0^{s_f} \kappa(\mathbf{q})\frac{\partial \kappa}{\partial \mathbf{q}}ds. \tag{C.0.2}$$

This relation is used to calculate the following derivatives:

$$\frac{\partial J}{\partial \mathbf{q}} = \left[\frac{\partial J}{\partial b} \; \frac{\partial J}{\partial c} \; \frac{\partial J}{\partial d} \; \cdots \; \frac{\partial J}{\partial s_f} \right]. \tag{C.0.3}$$

Equations (2.1.4) and (A.0.3) give the curvature and partial derivatives of the curvature with respect to $\mathbf{q}$:

$$\begin{aligned} \kappa(s) &= \kappa_0 + bs + cs^2 + ds^3 + \cdots \\ \frac{\partial \kappa}{\partial \mathbf{q}} &= \left[s_f \; s_f^2 \; s_f^3 \; \cdots \; \kappa'(\mathbf{q}) \right]. \end{aligned} \tag{C.0.4}$$

Using these partial derivatives along with (C.0.3), the derivatives of the cost function can be calculated. Defining the following equation is helpful for the notation of these derivatives:

$$K^n = \kappa_0 \frac{s^{n+1}}{n+1} + b\frac{s^{n+2}}{n+2} + c\frac{s^{n+3}}{n+3} + d\frac{s^{n+4}}{n+4} + \cdots . \tag{C.0.5}$$

Then the derivatives are:

$$\frac{\partial J}{\partial \mathbf{q}} = \left[\begin{array}{cccc} K^1 & K^2 & K^3 & \cdots \quad \dfrac{\partial J}{\partial s_f} \end{array} \right] \tag{C.0.6}$$

where

$$\frac{\partial J}{\partial s_f} = \frac{1}{2}\kappa(\mathbf{q})^2. \tag{C.0.7}$$

The derivatives of the cost function needed for the way point optimization problem in Section 2.4 is now formulated. The cost function for this problem

can be written:

$$\begin{aligned} J(\mathbf{q}) &= J_1(\mathbf{q}) + J_2(\mathbf{q}) \\ J_1(\mathbf{q}) &= \frac{1}{2}\int_0^{s_{f1}} \kappa_1^2(\mathbf{q})ds \\ J_2(\mathbf{q}) &= \frac{1}{2}\int_0^{s_{f2}} \kappa_2^2(\mathbf{q})ds \end{aligned} \tag{C.0.8}$$

where $\kappa_1(\mathbf{q}), \kappa_2(\mathbf{q})$, and $\mathbf{q}$ are as in (B.0.1). Noticing that $\kappa_1(\mathbf{q})$ does not depend at all on $\mathbf{q}_2$ and that $\kappa_2(\mathbf{q})$ does not depend at all on $\mathbf{q}_1$, the derivatives of $J(\mathbf{q})$ with respect to $\mathbf{q}_1$ and $\mathbf{q}_2$ can be calculated using the same method as in the previous section. If we define

$$\begin{aligned} K_1^n &= \kappa_0\frac{s^{n+1}}{n+1} + b_1\frac{s^{n+2}}{n+2} + c_1\frac{s^{n+3}}{n+3} + d_1\frac{s^{n+4}}{n+4} + \cdots \\ K_2^n &= \kappa_w\frac{s^{n+1}}{n+1} + b_2\frac{s^{n+2}}{n+2} + c_2\frac{s^{n+3}}{n+3} + d_2\frac{s^{n+4}}{n+4} + \cdots, \end{aligned} \tag{C.0.9}$$

then

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{q}_1} &= \left[\begin{array}{cccc:c} K_1^1 & K_1^2 & K_1^3 & \vdots & \frac{1}{2}\kappa_1(\mathbf{q})^2 \end{array}\right] \\ \frac{\partial J}{\partial \mathbf{q}_2} &= \left[\begin{array}{cccc:c} K_2^1 & K_2^2 & K_2^3 & \vdots & \frac{1}{2}\kappa_2(\mathbf{q})^2 \end{array}\right]. \end{aligned} \tag{C.0.10}$$

It is also apparent that $J_1(\mathbf{q})$ does not depend on $\mathbf{q}_3$ and $J_2(\mathbf{q})$ does not depend on x_w, y_w and θ_w. The derivative of $J_2(\mathbf{q})$ with respect to κ_w can be calculated using the Leibnitz's rule and is:

$$\frac{\partial J}{\partial \kappa_w} = s_{f2}. \tag{C.0.11}$$

Then

$$\frac{\partial J}{\partial \mathbf{q}_3} = [0 \ 0 \ 0 \ s_{f2}]^T . \tag{C.0.12}$$

And finally

$$\frac{\partial J}{\partial \mathbf{q}} = \left[\left(\frac{\partial J}{\partial \mathbf{q}_1} \right)^T \left(\frac{\partial J}{\partial \mathbf{q}_2} \right)^T \left(\frac{\partial J}{\partial \mathbf{q}_3} \right)^T \right]^T . \tag{C.0.13}$$

Appendix D

MATLAB Simulation Code

```
function [ W, x, y, Q, J ] = Algorithm_1( xo, yo, thetao, ko, w, l, ...
    thetaguesses, sfguesses )
% The inputs are: a matrix, w, where the row vectors are the center points
% of the waypoints, the length, l, of the waypoint boundaries, a vector
% of the initial theta values, thetaguess, for each waypoint, the initial
% values for the arc length, sfguesses, of each path segment connecting the
% waypoints, and the starting point of the track (xo, yo, thetao, ko). Note
% that w ends with the final waypoint and theta guesses ends with the
% angle of the tangenth to the end of the track (not the final waypoint
% theta value).

% The outputs are: a matrix, W, where the row vectors are the optimized
% waypoints, the x and y values, x and y, of the optimized paths
% connecting the waypoints, a matrix, Q, where the row vectors are the
% paramters describing the optimized paths connecting the waypoints, and a
% vector, J, that has the cost function values at the end of each
% iteration.


% Calculate the number of waypoints.
num_wp = size(w) - 1;
num_wp = num_wp(1);

% Initial guesses for the waypoint theta values.
theta = ones(num_wp,1);
for i = 1:num_wp
    theta(i) = thetaguesses(i);
end

% Initialize the matrix of optimized waypoints including intial and final
```

```
% conditions.
W = ones(num_wp, 4);
for i = 1:num_wp
    W(i,:) = [w(i,:), theta(i), 0];
end
W = [xo, yo, thetao, ko; W; w(end,:), thetaguesses(end), 0];

% Algorithm 1
j = 0;
dJ = 1;
 while dJ > .001
    j = j + 1;
    for i = 2:size(W) - 1

        % Optimize the waypoint postions.
        [ q ] = wp_opt(W(i-1,1), W(i-1,2), W(i-1,3), W(i-1,4),...
                    W(i+1,1), W(i+1,2), W(i+1,3), W(i+1,4), w(i-1,1),...
                    w(i-1,2), l, W(i,1), W(i,2), W(i,3), W(i,4),...
                    sfguesses(i-1), sfguesses(i));
        W(i,:) = [q(9), q(10), q(11), q(12)];
        sfguesses(i-1) = q(4);
        sfguesses(i) = q(8);
    end

    % Create the paths and calculate the cost function value at the
    % jth iteration.
    Q = ones(num_wp + 1, 4);
    for i = 1:num_wp + 1
        Q(i,:) = path_opt(W(i,1), W(i,2), W(i,3), W(i,4), W(i+1,1),...
        W(i+1,2), W(i+1,3), W(i+1, 4), sfguesses(i));
    end
    J(j) = cost_function(Q, W(:,4));
    if j > 1
        dJ = abs(J(j) - J(j-1));
    end
 end

% Create the path x and y vectors.
x = ones(101, num_wp + 1);
y = x;
    for i = 1:num_wp + 1
        xy = x_vs_y(W(i,1), W(i,2), W(i,3), W(i,4), Q(i,:));
        x(:,i) = xy(:,1);
        y(:,i) = xy(:,2);
    end
end
```

```
function [ wp, Q ] = Algorithm_2( xin, yin, L, num_wp, is_loop )
% The inputs are: two column vectors, xin and yin, that are the x and y
% values of the inner track line, a width, L, of the track, an intial
% number of waypoints, num_wp, and a Boolean indicating if the track is a
% loop.

% The outputs are: the optimized waypoint positions, wp, the optimized
% path as a matrix, Q, of parameter vectors describing each path section
% connecting the optimized waypoints, and a plot of the optimal path
% through the track.

% Center and outer lines of the track and the angle of the tangent to
% center line at each point.
[XY, theta_c] = track(xin ,yin, L);

% Distance vector corresponding to the track center line.
S = arclength(XY(:,1), XY(:,2));

% Algorithm 2
in_track = false;
J = 1000;
dJ = -1;
j = 0;
while (~(in_track && dJ > 0))

    % Setting intial values for the waypoints.
    X = ones(num_wp + 1, 4);
    s_increment = S(end)/(num_wp + 1);
    sfguesses = s_increment * ones(num_wp + 1, 1);
    for i = 1:num_wp
        k = find(S >= i*s_increment, 1);
        X(i, 1) = XY(k, 1);
        X(i, 2) = XY(k, 2);
        X(i, 3) = theta_c(k + 1);
        X(i, 4) = 0;
    end

    % Adding the track endpoint
    X(num_wp + 1, 1) = XY(end, 1);
    X(num_wp + 1, 2) = XY(end, 2);
    if (is_loop)
        X(num_wp + 1, 3) = 2*pi;
    else
        X(num_wp + 1, 3) = theta_c(end);
    end
    X(num_wp + 1, 4) = 0;

    % Setting length of the square waypoint boundaries.
```

```
    l = L/sqrt(2);

    % Optmize the waypoints and create the optimal paths conecting the
    % waypoints.
    [wp_new, Xtrj_new, Ytrj_new, Q_new, J1] = ...
        Algorithm_1(0, 0, 0, 0, X(:,1:2), l, X(:,3), sfguesses);
    Jnew = J1(end);

    % Check the path for collisions with the track and if the cost function
    % is still decreasing.
    xq = reshape(Xtrj_new,[],1);
    yq = reshape(Ytrj_new,[],1);
    if (is_loop)
        xv = [XY(:,3); NaN; xin];
        yv = [XY(:,4); NaN; yin];
    else
        xv = [xin; flipud(XY(:,3))];
        yv = [yin; flipud(XY(:,4))];
        xq = xq(30:end-30);
        yq = yq(30:end-30);
    end
    in = inpolygon(xq, yq, xv, yv);
    in_track = numel(xq(~in)) == 0;
    j = j + 1;
    dJ = Jnew - J;
    if (~(in_track && dJ > 0) || j == 1)
        Xtrj = Xtrj_new;
        Ytrj = Ytrj_new;
        Q = Q_new;
        wp = wp_new;
        J = Jnew;
        num_wp = num_wp + 1;
    end
end

% Plot the optimal path through the track.
plot(xin, yin, XY(:,3), XY(:,4), Xtrj, Ytrj)
end
```

```
function [ q ] = wp_opt( xo, yo, thetao, ko, xf, yf, thetaf, kf, xcwp,...
    ycwp, l, x1o, y1o, theta1o, k1o, s1o, s2o)
% The inputs are: the path initial constraints (xo, yo, thetao, ko), the
% path final constraints (xf, yf, thetaf, kf), the center of the waypoint
% (xcwp, ycwp), the width of the waypoint, l, the initial guess for the
% optimal waypoint (x1o, y1o, theta1o, k1o), and the initial guesses for
% the lengths of paths 1 and 2, s1o and s2o, respectively (if s1o and s2o
```

```
% are inputted as zero a default value is used).

% The output is: the parameter vector for the optimization problem.

% Functions to break down the parameter vector.
q1_in_q = @(q) [q(1); q(2); q(3); q(4)];
q2_in_q = @(q) [q(5); q(6); q(7); q(8)];
q3_in_q = @(q) [q(9); q(10); q(11); q(12)];

% Setting up intial values & guesses.
if s1o == 0
    s1o = ((theta1o - thetao)^2)/5 + 1;
end
if s2o == 0
    s2o = ((thetaf - theta1o)^2)/5 + 1;
end
q3o = [x1o; y1o; theta1o; k1o];

% Get more accurate initial values for the parameters.
q1o = transpose(path_opt(xo, yo, thetao, ko, x1o, y1o, theta1o, k1o, s1o));
q2o = transpose(path_opt(x1o, y1o, theta1o, k1o, xf, yf, thetaf, kf, s2o));
q1o = transpose(q1o);
q2o = transpose(q2o);
qo = [q1o;q2o;q3o];

% Calculate the boundaries on the waypoint.
xlb = xcwp - l/2;
xub = xcwp + l/2;
ylb = ycwp - l/2;
yub = ycwp + l/2;

% Define the system equations.
fCn = @(thetao, ko, b, c, d, s, n) (s.^n).*cos(thetao + ko.*s +...
    b.*(s.^2)/2 + c.*(s.^3)/3 + d.*(s.^4)/4);
fSn = @(thetao, ko, b, c, d, s, n) (s.^n).*sin(thetao + ko.*s +...
    b.*(s.^2)/2 + c.*(s.^3)/3 + d.*(s.^4)/4);
Cn = @(thetao, ko, q, n) integral(@(s) fCn(thetao, ko, q(1), q(2), q(3),...
    s, n), 0, q(4));
Sn = @(thetao, ko, q, n) integral(@(s) fSn(thetao, ko, q(1), q(2), q(3),...
    s, n), 0, q(4));

g1 = @(xo, thetao, ko, q) xo + Cn(thetao, ko, q, 0);
g2 = @(yo, thetao, ko, q) yo + Sn(thetao, ko, q, 0);
g3 = @(thetao, ko, q) thetao + ko*q(4) + q(1)*(q(4)^2)/2 +...
    q(2)*(q(4)^3)/3 + q(3)*(q(4)^4)/4;
g4 = @(ko, q) ko + q(1)*q(4) + q(2)*(q(4)^2) + q(3)*(q(4)^3);

g = @(xo, yo, thetao, ko, q) [g1(xo, thetao, ko, q); g2(yo, thetao,...
    ko, q); g3(thetao, ko, q); g4(ko, q)];
```

```
X1 = @(q) [q(9); q(10); q(11); q(12)];
Xf = [xf; yf; thetaf; kf];
g4prime = @(q) q(1) + 2*q(2)*q(4) + 3*q(3)*q(4)^2;
gneg = @(q) [X1(q) - g(xo, yo, thetao, ko, q1_in_q(q));
             Xf - g(q(9), q(10), q(11), q(12), q2_in_q(q))];

Jeq = @(b, c, d, s) (ko + b.*s + c.*(s.^2) + d.*(s.^3)).^2;
J = @(q) integral(@(s) Jeq(q(1), q(2), q(3), s), 0, q(4))/2 +...
    integral(@(s) Jeq(q(5), q(6), q(7), s), 0, q(8))/2;

% First order conditions.
dgdq1 = @(q) [-Sn(thetao, ko, q1_in_q(q), 2)/2, -Sn(thetao, ko,...
    q1_in_q(q), 3)/3, -Sn(thetao, ko, q1_in_q(q), 4)/4,...
    cos(g3(thetao, ko, q1_in_q(q))), 0, 0, 0, 0,-1, 0, 0, 0];
dgdq2 = @(q) [Cn(thetao, ko, q1_in_q(q), 2)/2, Cn(thetao, ko,...
    q1_in_q(q), 3)/3, Cn(thetao, ko, q1_in_q(q), 4)/4,...
    sin(g3(thetao, ko, q1_in_q(q))), 0, 0, 0, 0, 0, -1, 0, 0];
dgdq3 = @(q) [q(4)^2/2, q(4)^3/3, q(4)^4/4, g4(ko, q1_in_q(q)),...
    0, 0, 0, 0, 0, 0, -1, 0];
dgdq4 = @(q) [q(4), q(4)^2, q(4)^3, g4prime(q1_in_q(q)), 0,...
    0, 0, 0, 0, 0, 0, -1];
dgdq5 = @(q) [0, 0, 0, 0, -Sn(q(11), q(12), q2_in_q(q), 2)/2,...
    -Sn(q(11), q(12), q2_in_q(q), 3)/3, -Sn(q(11),...
    q(12), q2_in_q(q), 4)/4, cos(g3(q(11), q(12), q2_in_q(q))), 1, 0,...
    -Sn(q(11), q(12), q2_in_q(q), 0), -Sn(q(11), q(12), q2_in_q(q), 1)];
dgdq6 = @(q) [0, 0, 0, 0, Cn(q(11), q(12), q2_in_q(q), 2)/2,...
     Cn(q(11), q(12), q2_in_q(q), 3)/3, Cn(q(11), q(12), q2_in_q(q),...
     4)/4, sin(g3(q(11), q(12), q2_in_q(q))), 0, 1,...
     Cn(q(11), q(12), q2_in_q(q), 0), Cn(q(11), q(12), q2_in_q(q), 1)];
dgdq7 = @(q) [0, 0, 0, 0, q(8)^2/2, q(8)^3/3, q(8)^4/4, g4(q(12),...
    q2_in_q(q)), 0, 0, 1, q(8)];
dgdq8 = @(q) [0, 0, 0, 0, q(8), q(8)^2, q(8)^3, g4prime(q2_in_q(q)),...
    0, 0, 0, 1];
dgdq = @(q) [dgdq1(q); dgdq2(q); dgdq3(q); dgdq4(q); dgdq5(q); dgdq6(q);...
    dgdq7(q); dgdq8(q)];

kn = @(ko, q, n) ko*q(4)^(n + 1)/(n + 1) + q(1)*q(4)^(n + 2)/(n + 2) +...
    q(2)*q(4)^(n + 3)/(n + 3) + q(3)*q(4)^(n + 4)/(n + 4);
dJdq = @(q) [kn(ko, q1_in_q(q), 1), kn(ko, q1_in_q(q), 2),...
    kn(ko, q1_in_q(q), 3), g4(ko, q1_in_q(q))^2/2, kn(q(12), q2_in_q(q),...
    1), kn(q(12), q2_in_q(q), 2), kn(q(12), q2_in_q(q), 3),...
    g4(q(12), q2_in_q(q))^2/2, 0, 0, 0, g3(0, q(12), q2_in_q(q))];

% Run the solver.
lb = [-1000; -1000; -1000; -1000; -1000; -1000; -1000; -1000;
      xlb; ylb; -1000; -1000];
ub = [1000; 1000; 1000; 1000; 1000; 1000; 1000; 1000; xub; yub;
      1000; 1000];
```

```
options = optimoptions(@fmincon, 'SpecifyObjectiveGradient',true,...
    'SpecifyConstraintGradient',true,'HessianApproximation',...
    'finite-difference', 'SubproblemAlgorithm', 'cg',...
    'MaxFunctionEvaluations',10000, 'MaxIterations',2000);
nonlcon = @con;
q = fmincon(@fun,qo,[],[],[],[],lb,ub,nonlcon,options);

% Functions used in the solver.
    function [c,ceq, gc, gceq] = con(x)
        c = [];
        ceq = gneg(x);
        gc = [];
        gceq = transpose(dgdq(x));
    end

    function [f, g] = fun(x)
        f = J(x);
        g = dJdq(x);
    end
end
```

```
function [ q ] = path_opt( xo, yo, thetao, ko, xf, yf, thetaf, kf, sfo)
% The inputs are: the initial constraints for a path (xo, yo,
% thetao, ko), the final constraints for the path (xf, yf, thetaf, kf), and
% an initial guess, sfo, for the parameter sf. If sfo is zero a default
% value is used.

% The outputs are: the parameter vector for a path that satifies the
% inputted constraints and is optimized with the smoothness cost function.

% Setting up initial guesses.
a = ko;
if (sfo == 0)
    sfo = ((thetaf - thetao)^2)/5 + 1;
end

% Calculating inproved initial guesses.
qo = initial_path( xo, yo, thetao, ko, xf, yf, thetaf, kf, sfo);
Xf = [xf; yf; thetaf; kf];

% Defining the system equations.
fCn = @(b, c, d, s, n) (s.^n).*cos(thetao + a.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
fSn = @(b, c, d, s, n) (s.^n).*sin(thetao + a.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
Cn = @(q, n) integral(@(s) fCn(q(1), q(2), q(3), s, n), 0, q(4));
```

```
Sn = @(q, n) integral(@(s) fSn(q(1), q(2), q(3), s, n), 0, q(4));

g1 = @(q) xo + Cn(q, 0);
g2 = @(q) yo + Sn(q, 0);
g3 = @(q) thetao + a*q(4) + q(1)*(q(4)^2)/2 + q(2)*(q(4)^3)/3 +...
    q(3)*(q(4)^4)/4;
g4 = @(q) a + q(1)*q(4) + q(2)*(q(4)^2) + q(3)*(q(4)^3);
g = @(q) [g1(q); g2(q); g3(q); g4(q)];
gneg = @(q) Xf - g(q);

Jeq = @(b, c, d, s) (a + b.*s + c.*(s.^2) + d.*(s.^3)).^2;
J = @(q) integral(@(s) Jeq(q(1), q(2), q(3), s), 0, q(4))/2;

% First order conditions.
kprime = @(q) q(1) + 2*q(2)*q(4) + 3*q(3)*q(4)^2;
dgdq = @(q) [-Sn(q, 2)/2, -Sn(q, 3)/3, -Sn(q, 4)/4, cos(g3(q));
             Cn(q, 2)/2, Cn(q, 3)/3, Cn(q, 4)/4, sin(g3(q));
             (q(4)^2)/2, (q(4)^3)/3, (q(4)^4)/4, g4(q);
             q(4), q(4)^2, q(4)^3, kprime(q)];
Kn = @(q, n) a*q(4)^(n + 1)/(n + 1) + q(1)*(q(4)^(n + 2))/(n + 2) +...
    q(2)*(q(4)^(n + 3))/(n + 3) + q(3)*(q(4)^(n + 4))/(n + 4);
dJdq = @(q) [Kn(q, 1), Kn(q, 2), Kn(q, 3), (g4(q)^2)/2];

% Run the solver.
nonlcon = @con;
options = optimoptions(@fmincon, 'SpecifyObjectiveGradient',true,...
    'SpecifyConstraintGradient',true,'HessianApproximation',...
    'finite-difference', 'SubproblemAlgorithm', 'cg',...
    'MaxFunctionEvaluations',10000, 'MaxIterations',2000);
q = fmincon(@fun,qo,[],[],[],[],[],[],nonlcon,options);

% Functions needed for the solver.
    function [c, ceq, gc, gceq] = con(x)
        c = [];
        ceq = gneg(x);
        gc = [];
        gceq = transpose(dgdq(x));
    end

    function [f, g] = fun(x)
        f = J(x);
        g = dJdq(x);
    end
end




function [ J ] = cost_function( Q, ko )
```

```
% The inputs are: a matrix, Q, where each row vector contains the
% parameters describing a path and a vector, ko, that contains the intial
% curvature values for each path described by Q.

% The output is: a value of the sum of the cost functions for each path.

% Defining the cost function.
Jeq = @(ko, b, c, d, s) (ko + b.*s + c.*(s.^2) + d.*(s.^3)).^2;

% Summing the cost function values.
Jf = @(ko, q) integral(@(s) Jeq(ko, q(1), q(2), q(3), s), 0, q(4))/2;
if length(ko) == 1
    J = Jf(ko, Q);
else
    J = 0;
    for i = 1:length(Q(:,1))
        J = J + Jf(ko(i), [Q(i,1), Q(i,2), Q(i,3), Q(i,4)]);
    end
end
end
```

```
function [ s ] = arclength( x, y )
% The inputs are: two column vectors decribing the points on a line, x
% and y.

% The outputs are: a cooresponding distance vector, s.

% Initialize the vectors
size = length(x);
s = ones(size, 1);
dx = diff(x);
dy = diff(y);
s(1) = 0;

% Calculate the distance vector.
for i = 2:size
    int = sqrt(dx(i-1)^2 + dy(i-1)^2);
    s(i) = s(i-1) + int;
end
end
```

```
function [ q ] = initial_path( xo, yo, thetao, ko, xf, yf, thetaf, kf, sfo)
% The inputs are: the initial constraints for a path (xo, yo,
% thetao, ko), the final constraints for the path (xf, yf, thetaf, kf), and
```

```
% an initial guess, sfo, for the parameter sf. If sfo is zero a default
% value is used.

% The outputs are: the parameter vector for a path that satifies the
% inputted constraints.

% Setting up initial conditions.
if (sfo == 0)
    sfo = ((thetaf - thetao)^2)/5 + 1;
end
a = ko;
AA = [sfo, sfo^2;
    (sfo^2)/2, (sfo^3)/3];
AB = [kf - ko;
     thetaf - thetao - ko*sfo];
BC = AA\AB;
bo = BC(1);
co = BC(2);
do = 0;
qo = [bo; co; do; sfo];
Xf = [xf; yf; thetaf; kf];

% Defining the system equations.
fCn = @(b, c, d, s, n) (s.^n).*cos(thetao + a.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
fSn = @(b, c, d, s, n) (s.^n).*sin(thetao + a.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
Cn = @(q, n) integral(@(s) fCn(q(1), q(2), q(3), s, n), 0, q(4));
Sn = @(q, n) integral(@(s) fSn(q(1), q(2), q(3), s, n), 0, q(4));

g1 = @(q) xo + Cn(q, 0);
g2 = @(q) yo + Sn(q, 0);
g3 = @(q) thetao + a*q(4) + q(1)*(q(4)^2)/2 + q(2)*(q(4)^3)/3 +...
    q(3)*(q(4)^4)/4;
g4 = @(q) a + q(1)*q(4) + q(2)*(q(4)^2) + q(3)*(q(4)^3);
g = @(q) [g1(q); g2(q); g3(q); g4(q)];
gneg = @(q) Xf - g(q);

% Run the solver.
options = optimoptions(@lsqnonlin, 'MaxFunctionEvaluations',10000,...
    'MaxIterations',2000);
q = lsqnonlin(gneg, qo,[], [], options);
end



function [ XY, theta_c ] = track( xin, yin, L )
% The inputs are: two column vectors, xin and yin, which define the points
% for the inside of a track and the track width, L.
```

```
% The outputs are: a matrix whose column vectors are xc, yc, xo, and yo
% (xc and yc are the points for the center track line and xo and yo are
% the points for the outer track line) and a vector, theta_c, which
% contains the angle of the tangents to the center line for each point
% along the center line.

% Initializing dy and dx.
dy = diff(yin);
dx = diff(xin);
yin = yin(2:end);
xin = xin(2:end);

% Calulate the inner and outer track line vectors.
xc = L/2*sin(atan2(dy, dx)) + xin;
yc = -L/2*cos(atan2(dy, dx)) + yin;
xo = L*sin(atan2(dy, dx)) + xin;
yo = -L*cos(atan2(dy, dx)) + yin;

% Define theta corresponding to each x and y point on the center line.
% Making sure that the angle is commulative from the start of the track
% and starts at zero.
theta_c_1 = atan2(diff(yc),diff(xc));
for i = 1:length(theta_c_1)
    if (theta_c_1(i) < 0)
        theta_c_1(i) = theta_c_1(i) + 2*pi;
    end
end
d_theta_c = diff(theta_c_1);
k = find(abs(d_theta_c) >= pi/2);
for i = 1:length(k)
    d_theta_c(k(i)) = 0;
end
theta_c = ones(length(d_theta_c) + 1, 1);
theta_c(1) = 0;
for i = 1:length(d_theta_c)
    theta_c(i + 1) = theta_c(i) + d_theta_c(i);
end

% Define the XY matrix
XY= [xc, yc, xo, yo];
end



function [ xy ] = x_vs_y( xo, yo, thetao, ko, q)
% The inputs are: a vector, q, of the parameters for a
% path and the intial conditions for the path (xo, yo, thetao, ko).
```

```
% The outputs are: two vectors that are 100 cooresponding x and y
% values for a given path described by the inputs.

% Defining the system equations.
fCn = @(ko, b, c, d, s, n) (s.^n).*cos(thetao + ko.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
fSn = @(ko, b, c, d, s, n) (s.^n).*sin(thetao + ko.*s + b.*(s.^2)/2 +...
    c.*(s.^3)/3 + d.*(s.^4)/4);
Cn = @(ko, q, n) integral(@(s) fCn(ko, q(1), q(2), q(3), s, n), 0, q(4));
Sn = @(ko, q, n) integral(@(s) fSn(ko, q(1), q(2), q(3), s, n), 0, q(4));
g1 = @(ko, q) xo + Cn(ko, q, 0);
g2 = @(ko, q) yo + Sn(ko, q, 0);

% Create the x and y vectors.
x = ones(1,101);
y = ones(1,101);
i = 1;
for si = 0:q(4)/100:q(4)
    x(i) = g1(ko, [q(1), q(2), q(3), si]);
    y(i) = g2(ko, [q(1), q(2), q(3), si]);
    i = i + 1;
end
xy = transpose([x; y]);

end
```