TÉCNICO LISBOA

Learning to Drive Autonomously a Race Car

João Gomes de Oliveira Pinho

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisors: Prof. Pedro Manuel Urbano de Almeida Lima
Prof. Miguel Afonso Dias de Ayala Botto

Examination Committee

Chairperson: Prof. Duarte Pedro Mata de Oliveira Valério
Supervisor: Prof. Miguel Afonso Dias de Ayala Botto
Member of the Committee: Prof. João Manuel Lage de Miranda Lemos

September 2021

Dedicated to engineering and science.

Acknowledgments

First, I would like to thank both my supervisors - professors Pedro and Miguel - for their support. I have learned a lot throughout the course of our meetings and the fruitful discussions therein. I extend that gratitude to Gabriel, a fellow master thesis student, who joined our meetings as he started working in the same research topic. In particular, for further enriching these meetings and for the stimulating discussions and productive cooperation we endured outside the meetings.

I would like to thank FST Lisboa as an organization and the people I have met throughout my tenure there, from alumni to teammates. I have learned tremendously and appreciate the singular opportunity of building fast race cars. I have made some good friends there of whom I highlight Revés and Crespo. Lastly, a special thank you to both teams that have built our first autonomous race car. I especially thank Pedro, Francisco, Luís, Morgado, Amaral, Ivo and Miguel for their unrelenting dedication. Despite the struggles inherent in building an autonomous Formula Student race car, I appreciate the time well spent together, at night in covil and in the blazing sun of Vendas Novas. What a team we've built. Cheers for that!

My time in Técnico also meant making new friends of whom I would like to thank those that became very close friends: Xana, Mariana, Pimentinha, Alda, Miguel, Cuco, Guilherme, Tomás and Constança. We have shared already some great experiences from rally tascas to kayak trips, and exploring Portugal or visiting each other abroad. I hope to continue sharing great moments with you! I also thank those friends from the earlier Técnico days and with whom I have developed great pieces of work, namely Vasco, João, João and João.

Special thanks to the Judo All In family for the lifelong friendship. Thank you Alexandre, Miguel, Marcos, Francisco, Bernardo and Xavier. Also, thanks to those who we welcomed later: Kunta, Pedro and Pedro. Slightly more special thanks to Ricardo!

Finally, but most importantly, I wholeheartedly thank my family and girlfriend for their unconditional love. I further thank (apologize would probably be a more appropriate word) my sisters for enduring my provocations. I also thank my parents for their complete support in the broadest scope possible which allowed to fully commit myself to whatever project I wished from Formula Student to this master thesis. I thank Francisca for all the time we spend together. You make me happy.

Resumo

Nesta tese, apresentamos a adaptação de uma arquitetura de Controlo por Modelo Preditivo (MPC) baseada em Aprendizagem para corridas autónomas à competição Formula Student Driverless (FSD). Este controlador aprende sobre iterações passadas através da construção de determinados conjunto e funcional de custo terminal por base em trajetórias e sequências de ação de controlo recolhidas. Melhoramos a capacidade de atuação em tempo-real para que satisfaça os requisitos de FSD ao implementar o controlador em C++ e resolver a optimização num programa comercial desenhado para a solução embebida de MPCs. Uma das principais dificuldades na corrida autónoma é que os modelos que cobrem toda a janela dinâmica são não-lineares e difíceis de identificar. Para atenuar este problem, usamos medidas passadas e atuais e técnicas de aprendizagem automática para prever o error do modelo nominal. Em particular, usamos duas aproximações esparsas de Gaussian Process Regression (GPR) para aprendizagem do erro do modelo. Em seguida, testamos este controlador num simulador de veículo dedicado à competição FSD. Mostramos que a arquitectura original é capaz de melhorar a sua performance, medida em tempos de volta, em cerca de 10%. No entanto, a disparidade entre o modelo nominal e as medidas torna-se bastante acentuada à medida que o controlador se torna incrementalmente mais agressivo o que resulta no não cumprimento do constrangimento da largura da pista. Seguidamente, demonstramos que GPR consegue reduzir o erro do modelo nominal até 75% e reduzir o tempo de volta até 12%.

Palavras-chave: Controlo por Modelo Preditivo, Aprendizagem de Modelo para Controlo, Aprendizagem e Sistemas Adaptativos, Trajetória e Seguimento de Caminho, Corrida Autónoma.

Abstract

In this thesis, we present the adaptation of a Learning-based Model Predictive Control (LMPC) architecture for autonomous racing to the Formula Student Driverless (FSD) context. This reference-free controller is able to learn from previous iterations by building an appropriate terminal set and cost function from collected trajectories and input sequences. We improve the real-time capability of the framework to satisfy the FSD requirements by implementing the controller in C++ and solving the optimization in a commercial solver designed for embedded solutions of MPCs. One major setback in autonomous racing is that accurate vehicle models that cover the entire performance envelope are highly nonlinear and difficult to identify. To address this problem, we use both past and current measurements and machine learning techniques to predict the nominal model error. In particular, we use two sparse Gaussian Process Regression (GPR) approximations for model learning. We then test this controller in a vehicle simulator dedicated to the FSD competition. We show that the original architecture is able to improve its performance by around 10% measured as lap time reduction. However, the nominal model mismatch becomes severe as the controller pushes for incrementally more aggressive behaviour which results in not fully abiding by the track width constraint. We then employ the GPR model which is able to reduce the nominal model error by as much as 75% and safely improve the lap times by up to 12%. We have tested on two different tracks to show signs of the framework being track agnostic.

Keywords: Model Predictive Control, Model Learning for Control, Learning and Adaptive Systems, Motion and Path Planning, Autonomous Racing.

Contents

List of Tables

List of Figures

List of Algorithms

Nomenclature

Miscellaneous

$\boldsymbol{\theta}$	Coefficients of Least Mean Square problem.
δ_{ij}	Kronecker delta.
κ	Path curvature.
ρ	Air density.

Learning-based Model Predictive Control

α_i	Convex set coefficients.
$\mathcal{SS}$	Safe set.
$\mathbf{D}$	Local safe set.
$\mathbf{u}$	Control input vector.
$\mathbf{x}$	State vector.
c	Candidate terminal state.
e_y	Lateral deviation from the centerline.
e_y^{max}	Maximum lateral deviation from the centerline.
e_ψ	Heading error from the centerline.
N	Prediction horizon.
P	Acceleration pedal setpoint.
Q	Q-function.
q	Stage cost.
s	Track progress measured along centerline.
x_F	Final state.
z^{opt}	Optimization variables.

Gaussian Process Regression

$\bar{g}$	Latent variables or inducing points.
$\boldsymbol{\phi}$	Vector of hyperparameters.
$\boldsymbol{\Sigma}$	Variance matrix.
$\mathcal{D}$	Training dataset.
$\mathcal{N}$	Gaussian (Normal) distribution.
$\mathbf{K}$	Covariance matrix.
$\mathbf{Y}$	Matrix of training targets - multiple models.
$\mathbf{Z}$	Matrix of training points.
μ	Mean function of a GP.
σ_f^2	Signal variance.
σ_n^2	Noise variance.
$\mathbf{i}$	Information vector.
$\mathbf{y}$	Vector of training targets - single model.
$\mathbf{z}$	Input feature.
k	Covariance (or kernel) function.
l	Length-scale.
m	Number of inducing points.
n	Number of training cases for a Full Gaussian Process Regression model.
n_g	Number of GP outputs.
n_z	Number of GP inputs.

Vehicle Properties

$\alpha_{F,R}$	Tire slip angle.
δ	Steering angle.
ψ	Yaw angle.
C_D	Coefficient of drag.
C_L	Coefficient of downforce.
F	Force.

I Inertia.

a Acceleration.

m Mass.

r_{wheel} Tire radius.

v Velocity.

Subscripts

F, R Front and rear vehicle axles.

x, y, z Cartesian components.

Glossary

AD	Autonomous Driving.
ARD	Automatic Relevance Determination.
AV	Autonomous Vehicle.
BARC	Berkeley Autonomous Race Car.
BLR	Bayesian Linear Regression.
CNN	Convolutional Neural Networks.
CV	Cross-validation.
FIC	Fully Independent Conditional.
FITC	Fully Independent Training Conditional.
FSD	Formula Student Driverless.
FSG	Formula Student Germany.
FSI	Formula Student Italy.
FS	Formula Student.
FTCOC	Finite Time Constrained Optimal Control.
FoV	Field of View.
GPR	Gaussian Processes Regression.
LMPC	Learning-based Model Predictive Control.
LMS	Least Mean Square.
LiDAR	Light Detection And Ranging.
ML	Machine Learning.
MPCC	Model Predictive Contouring Control.
MPC	Model Predictive Control.
NLP	Nonlinear Programming.
NMPC	Nonlinear Model Predictive Control.
ODD	Operation Design Domain.
PITC	Partially Independent Training Conditional.
PU	Processing Unit.
RHC	Receding Horizon Control.
RL	Reinforcement Learning.
ROS	Robot Operating System.

SE Squared Exponential.

SLAM Simultaneous Localization and Mapping.

SVM Support Vector Machine.

SoD Subset of Data.

SoR Subset of Regressors.

Chapter 1

Introduction

1.1 Motivation

Autonomous driving (AD) has been an increasingly active field of research for academia and the industry, especially in the last two decades. This has been paralleled by growing amounts of investment from the automotive industry's established Original Equipment Manufacturers in autonomous vehicle (AV) startups and their own research labs [1]. Most stakeholders involved aim at improving road safety [2], reducing traffic congestion and emissions, and deploying universal mobility [3]. It holds great promises of societal progress.

Arguably, the pioneering work in autonomous driving was performed by professor Ernst Dickmanns and his team in the 80s and 90s. The culmination point of his team's endeavours was the 1995 drive from Munich to Odense in Denmark. The autonomous S-Class Mercedes-Benz drove 95% of the $1758\,\mathrm{km}$ autonomously and reached a peek velocity of $175\,\mathrm{km\,h^{-1}}$. Notwithstanding, the catalyst of recent attention was likely the 2005 DARPA Grand Challenge, where Stanley [4], the Stanford University team's vehicle led by Sebastian Thrun, won the competition, followed by Carnegie Mellon University's car - Sandstorm. Thrun is a prominent figure in the self-driving and robotics field, having contributed to Simultaneous Localization and Mapping (SLAM) algorithms [5, 6], which is a key part of a usual autonomous driving software pipeline.

The typical AD software pipeline can generically be represented by the scheme in Figure 1.1 [7, 8]. From left to right, the first block corresponds to sensing which concerns the collection of data from a multitude of sensors such as Light Detection And Ranging (LiDAR) and cameras. Perceive or Perception algorithms - the first three algorithms of the second block - aim to detect road features and agents. The remaining two techniques in the second block are examples of approaches used to localize the vehicle with respect to some map of the world. The third block, in particular Behaviour Prediction, targets the interpretation of Perception and Localization data to make sense of the real-world, *e.g.* estimate other agents' intents and driving style. The fourth block's algorithms plan a feasible trajectory that respects traffic rules given some route indications. Finally, control algorithms - last block - compute the necessary actuation of the to follow the desired trajectory.

Perception uses exteroceptive sensors' data, *e.g.* from LiDAR, camera and radar, to detect and track other road agents, such as cyclists and pedestrians, and road signals [9]. State of the art perception algorithms frequently resort to Deep Learning techniques [10, 11]. Furthermore, this module may be responsible for creating information-rich instance segmentation images [12].

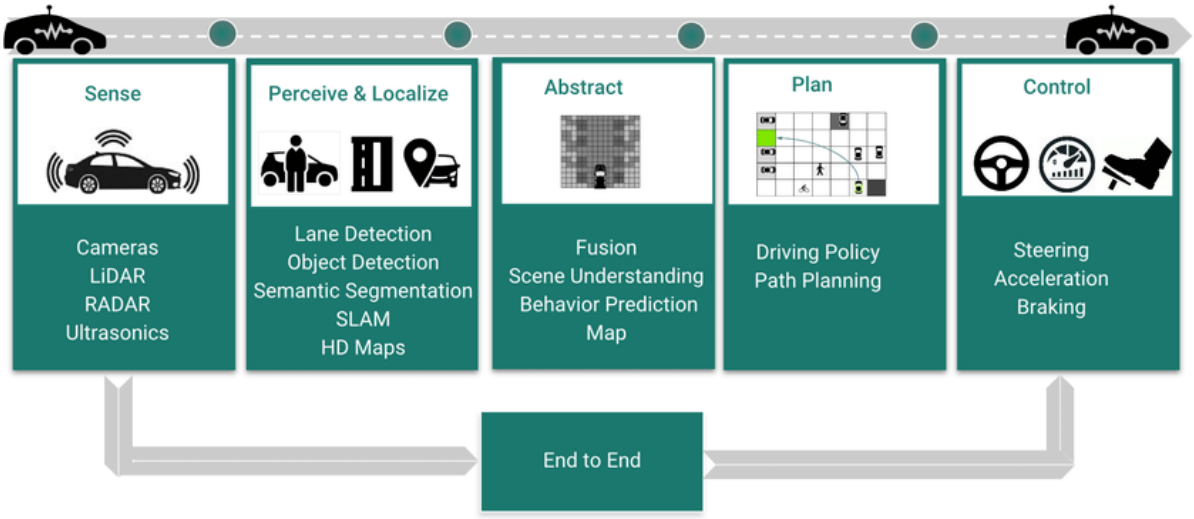


Figure 1.1: Autonomous Driving Software Pipeline [reprinted from [13]]

The Localization module aims to determine the ego-position with respect to an external reference frame. Most successful real-world implementations compare online readings with pre-built high-definition maps [14] to estimate the vehicle's pose [15]. Proprioceptive sensors, *e.g.* Inertial Measurement Units and wheel encoders, provide information for internal vehicle state monitoring and odometry, which is also used in this module.

Behaviour Prediction is a fundamental part of urban AV platforms. It aims to predict other agents' future trajectories and driving style [16] and [17]. Planning or Decision takes all this information to compute a feasible motion based on the existing obstacles and reference path [18]. However, in complex scenarios such as roundabouts, it is often hard to hand-code rules that can generalize well. [19] proposes learning models of behaviour from human demonstrations of unlabelled raw video data. Finally, Control algorithms compute the necessary actuation to follow the trajectory reference [20].

These technologies have numerous applications ranging from personal vehicles to long-haul trucks and shuttles to mining vehicles. The American Society of Automotive Engineer proposes a taxonomy for driving automation systems [21]. It provides detailed definitions for six levels of driving automation, ranging from no driving automation (level 0) to full driving automation (level 5).

Audi, a german automaker, claims to have been the first to deliver a production car with level 3[1], *i.e.* conditional, automation. This level enables drivers to focus on other tasks so as long as the vehicle remains in a given Operational Design Domain (ODD) - in this case, highways. While level 5 automation corresponding to fully autonomous driving in any road or weather has not been achieved yet, the same is not true for level 4 automation - autonomous driving in a limited ODD. A few companies have been authorized by California's Department of Motor Vehicles to operate with a safety driver provided some

[1] `https://store.hbr.org/product/audi-a8-the-world-s-first-level-3-autonomous-vehicle/W20134`

weather conditions are met and within a well-defined perimeter of public roads. Waymo, owned by Google's parent company Alphabet, is one of such companies, having driven more than 20 million miles on the road and more than 15 billion miles in simulation - Figure 1.2. Zoox has also been given such a permit. This startup aims to redefine the transportation landscape by offering Mobility as a Service (MaaS) with its four-wheel steering, bi-directional, electric car. Nuro, an autonomous delivery startup, has too vehicles with level 4 automation on the streets. The trucking industry has also been targeted with automation, with TuSimple and Aurora having achieved significant progress.

Autonomous Racing aims to contribute to Autonomous Driving's broader problem by introducing innovations in autonomous technology through sport [22]. This kind of synergy is well established between Formula One and the automotive industry. For instance, technologies such as disk brakes, active suspension, energy recovery systems and traction control were developed within this motorsport competition and afterwards saw widespread use in commercial vehicles. This is exactly what Roborace strives to achieve: combine motorsport entertainment and innovation to accelerate enhancements to road safety. Currently, in its Season Beta, six teams are competing with Dev Bot 2.0, Figure 1.3, on fully autonomous time trial challenges that include mixed reality obstacles and time benefits.

F1TENTH is an international community that has designed the F1TENTH Autonomous Vehicle System an open-source platform for autonomous systems research and education [23]. International competitions are organized where teams compete against each other with the 1/10 scale race car.

The Indy Autonomous Challenge is promoting a university competition to program a modified Dallara IL-15 racecar, Figure 1.4, in the world's first head-to-head, high-speed autonomous race at the Indianapolis Motor Speedway. The race is set to take place in October 2021. The organizers also highlight the fundamental connection between innovations on the racetrack and real-world improvements. This is yet another statement to the aforementioned ever-increasing interest, which is likely to result in more mature AD technology.

Moreover, in the Stanford Dynamic Design Lab, Professor Chris Gerdes' team has adapted MARTY - a 1981 DMC DeLorean, Figure 1.5 - to equip it with autonomous drifting capacities [24]. This lab raises the issue that preventing drifting is actually narrowing the range of scenarios where an AV can safely operate. Often, vehicles are restricted to stay within conservative limits wherein it is always stable and easy to control. Thus, sacrificing agility as drifting is not a skill showed by regular drivers. However, a vehicle can achieve a wider range of manoeuvres by understanding how to control it beyond the stability limits.

Formula Student hereafter referred to as FS, is a student engineering design competition where participating university teams design, build, test and compete with a single seat formula racecar. There are two classes for the propulsion system: electric (EV) and internal combustion engine (CbV). Teams are evaluated not only by how fast their car is but also by their design and manufacturing choices.

FS competitions are divided into a series of static and dynamic events. In the Business Plan Presentation, the goal is to evaluate the team's ability to develop and deliver a comprehensive business model which demonstrates their product – a prototype race car – could become a rewarding business opportunity that creates a monetary profit. The Cost and Manufacturing event's objective is to evalu-

Figure 1.2: Waymo Car

Figure 1.3: Roborace DevBot 2.0

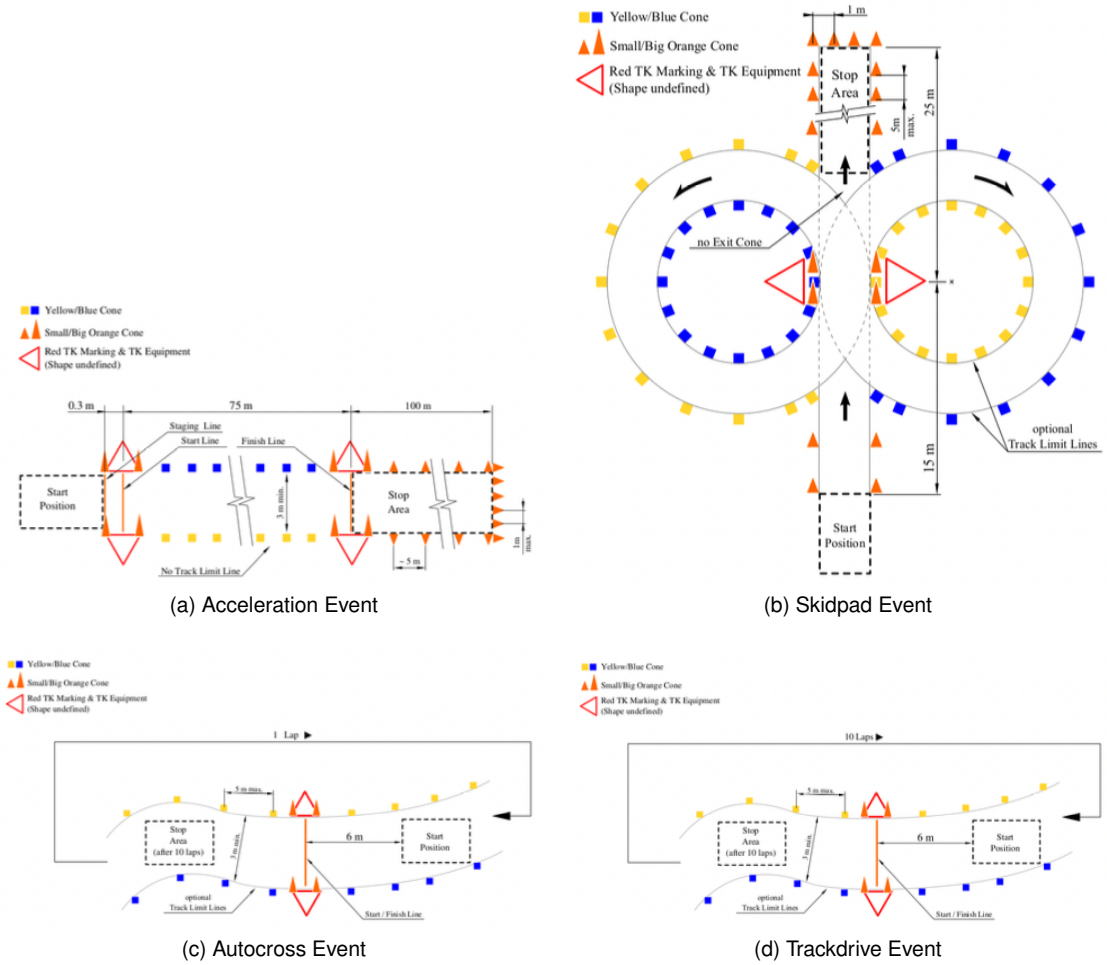
Figure 1.4: Dallara IL-15

Figure 1.5: MARTY

ate the team's understanding of the manufacturing processes and costs associated with constructing a prototype race car. This includes trade-off decisions between content and cost, make or buy decisions and understanding the differences between prototype and mass production. Lastly, in the Engineering Design event, the teams must support their engineering process and compose a design report. The author refers to Table 1.1 for the maximum points awarded per dynamic event. The dynamic events, see Figure 1.6, aim to test the fabricated prototype in different conditions, as follows:

- **Acceleration**: straight line with a length of 75 m - test longitudinal acceleration capability.
- **Skidpad**: two pairs of concentric circles in a figure of eight pattern - test lateral acceleration capability.
- **Autocross**: single lap in a handling track with components such as hairpins, slaloms and chicanes with a length of approximately 1 km.
- **Trackdrive**: 10 laps on a closed-loop circuit with similar characteristics to Autocross.
- **Efficiency**: evaluate the vehicle's energy efficiency in the previous event.

In 2017, Formula Student Germany (FSG) decided to create an additional class for autonomous racecars - Driverless Vehicle (DV). In this class, the vehicles must complete the dynamic events in fully autonomous mode. These cars could either be powered by electric motors or internal combustion engines. This decision was influenced by the automotive stakeholders like Audi, Daimler or Continental that provide officials and design judges to this competition, and that are also sponsors.



(a) Acceleration Event

(b) Skidpad Event

(c) Autocross Event

(d) Trackdrive Event

Figure 1.6: Dynamic Events Track Layout

From 2022 onwards, the DV class will cease to exist independently. Instead, CbV and EV categories will have to perform one of the dynamic events in an autonomous mode. If teams are not capable of doing so, they will not score on this discipline. In the following year, 2023, an additional dynamic event will also have to be performed autonomously. Hence, reinforcing the incentive of developing this technology.

The autonomous driving competition environment is a rather controlled one. For instance, no other agents, such as other vehicles or pedestrians, are immediately near the track. The track is composed of blue and yellow cones on the left and right borders, respectively. Exit and entry lanes are marked with small orange cones when applicable, while big orange cones are used in the start, finish and timekeeping line. Therefore, a common Formula Student Driverless (FSD) software pipeline is simpler. Behaviour Prediction and Decision modules are no longer necessary. Additionally, the Perception challenge is also simpler as there are only four classes of stationary objects to detect and track. In contrast, the Motion Planning and Control problem can be harder as teams aim to go as fast as possible and drive at the limits of handling, *i.e.* exploiting the whole performance envelope. A priori track information is not allowed for the Autocross event, while for the Trackdrive, some competitions allow teams to collect data from their Autocross runs. Therefore, the SLAM problem is a fundamental part of the pipeline.

	CbV \ EV	DV
Static Events:		
Business Plan Presentation	75 points	75 points
Cost and Manufacturing	100 points	100 points
Engineering Design	150 points	300 points
Dynamic Events:		
Skidpad	75 points	75 points
Acceleration	75 points	75 points
Autocross	100 points	100 points
Endurance	325 points	-
Efficiency	100 points	75 points
Trackdrive	-	200 points
Overall	1000 points	1000 points

Table 1.1: Maximum Points Awarded per Event

Nevertheless, teams should be able to complete any event using current measurements only, though, at a slower speed.

FST Lisboa is the Instituto Superior Técnico FS team. It has been fabricating these prototypes since 2001. Since September 2019, the team has been working on its first autonomous prototype - FST10d. The base vehicle to be adapted is the 9^{th} FST Lisboa's prototype (Figure 1.7), its 6^{th} electric. It was designed, built and put to competition during the 2018/19 season, where it achieved the 9^{th} place in the overall classification in FSG 2019.

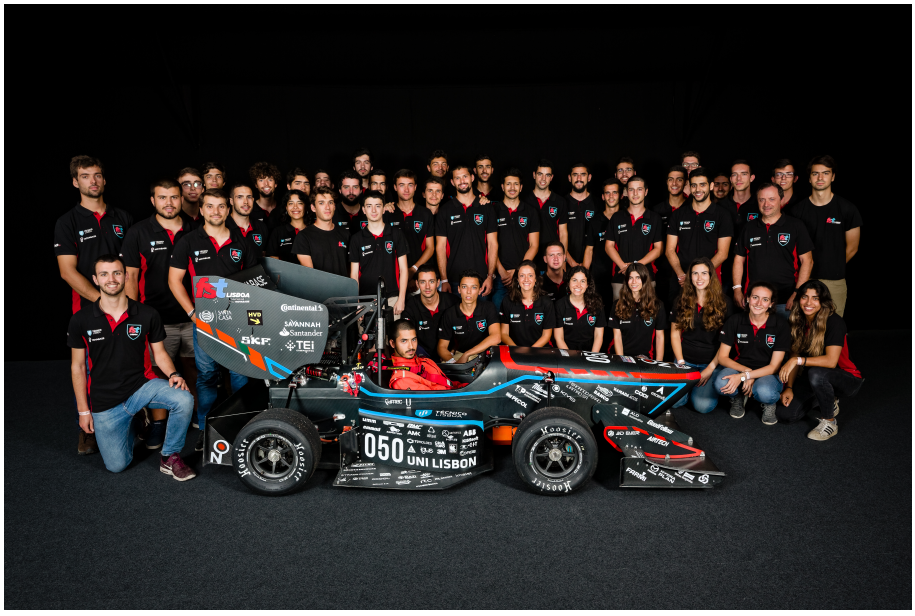


Figure 1.7: FST09e Car and Team at FSG19 [©FSG - Johannes Klein]

The current FST Lisboa's software pipeline includes only simple controllers, while the rest of the pipeline is more mature - more details on this in Section 4.2. The controllers implemented are a Pure Pursuit Controller [25] for lateral control and Proportional-Integral (PI) controller for longitudinal control. The latter receives a velocity setpoint computed from a point-mass model. Therefore, this thesis work aims to contribute to the FST Lisboa's Path Planning and Control pipeline and, in turn, to the AD endeavour.

1.2 Topic Overview

Paden et al. [26] surveyed the state of the art on planning and control algorithms for AVs in the urban setting. The authors divide the motion planning layer into three main categories. *Variational methods* represent the path as a function parameterized by a finite-dimensional vector, and the optimal path is sought by optimizing over the vector parameter using non-linear continuous optimization techniques [27]. On the other hand, *graph-search methods* discretize the configuration space of the vehicle as a graph, where the vertices represent a finite collection of vehicle configurations and the edges represent transitions between vertices. The desired path is found by performing a search for a minimum-cost path in such a graph [28, 29]. Finally, *incremental search methods* sample the configuration space and incrementally build a reachability graph that maintains a discrete set of reachable configurations and feasible transitions between them. Once the graph is large enough so that at least one node is in the goal region, the desired path is obtained by tracing the edges that lead to that node from the start configuration [30, 31]. Regarding control techniques, the authors claim that the feedback controller's role is to stabilize to the reference path or trajectory in the presence of modelling error and other forms of uncertainty. Several controllers that resort to a kinematic bicycle model have been designed. For instance, the Pure Pursuit [25] and Stanley [4] controllers. To handle more demanding driving manoeuvres, more complex controllers must be designed. In particular, predictive control approaches have achieved some success [32].

Model Predictive Control applied to AD consists of solving the motion planning problem over a short time horizon - the *Prediction Horizon* - and applying the corresponding control action to the physical system. This problem is solved at each iteration starting at the current state, therefore acting as a feedback loop - *Receding Horizon Control*. Advances in computing hardware and mathematical programming algorithms have made predictive control feasible for real-time use in AVs. There are several examples of MPC applied to AD [33–35].

On the opposite side is end-to-end driving which aims to generate ego-motion directly from sensory inputs. Direct supervised Deep Learning [36] has been used for this problem, but more recent focus has been put to Deep Reinforcement Learning [37]. Sutton and Barto [38] define Reinforcement Learning (RL) as simultaneously a problem and a class of solution methods where the agent learns to map situations to optimal actions, *i.e.* the optimal policy, to maximize a future cumulative reward not by being explicitly told so but, instead, by learning from interaction with the environment. Deep RL has been applied with remarkable success in controlled simulated environments where state transitions are known precisely. For instance, it has achieved human-level performance in Atari games [39] and in the board game Go [40].

Kiran et al. [41] survey the applications of RL to autonomous driving. While there are a reasonable number of such applications in a simulation environment, *e.g.* [42], the number of real-world successful experiments is far more limited. Wayve, a deep RL-based AD startup, claims to have been the first to show that RL is a viable approach to AD [43]. To this end, the authors have demonstrated the ability to learn to lane follow in a $250\,\mathrm{m}$ off-road section, with under thirty minutes of training. Nonetheless,

there are significant barriers to the deployment of this technology in a real-world scenario. Namely, in bridging the simulation-reality gap, validating and ensuring these systems' safety, and being data-efficient. Further, Dulac-Arnold et al. [44] perform an empirical investigation of the challenges of real-world RL and highlight nine major challenges, *e.g.* high-dimensionality of continuous state and action space. Finally, Recht [45] uses the Linear Quadratic Regulator, a simple and well-studied problem in optimal control, with unknown dynamics as a baseline for comparison. He concludes that model-based control combined with model learning seems to be much more data-efficient than the tested RL methods both in theory and practice. In other words, Machine Learning (ML) techniques seem best suited for model fitting and accounting for model uncertainty rather than for direct control.

Hence, one arrives at a research area that lies in the *intersection between control, learning and optimization*. When compared to RL, it has many more sound experimental results in safety-critical systems. Control and optimization intersect naturally in MPC. There are two major branches in this research area: *Learning for MPC* and *MPC for Learning with Constraints*. The latter uses MPC as a safety-filter for any learning-based controller, in particular RL. Specifically, the input signal computed by the learning-based controller is fed to the MPC which, in turn, verifies its safety. If the MPC deems the control action safe and feasible, the control is applied to the system. Otherwise, the MPC computes a safe input. Wabersich and Zeilinger [46] propose using an MPC as a safety certification of a learning-based input which is modified to a feasible trajectory towards a safe set if necessary. The former branch focuses on improving the formulation of the MPC, resorting to ML methods. In [47], the authors provide a summary of this type of controllers.

This thesis' pivot will be Learning-based Model Predictive Control, henceforth referred to as LMPC, which in this case does not include the use of MPC as safety filters. Most research has focused on using ML as a data-based adaptation of the prediction model or uncertainty description - *Model Learning*. This is of evident substance as the MPC relies on suitable accurate model representations of the system dynamics. Notwithstanding, learning has also targeted an MPC controller's parameterisation, *e.g.* the cost function, horizon length, or terminal components. This thesis's work builds on the LMPC architecture proposed by Professor Borrelli's MPC Lab - more details in Section 2.2.

Several other LMPC architectures have been proposed. Zeilinger's research group introduced a cautious LMPC that combines a nominal model with Gaussian Process Regression (GPR) techniques to model the unknown dynamics [48]. It has been shown to increase safety and performance and has been applied to trajectory tracking with a robotic arm and autonomous racing, respectively [49] and [50]. Schoellig's Dynamic Systems Lab proposed using Bayesian Linear Regression (BLR) to model the unknown dynamics [51]. These researchers argue that this simple model is more accurate in estimating the mean behaviour and model uncertainty than GPR and generalizes to novel operating conditions with little or no tuning. Further, this group designed a framework that combines BLR model learning with cost learning [52]. This method is inspired by ideas from reward shaping - encouraging and discouraging behaviours that lead to high and low reward, respectively - and models the prediction cost error.

Model Predictive Path Integral Control provides a mathematical methodology for developing optimal control algorithms based on the stochastic sampling of trajectories [53]. It has been applied to aggressive

autonomous racing on scale vehicles. In [54], RL updates the parameterization of a robust MPC scheme and the safety constraint to reduce conservatism while preserving safety.

Regarding what other groups with similar autonomous racing applications have investigated, namely in Roborace: the team from the University of Pisa published a paper demonstrating the software pipeline, namely their Planning and Control framework as well as the SLAM methods [55]. The Roborace organization was responsible for implementing low-level controllers that ensured closed-loop control of the actuators. The authors adopted three models: a kinematic model for path optimization, a dynamic mass model for speed profile optimization, and the dynamic single-track model for real-time Nonlinear Model Predictive Control (NMPC). This paper builds on the work of [56].

The opposing team from the Technical University of Munich also demonstrated their overall software stack to tackle the Roborace competition [57] and more relevant to this work, the architecture for the Path Planning module, [58]. Similarly to U Pisa's team, it includes a global race trajectory computed offline but uses Optimal Control techniques to minimize the lap time. Following is a local trajectory planning module to handle obstacles that generates an action set of multiple drivable trajectories through a graph-based planner [59]. Vehicle control is split into trajectory-tracking and low-level control. The former utilizes a gain-scheduled PD-controller accompanied by a feedforward term. In comparison, the latter is a P-controller combined with disturbance estimation and also a feedforward term.

This team from Munich has recently addressed known control algorithms' limitations using learning-based controllers. In [60], a model-free learning method performs an online adaptation of the maximum longitudinal and lateral accelerations used in the trajectory planning so that the vehicle does not violate the safety constraints. This is done through a safe Bayesian optimization that aims to maximize the accelerations scale factor and where GPR models the cost and constraint functions. The constraints are two heuristic measures for vehicle stability from vehicle dynamics science: understeer and wheel slip; and the lateral tracking error. The authors claim that this architecture is at times conservative, in part justified by the focus on safety, and is unable to adjust to different conditions in particular parts of the track. In [61], the authors aim to tackle performance limitations associated with conservative uncertainty assumptions in robust controllers. They propose a combination of a normalized Least Mean Square (LMS) filter with a recursive quantile estimator to identify feature-dependent upper and lower uncertainty bounds. Unlike BLR or GPR, there are no implicit assumptions about the samples' probability distribution, and it is computationally more efficient. This could be integrated with a Tube-MPC [62].

Regarding the FSD context, several teams have published papers about their approach to this competition [63–65]. In [63] and [66], an MPC is used for the whole motion control problem. While in [65], an MPC is used for lateral control only, using a feedforward PI-controller for the longitudinal dynamics. In [67], AMZ Driverless - the team from ETH Zurich - provides an in-depth description of their 2018 autonomous race car's algorithms and system architecture. This team exploited a hierarchical controller with a two-level optimization framework for motion planning in their 2019 competition vehicle [68]. It includes an offline lap time optimization trajectory and an online NMPC. The authors propose to use a terminal constraint on the longitudinal velocity computed by the offline module in the NMPC. Thus, efficiently coupling the two levels while reducing the NMPC prediction horizon and ensuring safety and

performance are guaranteed. In [69], they extended this architecture by introducing a high-frequency low-level controller that tracks the states predicted by the NMPC which enables reducing model mismatch and optimizing the low-level torque vectoring command in the higher level motion planners. The authors claim this architecture is able to outperform a professional driver.

AMZ used a learning-based controller to tackle the issue relevant to autonomous racing that accurate vehicle models that cover the entire performance envelope are highly nonlinear and difficult to identify [70]. Therefore, the proposed formulation considers a simple nominal vehicle model where GPR models residual model uncertainty. The approach is based on Model Predictive Contouring Control (MPCC) [71] and cautions MPC [50]. This framework was tested on an FSD prototype, achieving lap-time improvements of 10%.

Relevant works from Instituto Superior Técnico include the development of an MPC controller for the FSD competition whose model is a Neural Network trained from a detailed vehicle simulator [72]. The author also trains another Neural Network from the MPC controller control actions. A torque-vectoring [73] and sideslip estimation [74] algorithms for a Formula Student prototype.

1.3 Objectives and Contributions

The chief objective of this thesis is to improve the motion planning and control pipeline of FST Lisboa. Hence, this algorithm is responsible for both trajectory planning and following. Further, there is no offline pre-computed trajectory. We target the prime FSD event, *i.e.* most points available in a single dynamic discipline, of Trackdrive. To that end, we believe a sensible manner of tackling such an event ought to use existing data, both pre-recorded and collected online, to improve a model-based controller's knowledge. The LMPC architecture on top of which we build this work is a natural candidate. This architecture should enable iterative performance improvements. That is, decreasing lap time is expected at each completed lap.

The contribution of this thesis is fourfold:

- Execute the necessary adaptations of this architecture to the FST Lisboa software pipeline.
- Improve the computational performance of the algorithm (double the controller sampling frequency and more than double the prediction horizon length).
- Validate the LMPC architecture in an FSD simulation environment context.
- Implement model learning schemes to reduce model mismatch.

1.4 Thesis Outline

This thesis is organized as follows.

Chapter 2 provides a theoretical background on the methods used in the LMPC architecture. In particular, it starts with a brief introduction to Model Predictive Control. Then, thoroughly describes the

LMPC algorithm architecture and introduces the underlying theoretical guarantees. Finally, the Gaussian Process Regression Machine Learning technique used for model learning is introduced.

In Chapter 3, we describe the final algorithm developed. In particular, we introduce the LMPC formulation that enables approaching autonomous racing as a minimum lap time problem. We also detail how to build the terminal components in the autonomous racing case. Further, we explain the optimization problem that characterizes the MPC and the vehicle models used therein. Finally, we characterize the GPR relevant components such as the covariance function and the feature variables.

In Chapter 4, a detailed description is given regarding the simulator's underlying dynamics and relevant parameters. Subsequently, one provides an overview of the platform's hardware and software setup used for testing, the FST Lisboa autonomous racing prototype. Lastly, the details of the controller implementation are given. Particularly, concerning new features implemented and language-relevant decisions.

The results and some immediate conclusions are drawn in Chapter 5. Starting with the model learning analysis of the sparse Gaussian Processes Regression approximations' structure and the respective results. After, the overall LMPC architecture results on two different Formula Student tracks are shown.

Ultimately, Chapter 6 provides a summary of the main conclusions and contributions of this research work. Furthermore, one discusses possible future work directions, both regarding algorithm architecture and machine learning techniques.

Chapter 2

Theoretical Background

In this chapter, we will lay the theoretical foundations for which the algorithms in Chapter 3 are based. First, with an introduction to Model Predictive Control and the concept of Receding Horizon Control (RHC). Then, extend the general MPC framework to the particular case of Learning-based MPC this thesis addresses. Finally, we will introduce the algorithms used in the scope of Model Learning.

We use bold lowercase letters for vectors $\mathbf{x} \in \mathbb{R}^n$ and bold capitalized letters for matrices $\mathbf{X} \in \mathbb{R}^{n \times m}$, while scalars are non-bold.

2.1 Model Predictive Control

The idea of Receding Horizon Control is that an infinite horizon sub-optimal controller can be designed by repeatedly solving Finite-Time Constrained Optimal Control (FTCOC) problems in a receding horizon fashion [32]. At each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon (top diagram of Figure 2.1). The computed optimal manipulated input signal is applied to the process only during the following sampling interval $[t,\ t+1]$. At the next time-step $t+1$ a new optimal control problem based on new measurements of the state is solved over a shifted horizon (bottom diagram of Figure 2.1).

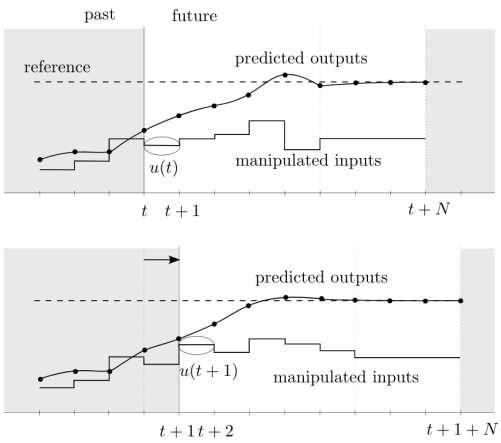


Figure 2.1: Receding Horizon Idea. [reprinted from [32]]

Model Predictive Control is an RHC where the FTCOC problem with a prediction horizon of N is computed by solving the following optimization problem online:

$$J_{t\to t+N}(x_{t|t}) = \min_{u_{t|t},\cdots,u_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} q(x_{k|t}, u_{k|t}) + p(x_{t+N|t}) \right] \tag{2.1a}$$

$s.t.$

$$x_{k+1|t} = Ax_{k|t} + Bu_{k|t} \qquad \forall k \in [t, \cdots, t+N-1] \tag{2.1b}$$

$$x_{k|t} \in \mathcal{X}, \quad u_{k|t} \in \mathcal{U} \qquad \forall k \in [t, \cdots, t+N-1] \tag{2.1c}$$

$$x_{t+N|t} \in \mathcal{X}_f \tag{2.1d}$$

$$x_{t|t} = x(t) \tag{2.1e}$$

where x is the state and u the control input. The subscript $k|t$ represents a given quantity in the prediction horizon with respect to time t. $x_{t|t}$ and $x_{t+N|t}$ (whose short notation is x_N) represent the initial and terminal state of the system starting at time t, respectively. This becomes evident by Equation (2.1e) which imposes the current system state to be the initial condition of the generic FTCOC problem. Equation (2.1b) represents the discrete-time linear time-invariant system dynamics. State and input constraints are given by Equation (2.1c). The terminal constraint is given by Equation (2.1d) which forces the terminal state into some set $\mathcal{X}_f$. The stage $q(\cdot,\cdot)$ and terminal cost $p(x_N)$ are any arbitrary continuous, strictly positive functions.

In RHC, the optimization problem is solved over a finite horizon repeatedly at each time step, in the hope that the controller resulting from this *short-sighted* strategy will lead to a closed-loop behavior that mimics that of the infinite horizon controller. In general, however, stability and feasibility are not ensured by the RHC law. Feasibility corresponds to the existence of a sequence of control inputs for which the constraints are obeyed. When the trajectories converge to the origin, one may say the closed-loop system is asymptotically stable. This definition concerns the problem of regulating to the origin but it can be shown equivalently to the problem of following some reference.

In principle, one could analyze the RHC law for feasibility, stability and convergence but this is difficult. One of the most popular approaches to guarantee persistent feasibility, *i.e.* feasibility at all future times, and stability of the RHC law makes use of a control invariant terminal set $\mathcal{X}_f$ and a terminal cost $p(x_N) = ||Px_N||_p$, where $p = 1$ or $p = \infty$, which drives the closed-loop optimal trajectories towards the origin [32]. Note that the terminal set $\mathcal{X}_f$ is introduced artificially for the sole purpose of leading to a sufficient condition for persistent feasibility. If the following conditions hold:

1. The stage cost $q(x,u)$ and terminal cost $p(x)$ are continuous and positive definite functions.
2. The sets $\mathcal{X}$, $\mathcal{X}_f$ and $\mathcal{U}$ contain the origin in their interior and are closed.
3. $\mathcal{X}_f$ is a control invariant, *i.e.* $\mathcal{X}_f \subseteq \mathcal{X}$.
4. $\min_{v\in\mathcal{U},\ Ax+Bv\in X_f} \big(-p(x) + q(x,v) + p(Ax+Bv) \big) \leq 0, \forall x \in \mathcal{X}_f$.

then, the closed-loop system is asymptotically stable with domain of attraction $x(0) \in \mathcal{X}_0$ [75].

2.2 Learning-based Model Predictive Control

Rosolia and Borrelli [76] first proposed the LMPC architecture which this work builds upon. It resembles the Iterative Learning Control strategy, which has been used with MPC to minimize the tracking error in an iterative manner [77]. However, this is a reference-free iterative control strategy able to learn from previous iterations. At each iteration, the initial condition, the constraints, and the objective function do not change. The authors show how to design a terminal safe set - $\mathcal{SS}$ - and a terminal cost function - Q-function - such that the following theoretical guarantees hold:

- **Nonincreasing cost** at each iteration.
- **Recursive feasibility**, *i.e.* state and input constraints are satisfied at iteration j if they were satisfied at iteration $j-1$.
- Closed-loop equilibrium is **asymptotically stable**.

This framework's main contribution is to learn terminal constraints rather than model learning. The LMPC solves at each sampling time t of iteration j the FTCOC problem of the form:

$$J^j_{t\to t+N}(x^j_t) = \min_{u_{t|t},\cdots,u_{t+N-1|t}} \left[\sum_{k=t}^{t+N-1} q(x_{k|t}, u_{k|t}) + Q^{j-1}(x_{t+N|t}) \right] \tag{2.2a}$$

$s.t.$

$$x_{k+1|t} = h_t(x_{k|t}, u_{k|t}) \qquad \forall k \in [t, \cdots, t+N-1] \tag{2.2b}$$

$$x_{k|t} \in \mathcal{X}, \quad u_{k|t} \in \mathcal{U} \qquad \forall k \in [t, \cdots, t+N-1] \tag{2.2c}$$

$$x_{t+N|t} \in \mathcal{SS}^{j-1} \tag{2.2d}$$

$$x_{t|t} = x^j_t \tag{2.2e}$$

Note that the major differences to the generic MPC in Equation (2.1) concern the terminal components: the terminal cost is given by the Q-function: $p(x_{t+N|t}) = Q^{j-1}(x_{t+N|t})$; whereas the terminal constraint corresponds to the terminal safe set $\mathcal{SS}$: $\mathcal{X}_f = \mathcal{SS}^{j-1}$. The system dynamics in Equation (2.2b) have been extended to the nonlinear case, *i.e.* some continuous function h maps the transition from the current state x_t given a control input u_t to the subsequent state x_{t+1}. In this chapter, scalar quantities are used but it is equivalent for vector quantities.

$$\mathbf{x}^{*,j}_{t:t+N|t} = \left[x^{*,j}_{t|t}, \cdots, x^{*,j}_{t+N|t} \right] \tag{2.3a}$$

$$\mathbf{u}^{*,j}_{t:t+N|t} = \left[u^{*,j}_{t|t}, \cdots, u^{*,j}_{t+N-1|t} \right] \tag{2.3b}$$

The Equations (2.3a) and (2.3b) are the optimal state and control solution at time t of iteration j, respectively. At this instance, the control input applied to the system is the first element of $\mathbf{u}^{*,j}_{t:t+N|t}$:

$$u^j_t = u^{*,j}_{t|t} \tag{2.4}$$

The FTCOC problem, Equation (2.2), is solved at the following sampling time $t+1$, based on the new measurements of state x_{t+1}^j, yielding a receding horizon control strategy. At the jth iteration, the inputs applied to system and the corresponding state evolution are collected in the vectors given by Equation (2.5a) and Equation (2.5b), respectively.

$$\mathbf{u}^j = \left[u_0^j, u_1^j, \cdots, u_t^j, \cdots\right] \tag{2.5a}$$

$$\mathbf{x}^j = \left[x_0^j, x_1^j, \cdots, x_t^j, \cdots\right] \tag{2.5b}$$

The safe set $\mathcal{SS}^j$, given by Equation (2.6) is the collection of all state trajectories at iteration i for $i \in M^j$. M^j in Equation (2.7) is the set of indexes k corresponding to the iterations that successfully steered the system to the final point x_F.

$$\mathcal{SS}^j = \left\{ \bigcup_{i \in M^j} \bigcup_{t=0}^{\infty} x_t^i \right\} \tag{2.6}$$

$$M^j = \left\{ k \in [0, j] : \lim_{t \to \infty} x_t^k = x_F \right\} \tag{2.7}$$

Note that the safe set can be interpreted as a sampled subset of the maximal stabilizable set $\mathcal{K}(x_F)$, since for every point in the set, there exists a feasible control sequence that satisfies the state constraints and steers the system toward x_F in $K \in \mathbb{N}$ steps. This is a special case of the K-step controllable set where the target is a control invariant set, which is the case with x_F since it is assumed that this final state is a feasible equilibrium for the unforced system, *i.e.* $h(x_F, 0) = x_F$.

The Q^j function, defined in Equation (2.8), assigns to every point in the sampled safe set the minimum cost-to-go along the trajectories therein.

$$\forall x \in \mathcal{SS}^j, Q^j(x) = J^{i^*}_{t^* \to \infty}(x) = \sum_{k=t^*}^{\infty} q(x_k^{i^*}, u_k^{i^*}) \tag{2.8}$$

where i^* corresponds to the iteration that minimizes such cost starting at that particular state x and t^* is the respective time of that state in that iteration.

[76] provides the detailed proof of the theoretical guarantees stated and the conditions for which these hold.

Let one now reflect on the merit of the safe set $\mathcal{SS}$ and Q-function. The safe set works as a safe region for the shorter horizon N. For example, in autonomous racing, it can account for the shape of the track beyond the horizon. This way, the controller is informed from past experimental data of how fast he can push in a particular part of the track without having to compute the global optimal racing line. Similarly, the Q-function enlightens the controller with respect to which states in the safe region yield the minimum cost in the long-term, just like in Reinforcement Learning (although, there one would try to maximize the Q-function). In autonomous racing, that would correspond to faster laps.

Checking if a state is in $\mathcal{SS}$ is a simple search. However, the optimization problem becomes challenging to solve even in the linear case due to its integer nature. Rosolia and Borrelli [78] build upon the

former for linear systems by proposing convexfying the terminal constraints to significantly reduce the computational burden without compromising the LMPC scheme's guarantees. As $\mathcal{X}$ and $\mathcal{U}$ are convex, for each combination of the elements in $\mathcal{SS}^j$ there is a control sequence that steers the system to x_F. Thus, $\mathcal{CS}^j$, defined as Equation (2.9), is a control invariant set.

$$\mathcal{CS}^j = \text{Conv}(\mathcal{SS}^j) = \left\{ \sum_{i=1}^{|\mathcal{SS}^j|} \alpha_i z_i : \alpha_i \geq 0, \sum_{i=1}^{|\mathcal{SS}^j|} \alpha_i = 1, z_i \in \mathcal{SS}^j \right\} \tag{2.9}$$

where $|\mathcal{SS}^j|$ is the cardinality of $\mathcal{SS}^j$. For the terminal cost, the barycentric function P^j is used to assign to every point in $\mathcal{CS}^j$ the minimum cost-to-go along the trajectories in $\mathcal{CS}^j$:

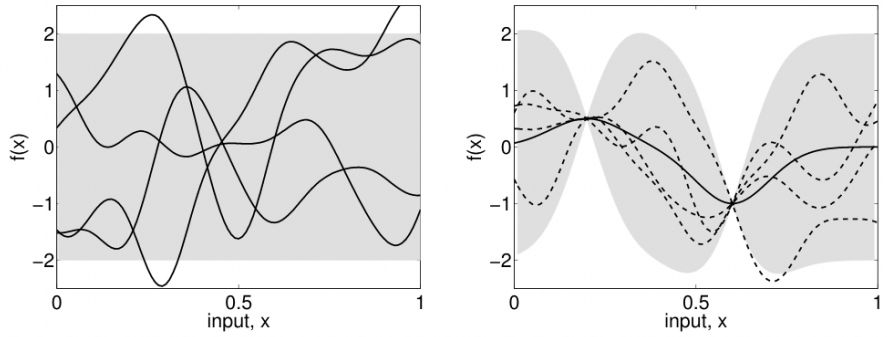
$$\begin{aligned} \forall x \in \mathcal{CS}^j, P^j(x) = & \min_{\lambda_t \geq 0, \forall t \in [0,\infty]} \sum_{k=0}^{j} \sum_{t=0}^{\infty} \lambda_t^k J_{t\to\infty}^k(x_t^k) \\ s.t. & \\ & \sum_{k=0}^{j} \sum_{t=0}^{\infty} \lambda_t^k = 1 \\ & \sum_{k=0}^{j} \sum_{t=0}^{\infty} \lambda_t^k x_t^k = x \end{aligned} \tag{2.10}$$

The FTCOC problem then becomes a convex optimization problem as the terminal constraint enforces the terminal state in the convex set $\mathcal{CS}^j$ and the terminal cost P^j is a convex function.

In [79], the authors present the LMPC architecture for unconstrained uncertain linear systems subject to bounded additive uncertainty. The goal is to solve an infinite time robust optimal control problem. Akin to the deterministic case, some guarantees hold: *i)* state and input constraints are robustly satisfied, *ii)* the closed-loop system converges asymptotically to a neighbourhood of the origin, *iii)* the worst-case iteration cost is non-increasing, and *iv)* the domain of the policy is not shrinking (exploration). The initial proposed control strategy is computationally intensive. Therefore, the authors propose a sample-based approach to approximate the safe set and the value function using historical noisy data.

2.3 Gaussian Processes Regression

Gaussian Processes is a Machine Learning approach that can be interpreted as a Bayesian version of the Support Vector Machine (SVM) method. It is a non-parametric, probabilistic procedure to learning in *kernel* machines. By focusing on Gaussian processes, the problem becomes computationally tractable. Furthermore, it provides a fully probabilistic predictive distributions, including estimates of the uncertainty of the predictions. See Figure 2.2 for an illustration of how GPR inference works. The graph on the left shows four samples drawn from the prior distribution. While the graph on the right shows the situation after two datapoints have been observed. The mean prediction is shown as the solid line and four samples from the posterior are shown as dashed lines. In both plots the shaded region denotes twice the standard deviation at each input value. One can see that the uncertainty is null where datapoints have been observed and increases as a function of the distance from those observations.



(a), prior (b), posterior

Figure 2.2: Gaussian Process Inference. [reprinted from [80]]

The combination of the prior and the data leads to the posterior distribution over functions. The specification of the prior is important because it fixes the properties of the covariance functions considered for inference. In particular, the type of covariance function used and its hyperparameters. In turn, the covariance function expresses some prior notion of smoothness of the underlying function. For a detailed description of GPR, the author refers the reader to the book Gaussian Processes for Machine Learning by Rasmussen and Williams [80].

Consider now an unknown latent function $\mathbf{g} : \mathbb{R}^{n_z} \to \mathbb{R}^{n_g}$ that is identified from a collection of inputs $\mathbf{z}_k \in \mathbb{R}^{n_z}$ and corresponding outputs $\mathbf{y}_k \in \mathbb{R}^{n_g}$.

$$\mathbf{y}_k = \mathbf{g}(\mathbf{z}_k) + \mathbf{w}_k \tag{2.11}$$

where $\mathbf{w}_k \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma}^{\boldsymbol{w}})$ is independent and identically distributed Gaussian noise with diagonal variance $\boldsymbol{\Sigma}^{\boldsymbol{w}} = \text{diag}\big([\sigma_1^2, \cdots, \sigma_{n_g}^2]\big)$. The set of n input and output data pairs form a dictionary $\mathcal{D}$:

$$\mathcal{D} = \Big\{ \mathbf{Y} = [\mathbf{y}_1^T; \cdots; \mathbf{y}_n^T] \in \mathbb{R}^{n \times n_g}, \mathbf{Z} = [\mathbf{z}_1^T; \cdots; \mathbf{z}_n^T] \in \mathbb{R}^{n \times n_z} \Big\} \tag{2.12}$$

Assuming a Gaussian prior on $\mathbf{g}$ in each output dimension $d \in \{1, \cdots, n_g\}$, such that they can be treated independently, the posterior distribution in dimension d at an evaluation point $\mathbf{z}$ has mean and variance given by Equations (2.13a) and (2.13b), respectively. Further, in this situation, one refers to $\mathbf{Y}$ as $\mathbf{y}$. That is, there is a collection of n_d n-dimensional vectors $\mathbf{y}^d$. To get the posterior distribution over functions, the joint prior distribution must be restricted to contain only those functions which agree with the observed data points. Graphically one may think of generating functions from the prior, and rejecting the ones that disagree with the observations, although this strategy would not be computationally efficient. However, in probabilistic terms, this operation is extremely simple, corresponding to conditioning the joint Gaussian prior distribution on the observations.

$$\mu^d(\mathbf{z}) = \mathbf{k}_{\mathbf{zZ}}^d \big(\mathbf{K}_{\mathbf{ZZ}}^d + \mathbf{I}\sigma_d^2\big)^{-1} \mathbf{y}^d \tag{2.13a}$$

$$\Sigma^d(\mathbf{z}) = k_{\mathbf{zz}}^d - \mathbf{k}_{\mathbf{zZ}}^d \big(\mathbf{K}_{\mathbf{ZZ}}^d + \mathbf{I}\sigma_d^2\big)^{-1} \mathbf{k}_{\mathbf{Zz}}^d \tag{2.13b}$$

where $\mathbf{K}^d_{\mathbf{ZZ}}$ is the Gram matrix, *i.e.* $[\mathbf{K}^d_{\mathbf{ZZ}}]_{ij} = k^d(\mathbf{z}_i, \mathbf{z}_j)$, $[\mathbf{k}^d_{\mathbf{Zz}}]_j = k^d(\mathbf{z}_j, \mathbf{z})$, $\mathbf{k}^d_{\mathbf{Zz}} = (\mathbf{k}^d_{\mathbf{Zz}})^{\mathrm{T}}$ and $k^d_{\mathbf{zz}} = k^d(\mathbf{z}, \mathbf{z})$ corresponds to the kernel function used. Note that the variance does not depend on the observed targets, but only on the inputs; this is a property of the Gaussian distribution.

The squared exponential (SE) kernel function in Equation (3.23) is often used for its smoothness properties of the Gaussian distribution. It is sometimes referred to as Radial Basis Function. Furthermore, it provides a good measure of the correlation between different inputs.

$$k^d_{SE}(\mathbf{z}, \bar{\mathbf{z}}) = \sigma^2_{f,d} \exp\left(-\frac{1}{2}\frac{(\mathbf{z}-\bar{\mathbf{z}})^{\mathrm{T}}(\mathbf{z}-\bar{\mathbf{z}})}{l_d^2} \right) + \sigma^2_{n,d}\delta_{\mathbf{z}\bar{\mathbf{z}}} \tag{2.14}$$

where $l_d \in \mathbb{R}$ is the positive length-scale, $\sigma^2_{f,d}$ the squared signal variance, $\sigma^2_{n,d}$ is the squared noise variance (or noise level parameter) and $\delta_{\mathbf{z}\bar{\mathbf{z}}}$ is the Kronecker delta, *i.e.* $\delta_{\mathbf{z}\bar{\mathbf{z}}} = 1$ if and only if $\mathbf{z} = \bar{\mathbf{z}}$. In this case, $\phi_d = \{l_d, \sigma^2_{f,d}, \sigma^2_{n,d}\}$ is a vector that contains all hyperparameters. Decreasing the length-scale provides more flexibility, *i.e.* sharp variations in the value of the underlying function g are allowed to fit the data points better. However, the variance grows rapidly away from the data points. On the other hand, increasing the length-scale yields a slowly varying function with significant noise. See Figure 2.3 for a graphical depiction of this behaviour.

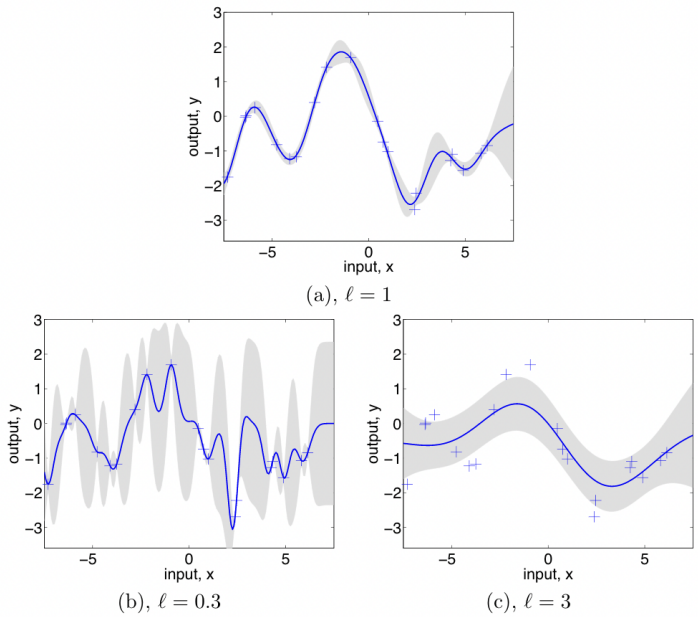


Figure 2.3: Length-scale influence on GPs. (a) Data is generated from a GP with hyperparameters (l, σ_f, σ_n) =(1,1,0.1), as shown by the + symbols. Using Gaussian process prediction with these hyperparameters yields a 95% confidence region for the underlying function f (shown in grey). Panels (b) and (c) again show the 95% confidence region, but this time for hyperparameter values (0.3, 1.08, 0.00005) and (3.0, 1.16, 0.89) respectively. [reprinted from [80]]

The resulting multivariate Gaussian Process approximation is given by:

$$\mathbf{g}(\mathbf{z}) \sim \mathcal{N}\big(\boldsymbol{\mu}^g(\mathbf{z}), \boldsymbol{\Sigma}^g(\mathbf{z})\big) \tag{2.15}$$

where $\boldsymbol{\mu}^g(\mathbf{z}) = [\mu^1(\mathbf{z}); \cdots; \mu^{n_g}(\mathbf{z})] \in \mathbb{R}^{n_g}$ and $\boldsymbol{\Sigma}^g(\mathbf{z}) = [\Sigma^1(\mathbf{z}); \cdots; \Sigma^{n_g}(\mathbf{z})] \in \mathbb{R}^{n_g \times n_g}$.

Sparse Approximation Methods for Gaussian Process Regression

The computational complexity of GPR strongly depends on the number of data points n. In particular, a computational cost of $\mathcal{O}(n^3)$ is incurred whenever a new training point is added to the dictionary $\mathcal{D}$. This is due to the need to invert $\left(\mathbf{K}_{\mathbf{ZZ}}^{d} + \mathbf{I}\sigma_d^2\right)$ which is a $n \times n$ matrix. Besides, the evaluation of the mean and variance have complexity cost of $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$, respectively.

A host of sparse approximation techniques have been proposed to allow the application of GPs to large problems in Machine Learning [81]. An additional set of $m < n$ latent variables $\bar{\mathbf{g}} = [\bar{g}_1, \cdots, \bar{g}_m]$, which are called inducing variables or support points, are used to approximate Equation (2.13). These are values of the Gaussian Process evaluated at the inducing inputs $\mathbf{Z}_{ind} = [\bar{\mathbf{z}}_0^T; \cdots; \bar{\mathbf{z}}_m^T]$. The latent variables are represented as $\bar{\mathbf{g}}$ rather than $\bar{\mathbf{y}}$ as they are not real observations. Thus, it does not make sense to include a noise variance.

Most approximations consider that $\mathbf{g}$ and $\mathbf{g}(\mathbf{z})$ are conditionally independent given $\bar{\mathbf{g}}$. This is to say that $\mathbf{g}$ and $\mathbf{g}(\mathbf{z})$ can only communicate through $\bar{\mathbf{g}}$. The joint prior is thus given as follows:

$$p(\mathbf{g}(\mathbf{z}), \mathbf{g}) \simeq q(\mathbf{g}(\mathbf{z}), \mathbf{g}) = \int q(\mathbf{g}(\mathbf{z})|\bar{\mathbf{g}})\; q(\mathbf{g}|\bar{\mathbf{g}})\; p(\bar{\mathbf{g}})\; d\bar{\mathbf{g}} \tag{2.16}$$

The different approximations proposed correspond to different additional assumptions on the two approximate inducing conditionals $q(\mathbf{g}(\mathbf{z})|\bar{\mathbf{g}})$ and $q(\mathbf{g}|\bar{\mathbf{g}})$, for which the exact solutions are given below:

$$\text{training conditional:} \quad p(\mathbf{g}|\bar{\mathbf{g}}) = \mathcal{N}(\mathbf{K}_{\mathbf{g},\bar{\mathbf{g}}}\, \mathbf{K}_{\bar{\mathbf{g}},\bar{\mathbf{g}}}^{-1}\, \bar{\mathbf{g}}, \mathbf{K}_{\mathbf{g},\mathbf{g}} - \mathbf{Q}_{\mathbf{g},\mathbf{g}}) \tag{2.17a}$$

$$\text{test conditional:} \quad p(\mathbf{g}(\mathbf{z})|\bar{\mathbf{g}}) = \mathcal{N}(\mathbf{K}_{\mathbf{z},\bar{\mathbf{g}}}\, \mathbf{K}_{\bar{\mathbf{g}},\bar{\mathbf{g}}}^{-1}\, \bar{\mathbf{g}}, \mathbf{K}_{\mathbf{z},\mathbf{z}} - \mathbf{Q}_{\mathbf{z},\mathbf{z}}) \tag{2.17b}$$

where we have introduced the shorthand notation $\mathbf{Q}_{\mathbf{a},\mathbf{b}} = \mathbf{K}_{\mathbf{a},\bar{\mathbf{g}}}\mathbf{K}_{\bar{\mathbf{g}},\bar{\mathbf{g}}}^{-1}\mathbf{K}_{\bar{\mathbf{g}},\mathbf{b}}$.

The simplest sparse approximation method (and does not fit inside the general scheme of sparse methods just introduced) is the Subset of Data (SoD) approximation, *i.e.* solves Equation (2.13) by substituting $\mathbf{Z}$ by $\mathbf{Z}_{ind}$. It is often used as a baseline for sparse approximations. The computational complexity is reduced to $\mathcal{O}(m^3)$ for training; and $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ for the mean and variance, respectively. Nevertheless, it is not expected to be a competitive method as it wasteful of data. Even with redundant data and a good selection of the active set, it seems unlikely to get a realistic estimation of the uncertainties. In order to improve the chances of good performance, rather than selecting the m points randomly, researchers have designed methods to select which points are included in the active set. Informative Vector Machine [82] chooses the next point for inclusion the one that maximizes the *differential entropy score*, *i.e.* the site with the largest variance.

If $\mathcal{D}$ is not updated online, that is with new datapoints collected as they are generated, every quantity except those that depend on the evaluated test case $\mathbf{z}$ can be precomputed. Specifically, only $\mathbf{k}_{\mathbf{z}\mathbf{Z}_{\mathbf{ind}}}^{d}$ needs to be computed at each sampling time since it depends on new regression feature states $\mathbf{z}$.

A reasonable approximation to the training conditional is to preserve a block-diagonal of the true covariance matrix and set the remaining entries to zero. Partially Independent Training Conditional (PITC) divides the dataset $\mathcal{D}$ in k groups, such that the conditional independence is only active for part

of the training function values - those with null covariance entries. This method is *transductive* rather than inductive, in the sense that it computes a test set dependent model making use of the test set input locations. The computational complexity is $\mathcal{O}(km^3)$. Frequently, one takes $k = n/m$ resulting in computational cost for training $\mathcal{O}(nm^2)$.

$$q_{\text{PITC}}(\mathbf{g}|\bar{\mathbf{g}}) = \mathcal{N}\big(\mathbf{K}_{\mathbf{g},\bar{\mathbf{g}}}\, \mathbf{K}_{\bar{\mathbf{g}},\bar{\mathbf{g}}}^{-1}\, \bar{\mathbf{g}},\ \text{blockdiag}[\mathbf{K}_{\mathbf{g},\mathbf{g}} - \mathbf{Q}_{\mathbf{g},\mathbf{g}}]\big) \tag{2.18a}$$

$$q_{\text{PITC}}(\mathbf{g}(\mathbf{z})|\ \bar{\mathbf{g}}) = p(\mathbf{g}(\mathbf{z})|\ \bar{\mathbf{g}}) \tag{2.18b}$$

The PITC approximation does not correspond exactly to a Gaussian process since the covariance is computed differently for training and test cases. To obtain a GP from the PITC, one would need to extend the partial conditional independence assumption to the joint conditional $p(\mathbf{g}, \mathbf{g}(\mathbf{z}|\bar{\mathbf{g}})$, which would require abandoning the primal assumption that the training and the test function values are conditionally independent, given the inducing variables. The Fully Independent Training Conditional (FITC) [83] is an extreme case of PITC, with $k = n$ unitary groups (blocks). The computational complexity is $\mathcal{O}(nm^2)$ initially, and $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ per test case for the predictive mean and variance, respectively.

$$q_{\text{FITC}}(\mathbf{g}|\bar{\mathbf{g}}) = \mathcal{N}\big(\mathbf{K}_{\mathbf{g},\bar{\mathbf{g}}}\, \mathbf{K}_{\bar{\mathbf{g}},\bar{\mathbf{g}}}^{-1}\, \bar{\mathbf{g}},\ \text{diag}[\mathbf{K}_{\mathbf{g},\mathbf{g}} - \mathbf{Q}_{\mathbf{g},\mathbf{g}}]\big) \tag{2.19a}$$

$$q_{\text{FITC}}(\mathbf{g}(\mathbf{z})|\ \bar{\mathbf{g}}) = p(\mathbf{g}(\mathbf{z})|\ \bar{\mathbf{g}}) \tag{2.19b}$$

FITC can be viewed as a standard GP with a particular non-stationary covariance function parameterized by the pseudo-inputs. In their paper, the authors tackle a common setback in sparse approximations in that they lack a reliable way of learning kernel hyperparameters, because the active set selection interferes with this learning procedure. They propose to use a covariance function parameterized by the locations of inducing inputs — an active set not constrained to be a subset of the data, found by a continuous optimization. Furthermore, rather than simply maximizing the marginal likelihood with respect to $\bar{\mathbf{Z}}_{ind}$ and $\bar{\mathbf{g}}$, one can integrate out the inducing variables $\bar{\mathbf{g}}$. They place a Gaussian prior on the pseudo targets. This is sensible because one expects the inducing data to be distributed in a very similar manner to the real data, if they are to model them well. The marginal likelihood can then be maximized with respect to the hyperparameters and the pseudo-input locations by gradient ascent.

Quiñonero et al. [84] underline there are two options for FITC joint predictions: *i)* use the full test conditional in Equation (2.17b) or *ii)* extend the additional factorization assumption to the test conditional. The authors posit as likely that the original authors Snelson and Ghahramani [83] intended the second, despite the fact that joint predictions are not targeted explicitly in their paper. This is the case since under the first option the training and test covariances are calculated differently, which amounts to a degenerate GP. Nonetheless, this is strictly a theoretical hypothesis for the nature of the approximation because the additional independence assumption for the test cases is not necessary for computational procedures. Quiñonero et al. [84] have called this approximation Fully Independent Conditional (FIC).

The mean and variance are given by:

$$\mu^d(\mathbf{z}) = \mathbf{k}^d_{\mathbf{z}\mathbf{Z}_{ind}}\big(\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{Z}_{ind}} + \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}}\mathbf{\Lambda}^{-1}\mathbf{K}_{\mathbf{z}\mathbf{Z}_{ind}}\big)^{-1}\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}}\mathbf{\Lambda}^{-1}y^d = \mathbf{k}^d_{\mathbf{z}\mathbf{Z}_{ind}}i^d \tag{2.20a}$$

$$\Sigma^d(\mathbf{z}) = \mathbf{k}^d_{\mathbf{z}\mathbf{Z}_{ind}}\mathbf{K}^{-1}_{\mathbf{Z}_{ind}\mathbf{Z}_{ind}}\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}} - \mathbf{K}_{\mathbf{z}\mathbf{Z}_{ind}}\mathbf{\Theta}\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}} \tag{2.20b}$$

where $\mathbf{\Theta} = \Big(\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{Z}_{ind}} + \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}}\Lambda^{-1}\mathbf{K}_{\mathbf{z}\mathbf{Z}_{ind}}\Big)^{-1}$ and $\mathbf{\Lambda} = \text{diag}\big(\mathbf{K}_{\mathbf{z}\mathbf{z}} - \mathbf{K}_{\mathbf{z}\mathbf{Z}_{ind}}\mathbf{K}^{-1}_{\mathbf{Z}_{ind}\mathbf{Z}_{ind}}\mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}}\big)$. $\mathbf{i}^d$ is the information vector taken for each d sparse model.

Model Selection and Adaptation of Hyperparameters

Model selection is a problem of practical interest. Herein, model selection comprises both discrete choices, *e.g.* the type of covariance function used, and the setting of continuous hyperparameters of the covariance function. It is often hard to immediately define all aspects of the covariance function. While some properties may be evident to determine from a priori knowledge of the application, we typically have only vague information about the values other properties should take. Hence, model selection can help both to refine the predictions of the model, and give a valuable interpretation to the user about the properties of the data [80].

Bayesian principles provide a persuasive and consistent framework for inference. However, for most interesting models in machine learning, the required computations (integrals over parameter space - stated below) are analytically intractable, and good approximations are not easily derived. Inference takes place one level at a time by applying the rules of probability theory. At the bottom level, the *posterior* over the parameters; subsequently, the posterior over the hyperparameters; and, at the top level, the posterior over the model. All these posteriors are given by the Bayes' rule. For the sake of brevity and relevance on the particular case of GPs, one shows only the bottom-most level posterior which is given as follows:

$$p(\mathbf{w}|\mathbf{y}, \mathbf{Z}, \boldsymbol{\phi}, \mathcal{H}_i) = \frac{p(\mathbf{y}|\mathbf{Z}, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\phi}, \mathcal{H}_i)}{p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\phi}, \mathcal{H}_i)} \tag{2.21}$$

where $\mathbf{w}$ are the lowest level parameters, *e.g.* the weights in a neural network, and $\mathcal{H}_i$ are a discrete set of possible model structures under consideration. $p(\mathbf{y}|\mathbf{Z}, \mathbf{w}, \mathcal{H}_i)$ is the *likelihood* and $p(\mathbf{w}|\boldsymbol{\phi}, \mathcal{H}_i)$ is the parameter *prior*. The prior encodes as a probability distribution our knowledge about the parameters prior to seeing data. The posterior combines the information from the prior and the data through the likelihood. The normalizing constant in the denominator is the *marginal likelihood* or evidence - refers to the marginalization over the function values - and is given by:

$$p(\mathbf{y}|\mathbf{Z}, \boldsymbol{\phi}, \mathcal{H}_i) = \int p(\mathbf{y}|\mathbf{Z}, \mathbf{w}, \mathcal{H}_i)p(\mathbf{w}|\boldsymbol{\phi}, \mathcal{H}_i)\, d\mathbf{w} \tag{2.22}$$

GPR models with Gaussian noise are a rare exception in the ML field in that they enable the application of Bayesian inference. This is because the integrals over the parameters are analytically tractable and at the same time the models are very flexible. In this case, the former quantities are derived in the simplified form where hyperparameters are optimized over and the parameters $\mathbf{w}$ are not applicable.

Thus, the marginal likelihood in Equation (2.22) becomes in its logarithmic form:

$$\log p(\mathbf{y}, \mathbf{Z}, \boldsymbol{\phi}) = -\frac{1}{2}\mathbf{y}^{\mathrm{T}}\mathbf{K}_y\mathbf{y} - \frac{1}{2}\log|\mathbf{K}_y| - \frac{n}{2}\log 2\pi \tag{2.23}$$

where $\mathbf{K}_y = \left(\mathbf{K}^d_{\mathbf{ZZ}} + \mathbf{I}\sigma_d^2\right)^{-1}$ is the covariance matrix for the noisy targets. The three terms are readily interpretable: the only term involving the observed targets is the *data-fit* $\mathbf{y}^{\mathrm{T}}\mathbf{K}_y\mathbf{y}/2$; $\log|\mathbf{K}_y|/2$ is the complexity penalty depending only on the covariance function; and the inputs and $n\log(2\pi)/2$ is a normalization constant.

Cross-validation (CV) can be used for estimating the performance of a model selection procedure, *e.g.* maximization of the marginal likelihood in Equation (2.23). The fundamental idea is to split the training set into two disjoint sets, one which is used for training, and the other, the *validation set*, which is used to monitor performance. The performance on the validation set is used as a proxy for the generalization error and model selection is carried out using this measure.

In practice, a downside of the hold-out method is that only a fraction of the full data set can be used for training, and that if the validation set is small, the performance estimate obtained may have large variance. To minimize these problems, CV is routinely used in the k-fold cross-validation setting: the data is split into k disjoint, equally sized subsets; validation is done on a single subset and training is done using the union of the remaining $k-1$ subsets, the entire procedure being repeated k times, each time with a different subset for validation. Thus, a large fraction of the data can be used for training, and all cases appear as validation cases. The price is that k models must be trained instead of one. Typical values for k are in the range of 3 to 10. The logarithmic predictive probability which can be used as a measure of performance given by:

$$L_{\text{k-Folds}}(\mathbf{Z}, \mathbf{y}, \boldsymbol{\phi}) = \sum_{i=1}^{n} \log p(y_i|\mathbf{Z}, \mathbf{y}_{(\setminus I_{k(i)})}, \boldsymbol{\phi}) \tag{2.24}$$

where $\setminus I_{k(i)}$ correspond to the indices that are not in the the CV fold of observation i.

Chapter 3

Methods and Algorithms

In this chapter, we explain the necessary adaptations to the LMPC original architecture that enable tackling the autonomous racing problem. In particular, we show how to build the terminal components and the resulting optimization problem. Furthermore, we describe the vehicle models used to describe the dynamics in the MPC. Finally, we demonstrate how Gaussian Process Regression is used to predict the nominal model error.

3.1 LMPC for Autonomous Racing

Rosolia and Borrelli first introduced in [85] the adaptation of the core LMPC architecture to the autonomous racing problem. This is formulated as a *minimum time problem*, where an iteration j corresponds to a lap. Therefore, the stage cost is given as follows:

$$q(x_k, u_k) = \begin{cases} 1 & \text{if} \quad x_k \notin \mathcal{L} \\ 0 & \text{if} \quad x_k \in \mathcal{L} \end{cases} \tag{3.1}$$

where $\mathcal{L}$, given by Equation (3.3), is the set of points beyond the finish line. A slower trajectory contains more points until the finish line. Thus, has a greater cost associated.

The vehicle's dynamics are represented by the states and inputs vector quantities in Equation (3.2a) and Equation (3.2b), respectively.

$$\mathbf{x} = [s, e_y, e_\psi, v_x, v_y, r] \tag{3.2a}$$

$$\mathbf{u} = [a, \delta] \tag{3.2b}$$

where s is the distance travelled along the centerline of the track, e_y and e_ψ are the lateral distance and heading angle errors between the vehicle and the centerline, respectively. These quantities are given in the curvilinear abscissa reference frame, see Figure 3.1, also known as the Frenet reference frame [86]. In particular, a given track is defined by the curvature $k(s)$ and maximum admissible lateral error $e_y^{max}(s)$ along the track's centerline. v_x and v_y are the longitudinal and lateral vehicle velocities, respectively,

see Figure 3.2, while r is the vehicle's yaw rate. The inputs are the longitudinal acceleration a and the steering angle δ.

$$\mathcal{L} = \{\mathbf{x} \in \mathbb{R}^6 : \mathbf{x}e_1^{\mathrm{T}} = s > s_{target}\} \tag{3.3}$$

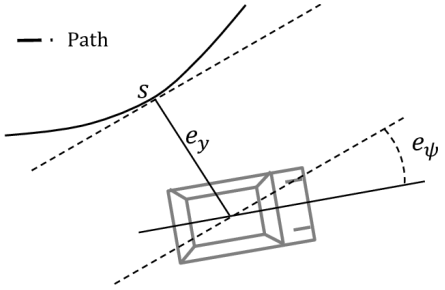


Figure 3.1: Frenet Frame [reprinted from [85]]

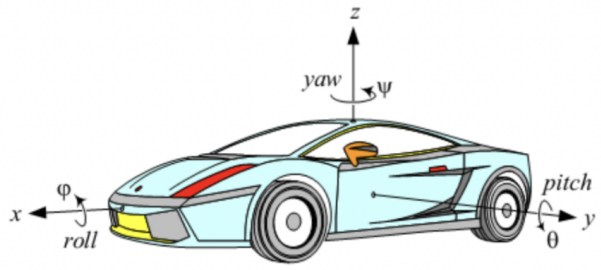


Figure 3.2: Vehicle Frames [reprinted from [87]]

In order to avoid the integer nature of the original framework mentioned before and to enable computational tractability, the authors introduce a time-varying approximated safe set and Q-function. These approximated quantities consist of 5-th order polynomial function approximations of the original quantities using the corresponding data from a given number of data points from the $j-1$ trajectory whose state is closest to the current state at time t of iteration j.

In [88], the authors build upon this formulation by targeting the ILC issue of repetitive tasks, *i.e.* the initial state is the same for all trials. In the suggested approach, the terminal condition of one iteration becomes the initial condition of the next iteration.

Rosolia and Borrelli [89] further extended this architecture by proposing a local LMPC that significantly reduced the computational burden by using a subset of the stored data. In particular, the local convex safe set $\mathcal{CS}_l^j$, Equation (2.9), is built around the candidate terminal state c_t using the N_p^{SS}-nearest neighbours from each of the previous N_l^{SS} laps. Notice that $N_l^{SS} = j - l$. These points are collected in the matrix $\mathbf{D}_l^j$, defined in Equation (3.4), which is updated at each time step. The candidate terminal state c_t is the estimated value for $\mathbf{x}_{t+N|t}$, calculated at time $t-1$. The approximation of the cost-to-go is computed as in Equation (2.10), using the costs associated with the selected states in $\mathbf{D}_l^j$.

$$\mathbf{D}_l^j = [\mathbf{x}_{t_1}^l, \cdots, \mathbf{x}_{t_{N_p^{SS}}}^l, \cdots, \mathbf{x}_{t_1}^j, \cdots, \mathbf{x}_{t_{N_p^{SS}}}^j] \tag{3.4}$$

See Figures 3.3 and 3.4 for a visual representation of the learning terminal components. The terminal state $\mathbf{x}_{t+N|t}$ must lie inside the convex hull defined by the blue points in Figure 3.3 which constitute the local safe set $\mathbf{D}_l^j$ in Equation (3.4). The slower the lap, the more datapoints are collected in that iteration which amounts to a greater cost as seen in Figure 3.4.

The original LMPC architecture represents the vehicle's pose in the local coordinate frame, Equation (3.2a). However, we found that the discretization method used that assumes constant track curvature κ within a sampling period fails to properly describe complex tracks. In the FSG track, within a single meter, the curvature can change as much $0.14\,\mathrm{m}^{-1}$ which is almost half of the track's maximum curvature. At a control frequency of $10\,\mathrm{Hz}$ it takes a velocity of just $10\,\mathrm{m\,s}^{-1}$ to travel $1\,\mathrm{m}$. Thus, proving that this modelling strategy is unfit. The trajectory behaviour at a given corner becomes very sharp

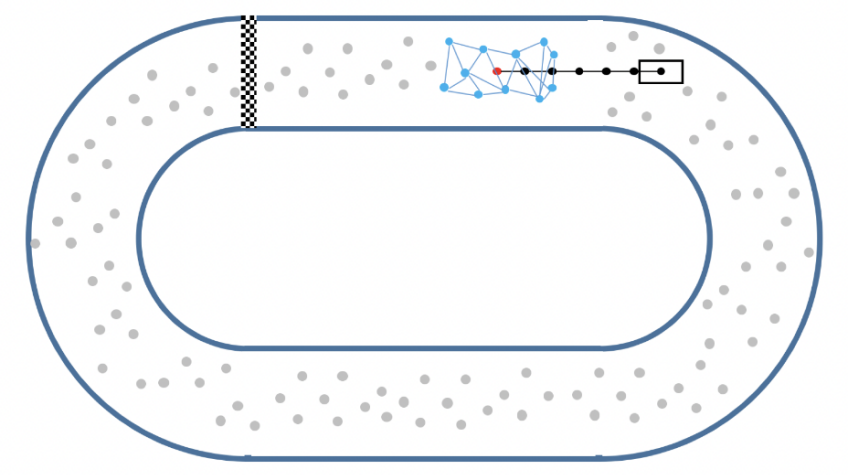


Figure 3.3: Local Safe Set [reprinted from [90]]

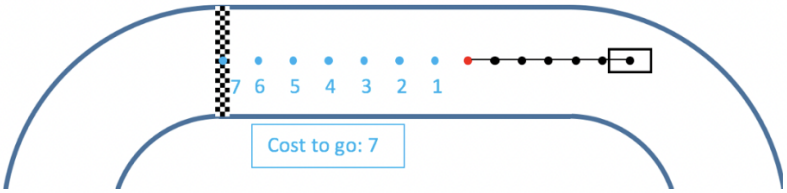


Figure 3.4: Q-function: cost-to-go [reprinted from [90]]

rather than smooth. I argue that the authors in [89, 90] might have not faced this issue because they have showed results for constant curvature corners.

Therefore, I have decided to change the vehicle model's pose states to global coordinates to fix this issue. x and y are the vehicle's longitudinal and lateral position with respect to a global coordinates frame, and ψ is the heading angle. s and e_y are still calculated because they provide immediate information regarding the track but they are not used to characterize the vehicle's pose dynamics. The longitudinal control input is $P \in [-1, 1]$ which represents a pedal setpoint. This corresponds to the normalized acceleration and brake pedal travel. This is the way the actual prototype is controlled both in simulation and reality. Thus, for our application, Equation (3.2) becomes:

$$\mathbf{x} = [x,\ y,\ \psi,\ v_x,\ v_y,\ r] \tag{3.5a}$$

$$\mathbf{u} = [P,\ \delta] \tag{3.5b}$$

where $\mathbf{x}$ is the vector of states that describe the vehicle's movement and $\mathbf{u}$ is the vector of control inputs. The resulting optimization problem is given as follows:

$$
\begin{aligned}
J_{t\to t+N}^{LMPC,j} = \min_{u_{t|t},\ldots,t_{t+N-1|t}} \Bigg[& \sum_{k=t}^{t+N-1} \Big(\big(\mathbf{x}_{k+1|t} - \mathbf{x}_{k|t}\big)^T \mathbf{Q}_{deriv} \big(\mathbf{x}_{k+1|t} - \mathbf{x}_{k|t}\big) \Big) + \\
& \sum_{k=t}^{t+N-2} \Big(\big(\mathbf{u}_{k+1|t} - \mathbf{u}_{k|t}\big)^T \mathbf{R}_{deriv} (\mathbf{u}_{k+1|t} - \mathbf{u}_{k|t}) \big) \Big) + \\
& \sum_{k=t+1}^{t+N} \Big(Q_{lane}^{lin} \cdot \epsilon_{lane_{k|t}} + Q_{lane}^{quad} \cdot \epsilon_{lane_{k|t}}^2 \Big) + \\
& \sum_{k=t+1}^{t+N} \Big(Q_{v_{ub}}^{lin} \cdot \epsilon_{v_{k|t}} + Q_{v_{ub}}^{quad} \cdot \epsilon_{v_{k|t}}^2 \Big) + \\
& \sum_{k=t+1}^{t+N} \Big(Q_{el}^{lin} \cdot \epsilon_{el_{k|t}} + Q_{el}^{quad} \cdot \epsilon_{el_{k|t}}^2 \Big) + \\
& \sum_{k=t+1}^{t+N} \Big(Q_{v_y} \cdot v_{y_{k|t}}^2 \Big) + \sum_{k=t}^{t+N} \Big(Q_{lag} \cdot e_{l_{k|t}}^2 \Big) + \\
& \sum_{i=1}^{N_l^{SS} \times N_p^{SS}} \Big(Q_{term\ cost} \big(\alpha_i \times \mathbf{Q}_i^j(c_t) \big) \Big) + \\
& \sum_{i=1}^{N_l^{SS} \times N_p^{SS}} \Big(\mathbf{Q}_{slack} \big(\mathbf{x}_{t+N|t} - \alpha_i \mathbf{D}_i^j(c_t) \big)^2 \Big) \Bigg]
\end{aligned} \tag{3.6a}
$$

s.t.

$$\mathbf{x}_{k+1|t} = h_t(\mathbf{x}_{k|t}, \mathbf{u}_{k|t}) \qquad \forall k \in [t, \cdots, t+N-1] \tag{3.6b}$$

$$P_{lb} \leq P_{k|t} \leq P_{ub} \qquad \forall k \in [t, \cdots, t+N-1] \tag{3.6c}$$

$$-\delta_b \leq \delta_{k|t} \leq \delta_b \qquad \forall k \in [t, \cdots, t+N-1] \tag{3.6d}$$

$$0 \leq s_{t|t} - s_{t-1|t-1} \leq \Delta s_{max} \tag{3.6e}$$

$$0 \leq s_{k+1|t} - s_{k|t} \leq \Delta s_{max} \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6f}$$

$$\Delta P_{lb} \leq P_{k+1|t} - P_{k|t} \leq \Delta P_{ub} \qquad \forall k \in [t+1, \cdots, t+N-2] \tag{3.6g}$$

$$\Delta \delta_{lb} \leq \delta_{k+1|t} - \delta_{k|t} \leq \Delta \delta_{ub} \qquad \forall k \in [t+1, \cdots, t+N-2] \tag{3.6h}$$

$$0 \leq \big(v_{x_{k|t}}/v_{x_{max}}\big)^2 + \big(v_{y_{k|t}}/v_{y_{max}}\big)^2 \leq 1 + \epsilon_{el_{k|t}} \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6i}$$

$$0 \leq v_{x_{k|t}} \leq v_{x_{ub}} + \epsilon_{v_{k|t}} \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6j}$$

$$-\epsilon_{lane_{k|t}} - e_{y_{k|t}}^{max}(s) \leq e_{y_{k|t}} \leq \epsilon_{lane_{k|t}} + e_{y_{k|t}}^{max}(s) \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6k}$$

$$\epsilon_{v_{k|t}} \geq 0 \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6l}$$

$$\epsilon_{lane_{k|t}} \geq 0 \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6m}$$

$$\epsilon_{el_{k|t}} \geq 0 \qquad \forall k \in [t+1, \cdots, t+N] \tag{3.6n}$$

$$\alpha_i \geq 0 \qquad \forall i \in [1, \cdots, N_l^{SS} \times N_p^{SS}] \tag{3.6o}$$

$$\sum_{i=1}^{N_l^{SS} \times N_p^{SS}} \alpha_i = 1 \qquad \forall i \in [1, \cdots, N_l^{SS} \times N_p^{SS}] \tag{3.6p}$$

$$\mathbf{x}_{t|t} = \mathbf{x}_t^j \tag{3.6q}$$

$$\mathbf{u}_{t|t} = \mathbf{u}_t^j \tag{3.6r}$$

The cost function given by Equation (3.6a) has five main parts. First, the derivative terms whose gains are denoted by the subscript *deriv*. These apply a penalty on the squared change of a given quantity between consecutive steps along the prediction horizon, both for dynamic states, *i.e.* $\mathbf{Q}_{deriv} = \text{diag}(0, 0, 0, Q^{v_x}_{deriv}, Q^{v_y}_{deriv}, Q^{r}_{deriv})$ and inputs - $\mathbf{R}_{deriv} = \text{diag}(R^{P}_{deriv}, R^{\delta}_{deriv})$. This enables one to control how aggressively the controller behaves and obtain smooth trajectories. The quadratic cost on v_y acts as a regularization cost which forces the vehicle into its stable domain and helps convergence.

The third part concerns the soft constraints on the states. Bounds on the states should not be implemented as hard constraints since one cannot exclude that the real system moves outside the constraint range due to, for instance, model mismatch which would render the problem infeasible [32]. Thus the bound on a given state $x \leq x_{max}$ can be approximated by $x \leq x_{max} + \epsilon$ where $\epsilon \geq 0$ and a term $l(\epsilon)$ is added to the cost functional. It can be shown that $l(\epsilon) = u\epsilon + v\epsilon^2$ with a sufficiently high u and $v > 0$ ensures that no constraint violation occurs provided there exists a feasible input. This term is preferable to $l(\epsilon) = u\epsilon$ for its smoothness properties. However, note that $l(\epsilon) = v\epsilon^2$ does not ensure such property holds at all times. The bounded states are v_x and e_y. The vehicle has a maximum velocity $v_{x_{max}}$ rising from the electric motor's maximum rotational velocity but for safety reasons can be electronically reduced to $v_{x_{ub}}$. e_y needs to be bounded to stay within the track width. Finally, a velocity ellipse is implemented to ensure the vehicle remains within its physical limits. These soft constraints correspond to the inequalities Equations (3.6i)–(3.6k) and Equations (3.6l)–(3.6n) and their respective cost functional terms.

The information of e_y, which in turn requires knowledge of s, based on a given position pair (x, y) is necessary to enforce the track layout constraint. One uses the MPCC architecture proposed by [71] to convert the global coordinates to local coordinates within the optimization problem. See Figure 3.5 for a depiction of the contouring error and lag error whose linear approximations we use in our optimization, *i.e.* $e_y = \hat{e}^c$ and $e_l = \hat{e}^l$. In that Figure, the capitalized quantities are analogous to the lower-case ones used throughout this thesis.

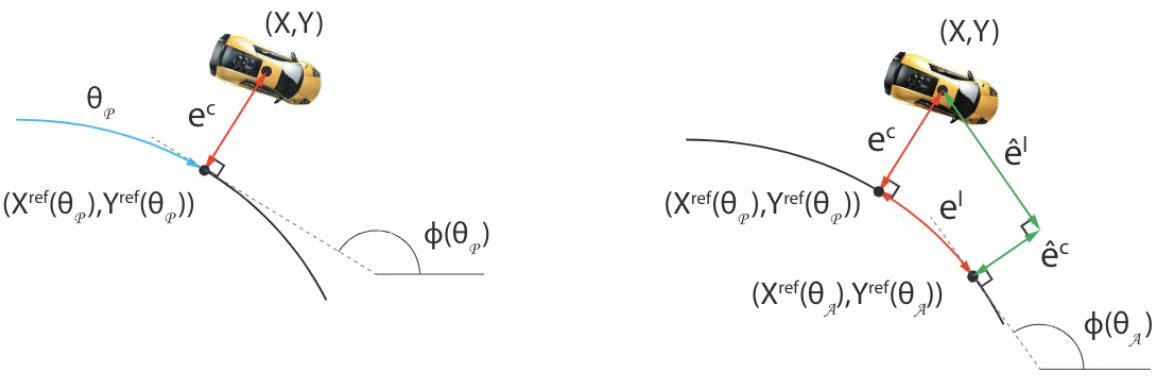


Figure 3.5: Contouring error e^c (left) and lag error e^l (right) with linear approximations $\hat{e}^c$ and $\hat{e}^l$ [reprinted from [71]]

$$\theta_{\mathcal{P}} = \mathcal{P}(x, y) = \underset{s}{\operatorname{argmin}} \left(\left(x - x^{ref}(s)\right)^2 + \left(y - y^{ref}(s)\right)^2 \right) \tag{3.7}$$

The projection operator $\theta_{\mathcal{P}}$ resembles an optimization problem. Therefore, the resulting optimization problem would be a bi-level Nonlinear Programming (NLP) optimization problem which is hard to solve in real-time. Therefore, an approximation $s = \theta_{\mathcal{A}}$ of $\theta_{\mathcal{P}}$ is introduced whose approximation quality is measured by the lag error, which should be as small as possible. Hence, Q_{lag} should be relatively high [91].

$$e^l = |\theta_{\mathcal{P}} - \theta_{\mathcal{A}}| \tag{3.8}$$

The track centerline is interpolated using third-order spline polynomials over N_s intervals. The linear approximations are computed as follows:

$$e^c \approx e_y(x, y, \theta_{\mathcal{A}}) = \sin\left(\Phi(\theta_{\mathcal{A}})\right)\left(x - x^{ref}(\theta_{\mathcal{A}})\right) - \cos\left(\Phi(\theta_{\mathcal{A}})\right)\left(y - y^{ref}(\theta_{\mathcal{A}})\right) \tag{3.9a}$$

$$e^l \approx e_l(x, y, \theta_{\mathcal{A}}) = -\cos\left(\Phi(\theta_{\mathcal{A}})\right)\left(x - x^{ref}(\theta_{\mathcal{A}})\right) - \sin\left(\Phi(\theta_{\mathcal{A}})\right)\left(y - y^{ref}(\theta_{\mathcal{A}})\right) \tag{3.9b}$$

where $\Phi(\theta_{\mathcal{A}})$ is the angle of the tangent to the centerline at the reference point with respect to the x-axis and is given as follows:

$$\Phi(\theta_{\mathcal{A}}) = \arctan \frac{\partial y^{ref}(\theta_{\mathcal{A}})}{\partial x^{ref}(\theta_{\mathcal{A}})} \tag{3.10}$$

Finally, the last two terms constitute the penalty on the terminal components of the MPC. α_i, already introduced in Equation (2.9), are the coefficients of the local safe set's convex hull, inherently bounded by Equations (3.6o) and (3.6p). α_i are also optimization variables. $\mathbf{Q}^j$ and $\mathbf{D}^j$ are given by Equations (2.8) and (3.4), respectively. For the former, one is still using the minimum time problem stage cost given by Equation (3.1). While the slack term ensures the terminal state lies within the convex hull, the terminal cost favours points in the safe set that resulted in faster laps.

Equations (3.6c) and (3.6d) constitute the bounds on the inputs. These bounds may be more restrictive than the actual physical limits imposed by the robotic platform. Equations (3.6g) and (3.6h) are the corresponding rate of change constraint. Again, both can be set to smaller values than the physical limits. The pedal setpoint has no relevant rate physical limit but imposing this bound ensures wheel slippage is avoided in case the derivative cost did not achieve this already. The steering setpoint has a maximum angular velocity constraint which results from the servomotor limits. The bound on change of s, Equations (3.6e) and (3.6f), aids the local coordinates approximation convergence, especially near the new lap segment.

Equations (3.6q)–(3.6r) define the initial conditions of the optimization. $\mathbf{x}_t^j$ corresponds to the most recent observation of the vehicle state. While $\mathbf{u}_t^j$ is the control actuation that is to be applied at the current sampling time. It corresponds to the previous sampling time solution shifted by one step, Equation (3.11). This delay applied to the system aims to account for solver processing time - which is significant in real-time and may be inconsistent across the experiment - and to keep a constant node rate.

$$\mathbf{u}_{t|t} = \mathbf{u}_t^j = \mathbf{u}_{t|t-1}^{*,j} \tag{3.11}$$

3.1.1 Vehicle Model

The system dynamics in Equation (3.6b) are given in its continuous-time form by Equation (3.12) which is the sum of an *a priori physics-based model* f_t and a term to model the *unknown dynamics* g_t, *i.e.* those not represented by f_t.

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) = \underbrace{f_t(\mathbf{x}_t, \mathbf{u}_t)}_{a\ priori\ model} + \underbrace{g_t(\mathbf{x}_t, \mathbf{u}_t)}_{unknown\ dynamics} \tag{3.12}$$

This section concerns the *a priori* model f_t. The pose dynamics can be derived to give the following equation:

$$\dot{x} = v_x \cos(\psi) - v_y \sin(\psi) \tag{3.13a}$$

$$\dot{y} = v_x \sin(\psi) + v_y \cos(\psi) \tag{3.13b}$$

$$\dot{\psi} = r \tag{3.13c}$$

The dynamic part of the vehicle model, *i.e.* related to Newton's first law, has been modelled with two different techniques. First, by employing a data-driven system identification scheme. Second, by deriving a physics-based first-principles model.

System Identification

f_t can be thought to be in the state-space form. Thus, given some datapoints, the goal is to identify the $\mathbf{A}$ and $\mathbf{B}$ matrices that best describe the state evolution using a system identification technique.

$$\dot{\mathbf{x}}(t) = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \tag{3.14}$$

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \theta_{x,1} & \theta_{x,2} & \theta_{x,3} \\ \theta_{y,1} & \theta_{y,2} & \theta_{y,3} \\ \theta_{\psi,1} & \theta_{\psi,2} & \theta_{\psi,3} \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ r \end{bmatrix} + \begin{bmatrix} \theta_{x,4} & \theta_{x,5} \\ 0 & \theta_{y,4} \\ 0 & \theta_{\psi,4} \end{bmatrix} \begin{bmatrix} P \\ \delta \end{bmatrix} \tag{3.15}$$

The $\boldsymbol{\theta}$ parameters are identified for each dynamic quantity - v_x, v_y, r - by solving the Least Mean Square problem of an overdetermined system. Using $l = N_P^{SI}$ points that belong to a given feature set of pre-collected data:

$$\mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_l \end{bmatrix} \qquad \mathbf{X} = \begin{bmatrix} x_{1,1} & x_{2,1} & \cdots & x_{o,1} \\ x_{1,2} & x_{2,2} & \cdots & x_{o,2} \\ \vdots & \vdots & \vdots & \vdots \\ x_{1,l} & x_{2,l} & \cdots & x_{o,l} \end{bmatrix} \qquad \boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \vdots \\ \theta_o \end{bmatrix} \tag{3.16}$$

where $o = 5$, corresponding to the feature vector $[v_x, v_y, r, P, \delta]$, for the computation of $\boldsymbol{\theta}_x$ and $o = 4$ for the remaining, corresponding to the feature vector $[v_x, v_y, r, \delta]$. For a given feature point, b_i corre-

sponds to the difference observed in one of the feature quantities in consecutive sampling times. The $\boldsymbol{\theta}$ parameters used in Equation (3.15) are the solution of the following problem:

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\| \mathbf{b} - \mathbf{X}\boldsymbol{\theta} \right\|_2 \tag{3.17}$$

Dynamic Bicycle Model

The physics-derived vehicle model used is a dynamic bicycle model - Figure 3.6. This model is frequently used in automotive control algorithms [87]. The model is similar to the one used to model the vehicle in the simulator - Section 4.1. Nonetheless, some simplifications have been implemented to ease the real-time solution of the MPC. Namely, the model is purely dynamic and does not blend with a kinematic bicycle model at low velocities; the wheels' assembly rotational inertia is disregarded, *i.e.* $m_{eq} = m$; the Pacejka tire model is simpler, both in terms of the tire modelling equation and normal load; and, the planar model used for the derivation of $\dot{r}$ is not employed, remaining a purely bicycle approximation, *i.e.* the term $(F_{F,y,left} - F_{F,y,right})\frac{t_F}{2}\sin\delta$ is disregarded. Thus, the model is given as follows:

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}(F_x - F_{F,y}\sin\delta + mv_y r) \\ \frac{1}{m}(F_{R,y} + F_{F,y}\cos\delta - mv_x r) \\ \frac{1}{I_z}(F_{F,y}l_F\cos\delta - F_{R,y}l_R) \end{bmatrix} \tag{3.18}$$

where m is the vehicle's mass and I_z is the rotational inertia about the vertical axis z. The front and rear axles are identified by the subscripts $a \in \{F, R\}$, respectively. l_a is distance between the vehicle's center of gravity and the corresponding axle. The lateral force $F_{a,y}$ is given as:

$$F_{a,y} = -2D_a \sin\left(C_a \arctan(B_a\alpha_a)\right)) \tag{3.19}$$

where the tire coefficients B_a, C_a and D_a are experimentally identified and α_a - the angle between the tire's centerline and its velocity vector - is given computed as follows:

$$\alpha_F = \arctan\left(\frac{v_y + l_F r}{v_x}\right) - \delta \tag{3.20a}$$

$$\alpha_R = \arctan\left(\frac{v_y - l_R r}{v_x}\right) \tag{3.20b}$$

The longitudinal force F_x is given as follows:

$$F_x = 2 \cdot \frac{T_{max} \cdot GR \cdot P}{r_{wheel}} - C_{roll} \cdot m \cdot g + \frac{1}{2}\rho \cdot C_d \cdot A_f \cdot v_x^2 \tag{3.21}$$

where T_{max} is the maximum available torque at each of the two rear axle in-wheel motors whose maximum practical value is $21\,\mathrm{Nm}$ but a smaller value may be used for safety reasons. GR is the transmission's gear ratio and C_{roll} is the roll resistance factor. Concerning the aerodynamic drag force, ρ is the

air density, A_f is the vehicle's frontal area used as a reference for the force calculation and C_d is the drag coefficient.

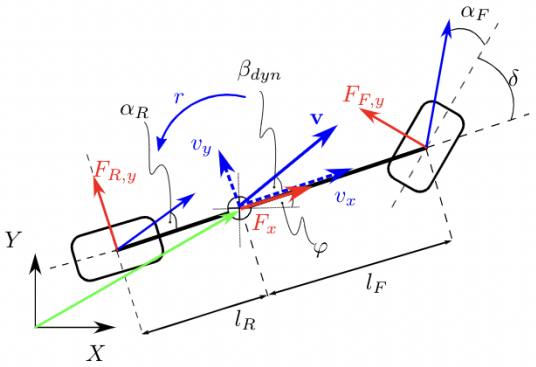


Figure 3.6: Dynamic Bicycle Model: position vectors are in depicted in green, velocities in blue, and forces in red. [reprinted from [67]]

3.2 Gaussian Processes Regression

GPR is used to predict the error between the vehicle model - Section 3.1.1 - and the available measurements, *i.e.* estimate g_t in Equation (3.12). One assumes that the modelling error only affects the dynamic part of the first-principle model, *i.e.* the velocity states. Therefore, the training outputs are given by the difference between the measurement $\mathbf{x}_{k+1}$ and the nominal model predictions:

$$\mathbf{y}_k = \mathbf{B}_d^{\dagger}\big(\mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k)\big) = \mathbf{g}(\mathbf{z}_k) + \mathbf{w}_k \tag{3.22}$$

where $\mathbf{B}_d^{\dagger}$ is the Moore-Penrose pseudo-inverse of $\mathbf{B}_d = [\mathbf{0}_{3\times3}; \mathbf{I}_{3\times3}]$. Hence, $d \in \{e_{v_x}, e_{v_y}, e_r\}$ and $n_g = 3$.

As a first iteration, we chose the GP training inputs, *i.e.* the feature state to be $\mathbf{z} = \{v_x; v_y; r; P; \delta\}$ and $n_z = 5$. This is based on the assumption that model errors are independent of the vehicle's position. This disregards potential modelling shortcomings introduced from, for instance, segments of the track that have different traction conditions, *e.g.* due to puddles. Later, we have decided to remove v_y as we identified a strong correlation between this quantity and r. This is not very surprising as both quantities characterize the lateral movement of the vehicle. The selection for removal of v_y instead of r is justified by the fact that it is hard to precisely estimate v_y while r is measured directly using a gyroscope. Notwithstanding, this seems like a reasonable approximation which, furthermore, reduces the learning problem dimensionality.

The covariance function used is the SE kernel function used as an example kernel function in Section 2.3 (and we restate below in Equation (3.23)) with the independent measurement noise component - $\sigma_{n,d}^2 \delta_{\mathbf{z}\bar{\mathbf{z}}}$.

$$k_{SE}^d(\mathbf{z}, \bar{\mathbf{z}}) = \sigma_{f,d}^2 \ \exp\left(-\frac{1}{2}\frac{(\mathbf{z}-\bar{\mathbf{z}})^{\mathrm{T}}(\mathbf{z}-\bar{\mathbf{z}})}{l_d^2}\right) + \sigma_{n,d}^2 \delta_{\mathbf{z}\bar{\mathbf{z}}} \tag{3.23}$$

Two sparse approximations have been explored: *i)* SoD and *ii)* FITC or FIC (see Section 2.3 for details on the theoretical formulation). The first approximation method is essentially a full GP that does not use all data available. This means, the computations for the error prediction in Equation (3.12) are those of an exact GP given by Equation (2.13a). While for the second approach the computations are those of Equation (2.20a).

Finally, it should be noted that some quantities can be pre-computed which otherwise could prevent real-time feasibility. Particularly, $\left(\mathbf{K}_{\mathbf{ZZ}}^{d} + \mathbf{I}\sigma_{d}^{2}\right)^{-1}\mathbf{y}^{d}$ in Equation (2.13a) only needs to be recomputed whenever the training data $\mathcal{D}$ is changed - this corresponds to `training`. In the SoD offline fashion it is only computed once before the controller is launched. `Inference` corresponds to the rest of the computation of Equation (2.13a).

In the FITC approximation, the `training` part corresponds to determination of the information vector which only requires recomputation when either the training dataset $\mathcal{D}$ or the inducing points Z_{ind} are changed. In our application, however, the inducing points are update at each sampling time. They are equally distributed along the last sampling time's shift predicted trajectory. This is a sensible placement of the inducing points since the new test cases are expected to be proximate as the trajectory does not change significantly at consecutive sampling times. This means the online adaptation of the dictionary $\mathcal{D}$ is immediately possible.

GPR is generally susceptible to outliers, which can hinder the model error learning performance [70]. Moreover, large and sudden changes in the GP predictions can lead to erratic driving behaviour. To attenuate these effects, one only includes datapoints in the dictionary $\mathcal{D}$ (both online and offline) whose measurements fall within predefined bounds $\pm\mathbf{y}_{lim}$, defined from physical considerations and empirical knowledge. The bounds are given by $ub_{e_{v_x}}$, $lb_{e_{v_x}}$, $b_{e_{v_y}}$, b_{e_r}, where the upper and lower bound are not symmetric for e_{v_x} as it is for e_{v_y} and e_r. These bounds are also enforced to the GPR prediction - Equations (2.13a) and (2.20a).

3.3 Control Architecture

There are two variations of the LMPC architecture given by Algorithm 1. They differ only in Line 11. In the pre-computed version the model error predictions are fed directly to the solver based on the previous sampling time's shifted trajectory. While in the solver-embedded version the model error predictions are calculated at each solver's iteration and only the information vector $\mathbf{i}^d$ is pre-computed. The embedded version is only available when using the FITC approximation as otherwise it would not be real-time feasible.

Algorithm 1: LMPC Architecture

```
1  Initializations;
2  while Event not finished do
3      Update car state from SLAM data;
4      Publish control command [Equation (3.11)];
5      Convert to local coordinates;
6      if Lap finished then
7          Add closed-loop data to safe set [Equation (2.6)];
8      if Using LMS model then
9          Identify model using LMS [Equation (3.17)];
10     if Model learning active then
11         Compute GPR [Section 4.3.2];
12     Find local safe set [Equation (3.4)];
13     Solve nonlinear optimization [Section 4.3.1];
14     Store measurement data [Equation (2.2)];
15     if Online learning active then
16         Update dictionary [Equation (2.12)]
```

Chapter 4

Implementation

In this chapter, we describe the relevant aspects of the controller implementation. We start by detailing the simulation environment's vehicle model. Subsequently, we characterize the platform's hardware and software setup. Lastly, the details of the Learning-based Model Predictive Controller implementation are given. In particular, the solver used for running the Model Predictive Control optimization and the Gaussian Processes Regression inference and tuning processes.

4.1 Simulation Platform

FSSIM[1] is the vehicle simulator used to the test the controllers developed throughout this thesis work. It is also the simulator of choice for FST Lisboa where the algorithms are initially tested before moving to testing over pre-collected data from the prototype and, finally, testing on the actual race car. AMZ Driverless developed this vehicle simulator dedicated to the FSD competition and released it open-source to other teams. This team reported 1% lap-time accuracy compared with their FSG 2018 trackdrive run [67].

FSSIM comes with the standard Acceleration and Skidpad competition tracks. Additionally, it includes track layout data mapped from the 2018 official FS events of Italy and Germany. This simulator features characteristics akin to the real competition such as Remote Emergency Stop activation when the vehicle leaves the track with all four wheels, time penalization when hitting a track-delimiting cone and lap-time counter.

Due to real-time requirements, this simulator does not simulate raw sensor data, *e.g.* camera or LiDAR data. Instead, cone observations around the vehicle are simulated using a given cone-sensor model. Both exteroceptive sensors are modelled with the parameters of Table 4.1.

[1] `https://github.com/AMZ-Driverless/fssim`

Parameters	Camera	LiDAR
$\Delta t_{lat}\ (s)$	0.2	0.2
$\sigma_{lat}\ (s)$	0.002	0.002
$r_{obs}^{max}(m)$	10	15
d_{obs}	100	100
l_{obs}	1	1
$r_{color}^{max}(m)$	10	10
d_{color}	200	200
l_{color}	0.99	0.8
$\sigma_r\ (m)$	0.2	0.05
$\sigma_\theta\ (rad)$	0.007	0.007

Table 4.1: Cone-Sensor Model Parameters

Δt_{lat} replicates the average processing time of the corresponding perception pipeline algorithms and σ_{lat} is the standard deviation of the additive white Gaussian noise added on the latency measurement to mimic the real-time variation of the computations. The subscripts $det \in \{obs, color\}$ refer to the capability to detect a cone and to correctly identify the cone color, respectively. Note that LiDAR can detect cones at distance smaller than $15\,\text{m}$ but it can only detect their color at less than $10\,\text{m}$. r_{det} is the distance between the vehicle and a given cone and θ is the angle between the vehicle longitudinal axis and a cone. r_{det}^{max} represents the distance before which the cones can be detected and their class identified, respectively. d_{det} is a scaling-factor used to calculate the probability p_{det}, for either subscript, that a given cone is detected or correctly classified as a function of its distance from the car, given as follows:

$$p_{det} = l_{det} \times \left(1 - \frac{r_{det}}{d_{det}}\right) \tag{4.1}$$

Gaussian noise is added to the distance and angle measurements in the polar coordinates. σ_r and σ_θ are the corresponding standard deviations of the noise model. We have set the distributions' mean to be zero but this ought not to be the case.

FSSIM simulates the vehicle dynamics using a model that blends a dynamic and a kinematic bicycle model [67]. The dynamic bicycle model is ill-defined for slow velocities due to the tire slip angles considered to compute the tire forces. In a racing application, most of the track is spent at high-speeds where this model accurately represents the vehicle behaviour. At low speeds, *e.g.* at race start or in sharp corners, the kinematic bicycle model constitutes a faithful description of the vehicle dynamics. On the other hand, this model provides inaccurate estimations at high-speeds since it neglects the highly nonlinear tire forces that shape the movement. The blended model is discretized with the Euler Forward discretization.

The vehicle model is derived under the assumptions that: *i)* the vehicle drives on a flat surface, *ii)* load transfer can be neglected, *iii)* combined slip can be neglected, and *iv)* the longitudinal drivetrain forces act on the center of gravity. The state of the model is given by the vector $\mathbf{x} = [x, y, \psi, v_x, v_y, r]^T$

and the control inputs are $\mathbf{u} = [P, \delta]^T$. The longitudinal control input is $P \in [-1, 1]$ which represents a pedal travel setpoint,*i.e.* from full actuation on the brake pedal to full actuation on the accelerator pedal. The dynamic bicycle model dynamics are given by:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi \\ v_x \sin\psi + v_y \cos\psi \\ r \\ \frac{1}{m_{eq}}(F_x - F_{F,y} \sin\delta + m_{eq} v_y r) \\ \frac{1}{m}(F_{R,y} + F_{F,y} \cos\delta - m v_x r) \\ \frac{1}{I_z}(F_{F,y} l_F \cos\delta + (F_{F,y,left} - F_{F,y,right}) \frac{t_F}{2} \sin\delta - F_{R,y} l_R) \end{bmatrix} \tag{4.2}$$

where m_{eq} is the vehicle's equivalent inertia which corresponds to the longitudinal linear inertia, given by Equation (4.3). For the applicable subscripts, F and R correspond to the front and rear axles while x and y pertain to quantities along the longitudinal and lateral axes, respectively. The $left$ and $right$ subscripts concern the corresponding side of the car, facing forward. t_F is the vehicle's front track, *i.e.* distance between the front wheels.

$$m_{eq} = m + 4\frac{I_{rot}}{r_{wheel}^2} \tag{4.3}$$

where the second term corresponds to the four wheels' combined equivalent inertia. I_{rot} is the wheel assembly rotational inertia and r_{wheel} is the tire radius.

The lateral forces $F_{a,y}$, $a \in \{F, R\}$ exerted by the tires are modelled by a simplified Pacejka tire model [92].

$$F_{a,y} = F_{a,z} \mu_{a,y} \tag{4.4a}$$

$$\mu_{a,y} = D_a \sin(C_a \arctan\big((1 - E_a) B_a \alpha_a + E_a \arctan(B_a \alpha_a)\big) \tag{4.4b}$$

$$F_{a,z} = mg + \frac{1}{2} W_a C_L v_x^2 \tag{4.4c}$$

where $F_{a,z}$ is the normal load acting on a given tire, W_a is the front/rear weight distribution and C_L is the downforce coefficient - identified from experiments. B_a, C_a, D_a and E_a are experimentally identified coefficients and α_a are the tires slip angles given by Equation (3.20).

The resulting longitudinal force F_x applied on the CG can be calculated as follows:

$$F_x = C_m P - C_{r0} + C_D v_x^2 \tag{4.5}$$

where the three components correspond to a motor/drivetrain model, the rolling resistance and the aerodynamic drag. Their parameters are identified from experiments.

The lateral and angular accelerations of the kinematic model - two bottom rows of Equation (4.6) - are obtained through the differentiation of $v_{y,kin} = l_R \delta \frac{v_x}{l_F + l_R}$ and $r_{kin} = \tan(\delta) \frac{v_x}{l_F + l_R}$, using the small

angle approximation on δ, *i.e.* $\cos^2(\delta) \approx 1$ and $\tan\delta \approx \delta$. Thus, yielding the following dynamics:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \\ \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos\psi - v_y \sin\psi \\ v_x \sin\psi + v_y \cos\psi \\ r \\ \dfrac{F_x}{m_{eq}} \\ (\dot{\delta} v_x + \delta \dot{v}_x) \tan\delta \dfrac{l_R}{l_F + l_R} \\ (\dot{\delta} v_x + \delta \dot{v}_x) \tan\delta \dfrac{1}{l_F + l_R} \end{bmatrix} \tag{4.6}$$

The velocities $\mathbf{u} = [v_x, v_y, r]^T$ are linearly blended in the velocity range $v_x \in [v_{x,blend\ min}, v_{x,blend\ max}]$, where $v_{x,blend\ min} = 3\ m/s$ and $v_{x,blend\ max} = 5\ m/s$. Below the lower velocity limit, the kinematic model is used whereas above the upper limit the dynamic model imposes the dynamics.

$$\lambda = \min\left(\max\left(\frac{v_x - v_{x,blendmin}}{v_{x,blendmax} - v_{x,blendmin}}, 0\right), 1\right) \tag{4.7}$$

$$\mathbf{u} = \lambda \mathbf{u}_{dyn} + (1 - \lambda)\mathbf{u}_{kin} \tag{4.8}$$

The parameters used to simulate FST10d can be seen in Tables 4.2 and 4.3.

Parameter	Value	SI Units
m	250	kg
I_z	110	kg m^2
C	1.9	kg m^{-1}
C_D	0.7	kg m^{-1}
C	5000	N
C_{r0}	180	N
I_{rot}	0.4	kg m^2
r_{wheel}	0.231	m

Table 4.2: Simulator Vehicle Parameters

Parameter	Front	Rear
$l\ (m)$	0.765	0.765
$t\ (m)$	1.22	1.22
W	0.5	0.5
B	12.56	12.56
C	-1.38	-1.38
D	1.6	1.6
E	-0.58	-0.58

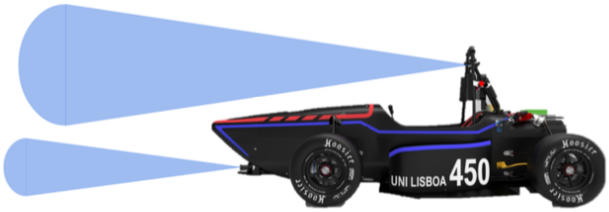
Table 4.3: Simulator Vehicle Parameters - Axle

4.2 Experimental Platform

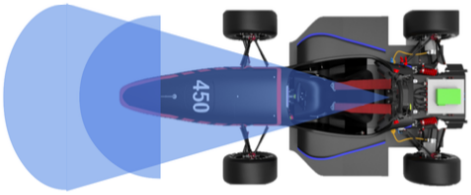
In this section, one will introduce the robotic platform in which the algorithms developed will be tested.

FST10d is the autonomous-empowered version of FST09e - a Formula Student electric prototype with a Carbon-fiber-reinforced-polymers monocoque chassis, a full aerodynamic package and a self-developed $600\,\mathrm{V}$ high-voltage battery with $8\,\mathrm{kW\,h}$ of energy. This battery supplied four $32.5\,\mathrm{kW}$ in-wheel motors which amounts to a peak power of $130\,\mathrm{kW}$ or 174 horsepower. The vehicle's top velocity is $106\,\mathrm{km\,h^{-1}}$ and it performed a $0\text{-}100\,\mathrm{km\,h^{-1}}$ acceleration in under $2.5\,\mathrm{s}$. Due to financial constraints, the front wheel motors had to be removed in FST10d.

The adaptation started with the addition of new sensors to enable autonomous driving. Figure 4.1 shows the location and Field of View (FoV) of the exteroceptive sensors used for perception. An OS1-16 LiDAR with a vertical resolution of $0.53°$ was placed on the forepart of the vehicle, approximately at cone height. A LUCID Triton RGB camera complements the LiDAR sparse points with information-rich data of the environment. The camera is placed close to where the pilot's head would be.



(a) Side view



(b) Top View

Figure 4.1: Location and FoV of Exteroceptive Sensors

A proprioceptive sensor in the form of a Xsens MTi-670 GNSS/INS sensor (short for Global Navigation Satellite/Inertial Navigation Systems). This sensor offers an estimate of the position, velocity and orientation by means of its sensor fusion algorithms, at up to $2000\,\mathrm{Hz}$. Inputs come from the module's on-board gyroscope, accelerometer, magnetometer and an external GNSS receiver.

Figure 4.2 exhibits the software architecture. The software stack is implemented using ROS (stands for Robot Operating System) that handles the communication and synchronization layer. ROS is an open-source robotics middleware suite and although it is not an operating system but a collection of software frameworks for robot software development, it provides services such as hardware abstraction, message-passing between processes and package management. Every module that requires heavy computing is implemented in C++ for its enhanced computational performance. Neural networks and non-critical nodes such as logging and visualization tools were developed in Python, given its swift and

simple development process. The software stack runs on an In-CarPC CQ77 rugged Processing Unit (PU) which features an NVIDIA GeForce GTX 1060 GPU with 1280 CUDA cores - where the perception's Convolutional Neural Networks (CNN) are run - and a 6 core Intel i7-8700T CPU.

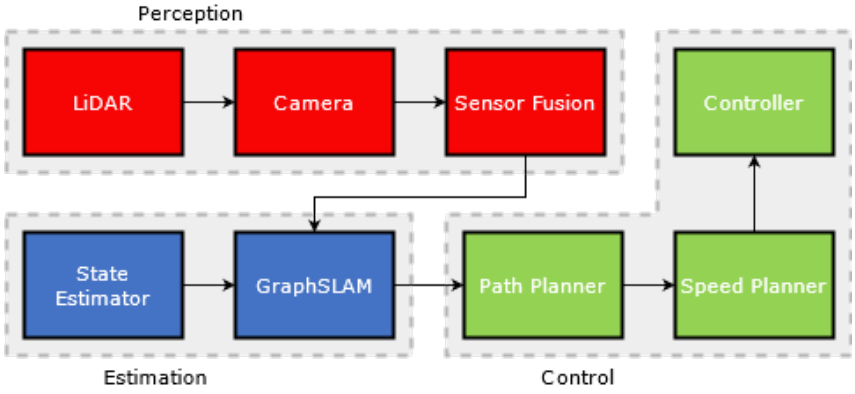


Figure 4.2: FST10d Autonomous Racing Software Stack

The *perception* module is responsible for detecting the track-delimiting cones and their features, *i.e.* position and color. Cones detected by the LiDAR pipeline serve as hypotheses of regions of interest in the image. While the *estimation* pipeline estimates the vehicle pose and velocities. Finally, the *control* pipeline computes the necessary actuation to follow the centerline.

The point cloud processing uses a combination of self-developed and open-source algorithms from PCL [93]. Initially, the LiDAR's FoV is trimmed using a pass-through filter which enables removing points along any given axis, so that points that undoubtedly lay outside the regions of interest are disregarded. Subsequently, a ground removal algorithm is applied by removing the biggest plane found using the iterative method of RANdom SAmple Consensus or RANSAC [94]. The cone proposals are identified by means of an Euclidean Clustering algorithm [95]. In order to add color information for cones nearby - less than 5 m - that fall outside the camera's FOV, a classification CNN was implemented based on the point cloud intensity pattern.

With the LiDAR data processing done, a 3D bounding box is calculated around each cone centroid, which is then projected onto the image plane. The region of the image where the bounding box falls is cropped and fed into a CNN. The output of the network is normalized over the five classes of cones (yellow, blue, orange, big orange, unknown). The network has been trained with a custom dataset of over 110000 images containing Driverless Formula Student cones. Our dataset was originally only composed by images from FSOCO[2] - a collaboration between FS teams that aims to accelerate the development of camera-based solutions in the context of FSD - but has since been enlarged with more images gathered from runs with our camera. Once all information has been extracted from both sensors (position and color from LiDAR, and color from camera), it is fused and conflicts are handled. See Figure 4.3 for a visual representation of the perception pipeline.

In order to obtain an accurate velocity estimation of the car, a Kalman Filter - State Estimator in Figure 4.2 - was implemented to fuse the measurements of the available sensors present in the car, namely inertial data, wheel speeds and steering angle. In the prediction step of the Kalman Filter, a car model

[2] https://ddavid.github.io/fsoco/

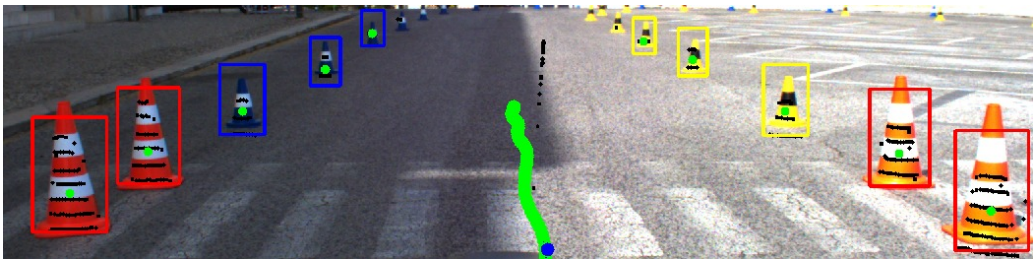
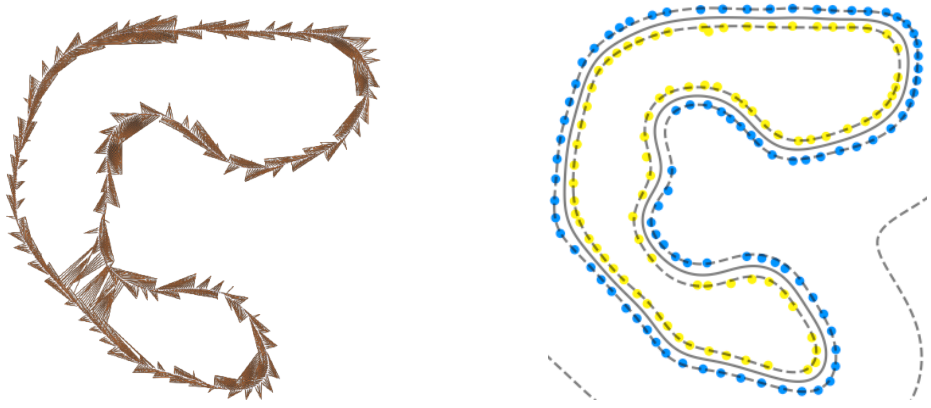


Figure 4.3: LiDAR Point Cloud Projection to Image Plane

was developed where a different approach from the current state of the art - bicycle model - was taken, by deducing the car dynamics through a linearized *LuGre tire model* for the whole car [96]. This node runs at 100 Hz.

The maximum range of the perception sensors limits the length of the path planning horizon. In order to explore the car's full potential a SLAM algorithm is implemented for mapping the track during the first lap at lower speeds and localizing the vehicle within the built map on the subsequent laps. Furthermore, track centerline information from this map and pose estimation are used in the LMPC.

We run a GraphSLAM algorithm with each new set of cone observations, *i.e.* detections from the sensor fusion node whose rate is limited by the LiDAR's 20 Hz acquisition rate. GraphSLAM is a full SLAM method that explores the SLAM sparse graph structure - Figure 4.4, where nodes represent either pose estimates or landmark locations with edges, denoting measurements, connecting them. This graph leads to a sum of non-linear constraints that, when linearized, form a least squares problem that can be optimized using standard optimization techniques. We have shown to have an average mapping error of just 30 cm.



(a) SLAM Sparse Graph Structure

(b) Resulting Optimized Map with SVM

Figure 4.4: GraphSLAM Mapping Results

In the Trackdrive event, the entire *control* module in Figure 4.2 (green nodes) is substituted by the LMPC algorithm. However, in other events where there is no a priori information of the map, or during the mapping lap, the green nodes are active. The path planner receives cone detections data and computes the track centerline, which is fed to the speed planner to create a speed reference for such path. Finally, the controllers follow the reference path.

Intuitively, one way to approach the path planning in an FSD track is to consider the left boundary cones (blue) one class and the right boundary cones (yellow) as another. Given this reasoning, one way to extract the center line of the track is to find the boundary between these two classes. The algorithm developed uses a Support Vector Machine, as a base, from which the center line can be extracted, as well as the track boundaries.

Once the centerline is discovered, the curvature of the path is calculated using three points at a time. With this curvature, a maximum lateral acceleration can be obtained from a point-mass model which is then used to look up the maximum longitudinal acceleration from a GG-diagram. Finally, a forward and backwards correction algorithm is employed to create a smooth velocity profile.

Longitudinal and lateral control are decoupled in this rather simple approach. The lateral controller is a Pure Pursuit Controller [25] which is essentially a proportional controller based on the lateral deviation measured from the vehicle's longitudinal axis and the distance from a point-to-follow. The point-to-follow is chosen based on the car's velocity as well as a look-ahead time. The longitudinal controller is a PI-controller whose setpoint is the velocity reference computed by the speed planner.

4.3 Controller Implementation

The LMPC algorithm this thesis work is based on has been developed by Professor Borrelli's Model Predictive Control Lab at the University of California Berkeley. Some of the architecture's variants have been tested and its code released open-source on the Berkeley Autonomous Race Car (BARC) [97]. The BARC is a development platform for autonomous driving on a 1/10 scale RC car. The electrical, mechanical and software architecture design have been released open-source[3].

The theoretical guarantees of LMPC presented under no model mismatch may not hold as such assumption is not satisfied in the racing application, especially due to the highly dynamic range of manoeuvres. Therefore, the researches at the MPC Lab propose a range of system identification and machine learning techniques to reduce the model error. We have also tested and extended some of the original proposals.

We have started the development of this thesis by testing some of the available code versions. In this step, I performed the necessary adaptations to FSSIM and the FST10d software pipeline; as well as, the required data collection and controller tuning.

Rosolia and Borrelli [89] suggested building an affine time-varying model to approximate the vehicle dynamics. The kinematic equations of motion that describe the evolution of the vehicle's position as a function of its velocities are linearized. Moreover, instead of identifying the parameters of a nonlinear dynamical model and then linearize it, the authors propose to directly learn a linear model around a state $\mathbf{x}$ using a local linear regressor - the Epanechnikov kernel function [98]. These quantities are evaluated along the shifted optimal solution of the LMPC problem at time $t-1$. Provided with these simplifications, the problem can be reformulated as a Quadratic Program (QP) which can be solved efficiently.

The control architecture has been implemented by the author in ROS, using Python and OSQP [99].

[3] `http://www.barc-project.com`

The code is available online[4]. However, the system identification scheme proved to be inadequate for our application. Both in terms of poor modelling capability and prohibitively high computational effort. The time allocated to system identification was greater than that of solving the optimization (QP), which does not seem to be sensible. This way, this Python implementation frequently enough broke the real-time requirement such that the controller performance was visually deteriorated. Processing times often surpassed the maximum acceptable processing time when running the node at $10\,\text{Hz}$, even though the average processing time was within that threshold. Therefore, a different implementation was necessary.

Xu [90] tested different modelling strategies for the LMPC applied to the autonomous racing problem on the 1/10 RC car. Besides testing the LMS scheme with different vehicle models, the author used GPR to model the system identification modelling error, *i.e.* model the unknown dynamics - g in Equation (3.12). Xu also employed some methods to reduce the computational burden of the LMS scheme.

The controller has been implemented in Julia, using JuMP [100] to formulate the optimization problem and Ipopt [101] as the solver. The code is available online[5]. One could set Ipopt's CPU time limit in such a way that the solver-related processing tasks respected the real-time constraint. And, while the average solver processing time was just $0.04\,\text{s}$, it did fail frequently to reach optimality within the available time interval. There are two other aspects related to Julia that hinder the implementation success. First, the integration with ROS is not widely supported and therefore misses many of the features. Second, Julia's garbage collection scheme results in rare but existent prohibitively high processing times. Furthermore, it did not seem possible to increase the node's frequency from 10 to $20\,\text{Hz}$ nor increase the prediction horizon to more than $N = 12$ which results in a lookahead time of just $1.2\,\text{s}$. From experience, the combination of these two factors are clearly insufficient for an autonomous racing application.

In conclusion, Julia and Ipopt while very useful for solving other optimization problems are not fit for autonomous racing real-time MPC. Hence, I decided to develop a custom C++ implementation of the LMPC architecture using FORCESPRO - a solver designed for embedded solving of MPCs - to solve the optimization problem. With these changes, the controller is able to run at $20\ Hz$ with $N > 20$.

The controller - Algorithm 1 - is implemented in ROS/C++. In the initialization procedure (Line 1), the track layout data previously mapped by the SLAM pipeline is loaded. This map is used for global to local coordinate transformation (Line 5). Then, reads the pre-collected data used to build the safe set (Equation (3.4)) and the corresponding cost-to-go (Equation (2.8)). This pre-collected data in a competition setting corresponds to the Autocross runs using the pipeline described in Section 4.2. Although, one could also use data from previous LMPC runs. Lastly, the GPR model is initialized, *i.e.* specify the covariance matrix, create the training points Dictionary (Equation (2.12)) and set the hyperparameters ϕ.

Finding the local safe set $\mathbf{D}_{\mathbf{l}}^{\mathbf{j}}$ in Equation (3.4) consists of a simple search of the N_p^{SS} closest points to the candidate terminal state c_t, for each of the last N_l^{SS} laps. The distance measure is taken about the respective points' track progress s. The choice of c_t greatly influences the controller's performance since the terminal state $x_{t+N|t}$ is to lay inside the convex set given by Equation (2.9) of the local safe set $\mathbf{D}_{\mathbf{l}}^{\mathbf{j}}$,

[4] https://github.com/MPC-Berkeley/barc/tree/devel-ugo
[5] https://github.com/MPC-Berkeley/barc/tree/LMPC-Shuqi

i.e. Conv($\mathbf{D}_1^j$). A candidate safe set that is too close to the vehicle results in unnecessary conservative behaviour. While the opposite results in the local safe set not properly conveying the *safety* aspect. The candidate terminal safe set c_t is calculated by propagating the terminal state $s_{t+N|t}$ by one time-step given the velocity $v_{x_{t+N|t}}$ at that point. A minimum and maximum distance from c_t to the vehicle position $\mathbf{x}_t$ have been introduced for it aids the solver converge. With a similar goal, the candidate terminal state cannot correspond to a point that precedes the previous sampling time's candidate, *i.e.* $c_t \geq c_{t-1}$, such that track progress is encouraged.

The coordinate conversion (Line 5) to the Frenet frame resorts to a library[6] that fits the track centerline with splines developed by Liniger et al. [71]. This library enables finding the track progress s and lateral deviation e_y from the centerline given a location (x, y) and conversely.

The MPC's dynamic bicycle model parameters can be found in Table 4.4.

Parameter	Name	Value	SI Units
m	Mass	250	kg
I_z	Moment of Inertia about the z-axis	80	kg m
l_F	Distance from CG to front axle	0.832	m
l_R	Distance from CG to rear axle	0.708	m
C_d	Aerodynamic drag coefficient	1.2	-
A_f	Frontal area	1.18	m^2
GR	Gear ratio	15.74	-
r_{wheel}	Wheel radius	0.23	m
C_r	Rolling resistance coefficient	0.092	-
B_a	Tire stiffness factor	10	-
C_a	Tire shape factor	138	-
D_a	Tire peak lateral force	1500	N
ρ	Air density	1.18	$kg\,m^{-3}$
g	Acceleration of gravity	9.81	$m\,s^{-2}$

Table 4.4: MPC Model Vehicle Parameters

A self-developed C++ library has been developed for the system identification procedure - Line 9 - where the LMS problem is solved using Eigen's solution of the QR decomposition with column pivoting[7]. This process is repeated for each of the three dynamic quantities and for the $N-1$ models, *i.e.* at each step of the prediction horizon a different model is used. For each of these $N-1$ models, $\mathbf{b}$ and $\mathbf{X}$ is built of pre-collected datapoints from a larger dataset of N_T^{SS} points by finding those that minimize the norm distance of the difference vector between the feature points in the dataset and the predicted state given by the previous sampling time's shifted solution.

[6] https://github.com/alexliniger/MPCC/
[7] https://eigen.tuxfamily.org/dox/group__LeastSquares.html

4.3.1 Optimization Solver

The optimization problem in Equation (3.6) is solved in FORCESPRO. FORCESPRO is a numerical commercial software designed by Embotech AG specifically for the purpose of fast, embedded optimization [102, 103]. FORCESPRO enables users to generate tailor-made solvers from a high-level mathematical description of an optimization problem. Simultaneously, the solvers generated have a very small code size that can be embedded on many hardware platforms.

The code generation engine in FORCESPRO extracts the structure in the optimization problem and automatically synthesizes a custom C code optimization solver. Hence, it is an appropriate application for solving MPCs in real-time.

The workflow for solving MPCs involves three types of licenses. First, an *Engineering License* is used for specifying the structure of the optimization problem and generating the solver. Then, a *Software Testing License* is installed on a desktop PC with Ubuntu. This way, one may iterate the solver structure while testing with the rest of the pipeline described in Section 4.2, using the FSSIM simulator described in Section 4.1. Finally, an *Hardware Testing License* is installed on the vehicle's PU. This license enables controlling the physical platform.

Using the Matlab High-Level Interface[8], the optimization problem in Equation (3.6), a Nonlinear Programming optimization problem, is solved using the FORCESPRO NLP for non-convex finite-time nonlinear optimal control problems with horizon N of the form:

$$\min \left[\sum_{k=1}^{N-1} f_k(z_k, p_k) \right] \qquad \text{(separable objective)} \tag{4.9a}$$

$s.t.$

$$z_1(\mathcal{I}) = z_{\text{init}} \qquad \text{(initial equality)} \tag{4.9b}$$

$$E_K z_{k+1} = c_k(z_k, p_k) \qquad \text{(inter-stage equality)} \tag{4.9c}$$

$$z_N(\mathcal{N}) = z_{\text{final}} \qquad \text{(final equality)} \tag{4.9d}$$

$$\underline{z}_k \leq z \leq \bar{z}_k \qquad \text{(upper-lower bounds)} \tag{4.9e}$$

$$\underline{h}_k \leq h_k(z_k, p_k) \leq \bar{h}_k \qquad \text{(nonlinear constraints)} \tag{4.9f}$$

for $k = 1, \ldots, N$, where $z_k \in \mathbb{R}^{n_k}$ are the optimization variables, for example a collection of inputs, states or outputs in an MPC problem; $p_k \in \mathbb{R}^{l_k}$ are the real-time parameters; the functions $f_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}$ are stage costs; the functions $c_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}$ represent (potentially nonlinear) equality constraints, such as state transition function; the matrices E_k are used to couple the variables from the $(k+1)$-th stage to those of stage k through function c_k; and the functions $h_k : \mathbb{R}^{n_k} \times \mathbb{R}^{l_k} \to \mathbb{R}^{m_k}$ are used to express potentially nonlinear, non-convex inequality constraints. The index sets $\mathcal{I}$ and $\mathcal{F}$ are used to determine which variables are fixed to initial and final values, respectively. The initial and final values z_{init} and z_{final} can also be changed in real-time.

[8] `https://forces.embotech.com/Documentation/high_level_interface/index.html`

Four different solvers with a very similar formulation have been developed. One for each LMPC architecture (Line 11 of Algorithm 1) and with each of those depending on the model used (Line 9 - Section 3.1.1). Tables 4.5 and 4.6 specify the number of components on each line of Equation (4.9). $nvar_k$ corresponds to the number of optimization variables at each stage k, *i.e.* the size of $\mathbf{z}_k^{opt}$. neq_k and nh_k are the number of equalities, or inter-stage transitions, and inequalities, Equations (4.9c) and (4.9f), respectively. Finally, in Table 4.6, P and E correspond to the pre-computed and embedded model error prediction architectures. $npar_k$ is the number of real-time parameters, *i.e.* the size of $\mathbf{p}_k$.

Parameter	Value
$nvar_1$	12
$nvar_{2:N-1}$	18
$nvar_N$	14 + $N_P^{SS} \times N_L^{SS}$
$neq_{1:N-2}$	12
neq_{N-1}	10
nh_1	1
$nh_{2:N-1}$	7
nh_N	6

Table 4.5: Solver Dimensions

$$\mathbf{z}_1^{opt} = [P_1,\ \delta_1, s_1,\ x_1,\ y_1,\ \psi_1,\ v_{x_1},\ v_{y_1},\ r_1,\ s_{t-1}, 0, 0] \tag{4.10a}$$

$$\begin{aligned}\mathbf{z}_{k=2:N-1} = [&P_k,\ \delta_k,\ \epsilon_{lane_k},\ \epsilon_{v_k},\ \epsilon_{el_k},\ s_k,\ x_k,\ y_k,\ \psi_k,\ v_{x_k},\ v_{y_k},\ r_k,\\ &P_{k-1},\ \delta_{k-1},\ v_{x_{k-1}},\ v_{y_{k-1}},\ r_{k-1},\ s_{k-1}]\end{aligned} \tag{4.10b}$$

$$\begin{aligned}\mathbf{z}_N = [&\epsilon_{lane_N},\ \epsilon_{v_N},\ \epsilon_{el_N},\ s_N,\ \alpha_i,\ x_N,\ y_N,\ \psi_N,\ v_{x_N},\ v_{y_N},\ r_N,\\ &v_{x_{N-1}},\ v_{y_{N-1}},\ r_{N-1},\ s_{N-1}]\end{aligned} \tag{4.10c}$$

The initial index set is $\mathcal{I} = [1:2,\ 4:12]$ such that the optimization starts with the new state measurements and the applied control input computed at the previous sampling time, Equations (3.6q) and (3.6r). Therefore, s_1 is the only free optimization variable. The last two elements are artificial variables that exist only because $nvar_1$ cannot be smaller than neq_1. They have been set to 0. s_{t-1} is the vehicle's track progress measure at the previous sampling time which is used to enforce Equation (3.6e). Equivalently for s_{k-1}, but instead the constraint in Equation (3.6f) is applied between consecutive stages. One must use inter-stage equalities to have access to variables from a previous stage. Thus, similarly to s_{k-1}, the control inputs and dynamic state variables are transmitted to the subsequent stage to apply the derivative costs. Hence, the number of inter-stage equalities neq_k is 6 (state transition in Equation (3.6b)) plus the number of variables from the previous stage. Embotech recommends organizing $\mathbf{z}_k$ such that the variables with inter-stage equalities are placed at end of the vector and the remaining variables are considered inputs.

The only inequality at the initial stage is Equation (3.6e). At the intermediate stages, the inequalities correspond to Equations (3.6f)–(3.6j) and right and left-hand side of Equation (3.6k). For the final stage, the inequalities correspond to Equations (3.6f) and (3.6i)–(3.6k) and the sum of the convex hull

coefficients in Equation (3.6p) which formulated as a soft equality:

$$1-\epsilon \leq \sum_{i=1}^{N_l^{SS}\times N_p^{SS}} \alpha_i \leq 1+\epsilon \tag{4.11}$$

where ϵ is an arbitrarily small number. There are no fixed variables on the final stage, *i.e.* $\mathcal{F}$ is an empty set. Equations (4.12) and (4.13) contain the real-time parameters vector which are fed to the solver when using the bicycle model or the LMS model, respectively. The parameters for the final stage are the same for any formulation and are given given by Equation (4.14).

$$\mathbf{p}_1 = \left[Q_{lag},\ T_{max},\ g_t^{v_x},\ g_t^{v_y},\ g_t^{\dot{\psi}}\right] \tag{4.12a}$$

$$\mathbf{p}_k^P = \left[Q_{lag},\ Q_{v_y},\ \mathbf{Q}_{deriv},\ \mathbf{R}_{deriv},\ \mathbf{Q}_{lane},\ \mathbf{Q}_{v_{ub}},\ \mathbf{Q}_{el},\ e_y^{max}(s),\ v_{x_{ub}},\ T_{max},\ g_t^{v_x},\ g_t^{v_y},\ g_t^{\dot{\psi}}\right] \tag{4.12b}$$

$$\begin{aligned}\mathbf{p}_k^E = \big[&Q_{lag},\ Q_{v_y},\ \mathbf{Q}_{deriv},\ \mathbf{R}_{deriv},\ \mathbf{Q}_{lane},\ \mathbf{Q}_{v_{ub}},\ \mathbf{Q}_{el},\ e_y^{max}(s),\ v_{x_{ub}},\ \mathbf{Z}_{ind},\ \mathbf{i}_{e_{v_x}},\ \mathbf{i}_{e_{v_y}},\ \mathbf{i}_{e_r},\\ &\sigma_n^{v_x},\ \sigma_f^{v_x},\ l^{v_x},\ \sigma_n^{v_y},\ \sigma_f^{v_y},\ l^{v_y},\ \sigma_n^r,\ \sigma_f^r,\ l^r,\ ub_{e_{v_x}},\ lb_{e_{v_x}},\ b_{e_{v_y}},\ b_{e_r},\ T_{max}\big]\end{aligned} \tag{4.12c}$$

$$\mathbf{p}_1 = \left[Q_{lag},\ \boldsymbol{\theta}_x,\ \boldsymbol{\theta}_y,\ \boldsymbol{\theta}_\psi,\ g_t^{v_x},\ g_t^{v_y},\ g_t^{\dot{\psi}}\right] \tag{4.13a}$$

$$\mathbf{p}_k^P = \left[Q_{lag},\ Q_{v_y},\ \mathbf{Q}_{deriv},\ \mathbf{R}_{deriv},\ \mathbf{Q}_{lane},\ \mathbf{Q}_{v_{ub}},\ \mathbf{Q}_{el},\ e_y^{max}(s),\ v_{x_{ub}},\ \boldsymbol{\theta}_x,\ \boldsymbol{\theta}_y,\ \boldsymbol{\theta}_\psi,\ g_t^{v_x},\ g_t^{v_y},\ g_t^{\dot{\psi}}\right] \tag{4.13b}$$

$$\begin{aligned}\mathbf{p}_k^E = \big[&Q_{lag},\ Q_{v_y},\ \mathbf{Q}_{deriv},\ \mathbf{R}_{deriv},\ \mathbf{Q}_{lane},\ \mathbf{Q}_{v_{ub}},\ \mathbf{Q}_{el},\ e_y^{max}(s),\ v_{x_{ub}},\ \mathbf{Z}_{ind},\ \mathbf{i}_{e_{v_x}},\ \mathbf{i}_{e_{v_y}},\ \mathbf{i}_{e_r},\\ &\sigma_n^{v_x},\ \sigma_f^{v_x},\ l^{v_x},\ \sigma_n^{v_y},\ \sigma_f^{v_y},\ l^{v_y},\ \sigma_n^r,\ \sigma_f^r,\ l^r,\ ub_{e_{v_x}},\ lb_{e_{v_x}},\ b_{e_{v_y}},\ b_{e_r},\ \boldsymbol{\theta}_x,\ \boldsymbol{\theta}_y,\ \boldsymbol{\theta}_\psi\big]\end{aligned} \tag{4.13c}$$

$$\mathbf{p}_N = \left[Q_{lag},\ \mathbf{Q}_{deriv},\ \mathbf{Q}_{lane},\ \mathbf{Q}_{v_{ub}},\ \mathbf{Q}_{el},\ Q_{term\ cost},\ \mathbf{Q}_{slack},\ e_y^{max}(s),\ v_{x_{ub}},\ \mathbf{Q}^j(c_t),\ \mathbf{D}^j(c_t)\right] \tag{4.14}$$

Parameter	LMS-P	LMS-E	Bic-P	Bic-E
$npar_1$	17	17	5	5
$npar_{2:N-1}$	31	41 + 7×m	19	29 + 7×m
$npar_N$		19 + 7 × N_P^{SS} × N_L^{SS}		

Table 4.6: Solver Real-Time Parameters Dimensions

The bounds on the optimization variables, Equations (3.6c), (3.6d) and (3.6l)–(3.6o), are provided in real-time when calling the solver. Before, one just needs to define the corresponding index set $\mathcal{B}$ for each stage. Hence, the other variables are unbounded.

The computations of the track local coordinates approximation in Equation (3.9) evaluate the third-order spline polynomials which are divided in N_s intervals. The centerline path given by a set of coordinates (x, y, s) that compose a mesh created by the Matlab function `spline`. In addition to the path partial derivatives, a Matlab symbolic function is generated using the Symbolic Math Toolbox. This function is evaluated within the solver to compute e_l and e_y. This scheme has been implemented by a fellow master thesis student - Gabriel - who worked in the same research group.

FORCESPRO resorts to an automatic differentiation tool to generate C code from Matlab code.

We have used CasADi v3.5.1. CasADi [104] is an open-source tool for nonlinear optimization and algorithmic differentiation. It facilitates rapid — yet efficient — implementation of different methods for numerical optimal control, namely NMPC.

In order to abide by the application's real-time requirement, one exploits the solver's timeout option. The timeout works by checking the execution time of each iteration of the solver and making an estimate for next iterations as the product of the currently slowest iteration and a coefficient used to make the estimate more conservative or forgiving. Both this coefficient and the total solver processing time budget are real-time parameters. Frequently, the solver cannot reach the optimal solution within the available time. Nevertheless, unless the solution is infeasible - which would yield a different solver output flag, we have empirically concluded that the predicted trajectory shows no noticeable differences to the optimal solution. If the solution is in fact infeasible or another issue with the optimization solution is raised, the previous sampling time shifted solution is used.

The compiler optimization level is set to its maximum value of 3 so that the code generated is as efficient - thus fast - as possible. I have also enabled the option for computation on multiple CPU cores in which the workload is split along the horizon to multiple cores.

4.3.2 Gaussian Processes Regression

Gaussian Processes Regression Inference

The computations associated with GPR model error prediction described in Section 3.2 resort to the C++ open-source `albatross`[9] library developed by Swift Navigation. We have decided against developing a custom C++ implementation for GPR because only modelling with this technique falls within the scope of this thesis. Therefore, provided there exists a free of charge library that seamlessly integrates with the C++ ROS node one would choose such option. Furthermore, an efficient computational procedure requires several algebraic optimizations which would require extensive workload.

`albatross` is a framework for statistical modelling in C++, with a focus on Gaussian Processes. It enables modelling with a GP using composable covariance functions and custom data types in a way that accommodates the research phase of development (rapid model iteration, evaluation, comparison, and tuning) but also runs fast in a production environment. As an header only library, it eases the integration process with the existing software stack.

Recall from Section 3.3 that there are two variants with respect to the computation of model error prediction. In the pre-computed way, the two equations aforementioned are computed in full by the `albatross` library. On the other hand, in the embedded prediction form, only the information vector $\mathbf{i}^d$ is computed by this library. In this case, the covariance function between the inducing points and the test case and the model error prediction are calculated in the model evaluation segment at each solver's iteration - Section 4.3.1. Here, iteration refers to the numerical optimization algorithm and ought not to be confused with iteration j of the LMPC algorithm.

[9] `https://swiftnav-albatross.readthedocs.io/en/latest/index.html`

Hyperparameter Tuning

The hyperparameters are tuned offline based on pre-collected data. There are two main reasons for which this is not done online. First, this optimization is not real-time feasible. Second, it is assumed the general trend of the model error remains constant throughout the vehicle operation.

The hyperparameter estimation procedure resorts to the Matlab function `fitrgp`[10]. For a particular kernel function and sparse approximation, given the original dataset of size n, this function outputs the hyperparameters and the active set of size m that maximize the marginal logarithmic likelihood in Equation (2.23). The active set choice is of particular relevance so as to maximize the information provided by the subset of the original dataset, *i.e.* how well the subset conveys the general character of the full dataset. `fitrgp` contains four active set selection methods: *i)* random selection; *ii)* sparse greedy matrix approximation; *iii)* differential-entropy based selection; and *iv)* Subset of Regressors (SoR) log likelihood-based selection. The differential-entropy based selection yielded better results on a 10-Folds CV scheme. Accordingly, it has been used for the remaining tuning.

This function may use either an exact GP or one of the following three sparse approximations for estimating the model parameters: *i)* SoD; *ii)* SoR; or *iii)* FIC. Quiñonero-Candela et al. [81] argue that for a given sparse approximation it makes most sense to both optimize the hyperparameters and make predictions under the same approximation. The hyperparameters that are optimal for the full GP model, if one were able to obtain them, may also well be very different from those optimal for a specific sparse approximation. We have decided to follow such advice. We use the active set of the SoD approximation as the training set for the FITC approximation, *i.e.* $n_{FITC} = m_{SoD}$.

`fitrgp` also allows for broader model optimization, *e.g.* of the kernel function or whether or not to standardize the training data. It turns to Bayesian optimization[11] to minimize $\log(1 + \text{CV Loss})$.

[10] `https://www.mathworks.com/help/stats/fitrgp.html`
[11] `https://www.mathworks.com/help/stats/bayesian-optimization-algorithm.html`

4.3.3 Controller Parameters

Parameter	FSG Default	FSG Aggressive	FSI
$Q_{deriv}^{v_x}$	12	3	12
$Q_{deriv}^{v_y}$	0.3	0.3	0.3
Q_{deriv}^{r}	0.7	0.7	0.7
R_{deriv}^{P}	180	100	180
R_{deriv}^{δ}	180	100	180
Q_{lane}^{lin}	100	100	200
Q_{lane}^{quad}	10	10	10
$Q_{v_{ub}}^{lin}$	100	100	100
$Q_{v_{ub}}^{quad}$	10	10	10
Q_{el}^{lin}	100	100	100
Q_{el}^{quad}	5	5	5
Q_{v_y}	10	2	10
Q_{lag}	300	300	300
$Q_{term\ cost}$	65	150	65
Q_{slack}^{x}	10	10	10
Q_{slack}^{y}	10	10	10
Q_{slack}^{ψ}	0	0	00
$Q_{slack}^{v_x}$	15	15	15
$Q_{slack}^{v_y}$	1	1	1
Q_{slack}^{r}	1	1	1
N_l^{SS}	4	4	4
N_p^{SS}	10	10	10
P_{lb}	-1	-1	-1
P_{ub}	1	1	1
δ_b	0.47	0.47	0.47
$\Delta_{s_{max}}$	4	4	4
$\Delta P_{ub} = -\Delta P_{ub}$	0.25	0.25	0.25
$\Delta\delta_{ub} = -\Delta\delta_{lb}$	0.25	0.25	0.25
$v_{x_{max}}$	31	31	31
$v_{y_{max}}$	31	31	31
$v_{x_{ub}}$	30	30	30
T_{max}	21	21	21
FORCESPRO CPU Cores	3	3	3

Table 4.7: Controller Parameters

Chapter 5

Results

In this chapter, we will show evidence of the iterative improvement of the LMPC architecture in the FSD simulation environment in two different tracks. Further, we will demonstrate that while learning of the terminal components does improve performance it cannot alone reach the full performance envelope of an autonomous racing prototype unless the severe model mismatch inherent to simplified vehicle dynamics modelling is reduced. To that end, this chapter starts with an analysis of the machine learning technique tuning process and its model error prediction results.

In Section 5.1, we show results for both sparse approximations by varying the size of their latent variables m. There, we aim to study the influence of these parameters in the ability to predict the nominal model error. We test both sparse models in an offline and online learning fashion. In the former, the training dataset is not updated online with current measurements while in the latter it is. We also analyse how the computational cost of GPR prediction evolves for these strategies. The resulting data has been collected in closed-loop in the FSG track. Thus, the model error fitting ability influences the controller behaviour. The results shown there correspond to the average over 10 laps. The parameters used correspond to the second column of Table 4.7.

We use the compound average error to evaluate the overall model learning performance: $\overline{||e_{nom}||}$ is the average 2-norm error of the nominal model, *i.e.* $||e_{nom}|| = ||\mathbf{B}_d^{\dagger}\big(\mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k)\big)||$; and, $\overline{||e_{GP}||}$ is the corresponding average error of the corrected dynamics, *i.e.* $||e_{GP}|| = ||\mathbf{B}_d^{\dagger}\big(\mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{g}(\mathbf{z}_k)\big)||$. We also analyse the average error of each individual GP model. For the SoD computational cost analysis, we measure $\overline{T}_{inf}$ and $\overline{T}_{trn}$ - the inference and training time, respectively. $\overline{T}_{GP}$ is the time spent on GP computations which for the FITC method corresponds to inference and training.

In Section 5.2, we use the best performing models found in Section 5.1 to evaluate the complete LMPC architecture. We evaluate its performance with the lap times over the 10-lap trackdrive event. We show results for two tracks - FSG and FSI - and change the controller parameters to achieve a more aggressive controller. Finally, we also compare the FITC pre-computed and embedded versions of the architecture - Algorithm 1. The results showed throughout this chapter make use of the simulator's pose ground truth rather than the SLAM's pose estimation.

5.1 Model Learning

For the Subset of Data approximation, we use the tuning scheme described in Section 4.3.2 to find the active set of different sizes m_{SoD} for each model from a dictionary of $n_{SoD} = 43329$ collected over 10 different runs. Over those runs, we have changed the controller parameters in order to collect data from the whole performance envelope by yielding more conservative or aggressive controllers.

In Table 5.1, we show the model error prediction fitness for the SoD approximation. The dashed line separates results of offline (above) and online (below) schemes. Recall, in the offline method where the training points are not updated online the computation of $\left(\mathbf{K}^d_{\mathbf{ZZ}} + \mathbf{I}\sigma_d^2\right)^{-1}\mathbf{y}^d$ is performed before-hand and therefore incurs no extra computational cost.

m_{SoD}	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$	$\overline{T}_{inf}$ [ms]	$\overline{T}_{trn}$ [ms]
200	0.25	0.09	0.5	-
300	0.26	0.08	0.8	-
400	0.26	0.08	1.0	-
500	0.27	0.09	1.3	-
600	0.25	0.07	1.5	-
200	0.27	0.09	0.5	4.9
300	0.27	0.07	0.7	13.6

Table 5.1: SoD Active Set Size Analysis

Table 5.1 shows that the model learning procedure reduces model mismatch, *i.e.* difference between $\overline{||e_{nom}||}$ and $\overline{||e_{GP}||}$, by at least 60 %. There is a positive correlation between m_{SoD} and model error fitting ability. For instance, in the offline fashion, the 2-norm average error reduction is 63.9% and 70.7% with $m_{SoD} = 200$ and $m_{SoD} = 600$, respectively. The computation cost evolves linearly with m but within this range takes acceptable values given the current node rate of 20 Hz. Arguably, one could further increase m_{SoD} but likely with negligible model learning improvements.

Instead, one should aim to adapt the training data online as that would enable adapting to changing conditions or even just collecting data from dynamic maneuvers not included in the original dictionary. The last two lines of Table 5.1 corroborate this hypothesis. With a dictionary of under 300 datapoints, the model error reduction is 65.4% and 75.7% for $m_{SoD} = 200$ and $m_{SoD} = 300$, respectively. In particular, it enables more aggressive maneuvers towards the end of the event while keeping the corrected dynamics error relatively low. See Figures 5.1–5.3 which depict the model error fitting for the three models currently in discussion, in the first and last lap. The model error fitting ability is very similar in the first few laps. But, as the LMPC architecture pushes the vehicle to the limits of friction, the offline method remains more conservative. This is evidenced by the last lap data where in the offline version with $m_{SoD} = 600$ the nominal model average norm error increases to 0.28, while the online versions increase to around 0.32, from around 0.21 in the first few laps. The model learning is then able to reduce the corrected dynamics model error to 0.09 (offline - $m_{SoD} = 600$), 0.13 (online - $m_{SoD} = 200$) and 0.08 (online - $m_{SoD} = 300$) which amounts to a reduction of 69, 59 and 76%, respectively.

It has been demonstrated the importance of online learning. The model with $m_{SoD} = 300$ performs significantly better than the model with $m_{SoD} = 200$. While the average node processing time is well within the limits, it too often breaks the real-time requirement. Thus, the latter model is used for benchmark later. I argue that the performance deterioration from this reduction comes mainly from the naïve dictionary update process. When online learning is active every new measurement and its corresponding model error is added to the dictionary by removing the oldest point. For the FSG track, with lap times around $17\,\mathrm{s}$ and a node rate of $20\,\mathrm{Hz}$ it would be required a dictionary size of 340 points to cover the whole track. Note that I am not claiming there is necessarily a spatial correlation to model error that would require data from the whole track. Nevertheless, there might exist parts of the track that result in model error different than the GPs trend or specific dynamic maneuvers not repeated.

Table 5.2 shows the prediction behaviour of the individual GP models. Note that the corrected dynamics for the yaw rate are consistently within acceptable bounds and the model error reduction is high - frequently above 70%. Models struggle mostly with learning the lateral velocity dynamics which is verified with a reduction of around 50% and only 33% for the offline model with $m_{SoD} = 200$.

	$v_x\ [\mathrm{m\,s^{-1}}]$		$v_y\ [\mathrm{m\,s^{-1}}]$		$r\ [\mathrm{rad\,s^{-1}}]$	
m_{SoD}	$\bar{e}_{nom}$	$\bar{e}_{GP}$	$\bar{e}_{nom}$	$\bar{e}_{GP}$	$\bar{e}_{nom}$	$\bar{e}_{GP}$
200	0.17	0.06	0.11	0.07	0.19	0.05
300	0.17	0.05	0.11	0.07	0.21	0.05
400	0.17	0.07	0.12	0.06	0.21	0.06
500	0.18	0.05	0.12	0.06	0.22	0.07
600	0.17	0.05	0.11	0.06	0.20	0.05
200	0.18	0.06	0.13	0.06	0.21	0.10
300	0.18	0.04	0.12	0.06	0.21	0.05

Table 5.2: SoD Active Set Size per Model Analysis -

As explained in Section 2.3, the FITC sparse approximation is a natural candidate to enable online learning. In Table 5.3, we show average error similarly to Table 5.1. There are three groups separated by the horizontal dashed lines. In order, we first show the results for offline and online learning with $n_{FITC} = 300$. Subsequently, we extend those with data from $n_{FITC} = 400$. For each of these, we test three different inducing points strategy along the prediction horizon which is equivalent to changing m_{FITC}.

We show that there is no further computational cost incurred by choosing the online framework which is inherent to the proposed architecture where the latent variables are changed at each sampling time. Although in a comparable order of magnitude, the offline version exhibits better model error fitting ability. I once again argue that this is due to the dictionary update procedure. Therefore, we have increased the training dataset to $n_{FITC} = 400$. The best performing model with $m_{FITC} = 10$ yields model learning performance comparable to that of the SoD approximation. The corrected dynamics average error is 0.12 slightly above of the SoD benchmark value of 0.09.

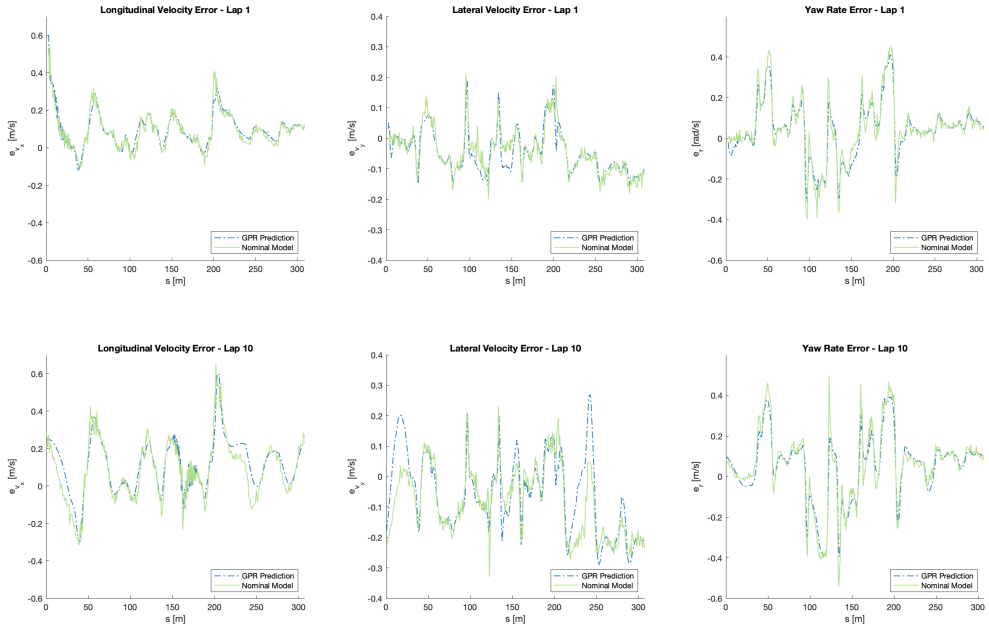


Figure 5.1: Model Error Fitting Ability - Offline SoD with $m_{SoD} = 600$ (Laps 1 and 10)

m_{FITC}	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$	$\overline{T}_{GP}$ [ms]
5	0.27	0.14	2.6
10	0.26	0.15	3.0
20	0.26	0.15	4.1
5	0.28	0.15	2.6
10	0.26	0.17	3.0
20	0.26	0.15	4.1
5	0.28	0.15	3.3
10	0.27	0.12	3.8
20	0.27	0.13	5.2

Table 5.3: FITC Inducing Points Strategy Analysis

In Table 5.4, we show the performance of each model for the FITC approximation method. Again, one can conclude that the yaw rate error dynamics are well modelled. The error reduction is generally above 70 %, reaching 75 % reduction in the best performing model. For this model, the remaining error reductions are around 40 % which is at least 10 % less than the best SoD models. However, the nominal model error average are comparable with the SoD's. Thus, proving the controller also pushes for aggressive behaviour. This could imply unsafe driving due to model mismatch. Finally, see Figure 5.4 for a visual comparison with the best performing SoD models.

Figure 5.2: Model Error Fitting Ability - Online SoD with $m_{SoD} = 200$ (Laps 1 and 10)

	v_x [m s^{-1}]		v_y [m s^{-1}]		r [rad s^{-1}]	
m_{FITC}	$\bar{e}_{nom}$	$\bar{e}_{GP}$	$\bar{e}_{nom}$	$\bar{e}_{GP}$	$\bar{e}_{nom}$	$\bar{e}_{GP}$
5	0.18	0.12	0.12	0.09	0.22	0.08
10	0.18	0.15	0.12	0.08	0.21	0.05
20	0.18	0.15	0.12	0.06	0.21	0.06
5	0.18	0.11	0.13	0.12	0.22	0.07
10	0.18	0.17	0.12	0.08	0.20	0.05
20	0.17	0.16	0.12	0.06	0.21	0.06
5	0.18	0.14	0.13	0.07	0.22	0.08
10	0.18	0.10	0.12	0.07	0.21	0.05
20	0.18	0.13	0.12	0.04	0.21	0.07

Table 5.4: FITC Inducing Points Strategy per Model Analysis

5.2 Learning-based Model Predictive Control

In Table 5.5, we show the lap times along the 10-lap trackdrive event and average model error for the FSG track. The initial safe set was collected using the simple controllers described in Section 4.2. It is composed of four laps with lap times of around $28.8\,\mathrm{s}$. The controllers herein have a prediction horizon of $N = 20$ which corresponds to a look-ahead time of $1\,\mathrm{s}$, until stated otherwise. The LMPC results without model learning prove the iterative improvement character of the architecture. The first

Figure 5.3: Model Error Fitting Ability - Online SoD with $m_{SoD} = 300$ (Laps 1 and 10)

lap is immediately 33% faster compared to the path-following controller. Equivalently, the last lap is 39% faster. Furthermore, the last lap corresponds to a 10% improvement compared to the first LMPC lap.

Figure 5.5 shows the FSG trackdrive trajectories. The finish line is at the origin and the vehicle runs clockwise. It can be seen that the LMPC exploits the track layout to improve performance measured by lap time. Nevertheless, the third column of Table 5.5 exhibits severe model mismatch which causes the vehicle to break the track constraint. See, for instance, the exit of the hairpin or the first corner where the trajectory is on top of the track boundary which entails a cone was hit since the trajectory corresponds to the center of mass. The car starts at the origin. The hairpin is the sharp corner on the rectangle region given by the top left corner of $(10, -65)$ and the bottom right corner of $(30, -75)$. Furthermore, the approach to the slalom segment is not optimal per empirical vehicle dynamics standards. The vehicle is braking too late which leads to a slower slalom with greater steering actuation required. The slalom segment is given by the corners $(-10, -15)$ and $(0, -45)$. In this case, I reckon it is due to a combination of a relatively short prediction horizon and model mismatch.

Let one now analyse the performance of the LMPC when the GP model learning scheme is deployed. Table 5.5 shows that the last lap is 42 and 12% faster when compared to the path-following lap and the first lap, respectively. These results correspond to the online SoD model with $m_{SoD} = 200$. Figure 5.5 shows the corresponding FSG trackdrive trajectories. It is clear the reduced model mismatch prevents the vehicle from disrespecting the track constraint. However, the slalom trajectory does not yet look optimal.

Figure 5.7 shows that after activating the model learning scheme the LMPC is able to safely increase the velocity across most track segments.

Table 5.6 exhibits the equivalent data for the FITC approximation for the best performing model:

Figure 5.4: Model Error Fitting Ability - Online FITC with $n_{FITC} = 400$ and $m_{FITC} = 10$ (Laps 1 and 10)

	LMPC		LMPC + SoD		
Lap	Time [s]	$\overline{\|\|e_{nom}\|\|}$	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$
1	19.36	0.22	18.92	0.21	0.06
2	19.33	0.21	18.75	0.21	0.07
3	19.34	0.21	18.73	0.21	0.07
4	19.33	0.22	18.71	0.21	0.07
5	17.78	0.29	17.17	0.29	0.08
6	17.45	0.30	16.85	0.31	0.11
7	17.44	0.31	16.77	0.32	0.13
8	17.51	0.29	16.84	0.32	0.11
9	17.43	0.30	16.80	0.31	0.12
10	17.43	0.30	16.96	0.32	0.13

Table 5.5: LMPC Lap Times and Model Error

online with $n_{FITC} = 400$ and $m_{FITC} = 10$. The fact that in the embedded scheme the model error prediction is computed for every control input considered prompts comparatively more aggressive behaviour on the initial laps. This is evidenced by the lap times and nominal model error. However, this scheme fails to significantly improve performance and quickly converges to a lap time of around $17.3\,\mathrm{s}$. First to last lap time reduction of 13 and 9% on the pre-computed and embedded architectures, respectively. Both schemes are able to sustain an approximately constant corrected dynamics average error of 0.13, which is low enough for fast and feasible racing.

Table 5.7 exhibits the processing timings for both architectures using the FITC approximation. The current results do not sustain as beneficial using the embedded scheme. Furthermore, despite being average values there seems to exist some leeway to increase the prediction horizon and the controller frequency.

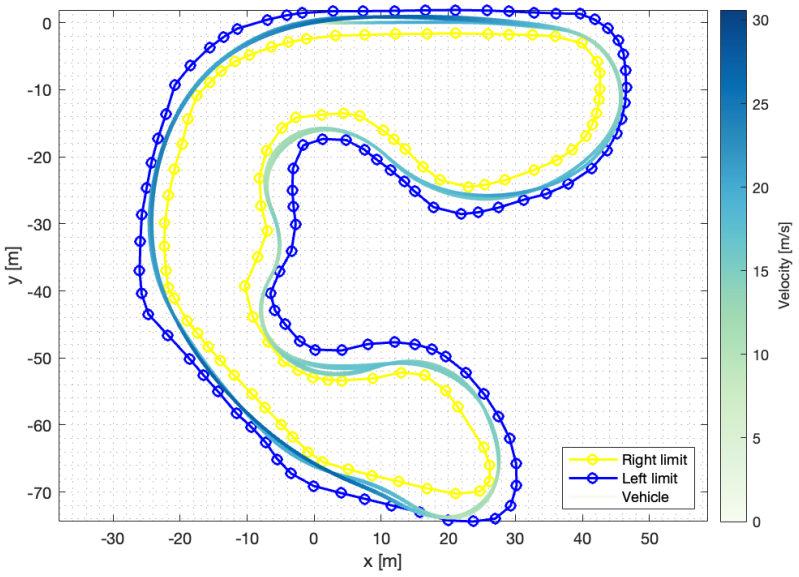


Figure 5.5: FSG Trackdrive Trajectory of LMPC without Model Learning

	LMPC + FITC-P			LMPC + FITC-E		
Lap	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$
1	19.13	0.22	0.11	18.87	0.25	0.12
2	19.03	0.21	0.10	18.63	0.25	0.11
3	18.99	0.21	0.10	18.64	0.28	0.12
4	19.16	0.22	0.10	18.61	0.24	0.11
5	17.32	0.28	0.13	17.72	0.27	0.13
6	16.87	0.30	0.13	17.43	0.29	0.14
7	16.75	0.31	0.14	17.27	0.28	0.13
8	16.71	0.31	0.13	17.12	0.32	0.13
9	16.67	0.31	0.14	17.27	0.30	0.13
10	16.68	0.31	0.13	17.27	0.30	0.13

Table 5.6: LMPC Lap Times and Model Error - FITC Approximation

Architecture	$\overline{T}_{GP}$ [ms]	$\overline{T}_{solver}$ [ms]	$\overline{T}_{LMPC}$ [ms]
FITC Bic-P	3.8	14.4	18.5
FITC Bic-E	3.5	17.1	21.1

Table 5.7: FITC Processing Time

We have subsequently tested with increasing prediction horizons. In Table 5.8, we display the lap times and modelling errors for two sets of controller gains with $N = 30$. The results on the left correspond to the parameters used thus far in this chapter. On the other hand, for the controller on the right we reduce some derivative costs and the regularization cost on v_y, and increase the Q-function associated cost to promote greater track progress at each sampling time - the corresponding costs can be found in the third column of Table 4.7.

With a longer horizon both controllers can safely navigate around the track such that the safe set

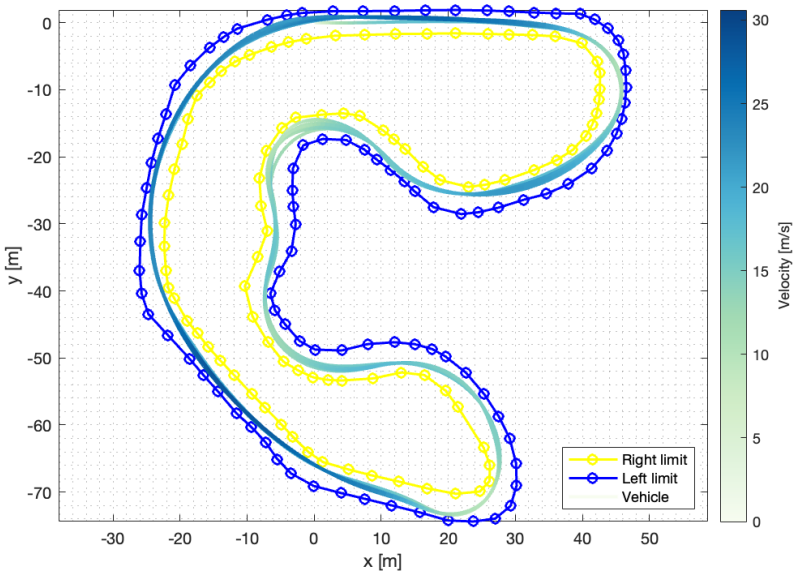


Figure 5.6: FSG Trackdrive Trajectory of LMPC with Model Learning [$N = 20$]

loses its relative importance. That is, the controller is able to predict consistently until the slowest point on a given corner. This way, the information conveyed by the safe set regarding what sort of maneuvers come after is not as valuable. The safety character referred only applies when model learning is deployed. Otherwise, the severe model mismatch hinders performance. This is substantiated by the fact that both achieve small lap times in the first few laps and quickly converge to their steady-state lap times of around 16.7 s for the default controller and 16.2 s for the aggressive controller.

	Default Parameters			Aggressive Parameters		
Lap	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$
1	17.16	0.27	0.07	16.37	0.39	0.14
2	16.92	0.26	0.07	16.18	0.37	0.15
3	16.91	0.26	0.07	16.17	0.37	0.15
4	16.88	0.26	0.07	16.13	0.37	0.15
5	16.69	0.28	0.08	16.17	0.36	0.15
6	16.66	0.28	0.09	16.21	0.37	0.15
7	16.67	0.28	0.08	16.19	0.36	0.15
8	16.68	0.28	0.08	16.14	0.37	0.16
9	16.68	0.27	0.08	16.15	0.37	0.16
10	16.69	0.27	0.08	16.11	0.36	0.15

Table 5.8: LMPC Lap Times and Model Error [$N = 30$]

Both models used the offline version of the SoD approximation with $m_{SoD} = 600$. Table 5.8 further corroborates these claims. For instance, the nominal model average error on the first lap of the default

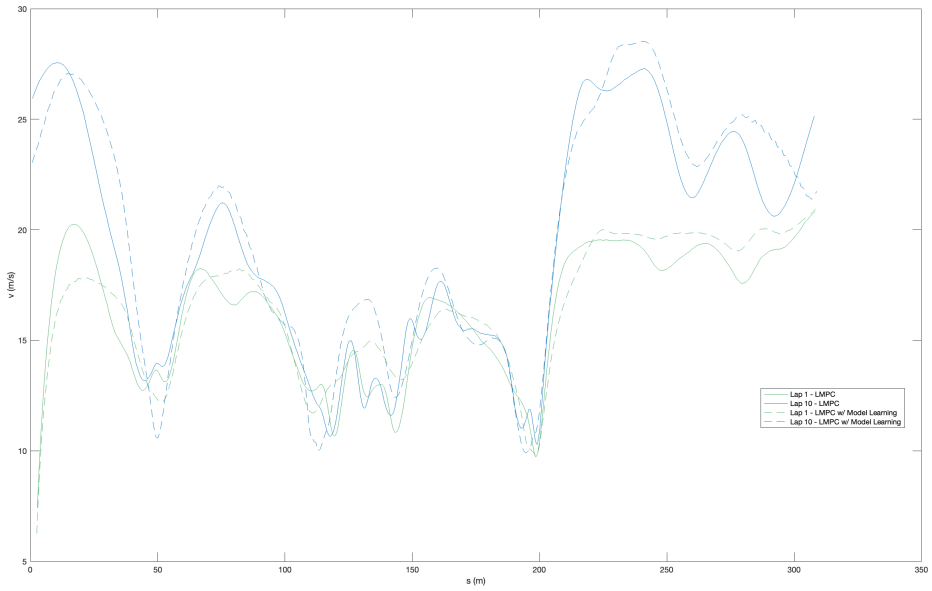


Figure 5.7: FSG Trackdrive Velocity Profile of LMPC with and without Model Learning [$N = 20$]

controller with $N = 30$ of 0.27 is substantially higher than those with $N = 20$ which is on average 0.21. The increased driving behaviour aggressiveness is verified by the larger nominal model errors of around 0.37. The model learning scheme is able to significantly reduce the model mismatch in order to enable safe aggressive racing. For the first case, the corrected dynamics model average error is around 0.08 which corresponds to a reduction of about 70%. While for the aggressive controller, the final model mismatch is on average 0.15, a reduction of 60%. The model learning scheme is able to keep an acceptable value of corrected dynamics model mismatch even in the case of offline model learning on the aggressive controller. It should be noted that such high nominal model errors data were not present in the original training dataset.

Figure 5.8 displays the trajectories of both controllers with $N = 30$. The trajectories look similar but by analysing the velocity profiles in Figure 5.9, we conclude that the aggressive controller consistently achieves greater velocities in all track segments because more aggressive acceleration and braking maneuvers are allowed.

Finally, we include results for a different track - FSI. Again, the finish line is at the origin and the vehicle runs clockwise. In Table 5.9, we show the lap times along the 10-lap trackdrive event and average model error for the FSI track. The initial safe set is composed of four laps with lap times of around $66.6\,\mathrm{s}$. This track is quite challenging due to its many sharp corners. This explains the very low-speeds at which the path-following controller was driving. Both LMPC architectures have a prediction horizon of $N = 20$. The LMPC without model learning shows a 36% first to last lap time reduction. Nevertheless, if one inspects its trackdrive trajectories in Figure 5.10 it is clear it disrespects the track limits. While it remains safe in the sense that it the trajectories are smooth and feasible, the team would be time-penalized for hitting the track-delimiting cones. Hence, one could still argue that the safe set failed to properly convey the

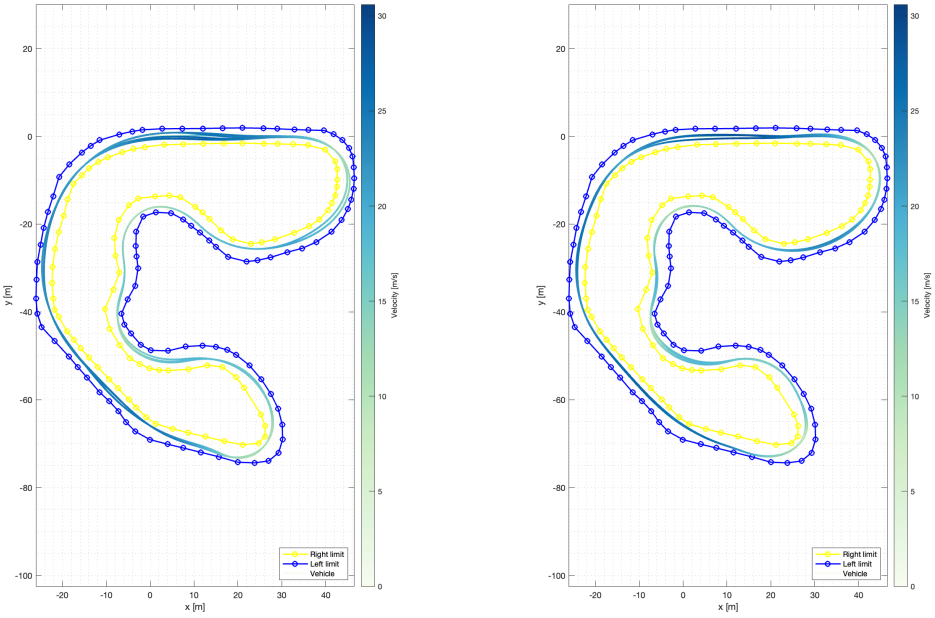


Figure 5.8: FSG Trackdrive Trajectory of LMPC with Model Learning using Default Parameters (Left) and Aggressive Parameters (Right) [$N = 30$]

safety information. However, this is not true because in fact it was a combination of the safety criterion, *i.e.* a lap or LMPC iteration is considered safe if it completes that lap - Equation (2.7), and too aggressive parameters that pushed the vehicle to the limits of the performance where the model failed to suitably explain the vehicle dynamics. This is confirmed by the average nominal model error - third column of Table 5.9.

	LMPC		LMPC + SoD		
Lap	Time [s]	$\overline{\|\|e_{nom}\|\|}$	Time [s]	$\overline{\|\|e_{nom}\|\|}$	$\overline{\|\|e_{GP}\|\|}$
1	21.37	0.10	25.76	0.08	0.09
2	16.34	0.20	18.32	0.15	0.06
3	14.16	0.31	14.52	0.27	0.07
4	13.64	0.35	13.37	0.33	0.10
5	13.62	0.35	13.25	0.34	0.11
6	13.61	0.35	13.25	0.34	0.11
7	13.60	0.35	13.23	0.34	0.11
8	13.63	0.35	13.26	0.34	0.11
9	13.62	0.35	13.25	0.34	0.11
10	13.60	0.36	13.38	0.34	0.11

Table 5.9: LMPC Lap Times and Model Error in FSI

When the model learning configuration (offline SoD with $n_{SoD} = 600$) is activated, the controller resumes to be wholly safe, *i.e.* track constraints are once again respected as can be concluded from Fig-

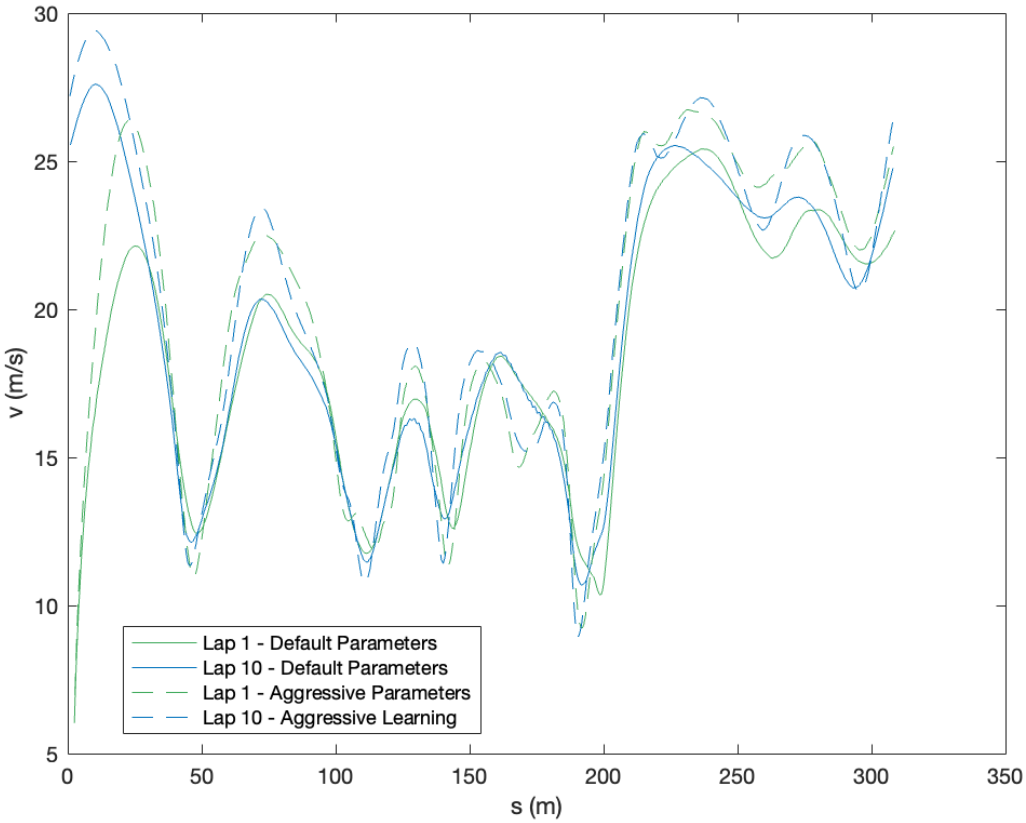


Figure 5.9: FSG Trackdrive Velocity Profile of LMPC with Model Learning [$N = 30$]

ure 5.11. This is further demonstrated by the fact that the lap times for the first two laps are considerably slower than the corresponding without model learning. That is, when the model mismatch is reduced, the safe set can *safely* drive the controller to iterative improvements of as much as 49%. The model learning scheme keeps the corrected dynamics average 2-norm error at about 0.11 which is a reduction of about 70%. Note that in the first lap the *corrected* dynamics have a greater mismatch than the nominal model. This is explained by the absence of training data at this slower race pace. It is also expected that the bicycle model accurately represents the dynamics in these conditions. Again, one would expect that a proper online learning apparatus would not face this setback. It should be noted however that the net model error was still smaller than when driving at high speeds.

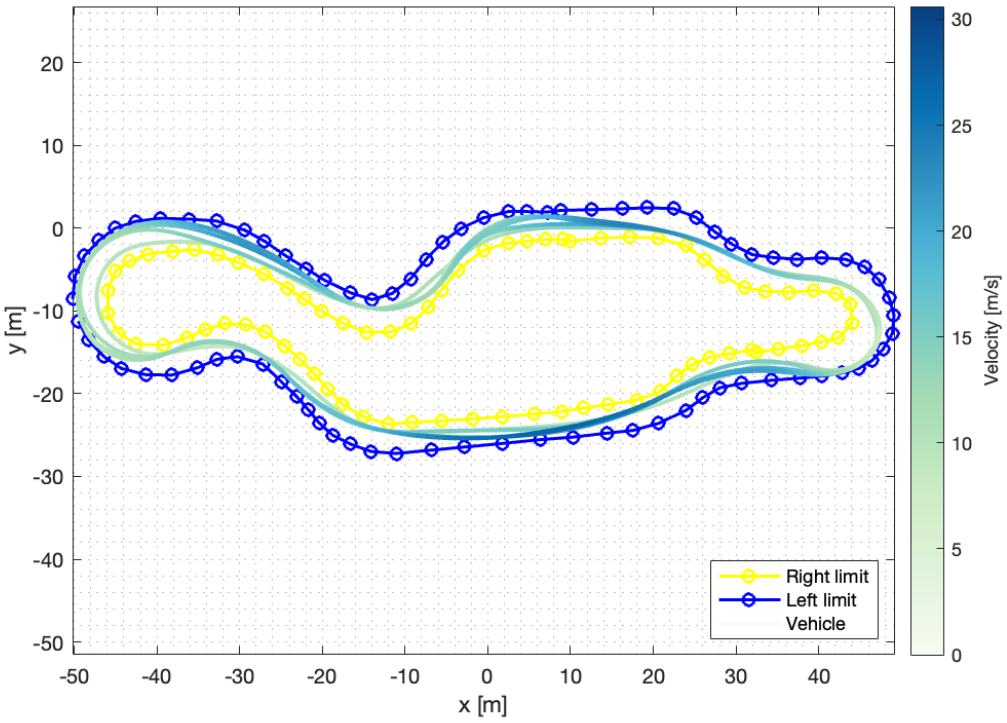


Figure 5.10: FSI Trackdrive Trajectory of LMPC without Model Learning [$N = 20$]

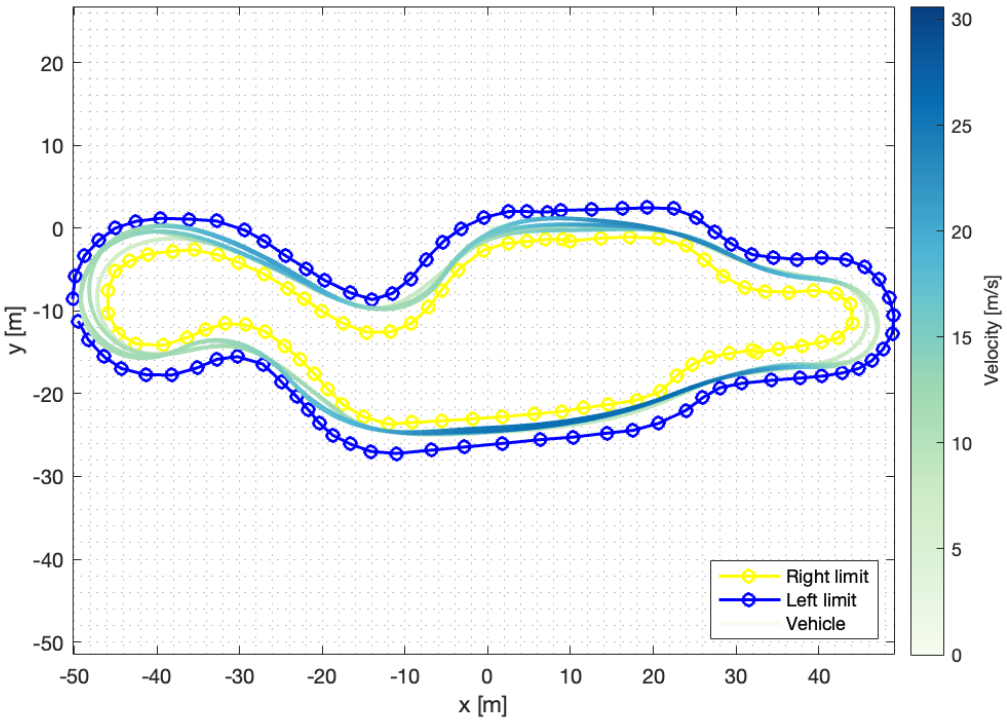


Figure 5.11: FSI Trackdrive Trajectory of LMPC with Model Learning [$N = 20$]

Chapter 6

Conclusions

6.1 Achievements

In this thesis, we have adapted the LMPC architecture, proposed by Professor Borrelli's Model Predictive Control Lab at University of California Berkeley, to the FSD context. We have implemented the framework in C++ and used a state-of-the-art optimization solver developed especially for solving MPCs in embedded platforms. Hence, we have been able to improve the real-time capability of this controller which is paramount at high-speed racing. In particular, we have been able to double the controller sampling frequency and more than double the prediction horizon length.

We have demonstrated that the LMPC using a dynamic bicycle model with an appropriately built safe set and Q-function leads to safe iterative improvements. In this racing application, the improvements were measured on a lap time basis. However, as the controller pushes the vehicle to the limits of the performance envelope the nonlinearities drastically increase model mismatch which hinders performance. This performance deterioration has been shown qualitatively by the suboptimal trajectories taken and by slightly breaking the track width constraint.

The problem of model mismatch in model-based control was by no means unknown beforehand. Indeed, it has been proven to be a significant hurdle. In order to overcome this issue, we use Gaussian Processes Regression to predict the nominal model error. The GPs are able to successfully model this error such that the corrected dynamics exhibit reasonably low model errors - model error reduction of at least 65% and by as much as 75%. This error remains about constant as the LMPC strives for faster laps in which the nominal model error increases to prohibitively high errors.

The tale of model mismatch does not end here. We have been unable to reach satisfactory results with the data-driven system identification technique employed that aims to learn a model representation by solving the least mean square problem from pre-collected data, even after deploying GPs.

We have shown how the model error fitting ability changes with varying parameters for the sparse GP approximations. For the Subset of Data approximation, we change the active set size which is selected from a vast dataset by maximizing the differential-entropy criterion. We show that increasing the size does improve prediction accuracy at a small computational cost. Nevertheless, we have concluded that

it is more important to enable online learning, *i.e.* adaptation of the training dataset with data collected on a given run. In the SoD approximation, this comes with a considerable computational cost burden due to training which only allows relatively small active sets whilst still abiding by the real-time requirement. With the naïve dictionary update technique implemented, which stores the most recent datapoints, the small active set size does not yield significant prediction improvements. I argue that a better selection procedure should improve results although the active set size would still be rather short.

The Fully Independent Training Conditional approximation is a natural candidate for online learning as the training procedure is less costly. Furthermore, since the inducing points are changed at each sampling time, which implies training the model, changing the training set does not hold further computational cost. The model error results are satisfactory but not as good as SoD's. We have also a implemented a variation where the model error predictions, *i.e.* GP's mean, is computed for each control input pair considered at each solver's iteration. This is in contrast with the previous variant where the model error predictions were pre-computed for each prediction stage based on previous sampling time's trajectory. However, the embedded architecture did not improve the performance significantly.

Subsequently, we have demonstrated that the full architecture, *i.e.* LMPC with model learning, outperforms the one without model learning in the metrics considered. We have shown data resulting from simulations in two different tracks.

Finally, I argue that with longer horizons the safe set loses some of its merits. This is because the prediction horizon covers the track farther enough such that it can react in due course to the forthcoming track segments. With a shorter horizon the safe set has been proven to be crucial.

Until the moment this thesis was concluded, the FST10d was not yet ready to test the LMPC controller. That is because the localization algorithm that estimates the car's pose was not yet showing reliable results. The results in this thesis should be replicated on the actual prototype to further substantiate the results achieved in the simulation environment.

6.2 Future Work

The greatest focus of the academia that studies learning model-based controllers has been on reducing model mismatch. Thus, one should also attempt to improve the modelling ability. For instance, by improving the nominal model accuracy which likely comes at cost of increased complexity and thus slower solver solution. However, the dynamic bicycle model is the state-of-the-art model used for autonomous racing control.

The alternative is to further explore Machine Learning techniques to fit the nominal model error. Gaussian Processes have been used with success by other research groups in similar applications and also proved encouraging in this thesis' results. First, one should start by testing the Automatic Relevance Determination (ARD) kernel which has individual length-scales for each feature dimension. It gets its name from having individual length-scales which enables identifying irrelevant inputs - those with a very large length-scale where the covariance function becomes effectively independent of that input. One could also test the squared exponential covariance function by removing the feature inputs that ARD

deems irrelevant. Furthermore, there are many other covariance function types which may improve prediction performance. Actually, by running the Bayesian optimization scheme referred in Section 4.3.2 none of the covariance functions recommended were the squared exponential, ARD or not. This study should be done for each GP rather than just evaluating the overall corrected dynamics performance. Recall we have shown worse and varying model fitting ability for the v_y error while the r error proved to be consistently accurate.

Other feature quantities should also be considered. For instance, the vehicle's pose as there might exist some parts of the track that have different adherence characteristics. Another recommended implementation is to test standardizing input data. Other ML techniques should also be considered such as the use of Bayesian Linear Regression or Neural Networks. Finally, one has already claimed that a dictionary data selection criterion that considers the information each data point conveys should improve prediction accuracy considerably.

With respect to MPC and LMPC, one should be thorough in prioritizing different soft constraint as some strongly relate with vehicle safety. In this LMPC architecture, the Q-function could be improved. The minimum-time stage cost should be augmented such that points that yield long-term benefits are more favoured. The state uncertainty measurement which is in inherent with the use of GPs should be considered in a Robust MPC framework. Finally, learning of other MPC parameters should also be targeted. For instance, one could explore methods to automatically adjust the MPCs parameters using some kind of reward function exploited by a Reinforcement Learning algorithm.

Bibliography

[1] S. Devi, P. Malarvezhi, R. Dayana, and K. Vadivukkarasi. A comprehensive survey on autonomous driving cars: A perspective view. *Wireless Personal Communications*, 114:2121–2133, 2020.

[2] J. M. Scanlon, K. D. Kusano, T. Daniel, C. Alderson, A. Ogle, and T. V. Waymo. Waymo simulated driving behavior in reconstructed fatal crashes within an autonomous vehicle operating domain, 2021. URL `https://waymo.com/safety/simulated-reconstruction`.

[3] T. J. Crayton and B. M. Meier. Autonomous vehicles: Developing a public health research agenda to frame the future of transportation policy. *Journal of Transport and Health*, 6:245–252, 9 2017.

[4] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, and P. Mahoney. Stanley: The robot that won the darpa grand challenge. *J. Field Robotics*, 23:661–692, 2 2006.

[5] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[6] M. Montemerlo and S. Thrun. *FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics (Springer Tracts in Advanced Robotics)*. Springer-Verlag, 2007.

[7] E. Yurtsever, J. Lambert, A. Carballo, and K. Takeda. A survey of autonomous driving: Common practices and emerging technologies. *IEEE Access*, 8:58443–58469, 2020.

[8] J. Wei, J. M. Snider, J. Kim, J. M. Dolan, R. Rajkumar, and B. Litkouhi. Towards a viable autonomous driving research platform. *IEEE Intelligent Vehicles Symposium, Proceedings*, pages 763–770, 2013.

[9] A. Petrovskaya and S. Thrun. Model based vehicle detection and tracking for autonomous urban driving. *Auton. Robots*, 26:123–139, 2 2009.

[10] Z.-Q. Zhao, P. Zheng, S. tao Xu, and X. Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30:3212–3232, 7 2018.

[11] B. Yang, W. Luo, and R. Urtasun. Pixor: Real-time 3d object detection from point clouds. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 7652–7660, 2 2019.

[12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:834–848, 6 2016.

[13] A. Ranga, F. Giruzzi, J. Bhanushali, E. Wirbel, P. Pérez, T.-H. Vu, and X. Perotton. Vrunet: Multi-task learning model for intent prediction of vulnerable road users. *Electronic Imaging*, 2020(16): 109–1–109–10, 1 2020.

[14] S. Bauer, Y. Alkhorshid, and G. Wanielik. Using high-definition maps for precise urban vehicle localization. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, pages 492–497, 12 2016.

[15] S. Kuutti, S. Fallah, K. Katsaros, M. Dianati, F. Mccullough, and A. Mouzakitis. A survey of the state-of-the-art localization techniques and their potentials for autonomous vehicle applications. *IEEE Internet of Things Journal*, 5:829–846, 4 2018.

[16] M. Bahram, C. Hubmann, A. Lawitzky, M. Aeberhard, and D. Wollherr. A combined model- and learning-based framework for interaction-aware maneuver prediction. *IEEE Transactions on Intelligent Transportation Systems*, 17:1–13, 7 2016.

[17] C. M. Martinez, M. Heucke, F. Y. Wang, B. Gao, and D. Cao. Driving style recognition for intelligent vehicle control and advanced driver assistance: A survey. *IEEE Transactions on Intelligent Transportation Systems*, 19:666–676, 3 2018.

[18] D. González, J. Pérez, V. Milanés, and F. Nashashibi. A review of motion planning techniques for automated vehicles. *IEEE Transactions on Intelligent Transportation Systems*, 17:1135–1145, 4 2016.

[19] F. Behbahani, K. Shiarlis, X. Chen, V. Kurin, S. Kasewa, C. Stirbu, J. Gomes, S. Paul, F. A. Oliehoek, J. Messias, and S. Whiteson. Learning from demonstration in the wild. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:775–781, 5 2019.

[20] R. Rajamani. *Vehicle Dynamics and Control*. Springer US, 2012.

[21] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*, 6 2018. URL `https://www.sae.org/standards/content/j3016_202104/`.

[22] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, T. Stahl, L. Hermansdorfer, B. Lohmann, and M. Lienkamp. What can we learn from autonomous level-5 motorsport? *9th International Munich Chassis Symposium 2018*, pages 123–146, 2019.

[23] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*, 123:77–89, 7 2020.

[24] J. Y. Goh, T. Goel, and J. C. Gerdes. Toward automated vehicle control beyond the stability limits: Drifting along a general path. *Journal of Dynamic Systems, Measurement, and Control*, 142, 2 2020.

[25] R. Coulter. Implementation of the pure pursuit path tracking algorithm, 1 1992. URL `https://www.ri.cmu.edu/publications/implementation-of-the-pure-pursuit-path-tracking-algorithm/`.

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

[27] J. T. Betts. Survey of numerical methods for trajectory optimization. *Journal of Guidance, Control, and Dynamics*, 21:193–207, 5 1998.

[28] L. E. Kavraki, M. N. Kolountzakis, and J. C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14:166–171, 1998.

[29] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30:846–894, 6 2011.

[30] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *Proceedings - IEEE International Conference on Robotics and Automation*, 1:473–479, 1999.

[31] Y. Li, Z. Littlefield, and K. E. Bekris. Sparse methods for efficient asymptotically optimal kinodynamic planning. *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 263–282, 2015.

[32] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press, 6 2017.

[33] G. V. Raffo, G. K. Gomes, J. E. Normey-Rico, C. R. Kelber, and L. B. Becker. A predictive controller for autonomous vehicle path tracking. *IEEE Transactions on Intelligent Transportation Systems*, 10:92–102, 2009.

[34] E. Kim, J. Kim, and M. Sunwoo. Model predictive control strategy for smooth path tracking of autonomous vehicles with steering actuator dynamics. *International Journal of Automotive Technology*, 15:1155–1164, 11 2014.

[35] P. Falcone, M. Tufo, F. Borrelli, J. Asgarit, and H. E. Tseng. A linear time varying model predictive control approach to the integrated vehicle dynamics control problem in autonomous systems. *Proceedings of the IEEE Conference on Decision and Control*, pages 2980–2985, 2007.

[36] C. Chen, A. Seff, A. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2722–2730, 2015.

[37] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani. Deep reinforcement learning framework for autonomous driving. *IS and T International Symposium on Electronic Imaging Science and Technology*, pages 70–76, 4 2017.

[38] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, 2018.

[39] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518:529–533, 2 2015.

[40] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. V. D. Driessche, T. Graepel, and D. Hassabis. Mastering the game of go without human knowledge. *Nature*, 550:354–359, 10 2017.

[41] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez. Deep reinforcement learning for autonomous driving: A survey. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–18, 2020.

[42] P. Wang, C. Y. Chan, and A. D. L. Fortelle. A reinforcement learning based approach for automated lane change maneuvers. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June:1379–1384, 10 2018.

[43] A. Kendall, J. Hawke, D. Janz, P. Mazur, D. Reda, J.-M. Allen, V.-D. Lam, A. Bewley, and A. Shah. Learning to drive in a day. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:8248–8254, 7 2018.

[44] G. Dulac-Arnold, N. Levine, D. J. Mankowitz, J. Li, C. Paduraru, S. Gowal, and T. Hester. An empirical investigation of the challenges of real-world reinforcement learning. *arXiv:2003.11881*, 2020.

[45] B. Recht. A tour of reinforcement learning: The view from continuous control. *Annual Review of Control, Robotics, and Autonomous Systems*, 2(1):253–279, 2019.

[46] K. P. Wabersich and M. N. Zeilinger. Linear model predictive safety certification for learning-based control. *Proceedings of the IEEE Conference on Decision and Control*, 2018-December:7130–7135, 1 2019.

[47] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annu. Rev. Control Robot. Auton. Syst. 2020*, 3:269–296, 2020.

[48] L. Hewing, J. Kabzan, and M. N. Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28:2736–2743, 11 2020.

[49] A. Carron, E. Arcari, M. Wermelinger, L. Hewing, M. Hutter, and M. N. Zeilinger. Data-driven model predictive control for trajectory tracking with a robotic arm. *IEEE Robotics and Automation Letters*, 4:3758–3765, 7 2019.

[50] L. Hewing, A. Liniger, and M. N. Zeilinger. Cautious nmpc with gaussian process dynamics for autonomous miniature race cars. *2018 European Control Conference, ECC 2018*, pages 1341–1348, 11 2018.

[51] C. D. McKinnon and A. P. Schoellig. Learn fast, forget slow: Safe predictive learning control for systems with unknown and changing dynamics performing repetitive tasks. *IEEE Robotics and Automation Letters*, 4:2180–2187, 4 2019.

[52] C. D. McKinnon and A. P. Schoellig. Context-aware cost shaping to reduce the impact of model error in receding horizon control. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 2386–2392, 5 2020.

[53] G. Williams, P. Drews, B. Goldfain, J. M. Rehg, and E. A. Theodorou. Aggressive driving with model predictive path integral control. *Proceedings - IEEE International Conference on Robotics and Automation*, 2016-June:1433–1440, 6 2016.

[54] M. Zanon and S. Gros. Safe reinforcement learning using robust mpc. *IEEE Transactions on Automatic Control*, 6 2019.

[55] D. Caporale, A. Fagiolini, L. Pallottino, A. Settimi, A. Biondo, F. Amerotti, F. Massa, S. D. Caro, A. Corti, and L. Venturini. A planning and control system for self-driving racing vehicles. *IEEE 4th International Forum on Research and Technologies for Society and Industry, RTSI 2018 - Proceedings*, 11 2018.

[56] D. Caporale, A. Settimi, F. Massa, F. Amerotti, A. Corti, A. Fagiolini, M. Guiggian, A. Bicchi, and L. Pallottino. Towards the design of robotic drivers for full-scale self-driving racing cars. *Proceedings - IEEE International Conference on Robotics and Automation*, 2019-May:5643–5649, 5 2019.

[57] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, T. Stahl, L. Hermansdorfer, and M. Lienkamp. A software architecture for an autonomous racecar. *2019 IEEE 89th Vehicular Technology Conference (VTC2019-Spring)*, 2019-April:1–6, 4 2019.

[58] J. Betz, A. Wischnewski, A. Heilmeier, F. Nobis, L. Hermansdorfer, T. Stahl, T. Herrmann, and M. Lienkamp. A software architecture for the dynamic path planning of an autonomous racecar at the limits of handling. *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, pages 1–8, 11 2019.

[59] T. Stahl, A. Wischnewski, J. Betz, and M. Lienkamp. Multilayer graph-based trajectory planning for race vehicles in dynamic scenarios. *2019 IEEE Intelligent Transportation Systems Conference, ITSC 2019*, pages 3149–3154, 10 2019.

[60] A. Wischnewski, J. Betz, and B. Lohmann. A model-free algorithm to safely approach the handling limit of an autonomous racecar. *2019 8th IEEE International Conference on Connected Vehicles and Expo, ICCVE 2019 - Proceedings*, 11 2019.

[61] A. Wischnewski, J. Betz, and B. Lohmann. Real-time learning of non-gaussian uncertainty models for autonomous racing. *2020 59th IEEE Conference on Decision and Control (CDC)*, pages 609–615, 1 2021.

[62] D. Q. Mayne, M. M. Seron, and S. V. Raković. Robust model predictive control of constrained linear systems with bounded disturbances. *Automatica*, 41:219–224, 2 2005.

[63] M. Zeilinger, R. Hauk, M. Bader, and A. Hofmann. Design of an autonomous race car for the formula student driverless (fsd). *Proceedings of the OAGM & ARW Joint Workshop*, 1 2017.

[64] H. Tian, J. Ni, and J. Hu. Autonomous driving system design for formula student driverless racecar. *IEEE Intelligent Vehicles Symposium, Proceedings*, 2018-June:874–879, 10 2018.

[65] S. Nekkah, J. Janus, M. Boxheimer, L. Ohnemus, S. Hirsch, B. Schmidt, Y. Liu, D. Borbély, F. Keck, K. Bachmann, and L. Bleszynski. The autonomous racing software stack of the kit19d. *arXiv:2010.02828*, 10 2020.

[66] R. Shreyas, A. Bradley, and G. Collier. Mpc controller for autonomous formula student vehicle. *SAE Technical Papers*, 2020-April, 4 2020.

[67] J. Kabzan, M. Valls, V. Reijgwart, H. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa, R. Dubé, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 2020.

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

[69] S. Srinivasan, S. N. Giles, and A. Liniger. A holistic motion planning and control solution to challenge a professional racecar driver. *arXiv:2103.00358*, 2 2021.

[70] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4:3363–3370, 10 2019.

[71] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale rc cars. *Optimal Control Applications and Methods*, 36(5):628–647, 7 2017.

[72] H. Furtado. Model predictive control with a neural network model of a formula student prototype. Master's thesis, Instituto Superior Técnico, 2020.

[73] J. Antunes. Torque vectoring for a formula student prototype. Master's thesis, Instituto Superior Técnico, 2017.

[74] A. Antunes. Sideslip estimation of formula student prototype. Master's thesis, Instituto Superior Técnico, 2017.

[75] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 6 2000.

[76] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63:1883–1896, 7 2018.

[77] J. R. Cueli and C. Bordons. Iterative nonlinear model predictive control. stability, robustness and applications. *Control Engeneering Practice*, pages 1023–1034, 2008.

[78] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks: A computationally efficient approach for linear system. *IFAC-PapersOnLine*, 50:3142–3147, 7 2017.

[79] U. Rosolia and F. Borrelli. Sample-based learning model predictive control for linear uncertain systems. *Proceedings of the IEEE Conference on Decision and Control*, 2019-December:2702–2707, 12 2019.

[80] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006.

[81] J. Quiñonero-Candela, C. Rasmussen, and C. Williams. Approximation methods for gaussian process regression. 4 2007. URL `https://www.researchgate.net/publication/228618014_Approximation_methods_for_Gaussian_process_regression`.

[82] N. D. Lawrence and J. C. Platt. Learning to learn with the informative vector machine. *Proceedings, Twenty-First International Conference on Machine Learning, ICML 2004*, pages 512–519, 2004.

[83] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. *Proceedings of the 18th International Conference on Neural Information Processing Systems*, pages 1257–1264, 2005.

[84] J. Quiñonero, Q. Quiñonero-Candela, C. E. Rasmussen, and C. M. De. A unifying view of sparse approximate gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005.

[85] U. Rosolia, A. Carvalho, and F. Borrelli. Autonomous racing using learning model predictive control. *Proceedings of the American Control Conference*, pages 5115–5120, 6 2017.

[86] A. Micaelli and C. Samson. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots, 1993. URL `https://hal.inria.fr/inria-00074575`.

[87] R. N. Jazar. *Vehicle dynamics: Theory and applications*. Springer US, 2008.

[88] M. Brunner, U. Rosolia, J. Gonzales, and F. Borrelli. Repetitive learning model predictive control: An autonomous racing example. *2017 IEEE 56th Annual Conference on Decision and Control, CDC 2017*, 2018-January:2545–2550, 1 2018.

[89] U. Rosolia and F. Borrelli. Learning how to autonomously race a car: a predictive control approach. *IEEE Transactions on Control Systems Technology*, 1 2019.

[90] S. Xu. Learning model predictive control for autonomous racing improvements and model variation in model based controller. Master's thesis, KTH Royal Institute of Technology, 2018.

[91] D. Lam, C. Manzie, and M. Good. Model predictive contouring control. *Proceedings of the IEEE Conference on Decision and Control*, pages 6137–6142, 2010.

[92] H. B. Pacejka and E. Bakker. The magic formula tyre model. *Vehicle System Dynamics*, 21:1–18, 1 1992.

[93] R. B. Rusu and S. Cousins. 3d is here: Point cloud library (pcl). *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[94] J. D. Foley, M. A. Fischler, and R. C. Bolles. Graphics and image processing random sample consensus: A paradigm for model fitting with apphcatlons to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 6 1981.

[95] R. B. Rusu. Semantic 3d object maps for everyday manipulation in human living environments. *KI - Kunstliche Intelligenz*, 24:345–348, 11 2010.

[96] C. C. de Wit, P. Lischinsky, K. J. Åström, and H. Olsson. A new model for control of systems with friction. *IEEE Transactions on Automatic Control*, 40:419–425, 1995.

[97] J. Gonzales, F. Zhang, K. Li, and F. Borrelli. Autonomous drifting with onboard sensors. *13th International Symposium on Advanced Vehicle Control. (AVEC).*, pages 133–138, 2017.

[98] V. A. Epanechnikov. Non-parametric estimation of a multivariate probability density. *Theory of Probability and Its Applications*, 14:153–158, 1969.

[99] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd. Osqp: An operator splitting solver for quadratic programs. *2018 UKACC 12th International Conference on Control, CONTROL 2018*, page 339, 10 2018.

[100] I. Dunning, J. Huchette, and M. Lubin. Jump: A modeling language for mathematical optimization. *SIAM Review*, 59(2):295–320, 2017.

[101] A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming 2005 106:1*, 106:25–57, 4 2005.

[102] A. Domahidi and J. Jerez. Forces professional. *Embotech AG*, 2014. URL `https://embotech.com/FORCES-Pro`.

[103] A. Zanelli, A. Domahidi, J. Jerez, and M. Morari. Forces nlp: an efficient implementation of interior-point methods for multistage nonlinear nonconvex programs. *International Journal of Control*, 93: 13–29, 1 2020.

[104] J. A. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11:1–36, 3 2019.