Designing a Receding Horizon Planner for an Autonomous Formula Student Racecar

by

Kerry He

Supervised by:

Hoam Chung

Final Year Project

Mechanical and Aerospace Engineering
Monash University
October 2021

Summary

In autonomous driving, the motion planning subsystem is required to determine a feasible state and control trajectory to navigate the vehicle to perform a specific task. This thesis presents an implementation of a receding horizon planner (RHP) to perform motion planning for Monash Motorsport's autonomous racecar to compete in the Formula Student Driverless competition.

Multiple modelling and discretisation methods were explored to determine the best performing RHP formulation. An extensive comparison of each formulation was conducted by prototyping each RHP in MATLAB, and performing simulated experiments in a simplified environment. The best performing RHP was then translated into C++ and validated in a full real time hardware-in-the-loop simulation.

Overall, a linear time-varying RHP formulation utilising a dynamic bicycle model is proposed, where the vehicle dynamics and path constraints are linearised at each time step, allowing the RHP to be formulated and solved as a quadratic program. An optimal racing line is precomputed offline by solving for a periodic time-optimal trajectory along the entire track, which the RHP then tracks in real time. Through simulated experiments, the proposed RHP is shown to be robust to noise, time delay and modelling error, and successfully outperforms Monash Motorsport's previous motion planning and control implementations. The RHP is demonstrated to safely achieve speeds of up to 25 m/s while running in real time at 50 Hz.

Acknowledgements

Firstly, I would like to thank my supervisor, Dr. Hoam Chung, for giving me guidance on the project throughout the year. His expertise was hugely beneficial to the success of the project and my own academic growth. I appreciate the weekly meetings, extremely prompt email responses, time spent looking over my work, the flexibility he allowed me to explore my project, and the informal chats we had about topics other than my honours project.

I am also grateful to the Monash Motorsport team as a whole, for the opportunities of personal and professional development it has provided, and for providing such a thrilling context to be able to develop my research for. The support, friendships, and lessons Monash Motorsport has provided me have been invaluable to me. I hope that this thesis will provide the team with a solid foundation for motion planning, and help in making the autonomous vehicle concept competitive on a global scale.

Finally, I would like to thank my friends who have helped proof-read my thesis: J. Coleman, A. Ye-Lin, A. Romesh, and L. Candido. It is a very long document, and I greatly appreciate their time spent reading through it and providing feedback for it.

Acronyms

BFGS Broyden–Fletcher–Goldfarb–Shanno

CoG center of gravity

ECU electronic control unit

FSD Formula Student Driverless

GPS global positioning system

IMU inertial measurement unit

IP interior-point

LiDAR light detection and ranging

LTV linear time-varying

MMS Monash Motorsport

NLP nonlinear program

PID proportional–integral–derivative

QP quadratic program

RHP receding horizon planner

RK Runge-Kutta

RTI real-time iterative

SLAM simultaneous localisation and mapping

SQP sequential quadratic program

Contents

Chapter 1

Introduction

1.1 Background

Formula Student Driverless (FSD) is a university engineering design competition which started in 2017 and is ran by Formula Student[1]. The competition tasks student teams with designing, building, and programming a Formula-style race car which can autonomously navigate through known and unknown tracks as quickly as possible. There are four different missions that the autonomous vehicle must complete:

- Acceleration: 75 m of a simple straight line track.
- Skidpad: A figure of eight track, where the car performs two laps around the right-hand loop, followed by two laps around the left-hand loop.
- Autocross: A single lap of an unknown, closed-loop track consisting of straights, constant turns, hairpin turns, slaloms and other miscellaneous track features.
- Trackdrive: 10 laps of the same track as the autocross mission. However, data obtained from the autocross mission can be utilised.

Each of these tracks have boundaries which are outlined by cones. Points are awarded for each mission based on the times taken to complete them, scaled relative to the team with the fastest times. A 2 s time penalty is added onto the lap times if the the car collides into a cone, and a 10 s time penalty is added if the vehicle exits the track boundaries completely. Vehicles race independently on the track for each mission.

Monash Motorsport (MMS) is Monash University's Formula Student team which began development of a driverless car in 2018. In 2019, the team converted their existing rear-wheel drive electric car into a driverless car named M19-D. On M19-D, cameras and light detection and ranging (LiDAR) sensors are used to detect cones around the track. The simultaneous localisation and mapping (SLAM) algorithm uses these landmark detections and additional data

[1] https://www.formulastudent.de/fsg/

from the global positioning system (GPS) and inertial measurement unit (IMU) to perform state estimation and mapping. A racing trajectory is computed by the motion planning algorithm using this state estimate, and the corresponding velocity and steering wheel angle setpoints are communicated to low-level proportional–integral–derivative (PID) controllers which interface directly with the physical actuators. High-level autonomous systems algorithms are run on the NVIDIA Jetson AGX Xavier computing unit, while low-level safety and control algorithms are run on the MoTeC M150 electronic control unit (ECU). An overview of the autonomous systems currently on M19-D is shown in Figure 1.1. Currently, the team is in development of a new electric-driverless car, M21, with the aim to compete in FSD in Europe.

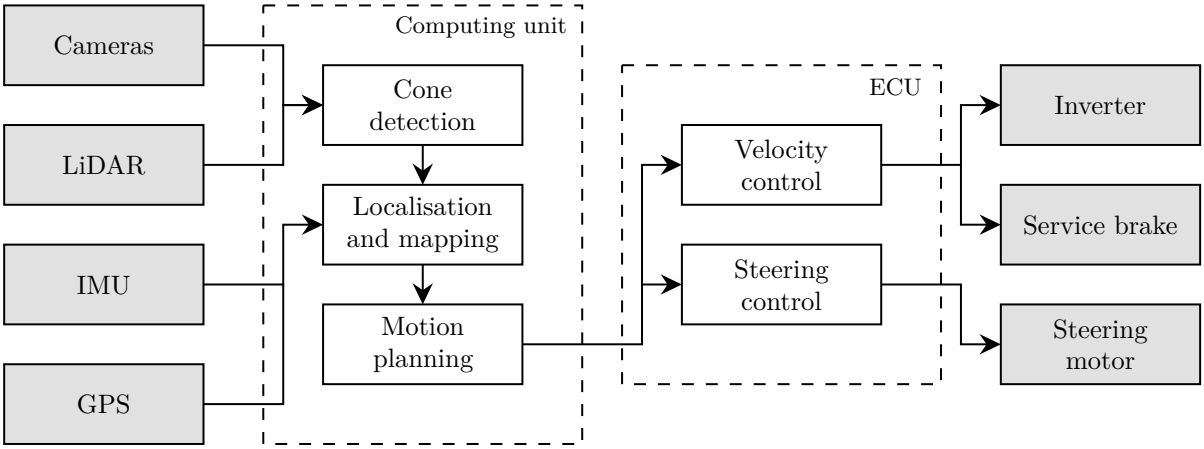

Figure 1.1: System overview of M19-D's autonomous systems. Hardware is represented by grey blocks, and software is represented by white blocks.

1.2 Research problem

M19-D is currently using an early implementation of a receding horizon planner (RHP) as its motion planner. Due to being an early implementation, the main goal for the planner was to be able to successfully finish missions, with lap time performance only being a secondary subgoal. Therefore, the RHP was implemented using a relatively simplistic formulation, and an optimal control library, ACADO Toolkit [1], was used as a high-level interface to solve the RHP problem.

This thesis aims to improve upon the previous RHP implementation for use on M19-D, and ultimately improve the lap times that can be achieved by the autonomous systems. This was achieved by exploring two key facts to the formulation of an RHP:

- Modelling: The plant model determines how accurately the RHP can predict its trajectory, and therefore how much the vehicle can be pushed to the limits while ensuring the RHP still knows how the vehicle will behave. The kinematic bicycle model is currently used, but a more complex vehicle model such as a dynamic bicycle model may yield significant improvements. In addition to comparing vehicle models, other state representations

such as using a different coordinate frame, or exploring new or different ways of enforcing constraints, are also explored. Moreover, as time delay is a significant factor in the performance of any controller, methods for compensating for it are also explored.

- Discretisation: The high-level RHP interface performs the conversion from optimal control problem to optimisation problem internally, known as discretisation or transcription. However, this imposes certain restrictions on how the problem can be formulated, such as enforcing a certain template on how the objective function must be defined. By discretising the problem by hand, the optimal control problem can be formulated much more flexibly. Multiple discretisation schemes are explored and compared against each other in this thesis, including successive linearisation, multiple shooting, and collocation schemes.

To achieve these aims, different modelling and discretisation methods were first prototyped and evaluated in MATLAB in a simplified simulation environment. From these experiments, the best performing formulation was implemented in C++, and evaluated in a more accurate hardware-in-the-loop simulation. The performance of this RHP was benchmarked against MMS's existing motion planning implementations to verify whether the proposed RHP could yield improved lap times over these existing implementations.

The key performance metric of the RHP was the total lap time achieved in the trackdrive mission, as this would evaluate its performance over a variety of track features including straight lines, hairpins and slaloms, as well as testing the robustness of the planner over a long racing period. Racing around an unknown track as is required for the autocross mission is not considered in the scope of this project.

1.3 Literature review

1.3.1 Motion planning

Autonomous vehicle racing is a growing field of research. Alongside FSD, the DARPA Grand Challenge [2], DARPA Urban Challenge [3], Roborace [4] and the Indy Autonomous Challenge [5] are examples of other autonomous racing competitions. Stanford's autonomous Audi TTS has also served as a testbench for autonomous racing research [6]. Apart from these examples however, most other research in this field validates results using simulation [7, 8] or small-scale model cars [9]. A key system in autonomous driving software architectures which is pivotal to maximising the racing capabilities of autonomous vehicles is motion planning, which aims to find a dynamically feasible trajectory, then follow this trajectory in real time in the presence of noise, disturbances, and other uncertainties.

A common strategy for motion planning if the track is known is to precompute an optimal racing line and velocity profile, which is then tracked in real time using an online motion planning algorithm [4, 5, 10]. Generating this optimal racing line typically involves formulating the problem as an optimisation problem which minimises the lap time, which can be solved using

traditional optimisation techniques [10] or alternative methods such as genetic algorithms [11]. Other approaches solve for the optimal racing line and velocity profile separately. These approaches typically involve solving for a minimum curvature racing line, after which a feasible velocity profile constrained by the curvature of the path and maximum accelerations of the vehicle is fit to the path [4, 12].

Several online motion planning methods have been proposed in literature. Kinematic methods such as pure pursuit [5] and Stanley control [13] use kinematic vehicle models to derive a steering control law. These methods are computationally inexpensive and perform well at low speeds. However, the performance degrades at higher speeds and accelerations when tire dynamics become nontrivial. Another class of controllers commonly used in autonomous racing are optimal control techniques such as model predictive control (MPC) or RHP[2] [9, 10]. Compared to other control algorithms, the most significant advantage of optimal control is its ability to implicitly incorporate state and control constraints into the problem formulation in real time. Moreover, optimal control can be formulated using high-fidelity vehicle models, and can flexibly choose how to define its objective function. These factors are particularly advantageous for autonomous racing, where maximising lap time performance requires a thorough understanding of the vehicle dynamics and handling limits of the vehicle. However, the main drawback of optimal control is computational burden, as an optimisation problem needs to be solved online at every time step.

1.3.2 Receding horizon planning

Optimal control views a control problem through the lens of an optimisation problem by solving for a control trajectory which minimises a given objective function. The form of a typical constrained continuous-time optimal control problem is shown below

$$\min_{\xi,u} \quad J(\xi,u) = l_N(\xi(t_f)) + \int_{t_0}^{t_f} l(\xi(\tau),u(\tau))\,d\tau \tag{1.1a}$$

$$\text{subj. to} \quad r(\xi(t_0),\xi(t_f)) = 0 \qquad \text{(boundary conditions)} \tag{1.1b}$$

$$\dot{\xi}(t) = f(\xi(t),u(t)), \qquad t_0 \le t \le t_f \qquad \text{(plant model)} \tag{1.1c}$$

$$\underline{g} \le g(\xi(t),u(t)) \le \overline{g}, \qquad t_0 \le t \le t_f \qquad \text{(path constraints)}, \tag{1.1d}$$

which consists of an objective function that is minimised (1.1a), boundary conditions which define the initial and final states the trajectory must traverse between (1.1b), the plant model which determines the dynamics of the system (1.1c), and path constraints which constrain the state and controls within specified boundaries (1.1d).

Once the optimal control problem has been defined, it can be solved numerically using either

[2] As a note on terminology, the terms RHP and MPC are often used interchangeably in literature. However, although sharing similar theories, techniques and formulations, this thesis makes a distinction such that the solution of an RHP is tracked by a lower-level controller which acts as an intermediary between the RHP and the actuator, while the solution of an MPC is directly used by the actuators.

direct methods or indirect methods [14]. Direct methods discretise the continuous-time optimal control problem in time, after which the problem can be transcribed as an optimisation problem that can be solved by using a suitable optimisation algorithm, such as a quadratic program (QP), sequential quadratic program (SQP) or interior-point (IP) method. Indirect methods instead solve the original optimal control problem by finding the necessary conditions for optimality, which results in a boundary value problem that can be solved numerically. While some works use indirect methods to solve the offline optimal racing line problem [15], direct methods are more popular in practice due to being more robust, implements path inequalities more easily, and is an easier problem to formulate [16].

There are two main categories of discretisation methods for direct methods [17]:

- Sequential: Only the control variables are discretised. The state variables appearing in the objective function and path constraints become a function of the control trajectory defined by the plant model dynamics. This leads to a smaller but denser optimisation problem. More complex derivatives need to be computed arising from the chain rule relationships between the state and control variables. The most common sequential method is the direct single shooting method.

- Simultaneous: Both the control variables and state variables are discretised. Equality constraints are enforced between neighbouring state variables to enforce the plant dynamics. This leads to a larger but sparser optimisation problem which modern optimisation solvers such as IPOPT [18] can efficiently solve. The most common simultaneous methods include the direct multiple shooting method and the direct collocation method.

RHP is a type of optimal control which recomputes an optimal control trajectory at each time step. This allows the open-loop optimal control formulation to inherit closed-loop properties such as being robust to disturbances, state estimation errors and model errors. In the conversion between optimal control problem and optimisation problem, a number of works utilise optimal control libraries to perform the transcription internally using libraries such as ACADO [9] or FORCES Pro [10]. Other works perform this transcription manually [19].

Another important classification of RHP formulations is whether the plant model is linear or nonlinear, as this significantly impacts the computational burden of the problem. Linear systems can be formulated as a QP, which are significantly easier to solve compared to nonlinear systems which must be formulated as a nonlinear program (NLP). As vehicle dynamics are highly nonlinear, a natural route is to retain the nonlinearities and formulate it as a nonlinear RHP which is solved using an NLP [10]. An alternative approach to improve the real-time feasibility of the RHP is to linearise the vehicle dynamics and nonlinear constraints at each time step to achieve a linear time-varying (LTV) RHP, which allows the problem to be formulated as a QP [7, 8, 20]. One work simplified the RHP even further by discarding the path constraints, resulting in a least-squares problem which could be solved using linear algebra [19]. Alternatively, a real-time iterative (RTI) scheme can be used, which achieves fast convergence by performing

only a single optimisation step of a NLP by hot-starting from the time shifted previous optimal solution [21].

1.3.3 Vehicle modelling

In autonomous driving, the two most common plant models used for RHP are the kinematic bicycle model [22, 23] and dynamic bicycle model [9, 10, 19]. The kinematic bicycle model is simpler and therefore computationally less expensive, and is accurate at low velocities. However, particularly in the autonomous racing context, the dynamic bicycle model is often preferred due to its superior accuracy at higher velocities and accelerations. The most important factor in traditional race car dynamics is the acceleration of the vehicle, and the tyre forces that achieve these. An important property of tyre forces is that the total combined longitudinal and latitudinal tyre forces is constrained, such that the larger the latitudinal tyre force, the smaller the longitudinal tyre force that can be achieved. This relationship is often referred to as the tyre friction ellipse, and it is well known that to make the most of the vehicle's performance, the tyre forces must be as close to the friction ellipse boundary as possible [24]. Therefore, the dynamic bicycle model's ability to capture these important factors is expected to contribute to superior racing performance.

However, the dynamic bicycle model is ill-defined at low velocities due to velocity being on the denominator of the slip angle equation [25]. Techniques used to circumvent this issue include using a hybrid kinematic-dynamic model which chooses which model to prioritise based on the vehicle's velocity [26], or adding a small constant to the culprit fraction's denominator [27]. Another issue with the dynamic bicycle model is that the tyre dynamics are highly nonlinear, particularly at the boundaries of the friction ellipse. This can lead to poor model approximations if linearisation techniques are used. Proposed methods to prevent this issue for LTV-RHP formulations typically involve imposing constraints on the slip angles to avoid the highly nonlinear region [7].

For autonomous racing, another popular modelling technique is to use curvilinear coordinates as opposed to Cartesian coordinates [10, 27], in which coordinates are defined relative to the track path as opposed to a global coordinate system. This is a more intuitive formulation which reflects the perception of real drivers, allows for a more natural expression for the progress made along the track, and allows track boundaries to be expressed as a simple box constraint on the lateral deviation of the path.

In addition to traditional vehicle modelling, recent works in autonomous racing have investigated learning-based techniques [28, 29]. In this formulation, data about the vehicle dynamics obtained while the vehicle is driving is used to improve the plant model in real time.

Chapter 2

Plant Model

In this chapter, the plant models used for the RHP are explored. There are two primary vehicle models that are considered: the kinematic bicycle model and the dynamic bicycle model. The kinematic bicycle model is expected to be computationally less expensive, while the dynamic bicycle model is expected to be more accurate particularly at higher vehicle speeds by considering important tyre dynamics, and thus achieving better tracking performance. Modifications on these vehicle models to improve the RHP performance are also presented.

2.1 Kinematic bicycle model

The kinematic bicycle model as depicted in Figure 2.1 is derived by combining the pair of front and rear wheels together into a rigid single-track model, then assuming that the wheels travel in the direction that they are pointed, v_f and v_r. The vehicle is defined by its mass m and dimensions from its center of gravity (CoG) l_f and l_r. The state is defined as the coordinates (X, Y) of the CoG, the vehicle yaw θ, and the velocity v of the CoG, all relative to a global coordinate frame, as well as the front steering wheel angle δ. The control variables include the acceleration a and steering rate $\Delta\delta$ of the vehicle. This Δu control formulation aims to provide a regularising effect on the vehicle state for a suitably designed objective function that penalises Δu [30].

The vehicle model is defined as the following nonlinear state equations

$$\dot{X} = v\cos(\theta + \alpha) \tag{2.1a}$$

$$\dot{Y} = v\sin(\theta + \alpha) \tag{2.1b}$$

$$\dot{\theta} = \frac{v}{l_r}\sin(\alpha) \tag{2.1c}$$

$$\dot{v} = a \tag{2.1d}$$

$$\dot{\delta} = \Delta\delta, \tag{2.1e}$$

where the slip angle of the vehicle α is defined as

$$\alpha = \arctan\left(\frac{l_r}{l_r + l_f}\tan(\delta)\right). \tag{2.2}$$

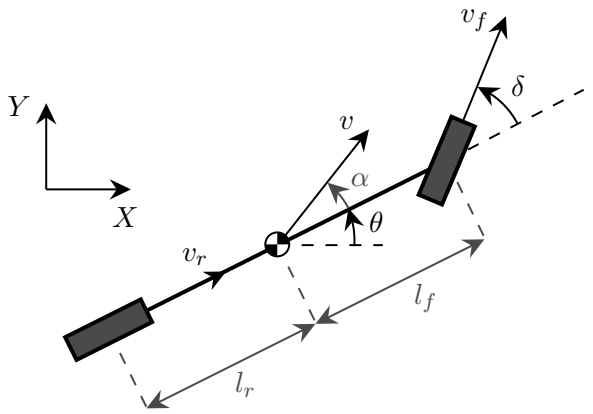

Figure 2.1: Kinematic bicycle model

2.2 Dynamic bicycle model

The dynamic bicycle model as depicted in Figure 2.2 is a more complex single-track model which is derived by considering tyre forces and kinetic equations. Longitudinal tyre forces arising from slip from the front tyres, as well as aerodynamic forces, are assumed to be negligible. In addition to the vehicle parameters introduced in Section 2.1, the dynamic bicycle model also defines the vehicle's yaw moment of inertia I. As opposed to the kinematic bicycle model, the dynamic bicycle model breaks up the state velocity into the longitudinal $\dot{x}$ and latitudinal $\dot{y}$ components, defined in the vehicle's inertial frame, and also introduces the yaw rate ω as a state variable. The control variables remain similar, but uses the driving force F_x, or rear wheel longitudinal tyre force, instead of acceleration.

The vehicle model is defined as the following nonlinear state equations

$$\dot{X} = \dot{x}\cos(\theta) - \dot{y}\sin(\theta) \tag{2.3a}$$
$$\dot{Y} = \dot{x}\sin(\theta) + \dot{y}\cos(\theta) \tag{2.3b}$$
$$\dot{\theta} = \omega \tag{2.3c}$$
$$\ddot{x} = \frac{1}{m}(F_x - F_{cf}\sin(\delta) + m\dot{y}\omega) \tag{2.3d}$$
$$\ddot{y} = \frac{1}{m}(F_{cr} + F_{cf}\cos(\delta) - m\dot{x}\omega) \tag{2.3e}$$
$$\dot{\omega} = \frac{1}{I}(l_f F_{cf}\cos(\delta) - l_r F_{cr}) \tag{2.3f}$$
$$\dot{\delta} = \Delta\delta. \tag{2.3g}$$

The front and rear lateral tyre forces are defined using the Pacejka Magic Formula [31], where the parameters A, B, C, D and E define the shape of the tyre curve. The shape of this curve can be seen in Figure 3.2, and is defined by

$$F_{c(\cdot)} = F_{z(\cdot)} D \sin\Big(C \arctan\Big(B\alpha_{(\cdot)} - E\left(B\alpha_{(\cdot)} - \arctan\Big(B\alpha_{(\cdot)}\Big)\right)\Big)\Big), \tag{2.4}$$

where the slip angles of the tyres are defined as

$$\alpha_f = \delta - \arctan\left(\frac{\dot{y} + l_f\omega}{\dot{x}}\right) \tag{2.5a}$$

$$\alpha_r = -\arctan\left(\frac{\dot{y} - l_r\omega}{\dot{x}}\right). \tag{2.5b}$$

The weight distribution between front and rear tyres is estimated using a simple constant moment balance about the centre of gravity. Note that this simplification ignores load transfer and aerodynamic downforce effects typically experienced by racecars.

$$F_{zf} = \frac{l_r}{l_f + l_r} mg \tag{2.6a}$$

$$F_{zr} = \frac{l_f}{l_f + l_r} mg \tag{2.6b}$$

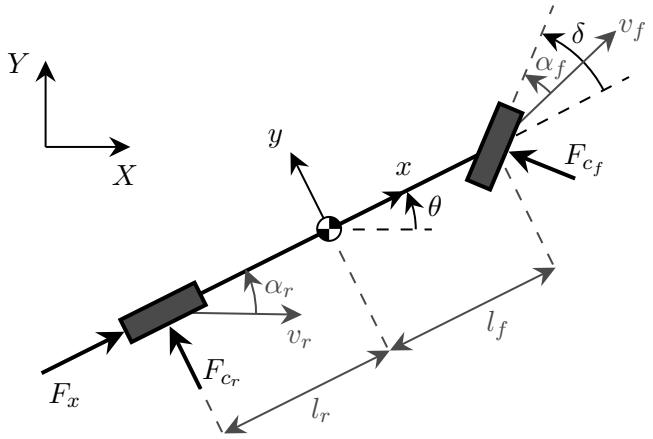

Figure 2.2: Dynamic bicycle model

2.2.1 Low-velocity compensation

The dynamic bicycle model is known to have numerical issues at low velocities as the longitudinal velocity $\dot{x}$ appears on the denominator for the equations for the slip angles (2.5), causing tyre forces to become singular [25]. Although in racing scenarios the vehicle velocity ideally avoids this low-velocity region, it is still important for starting the car from its initial stationary state, as well as navigating around sharp corners at lower speeds [26].

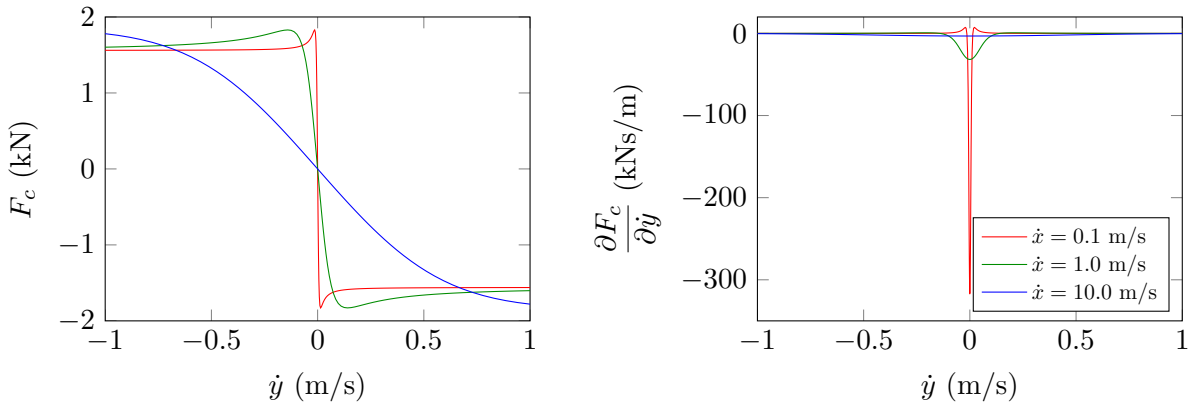

Figure 2.3: Lateral tyre force (left) and its partial derivative with respect to lateral velocity (right) at varying longitudinal velocities when $\omega = 0$.

The singular nature of tyre forces at low velocities manifests as a sharp step-like function as seen in Figure 2.3. This results in a similar issue to the exploding gradient problem found in machine learning literature [32], where the significantly larger gradient can result in instability in the gradient descent algorithm used for optimisation.

To circumvent this, the following modification was made to the longitudinal velocity in the slip angle formula denominator in (2.5) to avoid the singularity.

$$\dot{x}' = \dot{x} + \dot{x}_{min} e^{-\dot{x}/\dot{x}_{min}} \tag{2.7}$$

As opposed to a constant offset or piecewise linear modification, this formula was chosen as it is smooth, strictly increasing, and the additional offset term decays at higher velocities. A floor of $\dot{x}_{min} = 5\,\text{m/s}$ was found to work well.

2.3 Curvilinear coordinate system

Both the kinematic and dynamic bicycle models were converted to curvilinear coordinates to define the vehicle state relative to the track centreline, as depicted in Figure 2.4. The curvilinear coordinate equations of motion are given in (2.8) for the kinetmatic bicycle model (left) and dynamic bicycle model (right), where s is the arclength travelled along the path, n is the lateral deviation from the path, μ is the angular deviation from the path, and $\kappa(s)$ is the curvature of the path. These equations are derived by assuming that $\kappa(s)$ is piecewise constant [33], so any

derivatives of these equations will similarly treat $\kappa(s)$ as a constant.

$$\dot{s} = \frac{v\cos(\mu+\alpha)}{1-n\kappa(s)}, \qquad \dot{s} = \frac{\dot{x}\cos(\mu) - \dot{y}\sin(\mu)}{1-n\kappa(s)} \tag{2.8a}$$

$$\dot{n} = v\sin(\mu+\alpha), \qquad \dot{n} = \dot{x}\sin(\mu) + \dot{y}\cos(\mu) \tag{2.8b}$$

$$\dot{\mu} = \frac{v}{l_r}\sin(\alpha) - \frac{v\cos(\mu+\alpha)}{1-n\kappa(s)}\kappa(s), \qquad \dot{\mu} = \omega - \frac{\dot{x}\cos(\mu) - \dot{y}\sin(\mu)}{1-n\kappa(s)}\kappa(s) \tag{2.8c}$$

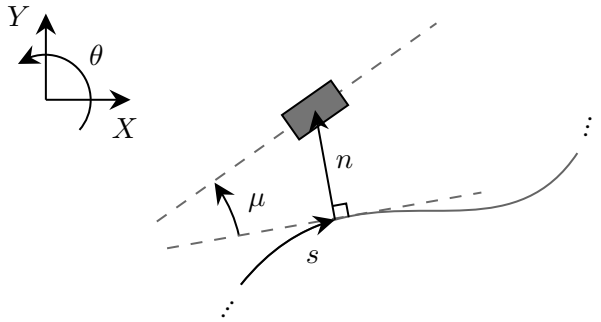

Figure 2.4: Curvilinear coordinate system definition

2.3.1 Path formulation

To represent the state in curvilinear coordinates, the track must be expressed as a continuous function. Smooth paths are desired so that jerky motions and unnecessarily high accelerations are avoided [34]. A common way a smooth continuous path is represented for trajectory planning is by using splines. In particular, a cubic spline is able to achieve a C^2 continuity, which is required for the curvature $\kappa(s)$ to be continuous. Using Bezier curves to define the cubic segments, the path can be represented as a parametric piecewise spline function $\{X_p, Y_p\}$ with N_p segments of the form

$$\begin{cases} X_{p,i}(\sigma) = a_{x,i}(1-\sigma)^3 + 3b_{x,i}(1-\sigma)^2\sigma + 3c_{x,i}(1-\sigma)\sigma^2 + d_{x,i}\sigma^3 \\ Y_{p,i}(\sigma) = a_{y,i}(1-\sigma)^3 + 3b_{y,i}(1-\sigma)^2\sigma + 3c_{y,i}(1-\sigma)\sigma^2 + d_{y,i}\sigma^3 \end{cases}, \quad \sigma \in [0,1]. \tag{2.9}$$

To solve for the the spline coefficients a_i, b_i, c_i and d_i for a set of points p_i, a system of linear equations can be derived through C^0, C^1 and C^2 continuity conditions between neighbouring segments. However, naively solving the system requires the inversion of a large $4N_p \times 4N_p$ matrix, which is undesirable in situations when the spline needs to be computed online. Instead, a more efficient method inspired by [35] is presented.

The following derivation will consider fitting N_p points to an arbitrary spline, and is a technique that is applied for both the X_p and Y_p parametric spline components independently. As the track is assumed to be a closed loop, for the sake of brevity, the set of points and

spline segments are defined to be periodic by abuse of notation such that $p_i = p_{i+N_p}$, $X_{p,i}(t) = X_{p,i+N_p}(t)$ and $Y_{p,i}(t) = Y_{p,i+N_p}(t)$.

Firstly, the beginning and end of each spline segment is defined to be between two neighbouring points p_i and p_{i+1}. The Bezier cubic expression allows us to immediately define two of the spline coefficients

$$a_i = p_i, \qquad i = 0, \dots, N_p - 1 \tag{2.10a}$$

$$d_i = p_{i+1}, \qquad i = 0, \dots, N_p - 1. \tag{2.10b}$$

Equations obtained from the C^1 and C^2 conditions give the following conditions

$$d_i + c_{i+1} = 2p_i, \qquad i = 0, \dots, N_p - 1 \tag{2.11a}$$

$$c_i - 2d_i + 2c_{i+1} - d_{i+1} = 0, \qquad i = 0, \dots, N_p - 1. \tag{2.11b}$$

Equations (2.11) are combined to obtain

$$c_{i-1} + 4c_i + c_{i+1} = 4p_i + 2p_{i+1}, \qquad i = 0, \dots, N_p - 1. \tag{2.12}$$

Equation (2.12) can be expressed as the following tridiagonal matrix with periodic boundary conditions

$$\begin{bmatrix} 4 & 1 & & & & & 1 \\ 1 & 4 & 1 & & & & \\ & 1 & 4 & 1 & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & 1 & 4 & 1 & \\ & & & & 1 & 4 & 1 \\ 1 & & & & & 1 & 4 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_1 \\ \vdots \\ c_{N_p-3} \\ c_{N_p-2} \\ c_{N_p-1} \end{bmatrix} = \begin{bmatrix} 4p_0 + 2p_1 \\ 4p_1 + 2p_2 \\ 4p_2 + 2p_3 \\ \vdots \\ 4p_{N_p-3} + 2p_{N_p-2} \\ 4p_{N_p-2} + 2p_{N_p-1} \\ 4p_{N_p-1} + 2p_0 \end{bmatrix}. \tag{2.13}$$

Systems of equation involving diagonally dominant tridiagonal matrices can be solved efficiently using the Thomas algorithm [36]. For tridiagonal matrices with periodic boundary conditions such as (2.13), the Sherman-Morrison algorithm can first be used to modify the problem to be solvable using the Thomas algorithm [37]. Once the system has been solved, the remaining coefficients d_i can be solved using (2.11a).

Once the spline has been solved, it is reparametrised as a function of s by interpolating the spline at evenly spaced arc lengths, and refitting a spline to the interpolated points [38]. The parameterisation error improves the smaller the arc length step size, but at the cost of overfitting the curve.

Once the final arc-length parameterised spline is defined, the curvature of the path is therefore

given by

$$\kappa(s) = \frac{\dot{X}_p(s)\ddot{Y}_p(s) - \ddot{X}_p(s)\dot{Y}_p(s)}{\left(\dot{X}_p(s)^2 + \dot{Y}_p(s)^2\right)^{3/2}}. \tag{2.14}$$

2.3.2 Coordinate transform

As the state estimate $\hat{\xi}$ from the SLAM algorithm is given in Cartesian coordinates, a coordinate transform will be required to transform it into curvilinear coordinates. To do this, the closest point on the path to the vehicle needs to be found to find $\hat{s}$. The other two states $\hat{n}$ and $\hat{\mu}$ can then be found as

$$\hat{s} = \arg\min_s D(s), \quad D(s) = \left(\hat{X} - X_p(s)\right)^2 + \left(\hat{Y} - Y_p(s)\right)^2 \tag{2.15a}$$

$$\hat{n} = \frac{-\dot{Y}_p(\hat{s})\left(\hat{X} - X_p(\hat{s})\right) + \dot{X}_p(\hat{s})\left(\hat{Y} - Y_p(\hat{s})\right)}{\sqrt{\dot{X}_p(\hat{s})^2 + \dot{Y}_p(\hat{s})^2}} \tag{2.15b}$$

$$\hat{\mu} = \hat{\theta} - \operatorname{atan2}\left(\dot{Y}_p(\hat{s}), \dot{X}_p(\hat{s})\right). \tag{2.15c}$$

Common methods to solve the closest-point-on-path problem (2.15a) typically involve some variation of the Newton-Raphson root finding method [39].

$$\hat{s}^*_{i+1} = \hat{s}^*_i - \frac{D'(\hat{s}^*_i)}{D''(\hat{s}^*_i)} \tag{2.16}$$

The main issues with this approach are slow convergence or non-global optimality, typically due to bad initialisations [39]. However, due to the nature of our application, the previous solution for $\hat{s}$ will always be a good initialisation candidate.

2.4 Actuator modelling

The RHP control outputs interact with lower-level PID controllers, which in turn interact with the inverter, braking, and steering actuators. These factors manifest as a transient response in the vehicle's velocity and steering angle. In applications such as chemical plants, these transient responses are typically trivial due to the long prediction horizons used relative to the response times. However, in the autonomous racing context, the transient response represents a significant time delay which can lead to controller instability if unaccounted for [40].

To account for this time delay component, the transient responses were modelled and incorporated into the vehicle dynamics. While it is possible to incorporate a full powertrain model to capture these dynamics, it would increase the complexity of the problem. Instead, a simple first order transient response was used, which is able to capture the important characteristics of the transient delay characteristics. The RHP controls were subsequently defined as the target

setpoints $u = [\, v_t \;\; \delta_t \,]^T$, and the dynamics were modified to reflect this as

$$\Delta\delta = K_\delta(\delta_t - \delta) \tag{2.17a}$$

$$F_x = mK_v(v_t - \dot{x}), \tag{2.17b}$$

where K_δ and K_v represent the equivalent gains of a proportional controller.

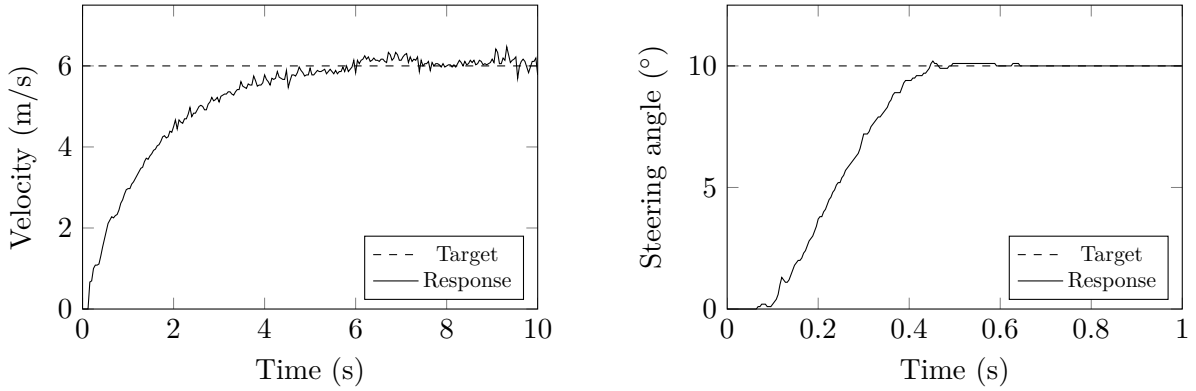

Figure 2.5: Experimental data of the velocity (left) and steering angle (right) step responses of M19-D.

2.5 Summary

A final vehicle model can be obtained by combining the equations shown in the previous sections. For example, the vehicle model for a dynamic bicycle model in curvilinear coordinates with transient actuator responses is shown in (2.18), which was ultimately used in the final RHP formulation. The corresponding state vector is $\xi = [\, s \;\; n \;\; \mu \;\; \dot{x} \;\; \dot{y} \;\; \omega \;\; \delta \,]^T$, and the control vector is $u = [\, v_t \;\; \delta_t \,]^T$.

$$\dot{s} = \frac{\dot{x}\cos(\mu) - \dot{y}\sin(\mu)}{1 - n\kappa(s)} \tag{2.18a}$$

$$\dot{n} = \dot{x}\sin(\mu) + \dot{y}\cos(\mu) \tag{2.18b}$$

$$\dot{\mu} = \omega - \frac{\dot{x}\cos(\mu) - \dot{y}\sin(\mu)}{1 - n\kappa(s)}\kappa(s) \tag{2.18c}$$

$$\ddot{x} = \frac{1}{m}(mK_v(v_t - \dot{x}) - F_{cf}\sin(\delta) + m\dot{y}\omega) \tag{2.18d}$$

$$\ddot{y} = \frac{1}{m}(F_{cr} + F_{cf}\cos(\delta) - m\dot{x}\omega) \tag{2.18e}$$

$$\dot{\omega} = \frac{1}{I}(l_f F_{cf}\cos(\delta) - l_r F_{cr}) \tag{2.18f}$$

$$\dot{\delta} = K_\delta(\delta_t - \delta) \tag{2.18g}$$

Chapter 3

Optimal Planning for Racing

This chapter presents the formulations of optimal control problems used for motion planning. The first is the RHP problem, in which a finite horizon optimal control problem is solved online at each time step in a receding horizon fashion to calculate a trajectory which optimises for a given objective. The second is the optimal racing line problem, where a periodic time-optimal reference trajectory is computed offline to provide a reference trajectory for the RHP. In the following sections, $\|\cdot\|_W^n$ represents the W-weighted n-norm function.

3.1 Receding horizon planning

Consider the nonlinear system $f\colon \mathbb{R}^{n_\xi} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_\xi}$ defined by the models presented in Chapter 2

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

where $\xi \in \mathbb{R}^{n_\xi}$ and $u \in \mathbb{R}^{n_u}$ represent the state and control vectors respectively. The RHP is formulated as a reference tracking optimal controller with the following objective function

$$J(\xi, u, t_0) = \|\xi(t_f) - \xi_r(t_f)\|_{Q_f}^2 + \int_{t_0}^{t_f} \|\xi(\tau) - \xi_r(\tau)\|_Q^2 + \|e(\tau)\|_R^2 \, d\tau, \tag{3.2}$$

where $\xi_r \in \mathbb{R}^{n_\xi}$ is the state reference trajectory, $e(t) = [\, F_x(t) \;\; \Delta\delta(t) \,]^T$ represents the effort exerted by the vehicle, and Q, $Q_f \in \mathbb{R}^{n_\xi \times n_\xi}$ and $R \in \mathbb{R}^{n_u \times n_u}$ are state, terminal state, and control weight positive semi-definite diagonal matrices respectively. The objective function is constructed to penalise deviations from the reference trajectory, while an effort minimising regularisation term is included to have a smoothing effect on the solution trajectory. A terminal penalty is included to help enforce stability [41].

Path constraints $g(\xi(t), u(t))\colon \mathbb{R}^{n_\xi} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_g}$ consisting of a combination of state, control and functional constraints shown in (3.3) are enforced. Upper and lower bounds on quantities

are expressed as $\overline{(\cdot)}$ and $\underline{(\cdot)}$ respectively.

$$\underline{n} \leq n(t) \leq \overline{n} \tag{3.3a}$$
$$\underline{\delta} \leq \delta(t) \leq \overline{\delta} \tag{3.3b}$$
$$0 \leq \dot{x}(t) \tag{3.3c}$$
$$\underline{\Delta\delta} \leq \Delta\delta(t) \leq \overline{\Delta\delta} \tag{3.3d}$$
$$\left(\frac{F_x(t)}{m\overline{a_l}}\right)^2 + \left(\frac{F_{cr}(t)}{m\overline{a_c}}\right)^2 \leq 1 \tag{3.3e}$$

These constraints describe the track boundary constraints (3.3a), steering angle limits (3.3b), prevention from backwards movement (3.3c), slew rate of the steering angle (3.3d), and tyre friction ellipse constraints (3.3e). If actuator transient response as described in Section 2.4 is modelled, then (3.3b) and (3.3c) can be replaced with constraints on δ_t and v_t respectively.

Overall, the RHP problem is formulated as

$$\min_{\xi,u} \quad J(\xi, u, t_0) \tag{3.4a}$$
$$\text{subj. to} \quad \xi(t_0) = \hat{\xi} \tag{3.4b}$$
$$\dot{\xi}(t) = f(\xi(t), u(t)), \qquad t_0 \leq t \leq t_f \tag{3.4c}$$
$$\underline{g} \leq g(\xi(t), u(t)) \leq \overline{g}, \qquad t_0 \leq t \leq t_f. \tag{3.4d}$$

Once the RHP (3.4) has been discretised using a method described in Chapter 4, it is solved in a receding horizon fashion. This involves repeatedly solving the RHP at each discrete time step, where at each iteration, the prediction horizon $[t_0, t_f]$ is shifted forwards by a time step $[t_0 + \Delta t, t_f + \Delta t]$, resulting in an updated state estimate $\hat{\xi}$ and reference trajectory ξ_r that the RHP is solved from. The controls at the first time step of the solved optimal control trajectory $u_0^*(i)$ are then used as target setpoints for the lower-level PID controllers before the optimal trajectory is updated in the next iteration. This algorithm is visualised in Figure 3.1.

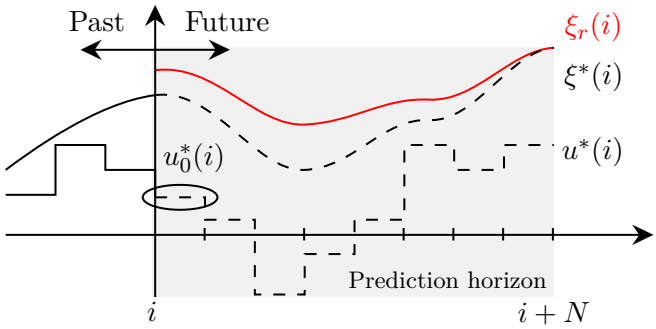

Figure 3.1: Receding horizon planning algorithm.

3.1.1 Soft constraints

It can be advantageous to represent constraints involving state variables such as (3.3a) and (3.3e) as soft constraints to allow for an additional degree of flexibility in situations when the problem would otherwise be infeasible, and no solution exists which would satisfy the constraints [41]. To do this, slack variables $\epsilon(t) = [\,\epsilon_n(t)\;\; \epsilon_a(t)\,]^T$ are introduced to modify the constraints such that

$$\underline{n} \leq n(t) + \epsilon_n(t) \leq \overline{n} \tag{3.5a}$$

$$\left(\frac{F_x(t))}{m\overline{a_l}}\right)^2 + \left(\frac{F_{cr}(t)}{m\overline{a_c}}\right)^2 + \epsilon_a(t) \leq 1. \tag{3.5b}$$

To ensure that in situations when the original problem would be feasible that the optimal solution can be recovered, ∞-norm penalties on the slack variables weighted by $q_{(\cdot)} \in \mathbb{R}_+$ are added to the cost function [42].

$$J_\epsilon(\xi, u, \epsilon, t_0) = J(\xi, u, t_0) + q_n\|\epsilon_n(t)\|^\infty + q_a\|\epsilon_a(t)\|^\infty \tag{3.6}$$

The ∞-norm slack variable penalty $q_n\|\epsilon_n(t)\|^\infty + q_a\|\epsilon_a(t)\|^\infty$ can be treated similarly to the Chebyshev approximation problem or minmax approximation problem, and formulated in the following linear program (LP)-like form [43]

$$\min_{\epsilon} \quad \|\epsilon\|_q^1 = q_n\epsilon_n + q_a\epsilon_a \tag{3.7a}$$

$$\text{subj. to} \quad n(t) - \epsilon_n \leq \overline{n}, \qquad t_0 \leq t \leq t_f \tag{3.7b}$$

$$n(t) + \epsilon_n \geq \underline{n} \qquad t_0 \leq t \leq t_f \tag{3.7c}$$

$$\left(\frac{F_x(t))}{m\overline{a_l}}\right)^2 + \left(\frac{F_{cr}(t)}{m\overline{a_c}}\right)^2 - \epsilon_a \leq 1 \qquad t_0 \leq t \leq t_f \tag{3.7d}$$

$$\epsilon_n \geq 0 \tag{3.7e}$$

$$\epsilon_a \geq 0, \tag{3.7f}$$

where $\epsilon_n, \epsilon_a \in \mathbb{R}^+$ are converted to scalar values, and $q \in \mathbb{R}^{n_\epsilon}$ is the slack variable weight vector. This presents a secondary advantage of the ∞-norm penalty, where by using the reformulation shown in (3.7), the slack variables are condensed from a continuous function to a scalar variable. Once the system is discretised, this corresponds to a significant reduction in the number of optimisation variables, where for each constraint, instead of requiring N slack variables for every time step, a single slack variable can be used for the entire horizon.

After introducing soft constraints, the optimal control problem becomes

$$\min_{\xi,u,\epsilon} \quad J_\epsilon(\xi, u, \epsilon, t_0) = J(\xi, u, t_0) + \|\epsilon\|_q^1 \tag{3.8a}$$

$$\text{subj. to} \quad \xi(t_0) = \hat{\xi} \tag{3.8b}$$

$$\dot{\xi}(t) = f(\xi(t), u(t)), \qquad t_0 \leq t \leq t_f \tag{3.8c}$$

$$\underline{g} \leq g_\epsilon(\xi(t), u(t), \epsilon) \leq \overline{g}, \qquad t_0 \leq t \leq t_f \tag{3.8d}$$

$$\epsilon \geq 0, \tag{3.8e}$$

where g_ϵ is the soft constraint-modified constraint function.

3.1.2 Friction ellipse

The friction ellipse constraint shown in (3.3e) works best for nonlinear discretisation using the dynamic bicycle model. Alternative formulations for the friction ellipse for other modelling and discretisation schemes are shown in the following sections.

Kinematic bicycle model

The constraint shown in (3.3e) requires the tyre forces to be modelled. If a kinematic bicycle model is used, an estimation of these forces is required. This can be done through the following approximation of longitudinal and lateral accelerations [22]

$$\underline{a_l} \leq \frac{F_x(t)}{m} \leq \overline{a_l} \tag{3.9a}$$

$$\underline{a_c} \leq \frac{v(t)^2 \delta(t)}{l_f + l_r} \leq \overline{a_c}. \tag{3.9b}$$

As the lateral acceleration calculated in (3.9b) is just an approximation, the bounds on a_c must be set more conservatively compared to the true vehicle capabilities.

Linearisation

Another instance when the friction ellipse needs to be reformulated is in linear discretisation schemes. In this case, it can be advantageous to linearise the ellipse beforehand to create a convex set using N_a linearised constraints [8]. As the tyre curve is highly nonlinear at the boundaries of the friction ellipse, an additional constraint on the front and rear slip angles of the tyres is enforced to ensure the vehicle remains within the linear regions of the lateral tyre force function [7].

These constraints are visualised in Figure 3.2 and formulated as

$$A_i \begin{bmatrix} F_{cr}(t) \\ F_x(t) \end{bmatrix} - b_i - \epsilon_a \leq 0, \quad i = 1, \ldots, N_a \tag{3.10a}$$

$$\alpha_f(t) - \epsilon_f \leq \overline{\alpha_f} \tag{3.10b}$$

$$\alpha_f(t) + \epsilon_f \geq \underline{\alpha_f} \tag{3.10c}$$

$$\alpha_r(t) - \epsilon_r \leq \overline{\alpha_r} \tag{3.10d}$$

$$\alpha_r(t) + \epsilon_r \geq \underline{\alpha_r} \tag{3.10e}$$

$$\epsilon_a \geq 0, \quad \epsilon_f \geq 0, \quad \epsilon_r \geq 0. \tag{3.10f}$$

The slip angle constraints are also formulated as soft constraints. Therefore, additional slack variables are added such that $\epsilon(t) = [\, \epsilon_n \;\; \epsilon_a \;\; \epsilon_f \;\; \epsilon_r \,]^T$.

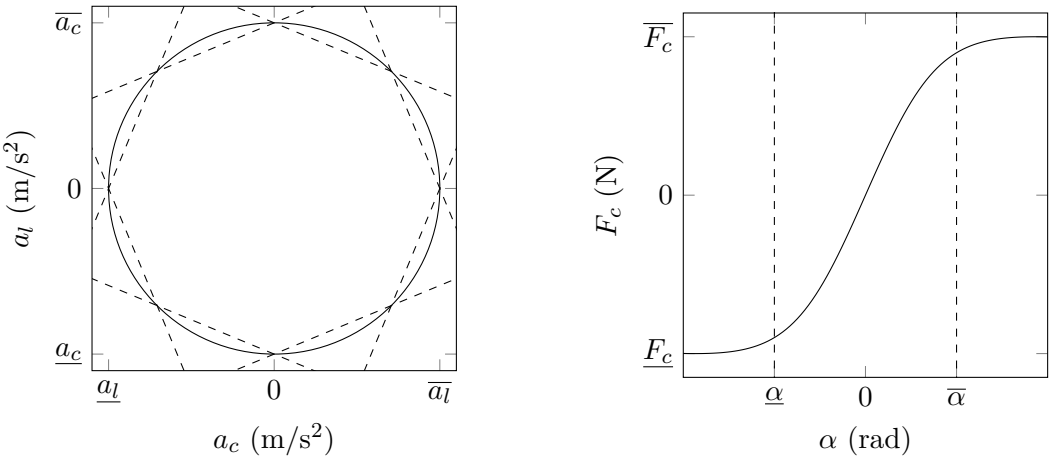

Figure 3.2: Linearised friction ellipse constraints (left) and lateral tyre force modelled using the Pacejka Magic Formula with slip angle constraints to ensure the vehicle operates in the linear region of the tyre force (right).

3.2 Time optimal planning

The RHP problem only considers a small horizon of the entire track. This may lead to suboptimal solution trajectories where there may be important information about the track beyond the considered horizon, such as an upcoming hairpin. To circumvent the issue of the RHP being blind to the track past the current time horizon, assuming that the track layout is known beforehand, an optimal racing trajectory can be precomputed offline, and act as the reference trajectory the RHP tracks [10].

To do this, the problem is reformulated from time-domain to spatial-domain, so that time t can be treated as a variable optimisation parameter. Instead, s now defines the horizon the

problem is optimised over, and is therefore no longer an optimsation variable and can be dropped from the state vector ξ. As the time optimal planner is run offline, accuracy is more important than computational complexity, so the dynamic bicycle model presented in (2.18) is used as the plant model. Overall, the state vector is redefined as $\xi = [\, n \;\; \mu \;\; \dot{x} \;\; \dot{y} \;\; \omega \;\; \delta \,]^T$. Note that time is not included as a state variable, as it can be expressed purely in terms of the existing state variables as $\Delta t = \Delta s / \dot{s}$, where $\dot{s}$ is given by (2.8a).

The objective function is reformulated to minimise the time taken to navigate the track, while retaining regularising terms on both the state and control variables

$$J_\epsilon(\xi, u, \epsilon) = \int_0^L \frac{1}{\dot{s}} + \|\xi(s)\|_Q^2 + \|e(s)\|_R^2 \, ds + \|\epsilon\|_q^1, \tag{3.11}$$

where L is the total length of the track. The plant dynamics are reformulated in terms of s instead of t by using the chain rule

$$\dot{\xi}(s) = \frac{d\xi}{ds} = \frac{d\xi}{dt}\frac{dt}{ds} = \frac{1}{\dot{s}} f(\xi(s), u(s)). \tag{3.12}$$

The plant model Jacobians are therefore given by

$$\frac{\partial}{\partial \xi}(\dot{\xi}(s)) = -\frac{1}{\dot{s}^2} f(\xi(s), u(s)) \frac{\partial \dot{s}}{\partial \xi} + \frac{1}{\dot{s}} \frac{\partial f}{\partial \xi} \tag{3.13a}$$

$$\frac{\partial}{\partial u}(\dot{\xi}(s)) = \frac{1}{\dot{s}} \frac{\partial f}{\partial u}. \tag{3.13b}$$

As the track is closed and periodic, a final constraint is enforced to ensure periodicity of the optimised path. All path constraints $g_\epsilon(\xi(s), u(s), \epsilon)$ are identical to those formulated in Section 3.1. Overall, the optimal racing line problem is formulated as

$$\min_{\xi, u, \epsilon} \quad J_\epsilon(\xi, u, \epsilon) \tag{3.14a}$$

$$\text{subj. to} \quad \xi(0) = \xi(L) \tag{3.14b}$$

$$\dot{\xi}(s) = \frac{1}{\dot{s}} f(\xi(s), u(s)), \qquad 0 \le s \le L \tag{3.14c}$$

$$\underline{g} \le g_\epsilon(\xi(s), u(s), \epsilon) \le \overline{g}, \qquad 0 \le s \le L \tag{3.14d}$$

$$\epsilon \ge 0. \tag{3.14e}$$

Chapter 4

Discretisation

Once the optimal control problem has been defined, it needs to be discretised to be in a form able to be solved by optimisers. This involves discretising the time horizon into N discrete intervals spaced evenly by a time step Δt to achieve a finite number of optimisation variables that can be solved for. Multiple discretisation methods were explored, including various linear and nonlinear schemes. For the notation that will be used, unless otherwise stated, a plain variable ξ refers to a trajectory or sequence of vectors, a subscript ξ_k refers to the k-th vector in the trajectory, and an asterisk $\xi^*(i)$ refers to the optimal trajectory evaluated at the i-th time step. Unless otherwise specified, a vector of vectors denoted as $v = [\, v_1 \;\; v_2 \;\; \dots \;\; v_N \,]^T$ is an $\mathbb{R}^{Nn_v}$ column stacked vector.

4.1 Successive linearisation

By linearising the nonlinear dynamics of the vehicle model, the problem can be solved more easily by QP solvers at the cost of losing important nonlinearities of the original plant model. To perform this transcription method, the RHP is first discretised, which introduces the discrete state trajectory vector $\xi = [\, \xi_1 \;\; \xi_2 \;\; \dots \;\; \xi_N \,]^T$ and control trajectory vector $u = [\, u_0 \;\; u_1 \;\; \dots \;\; u_{N-1} \,]^T$, and discretises the objective function as

$$J_\epsilon(\xi, u, \epsilon) = \|\xi_N - \xi_{r,N}\|^2_{Q_f} + \sum_{k=1}^{N-1} \|\xi_k - \xi_{r,k}\|^2_Q + \sum_{k=0}^{N-1} \|e_k\|^2_R + \|\epsilon\|^1_q. \tag{4.1}$$

An LTV sequential single shooting scheme is then used to transcribe and linearise the RHP problem. This involves expressing the optimisation variables purely as the control trajectory u, which requires the state trajectory ξ to be approximated as a linear function of u. This results

in the QP

$$\min_{u_\epsilon} \quad \frac{1}{2} u_\epsilon^T H u_\epsilon + f^T u_\epsilon \tag{4.2a}$$

$$\text{subj. to} \quad \xi_0 = \hat{\xi} \tag{4.2b}$$

$$\underline{\gamma} \leq G u_\epsilon \leq \overline{\gamma} \tag{4.2c}$$

$$\epsilon \geq 0, \tag{4.2d}$$

where the control vector and slack variables are combined into a single optimisation variable $u_\epsilon = [\, u \;\; \epsilon \,]^T$.

To achieve the QP formulation (4.2), consider the problem of transforming the objective function into a quadratic function solely in terms of u_ϵ. By introducing a padding $0_\epsilon \in \mathbb{R}^{n_\epsilon}$ as a zero column vector with a length equal to the number of slack variables, and redefining the error vector correspondingly as $e_\epsilon = [\, e \;\; 0_\epsilon \,]^T$, the objective function (4.1) can be rewritten as the following matrix equation

$$J_\epsilon(\xi, u_\epsilon) = (\xi - \xi_r)^T \tilde{Q} (\xi - \xi_r) + e_\epsilon^T \tilde{R} e_\epsilon + \tilde{q}^T u_\epsilon, \tag{4.3}$$

where $\tilde{Q} = \text{blockdiag}(Q, \ldots, Q, Q_f)$, $\tilde{R} = \text{blockdiag}(R, \ldots, R, 0_\epsilon)$ and $\tilde{q} = [\, 0 \;\; \ldots \;\; 0 \;\; q \,]^T$. Now consider the plant model

$$\dot{\xi}_k = f(\xi_k, u_k). \tag{4.4}$$

The model can be linearised by taking the first order Taylor series expansion at some linearisation trajectory $\xi_l = [\, \xi_{l,0} \;\; \ldots \;\; \xi_{l,N-1} \,]^T$, $u_l = [\, u_{l,0} \;\; \ldots \;\; u_{l,N-1} \,]^T$ as

$$\dot{\xi}_k \approx f(\xi_{l,k}, u_{l,k}) + \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi=\xi_{l,k} \\ u=u_{l,k}}} (\xi_k - \xi_{l,k}) + \left.\frac{\partial f}{\partial u}\right|_{\substack{\xi=\xi_{l,k} \\ u=u_{l,k}}} (u_k - u_{l,k}). \tag{4.5}$$

This can be rearranged into the affine expression

$$\dot{\xi}_k = A_k \xi_k + B_k u_k + d_k, \tag{4.6}$$

where

$$A_k = \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi=\xi_{l,k} \\ u=u_{l,k}}}, \quad B_k = \left.\frac{\partial f}{\partial u}\right|_{\substack{\xi=\xi_{l,k} \\ u=u_{l,k}}} \tag{4.7a}$$

$$d_k = f(\xi_{l,k}, u_{l,k}) - A_k \xi_{l,k} - B_k u_{l,k}. \tag{4.7b}$$

Using first order Euler integration, consecutive states can therefore be represented as

$$\xi_{k+1} = \tilde{A}_k \xi_k + \tilde{B}_k u_k + \tilde{d}_k, \tag{4.8}$$

where $\tilde{A}_k = I + A_k\Delta t$, $\tilde{B}_k = B_k\Delta t$ and $\tilde{d}_k = d_k\Delta t$. Alternative integration schemes are discussed in Section 4.4.

The linearisation trajectory that is used is the previously solved optimal trajectory $\xi_l = \xi^*(i-1)$ and $u_l = u^*(i-1)$. As this linearisation trajectory changes at every time step, the system is referred to as an LTV system, and results in an LTV-RHP. Similar to the motivation behind hot-starting discussed in Section 5.2, this approach is expected to work well as the previous optimal trajectory should be a good estimation for what the current optimal trajectory will be. This method also bears resemblance to the RTI approach, and therefore shares similar convergence properties [44].

Once the system has been linearised, the state trajectory ξ needs to be formulated as a function of controls u_ϵ. This can be done by iteratively applying (4.8) from a given initial condition ξ_0, which is a similar method to the batch approach method to solve linear quadratic regulators [45]. This can be expressed as the following matrix equation

$$\begin{bmatrix} \xi_1 \\ \xi_2 \\ \vdots \\ \xi_N \end{bmatrix} = \underbrace{\begin{bmatrix} \tilde{A}_0 \\ \tilde{A}_1\tilde{A}_0 \\ \vdots \\ \prod_{k=0}^{N-1}\tilde{A}_k \end{bmatrix}}_{\tilde{A}} x_0 + \underbrace{\begin{bmatrix} \tilde{B}_0 & 0 & \dots & 0 & 0_\epsilon^T \\ \tilde{A}_0\tilde{B}_0 & \tilde{B}_1 & \dots & 0 & 0_\epsilon^T \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ \prod_{k=1}^{N-1}\tilde{A}_k\tilde{B}_0 & \prod_{k=2}^{N-1}\tilde{A}_k\tilde{B}_1 & \dots & \tilde{B}_{N-1} & 0_\epsilon^T \end{bmatrix}}_{\tilde{B}} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \\ \epsilon \end{bmatrix} + \underbrace{\begin{bmatrix} I & 0 & \dots & 0 \\ \tilde{A}_1 & I & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \prod_{k=1}^{N-1}\tilde{A}_k & \prod_{k=2}^{N-1}\tilde{A}_k & \dots & I \end{bmatrix}}_{\tilde{d}} \begin{bmatrix} \tilde{d}_0 \\ \tilde{d}_1 \\ \vdots \\ \tilde{d}_{N-1} \end{bmatrix}, \tag{4.9}$$

which can be re-expressed as

$$\xi = \tilde{A}\xi_0 + \tilde{B}u_\epsilon + \tilde{d}. \tag{4.10}$$

A complication arises if the actuator transient response from Section 2.4 is used, as the effort term e_ϵ needs to be expressed as a difference between control and state variables, which is not straightforward to do when using a sequential discretisation scheme. This could be done by using a sparse matrix to map $\dot{x}$ and δ from the state trajectory ξ. However, this results in large matrix operations which is undesirable for real time applications. Instead, a simplification is made such that e_ϵ is defined relative to the initial state for the entire horizon.

$$e_\epsilon = [\, u \;\; 0_\epsilon \,]^T - u_0, \qquad u_0 = [\, \dot{x}_0 \;\; \delta_0 \;\; \dots \;\; \dot{x}_0 \;\; \delta_0 \;\; 0_\epsilon \,]^T \tag{4.11}$$

Overall, given (4.10) and (4.11), the original objective function (4.3) can be expressed as the

following multivariable quadratic equation

$$\begin{aligned} J_\epsilon(u_\epsilon) = u_\epsilon^T(\tilde{B}^T\tilde{Q}\tilde{B}+\tilde{R})u_\epsilon + 2(\tilde{A}\xi_0+\tilde{d}-\xi_r)^T\tilde{Q}\tilde{B}u_\epsilon - 2u_0^T\tilde{R}u_\epsilon \\ + (\tilde{A}\xi_0+\tilde{d}-\xi_r)^T\tilde{Q}(\tilde{A}\xi_0+\tilde{d}-\xi_r) + u_0^T\tilde{R}u_0 + \tilde{q}^Tu_\epsilon. \end{aligned} \tag{4.12}$$

Which implies that the QP objective function (4.2a) Hessian H and gradient f are

$$H = 2(\tilde{B}^T\tilde{Q}\tilde{B}+\tilde{R}) \tag{4.13a}$$

$$f = 2(\tilde{B}^T\tilde{Q}(\tilde{A}\xi_0+\tilde{d}-\xi_r) - \tilde{R}u_0) + \tilde{q}. \tag{4.13b}$$

4.1.1 Constraint linearisation

QP solvers are typically limited to linear inequality constraints of the form shown in (4.2c). Therefore, the path constraints $g_\epsilon(\xi, u, \epsilon)$ specified in Chapter 3 need to be discretised and linearised as well. To do this, the constraint function is linearised at each time step about the same linearisation trajectory ξ_l, u_l as the plant model linearisation scheme.

$$g_k \approx g_\epsilon(\xi_{l,k}, u_{l,k}, 0_\epsilon) + \left.\frac{\partial g_\epsilon}{\partial \xi}\right|_{\substack{\xi=\xi_{l,k}\\u=u_{l,k}}} (\xi_k - \xi_{l,k}) + \left.\frac{\partial g_\epsilon}{\partial u}\right|_{\substack{\xi=\xi_{l,k}\\u=u_{l,k}}} (u_k - u_{l,k}) + \frac{\partial g_\epsilon}{\partial \epsilon}\epsilon \tag{4.14}$$

This is rearranged into the following affine expression

$$g_k = C_k\xi_k + D_ku_k + E\epsilon + w_k, \tag{4.15}$$

where

$$C_k = \left.\frac{\partial g_\epsilon}{\partial \xi}\right|_{\substack{\xi=\xi_{l,k}\\u=u_{l,k}}}, \quad D_k = \left.\frac{\partial g_\epsilon}{\partial u}\right|_{\substack{\xi=\xi_{l,k}\\u=u_{l,k}}}, \quad E = \frac{\partial g_\epsilon}{\partial \epsilon} \tag{4.16a}$$

$$w_k = g_\epsilon(\xi_{l,k}, u_{l,k}, 0_\epsilon) - C_kx_{l,k} - D_ku_{l,k}. \tag{4.16b}$$

By stacking the constraints at each time step together, the following matrix constraint equation is obtained

$$\begin{bmatrix} g_0 \\ \vdots \\ g_{N-1} \end{bmatrix} = \underbrace{\begin{bmatrix} C_0 & & \\ & \ddots & \\ & & C_{N-1} \end{bmatrix}}_{\tilde{C}} \begin{bmatrix} \xi_1 \\ \vdots \\ \xi_N \end{bmatrix} + \underbrace{\begin{bmatrix} D_0 & & & E \\ & \ddots & & \vdots \\ & & D_{N-1} & E \end{bmatrix}}_{\tilde{D}} \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \\ \epsilon \end{bmatrix} + \underbrace{\begin{bmatrix} w_0 \\ \vdots \\ w_{N-1} \end{bmatrix}}_{\tilde{w}}, \tag{4.17}$$

which can also be expressed as

$$g = \tilde{C}\xi + \tilde{D}u_\epsilon + \tilde{w}. \tag{4.18}$$

Substituting the linearised state trajectory ξ found in (4.10), the following expression is obtained

$$g = (\tilde{C}\tilde{B} + \tilde{D})u_\epsilon + \tilde{C}(\tilde{A}\xi_0 + \tilde{d}) + \tilde{w}. \tag{4.19}$$

Therefore, the QP constraint matrix G and bounds $\overline{\gamma}$, $\underline{\gamma}$ can be expressed as

$$G = \tilde{C}\tilde{B} + \tilde{D} \tag{4.20a}$$
$$\overline{\gamma} = \overline{g} - \tilde{C}(\tilde{A}\xi_0 + \tilde{d}) - \tilde{w} \tag{4.20b}$$
$$\underline{\gamma} = \underline{g} - \tilde{C}(\tilde{A}\xi_0 + \tilde{d}) - \tilde{w}. \tag{4.20c}$$

4.2 Multiple shooting

Unlike the successive linearisation method, the RHP can alternatively retain its nonlinearities and be formulated as a nonlinear RHP. To do this, simultaneous methods which use both the state trajectory and control trajectory to define the optimisation variables are used to discretise the RHP. As opposed to the sequential method used for successive linearisation, the advantage of simultaneous methods is that derivatives are much simpler to calculate as the chain rule between states and controls is not required. Secondly, this discretisation method allows an initial state trajectory to be used to initialise the optimiser as opposed to just an initial control trajectory. The trade-off is that the number of optimisation variables increases significantly. However the resulting problem also becomes sparse, which modern optimisers such as IPOPT [18] can solve efficiently.

Multiple shooting is a type of simultaneous method which assumes a first-order hold control trajectory, and estimates the state in the next time step using at each node using explicit Runge-Kutta (RK) schemes. The augmented variable $\Xi_k = [\,\xi_{k+1} \;\; u_k\,]^T$ is introduced such that the optimisation variable can be defined as $\Xi = [\,\Xi_0 \;\; \dots \;\; \Xi_{N-1} \;\; \epsilon\,]^T$. Equality constraints (4.21c) are then enforced to ensure continuity of the state trajectory using this shooting prediction.

Overall, the multiple shooting problem using Euler integration is expressed as

$$\min_{\Xi} \quad J_\epsilon(\Xi) = \|\xi_N - \xi_{r,N}\|^2_{Q_f} + \sum_{k=1}^{N-1} \|\xi_k - \xi_{r,k}\|^2_Q + \sum_{k=0}^{N-1} \|e_k\|^2_R + \|\epsilon\|^1_q \tag{4.21a}$$
$$\text{subj. to} \quad \xi_0 = \hat{\xi} \tag{4.21b}$$
$$\xi_{k+1} = \xi_k + f(\xi_k, u_k)\Delta t, \qquad k = 0, \dots, N-1 \tag{4.21c}$$
$$\underline{g} \le g_\epsilon(\xi_k, u_k, \epsilon) \le \overline{g}, \qquad k = 0, \dots, N-1 \tag{4.21d}$$
$$\epsilon \ge 0. \tag{4.21e}$$

Alternative integration schemes are discussed in Section 4.4.

To define the optimisation problem for the NLP, in addition to the objective function $J_\epsilon(\Xi)$, the objective function gradient $\nabla J_\epsilon(\Xi)$, constraint function $c(\Xi)$, constraint function Jacobian

$\nabla c(\Xi)$, and Hessian of the Lagrange equation must also be specified. These matrices are discussed in more detail in Appendix A.1. For the Hessian of the Lagrange equation, the Broyden–Fletcher–Goldfarb–Shanno (BFGS) Quasi-Newton method is employed [18].

4.3 Trapezoidal collocation

The collocation method is another simultaneous discretisation method which uses piecewise polynomial splines to approximate the control trajectories between collocation points, and implicit RK methods to integrate the plant model.

The trapezoidal collocation method assumes a piecewise linear control scheme, which can be integrated using the trapezoidal method (4.22c), resulting in a piecewise quadratic state trajectory [46]. The optimisation variable is modified to include discretised state and control points at every collocation point including at the initial time step, such that $\Xi_k = [\, \xi_k \;\; u_k \,]^T$ and $\Xi = [\, \Xi_0 \;\; \dots \;\; \Xi_N \;\; \epsilon \,]^T$.

Overall, the trapezoidal collocation problem is expressed as

$$\begin{aligned}
\min_{\Xi} \quad & J_\epsilon(\Xi) = \frac{1}{2}\left(\|\xi_{N-1} - \xi_{r,N-1}\|^2_{Q_f} + \|\xi_N - \xi_{r,N}\|^2_{Q_f}\right) \\
& \quad + \sum_{k=0}^{N-2} \frac{1}{2}\left(\|\xi_k - \xi_{r,k}\|^2_{Q} + \|\xi_{k+1} - \xi_{r,k+1}\|^2_{Q}\right) && \text{(4.22a)} \\
& \quad + \sum_{k=0}^{N-1} \frac{1}{2}\left(\|e_k\|^2_{R} + \|e_{k+1}\|^2_{R}\right) + \|\epsilon\|^1_q \\
\text{subj. to} \quad & \xi_0 = \hat{\xi} && \text{(4.22b)} \\
& \xi_{k+1} = \xi_k + \frac{1}{2}(f(\xi_k, u_k) + f(\xi_{k+1}, u_{k+1}))\Delta t, \qquad k = 0, \dots, N-1 && \text{(4.22c)} \\
& \underline{g} \le g_\epsilon(\xi_k, u_k, \epsilon) \le \overline{g}, \qquad k = 0, \dots, N && \text{(4.22d)} \\
& \epsilon \ge 0. && \text{(4.22e)}
\end{aligned}$$

A slight modification is required for the soft constraints when using a collocation method, where lateral deviation constraints are removed for the initial step $k = 0$. Constraints purely on the initial state ξ_0 are meaningless as it is a fixed value, and using an ∞-norm penalty formulation on slack variables means that future states are not encouraged to move back into the feasible region if the initial state is already violated.

More details about the objective function gradient, constraints and constraint Jacobian are in Appendix A.2. Similar to the multiple shooting scheme, the BFGS method is used to approximate the Hessian of the Lagrange equation.

4.4 Integration schemes

In the successive linearisation and multiple shooting schemes, Euler integration was initially used to propagate the state forwards. To increase the accuracy of this integration, a finer discretisation grid can be used by decreasing the time step Δt. However, this either leads to a shorter time horizon if the NLP size is to be kept constant, or a larger optimisation problem if the time horizon is to be kept constant. Each of these will either lead to either less optimal solutions or slower computation times respectively, neither of which are desirable. Instead, a higher order integration scheme can be used instead to increase the accuracy while retaining the same amount of discretisation points and time horizon. The RK family was used to achieve this.

To introduce how these integration schemes are used, a generalised form of the integration of a state space equation is introduced

$$\xi_{k+1} = \xi_k + f_{(\cdot)}(\xi_k, u_k), \tag{4.23}$$

where $f_{(\cdot)}\colon \mathbb{R}^{n_\xi} \times \mathbb{R}^{n_u} \to \mathbb{R}^{n_\xi}$ is the integration scheme being considered. For example, the Euler integration scheme would be defined as

$$f_{\text{Euler}}(\xi_k, u_k) = f(\xi_k, u_k)\Delta t. \tag{4.24}$$

Now consider the RK4 integration scheme

$$f_{\text{RK4}}(\xi_k, u_k) = \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4), \tag{4.25}$$

where

$$k_1 = f(\xi_k, u_k)\,\Delta t \tag{4.26a}$$

$$k_2 = f\left(\xi_k + \frac{1}{2}k_1, u_k\right)\Delta t \tag{4.26b}$$

$$k_3 = f\left(\xi_k + \frac{1}{2}k_2, u_k\right)\Delta t \tag{4.26c}$$

$$k_4 = f(\xi_k + k_3, u_k)\,\Delta t. \tag{4.26d}$$

To use this for both the successive linearisation and multiple shooting schemes, the Jacobians

of this expression need to be derived. This is found as

$$\tilde{A}_k = I + \left.\frac{\partial f_{\text{RK4}}}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} = I + \frac{1}{6}\left(\frac{\partial k_1}{\partial \xi} + 2\frac{\partial k_2}{\partial \xi} + 2\frac{\partial k_3}{\partial \xi} + \frac{\partial k_4}{\partial \xi}\right) \tag{4.27a}$$

$$\tilde{B}_k = \left.\frac{\partial f_{\text{RK4}}}{\partial u}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} = \frac{1}{6}\left(\frac{\partial k_1}{\partial u} + 2\frac{\partial k_2}{\partial u} + 2\frac{\partial k_3}{\partial u} + \frac{\partial k_4}{\partial u}\right) \tag{4.27b}$$

$$\tilde{d}_k = f_{\text{RK4}}(\xi_{l,k}, u_{l,k}) - \left.\frac{\partial f_{\text{RK4}}}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} \xi_{l,k} - \left.\frac{\partial f_{\text{RK4}}}{\partial u}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} u_{l,k}, \tag{4.27c}$$

where the partial derivatives of each component k_i is found as [47]

$$\frac{\partial k_1}{\partial \xi} = \Delta t \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} \tag{4.28a}$$

$$\frac{\partial k_2}{\partial \xi} = \Delta t \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_1 \\ u = u_{l,k}}} \left(I + \frac{1}{2}\frac{\partial k_1}{\partial \xi}\right) \tag{4.28b}$$

$$\frac{\partial k_3}{\partial \xi} = \Delta t \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_2 \\ u = u_{l,k}}} \left(I + \frac{1}{2}\frac{\partial k_2}{\partial \xi}\right) \tag{4.28c}$$

$$\frac{\partial k_4}{\partial \xi} = \Delta t \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + k_3 \\ u = u_{l,k}}} \left(I + \frac{\partial k_3}{\partial \xi}\right), \tag{4.28d}$$

and

$$\frac{\partial k_1}{\partial u} = \Delta t \left.\frac{\partial f}{\partial u}\right|_{\substack{\xi = \xi_{l,k} \\ u = u_{l,k}}} \tag{4.29a}$$

$$\frac{\partial k_2}{\partial u} = \Delta t \left(\left.\frac{\partial f}{\partial u}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_1 \\ u = u_{l,k}}} + \frac{1}{2} \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_1 \\ u = u_{l,k}}} \frac{\partial k_1}{\partial u} \right) \tag{4.29b}$$

$$\frac{\partial k_3}{\partial u} = \Delta t \left(\left.\frac{\partial f}{\partial u}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_2 \\ u = u_{l,k}}} + \frac{1}{2} \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + \frac{1}{2}k_2 \\ u = u_{l,k}}} \frac{\partial k_2}{\partial u} \right) \tag{4.29c}$$

$$\frac{\partial k_4}{\partial u} = \Delta t \left(\left.\frac{\partial f}{\partial u}\right|_{\substack{\xi = \xi_{l,k} + k_3 \\ u = u_{l,k}}} + \frac{1}{2} \left.\frac{\partial f}{\partial \xi}\right|_{\substack{\xi = \xi_{l,k} + k_3 \\ u = u_{l,k}}} \frac{\partial k_3}{\partial u} \right). \tag{4.29d}$$

Note that $\tilde{A}_k$, $\tilde{B}_k$ and $\tilde{d}_k$ are the same expressions that are used in (4.8) and (A.4a).

Similarly, for the RK2 integration scheme

$$f_{\text{RK2}}(\xi_k, u_k) = k_2, \tag{4.30}$$

a similar derivation can be made to show that the Jacobians required for discretisation are given by (4.28b) and (4.29b).

Chapter 5

Implementation

5.1 Time delay compensation

Time delay in a system poses a problem to control systems, and even small amounts of delay can result in instability and poor performance [40]. As seen in Figure 5.1, even a small amount of uncompensated time delay can lead to extremely poor control performance. In the application of autonomous racing, time delay is expected in multiple areas, including algorithm processing times, communication delays between computing units, and actuator transient responses.

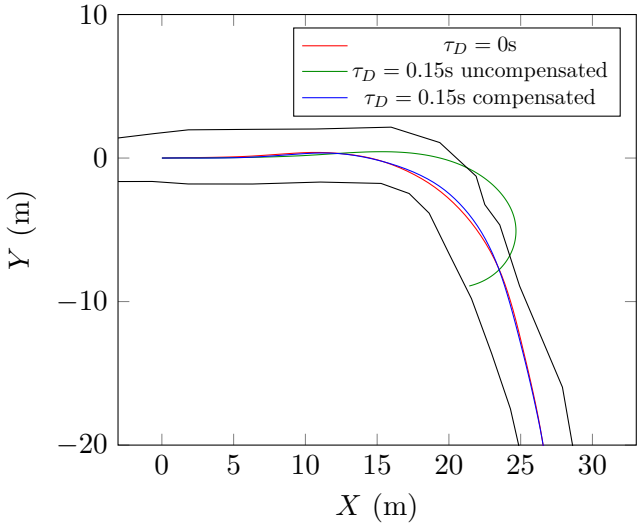

Figure 5.1: Simulated trajectories with and without time delay around the first corner of FSS2019. If time delay is uncompensated for, the RHP performs poorly as it reacts too slowly to perform required maneuvers. The proposed delay-compensation algorithm is able to achieve similar results to the trajectory without time delay.

Here, an important distinction is made between transient response delay as described in Section 2.4, and dead time which represents the time between when a command is sent and when the vehicle starts responding. Unlike transient response, dead time is difficult to model

in a way that can be used for control. For RHP problems, a common way to compensate for a period of dead time τ_D is to make a prediction of the state after an expected delay [48, 49]. Past controls are stored in a list $\{u_0^*(i-1), u_0^*(i-2), \ldots, u_0^*(i-M)\}$, which are then used to predict the delay-compensated state from the measured state $\hat{\xi}$. The delay-compensated state is then used as the initial condition for the RHP. The same model used for the RHP plant model can be used for the prediction estimations. A description of the algorithm used to implement this is detailed in Algorithm 1.

Algorithm 1 Time delay compensation

```
i ← 0
M ← ⌈τ_D/dt⌉                          ▷ Total length of past controls needed
for j in 1 to M do
    u_0^*(i − j) ← 0                  ▷ No control inputs before initialisation
end for

while carIsRunning do
    ξ̂ ← SLAM()                        ▷ Receive state estimation from SLAM algorithm
    for j in M downto 2 do
        ξ̂ ← ξ̂ + f(ξ̂, u_0^*(i − j))Δt   ▷ Forward prediction step
    end for
    ξ̂ ← ξ̂ + f(ξ̂, u_0^*(i − 1))(τ_D − (M − 1)Δt)   ▷ Final forward prediction step
    u_0^*(i) ← RHP(ξ̂)                 ▷ Store controls to use in future delay compensation
    i = i + 1
end while
```

It is important to note however that this approach does not perfectly compensate for time delay. The model used to predict the future state is imperfect, such that the longer the time delay that needs to be compensated, the more significant modelling error becomes, and the less accurately the delay is compensated for. Therefore, the root cause of dead time should always be minimised where possible.

5.2 Hot-starting

Choosing a good initial guess is important in optimisation problems to minimise the number of iterations required to reach an optimal solution, and also to avoid falling into bad local minima. Hot-starting or warm-starting is a technique where a solution from a similar optimisation problem is used as the initial guess, based on the assumption that similar optimisation problems should have similar optimal solutions.

For RHP problems, the ability to hot-start a solution arises naturally from the receding horizon nature of the algorithm, by using the previous optimal solution to hot-start the current optimal control problem. Notably, outside of model errors, disturbances, and sensor measurement noise, and assuming the terminal penalty perfectly predicts the infinite horizon, the last $N-1$ elements of the previous optimal solution should be exactly equal the first $N-1$ elements

of the following optimal solution, as the two problems are identical apart from being a time step apart from each other. This allows hot-starting to be further improved for RHP problems by shifting the previous solution forwards by a time step [50], in which case new terminal variables $u_N = u_{N-1}$ and $x_{N+1} = x_N + f(x_N, u_N)\Delta t$ are typically chosen to be appended to the initial guess.

Hot-starting is known to work well for active-set methods, which is the type of optimisation algorithm qpOASES [51] uses. However, IP methods such as that used by IPOPT [18] are less effective at leveraging information obtained from hot-starting [17]. These differences are observed in the results shown in Figure 5.2, where the impact of hot-starting is most prominent when using the active-set optimiser. Overall, hot-starting with the active-set optimiser resulted in a computation time improvement by a factor of 7.44, compared to the factor of 1.25 achieved by the IP method.

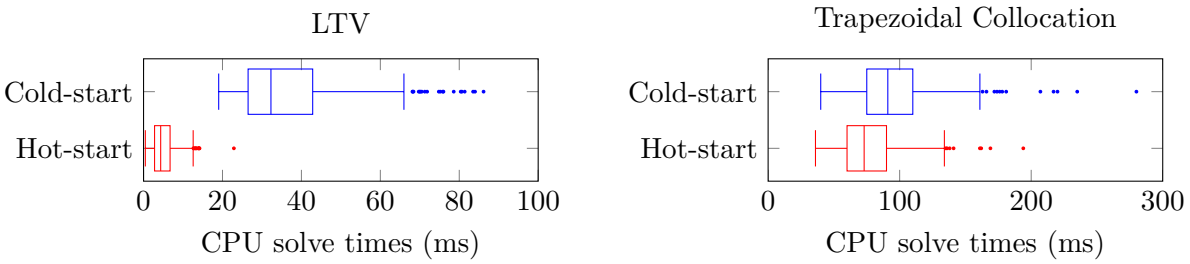

Figure 5.2: Comparisons between computation times of solving the optimisation problem when cold-starting and hot-starting. The LTV-RHP (left) is solved using qpOASES, while the trapezoidal collocation scheme (right) is solved using IPOPT. The most extreme outlier for hot-started LTV-RHP CPU times corresponds to the first iteration when there is no solution to hot-start from.

5.3 Block matrix multiplication

Block matrix multiplication is a method of improving the efficiency of matrix multiplications through cache-friendly operations. To optimise matrix operations, matrices are ideally stored in the L1 cache, which second only to registers are the fastest form of memory in a computer. However, due to their small storage size, large matrices cannot be stored on the L1 cache, and therefore must be stored on the slower L2 cache, L3 cache or main memory.

Blocking is a technique in matrix multiplication where the full matrices are subdivided into smaller blocks, which are then treated as elements to be used in a way similar to the standard matrix multiplication algorithm. By doing this, these submatrices can be fully stored in the L1 cache, which can significantly improve the matrix multiplication performance [52]. This algorithm is visualised in Figure 5.3.

For the LTV-RHP described in Section 4.1, matrix multiplication between large, dense matrices need to be performed to generate the QP problem. In C++, the Eigen 3 [53] library

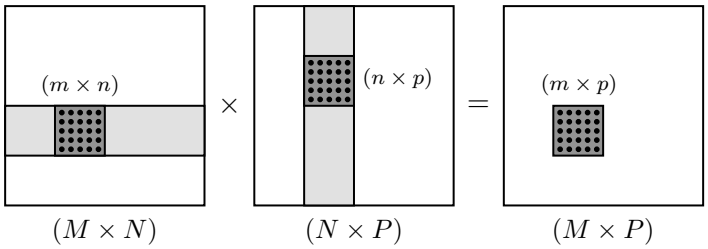

Figure 5.3: Representation of block matrix multiplication.

is used to optimise the performance of these operations. However, when initially testing with unblocked operations, the total computation times to perform these operations was relatively slow and erratic as seen in Figure 5.4. This effect may have been exacerbated by other simultaneously running algorithms such as the SLAM algorithm similarly needing to perform large matrix multiplications, resulting in cache thrashing.

Blocking can be implemented fairly naturally for the QP generation problem, as matrices are already constructed as block matrices. Another advantage of block operations is that the structure of the matrices can be taken advantage of. For example, many of the matrices that are used in generating the QP problem are block-triangular, and by using this known structure, the number of operations that need to be performed is halved.

Overall, blocking operations were able to significantly reduce the average computation times by about a factor of 10, and the worst-case computation time by a factor of 35.

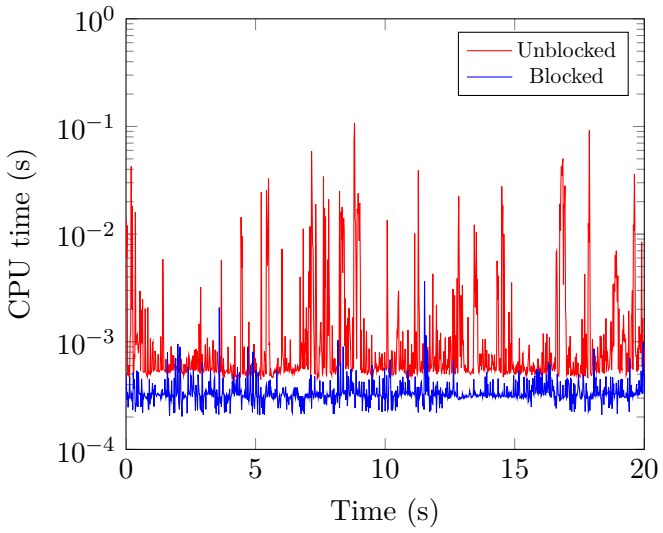

Figure 5.4: CPU times required to calculate the Hessian H, gradient f, constraint matrix G and bounds $\underline{\gamma}, \overline{\gamma}$ of the LTV-RHP QP problem with 20 time steps in C++ using the Eigen library over a lap of FSG2019 . The standard matrix multiplication had an average CPU time of 3.05 ms and maximum CPU time of 125.5 ms, compared to the average CPU time of 0.34 ms and maximum CPU time of 3.6 ms when using block matrix multiplication.

Chapter 6

Experimental Results

To determine the best performing formulation of the RHP problem, different vehicle models and discretisation schemes discussed in Chapters 2 and 4 were first prototyped in MATLAB, and their performance were evaluated against each other to determine which formulation performed the best for the FSD competition environment. Once an optimal formulation was determined, it was integrated with the rest of the autonomous systems pipeline in C++. A full hardware-in-the-loop simulation experiment was performed to validate the performance of the proposed RHP design, and its performance was benchmarked against MMS's existing motion planning implementations. Three tracks used in previous official FSD competitions including Formula Student Germany (FSG2019), Formula Student Spain (FSS2019) and Formula Student Online (FSO2020) are used to evaluate the planner performances, and are visualised in Figure 6.10.

6.1 RHP Formulation Analysis

Intitial experiments were conducted in MATLAB in a simplified simulation environment[1] to compare different RHP formulations. The simulation environment utilised the dynamic bicycle model in Cartesian coordinates described in (2.3), which the RHP interfaced with using velocity and steering wheel angle PID controllers. Sensor noise, dead time and actuator transient response were omitted from the simulation. RHP plant models in this section did not incorporate actuator transient response described in Section 2.4. The reference trajectory used was a naive 20 m/s constant velocity trajectory with no lateral or angular deviation from the centreline. Experiments were performed on an AMD Ryzen 5 3600 CPU running at 3.6 GHz with 16 GB of RAM on Windows 10. Unless otherwise specified, all RHP's used a time step of $\Delta t = 50\,\text{ms}$ and number of time steps $N = 40$.

[1] https://github.com/kerry-he/fsae-mpc

6.1.1 Vehicle model

The kinematic bicycle model and the dynamic bicycle model were evaluated against each other as the plant model used in the RHP. Lap times are compared in Table 6.1, and computation times are compared in Figure 6.2.

The dynamic bicycle model saw significant lap time advantages over the kinematic bicycle model. The main advantage of the dynamic bicycle model is that it is able to model tyre forces experienced by the car, and utilise tyre friction ellipse constraints to ensure the car safely remains within the limits of operation. In comparison, the lateral acceleration constraints that are enforced on the kinematic bicycle model have to be more restrictive to ensure the car remains within the same limits, as the lateral tyre force can only be approximated through the lateral acceleration of the car. The differences in these constraints are shown in Figure 6.1. As maximisation of tyre forces is often one of the most important factors in vehicle racing [24], the improved cornering ability is reflected by significantly faster velocities across the track, ultimately resulting in better lap times.

Table 6.1: Comparison of lap times achieved by RHP formulations using the kinematic and dynamic bicycle models.

	Lap Times (s)		
	FSG2019	FSS2019	FSO2020
Kinematic	27.27	28.13	35.27
Dynamic	22.43	22.17	27.70

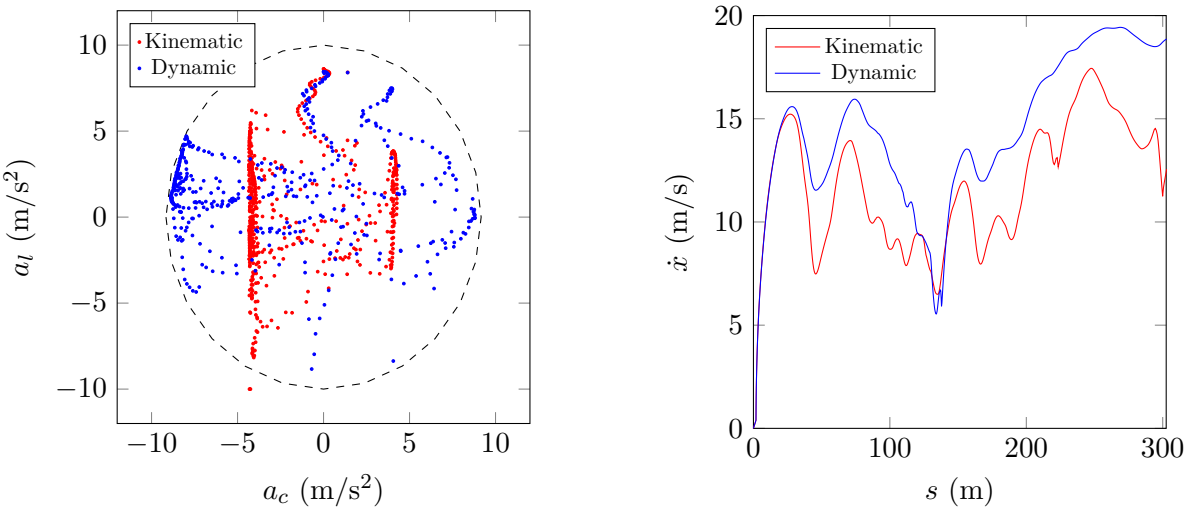

Figure 6.1: Comparison of simulated longitudinal and latitudinal accelerations experienced by the rear tyres (left) and the longitudinal velocity profile (right) between the kinematic and dynamic bicycle models over a lap of FSG2019. Enforced friction ellipse boundaries are shown as the dashed ellipse.

However, the trade off with using the dynamic bicycle model is the added computational complexity arising from an increase in state variables, constraints, and more complex gradient structures. Overall, it was found that the dynamic bicycle model solved approximately twice as slowly compared to the kinematic bicycle model. However, it was decided that the improved lap times was a more important factor than the slower CPU times. The key requirement for real-time feasibility of the RHP is for the solve times to be less than the discretised time step. Multiple methods to improve CPU times were not implemented yet, including using optimised C++ code, using hot-starting and MPC-specific optimisations offered by qpOASES [51], and using a precomputed optimal racing line as the reference trajectory as opposed to a naive constant velocity reference. These changes were expected to reduce the computation times by approximately a factor of 10, and bring the maximum computation time below the 50 ms time step used in the experiments.

Given these factors, the dynamic bicycle model was chosen as the plant model for the final RHP implementation.

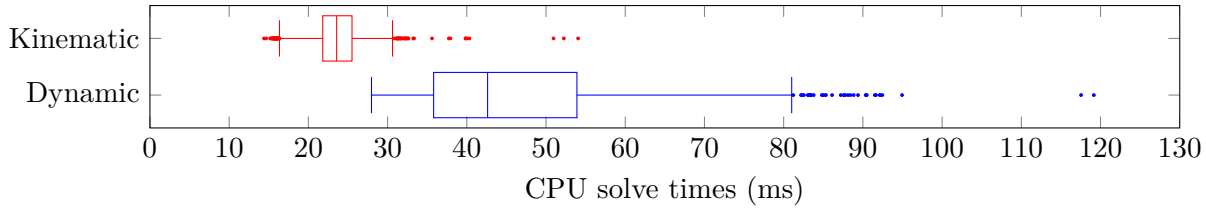

Figure 6.2: Distribution of CPU solve times across a lap of each of the FSG2019, FSS2019 and FSO2020 tracks.

6.1.2 Discretisation

The successive linearisation, multiple shooting, and trapezoidal collocation methods were explored to transcribe the RHP to an optimisation problem. All linear schemes were solved using qpOASES [51], and all nonlinear schemes were solved using IPOPT [18]. Lap times are compared in Table 6.2, and computation times are compared in Figure 6.3.

The successive linearisation schemes were observed to solve much faster than all other nonlinear schemes, and also exhibited a much smaller range of outliers. In particular, there were situations with the RK2 and RK4 multiple shooting schemes where the solution did not converge after the specified 5000 maximum iterations. As the worst-case computation times are important for being able to guarantee a feasible solution within a specified time limit, the large spread of outliers for the nonlinear schemes is highly undesirable. Another interesting observation is that while the integration scheme had a minimal impact on the computation speed of the successive linearisation schemes, it has a noticeable impact on the multiple shooting schemes. This may be attributed to how for the successive linearisation scheme, the integration scheme is only called a single time at each time step to construct the QP. However for the multiple shooting formula-

tions, the integration scheme needs to be called multiple times at each time step to calculate the constraints and constraint Jacobian at each optimisation iteration. A second major difference between the linear and nonlinear discretisation schemes with regards to computational speed is their sensitivity to the RHP problem size. As seen in Figure 6.4, the linear scheme is much more sensitive to the number of time steps N used in the RHP. This could be advantageous for the linear scheme as it provides an additional degree of freedom to tune the RHP to reduce the computation times if needed.

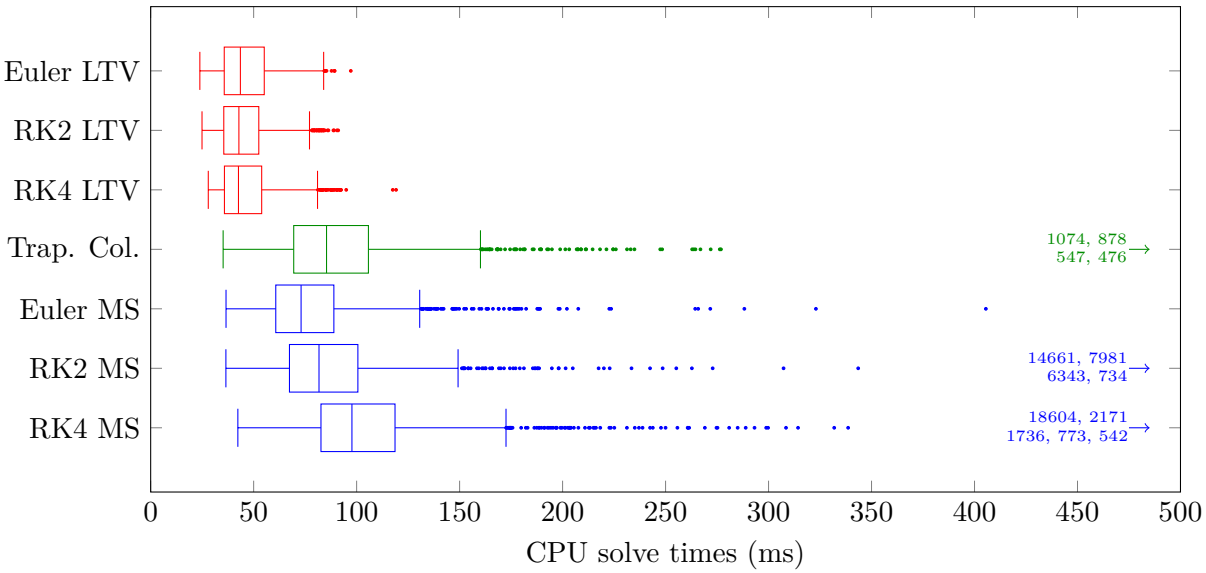

Figure 6.3: Distribution of CPU solve times across a lap of each of the FSG2019, FSS2019 and FSO2020 tracks.

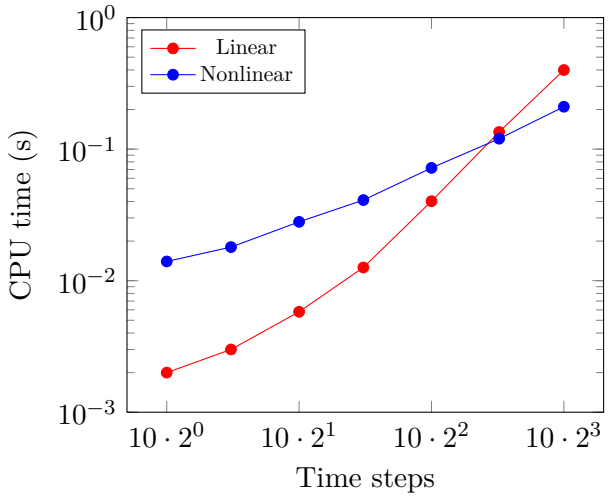

Figure 6.4: Comparison of CPU times between varying sizes of linear (LTV-RHP) and nonlinear (trapezoidal collocation) RHP discretisation schemes. The linear scheme is much more sensitive to the problem size.

Although the nonlinear schemes consistently achieved faster lap times over successive linearisation, the margin between the best and worst lap times was much smaller than the lap time differences between different vehicle models. As seen in the G-G plot comparison in Figure 6.5, the nonlinear schemes are only observed to have a marginal improvement in friction ellipse utilisaiton. However, the nonlinear schemes were observed to be able to better navigate through more complex parts of the track, such as the slalom in FSG2019 as seen in the velocity plot in Figure 6.5. Nevertheless, once implemented in the presence of noise, time delay and other imperfections, in addition to utilising a precomputed racing line reference, these improvements are not expected to be particularly significant. Another observation is that the RK2 and RK4 integration schemes generally outperformed the Euler integration scheme, which was expected due to their higher integration accuracy.

Table 6.2: Simulated lap time comparison between different discretisation methods. Shortest lap times for each track are bolded.

Discretisation Method		Lap Time (s)		
		FSG2019	FSS2019	FSO2020
LTV	Euler	22.96	22.33	27.98
	RK2	22.10	22.37	28.21
	RK4	22.43	22.17	27.70
Collocation	Trapezoidal	21.90	22.05	27.55
Multiple Shooting	Euler	22.36	21.69	27.86
	RK2	**21.85**	**21.42**	27.35
	RK4	21.89	21.43	**27.34**

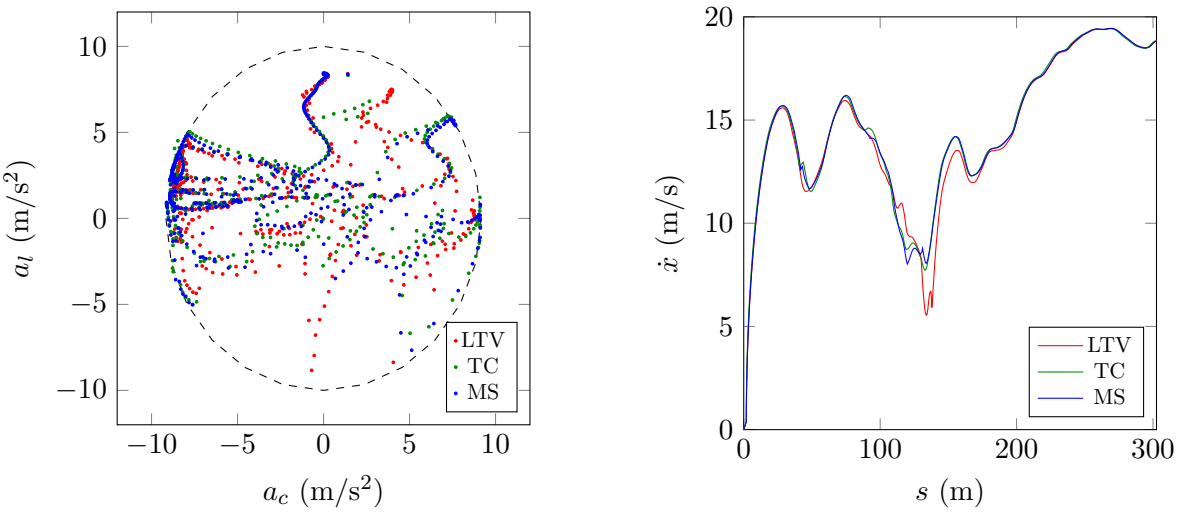

Figure 6.5: Comparison of simulated longitudinal and latitudinal accelerations experienced by the rear tyres between linear time-varying (LTV), trapezoidal collocaiton (TC) and multiple shooting (MS) discretisation schemes over a lap of FSG2019.

Table 6.3: Simulated track and friction ellipse soft constraint violations. The smallest amount of violations for each track are bolded.

Discretisation Method		Maximum Track Constraint Violation (m)			Total Friction Ellipse Constraint Violation (s)		
		FSG2019	FSS2019	FSO2020	FSG2019	FSS2019	FSO2020
LTV	Euler	**0.00**	0.03	**0.00**	**0.00**	**0.00**	0.06
	RK2	**0.00**	0.03	**0.00**	**0.00**	**0.00**	**0.00**
	RK4	0.01	**0.01**	**0.00**	**0.00**	**0.00**	0.02
Collocation	Trapezoidal	0.02	**0.01**	0.06	0.01	0.02	0.06
Multiple Shooting	Euler	**0.00**	0.03	**0.00**	0.27	0.75	0.29
	RK2	**0.00**	0.02	**0.00**	0.02	0.18	0.25
	RK4	0.01	0.02	0.02	0.01	0.04	0.61

In addition to lap times, soft constraint violations as summarised in Table 6.3 were also compared. Interestingly, the successive linearisation scheme was able to satisfy the constraints the most consistently, perhaps due to the problem being simpler and easier to solve. In comparison, the multiple shooting methods are much less consistent in satisfying the soft constraints, particularly the friction ellipse constraint.

The main drawback of the successive linearisation scheme is its instability and inability to find feasible solutions in difficult scenarios, which is a problem that is exacerbated in the presence of noise and dead time. As the linearisation trajectory used is the optimal solution of the previous solution, a bad solution leads to a bad linearisation trajectory, which can cause a cascading effect which the LTV-RHP may struggle to recover from. However, precomputing an optimal racing line and extracting reference waypoints from this path helped to guide the LTV-RHP towards a feasible, optimal solution which mitigated these issues, and the objective function weights were carefully chosen to punish deviations from this precomputed trajectory.

Overall, it was decided that for the RHP, the major computational speed benefits of the successive linearisation scheme outweighed the minor performance benefits brought on by the more complex nonlinear discretisation methods. Moreover, an RK4 integration scheme was chosen for its improved lap time performance, while avoiding any additional computational burden. However, for the offline optimal racing line problem, a nonlinear discretisation method was used for its higher accuracy, robustness, and ability to search through a wider state space without a good initial guess. Specifically, the trapezoidal collocation method was chosen over the multiple shooting methods for its ability to better satisfy constraints.

6.2 Full simulation

Following the previous experiments, the proposed RHP was designed using a successive linearisation discretisation scheme with an RK4 integrator, utilising a dynamic bicycle model described

in (2.18). The final LTV-RHP[2] was implemented in C++ to optimise its performance. A time step of $\Delta t = 20\,\text{ms}$ with $N = 20$ was used, and a dead time of 150 ms was compensated using the forward prediction method presented in Section 5.1. The LTV-RHP was solved using qpOASES [51], with hot-starting and MPC-specific settings enabled. The reference trajectory used is a precomputed optimal racing line computed using the method described in Section 3.2, solved using a trapezoidal collocation discretisation scheme.

A hardware-in-the-loop simulation environment implemented in ROS [54] shown in Figure 6.6 was used to evaluate the performance of the RHP when fully integrated into MMS's autonomous systems pipeline. From the simulated vehicle's current state, IMU, GPS, and landmark measurements were simulated with noise. The SLAM algorithm used these measurements to estimate the vehicle state, which the RHP used to calculate an optimal trajectory. The commercial vehicle modelling software IPG CarMaker[3] was used to simulate the vehicle dynamics, which considers vehicle suspension, powertrain, aerodynamic, and tyre force effects. IPG CarMaker was run as a separate process and communicates with ROS through a UDP protocol, which allowed the online capabilities of the RHP to be tested.

The proposed LTV-RHP was compared against two other planners which were previously used by MMS. The first was a baseline (BL) RHP formulation which utilised a kinematic bicycle model in Cartesian coordinates, had a time step of $\Delta t = 600\,\text{ms}$ with $N = 5$, and was transcribed and solved using ACADO Toolkit [1]. The second planner was a pure pursuit algorithm (PP). Both of these alternative planners used a precomputed optimal racing trajectory based off an algorithm previously developed by MMS which uses a simple point mass model [55].

All simulations were run on an Intel i7-6500U CPU running at 2.5GHz with 8GB of RAM on Linux Ubuntu 18.04.

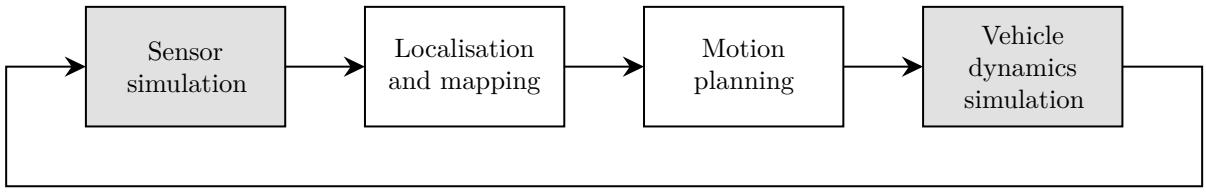

Figure 6.6: Hardware-in-the-loop simulation environment pipeline used for final evaluation of the RHP. Grey blocks represent simulated components of the real world equivalents, and white blocks function identically between simulated and real world missions.

6.2.1 Tuning

All planners were tuned to achieve the fastest lap times without hitting any cones, as hitting cones incurs a points penalty in the competition.

In tuning the LTV-RHP, the most important consideration was ensuring that it was robust

[2] https://bitbucket.org/monashmotorsport/mms_motion_control/src/master/
[3] https://ipg-automotive.com/products-services/simulation-software/carmaker/

and stable in the presence of noise and time delay. Firstly, the control weight matrix R was weighted relatively heavily to dampen unnecessary movements which may destabilise the vehicle. The lateral deviation n weight was similarly weighted to ensure the vehicle did not collide into the track boundaries. The velocity $\dot{x}$ and steering angle δ weights were also weighted more heavily as they govern the movement of the vehicle, and tracking the reference values should navigate it along an optimal trajectory in theory.

A full summary of the LTV-RHP parameters used for each track is shown in Table 6.4.

Table 6.4: Parameters for LTV-MPC used for each track in the full simulation experiments.

	Q	Q_f	R	q	$\overline{n}$
FSG2019	diag(1, 1.25e4, 1, 200, 1, 1, 100)	10Q	diag(50, 1e5)	$[2.5e7\ \ 1e5\ \ 2e5\ \ 2e5]^T$	1.0
FSS2019	diag(1, 2.5e4, 1, 200, 1, 1, 100)	10Q	diag(50, 1e5)	$[1e9\ \ 1e5\ \ 2e5\ \ 2e5]^T$	0.875
FSO2020	diag(1, 5e4, 1, 200, 1, 1, 100)	10Q	diag(50, 1e5)	$[1e9\ \ 1e5\ \ 2e5\ \ 2e5]^T$	0.75

6.2.2 Racing performance

The simulation results are summarised in Table 6.5, and the racing trajectories of each motion planner in each track are presented in Figure 6.10. For comparison, the winning team in FSG2019 obtained a total time of 244.90 s. Overall, the LTV-RHP consistently and significantly outperformed all other planners in all tracks.

Table 6.5: Simulated lap times in three different past competition tracks using the proposed LTV-RHP (LTV), previous baseline RHP (BL) and pure pursuit (PP).

Track		Lap Times (s)										
		1	2	3	4	5	6	7	8	9	10	**Total**
FSG2019	LTV	22.89	21.27	21.39	21.41	21.42	21.40	21.41	21.46	21.33	21.40	**215.4**
	BL	31.22	30.72	30.71	30.73	30.70	30.73	30.72	30.74	30.72	30.74	**307.7**
	PP	29.68	28.81	28.81	28.81	28.80	28.82	28.82	28.83	28.79	28.78	**288.9**
FSS2019	LTV	24.64	23.05	22.93	23.08	22.94	23.00	22.90	22.95	23.00	22.95	**231.5**
	BL	28.84	28.23	28.24	28.26	28.23	28.23	28.23	28.24	28.22	28.22	**282.9**
	PP	36.00	35.42	35.41	35.41	35.41	35.41	35.39	35.40	35.41	35.43	**354.7**
FSO2020	LTV	35.02	33.74	33.84	33.86	33.84	33.70	33.91	33.83	33.78	33.86	**339.4**
	BL	48.04	47.67	47.64	47.64	47.67	47.65	47.65	47.66	47.64	47.66	**476.9**
	PP	40.00	39.47	39.52	39.49	39.48	39.51	39.50	39.48	39.50	39.50	**395.4**

The improved performance of the LTV-RHP can be attributed to a few key factors. Firstly, by using the more complex dynamic bicycle model, thte LTV-RHP was able to exploit the vehicle dynamics much more effectively, as discussed in Section 6.1.1. As shown in the G-G plot in Figure 6.7, the LTV-RHP can be seen to be driving at the edges of the friction ellipse for the majority of the track as opposed to the other two planners. The overall effect is seen as a higher average acceleration; the LTV-RHP maintained an average acceleration of 9.44 $\mathrm{m/s^2}$ over

FSG2019, compared to the 4.86 m/s^2 and 6.12 m/s^2 average accelerations maintained by the BL and PP planners.

Secondly, the LTV-RHP had improved path tracking capabilities due to the use of curvilinear coordinates and enforcement of track boundary constraints. As FSD tracks tend to be relatively narrow, it can be quite difficult to race through the track at a high velocity without knowledge of the track boundaries. BL and PP were both significantly hindered by this; the reference trajectory was already fixed to the track centreline, and the only other tunable parameter for these planners to prevent track boundary violations was to reduce the maximum reference velocities and accelerations, as seen in Figure 6.8. On the other hand, by implementing track boundary constraints on the LTV-RHP, the car could drive along a much more efficient racing line while still ensuring that it remained within the tracks. This can be seen in Figure 6.9 and 6.10, where the LTV-RHP skirted about the boundaries and corner-cut in a controlled manner much more frequently, but never passed the specified boundaries. The other two planners had to remain closer to the centreline for the majority of the track, yet still exhibited large uncontrolled deviations which ultimately bottlenecked the performance of the planners. PP was also observed to exhibit undesirable fluctuations in the lateral deviation arising from oscillatory steering as opposed to efficient corner cutting.

Finally, new methods of compensating for time delay were also significant contributors to the improved performance. Transient delay represents a significant portion of the lag in the system, but was not considered in BL. Instead, BL employed large time steps of $\Delta t = 600$ms to create a dampening effect on the controls. However, this led to shallow controls which had trouble performing complex manoeuvres, which was a second reason why the velocity profile of BL had to be relatively constant. By modelling the transient response in the LTV-RHP, small time steps could be used while remaining robust to time delay, allowing the planner to perform much more aggressive manoeuvres to achieve faster lap times.

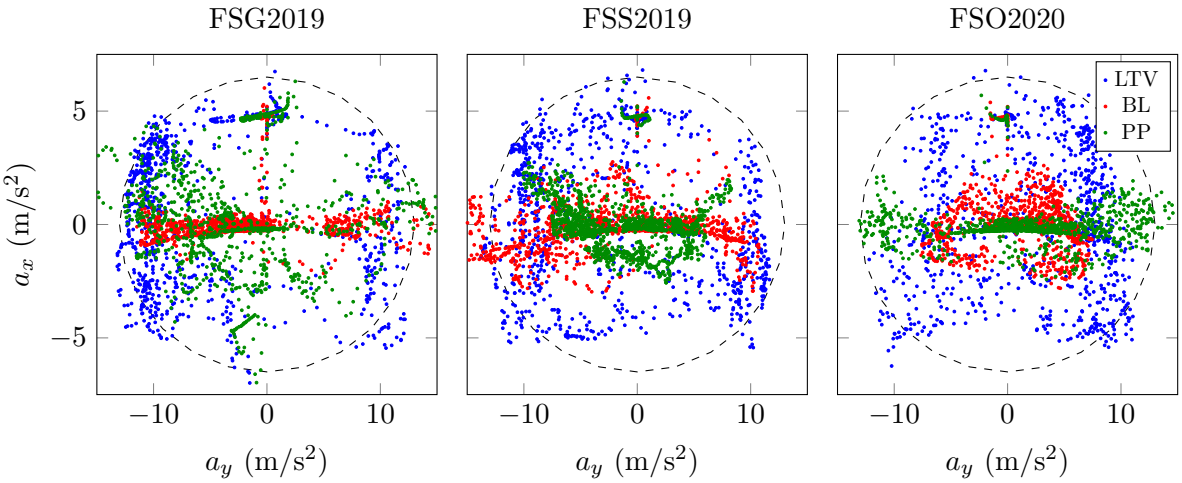

Figure 6.7: Comparison of G-G plots over the first lap of each planner. Estimation of the friction ellipse is shown as the dashed ellipses.

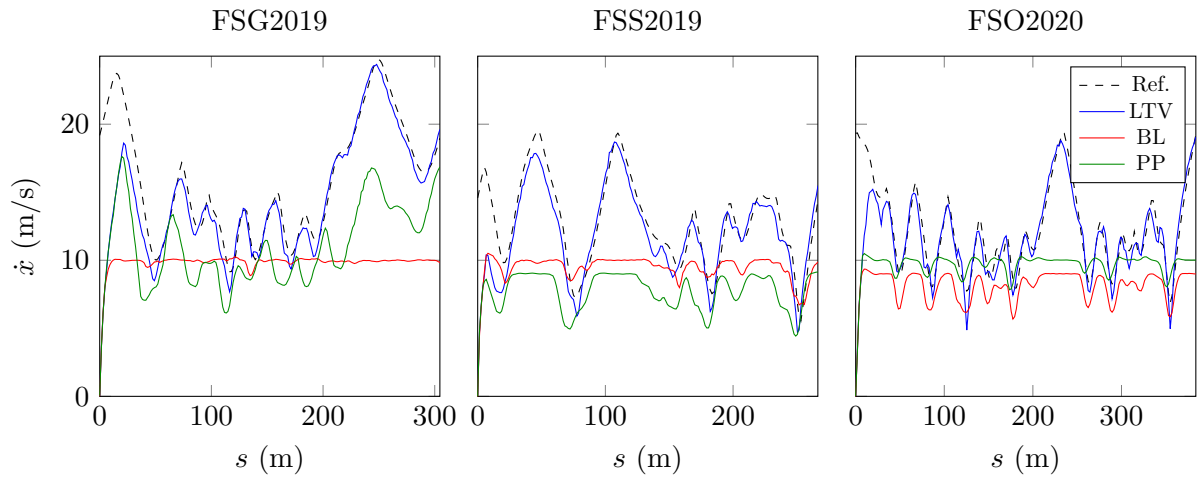

Figure 6.8: Comparison of the longitudinal velocities over the first lap of each planner. The precomputed reference velocity used for the proposed LTV-RHP is shown as the dashed line.

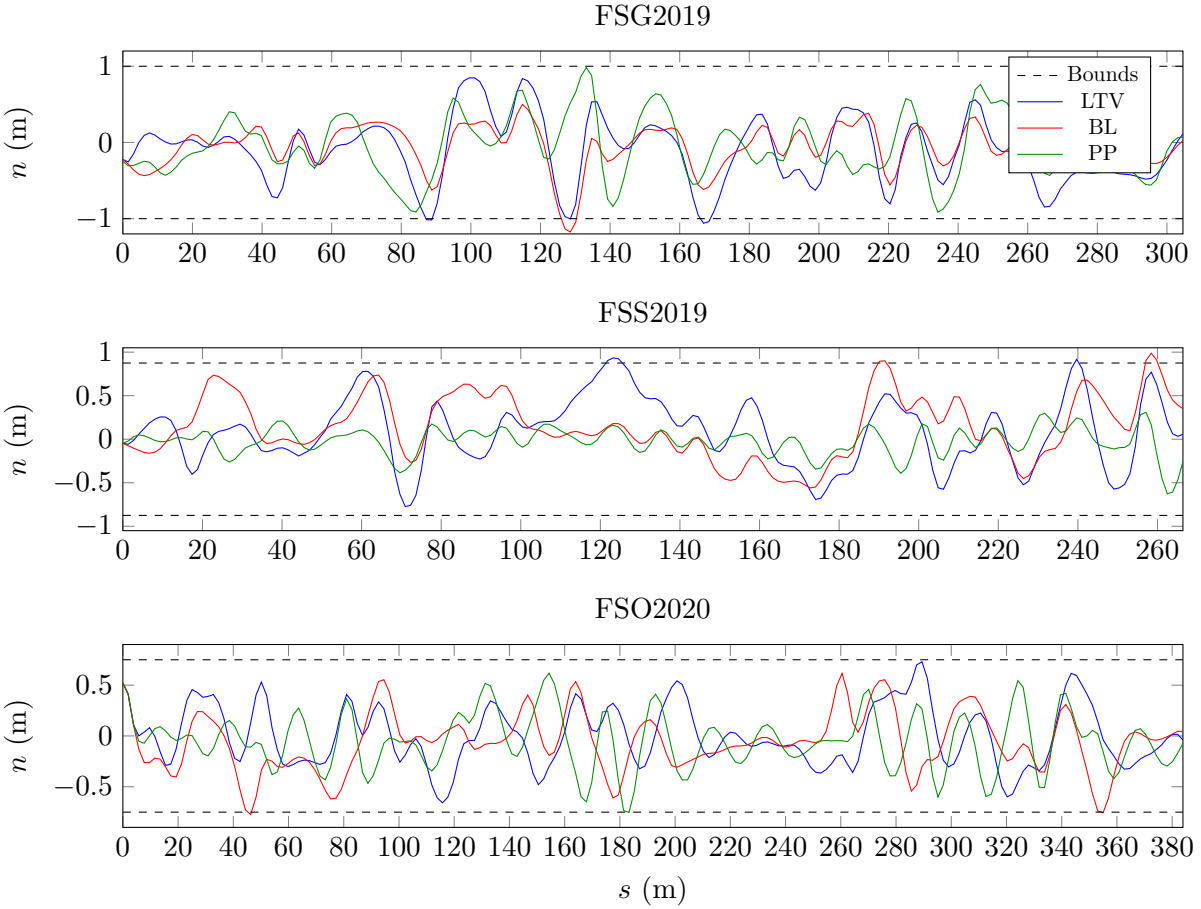

Figure 6.9: Comparison of the lateral deviations over the first lap of each planner. Lateral deviation constraints enforced on the proposed LTV-RHP are shown as dashed lines.

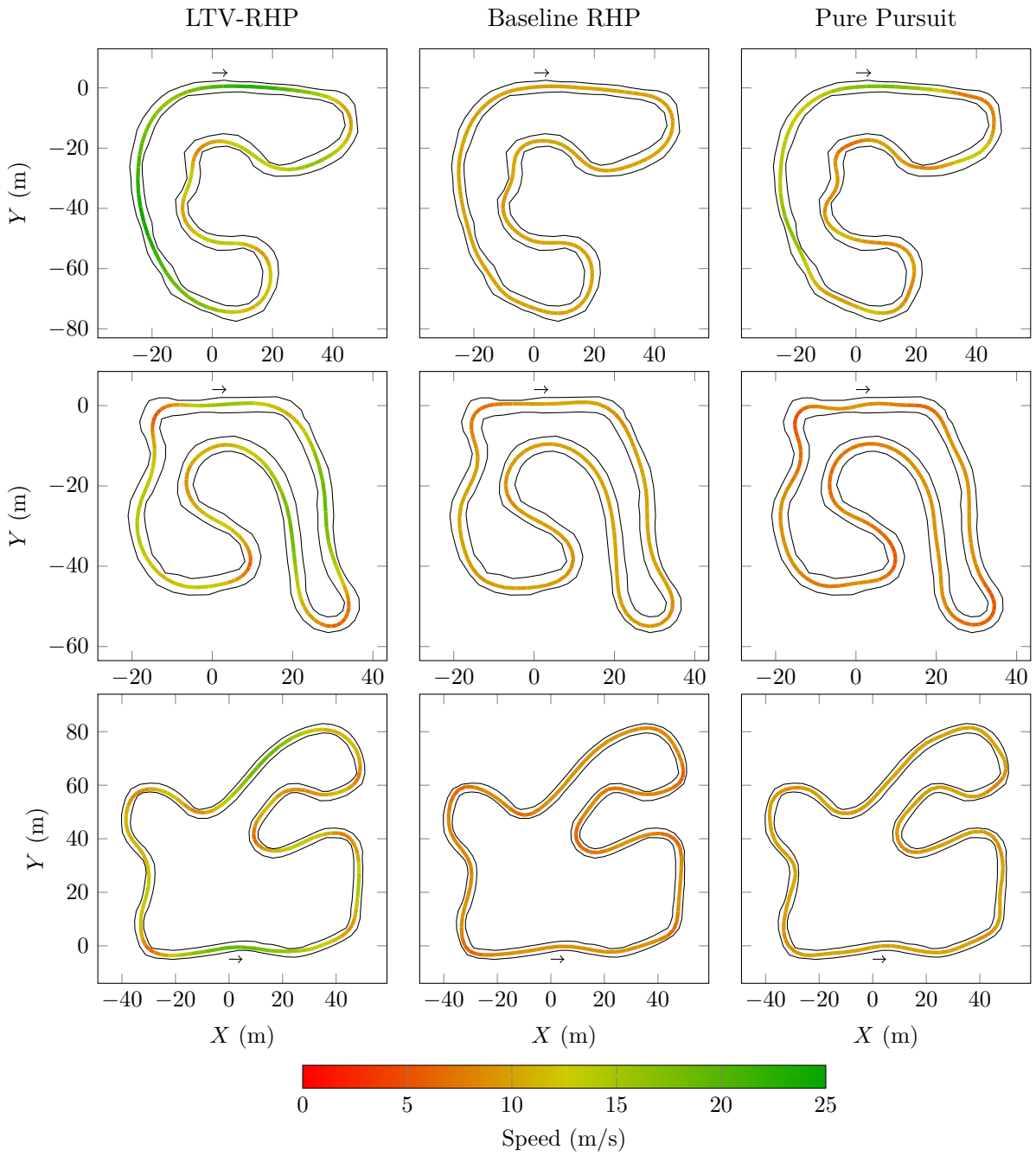

Figure 6.10: Comparison of simulated trajectories and velocity profiles over the second lap in the FSG2019 (top), FSS2019 (middle) and FSO2020 (bottom) tracks between different planners. The beginning of the tracks and directions the vehicle navigates the track in are indicated using an arrow.

6.2.3 Computation time

The distribution of computation times measured from all LTV-RHP experiments are shown in Figure 6.11. For the RHP to run online, it must have a computation time less than the time step $\Delta t = 20\,\text{ms}$. The median and maximum CPU times observed were 0.59 ms and 10.40 ms respectively, validating the real time performance capabilities of the RHP.

The computation time of the LTV-RHP can be divided into the preparation step which constructs the QP problem, and optimisation step which solves the QP problem. Although the median computation times are quite similar, the majority of the variability in computation times was due to the optimisation step. This variability was managed by setting the maximum CPU time of the qpOASES solver to 10 ms to constrain the upper bound of computation times. However, this limit was only hit once, representing 0.026% of all solutions. As stopping an optimiser before convergence means that feasibility of the optimal control solution is not guaranteed, the low probability that this maximum computation time limit is enforced is highly desirable.

The low median computation time is also beneficial. Compared to the predicted dead time of 150 ms, the RHP solve times have a negligible contribution to the dead time. Moreover, the short time step permitted by the fast solve times allowed the RHP to be run at a higher frequency, allowing for improved path tracking capabilities and reactions to disturbances. This characteristic was achieved by defining a relatively small number of time steps $N = 20$, where as discussed in Section 6.1.2, the computation time could decrease exponentially with the number of time steps used for successive linearisation schemes. However, the tradeoff is a shorter time horizon, which is undesirable as the computed control trajectory may not lead to an optimal or feasible trajectory beyond the considered time horizon. This issue is mitigated by using the precomputed racing trajectory to incorporate information about the entire track within each reference trajectory segment.

An important note with the presented computation times is that the NVIDIA Jetson AGX Xavier used on M19-D, which has an ARM CPU processor, is expected to be less powerful than the Intel i7-6500U CPU used for these experiments. From preliminary testing, the computation times on the Xavier are expected to be twice as slow. However, given that the maximum measured computation time is already half the discretised time step, this may not be an issue.

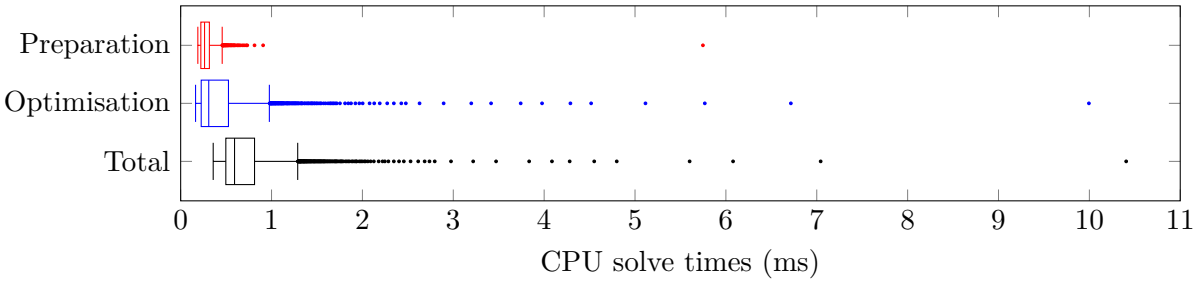

Figure 6.11: Distribution of CPU times of LTV-MPC throughout 10 laps of FSG2019, FSS2019 and FSO2020. Total computation time outliers make up 4.5% of all samples.

Chapter 7

Conclusion

This thesis presents a motion planning implementation for an autonomous Formula Student racecar to compete in the FSD competition. The proposed motion planner first computes an optimal racing line for the entire track horizon offline, then tracks this reference trajectory in real time using an LTV-RHP. The proposed RHP specifications were justified against alternative modelling and discretisation methods, before evaluating its performance in a full simulation.

By transcribing the optimal control problem to an optimisation problem, additional flexibility over the design of the RHP was achieved. Multiple discretisation methods were explored to perform this transcription, where the successive linearisation scheme was ultimately chosen for being significantly computationally lighter, while still performing adequately compared to the heavier but more accurate nonlinear discretisation schemes. The fast computation times not only aids in reducing the total dead time that needs to be compensated for, but also allows the LTV-RHP to run at a higher frequency to achieve superior tracking behaviour.

New modelling techniques were also explored given this additional flexibility. Significant performance improvements were achieved by utilising a dynamic bicycle model, which allowed the LTV-RHP to operate the vehicle closer to the limits of the vehicle dynamics, while the light successive linearisation discretisation scheme allowed the more complex vehicle model to be feasibly solved in real time. Curvilinear coordinates which required a parametric piecewise spline to describe the track path was implemented, which allowed for superior control of the car and consideration of the track boundaries. Modelling the transient response of the actuators also compensated for a significant portion of the time delay, thus reducing the amount of dead time the less stable forward-prediction compensation method needed to consider.

Overall, the proposed RHP was validated and compared against existing planners in a hardware-in-the-loop simulation environment. The RHP is successfully shown to significantly outperform MMS's previous motion planning implementations, and consistently achieve faster lap times in a variety of different tracks.

7.1 Future work

A few avenues for future work are proposed to validate or further improve the the performance of the RHP. Firstly, real-world testing of the RHP is a natural progression of the experimental work to properly validate and evaluate its on-track performance. Additional challenges expected in real-world testing include additional noise, uncertainty, disturbances and time delay which will hinder the performance and stability of the planner. The slower computing unit on the M19-D may also limit the performance of the RHP.

A second avenue for future work is to modify it to work for the autocross mission. The RHP was designed and validated for the trackdrive mission, where the track is known beforehand and therefore a reference racing trajectory could be precomputed. Several design decisions relied on this assumption, such as using a short time horizon and using a linearised discretisation scheme. Further investigation into the performance of the RHP on an unknown track and making modifications to how the RHP is implemented will allow the planner to be used for the autocross mission.

Furthermore, the dynamic bicycle model equations were derived for M19-D, which is a rear-wheel drive vehicle. However, MMS's newly developed vehicle, M21, is a four-wheel drive, which will require slight modifications to the vehicle model to be made to be used on M21. More significantly, M21 will implement torque-vectoring. By adding this as an additional control variable in the RHP, greater control over the vehicle's behaviour can be achieved, and ultimately allow lap times to be further improved. Related works include [56].

Finally, a proper comparison between the proposed discretisation scheme and that offered by optimal control libraries could be conducted. As the RHP formulations between the proposed and previous implementations were significantly different from each other, the performance differences between the discretisation schemes could not accurately be measured. Optimal control libraries often have features such as guaranteeing a feasible solution within a specified time frame, and validating the impact of these features would be an interesting exercise.

Bibliography

[1] B. Houska, H. J. Ferreau, and M. Diehl, "ACADO toolkit—an open-source framework for automatic control and dynamic optimization," *Optimal Control Applications and Methods*, vol. 32, no. 3, pp. 298–312, 2011.

[2] M. Buehler, K. Iagnemma, and S. Singh, *The 2005 DARPA grand challenge: the great robot race*. Springer, 2007, vol. 36.

[3] ——, *The DARPA urban challenge: autonomous vehicles in city traffic*. Springer, 2009, vol. 56.

[4] A. Heilmeier, A. Wischnewski, L. Hermansdorfer, J. Betz, M. Lienkamp, and B. Lohmann, "Minimum curvature trajectory planning and control for an autonomous race car," *Vehicle System Dynamics*, 2019.

[5] G. Hartmann, Z. Shiller, and A. Azaria, "Autonomous head-to-head racing in the indy autonomous challenge simulation race," *arXiv preprint arXiv:2109.05455*, 2021.

[6] K. Kritayakirana and J. C. Gerdes, "Autonomous vehicle control at the limits of handling," *International Journal of Vehicle Autonomous Systems*, vol. 10, no. 4, pp. 271–296, 2012.

[7] A. Katriniok and D. Abel, "LTV-MPC approach for lateral vehicle guidance by front steering at the limits of vehicle dynamics," in *2011 50th IEEE Conference on Decision and Control and European Control Conference*. IEEE, 2011, pp. 6828–6833.

[8] B. Alrifaee and J. Maczijewski, "Real-time trajectory optimization for autonomous vehicle racing using sequential linearization," in *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018, pp. 476–483.

[9] R. Verschueren, S. De Bruyne, M. Zanon, J. V. Frasch, and M. Diehl, "Towards time-optimal race car driving using nonlinear MPC in real-time," in *53rd IEEE conference on decision and control*. IEEE, 2014, pp. 2505–2510.

[10] J. L. Vázquez, M. Brühlmeier, A. Liniger, A. Rupenyan, and J. Lygeros, "Optimization-based hierarchical motion planning for autonomous racing," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 2397–2403.

[11] L. Cardamone, D. Loiacono, P. L. Lanzi, and A. P. Bardelli, "Searching for the optimal racing line using genetic algorithms," in *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games*. IEEE, 2010, pp. 388–394.

[12] N. R. Kapania, J. Subosits, and J. Christian Gerdes, "A sequential two-step algorithm for fast generation of vehicle racing trajectories," *Journal of Dynamic Systems, Measurement, and Control*, vol. 138, no. 9, 2016.

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

[14] A. V. Rao, "A survey of numerical methods for optimal control," *Advances in the Astronautical Sciences*, vol. 135, no. 1, pp. 497–528, 2009.

[15] N. Dal Bianco, E. Bertolazzi, F. Biral, and M. Massaro, "Comparison of direct and indirect methods for minimum lap time optimal control problems," *Vehicle System Dynamics*, vol. 57, no. 5, pp. 665–696, 2019.

[16] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[17] T. A. Johansen, "Introduction to nonlinear model predictive control and moving horizon estimation," *Selected topics on constrained and nonlinear control*, vol. 1, pp. 1–53, 2011.

[18] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[19] S. Nekkah, J. Janus, M. Boxheimer, L. Ohnemus, S. Hirsch, B. Schmidt, Y. Liu, D. Borbély, F. Keck, K. Bachmann *et al.*, "The autonomous racing software stack of the KIT19d," *arXiv preprint arXiv:2010.02828*, 2020.

[20] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat, "Predictive active steering control for autonomous vehicle systems," *IEEE Transactions on control systems technology*, vol. 15, no. 3, pp. 566–580, 2007.

[21] S. Gros, M. Zanon, R. Quirynen, A. Bemporad, and M. Diehl, "From linear to nonlinear mpc: bridging the gap via the real-time iteration," *International Journal of Control*, vol. 93, no. 1, pp. 62–80, 2020.

[22] J. A. Matute, M. Marcano, S. Diaz, and J. Perez, "Experimental validation of a kinematic bicycle model predictive control with lateral acceleration consideration," *IFAC-PapersOnLine*, vol. 52, no. 8, pp. 289–294, 2019.

[23] P. Polack, F. Altché, B. d'Andréa Novel, and A. de La Fortelle, "The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?" in *2017 IEEE intelligent vehicles symposium (IV)*. IEEE, 2017, pp. 812–818.

[24] C. Smith, *Tune to win*. Aero Publishers Fallbrook, 1978.

[25] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *2015 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2015, pp. 1094–1099.

[26] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan *et al.*, "AMZ driverless: The full autonomous racing system," *Journal of Field Robotics*, vol. 37, no. 7, pp. 1267–1294, 2020.

[27] E. Alcalá, V. Puig, J. Quevedo, and U. Rosolia, "Autonomous racing using linear parameter varying-model predictive control (LPV-MPC)," *Control Engineering Practice*, vol. 95, p. 104270, 2020.

[28] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, pp. 269–296, 2020.

[29] M. Brunner, U. Rosolia, J. Gonzales, and F. Borrelli, "Repetitive learning model predictive control: An autonomous racing example," in *2017 IEEE 56th annual conference on decision and control (CDC)*. IEEE, 2017, pp. 2545–2550.

[30] J. M. Maciejowski, *Predictive control: with constraints*. Pearson education, 2002.

[31] H. B. Pacejka and E. Bakker, "The magic formula tyre model," *Vehicle system dynamics*, vol. 21, no. S1, pp. 1–18, 1992.

[32] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *International conference on machine learning*. PMLR, 2013, pp. 1310–1318.

[33] T. D. Barfoot and C. M. Clark, "Motion planning for formations of mobile robots," *Robotics and Autonomous Systems*, vol. 46, no. 2, pp. 65–78, 2004.

[34] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, p. 3170, 2018.

[35] O. V. Polikarpotchkin and P. Lee, "Draw a smooth curve through a set of 2d points with bezier primitives," 2008.

[36] J. Weickert, B. T. H. Romeny, and M. A. Viergever, "Efficient and reliable schemes for nonlinear diffusion filtering," *IEEE transactions on image processing*, vol. 7, no. 3, pp. 398–410, 1998.

[37] M. Yarrow, "Solving periodic block tridiagonal systems using the sherman-morrison-woodbury formula," in *9th Computational Fluid Dynamics Conference*, 1989, p. 1946.

[38] H. Wang, J. Kearney, and K. Atkinson, "Arc-length parameterized spline curves for real-time simulation," in *Proc. 5th International Conference on Curves and Surfaces*, vol. 387396, 2002.

[39] ——, "Robust and efficient computation of the closest point on a spline curve," in *Proceedings of the 5th International Conference on Curves and Surfaces*, 2002, pp. 397–406.

[40] A. S. Matveev, "The instability of optimal control problems to time delay," *SIAM journal on control and optimization*, vol. 43, no. 5, pp. 1757–1786, 2005.

[41] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789–814, 2000.

[42] M. N. Zeilinger, C. N. Jones, and M. Morari, "Robust stability properties of soft constrained MPC," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5276–5282.

[43] S. Boyd, S. P. Boyd, and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[44] A. Liniger, A. Domahidi, and M. Morari, "Optimization-based autonomous racing of 1:43 scale RC cars," *Optimal Control Applications and Methods*, vol. 36, no. 5, pp. 628–647, 2015.

[45] F. Borrelli, A. Bemporad, and M. Morari, *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[46] M. Kelly, "An introduction to trajectory optimization: How to do your own direct collocation," *SIAM Review*, vol. 59, no. 4, pp. 849–904, 2017.

[47] A. Akpunar and S. Iplikci, "Runge-kutta model predictive speed control for permanent magnet synchronous motors," *Energies*, vol. 13, no. 5, p. 1216, 2020.

[48] G. Liu, J. Mu, D. Rees, and S. Chai, "Design and stability analysis of networked control systems with random communication time delay using the modified MPC," *International Journal of Control*, vol. 79, no. 4, pp. 288–297, 2006.

[49] S. Trimboli, S. Di Cairano, A. Bemporad, and I. V. Kolmanovsky, "Model predictive control for automotive time-delay processes: An application to air-to-fuel ratio control," *IFAC Proceedings Volumes*, vol. 42, no. 14, pp. 90–95, 2009.

[50] M. Diehl, H. J. Ferreau, and N. Haverbeke, "Efficient numerical methods for nonlinear MPC and moving horizon estimation," in *Nonlinear model predictive control*. Springer, 2009, pp. 391–417.

[51] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpOASES: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, no. 4, pp. 327–363, 2014.

[52] M. D. Lam, E. E. Rothberg, and M. E. Wolf, "The cache performance and optimizations of blocked algorithms," *ACM SIGOPS Operating Systems Review*, vol. 25, no. Special Issue, pp. 63–74, 1991.

[53] G. Guennebaud, B. Jacob *et al.*, "Eigen v3," http://eigen.tuxfamily.org, 2010.

[54] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng *et al.*, "ROS: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[55] A. Slomoi, "Path planning and control in an autonomous formula student vehicle," Bachelor's Thesis, Monash University, 2018.

[56] S. Srinivasan, S. N. Giles, and A. Liniger, "A holistic motion planning and control solution to challenge a professional racecar driver," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 7854–7860, 2021.

Appendix A

Nonlinear programming matrices

A.1 Multiple shooting

The objective function gradient is expressed as:

$$\nabla J_\epsilon(\Xi) = \begin{bmatrix} 2Q(\xi_1 - \xi_{l,1}) - 2MRe_1 \\ 2Re_0 \\ \vdots \\ 2Q(\xi_{N-1} - \xi_{l,N-1}) - 2MRe_{N-1} \\ 2Re_{N-2} \\ 2Q(\xi_1 - \xi_{l,1}) \\ 2Re_{N-1} \end{bmatrix} \tag{A.1}$$

where M is a matrix which maps the error e to the corresponding state variables.

The constraint vector is represented as a column-stacked vector of plant model constraints and path constraints:

$$c(\Xi) = \begin{bmatrix} \xi_0 + f_{(\cdot)}(\xi_0, u_0) - x_1 \\ \vdots \\ \xi_{N-1} + f_{(\cdot)}(\xi_{N-1}, u_{N-1}) - \xi_N \\ g(\xi_0, u_0, \epsilon) \\ \vdots \\ g(\xi_{N-1}, u_{N-1}, \epsilon) \end{bmatrix} \tag{A.2}$$

where M is a matrix which maps the error e to the corresponding state variables.

The constraint Jacobian can be expressed as:

$$\nabla c(\Xi) = \begin{bmatrix} F \\ G \end{bmatrix} \tag{A.3}$$

where F is the portion of the Jacobian arising from the model constraints, and G is the portion

of the Jacobian arising from the other functional constraints.

$$F = \begin{bmatrix} -I & \tilde{B}_0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & 0_\epsilon \\ \tilde{A}_1 & 0 & -I & \tilde{B}_1 & \dots & 0 & 0 & 0 & 0 & 0_\epsilon \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & -I & \tilde{B}_{N-2} & 0 & 0 & 0_\epsilon \\ 0 & 0 & 0 & 0 & \dots & \tilde{A}_{N-1} & 0 & -I & \tilde{B}_{N-1} & 0_\epsilon \end{bmatrix} \tag{A.4a}$$

$$G = \begin{bmatrix} 0 & D_0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & E \\ C_1 & 0 & 0 & D_1 & \dots & 0 & 0 & 0 & 0 & E \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 0 & D_{N-2} & 0 & 0 & E \\ 0 & 0 & 0 & 0 & \dots & C_{N-1} & 0 & 0 & D_{N-1} & E \end{bmatrix} \tag{A.4b}$$

A.2 Trapezoidal collocation

The objective function gradient is expressed as:

$$\nabla J_\epsilon(\Xi) = \begin{bmatrix} Q(\xi_0 - \xi_{r,0}) - MRe_0 \\ Re_0 \\ 2Q(\xi_1 - \xi_{r,1}) - 2MRe_1 \\ 2Re_1 \\ \vdots \\ 2Q(\xi_{N-1} - \xi_{r,N-1}) - 2MRe_{N-1} \\ 2Re_{N-1} \\ Q(\xi_N - \xi_{r,N}) - MRe_N \\ Re_N \end{bmatrix} \tag{A.5}$$

The constraint vector is represented as a column-stacked vector of collocation constraints and path constraints:

$$c(\Xi) = \begin{bmatrix} \xi_0 + \frac{1}{2}(f(\xi_0, u_0) + f(\xi_1, u_1))\Delta t - \xi_1 \\ \vdots \\ \xi_{N-1} + \frac{1}{2}(f(\xi_{N-1}, u_{N-1}) + f(\xi_N, u_N))\Delta t - \xi_N \\ g(\xi_0, u_0, \epsilon) \\ \vdots \\ g(\xi_N, u_N, \epsilon) \end{bmatrix} \tag{A.6}$$

The constraint Jacobian can be expressed as:

$$\nabla c(\Xi) = \begin{bmatrix} F \\ G \end{bmatrix} \tag{A.7}$$

where F is the portion of the Jacobian arising from the model constraints, and G is the portion of the Jacobian arising from the other functional constraints.

$$F = \begin{bmatrix} I + \frac{1}{2}A_0\Delta t & \frac{1}{2}B_0\Delta t & \dots & 0 & 0 & 0_\epsilon \\ 0 & 0 & \dots & 0 & 0 & 0_\epsilon \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0_\epsilon \\ 0 & 0 & \dots & -I + \frac{1}{2}A_N\Delta t & \frac{1}{2}B_N\Delta t & 0_\epsilon \end{bmatrix} \tag{A.8a}$$

$$G = \begin{bmatrix} C_0 & D_0 & 0 & 0 & \dots & 0 & 0 & 0 & 0 & E \\ 0 & 0 & C_1 & D_1 & \dots & 0 & 0 & 0 & 0 & E \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & \dots & C_{N-2} & D_{N-2} & 0 & 0 & E \\ 0 & 0 & 0 & 0 & \dots & 0 & 0 & C_{N-1} & D_{N-1} & E \end{bmatrix} \tag{A.8b}$$