Racing Line Optimisation for Autonomous Vehicles

Joe Davison

Bachelor of Science in Computer Science with Honours
University of Bath

May 2020

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed: 

Racing Line Optimisation for Autonomous Vehicles

Submitted by: Joe Davison

Copyright

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see http://www.bath.ac.uk/ordinances/22.pdf).
This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed: 

Abstract

This project develops a method for generating racing lines, that aims to minimise both lap time and run time. By reducing run time, the use of such methods in a 'live' path planning environment becomes feasible. On-track performance and time to compute a path are often competing objectives, so there is a particular focus on reducing run time for a small trade-off in lap time.

Existing methods in path planning and vehicle dynamics have been adapted for a specific path planning problem. In particular, a compromise between minimising path length and path curvature is used to optimise a racing line. This working model is then used to investigate two key improvements to the method.

A sector-based approach was developed, in which paths through individual corner sequences are optimised and later merged. By performing the individual optimisations in parallel a 50% reduction in run time was achieved. Due to the merging of adjacent sector paths, the full racing line saw lap times increase up to 3%.

The compromise between path length and curvature included a secondary optimisation of the weight between these two metrics. Significant improvements in run time were made by instead modelling track properties to estimate a well-performing compromise weight. This reduced the cost of the optimisation to that of a basic minimum curvature problem, but with lap times closer to the use of an optimal compromise than that of a minimum curvature path.

This dissertation and accompanying code is available at
`github.com/joedavison17/dissertation`.

Contents

List of Figures

List of Tables

List of Symbols

Path Geometry

α	Control Point Parameter
ε	Compromise Weight
ε^*	Optimal Compromise Weight
Γ^2	Sum of the Squares of Curvatures
κ	Curvature
κ_{min}	Corner Curvature Threshold
$\bar{\kappa}$	Mean Curvature
F_ε	Compromise Objective Function
n	Number of Control Points
n_s	Number of Sample Points
r	Radius of Curvature
S	Path Length
s	Path Distance

Vehicle Dynamics

μ	Coefficient of Friction
a	Acceleration
d	Deceleration
F_a	Aerodynamic Downforce Coefficient

F_{lat}	Lateral Force
F_{long}	Longitudinal Force
g	Gravitational Acceleration
m	Mass
T	Lap Time
T^*	Optimal Lap Time
v	Velocity

Other Symbols

ρ	Pearson's Correlation Coefficient

COVID-19 Impact

The coronavirus pandemic occurred during the final months of work on this dissertation, with social distancing measures implemented in the UK and university teaching moving entirely online from 18th March 2020.

Disruption for this project was limited, with testing software-based and no requirement for participants in a user study. Reduced communication with department staff and coursemates meant less informal feedback on the work, and adjustment to working within lockdown measures briefly halted progress. Ultimately, there has been no need to alter the course of the dissertation as a direct result of the pandemic or related measures.

Acknowledgements

Thanks go to my project supervisor Wenbin Li for his advice and feedback, and to friends and family for their support throughout the project.

Chapter 1

Introduction

The fundamental objective for this project is to produce well-performing racing lines in as short a time possible. Racing lines aim to minimise the time for a vehicle to complete laps of a given circuit. These are usually established by drivers from best practices, experience, and knowledge of the track and their vehicle. The wider context of this work is that of autonomous vehicles; the racing lines are to be generated for a driverless vehicle to follow.

The project is introduced with a discussion of autonomous vehicles and their development within motorsport. After covering the problem domain, the scope and objectives of the project are established. An outline for research and development is also given, to direct the reader through the various stages of the project.

1.1 Problem Background

1.1.1 Motivation and Context

Driverless cars are widely seen as the next step in private transport, building on the introduction of driver assistance systems such as cruise control and lane-keeping. High-profile car manufacturers and technology companies are engaging in an autonomous arms race, in an attempt to deliver driverless technology commercially. Previous advances in this industry have often originated in motorsport, from lightweight materials and hybrid technology to team communication and production line efficiency (Kanal, 2019). The competitive, performance-driven environment of motorsport is an ideal setting for the latest automotive technology to be developed. Implementing a path planning process for a race track environment is a useful abstraction of wider problems, such as planning paths and routes on public roads. Fundamental methods can be refined in this constrained and well-defined context, before introducing complications and building the problem up for other uses.

Formula Student is an engineering competition in which student teams design, build and race single-seater vehicles in a series of events. A new branch of the competition, Formula Student A.I., tasks teams with adapting their vehicles for driverless events. Closed-loop circuits are laid out by a series of cones: blue cones for the left boundary, and yellow cones for the right. The driverless car must navigate these circuits as quickly as possible, negotiating different corner types: hairpins, chicanes, and corners of varying radius. This competition sets a useful framework for the project, with a well-defined set of criteria and a simple end goal.

Figure 1.1: AMZ Driverless at Formula Student Germany (Schulz, 2017).

1.1.2 Autonomous Pipeline

In an introduction to the new autonomous Formula Student category, Georgiev and Day (2018) suggest the following autonomous pipeline.

1. **Perception**
   Identification and classification of cones.
2. **SLAM: Simultaneous Localisation and Mapping**
   Construction and maintenance of a track map and vehicle location.
3. **Path Planning**
   Construction of the trajectory.
4. **Vehicle Control**
   Execution of the path by controlling the steering, throttle and brakes.

This project lies within the path planning stage. There are two components to path planning: local and global optimisation. Local path planning is used while 'learning' the track, finding a suitable path through the local, visible region. Global optimisation will be the focus of this project, used after this reconnaissance phase to compute a racing line. The locations of all cones are assumed to be known and available from the previous stages of the pipeline. Their classification, i.e. which boundary they form, are also known. This defines the input for the path planning stage. The control component of the pipeline requires a path to follow and target velocities to match.

The Formula Student events and autonomous pipeline provide well-defined input, output and constraints for the problem domain. Methods for computing a racing line are developed and tested within this framework.

1.2 Project Description

This dissertation is best described as a research and development project. The research element will investigate appropriate methods for generating racing lines within the problem domain, with the development of a working prototype supporting investigations and acting as a proof of concept. Two key objectives motivate the work: minimising the lap time of resulting trajectories and minimising the computation time to produce them. These are competing objectives, so trade-offs between them are a common theme throughout the project.

Most attention is given to developing the method for optimising a racing line. Naturally, there are many branches from this core topic that could be taken, such as the mathematical detail for optimisation problems and intricate engineering modelling of vehicles. To limit the scope of this work, considerations for such areas will be made without losing focus of the fundamental aims described. The methods developed during this project should allow for extensions or refinement in these related topics.

Testing is purely software-based, for the fast and straightforward gathering of results. Ideally, real-world verification would be carried out, but this would require a fully functioning system as per the pipeline in Section 1.1.2. Lap times have been estimated using the dynamics model implemented.

Chapter 2 presents a survey of the current state-of-the-art and research into key concepts and methods. A fundamental working model is then developed in Chapter 3, with some early analysis of the chosen methods. Potential improvements to the core process are then implemented and analysed in Chapter 4. Conclusions in Chapter 5 summarise the work, its implications, and the potential for further investigation.

1.2.1 Requirements

Based on the context and aims described, the requirements for this project and the resulting proof of concept can be given. Most requirements apply to the method of generating racing lines. Other requirements are included for best practice in software development so that any prototype code is useful beyond this project. High priority requirements are considered essential to a minimum working example.

Functional Requirements

Fundamentals

R1.1 The system should take as input 2D coordinates of the cones, and their classification in terms of a left or right boundary.

High priority.

R1.2 A complete, closed path around a track of given boundaries must be generated.

High priority.

R1.3 A velocity profile must be generated for the path.

High priority.

R1.4 The system must handle all features of a Formula Student circuit. Specifically: chicanes, hairpins, and variable radius corners.

High priority.

Extensions

R2.1 The path must stay within the boundaries of the circuit. The width of the car needs to be considered to avoid cone contact.

Medium priority.

R2.2 Vehicle models of varying complexity should be supported.

Medium priority.

R2.3 Vehicle dynamics should be considered in the path optimisation.

Medium priority.

R2.4 The track map, optimised path, and velocity profile should be displayed for testing/verification purposes.

Medium priority.

R2.5 Parameters for vehicle dynamics should be customisable, to suit different conditions. For example, accounting for less grip in wet weather.

Low priority.

Non-Functional Requirements

Performance

R3.1 The generated trajectory must improve on lap times that follow a centerline path.

High priority.

R3.2 The generated trajectory should improve on lap times that follow a minimum curvature path.

Medium priority.

R3.3 A trajectory should be generated within 30 seconds of receiving cone locations.

Medium priority.

Software Quality

R4.1 The structure and format of data output by the system must be clearly and completely defined.

High priority.

R4.2 The system must be robust, handling errors gracefully and appropriately.

High priority.

R4.3 The system should be informative of progress and errors to the user (or other components in a full implementation).

Medium priority.

R4.4 The system should be well-documented, with appropriate commenting of code.

Medium priority.

Chapter 2

Literature Review

In this chapter, literature relevant to the project is described and reviewed. A brief overview of autonomous driving as a whole is given, after which different methods for path planning are discussed. These high-level approaches are reviewed in general, with further analysis of specific implementations. Path representation systems, velocity profiling and mathematical optimisation are also explored.

2.1 Autonomous Vehicles Overview

2.1.1 Early Automation

Research into autonomous vehicles has been conducted for several decades, with early projects laying the framework for developing driverless cars. One such project was Eureka's Prometheus - a combined academic and industrial effort commissioned in 1987. The work resulted in an impressive demonstration in 1995 of a modified Mercedes S-Class completing a return journey from Munich to Copenhagen, although a human driver supervised and often intervened (Billington, 2018).

The inaugural DARPA Grand Challenge, an off-road driverless competition held in 2004, required a significant step in autonomous technology. Teams were tasked with traversing a 140-mile route within a 10-hour limit, entirely unassisted (Iagnemma and Buehler, 2006). No vehicle made it beyond 7.4 miles, and prize money was not awarded. A second, more successful event was held in 2005 and five vehicles were able to complete the course. This improvement in performance over one year demonstrates the advances in autonomous technology that can be made when significant resources are applied.

2.1.2 Grading Autonomous Systems

The Society of Automotive Engineers (SAE) has defined a standard for classifying autonomous systems, which sets out six levels of autonomy (SAE International, 2018). The standard has been widely adopted, with the National Highway Traffic Safety Administration (NHTSA) even accepting SAE's system over their own. Figure 2.1 gives an overview of the different levels, ranging from warning systems at level 0, to a complete, all-purpose driverless car at level 5.

A level 4 system would be developed for the Formula Student competition, as there will not be a seated driver during dynamic events. However, the conditions required for autonomy are extremely constrained - with the system dependent on operating within a closed track defined by the two sets of cones. It is these constraints that both simplify the problem in this project and prevent it from being a completely independent autonomous system.

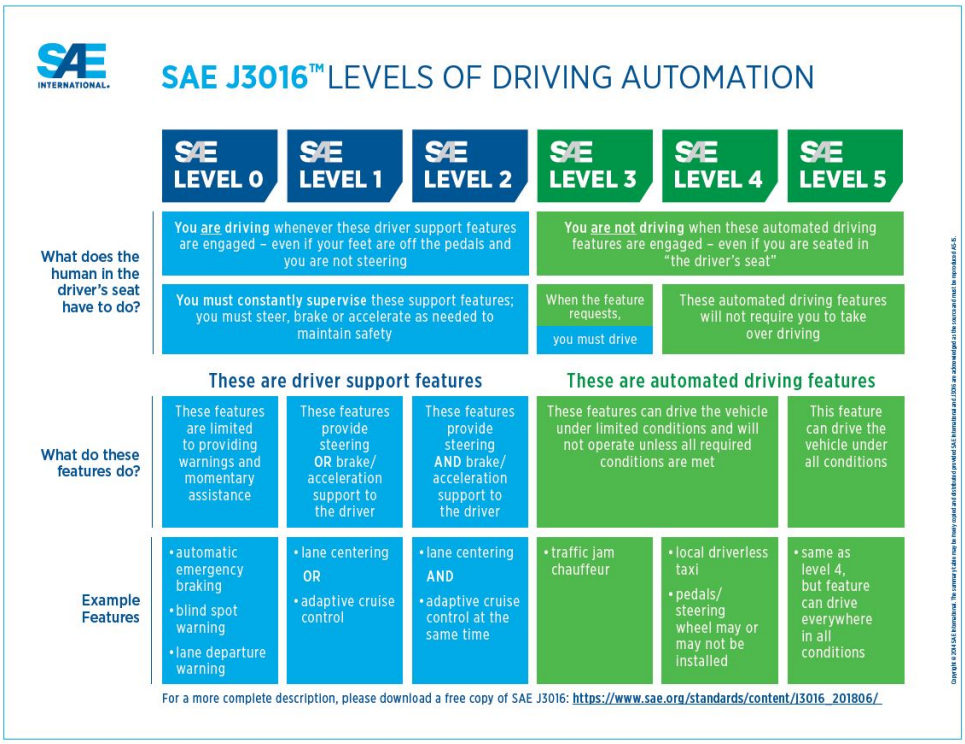


Figure 2.1: The six levels of automated driving (SAE International, 2019).

2.1.3 Recent High-Profile Projects

Now that level 1 features such as adaptive cruise control are relatively common on road cars, development of level 3+ features have entered mainstream focus. Tesla are considered the market leaders with their Autopilot system, which offers automated driving primarily on highways (Tesla, 2019). While Autopilot could be considered level 3 on SAE's scale, the driver is still required to "supervise" the autonomous system - with full attention and awareness of the situation expected.

Another recent development has been the introduction of Roborace - the first autonomous racing series. The competition was created to accelerate autonomous development, with vehicles tested to their limits in controlled environments (Roborace, 2019). This mirrors the aims of Formula Student A.I., allowing future developers and engineers to gain experience in the field of autonomous vehicles (IMechE, 2019). With the complexities of navigating public roads and traffic removed, the fundamental path planning problem is simplified. However, a different kind of complexity is introduced, as the limits of a vehicle's capabilities are explored at high speeds.

2.2 High-Level Approaches

This section will review common approaches to the path planning problem in a racing context. While the main focus of this project is the generation of a racing line, trends in autonomous racing such as receding horizons and optimal control are discussed. Advantages of these ideas may still apply to the narrower context of this system.

Particular attention will be given to constrained optimisation problems, that use some parametric function to represent the path, and produce an optimal path by applying continuous optimisation techniques (Paden et al., 2016). Constraints on the problem can then be introduced, such as track boundaries and the operational envelope of the vehicle. This method has two key benefits: the generation of smooth trajectories, and the potential inclusion of vehicle models to the optimisation (Schwarting, Alonso-Mora and Rus, 2018). However, such an approach may lead only to locally optimal solutions, and when formulated as a nonlinear problem can suffer from expensive computation time.

2.2.1 Two-Step Optimisation

The planning and control components of a software stack used for the high-speed autonomous vehicle *DevBot* are presented by Heilmeier et al. (2019). The work of Braghin et al. (2008) is used as a basis for their planning process, who describe a two-stage algorithm for generating a path. They define the optimal trajectory as the best compromise between the shortest path and the straightest path (i.e. the path of minimum curvature). The shortest and straightest paths are determined geometrically, and a vehicle dynamics model is then applied to appropriately weight the two paths and determine a speed profile.

Braghin et al. explain why the shortest vs. straightest compromise can lead to an optimal path. The maximum speed a vehicle can take around a corner, v_{max}, is limited by the radius, r, of the path taken:

$$v_{max} = \sqrt{\mu r(g + \frac{F_a}{m})} \implies v_{max} \propto \sqrt{r} \tag{2.1}$$

This property lends itself to the path of least curvature allowing the fastest speed around each corner. However, this maximum velocity may not be achievable for the entire track, due to the acceleration and top speed limits of the vehicle. In such cases, a shorter path with a greater curvature can be exe-

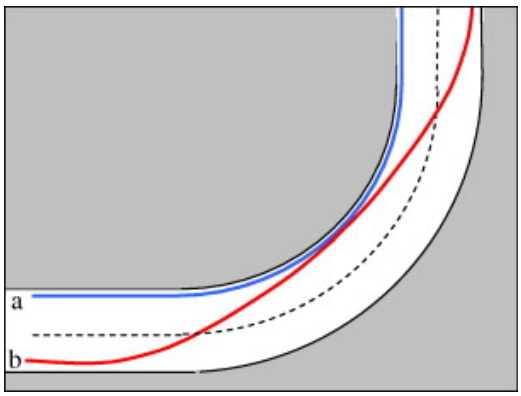


Figure 2.2: Comparison between the (a) shortest and (b) minimum curvature path through a corner (Braghin et al., 2008, p.1504).

cuted without sacrificing speed - further reducing the time taken around the corner. Applying a vehicle model to Equation 2.1 - with a friction coefficient, μ, downforce coefficient, F_a, and vehicle mass, m - this velocity limit can be determined for each corner to compute the path compromise. In Braghin et al.'s conclusions, the algorithm is described as "fast and reliable", although no computation times are presented or compared to other methods.

An extension of this method is implemented by Kapania, Subosits and Gerdes (2019), where the velocity profile is fed back into the optimisation process to refine the path geometry - forming an iterative process. In contrast to Braghin et al., only the path of minimum curvature is computed.

It is shown that optimising for minimum time results in "competing objectives" of minimising path length and maximising velocity. For example, increasing the radius of a turn allows greater speeds but also increases the path length. Kapania, Subosits and Gerdes claim that a nonlinear optimisation problem is required to manage this trade-off, which proves computationally expensive. By focusing on curvature minimisation, a relatively cheaper convex optimisation problem is proposed to further refine the path geometry. They acknowledge that minimising curvature and lap time are not equivalent, but show using both simulated and experimental results that similar performance can be achieved. The sensitivity of nonlinear approaches to initial conditions, and the resulting locally optimal solution, also favour a curvature minimisation objective.

Their approach has been validated experimentally, with predicted lap times for a 4.5km course converging in four or five iterations. The authors acknowledge that this method is sub-optimal relative to a nonlinear approach, but

demonstrate significant reductions in computation time. Kapania, Subosits and Gerdes profile their algorithm's run time in terms of 'lookahead' - the length of track considered during the optimisation. With solve times in the order of a few seconds for several hundred metres of track, the authors claim that their method could be implemented as an online (live) path planner, and encourages future work in doing so.

2.2.2 Receding Horizons

Gerdts et al. (2009) use the receding horizon technique to compute a reference trajectory through a test-course. The scenario is very similar to that being tackled in this project, with the circuit given by 'pylons' defining the left and right boundaries. After a slow reconnaissance lap to map the initially unknown course, a trajectory can be optimised.

A moving horizon technique is used to avoid difficult optimal control problems produced by long or complex circuits. Essentially, the problem is broken down into a sequence of easier, local problems, that can then be combined subject to continuity conditions. Gerdts et al. note that this technique reflects that of human drivers, who focus on short sections of a track independent of the global aim of achieving the quickest lap time.

When solved on a personal computer (2.66 GHz, 512 MB RAM), complete trajectories of tracks with varying length and complexity are computed between 4 and 12 minutes. It should be noted that these run times are for computing inputs for the vehicle, instead of just the trajectory generation that this project focuses on. This work demonstrates that a divide-and-conquer approach can be applied to path planning tasks to reduce the complexity of the overall problem.

2.2.3 Optimal Control

Kabzan et al. (2019) describe a combined approach to the trajectory planning and input control for their Formula Student A.I. entry, AMZ Driverless. Heilmeier et al. believe that the complexity of the integrated approach is to its disadvantage. The intricate parametrisation and implementation, combined with high computation times, inhibit the scalability of such methods. However, Kabzan et al. cite the direct consideration of vehicle limits when computing control inputs as a significant advantage over separate planning and control stages. As for reducing run time, a receding horizon approach was implemented to limit the scope of each computation. In practice, 90% of solve

times were faster than data sampling times, validating the on-line capability of the approach.

The effectiveness of AMZ's approach has been experimentally proven, with multiple overall victories at Formula Student events in Germany and Italy. While the advantages of this approach are acknowledged, tackling both planning and control simultaneously goes beyond the scope and resources of this project.

2.3 Path Representation

This section focuses on the geometry of path planning. Different representations for computed paths are compared, as are the optimisation problems that can be formulated for the given geometry.

2.3.1 Arcs and Clothoids

Many planning implementations use the concatenation of various components. For example, Rizano et al. (2013) use a basic description of a track as a series of 'turn sectors' and 'straight sectors'. Turns are defined as two concentric circle arcs, with straights between them formed simply as two parallel lines. While this representation is intuitive, it cannot handle complex designs such as corners with varying radius.

Funke et al. (2012) develop this idea with their 'three-phase' structure for defining a path around a corner, illustrated in Figure 2.3. Again, two straights are either side of a corner section. Instead of using an arc of constant radius to represent the entire turn, entry and exit clothoids are used to smooth the transition between the straights and a circular arc at the apex. Clothoids (also referred to as Euler spirals) have a linearly varying curvature and are often used for creating smooth transitions in road and railway design (Meek and Walton, 1992). The concatenation of straights, clothoids and arcs is referred to as a clothoid spline.

The starting positions, heading and length of each component become the parameters to be optimised when computing a racing line for a given corner. Funke et al. go on to describe the dependencies between these parameters and outline the algorithm for solving for a racing line.

There is no analysis of computation time given by the authors, and it should be noted that the description of this path design process assumes straights are

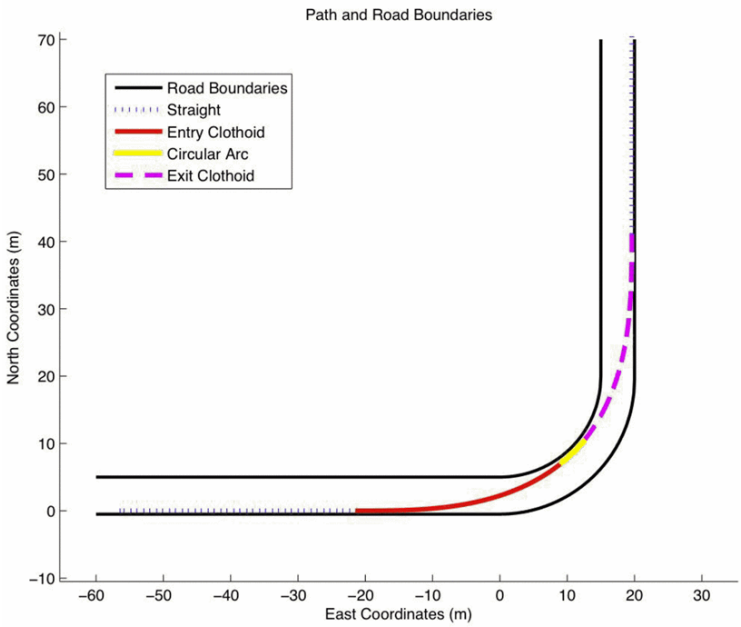


Figure 2.3: A racing line using the clothoid spline representation (Funke et al., 2012, p.545).

already known. The approach is presented as a pre-computed path, with vehicle control the only online component - which may hint at poor performance for computing in a live environment. However, paths around corners are generated independently. This allows for racing lines to be computed without the entire track known in advance.

Another issue with this representation is how it handles complex corner structures. Only a simple turn is demonstrated, which assumes two straights either end. It is not known how this representation will cope with varying radius corners, or chicanes that have multiple turns without separating straights.

2.3.2 Generating Splines from Control Points

The problem formulation by Braghin et al. (2008) involves discretisation of the track - segments are created and control points placed on the boundaries between them. These control points are then connected to form a path, as shown in Figure 2.4. The authors explain that cubic splines are used to avoid discontinuities in curvature.

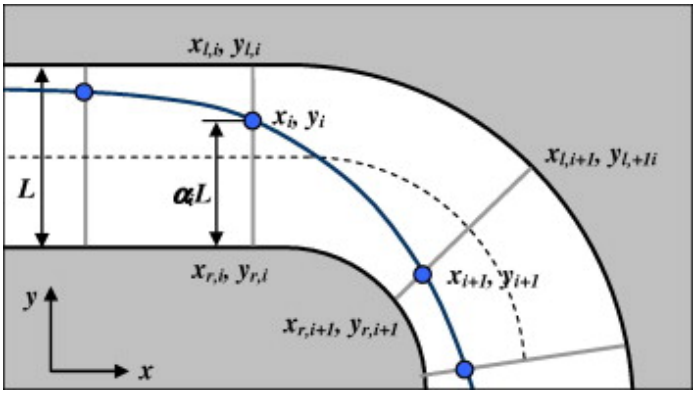


Figure 2.4: Control point and cubic spline path representation (Braghin et al., 2008, p.1504).

This could prove a useful representation in this project, as track boundaries are already defined by discrete cone locations. It also relates to the scenario Gerdts et al. worked with - pairs of opposite cones forming a series of gates around the track. Unfortunately, Braghin et al. do not give details on appropriate segment lengths for their discretisation of the track, only that fixed lengths are used. If a discrete model such as this is used, it would be interesting to investigate how this granularity affects performance, in terms of optimality and computation time.

2.3.3 Curvature Profiles

For the iterative two-step algorithm used by Kapania, Subosits and Gerdes, a different approach is taken. Their path is represented by three functions of the distance, s, along that path:

- Path curvature, $\kappa(s)$
- Lateral distance to inside boundary, $w_{in}(s)$
- Lateral distance to outside boundary, $w_{out}(s)$

This representation can then be translated to Cartesian coordinates post-optimisation. As discussed in Section 2.2.1, a curvature minimisation approach was used in this implementation. By representing a path in terms of this curvature it can be directly input and updated during the optimisation, avoiding the need for intermediate conversions. The same applies to the lateral distance functions, which are used for constraints on the problem to ensure track

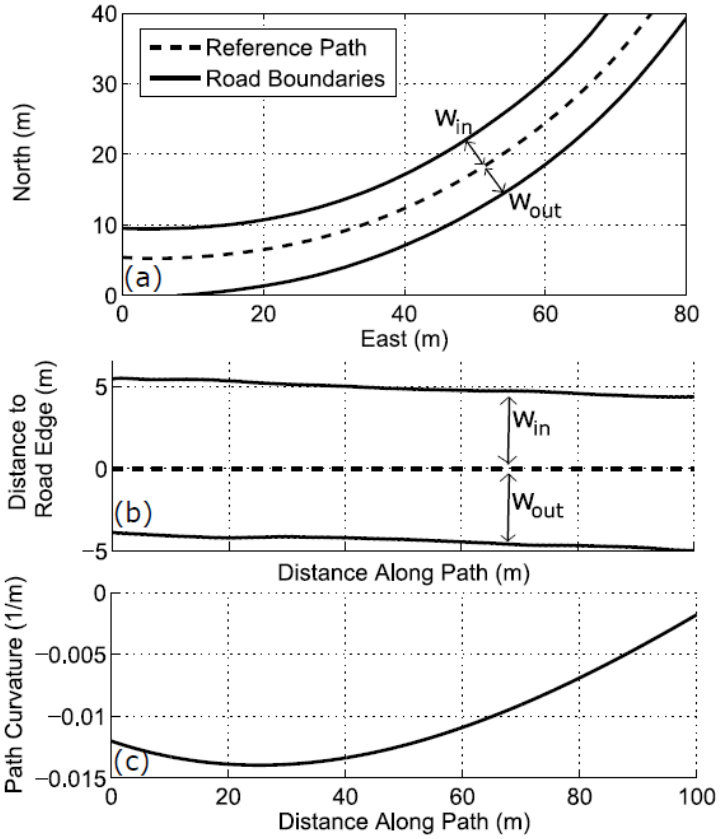


Figure 2.5: A lateral distance and curvature profile path representation from Kapania, Subosits and Gerdes (2019, p.3).

boundaries are not crossed. As a result, this representation is more efficient in terms of relating the output path to the optimisation problem.

There is no immediate discretisation of the track for this method. In practical testing, their system solved the curvature minimisation problem using 1843 'time steps' over a 4.5km circuit - averaging to a sample every 2.44 metres. If this representation were to be used, some pre-processing of the track would be required to establish appropriate sample points. Samples at each gate would need to be included to avoid contact with cones.

2.4 Generating Velocity Profiles

While the construction of a racing line is the main target for the project, generating a velocity profile and integrating vehicle dynamics as part of the optimisation will be important for both measuring and improving results. The combination of path geometry and velocity profile would form a reference trajectory that can be passed to the control stage of the pipeline.

The generation of velocity profiles by Kapania, Subosits and Gerdes is based on the work of Velenis and Tsiotras (2008). Optimal profiles are generated for a vehicle with limits on acceleration, deceleration, longitudinal forces and lateral (centripetal) forces. The constraints on these forces are given by a friction circle model, using tyre-road friction coefficients to determine the capabilities of the vehicle (Subosits and Gerdes, 2015). From these constraints, an optimisation problem is formed that maximises velocity along the generated path.

A 'three-pass' process is defined to generate the profile, as described by Kapania, Subosits and Gerdes (2019, p.3):

1. **First pass.** Find the maximum theoretical velocity at any point of the circuit, based on curvature, force limits and top speed.
2. **Forward integration.** Limit increases in velocity to the vehicle's maximum acceleration.
3. **Backward integration.** Limit decreases in velocity to the vehicle's maximum deceleration.

While the method is relatively intuitive, modelling the vehicle to obtain force constraints can quickly complicate the process. Tyre slip, aerodynamics, weight transfer and three-dimensional track properties such as incline and camber can all contribute to the constraints on velocity. While a fundamental model can be implemented for use in this project, extensions and complications need to be supported for future improvements in accuracy.

2.5 Mathematical Optimisation

Optimisation is a key component to computing a racing line. Much of the work covered in Section 2.2 looks to establish an appropriate objective function to minimise, such as estimated lap time or path curvature. An understanding of optimisation problems and how they are solved is required before developing a process using them. Snyman (2005) summarises optimisation as formulating and solving problems of the form

$$\underset{\text{w.r.t } \mathbf{x}}{\text{minimise}} \; f(\mathbf{x}), \quad \mathbf{x} = [x_1, \ldots, x_n]^T \in \mathbb{R} \tag{2.2}$$

subject to constraints

$$g_j(\mathbf{x}) \leq 0, \quad j = 1, \ldots, m \tag{2.3}$$
$$h_j(\mathbf{x}) = 0, \quad j = 1, \ldots, r \tag{2.4}$$

These are referred to as *constrained minimisation problems*, in which values of $\mathbf{x}$ that minimise the objective function $f(\mathbf{x})$ (while satisfying the constraints) are found.

Gradient-descent methods are the most common approach to these problems, often computationally less-expensive than evolutionary methods such as genetic algorithms (Snyman, 2005). *Newton* methods compute a search direction using the *Hessian matrix*, $\mathrm{H}(\mathbf{x})$, made up of partial second derivatives of the objective function. Cottle and Thapa (2017) explain that at every iteration, the Hessian needs updating using a finite-differences approximation and an $n \times n$ system is solved for the search direction. This approach scales poorly for problems with many variables, which saw the development of *Quasi-Newton* methods. The evaluations of the objective function $f(\mathbf{x})$ and its gradient $g(\mathbf{x})$ are used to approximate the Hessian. This reduces the computational load while retaining the quadratic convergence of Newton methods.

The *BFGS* algorithm for updating the Hessian is singled out by both Snyman and Cottle and Thapa as the most successful on practical problems. The algorithm has variants for reducing memory footprint (*Limited BFGS*) and support for box constraints, where each variable has a lower and upper bound. These two extensions form the *L-BFGS-B* algorithm for constrained minimisation problems. The box constraints are sufficient to model the track boundaries, so this algorithm is a viable candidate for the racing line optimisation.

The number of variables, n, is a crucial factor in the complexity of solving these problems. As seen in literature, solve times increased dramatically for

longer tracks. Kapania, Subosits and Gerdes (2019) also discussed the idea of ill-conditioned problems, particularly in regards to using the estimated lap time as the objective function to minimise. They explain that minimising lap time requires minimising both path length and curvature - as per Braghin et al. (2008)'s arguments. These are competing objectives, complicating the optimisation and slowing convergence. This was the main argument for using simpler objectives such as path curvature to reduce computation time.

2.6 Summary

The review of literature has covered the core components for a global path planning method. The two-step methods of Braghin et al. (2008) and Kapania, Subosits and Gerdes (2019) have been identified as suitable approaches for this project. The somewhat modular system of starting with path geometry, generating a velocity profile, then potentially reintegrating vehicle dynamics allows a flexible, iterative approach to the research and development of a working system. It is acknowledged that there has been a trend towards optimal control systems, but a separate approach will be used to limit the scope and complexity of this project. Receding horizon implementations are very common in the reviewed literature, highlighting the potential for a divide-and-conquer approach to the optimisation problem. While some compromise on optimality is required, the improvement in computation time strongly supports the use of this tactic.

As observed in the review of curve representations, the track can be split into a sequence of corners and straights, with a locally optimal path through each corner computed. Concerns were raised with how segmented representations - such as Funke et al. (2012)'s three-phase structure - handle complex corner sequences. The more flexible approaches of Braghin et al. (2008) or Kapania, Subosits and Gerdes (2019) may be more suitable. This does not prevent a divide-and-conquer method; complex corners or corner sequences can still be optimised in isolation between straights of adequate length. The detection of straights on a track will pose a new problem, as this information was assumed to be known in the reviewed algorithms.

An intuitive approach to velocity profile generation has been offered by Kapania, Subosits and Gerdes (2019), and it is hoped it can be used to output a full reference trajectory and improve the optimisation of a racing line. A suitable optimisation algorithm has been identified for prototyping, and considerations for its use include problem size and complexity of the objective function.

Chapter 3

Model Development

Initial development aims to formulate the optimisation problem and construct models for representing the trajectory and vehicle dynamics. What will result is a proof of concept, covering the fundamental requirements for the project. The effectiveness of methods can then be evaluated using this prototype, along with early analysis of performance, and ideas for the final program structure.

Python is a popular choice for quick prototyping and has libraries available with well-performing implementations of algorithms. Specifically, `scipy`'s `optimize` package provides functionality to solve the optimisation problems that the prototyping phase aims to formulate and verify. In addition, `numpy` presents a useful interface for array arithmetic, that will be required throughout the project.

This chapter will give a chronological discussion of how the model has been developed. Candidate objective functions are introduced throughout the chapter, before being summarised in Section 3.4. Findings from this prototyping phase, implications for the project, and lines of investigation for further development are also covered in this closing discussion.

3.1 Path Geometry

The starting point for the generated trajectory is to optimise the geometry of the path. This will form the basis of the trajectory, from which dynamic properties are derived and used to refine the path itself.

3.1.1 Problem Formulation

The first component of this optimisation problem is how to represent the path. The problem input is a series of left-right cone locations, so there is an inherent segmentation of the track to be exploited.

With this in mind, the control point and spline model used by Braghin et al. (2008) offers a useful definition of the path in relation to the cones. Left-right cone pairs form a 'gate', along which the control point is positioned according to a parameter, α. A cubic spline is generated through these control points to form a smooth path. The `scipy.interpolate` package provides a method of constructing splines from these control points, with respect to some parameter. In this case, straight-line distances between control points are used, as an approximation of path distance, s.

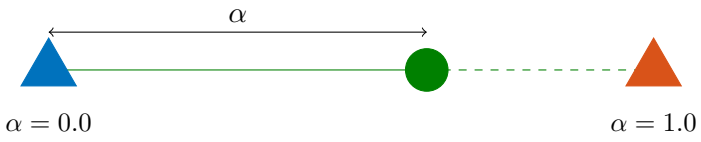


Figure 3.1: Control point positioning according to parameter α.

A closed-loop circuit will be assumed, and the need to minimise lap time over multiple laps considered. Therefore, the first and last control points will be identical and have a shared α value. The resulting spline will need to be periodic, to ensure a continuous curve from the end of one lap to the start of the next. Of course, optimising for a single lap could be supported by relaxing these constraints and allowing a different start and finish position.

With a path representation decided, the optimisation problem can be formed. The entire path can be defined by just the cone locations and α values for each control point. Therefore, it is the assignment of these α values that are to be optimised. The bounds for α values will be determined by the width of the vehicle, to avoid contact with cones - assuming the middle of the car is aligned with the path. For now, a simple $0 \leq \alpha \leq 1$ box bound is used.

A measure of optimality also needs to be defined. In Chapter 2, path curvature was identified as a strong candidate for a purely geometrical metric to minimise. Results from Kapania, Subosits and Gerdes (2019) showed that minimising curvature achieves lap times close to a direct minimisation of lap time. Braghin et al. calculates a curvature metric Γ^2 using Equation 3.1. The curvature, $\kappa(s)$, at n_s sample points around the track are squared and summed to quantify the overall curvature of the path. Note that the number of sample points, n_s, is independent of the number of control points, n.

$$\Gamma^2 = \sum_{i=1}^{n_s} \kappa_i(s)^2 = \sum_{i=1}^{n_s} x_i''(s)^2 + y_i''(s)^2 \tag{3.1}$$

The cubic spline representation of the path ensures that the second derivatives exist for the curvature metric given in Equation 3.1. This metric is used to avoid the need to find a square root when calculating κ_i. With all of these components, the curvature minimisation problem can be described.

$$\begin{aligned} &\text{Minimise} && \Gamma^2(\alpha) && && \text{(3.2a)} \\ &\text{Subject to} && 0 \leq \alpha_i \leq 1 && \text{for } i = 1, \ldots, n && \text{(3.2b)} \end{aligned}$$

3.1.2 Path Generation

The `scipy.optimize` package offers an intuitive L-BFGS-B implementation for solving the formulated optimisation problem. See Appendix A.1 for the relevant code. Plotted results for a sample track are given in Figure 3.2.

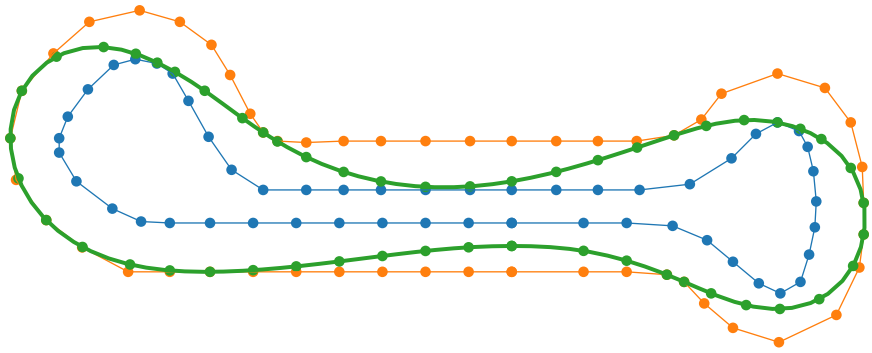


Figure 3.2: Prototype minimum curvature path (anticlockwise).

On first viewing, the path matches some key traits of a racing line: corner apexes are met, and chicane sequences are negotiated with straight paths. However, there are some questionable characteristics to investigate further. For example, the path around the long corner on the left-hand side follows the outside boundary for a significant distance. It is hard to tell in this case if the distance-curvature trade-off here translates to a better racing line.

This scenario is exaggerated in Figure 3.3, which shows the planned path around an isolated hairpin turn. The solid green path minimises total curvature by maintaining a straight line deep into the corner and following the outside boundary. The dashed line is the same path, shifted so that the inside apex is met. What results is a path of identical curvature with a shorter distance covered, indicating that minimising total curvature is sub-optimal. This reflects the comments of Braghin et al. regarding the need for compromise between path distance and curvature.

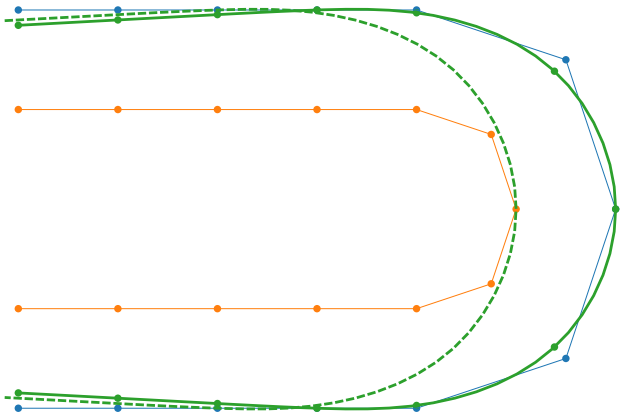


Figure 3.3: Path of minimum curvature around a hairpin, compared with an expected racing line (dashed).

3.2 Vehicle Dynamics

With paths now being generated, a measure of performance is needed - namely a predicted lap time. This ties in with the velocity profile requirement of the output trajectory and requires the dynamic properties of the vehicle to be simulated. First, a method for generating velocity profiles is proposed. From this process, the required properties of the vehicle are identified and modelled.

In Section 2.4, a three-stage process from Kapania, Subosits and Gerdes (2019) is described for generating a velocity profile. The following three sections discuss the modelling and implementation of these three steps. The technical report from Khan (2017) has provided a useful reference when building up a vehicle model for this process. Vehicle parameters for Team Bath Racing's 2018 entry to Formula Student have been taken from this report for testing. Appendix A.2 contains the Python code implementing the three-pass process described.

3.2.1 Pass 1: Limiting Local Velocities

The purpose of the first pass is to set a theoretical maximum velocity for each sample of the path. The capabilities of the vehicle's tyres are key to this limit. Essentially, the centripetal force generated by the vehicle as it corners needs to be balanced with the frictional force available from the tyres - as given in Equation 3.3.

$$F_{lat} = F_{friction} \quad \rightarrow \quad \frac{mv^2}{r} = \mu mg \quad \rightarrow \quad v = \sqrt{r\mu g} \tag{3.3}$$

The cornering radius is easily calculated as the reciprocal of curvature: $r = 1/\kappa$. With mass cancelled out, the only vehicle property required is a coefficient of friction, μ, between the road and tyres. This value gives a ratio from normal force (i.e. the weight of the vehicle through the tyres) to frictional force. Putting this all together creates a map from a sample's curvature to a local velocity limit. This model could be easily extended with further dynamic effects. For example, aerodynamic downforce adding to the normal force through the tyres, increasing grip and allowing greater cornering speeds.

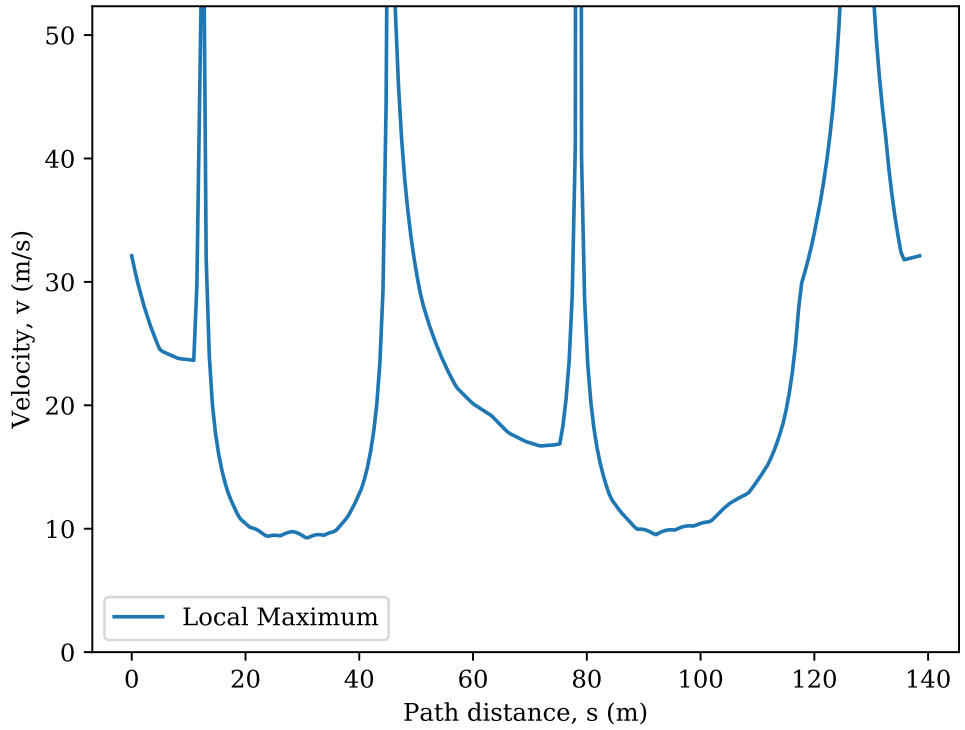


Figure 3.4: Plot of local maximum velocities.

3.2.2 Pass 2: Limiting Acceleration

With maximum local velocities computed, changes between local velocities must be constrained by the performance envelope of the vehicle. For the forward pass, the velocity at path distance $s+\Delta s$ is limited by the acceleration from point s, according to Equation 3.4.

$$v_{s+\Delta s} = \min\{v_{s+\Delta s}, \sqrt{v_s^2 + 2a_s\Delta s}\} \tag{3.4}$$

There are two key factors in computing acceleration at a given point, a_s: driving force from the engine, and available traction from the tyres. First, the force provided by the drivetrain gives an upper bound on acceleration. Drivetrains are often profiled with torque curves, defining the maximum rotational force they output as a function of engine speed (Zhang and Mi, 2011). These can be translated to give the maximum accelerating force, F_{out}, as a function of velocity. However, crucially, the delivery of this force is limited by the available longitudinal traction between the road and tyres. If the vehicle is cornering, some of the grip is 'used up' by the resulting lateral force through the tyre.

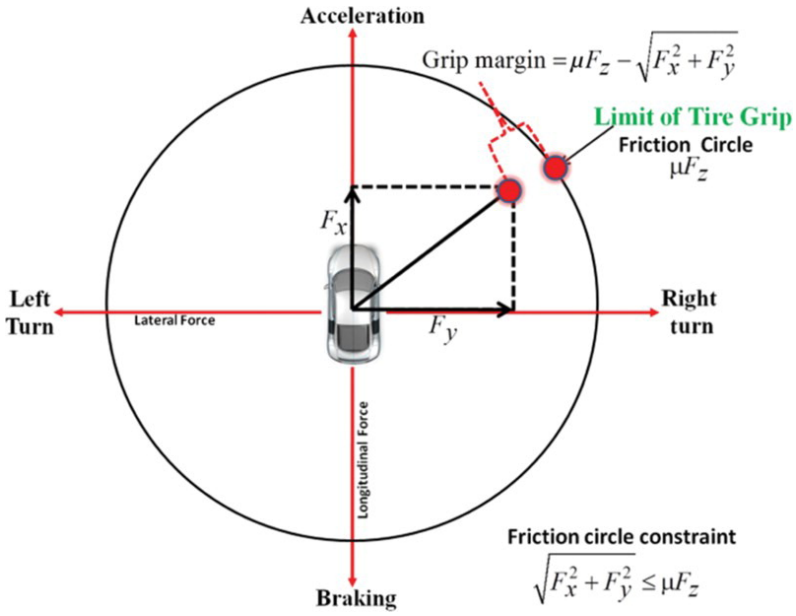


Figure 3.5: Friction circle model (Singh and Taheri, 2015).

The friction circle is a model that describes the combination of lateral and longitudinal force through the tyre, limited by the available frictional force. The available longitudinal force can be calculated from the *grip margin*, as shown in Figure 3.5. The two limits on acceleration force are now defined, and the acceleration derived using Equation 3.5. The iterative process described by Equation 3.4 begins from the slowest local velocity, passing through the current profile to limit increases in velocity.

$$a_s = \frac{\min\{F_{out}, F_{long}\}}{m} \tag{3.5}$$

3.2.3 Pass 3: Limiting Deceleration

For the final pass in limiting deceleration, a similar principle is applied while moving backwards through the velocity profile.

$$v_{s-\Delta s} = \min\{v_{s-\Delta s}, \sqrt{v_s^2 - 2d_s\Delta s}\} \tag{3.6}$$

Again, the available longitudinal force limits the possible deceleration. This mimics a driving technique known as *threshold braking*, in which brake pressure is applied such that the braking force matches the available traction in the tyre (Hoad, 2016; SGI, 2019). The application of these three passes results in a velocity profile for a given path and vehicle. Figure 3.6 illustrates a breakdown of the three passes and the final profile. The acceleration and deceleration passes were run independently, and the minimum of the two taken for the final velocity profile.

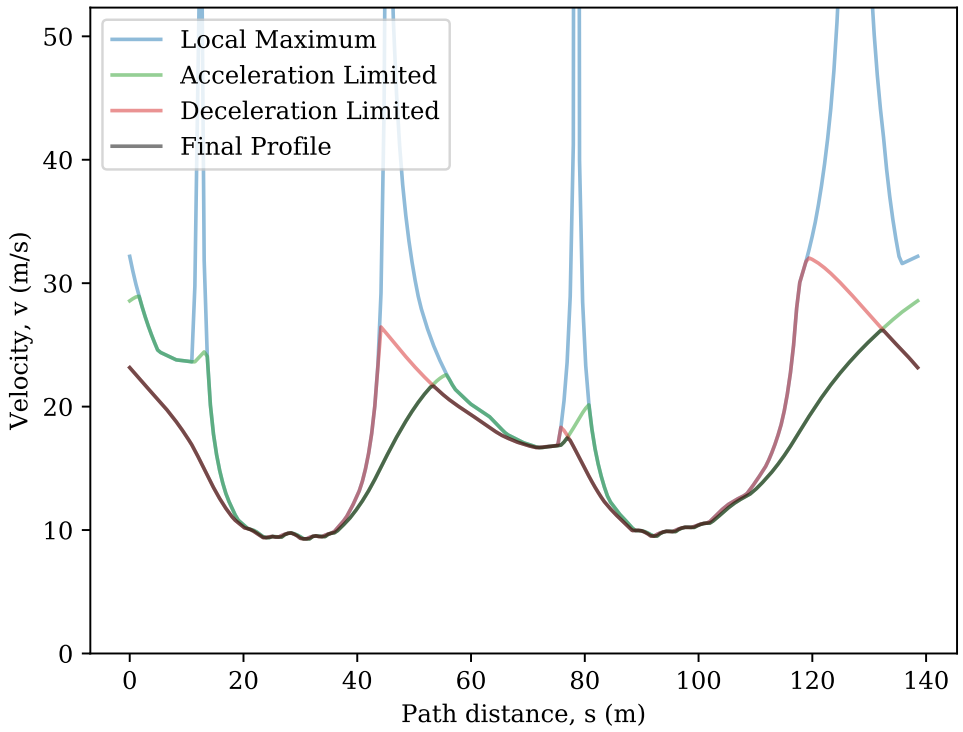


Figure 3.6: Prototype velocity profile.

While a profile is successfully produced, there are some artefacts of the process evident. For example, at local minima, an unstable velocity profile is observed. There is a high sensitivity between the path distance and curvature at tight

corners that needs to be addressed. It would also be expected to see gradual acceleration when exiting slow corners in an ideal trajectory. The initial pass maximises velocity by exploiting all available grip, with acceleration in the second pass allowed only when the traction to do so is available. If the maximum local velocities are achievable by the vehicle, there is no remaining grip to accelerate with and the velocity stalls. What results is a cycle of no traction, constant velocity, traction becoming available, and acceleration to the local velocity limit. Some smoothing process is required to produce a realistic velocity profile for the trajectory. This process would need to respect the upper bounds on the velocity that have been originally computed.

Figure 3.7 shows the complete trajectory, with a colour gradient used to represent the velocity along the path. This format will be used going forward to illustrate the results of trajectory generations.

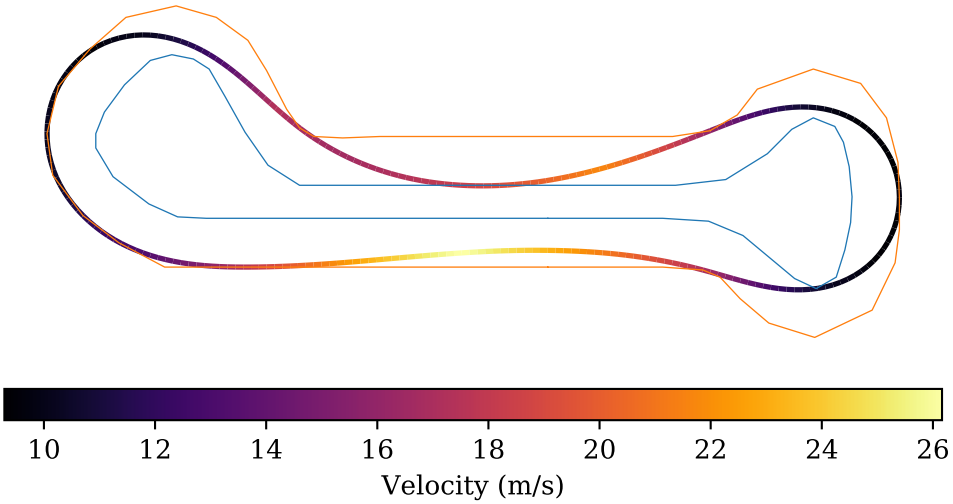


Figure 3.7: Prototype path with velocity gradient.

3.3 Minimising Lap Time

Using the path geometry and velocity profile constructed, the performance of a trajectory can now be measured in terms of lap time. This is computed using the fundamental `speed = distance / time` relationship over each sample of the trajectory.

With a performance measure defined, different methods for constructing and refining the trajectory can be evaluated. An obvious candidate is to directly minimise the resulting lap time. The objective function updates the path according to the given α values, generates the velocity profile, and returns the simulated lap time. As shown in Table 3.1, this achieves a significant improvement in lap time over the curvature minimisation technique used previously. However, the computation time is severely impacted - nearly a 17-fold increase over the minimum curvature problem. (Testing machine: 3.90GHz CPU, 8GB RAM.)

Strategy	Lap Time (s)	Run Time (s)
Minimise lap time	9.170	72.28
Minimise curvature	9.742	4.26

Table 3.1: Lap time and curvature minimisation performance.

This stems from the need to generate a velocity profile during each call of the objective function by `scipy.optimize.minimize`. In this particular run there were 15,080 callbacks during the optimisation, so reducing workload in the objective function is crucial for reducing run times. With this in mind, the compromise approach is considered. Braghin et al. (2008) define a new objective function, repeated here in Equation 3.7.

$$F_\varepsilon = (1 - \varepsilon) \cdot \Gamma^2 + \varepsilon \cdot S \tag{3.7}$$

A linear relationship between the curvature metric Γ^2 and path length S is defined by the parameter ε. Computational effort in the objective function is comparable to that of the pure minimisation of Γ^2, as the path length is readily available when the path is updated. A new optimisation problem is formed: find the value of ε that will produce the shortest lap time when F_ε is minimised. Figure 3.8 plots the resulting lap time when using an objective function of varying weights. The `scipy` function `minimize_scalar` was used to perform this search (see Appendix A.3). Each data point in the plot reflects an evaluation of the objective function by `minimize_scalar`.

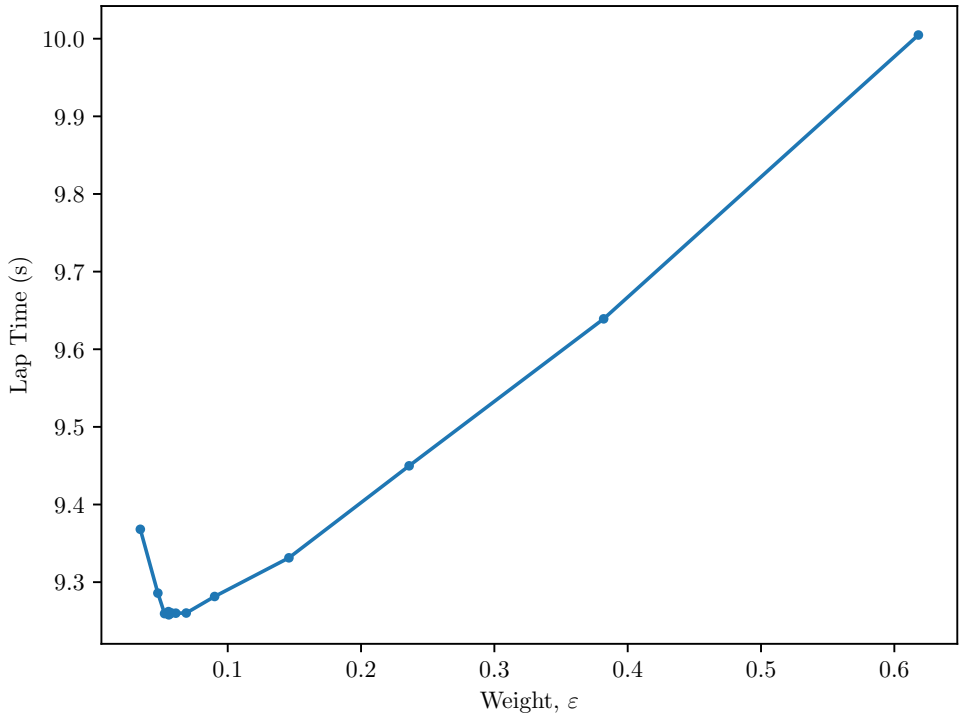


Figure 3.8: Effect of curvature-length weight on the resulting lap time.

This optimisation outputs an optimal weight $\varepsilon^* = 0.056$. This defines the best-performing compromise between minimum curvature and minimum path length for the sample track. Optimising α values for F_{ε^*} forms the trajectory illustrated in Figure 3.9.

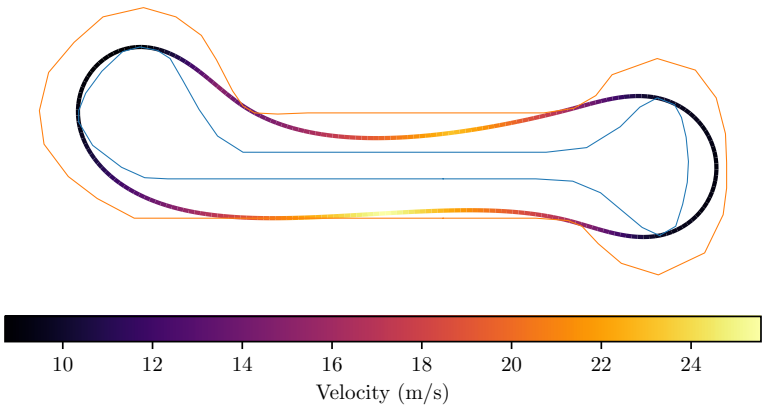


Figure 3.9: Optimal curvature-length compromise path with velocity gradient.

The sub-optimal characteristics discussed in Section 3.1.2 appear to have been corrected with this new strategy. The far-left corner in the sample track is negotiated with a much tighter line - reducing the distance covered for a small trade-off in velocity. Repeating the extreme example of an isolated hairpin turn in Figure 3.10, the path predicted earlier has been produced.

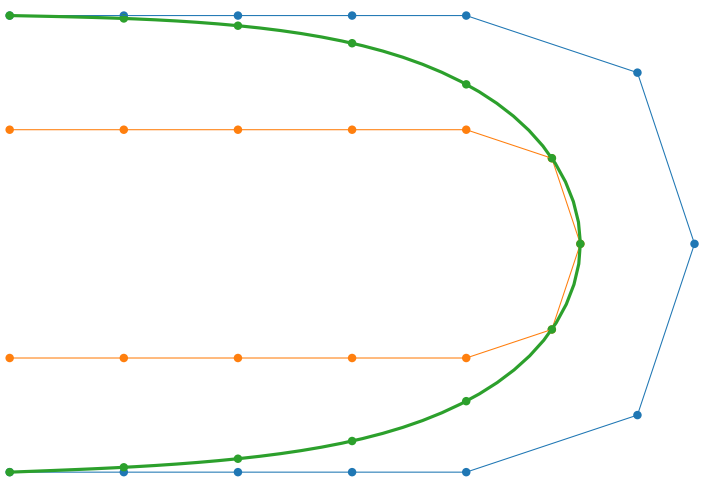


Figure 3.10: Path around a hairpin using the compromise generation method.

Table 3.2 compares the use of the compromise objective F_ε to the direct minimisation of lap time. The two lap times are very close - the compromise method achieving a 101.96% lap time compared to the optimal time. However, the computation time for finding an optimal compromise is disappointing - significantly slower than finding the minimum lap time.

Strategy	Evaluations	Lap Time (s)	Run Time (s)
Minimise lap time	-	9.170	72.28
Minimise curvature ($\varepsilon = 0$)	1	9.742	4.26
Optimise ε	24	9.258	93.65
Uniform search for ε	51	9.261	96.12
Uniform search for ε	11	9.293	19.59

Table 3.2: Optimal compromise performance comparison. *Evaluations* indicates the number of times a minimisation of F_ε was performed to retrieve a lap time.

During `scipy.optimize.minimize_scalar`'s search for an optimal weight, the second-level minimisation of F_ε was conducted 24 times. The uniform search tests weights at regular intervals and takes the best performing value. By avoiding the second optimisation of ε, many more evaluations can be conducted within the same time. The resolution of the final ε value depends on the search density, but there doesn't appear to a significant increase in lap time. This indicates that the search for a suitable curvature-length does not need to be incredibly precise to reduce lap time.

3.4 Implications

3.4.1 Candidate Objective Functions

The primary task during prototyping was to establish the optimisation problem that results in a well-performing trajectory. Three key candidates were identified, and from now on will be referred to as follows:

Curvature Method
Minimise the total curvature metric, Γ^2.

Compromise Method
Minimise the curvature-length compromise, F_ε, for some optimal weight ε.

Direct Method
Directly minimise lap time, T.

Minimising F_ε and T were the best performing methods in terms of the final lap time but were very slow in comparison to the Γ^2 minimisation. Further development should focus on reducing computation time to make the use of these two methods more feasible.

The additional computation time for the compromise method comes from the optimisation of ε, that weights the importance of minimising curvature versus path length. The use of an 'outer' optimisation of ε significantly impacted run time, with more coarse searches of potential curvature-length weights showing promising performance. A uniform search such as this would be easily parallelised, and a suitable precision could be achieved given enough parallel evaluations of F_ε.

Results from prototyping, and Braghin et al.'s results, indicate that weighting in favour of curvature is desirable. As such, the search space could be dramatically reduced to further improve the computation time and/or precision of ε.

More tracks need to be tested to identify a suitable range for ε. It could be that a default weight is used to produce a well-performing trajectory, which the vehicle could use while a full search is conducted.

Generating the velocity profile on each call of the objective function is the clear factor in long run times for direct lap time minimisation. The resulting lap time is ideal, so reducing the computational load of this process is key. It would be preferable to use this approach, but it remains to be seen whether the compromise method allows for more significant improvements in run time.

3.4.2 Limitations

To conclude the discussion of the prototyping phase, the limitations and simplifications so far are acknowledged. The dynamics model is a basic representation of the vehicle used to map from the path geometry to a velocity profile. In-depth vehicle dynamics models are considered out of scope for this project, but it is important to consider potential expansion of such models for the process being developed. The current method of constructing a velocity profile supports any additional modelling, so long as a local velocity can be computed for a given sample point on the track, and velocities are only dependent on neighbouring samples (i.e. acceleration and deceleration from one to the next). Therefore, the process will support more accurate models that meet these requirements.

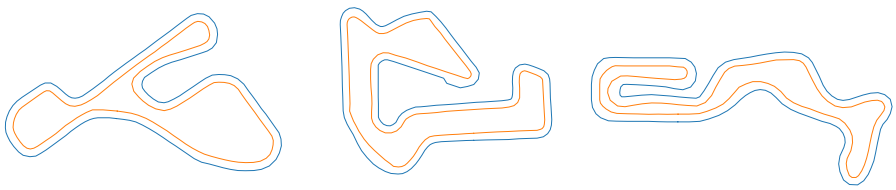
Performance of the methods developed in this project is also reliant on the optimisation functions used. The use of SciPy's `optimize` package has streamlined the development of a working model. No gradient information needs to be supplied to its L-BFGS-B implementation, reducing the complexity of the prototype. While improvements in the overall method are sought, it is acknowledged that careful selection and tuning of optimisation algorithms and their implementations also make a difference in the overall success of the method.

Chapter 4

Further Development & Analysis

This chapter contains a series of investigations into the performance of the working model with real-world circuits. These circuits are much longer than the sample track used for short tests on the prototype, with more gates and sample points increasing the difficulty of the problem.

Two improvements are considered to the overall method to reduce run time with minimal impact on the resulting lap time. A divide-and-conquer approach is explored in which individual sections of tracks are optimised and the resulting path combined. The search for an appropriate length-curvature compromise is also improved by investigating a relationship between track properties and the optimal weight.



(a) Clay Pigeon Raceway. (b) Whilton Mill. (c) Buckmore Park.

Figure 4.1: Real-world circuits used for testing.

4.1 Current Performance

To begin the analysis of the current working model, the sample track used during prototyping is replaced with a real, full-length circuit. Karting circuit *Clay Pigeon Raceway* (Figure 4.1a) has been selected for the variety of corner types, moderate length, and appropriate width for the Formula Student vehicle. This track has been modelled with 138 cone pairs, as opposed to the 52 pairs in the prototyping track. Each of the three candidate objective functions has been tested, with their performance summarised in Table 4.1.

Method	Lap Time (s)	Run Time (s)
Curvature	33.920	4.97
Compromise	32.064	110.47
Direct	35.305	153.44

Table 4.1: Performance of the three candidate methods for Clay Pigeon.

4.1.1 Direct Method Feasibility

The direct minimisation of lap time failed to converge to a solution for the longer circuit. The optimisation was halted as the number of objective function evaluations exceeded the default limit of 15,000. At this point, the path had not deviated far from the initial centerline path ($\alpha = 0.5$). This limit could be extended, but with a run time over 2.5 minutes, the feasibility of the direct method is already questionable.

Difficulty directly minimising lap time corresponds with the findings of Kapania, Subosits and Gerdes (2019) regarding ill-conditioned optimisation problems (see Section 2.5). Poor performance from the direct method cements the need for a faster method that approximates an optimal solution.

L-BFGS-B Convergence

SciPy's `minimize(method='L-BFGS-B')` implementation (Jones et al., 2001–) uses the underlying work of Zhu et al. (1997). In their description of L-BFGS-B, Zhu et al. note that the algorithm is not quick to converge for difficult problems. All three results in Table 4.1 were reached at the 15,000 evaluation limit. Within this limit the lap time minimisation could not converge, indicating an ill-conditioned problem for the L-BFGS-B method. This difficulty likely increases with the number of control points. As well as non-convergence,

the run time is poor due to the computational load in calculating a lap time with every evaluation. To continue with this algorithm, the difficulty of the problem needs to be reduced and the complexity of the objective function kept to a minimum.

4.1.2 Compromise Analysis

The compromise method successfully improved on the lap time of a minimum curvature approach. As before, the search for an appropriate compromise weight impacted run time. Figure 4.2 illustrates this trajectory.

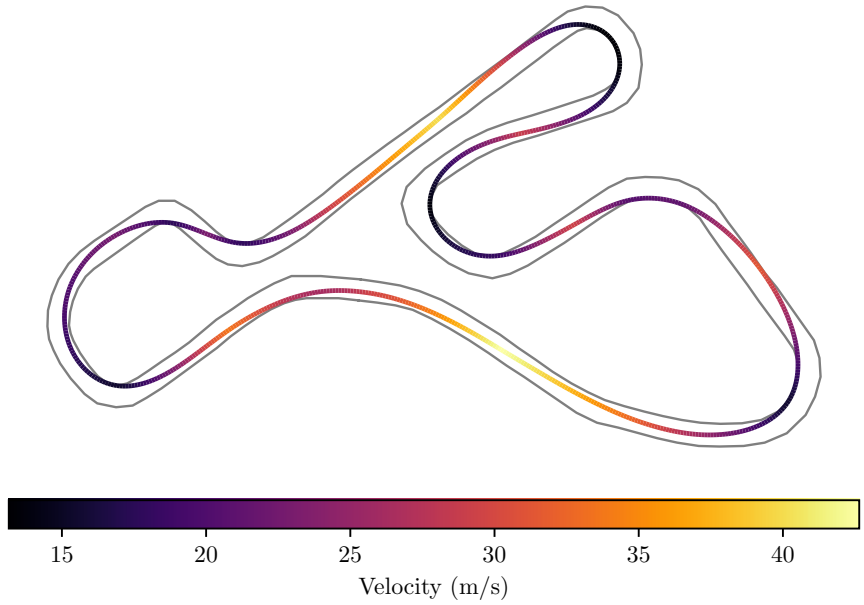


Figure 4.2: Clay Pigeon trajectory (clockwise), minimising F_{ε^*}.

For this test, an optimal weight of $\varepsilon^* = 0.0056$ was identified. This emphasises how even a tiny consideration for path length in the optimisation can improve lap time significantly. Figure 4.3 plots the resulting lap time for each F_ε tested in another run, this time with bounds of $0 \leq \varepsilon \leq 0.01$ set. Tiny variations in ε can impact the lap time by tens of milliseconds - not an insignificant difference in competitive motorsport. It is therefore critical to identify an appropriate ε for each track.

The optimal compromise has been found for further test circuits, with ε^* ranging by an order of magnitude from 0.0056 (Clay Pigeon) to 0.056 (prototyping circuit). A unique weight is required for different tracks, each with varying

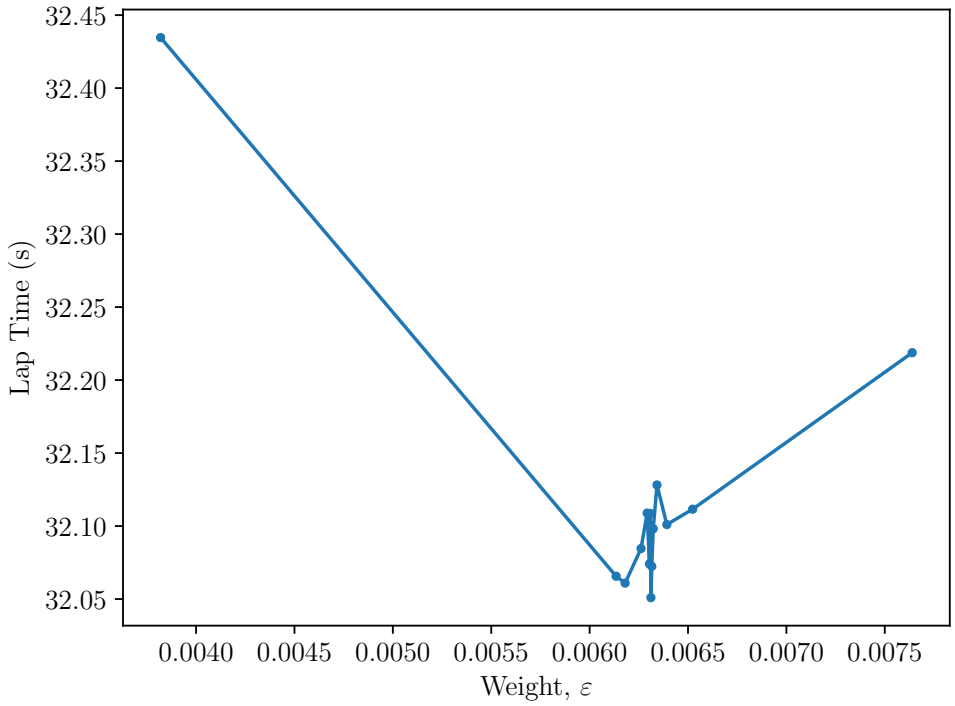


Figure 4.3: Search history for Clay Pigeon's optimal weight.

properties: length, corner types, corner curvatures, number of corners and straights. It is then sensible to investigate whether the compromise weight should be set on a corner-by-corner basis, instead of a single weight for the entire track. If so, being able to construct a trajectory from separately optimised corners will have a two-fold benefit.

On top of a localised optimisation of ε, there is an opportunity for a divide-and-conquer approach to parallelising the process by optimising each corner simultaneously. This relates to the concept of receding horizons discussed in Section 2.2.2, in which the scope of optimisation problems are reduced to improve run time. Section 4.2 investigates the potential of this approach.

4.2 Sector Optimisation

The proposed technique divides the track into several overlapping 'sectors', which will be optimised individually using the compromise method. A sector will typically consist of two straights, with a corner or sequence of corners in rapid succession between them. Adjacent sectors share a straight, along which the resulting paths will be merged. This process is implemented in Appendix A.4.

This technique mimics that of a racing driver developing their racing line for an unfamiliar circuit. The best lines through each corner are established and stitched together by moving from the exit point of one corner to the entry point for the next along the intervening straights.

4.2.1 Path Generation

The first task is to identify corners on the track. A simple curvature threshold has been used initially, to mark samples of the track's centerline as a corner, or straight. Straight regions below a certain length are converted to corners, to merge corners that are close enough to significantly influence each other's racing line. Very small corner regions are filtered out to give a set of significant corner regions.

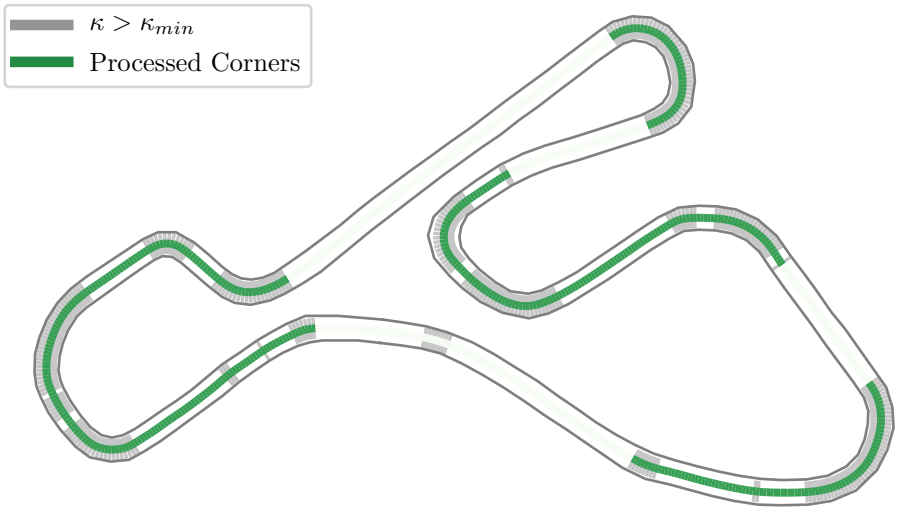


Figure 4.4: Corner sequences identified for Clay Pigeon. $\kappa_{min} = 0.02$. Corners within 40m are merged; corners less than 10m long are then suppressed.

The resulting corner sequences are sensitive to three parameters.

Curvature Threshold
This parameter is the basis for corner detection. Too high a threshold and some corners will not have a bespoke ε optimised for their path. Too low a threshold and few straights will remain to merge the paths of adjacent sectors.

Corner Proximity
Drivers often identify close sequences of corners when deciding a racing line, as the path through one corner will influence the next. A short straight between two corners may not give enough distance to move from the exit point to the next entry point without significant loss of speed.

Corner Length
Very short corners will not have individual optimisations, to reduce the computational load on the process. They will still be considered in the overlapping regions of sectors, but will not have a bespoke compromise weight determined.

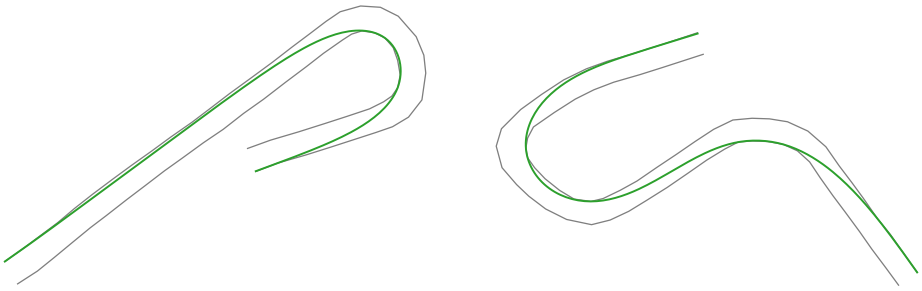
For each corner sequence, a new open-ended `Track` is created using the cones from the end of the previous corner to the start of the next. This forms the straight-corner-straight sectors to be optimised independently. The familiar process of finding an optimal ε for minimising F_ε is applied to each sector.

Sector	Control Points	Optimisation Time (s)	ε^*
All	*138*	*110.47*	*0.0056*
1	34	45.62	0.0192
2	37	44.56	0.0046
3	33	24.37	0.0296
4	75	67.96	0.0043

Table 4.2: Sector optimisations for Clay Pigeon.

The presence of overlapping sectors sees the total computation time increase from 110 seconds to 182 seconds. However, by optimising each sector concurrently the run time was reduced to 68 seconds - effectively reducing to the longest sector's solve time. Of course, this requires an adequate number of CPU cores for all sectors to be optimised within the worst-case run time.

Each sector had different values for ε^*, as listed in Table 4.2. This demonstrates that using a single compromise weight for the entire track is not ideal. However, this produces disjoint paths - Figure 4.5 shows consecutive sectors with different paths on their shared straight. The combination of sectors inevitably requires some compromise on these shared straights.



(a) Sector 1. (b) Sector 2.

Figure 4.5: Optimised sector paths for Clay Pigeon.

To merge neighbouring paths weighted averages of overlapping α values are calculated, with weights transitioning linearly from 100% Sector 1's path at the start to 100% Sector 2's path at the end. This forms a linear transition from one sector's path to the next. With a cubic spline generated through the resulting control points, a smooth path is still constructed.

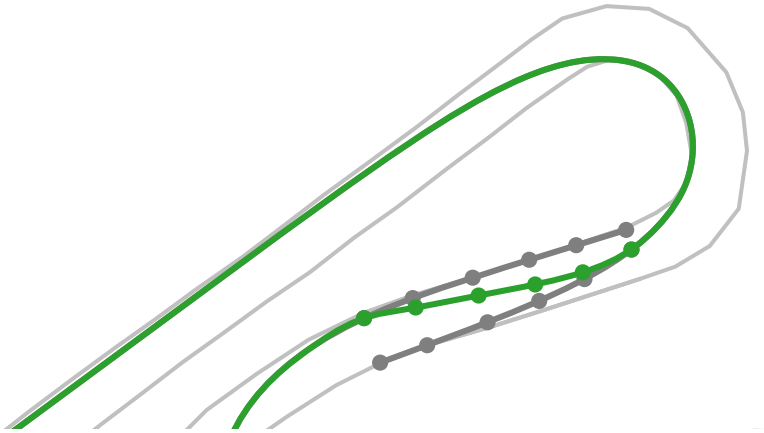


Figure 4.6: Weighted average of overlapping control points.

4.2.2 Performance

A lap time is calculated for the merged path as usual. For this attempt on Clay Pigeon Raceway, a lap time of 33.006 seconds was estimated, nearly a 3% increase on optimising the full track. In this case, the merging between sectors did more damage than the bespoke corner optimisations could counter. However, a significant decrease in run time has been achieved. Table 4.3 presents performance relative to the full circuit compromise method for multiple tracks.

Track	Sectors	Corner Ratio	Lap Time	Run Time
Buckmore Park	7	57%	102.1%	46.2%
Clay Pigeon	4	62%	102.9%	61.1%
Glan-y-Gors	7	46%	103.1%	39.2%
Whilton Mill	9	41%	101.8%	57.9%
Mean			102.5%	51.1%

Table 4.3: Sector-based optimisation performance, relative to optimising the full circuit.

The sector optimisation for Whilton Mill was able to reduce computation time further, with a lap time increase less than 2%. Figure 4.7 shows the sectors identified for the circuit. A clear trait is that many short corners are identified, often separated by long, straight sections. Clay Pigeon had much longer sequences of corners defining a sector, with few clear straights between them. The proportion of track samples identified as a corner given in Table 4.3 further demonstrates this point. With short straights, the transition from one sector's path to another can be much tighter - sacrificing lap time. The long overlaps on Whilton Mill allow a much more gradual transition between sectors.

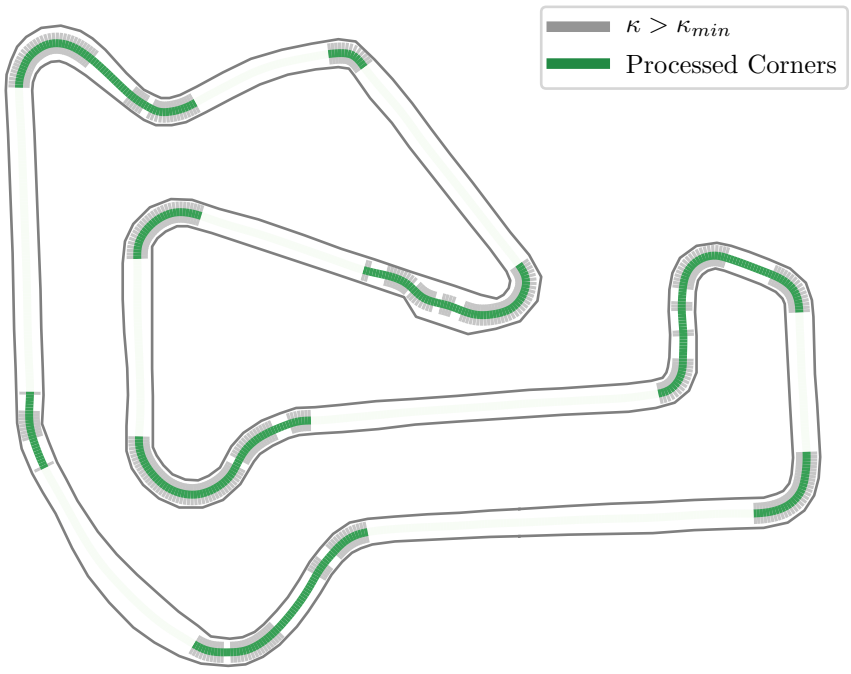


Figure 4.7: Corner sequences identified for Whilton Mill Circuit.

More complex tracks suffer from the divide-and-conquer approach of sector optimisations. At the start of this section, it was noted that this technique is similar to that of a human driver breaking up the track to form a racing line. Tracks such as Clay Pigeon are described as more 'technical', due to longer sequences of corners being dependent on each other. This makes the task of finding the best racing line much harder to find than for a series of isolated corners. This is reflected in the performance of the sector-based optimisation.

This approach has offered a useful tactic in reducing computation time, assuming adequate hardware for parallelism. Alternatively, local paths could be generated and merged with the global path as new sectors are identified on the reconnaissance lap. A partial racing line could then be ready for the start of the next lap. Unfortunately, the aims of improving lap time over the full circuit compromise method have not been achieved, although the potential is recognised with ε^* requiring different values across the circuit.

4.3 Estimating Compromise Weight

Searching for ε^* causes a large increase in run time from the curvature method to the compromise method. This section explores the possibility of deriving an appropriate ε from track properties. This would remove the costly secondary optimisation for the compromise weight. Minimising curvature is equivalent to using $\varepsilon = 0$, so the direct calculation of a weight would be the only additional computation.

4.3.1 Track-Weight Relationship

Hairpin turns have already demonstrated a positive response to higher compromise weights, as per Figure 3.10. During the sector-based optimisations, some sectors with higher ε^* values contained tight corners (e.g. Clay Pigeon's Sector 1, Figure 4.5a). The relationship between track curvature and the resulting ε^* will be tested first. Curvature here is calculated from the centerline path ($\alpha = 0.5$). The four test circuits will be used to evaluate the relationship: both the sector and full circuit optimisations for ε. This gives a total of 31 data points. Figure 4.8 plots this mean curvature and the resulting compromise weight.

There is no obvious relationship in this data, with a lot of variation in ε^* for similar curvatures. To progress this investigation, irrelevant information needs to be removed. For example, using the mean curvature instead of a

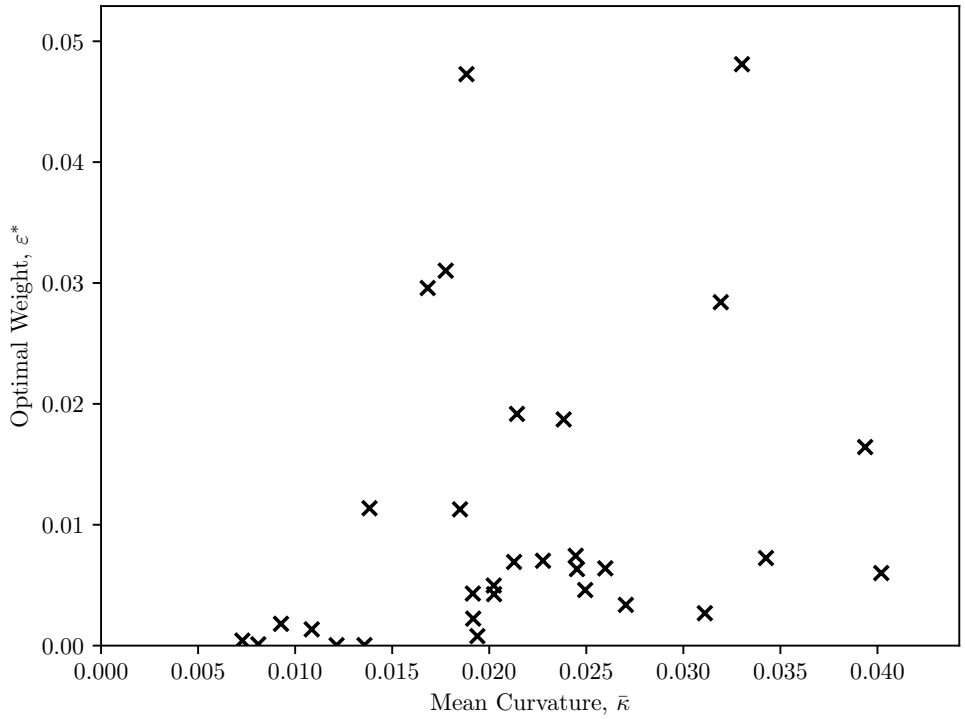
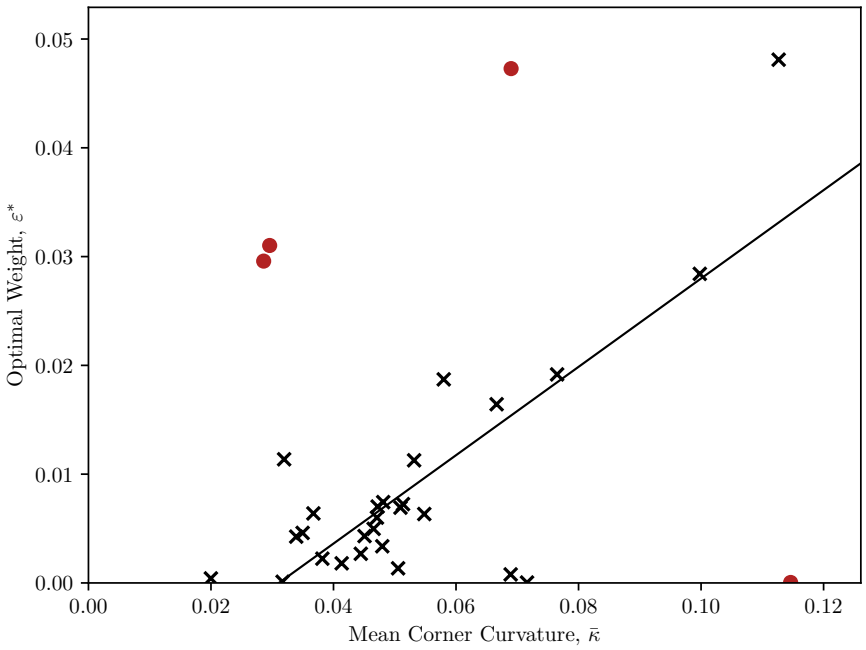


Figure 4.8: Plot of mean curvature and optimal compromise weight.

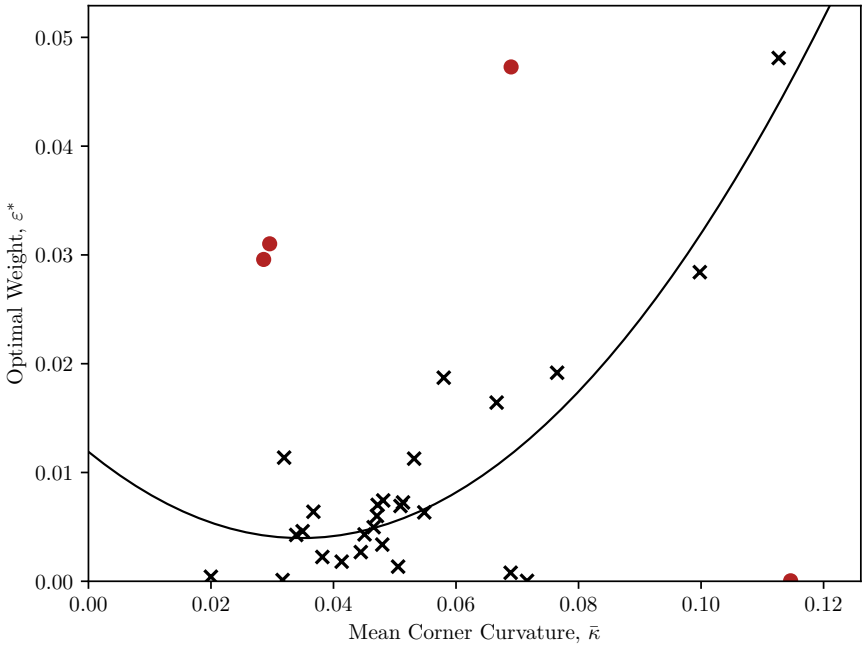
sum discounted the length of a track. Another aspect that can be ignored is the long straights between corners. The racing line through an isolated corner should not be dependent on the length of straights either side. Including straights would skew the mean curvature. The corner detection process from Section 4.2 will be used to identify the relevant curvatures. Figure 4.9 plots this refined data; anomalies are highlighted and ignored for the best fits. Both linear and quadratic fits were computed using `scipy.stats.polyfit`, which takes a least-squares approach to approximate the relationship.

Pearson's correlation coefficient, ρ, has been used to measure the linear correlation between mean corner curvature, $\bar{\kappa}$, and the computed weight ε^*. In this test, ρ was calculated to be 0.793. A quadratic relationship was also fitted, and $\rho = 0.686$ calculated by using $\sqrt{\varepsilon^*}$. Both results indicate a strong relationship between mean corner curvature and the resulting compromise weight - a linear fit especially so. This test gives a candidate (Equation 4.1) for pre-computing the compromise weight. Estimated values are bound to the range $0 \leq \varepsilon \leq 1$.

$$\varepsilon^* \approx 0.406\bar{\kappa} - 0.013 \tag{4.1}$$



(a) Linear fit; $\rho = 0.793$.



(b) Quadratic fit; $\rho = 0.686$.

Figure 4.9: Relationship between mean corner curvature and optimal compromise weight. Red circles indicate anomalies excluded from the analysis.

4.3.2 Performance

A *leave-one-out* cross-validation strategy has been used to test the performance of using the estimated weight (Sammut and Webb, 2010). For each track, a linear relationship is established using the optimal compromises of the three remaining tracks and their sectors. This fit is then used to generate an appropriate ε, and F_ε is minimised to optimise the trajectory. This strategy ensures tests are run on tracks that were not used to generate each model, avoiding overfitting or selection bias (Cawley, 2010).

Track	Lap Time vs. $\varepsilon = 0$	Lap Time vs. ε^*
Buckmore Park	94.8%	100.5%
Clay Pigeon	97.5%	103.0%
Glan-y-Gors	99.2%	101.5%
Whilton Mill	96.3%	100.1%
Mean	96.9%	101.3%

Table 4.4: Estimated compromise weight performance, relative to optimising the weight (ε^*) and minimising curvature ($\varepsilon = 0$).

On average, a weight was computed and used to optimise a trajectory in 5.04 seconds. As expected, this is comparable to the minimum curvature method. Table 4.4 compares the resulting lap times to the minimum curvature approach. Lap time has been improved by more than 3% with a negligible increase in run time.

Times are also compared to finding and using the optimal weight ε^*, a process that takes more than 100 seconds. Increases in lap time ranged from 0.1% to 3%, indicating varying accuracy in the curvature-to-weight model. This is unsurprising given the variation in ε^* seen in Figure 4.9a for similar curvatures.

The results are promising, especially given the limited sample size used. An immediate improvement in lap time has been made over the baseline curvature minimisation. Once again, a trade-off between lap time and run time is required to improve on the optimal compromise approach. However, using an estimated weight has achieved a much larger speedup than the sector-based approach, for a smaller loss in lap time.

The map from corner curvature to ε will differ vehicle to vehicle. For example, vehicles with higher grip will be able to take corners tighter without loss of speed/acceleration, meaning a higher weighting for track length is suitable. This approach has high potential but is dependent on a large data set to appropriately fit the weight estimation model, in terms of tracks and vehicles.

(a) Compromise method, lap time = 37.729s.

(b) Sector-based compromise method, lap time = 38.528s.

(c) Compromise method with estimated weight, lap time = 37.902s.

Figure 4.10: Trajectories generated for Buckmore Park (clockwise).

Chapter 5

Conclusions

The achievements, limitations, and impact of the project are discussed in this closing chapter. Findings and results are also compared to the literature surveyed in Chapter 2. Branches into out-of-scope topics that have been noted throughout the dissertation are also addressed, in a section on further work.

5.1 Project Summary

5.1.1 Achievements

A process for generating trajectories for an autonomous vehicle around a race track of known boundaries has been developed, consisting of a racing line (the path) and velocity profile. The racing line is optimised by minimising a linear combination of the resulting path length and its curvature at regular sample points. A rudimentary vehicle dynamics model is then used to generate velocity profiles for paths, from which lap times are estimated. An optimal combination between path length and curvature, referred to as the *compromise weight*, is found by computing a lap time for paths generated with different weights.

Using empirical data from this method, a relationship between track curvature and the optimal compromise weight was approximated, to bypass this additional optimisation of the weight. This led to significant decreases in run time, with a small cost in the resulting lap time. A divide-and-conquer approach to optimising tracks was also investigated, with paths for individual sectors generated and merged to form a complete trajectory. This also led to some improvement in run time when parallelised, although hopes of maintaining or even improving lap time were not fulfilled.

The core method of using a compromise between path length and curvature was based on the work of Braghin et al. (2008), which heavily influenced Heilmeier et al. (2019) in their Roborace DevBot trial. This dissertation has adapted Braghin et al.'s compromise method for Formula Student A.I. events, with tracks defined by regular cone gates. Run time of this implementation was greatly improved with the investigations carried out in Chapter 4.

It is difficult to make direct comparisons to the original method, as Braghin et al. do not provide run time data and Heilmeier et al. verified their implementation on a specific track much longer than the circuits used in this project. However, Heilmeier et al. state that for computing racing lines in a live environment, the reduced run time of a minimum curvature approach can be preferable over a direct minimisation of lap time (a 50% reduction in their case). A 3% improvement in lap time has been achieved in this project by approximating the optimal compromise weight, with a negligible increase in run time. This offers an attractive alternative for online path planning, while still avoiding long optimisations of the compromise weight or the lap time itself.

Kapania, Subosits and Gerdes (2019) report run times of their iterative curvature minimisation on a laptop computer, for varying 'lookahead' distances (i.e. the length of track optimised). A 900-metre lookahead was optimised in 6 seconds, and 1800 metres in 12 seconds. The four tracks tested in Chapter 4 varied from 815 to 1200 metres, each with solve times between 4 and 5 seconds - albeit on a different specification machine. At the very least this demonstrates comparable run time performance.

Kapania, Subosits and Gerdes go on to discuss a high-performance implementation written in C that can compute the minimum curvature path for a 650-metre lookahead in 0.005 seconds. This run time is used to justify their method for use as a real-time trajectory planner. The compromise method implemented in this project is comparable to the minimum curvature optimisation, with some additional consideration for path length and computing an appropriate compromise weight. If similar reductions in run time can be achieved for these additional computations, it could be feasible for the method to be used in an online capacity.

Sector-based construction of a racing line provided another trade-off between lap time and run time, opening the door for parallelisation of the planning algorithm. The independent optimisation of sectors could allow the racing line to be constructed during an initial reconnaissance lap. An online planner could optimise each sector of the track as they are discovered, building up a partial racing line ready for use at the start of the next lap. So long as the final sector could be optimised and merged into the full path before it is reached by the vehicle again, an optimised racing line could be used immediately for the second lap.

5.1.2 Limitations

While well-performing methods have been implemented in terms of run time, the racing line itself is sub-optimal. This is a trade-off acknowledged throughout relevant literature, with the complexity of a direct time minimisation problem difficult to overcome.

Investigations into the proposed modifications of Braghin et al.'s method relied on a limited data set and a simulated test environment. Cone locations for the test tracks used were manually collected, putting a bottleneck on data collection and, therefore, sample sizes in testing. Ideally, a larger set of tracks with varying lengths and other properties would be used to establish a more reliable model between tracks and their optimal compromise weight. Likewise, the impact on the mean run time and lap time would be more accurately determined.

The software-only testing process allowed quick and extensive investigations of the methods. While this was adequate for identifying relative improvements in run time, lap time data will be less representative. Real-world testing is required to assert actual changes in lap time for different methods, or at the very least verify accuracy of the predicted times.

5.2 Further Work

5.2.1 Continued Investigations

Promising improvements to Braghin et al.'s compromise method were identified in this work. Investigations into finer details of these extensions would be a sensible continuation, to try and further limit their costs in lap time.

Modelling a relationship between track properties and a well-performing compromise weight has a high potential for reducing run time by using previously collected data. A proof of concept was demonstrated using mean corner curvature, but other properties could be used to find a stronger correlation and, therefore, a better performing model.

The sector-based optimisation focused on independent and potentially parallel sub-optimisations of the racing line, but at the cost of lap time. There was hope for improving lap time as compromises suited to each sector were used, but merging paths linearly along straights resulted in a net increase. Other merging techniques or a more subtle approach to varying the length-curvature combination around the circuit may be needed to fully exploit this technique.

5.2.2 Beyond a Prototype

To investigate different approaches to the optimisation problem, prototype Python code was written and tested. This relied on libraries for minimising objective functions and other utilities such as spline construction and sampling. These were selected for ease of use and to reduce development time so that the methods for path planning remained the focus of the project. With more development time, more efficient implementations (in terms of resources and time) could be written. By improving the actual implementation of methods without changing them fundamentally, run time would be reduced with minimal impact on the resulting lap time.

All testing has been performed on a personal computer, with moderate CPU

speed and RAM capacity. Deploying the racing line generator as part of a real autonomous system would likely use a specialised processing unit. This would open up the opportunity for hardware-specific optimisation to the software.

A potential avenue considered for this project was that of GPU acceleration: utilising a (general-purpose) graphics processing unit to parallelise some of the processes. This could be in the form of GPU-enhanced optimisation algorithms, such as the parallelised L-BFGS-B algorithm from Fei et al. (2014). Some fine-grained parallelism could also be achieved on operations performed at each path sample, such as curvature calculations and velocity limits. To achieve reasonable speedup in this manner, large problem sizes are needed so that the overheads of parallelisation do not outweigh the benefits. This may mean limited improvement on small tracks used for testing in Chapter 4, but optimisation of much longer circuits can be achieved in a reasonable time. Direct minimisation of lap time was ruled out on the grounds of run time, so it would be interesting to see if it could be made feasible through parallelism and GPU acceleration.

5.2.3 Pipeline Integration

Optimising the racing line is just one component of the fully autonomous system described in Section 1.1.2. Integrating the process as a part of a complete system will enable its potential to be applied in a working environment. This could be a real-world test with an autonomous vehicle, or a fully simulated environment with all the stages of the pipeline working together.

Inputs, outputs and responsibilities will evolve with the whole system, so racing line generation may not be the complete product for a global path planning subsystem. For example, it may need to be expanded to handle routing the vehicle from its current position to the intended path. It may also need to consider additional parameters, such as wet weather reducing friction between road and tyre. This would have knock-on effects for the optimal path and the estimated compromise weight. It is these kinds of complications that see a fundamental process develop into a system applicable to real-world systems, and potentially for commercial use instead of the constrained motorsport environment.

If a fully autonomous system were implemented, practical testing would be the next logical step - improving the investigations carried out. The behaviour observed in simulations could be verified in practice, either supporting the findings or indicating a need for a different approach. Practical testing would enable another measure of performance employed by Heilmeier et al. (2019): comparing the lap time achieved by the autonomous vehicle to a human lap

record. This is an important statistic, indicating if the vehicle's performance is being utilised and, crucially, whether the autonomous system would be able to improve on current human drivers. In motorsport, comparisons between lap times, consistency, predictable behaviour and safety are all useful indicators of performance.

Real-world results could also be used to verify the accuracy of simulations or modify them to improve accuracy. The dynamics model from this work would need to be expanded, but the fundamental process could then be tested in the same manner. With more accurate simulations, tests could be performed with higher confidence, in greater quantities and at a reduced cost.

5.3 Final Remarks

Existing methods to generate racing lines have been adapted for Formula Student A.I. events and modified to reduce run time while minimising impact to lap time. By working in this constrained context, the methods could be improved without complications from other factors. The resulting process now stands to be refined and integrated for a completely autonomous system, and its potential applied to real-world environments.

References

Billington, J., 2018. The prometheus project: The story behind one of AV's greatest developments [Online]. Available from: `https://www.autonomousvehicleinternational.com/features/the-prometheus-project.html` [Accessed 21 November 2019].

Braghin, F., Cheli, F., Melzi, S. and Sabbioni, E., 2008. Race driver model. *Computers and Structures*, 86(13-14), pp.1503–1516.

Cawley, G.C., 2010. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11(7), pp.2079–2108.

Cottle, R.W. and Thapa, M.N., 2017. *Linear and nonlinear optimization*, International Series in Operations Research & Management Science, 253. 1st ed. New York, NY: Springer New York : Imprint: Springer.

Fei, Y., Rong, G., Wang, B. and Wang, W., 2014. Parallel L-BFGS-B algorithm on GPU. *Computers & Graphics*, 40(1), pp.1–9.

Funke, J., Theodosis, P., Hindiyeh, R., Stanek, G., Kritatakirana, K., Gerdes, C., Langer, D., Hernandez, M., Muller-Bessler, B. and Huhnke, B., 2012. Up to the limits: Autonomous Audi TTS. *IEEE intelligent vehicles symposium.* , pp.541–547.

Georgiev, I. and Day, O., 2018. Making a car drive itself: How to prepare for FS-AI [Online]. Available from: `https://www.imeche.org/docs/default-source/1-oscar/formula-student/2019/learn-to-win-2019/eufs_fs-ai_talk.pdf`.

Gerdts, M., Karrenberg, S., Mller-Beler, B. and Stock, G., 2009. Generating locally optimal trajectories for an automatically driven car. *Optimization and Engineering*, 10(4), pp.439–463.

Heilmeier, A., Wischnewski, A., Hermansdorfer, L., Betz, J., Lienkamp, M. and Lohmann, B., 2019. Minimum curvature trajectory planning and con-

trol for an autonomous race car. *Vehicle System Dynamics* [Online], pp.1–31. Available from: https://doi.org/10.1080/00423114.2019.1631455 [Accessed 19 November 2019].

Hoad, C., 2016. Threshold braking. *CAT Driver Training* [Online]. Available from: https://catdrivertraining.co.uk/threshold-braking/ [Accessed 5 March 2020].

Iagnemma, K. and Buehler, M., 2006. Editorial for journal of field robotics - special issue on the DARPA Grand Challenge. *Journal of Field Robotics*, 23(8), pp.461–462.

IMechE, 2019. FS-AI - Formula Student Artificial Intelligence [Online]. Available from: https://www.imeche.org/events/formula-student/team-information/fs-ai [Accessed 27 November 2019].

Jones, E., Oliphant, T., Peterson, P. et al., 2001–. SciPy: Open source scientific tools for Python [Online]. Available from: http://www.scipy.org.

Kabzan, J., Reijgwart, V., Ehmke, C., Prajapat, M., Bhler, A., Gosala, N., Gupta, M., Sivanesan, R., Dhall, A., Chisari, E., Karnchanachari, N., Brits, S., Dangel, M., Sa, I., Dub, R., Gawel, A., Pfeiffer, M., Liniger, A., Lygeros, J. and Siegwart, R., 2019. AMZ Driverless: The full autonomous racing system. *arXiv.org* [Online]. Available from: http://search.proquest.com/docview/2224722614/.

Kanal, S., 2019. How F1 technology has supercharged the world [Online]. Available from: https://www.formula1.com/en/latest/article.how-f1-technology-has-supercharged-the-world.6Gtk3hBxGyUGbNH0q8vDQK.html [Accessed 20 April 2020].

Kapania, N., Subosits, J. and Gerdes, J., 2019. A sequential two-step algorithm for fast generation of vehicle racing trajectories. *arXiv.org* [Online], 138(9). Available from: http://search.proquest.com/docview/2176097781/ [Accessed 19 November 2019].

Khan, S., 2017. *The development of a vehicle simulation tool from first principles.* Team Bath Racing, Department of Mechanical Engineering, University of Bath.

Meek, D.S. and Walton, D.J., 1992. Clothoid spline transition spirals. *Mathematics of Computation* [Online], 59(199), pp.117–133. Available from: http://www.ams.org/jourcgi/jour-getitem?pii=S0025-5718-1992-1134736-8.

Paden, B., Cap, M., Yershov, D. and Frazzoli, E., 2016. A survey of motion planning and control techniques for self-driving urban vehicles. *arXiv.org* [Online]. Available from: http://search.proquest.com/docview/2077112937/ [Accessed 21 November 2019].

Rizano, T., Fontanelli, D., Palopoli, L., Pallottino, L. and Salaris, P., 2013. Global path planning for competitive robotic cars. *52nd ieee conference on decision and control*. IEEE, pp.4510–4516.

Roborace, 2019. What is Roborace? [Online]. Available from: https://roborace.com [Accessed 27 November 2019].

SAE International, 2018. *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles* [Online]. Available from: https://doi.org/https://doi.org/10.4271/J3016_201806 [Accessed 22 November 2019].

SAE International, 2019. SAE standards news: J3016 automated-driving graphic update [Online]. Available from: https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic [Accessed 22 November 2019].

Sammut, C. and Webb, G.I., eds., 2010. *Leave-one-out cross-validation* [Online], Boston, MA: Springer US, pp.600–601. Available from: https://doi.org/10.1007/978-0-387-30164-8_469.

Schulz, E., 2017. [AMZ Driverless at Formula Student Germany] [Online]. Available from: https://drive.tech/en/stream-content/swiss-racing-team-wins-with-fastest-driverless-car [Accessed 21 April 2020].

Schwarting, W., Alonso-Mora, J. and Rus, D., 2018. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems* [Online], 1(1), pp.187–210. Available from: https://doi.org/10.1146/annurev-control-060117-105157 [Accessed 21 November 2019].

SGI, 2019. Saskatchewan driver's handbook [Online]. Available from: https://www.sgi.sk.ca/handbook/-/knowledge_base/drivers/braking [Accessed 5 March 2020].

Singh, K.B. and Taheri, S., 2015. Estimation of tire-road friction coefficient and its application in chassis control systems. *Systems Science & Control Engineering* [Online], 3(1), pp.39–61. Available from: http://www.tandfonline.com/doi/abs/10.1080/21642583.2014.985804 [Accessed 3 April 2020].

Snyman, J., 2005. *Practical mathematical optimization : An introduction to basic optimization theory and classical and new gradient-based algorithms*, Applied Optimization, 97. 1st ed. New York, NY: Springer US : Imprint: Springer.

Subosits, J. and Gerdes, J.C., 2015. Autonomous vehicle control for emergency maneuvers: The effect of topography. *2015 american control conference (ACC)*. pp.1405–1410. Available from: `https://doi.org/10.1109/ACC.2015.7170930`.

Tesla, 2019. Tesla Autopilot [Online]. Available from: `https://www.tesla.com/autopilot` [Accessed 23 November 2019].

Velenis, E. and Tsiotras, P., 2008. Minimum-time travel for a vehicle with acceleration limits: Theoretical analysis and receding-horizon implementation. *Journal of Optimization Theory and Applications* [Online], 138(2), pp.275–296. Available from: `https://doi.org/10.1007/s10957-008-9381-7`.

Zhang, X. and Mi, C., 2011. *Vehicle power management: Modeling, control and optimization*, Power Systems. London: Springer London.

Zhu, C., Byrd, R.H., Lu, P. and Nocedal, J., 1997. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *ACM Trans. Math. Softw.* [Online], 23(4), pp.550–560. Available from: `https://doi.org/10.1145/279232.279236`.

Appendices

A Code Listings

Appendix A contains Python code relevant to the discussion. The included code has been trimmed and/or edited for ease of reference. For example, package imports are removed and only methods relevant to the given task are included. Complete, unedited code is available at `github.com/joedavison17/dissertation`.

A.1 Curvature Minimisation

Listing 1: Evaluation of the path curvature metric, Γ^2.

```
class Path:
  """Wrapper for scipy.interpolate.BSpline."""

  def __init__(self, controls, closed):
    """Construct a spline through the given control points."""
    self.controls = controls
    self.closed = closed
    self.dists = cumulative_distances(controls)
    self.spline, _ = splprep( # scipy.interpolate
      controls, u=self.dists, k=3, s=0, per=self.closed
    )
    self.length = self.dists[-1]


  def gamma2(self, s):
    """Returns the sum of the squares of sample curvatures."""
    ddx, ddy = splev(s, self.spline, 2)
    return np.sum(ddx**2 + ddy**2)
```

Listing 2: Optimisation of control points to minimise curvature.

```
class Trajectory:
  """
  Stores the geometry and dynamics of a path, handling
  optimisation of the racing line. Samples are taken every
  metre.
  """

  def update(self, alphas):
    """
    Update control points and the resulting path.
    """
    self.alphas = alphas
    self.path = Path(
      self.track.control_points(alphas),
      self.track.closed
    )
    self.s = np.linspace(0, self.path.length, self.ns)


  def optimise_curvature(self):
    """Generate a path minimising curvature."""

    def objfun(alphas):
      self.update(alphas)
      return self.path.gamma2(self.s)

    t0 = time.time()
    res = minimize( # scipy.optimize
      fun=objfun,
      x0=np.full(self.track.size, 0.5),
      method='L-BFGS-B',
      bounds=Bounds(0.0, 1.0)
    )
    self.update(res.x)
    return time.time() - t0
```

A.2 Velocity Profiling

Listing 3: Generation of a velocity profile.

```
GRAV = 9.81 # ms^-2

class VelocityProfile:
  """
  Stores and generates a velocity profile for a given path
  and vehicle.
  """

  def __init__(self, vehicle, s, k, s_max=None):
    self.vehicle = vehicle
    self.s = s
    self.s_max = s_max
    self.limit_local_velocities(k)
    self.limit_acceleration(k)
    self.limit_deceleration(k)
    self.v = np.minimum(self.v_acclim, self.v_declim)


  def limit_local_velocities(self, k):
    self.v_local = np.sqrt(self.vehicle.cof * GRAV / k)


  def limit_acceleration(self, k_in):
    # Start at slowest point
    shift = -np.argmin(self.v_local)
    s = np.roll(self.s, shift)
    v = np.roll(self.v_local, shift)
    k = np.roll(k_in, shift)
    # Limit according to acceleration
    for i in range(s.size):
      wrap = i == (shift % s.size)
      if wrap and self.s_max is None: continue
      if v[i] > v[i-1]:
        traction = self.vehicle.traction(v[i-1], k[i-1])
        force = min(
          self.vehicle.engine_force(v[i-1]),
          traction
        )
        accel = force / self.vehicle.mass
        ds = self.s_max - s[i-1] if wrap else s[i] - s[i-1]
        vlim = sqrt(v[i-1]**2 + 2*accel*ds)
        v[i] = min(v[i], vlim)
    # Reset shift and return
    self.v_acclim = np.roll(v, -shift)
```

```
def limit_deceleration(self, k_in):
  # Start at slowest point, move backwards
  shift = -np.argmin(self.v_local)
  s = np.flip(np.roll(self.s, shift), 0)
  k = np.flip(np.roll(k_in, shift), 0)
  v = np.flip(np.roll(self.v_local, shift), 0)
  # Limit according to deceleration
  for i in range(s.size):
    wrap = i == (-shift)
    if wrap and self.s_max is None: continue
    if v[i] > v[i-1]:
      traction = self.vehicle.traction(v[i-1], k[i-1])
      decel = traction / self.vehicle.mass
      ds = self.s_max - s[i] if wrap else s[i-1] - s[i]
      vlim = sqrt(v[i-1]**2 + 2*decel*ds)
      v[i] = min(v[i], vlim)
  # Reset shift/flip and return
  self.v_declim = np.roll(np.flip(v, 0), -shift)
```

A.3 Compromise Optimisation

Listing 4: Minimising the compromise between path length and curvature.

```
def optimise_compromise(self, eps):

  def objfun(alphas):
    self.update(alphas)
    k = self.path.gamma2(self.s)
    d = self.path.length
    return (1-eps)*k + eps*d

  t0 = time.time()
  res = minimize( # scipy.optimize
    fun=objfun,
    x0=np.full(self.track.size, 0.5),
    method='L-BFGS-B',
    bounds=Bounds(0.0, 1.0)
  )
  self.update(res.x)
  return time.time() - t0
```

Listing 5: Optimisation of the compromise weight, ε.

```
def optimise_compromise_weight(self, eps_min, eps_max):

  def objfun(eps):
    self.optimise_compromise(eps)
    self.update_velocity()
    return self.lap_time()

  t0 = time.time()
  res = minimize_scalar( # scipy.optimize
    fun=objfun,
    method='bounded',
    bounds=(eps_min, eps_max)
  )
  self.epsilon = res.x
  self.optimise_compromise(self.epsilon)
  end = time.time()
  return end - t0
```

A.4 Sector Optimisation

Listing 6: Process for determining corner sequences.

```
def define_corners(path, s, k_min, proximity, length):
  """
  Analyse the track to find corners and straights.

  k_min: defines the minimum curvature for a corner
  proximity: corners within this distance are joined
  length: corners must exceed this length to be accepted

  Returns:
    corners: an array of control point index pairs defining
      the start and end corners
    is_corner: a boolean mask stating whether a sample is
      part of a corner or not
  """
  is_corner = path.curvature(s) > k_min
  is_corner = filter_corners(is_corner, s, length, proximity)
  corners = samples_to_controls(
    s, corner_idxs(is_corner), path.dists
  )
  return corners, is_corner


def filter_corners(is_corner, dists, length, proximity):
  """
  Update corner status according to length and proximity.
  """
  # Shift to avoid splitting a straight or corner
  shift = np.argwhere(is_corner != is_corner[0])[0][0]
  is_corner = np.roll(is_corner, -shift)

  # Remove short straights
  start = 0
  for i in range(1, is_corner.size):
    if is_corner[i-1]:
      if not is_corner[i]:
        # Record straight start
        start = i
    elif is_corner[i]:
      # Measure straight and convert if too short
      is_corner[start:i]
        = (dists[i] - dists[start]) < proximity

  # Remove short corners
  start = 0
  for i in range(1, is_corner.size):
```

```
    if is_corner[i-1]:
      if not is_corner[i]:
        # Measure corner and convert if too short
        is_corner[start:i]
          = (dists[i] - dists[start]) > length
    elif is_corner[i]:
      # Record corner start
      start = i
  return np.roll(is_corner, shift)


def corner_idxs(is_corner):
  """
  Determine samples at which corner sequences start/end.
  """
  # Shift to avoid splitting a straight or corner
  shift = np.argwhere(is_corner != is_corner[0])[0][0]
  is_corner = np.roll(is_corner, -shift)

  # Search for corners
  corners = np.array([],dtype=int)
  n = len(is_corner)
  start = shift
  for j in range(1, n+1):
    i = j % n
    if is_corner[i-1]:
      if not is_corner[i]: # Corner -> straight
        end = (i + shift) % n
        if len(corners) > 0:
          corners = np.vstack((corners, [start, end]))
        else:
          corners = np.array([start, end])
    else:
      if is_corner[i]: # Straight -> corner
        start = (i + shift) % n
  return corners


def samples_to_controls(s_dist, s_idx, c_dist):
  """Convert sample distances to control point indices."""
  n = s_idx.size
  s_flat = s_idx.ravel()
  c_flat = np.zeros(n, dtype=int)
  for i in range(n):
    j = 0
    while s_dist[s_flat[i]] > c_dist[j]: j += 1
    c_flat[i] = j
  return c_flat.reshape(s_idx.shape)
```

Listing 7: Parallel optimisation of sector paths and merging the results.

```
def optimise_sectors(self, k_min, proximity, length):
  """
  Generate a path that optimises the path through each
  sector, and merges the results along straights.
  """
  # Define sectors
  t0 = time.time()
  corners, _
    = self.track.corners(self.s, k_min, proximity, length)

  # Optimise path for each sector in parallel
  nc = corners.shape[0]
  pool = Pool(os.cpu_count() - 1)
  alphas = pool.map(
    partial(optimise_sector_compromise,
      corners=corners, traj=self),
    range(nc)
  )
  pool.close()

  # Merge sectors and update trajectory
  alphas = np.sum(alphas, axis=0)
  self.update(alphas)
  return time.time() - t0
```

Listing 8: Optimisation of individual sector paths.

```
def optimise_sector_compromise(i, corners, traj):
  """
  Builds a new Track for the given corner sequence, and
  optimises the path through it by the compromise method.
  """

  # Represent sector as new Track
  nc = corners.shape[0]
  n = traj.track.size
  a = corners[(i-1)%nc,1]  # Sector start
  b = corners[i,0]         # Corner entry
  c = corners[i,1]         # Corner exit
  d = corners[(i+1)%nc,0]  # Sector end
  idxs = idx_modulo(a,d,n)
  sector = Trajectory(
    Track(
      left=traj.track.left[:,idxs],
      right=traj.track.right[:,idxs]
    ),
    traj.vehicle
  )

  # Optimise path through sector
  rt = sector.optimise_compromise_weight()

  # Weight alphas for merging across straights
  weights = np.ones((d-a)%n)
  weights[:(b-a)%n] = np.linspace(0, 1, (b-a)%n)
  weights[(c-a)%n:] = np.linspace(1, 0, (d-c)%n)
  alphas = np.zeros(n)
  alphas[idxs] = sector.alphas * weights
  return alphas
```

B Ethics Checklist

Department of Computer Science
12-Point Ethics Checklist for UG and MSc Projects

Student Joe Davison

Academic Year or Project Title 2019-20

Supervisor Wenbin Li

Does your project involve people for the collection of data other than you and your supervisor(s)? [redacted] / NO

If the answer to the previous question is YES, you need to answer the following questions, otherwise you can ignore them.

This document describes the 12 issues that need to be considered carefully before students or staff involve other people ('participants' or 'volunteers') for the collection of information as part of their project or research. Each answer must be affirmative. Replace the text beneath each question with a statement of how you address the issue in your project.

1. *Have you prepared a briefing script for volunteers?* YES / NO
   Briefing means telling someone enough in advance so that they can understand what is involved and why – it is what makes informed consent informed.

2. *Will the participants be informed that they could withdraw at any time?* YES / NO
   All participants have the right to withdraw at any time during the investigation, and to withdraw their data up to the point at which it is anonymised. They should be told this in the briefing script.

3. *Is there any intentional deception of the participants?* YES / NO
   Withholding information or misleading participants is unacceptable if participants are likely to object or show unease when debriefed.

4. *Will participants be de-briefed?* YES / NO
   The investigator must provide the participants with sufficient information in the debriefing to enable them to understand the nature of the investigation. This phase might wait until after the study is completed where this is necessary to protect the integrity of the study.

5. *Will participants voluntarily give informed consent?* YES / NO
Participants MUST consent before taking part in the study, informed by the briefing sheet. Participants should give their consent explicitly and in a form that is persistent –e.g. signing a form or sending an email. Signed consent forms should be kept by the supervisor after the study is complete.

6. *Will the participants be exposed to any risks greater than those encountered in their normal work life (e.g., through the use of non-standard equipment)?* YES / NO
Investigators have a responsibility to protect participants from physical and mental harm during the investigation. The risk of harm must be no greater than in ordinary life.

7. *Are you offering any incentive to the participants?* YES / NO
The payment of participants must not be used to induce them to risk harm beyond that which they risk without payment in their normal lifestyle.

8. *Are you in a position of authority or influence over any of your participants?* YES / NO
A position of authority or influence over any participant must not be allowed to pressurise participants to take part in, or remain in, any experiment.

9. *Are any of your participants under the age of 16?* YES / NO
Parental consent is required for participants under the age of 16.

10. *Do any of your participants have an impairment that will limit Their understanding or communication?* YES / NO
Additional consent is required for participants with impairments.

11. *Will the participants be informed of your contact details?* YES / NO
All participants must be able to contact the investigator after the investigation. They should be given the details of the Supervisor as part of the debriefing.

12. *Do you have a data management plan for all recorded data?* YES / NO
All participant data (hard copy and soft copy) should be stored securely, and in anonymous form, on university servers (not the cloud). If the study is part of a larger study, there should be a data management plan.