Online Learning of MPC For Autonomous Racing

Gabriel Alexandre Francisco Costa

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

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

Examination Committee

Chairperson: Prof. Carlos Baptista Cardeira
Supervisor: Prof. Pedro Manuel Urbano de Almeida Lima
Member of the Committee: Prof. João Manuel Lage de Miranda Lemos

December 2021

Para a minha família.

Acknowledgments

First, I would like to thank both my supervisors, professors Pedro and Miguel, for their support throughout the development of this dissertation. Their technical overview on this subject was a fundamental contribution for this thesis. I would also like to thank them for encouraging me to develop this work in C++, which has become an extremely valuable tool in my skill set. I extend this gratitude to João Pinho, a fellow master thesis student, who welcomed me on his meetings with our supervisors and helped me to take the first steps by teaching me the basis of FST Lisboa's software pipeline. Besides this, João also contributed to this work through our interesting discussions on the autonomous driving topic, which lead to many of the ideas developed in this dissertation.

I would also like to thank in general FST Lisboa's team for providing me the necessary material for this dissertation as well as helping me with code management.

I also thank my friends from Instituto Superior Técnico, who not only make some of the best jokes I have ever heard but are also great and intelligent people to study and work with. Thank you Beatriz, Bruno, David, Paula, and Pedrocas Bombocas.

Last but not least, I would like to thank my family and girlfriend, Joana, for their incredible support provided not only during the development of this dissertation but for the past five years at Instituto Superior Técnico. They always helped to calm down during many stressful days and made sure I would laugh in situations that should not be possible. You bring out the best in me.

Resumo

Nesta dissertação, é projetada uma arquitetura de Controlo por Modelo Preditivo (MPC) baseada em técnicas de aprendizagem automática com o propósito de controlar um veículo autónomo de *Formula Student* (FS). Para a implementação em tempo real deste controlador e também para satisfazer os requisitos das competições de FS autónomas, o controlador é implementado em C++ e o problema de otimização do MPC é resolvido utilizando um *software* comercial.

Resumidamente, o controlador projetado é capaz de aprender iterativamente à medida que o veículo se auto conduz. O processo de aprendizagem é realizado para dois propósitos distintos: melhoria da precisão do modelo dinâmico do veículo utilizado pelo controlador e encontrar automaticamente os parâmetros do controlador que resultam nos tempos de volta mais rápidos possíveis. Encontrar as equações matemáticas que descrevam na totalidade a dinâmica de um veículo FS requer o uso de modelos nominais de veículo não lineares cuja obtenção é não trivial. Para este propósito, um modelo de Rede Neuronal Artificial (ANN) é adicionado ao modelo nominal com o intuito de modelar dinâmicas não consideradas no modelo original. A AAN é treinada com recurso a Aprendizagem Supervisionada (SL) de modo *online*, cuja aprendizagem se baseia em erros de previsão anteriores. Além disto, os parâmetros do controlador são afinados com recurso a métodos de Aprendizagem por Reforço (RL) com o objetivo de determinar os parâmetros que, iterativamente, permitem tempos de volta mais rápidos.

Em ambiente de simulação, serão realizados vários testes em três pistas diferentes. Nomeadamente, é demonstrado que, aplicando as duas técnicas mencionadas, o algoritmo de controlo permite uma redução até 16.5% dos tempos de volta.

Palavras-chave: Controlo por Modelo Preditivo, Aprendizagem de Modelo, Trajetória e Seguimento de Caminho, Corridas Autónomas.

Abstract

In this dissertation, a Learning based Model Predictive Control (LMPC) architecture is designed for the control of a Formula Student (FS) autonomous vehicle. For the implementation of this controller in real time to satisfy the FS driverless requirements, the C++ programming language is used and the MPC's optimization problem is solved using a commercial solver.

In summary, the developed controller is able to iteratively learn as the vehicle drives itself. This learning process is carried out for two distinct purposes: improving the accuracy of the vehicle model used by the controller and automatically finding the controller parameters that result in the fastest lap times. Finding the mathematical equations that fully describe the race car dynamics requires the use of highly nonlinear vehicle nominal models which are difficult to obtain. For this purpose, an Artificial Neural Network (ANN) is added to a vehicle nominal model in order to correct for unmodeled dynamics not considered in the nominal model. The ANN is trained in an online Supervised Learning (SL) approach, which learns based on past model prediction errors. Furthermore, the controller's parameters are tuned in a Reinforcement Learning (RL) environment in order to find the set of parameters that iteratively allow for faster lap times.

In a simulation environment, various tests on three different tracks are performed. Moreover, it is shown that by employing these two learning procedures, the full control algorithm is able to reduce lap times up to 16.5%.

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

Contents

List of Tables

List of Figures

Nomenclature

Controller parameter learning

λ_T	Segment average time low pass filter parameter.
$\mathbb{P}$	Probability.
$\mathbf{D}$	Population.
$\mathbf{d}$	Individual genes.
$\mathbf{f}$	Fitness vector.
$\mathbf{r}$	Reward vector.
$d^{\text{low/high}}$	Parameter space exploration limit.
k_c	Cone hit penalization.
k_T	Reward function shape parameter.
L	Track length [m].
$m^{\text{low/high}}$	Mutation random variable.
n^{cones}	Number of cones hit in a given track segment.
N_{CP}	Number of check points.
N_{pop}	Number individuals in a population.
n_d	Number of genes in an individual.
R	Track curvature radius [m].
r	Reward function.
T_i	Segment lap time.
T_{lap}	Point mass model theoretical lap time.
v	Point mass model velocity [m/s].

Error correction model

$\boldsymbol{\epsilon}$	Discrepancy vector.
Γ	Forgetting factor normalization term.
γ	Forgetting factor.
λ	Levenberg-Marquardt damping factor.
λ_{MSE}	MSE low pass filter parameter.
$\mathbf{A}$	Transformation matrix.
$\mathbf{J}$	Jacobian matrix.
$\mathbf{p}$	Parameter vector.
$\mathbf{v}$	Feature vector.
MSE	Mean squared error.
SMSE	Smoothed mean squared error.
e	Correction function.
n_e	Number of outputs.
n_h	Number of hidden neurons.
n_v	Number of features/inputs.
n_{batch}	Learning batch size.
R	Learning loss function.

Model predictive control

α	Center line penalization weight.
$\alpha_{CL_{\max}}$	Maximum center line error allowed.
β_δ	Steering smoothness weight.
$\Delta\mathbf{u}_{\max}$	Maximum control action variation.
$\Delta s_{\min\,/\,\max}$	Neighborhood track progress interval limits.
$\hat{e}_{CL}$	Center line error estimation [m].
$\hat{e}_L$	Lag error estimation [m].
λ_s	Last predicted track progress weight.
$\mathbf{u}_{\min\,/\,\max}$	Control actions' limits.
e_{CL}	Center line error [m].

g	Track's center line function.
J	MPC cost function.
N	Prediction horizon.
n	Center line penalization exponent.
q_{v_y}	Lateral velocity penalization weight.
$q_{v_{\max}}$	Exponential velocity soft contraint shape parameter.
s	Track length or track progress [m].
T_s	Sampling time [s].
$v_{\max}$	Maximum longitudinal velocity.

Kinematic, Dynamic and Blended bicycle models

α	Tire-slip angle [rad].
δ	Steering angle [rad].
η_{motor}	Motor and transmission efficiency [-].
λ	Blending factor [-].
$\mathbf{R}$	Rotation matrix [-].
$\mathbf{u}$	Input vector.
$\mathbf{x}$	State vector.
Ψ	Vehicle orientation [rad].
ρ	Air density [Kg m^{-3}].
A_F	Vehicle's frontal area [m^2].
$B_{f/r}$	Tire model shape parameter [-].
C_d	Drag coefficient [-].
$C_{f/r}$	Tire model shape parameter [-].
C_r	Rolling resistance coefficient [-].
d	Normalized throttle [-].
$D_{f/r}$	Tire peak force [N].
f	State transition function.
g	Gravity acceleration [m s^{-2}].

GR	Transmission gear ratio [-].
I_z	Vehicle's inertia around the yaw axis [kg m^2].
L	Wheelbase [m].
l_f	Distance from front axis to the vehicle's CG [m].
l_r	Distance from rear axis to the vehicle's CG [m].
m	Vehicle's mass [kg].
r	Vehicle's yaw rotation speed [rad s^{-1}].
r_{wheel}	Wheel radius [m].
$T_{\max}$	Motor's maximum torque [Nm].
V_{blend}	Blended model blending parameter [m s^{-1}].
$v_{x/y}$	Vehicle's longitudinal and lateral velocity [m s^{-1}].
X/Y	Vehicle's global spatial coordinates [m].

List of Abbreviations

ANN	Artificial Neural Network.
CG	Center of Gravity.
DL	Deep Learning.
DV	Driverless Vehicle.
EBFN	Elliptical Basis Function Network.
ERM	Empirical Risk Minimization.
FSG	Formula Student Germany.
FS	Formula Student.
GA	Genetic Algorithm.
GD	Gradient Descent.
LMPC	Learning-based Model Predictive Control.
LM	Levenberg-Marquardt.
ML	Machine Learning.
MPC	Model Predictive Control.
NLP	Nonlinear Programming.
OGD	Online Gradient Descent.
OLM	Online Levenberg-Marquardt.
PU	Processing Unit.
RBFN	Radial Basis Function Network.
RL	Reinforcement Learning.
ROS	Robot Operating System.
SAE	Society of Automotive Engineers.
SLAM	Simultaneous Localization And Mapping.
SL	Supervised Learning.
ZOH	Zero Order Hold.

Chapter 1

Introduction

In this first chapter, a comprehensive overview of driverless vehicles will be presented in order to provide insight into the work developed in this dissertation. Firstly, the motivation that drove this dissertation will be described. Furthermore, an overview of the control theory applied to driverless vehicles will be presented. Finally, the objectives of this thesis and an outline of this dissertation will be presented.

1.1 Motivation

Developing self-driving vehicles has been one of the major technology focuses of the twenty-first century. This topic has been experiencing large growth with breakthroughs in technology that enable such vehicles to drive themselves and to perform tasks like parking autonomously. Following this topic, Formula Student teams were recently introduced to a new type of racing competition: the driverless competition. To participate in such competitions, teams are required to build and test a self-driving vehicle, which relies on the use of sensors and complex control algorithms in order to guide the vehicle through the race track at fast speeds.

According to the American Society of Automotive Engineers (SAE), autonomous vehicles can be ranked on a zero to five scale as suggested in figure 1.1. In summary, level 0 represents a standard vehicle that is thoroughly controlled by a human driver without any kind of automation or supervision. Level 1 and level 2 include vehicles that offer automatic driving assistance such as cruise control, lane keeping assistance, highway assistance, and many others. Overall, from levels 0 to 2 the driver must actively participate in driving as the automation/supervision technologies available in these levels do not allow for the vehicle to drive autonomously. At level 3, the car is considered to be able to take over control in certain traffic conditions, for example, being able to drive autonomously on a highway or having a traffic jam assistance. At this level, the driver is able to do other tasks while the car drives itself. Nevertheless, an issue inherent to this automation level is that the driver is always expected to take over control if the automated functions are disturbed or unable to handle certain traffic situations. In level 4, full autonomous driving functions are available in the vehicle for all required traffic scenarios so that human interventions are not required. Furthermore, at this level, the driver still has the possibility to take over control in case of system failure or personal desire. Finally, level 5 represents the fully autonomous vehicle that does not need human interaction anymore. At this level, the cars are mainly robots that transport passengers and/or goods independently [1].

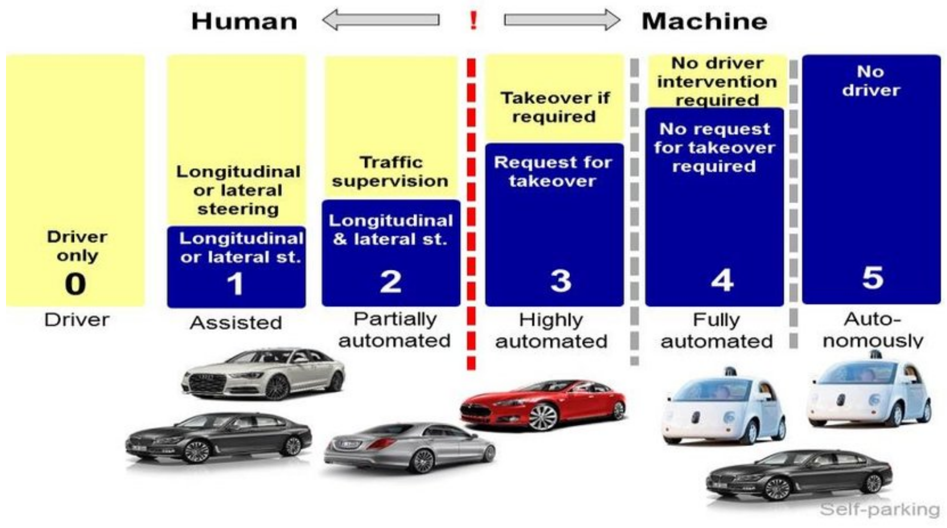


Figure 1.1: Levels of autonomous driving according to SAE J3016 [1].

Besides technology-related research, autonomous driving is a far reaching topic that is constantly under evaluation regarding legal, social, and ethical aspects. The advantages and disadvantages of autonomous driving are intensively discussed due to the big impact of this technology on human mobility behavior, infrastructure and environment. In general, autonomous driving includes a tremendous potential to improve life quality, reduce traffic accidents and decrease impacts on environmental pollution, but still, there are open points in terms of safety, costs, and ethical issues.

Several companies have been pioneers in the development of autonomous vehicles. In the last decade, these companies have seen clear improvements, both in safety and technology each year in their autonomous vehicles. Waymo, a company that began as the Google self-driving project in 2009, is currently operating in cities like Phoenix, San Francisco, and Mountain View in the United States of America (USA). Waymo's mission is to make autonomous driving safe and easy for people and things to get to their destination. As of today, Waymo's vehicles operate with a level 4 autonomy, meaning that, for the most part, Waymo's vehicles drive their passengers without any human occupying the driver's seat. At the moment, Waymo's fleet has driven more than 20 million miles on the road and more than 15 billion miles in simulation. During 2019 and the first nine months of 2020, Waymo claimed that their vehicles were involved in 18 car accidents and 29 near-miss collisions while driving more than 6.1 million miles, without any fatality or severe injuries [2]. An illustration of the Waymo One fully autonomous vehicle can be seen in figure 1.2a in page 4.

Tesla is a well known car manufacturer founded in the year 2003 in the state of California, USA. Tesla's mission is to accelerate the world's transition to sustainable energy by producing electric vehicles. Besides this objective, Tesla vehicles are known for their autopilot capabilities - their autonomous driving system. As for today, Tesla's vehicles are classified to have level 2 autonomy, with the objective being to achieve level 5 autonomy. Tesla's vehicle safety report [3] shows an improvement in the autopilot's safety by an accident rate reduction from one accident per 3.34 millions driven miles in the 3rd quarter of 2018 to one accident per 4.19 million driven miles in the 1st quarter of 2021 with the autopilot engaged. They also compare these autopilot's values to accidents that occurred when driving without the autopilot or any of the vehicle's safety features on, which amounted for one accident per 978 thousand miles driven in the 1st quarter of 2021 - a significantly higher accident rate when compared to the autopilot's rate. In [4], a survey on Tesla's users, showed that these drivers frequently use the autopilot and that these users in general do not perceive eventual autopilot's failures as posing a significant risk. Nevertheless, Tesla's autopilot strategy has been target to many criticism with claims that Tesla's self driving strategy is outdated and possibly dangerous [5]. An illustration of the well known Tesla Model S can be seen in figure 1.2b.

Regarding autonomous racing, the Indy Autonomous Challenge[1] is promoting an university competition with the objective of developing an autonomous racing vehicle using a modified Dallara IL-15 racecar to compete in the world's first head to head autonomous race at the Indianapolis Motor Speedway, which is set to take place in October 2021. One of the competition's objectives is to bring new and improved technologies to autonomous vehicle commercialization by tackling three issues related to autonomous driving:

1. **Solving edge case scenarios**: to develop innovative technologies for problems and situations that occur only at an extreme operating parameter, such as avoiding unanticipated obstacles at high speeds while maintaining vehicle control;

2. **Catalyzing new autonomous vehicle technologies and innovators**: claiming that autonomous vehicles are too expensive for scaled commercial deployment and that automakers and technology companies are seeking sources of new intellectual property and qualified engineers;

3. **Acceptance and use of autonomous vehicle technologies**: by engaging the public with increased exposure to autonomous vehicle technologies, the competition hopes for facilitating an understanding of autonomous vehicles and their potential.

An example modified Dallara IL-15 for the Indy Autonomous Challenge competition can be seen in figure 1.2c.

For the realization of autonomous driving functions reliable and robust determination of position and trajectory of the vehicle on the road for tracking and controlling the vehicle is fundamental. In addition to the vehicle's self behavior, the behavior of all other traffic participants, like vehicles and pedestrians, has to be observed and predicted. In combination with environmental models, like digital maps, the recognition of all traffic participants is required for continuous modeling of the traffic situation in different scenarios. All this information must be processed in real time, resourcing to sensor fusion and evaluation [6, 7].

The typical autonomous driving pipeline is illustrated in figure 1.3. Referring to diagram (a) in this figure, from left to right, the block that corresponds to sensing concerns the collection of sensor data from

[1] `https://www.indyautonomouschallenge.com/`

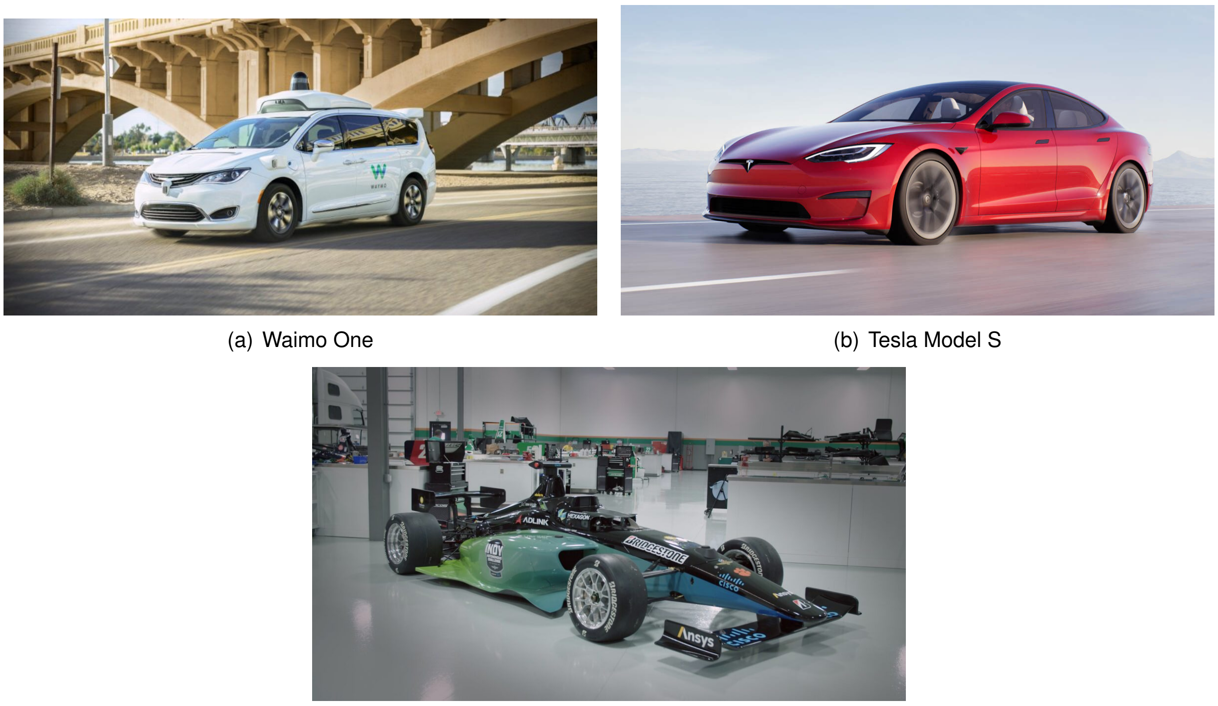

(a) Waimo One (b) Tesla Model S (c) Modified Dallara IL-15

Figure 1.2: Some autonomous vehicles.

a multitude of sensors such as cameras, Light Detection And Ranging (LiDAR) sensors, Radio Detection And Ranging (radar) sensors, ultrasonic sensors, Inertial Measurement Units (IMUs), and many others. The next block relates to the application of perception algorithms to the sensor data. Two main tasks are accomplished in this block: object detection and tracking and localization. Object detection and tracking concerns the necessary detection of road features and objects, like traffic signs and pedestrians, as well as the tracking of these objects. Localization, as suggested by the word itself, concerns the localization of the vehicle with respect to some map/coordinates in the world. The next block, regarding assessment and behavior prediction, concerns the interpretation of the object detection and localization data, for example, for identifying a pedestrian and tracking its movement to predict its path. Overall, this block is responsible for estimating other agents' intents. The planning block is responsible for planning a feasible vehicle trajectory that respects traffic rules while respecting the information extracted from the assessment and behavior prediction block. Finally, the control block is responsible for computing the required actuation, i.e., vehicle steering and throttle, in order to follow the planned trajectory.

Furthermore, object detection state-of-the-art algorithms often rely on Computer Vision (CV) [9] and/or Deep Learning (DL) [10] techniques, while generally using data from exteroceptive sensors, like LiDARs, cameras and radars. Localization can be accomplished resourcing to, for example, Simultaneous Localization and Mapping (SLAM) algorithms [11, 12]. SLAM can be defined as a process where a robot or a vehicle builds a map representing its spatial environment while keeping track of its position within the built map. SLAM algorithms take into account a variety of parameters like sensor measurements, map representation, robot dynamics, environmental dynamics, and many others. These algorithms commonly rely on Kalman Filters (KFs), the most popular being the Extended Kalman Fil-

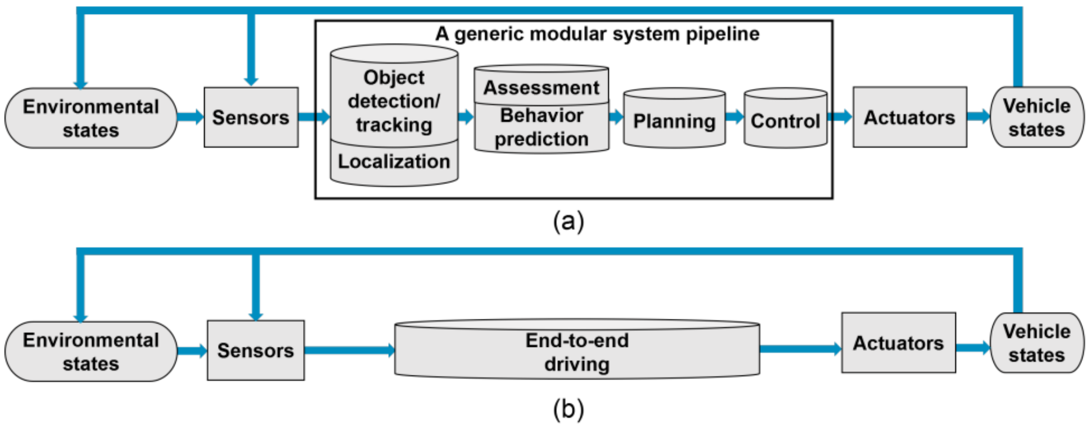


Figure 1.3: Typical autonomous driving pipeline: (a) a generic modular system, and (b) an end-to-end driving system [8].

ter (EKF) [13, 14]. SLAM algorithms in autonomous driving use a variety of sensors, namely a stereo vision system (cameras), LiDARs, IMUs and wheel encoders [15]. Behavior prediction algorithms typically resource to Deep Learning [16], nevertheless, recent research on this topic has been done using Monte Carlo algorithms [17] as well as using particle filter classifiers [18]. Path planning problems are typically formulated as nonlinear optimization problems that revolve around a set of waypoints - points that characterize where the vehicle should pass. Generally, these algorithms work by generating a set of candidate paths and the optimal path is selected according to path cost criteria, like path smoothness or consistency [19, 20]. Finally, control algorithms for autonomous driving may range from the well known Proportional Integral Derivative (PID) controllers [21], classical optimal control theory methods as iterative versions of the Linear Quadratic Regulator (LQR) [22] to more computationally complex methods like Model Predictive Control (MPC) [23]. Other successful control approaches on autonomous driving have used predictive control algorithms aided by some Machine Learning (ML) techniques in order to improve the controller's performance [24].

Similar to the previously mentioned Indy Autonomous Challenge, Formula Student (FS) is Europe's most established educational engineering competition, where various university teams design, build, test, and compete with a formula type racecar. FS competitions are subdivided into static and dynamic events.

Regarding static events, the goal of the Business Plan Presentation is to evaluate the team's ability to develop and deliver a comprehensive business model, which demonstrates how their product – a prototype racecar – could become a rewarding business opportunity that creates monetary profit. The Cost and Manufacturing event aims at evaluating the team's understanding of the costs and manufacturing processes associated with manufacturing the team's vehicle. Inherent to this, teams should justify their trade off decisions between content and cost, and make or buy as well as understanding the differences between developing a racecar prototype and mass production. For this purpose, students create a detailed report of all costs associated with materials, processes, and assembly of the car. The last static event, Engineering Design, concerns the evaluation of the student's knowledge of the vehicle and its related engineering concepts. As such, teams are expected to support their engineering design by delivering a detailed engineering report of the vehicle.

The dynamic events' purpose is to test the developed racecar's different driving abilities. These events can be characterized as follows:

- **Acceleration**: the vehicle drives in a straight track of approximately 75 m in length in order to test its longitudinal acceleration capabilities;
- **Skidpad**: the vehicle drives in a track formed by two pairs of circles in an "eight" pattern in order to test the vehicle's lateral acceleration capabilities;
- **Autocross**: the vehicle drives a single lap track, roughly with 1 km, with components such as hairpins, slaloms and chicanes;
- **Trackdrive**: the vehicle drives 10 laps in a closed loop track with similar characteristics to Autocross;
- **Efficiency**: the vehicle is evaluated in terms of energy efficiency regarding the previous event.

Since 2017, Formula Student Germany (FSG) - a FS competition held in Hockenheim, Germany - introduced a new competition class for autonomous racecars, the Driverless Vehicle (DV) class.

With the purpose of accelerating research in autonomous vehicles, this new class was mainly encouraged by automotive industry stakeholders like Audi, Daimler, and Continental which also sponsor the competition and provide officials and design judges for the competition. Indeed, throughout these years, the competition's new class has seen new fastest lap times when compared to the previous years[2]. Figure 1.4 displays the corrected Trackdrive times (i.e. clock time plus potential penalties) for the fastest year's team able to complete the event's 10 laps. This figure shows a significant performance improvement on this event. This is a product of the team's yearly evolution by developing an increasingly better vehicle control pipeline.

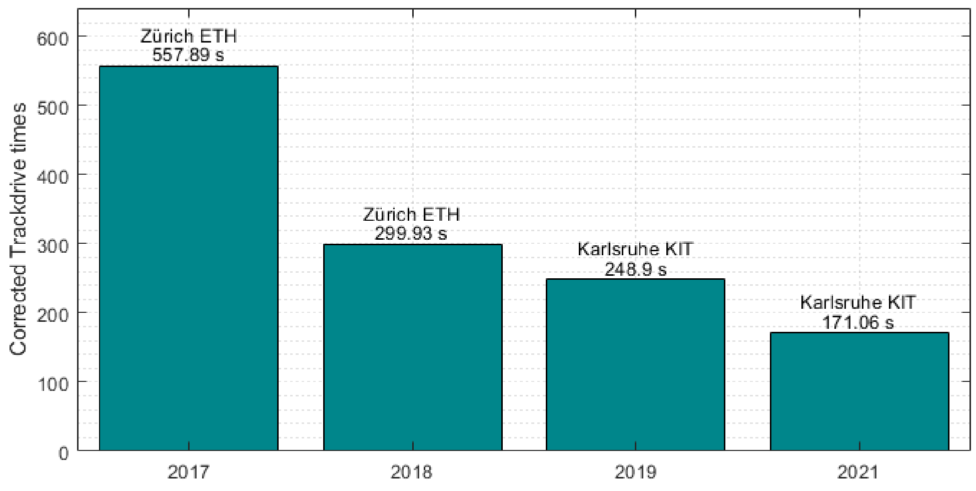


Figure 1.4: Fastest team's corrected Trackdrive time evolution. Note that in 2020 the competition was not realized due to the coronavirus pandemic.

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

Furthermore, as an incentive for teams to develop driverless vehicles, from 2022 onwards, the DV class will be merged with the human driver classes. As such, all the vehicles in the FSG competition must be adapted with an autonomous driving system, or else teams will not score in driverless events, losing points. In addition, in the following year, 2023, it is planned to add another driverless dynamic event.

Regarding the DV class competitions' environment, no pedestrians or other vehicles are allowed near the track. The track itself is composed of blue and yellow cones that represent the left and right limits of the track, respectively. Exit and entry points are marked by small orange cones and big orange cones are used in the track's start and finish lines as well as for timing the vehicle. Given this, the DV class competition environment is rather controlled and simplified when compared to the commercial vehicle's typical urban driving scenario, which allows FS teams to develop a simpler pipeline. For example, object detection tasks are made simpler since the vehicle only needs to detect the four different cone types mentioned previously. Despite this, localization, path planning, and control problems can become harder, as, in order to achieve fast lap times and maximizing the team's score, the vehicle should be pushed to its handling limits. A priori track information is not allowed for the Autocross event, while for the Trackdrive event some competitions allow for prior track data collection. This makes SLAM algorithms one of the main components of the DV class pipeline, as for example, for Trackdrive events without a priori track data, the vehicle can use the first lap to collect track information, driving slower while only using current measurements. As the second lap starts the SLAM algorithm has built a virtual map of the track which can be exploited by localization, path planning, and control algorithms that potentially allow the vehicle to drive much faster.

The FS team from Instituto Superior Técnico (IST), FST Lisboa, has been building FS vehicles since 2001. Since 2019, the team has been working on its first autonomous vehicle, FST10d. This vehicle was adapted from the 9th FST Lisboa's vehicle, designed for the 2018/19 competitions. Figure 1.5a and 1.5b show FST10d performing the Trackdrive event at FSG 2021 and figure 1.5c shows the original vehicle on which FST10d was built, FST9e.

With the goal of designing a self-driving vehicle, several theses have been developed addressing the topic of self-driving vehicles [25–29]. This dissertation is inserted in such topic, namely the development and application of control algorithms.

Currently, FST Lisboa's software pipeline includes simple controllers, while the rest of the pipeline is more advanced. The implemented controllers so far are a Pure Pursuit Controller [30] for vehicle lateral control and a Proportional Integral (PI) controller for its longitudinal control. Raffaello D'Andrea, a professor of dynamic systems and control at ETH Zürich, once made the analogy of robot athleticism, in which he claims that for humans to achieve remarkable athletic performance their athletic bodies require athletic minds, similarly, for pushing robots to their physical limits their "minds" must also be gifted with "athletic" power, with complex yet powerful algorithms. As such, this thesis aims at developing a control algorithm for FST Lisboa that is capable of not only guiding the car through the race track in the least possible amount of time but also to apply Machine Learning (ML) techniques that provide more accurate and precise control of the vehicle, allowing FST10d to be driven near its physical limits.

(a) FST10d at FSG 2021.

(b) FST10d at FSG 2021.

(c) Original FST9e.

Figure 1.5: First FST Lisboa's autonomous vehicle - FST10d.

1.2 Topic Overview

As expressed in the previous section, various control algorithms are being researched with the application of autonomous driving. These controllers vary widely in terms of complexity, ranging from simple and well studied controllers like Proportional Integral Derivative (PID) controllers [21] up to more complex optimal based controllers like Model Predictive Control (MPC) [23].

As an example, Levinson et al. [22] presents decoupled planning and control techniques, meaning that trajectory planning and control are two different consecutive tasks, as suggested in the pipeline scheme in figure 1.3. As such, in that paper, before the control tasks, the vehicle's trajectory is planned. In summary, this trajectory planning task is based on optimization algorithms that tend to minimize the lateral and longitudinal jerk - the derivative of acceleration. After the trajectory is planned, the control algorithm's task is to find the control actions - throttle and steering - that the vehicle has to maintain in order to follow that trajectory. The developed control techniques are based on a mixture of a MPC strategy based on physically based vehicle models, along with PID control for the lower level feedback control tasks such as applying a given torque to the wheels in order to achieve a given reference steering angle. The developed MPC's cost function reflects a quadratic cost based on the planned trajectory. More specifically, the MPC's cost is higher if the predicted vehicle states are far from the desired trajectory states and lower otherwise.

Contrary to the previous example, Liniger et al. [31] developed a MPC for autonomous racing in which the trajectory planning and control are coupled. The main advantage of this strategy is that trajectory planning and tracking can be combined into one nonlinear optimization problem, allowing to find the optimal trajectory and also the corresponding optimal control actions using the same optimization problem. As such, the developed MPC's cost function is based on the vehicle's progress along the track instead of a reference given by the trajectory planning, as in the previous example. The controller developed in this dissertation also follows the idea of Liniger et al. [31]. Moreover, the developed controller aims at finding the control actions that maximize the vehicle's center line progress at a given sampling instant. The mathematics and mechanisms used by the developed controller will be further introduced in chapter 4 - MPC Formulation.

Furthermore, another relevant topic in autonomous driving is the use of Machine Learning (ML) for controller improvement purposes. This is often referred to as learning based control, in which ML techniques are typically used with a MPC for several purposes. Hewing et al. [32] overviewed some of this recent research. The authors claim that this research addresses three main topics:

1. **Learning the system dynamics**: addressed by most of the research, this technique relates to learning for automatic improvement of the system's prediction model from recorded data. This considers the automatic adjustment of the system model, either during operation or between different operational instances;

2. **Learning the controller design**: With a recent increasing interest, this technique infers the parameterization of the MPC controller, i.e., the cost and constraints, that lead to the best closed-loop performance;

3. **MPC for safe learning**: a technique used to derive safety guarantees for learning-based controllers. This technique's main idea is to decouple the optimization of the objective function from the requirement of constraint satisfaction, which is addressed using MPC techniques.

This thesis mainly adopts techniques from the two first topics: learning the system dynamics and learning the controller design.

Regarding learning the system dynamics, as mentioned before, this approach mainly consists in collecting data offline and/or online, such as state measurements, and using this information either to derive a system's model or to improve the prediction model used in the MPC. Many learning based MPC techniques make use of an explicit distinction between a nominal system model and an additive learned term that corrects for the nominal model's uncertainty. In the scope of autonomous driving, the vehicle's nominal model is often derived through physical principles, namely using Newton's laws of motion. The additive correction model is then derived through data, which is learned by some ML technique. Researched ML techniques for this purpose include, for example, the use of Artificial Neural Networks (ANNs) [33, 34] and full or sparse Gaussian Process (GP) regressions [35, 36].

Despite the system's prediction model being a core element of a MPC, other elements in the MPC formulation, such as the design of the cost function and constraints also have a major influence on the resulting closed loop performance. As mentioned previously, learning the controller design concerns the parameterization of the MPC's cost function and constraints. Moreover, one can successively adjust these parameters according to a desired performance indicator. As an example regarding autonomous racing, consider a vehicle controlled by a MPC parameterized by n_d parameters in its cost function and constraints. Considering that, for example, the goal is to minimize the vehicle's lap time for a given closed loop track, the objective of an algorithm for learning the controller design would be to find the best value for the n_d parameters that result in the lowest lap time possible. There are many optimization algorithms one can use to solve such problems. Furthermore, in research papers like [37, 38] this is accomplished using Reinforcement Learning (RL) techniques. Moreover, Genetic Algorithms (GAs) have also been used for automatic parameter tuning [39–41].

Focusing on research being done in the Formula Student autonomous racing context, several teams have been publishing research papers detailing their approach for developing a driverless Formula Student vehicle. Namely, the driverless FS team from ETH Zürich named AMZ Driverless, which was previously mentioned in figure 1.4 as the fastest team completing the Trackdrive event in the years 2017 and 2018, published several research papers regarding the work performed on their autonomous vehicles [35, 42, 43]. Namely, [42] presents a vehicle model built on physical principles, which was also employed in this thesis - further detailed in chapter 3. This model results from a blend of two well known models in vehicle dynamics: the kinematic bicycle model and the dynamic bicycle model. Furthermore, in AMZ's most recent paper by Srinivasan et al. [43], the authors claim that the developed control architecture is capable of outperforming a professional driver racing the same car. The fastest team at completing the Trackdrive event in the recent years of 2019 and 2021, KA-RaceIng, the team from Karlsruhe Institute of Technology (KIT), also published some research papers regarding the development of their autonomous vehicles [44, 45]. In [45] two separate controllers are used for motion control. Namely, the team uses a MPC for the lateral movement of the vehicle, which is responsible for computing the steering control actions, and a PI controller for the longitudinal movement of the car, as the authors claim that the longitudinal dynamics of a racecar are well represented by a double integrator.

In this thesis, a nonlinear MPC for FST Lisboa's FST10d driverless vehicle is developed for the tasks of trajectory planning and control. This controller is developed based on the blended model suggested in [42] with an additive correction model based on Artificial Neural Networks. The additive correction model makes use of using real time data to improve itself in an online learning fashion. The MPC's cost function and constraints are based on the idea of Liniger et al. [31] in order to couple the path planning and control tasks in a comprehensive manner. Finally, in order to learn the controller design, the parameters used in the MPC's cost function and constraints are tuned in a Reinforcement Learning procedure consisting of time measurements between given track segments divided by check points, which will be further explained in chapter 5 - Controller Parameter Learning.

Several other theses from Instituto Superior Técnico have contributed to research on the topic of autonomous racing while also contributing to FST Lisboa's autonomous vehicles. Namely in [25] a MPC is designed while using a neural network model trained in a detailed vehicle simulator. The author also trains another neural network to learn the resulting MPC control actions. In [26] and [27] torque vectoring and sideslip estimation techniques are explored. In [29] path following techniques coupled with classical linear control methods, such as the Linear Quadratic Regulator (LQR) are developed. Finally, Pinho [28] proposes a LMPC design based on [46]. Furthermore, the author exploits Gaussian Process Regression (GPR) techniques in order to learn a vehicle model correction. João Pinho's thesis [28] can be seen as a thesis parallel to this dissertation, as Pinho's thesis also presents techniques for learning the model correction and learning the controller design.

1.3 Objectives and Deliverables

The objective of this thesis is to provide FST Lisboa with a new autonomous vehicle controller. As such, this controller aims at improving the trajectory planning and control algorithms currently being used in FST Lisboa's software pipeline. To achieve this, a Model Predictive Controller with characteristics mentioned in the previous section was chosen as the control architecture. This controller should improve itself by making use of previously collected data and real time data, in order to be able to improve the vehicle's performance in an iterative manner. Being the Trackdrive event the main dynamic event of driverless FS competitions, the learning methods employed by the controller should result in decreasing lap times as the vehicle completes more laps around the track. This will be achieved by learning in a supervised manner an additive correction term to the vehicle's physics based model and automatically tuning the MPC parameters in a Reinforcement Learning procedure.

Given this, the necessary adaptations to implement this controller should be made to the FST Lisboa's software pipeline mainly by delivering the required C++/ROS packages which should be in conformity with the rest of the pipeline. Other external tools required for some code generation, such as MATLAB scripts, shall be included as well.

1.4 Thesis Outline

This dissertation is organized in several chapters.

Chapter 2 - Theoretical Background - introduces the theoretical methods that serve as the basis of the algorithms developed for this thesis. Namely, this chapter starts with a description of Model Predictive Control. Then, Supervised Learning is introduced, which will be used in the context of improving the vehicle's predictive model. Algorithms relevant for Supervised Learning will be introduced as well. Finally, the generic environment of Reinforcement Learning will be introduced. Reinforcement Learning will be used to learn the controller parameters that maximize a given performance criterion. In this section, the Genetic Algorithm will also be presented in a Reinforcement Learning context.

Chapter 3 - Vehicle Models - contains the vehicle models used by the controller. This chapter starts with an overview of the vehicle geometry as well as how the vehicle reacts with its environment by applied forces. Then, two famous vehicle models are presented and deducted: the kinematic bicycle model and the dynamic bicycle model, both being continuous time models deduced from Newton's laws of motion. Then, the discretization method employed for transforming the previously mentioned models into discrete time models will be explained and the blended bicycle model will be introduced. Finally, the error correction model will be presented with the purpose of learning an additive correction term to correct for the blended model mismatch.

Chapter 4 - MPC Formulation - presents and explains the formulation of the MPC optimization problem for the autonomous racing purpose. In order to understand this formulation, this chapter starts by introducing the idea of center line track progress. Then, the MPC's cost function and constraints, which are based on track progress, are introduced.

Chapter 5 - Controller Parameter Learning - presents a method for automatically tuning the controller parameters. This method is based on testing different parameter combinations in given track segments. As such, this chapter starts with an introduction to this problem, and then the method used for track segmentation is presented. Finally, an explanation of how the parameter learning algorithm works is given.

Chapter 6 - Implementation - contains notes regarding the implementation of the developed controller in FST Lisboa's software pipeline. Moreover, the tools used for this implementation will be introduced.

Chapter 7 - Results - contains results withdrawn from simulations. Afterward, these results are discussed and explained.

Chapter 8 - Conclusions - presents final conclusions regarding the achievements accomplished in this dissertation and ideas for future work on this research topic are given.

Chapter 2

Theoretical Background

In this chapter, the theoretical foundations on which the following chapters are based will be introduced. First, a general overview of Model Predictive Control (MPC) will be introduced as well as the concept of Receding Horizon Control (RHC). Then, under the scope of model learning resourcing to Supervised Learning (SL), Artificial Neural Networks (ANNs) are introduced as well as two well known training algorithms: the Online Gradient Descent (OGD) and Levenberg-Marquardt (LM) algorithms. Finally, with the purposed of automatically tuning the MPC parameters, Reinforcement Learning (RL) will be introduced as well as the genetic algorithm which will be used to maximize the RL's reward function.

As a further note, column vectors will be denoted in bold lowercase, $\mathbf{x} \in \mathbb{R}^n$, and matrices will be denoted as bold uppercase, $\mathbf{X} \in \mathbb{R}^{n \times m}$. Scalars are denoted as non-bold uppercase or lowercase, $x, X \in \mathbb{R}$.

2.1 Model Predictive Control

Receding Horizon Control (RHC) is a general purpose control scheme that assumes that an infinite horizon sub-optimal controller can be attained by repeatedly solving a Finite-Time Constrained Optimal Control (FTCOC) problem at each discrete sampling instant [47]. Consider sampling instant t and the system's current state at this sampling instant. An open-loop optimal control problem is then solved over a finite time horizon, i.e., for the corresponding sampling instants in the interval $[t, t+N]$ as illustrated by the top diagram in figure 2.1. As a result of solving the optimal control problem, a sequence of control actions are obtained for each instant in the finite horizon, however, the computed control actions are only applied in the sampling interval $[t, t+1]$. In other words, only the first control action computed from the control actions sequence is applied during this interval. When the next sampling instant occurs at $t+1$ a new optimal control problem is solved according to the new measured system state at this sampling instant over a shifted finite time horizon, i.e. for sampling instants in $[t+1, t+1+N]$ as illustrated by the bottom diagram in figure 2.1. This process is then repeated over each next sampling instant attaining an infinite horizon sub-optimal controller.

Model Predictive Control (MPC) is based on the idea of RHC. Similarly to RHC, the MPC objective

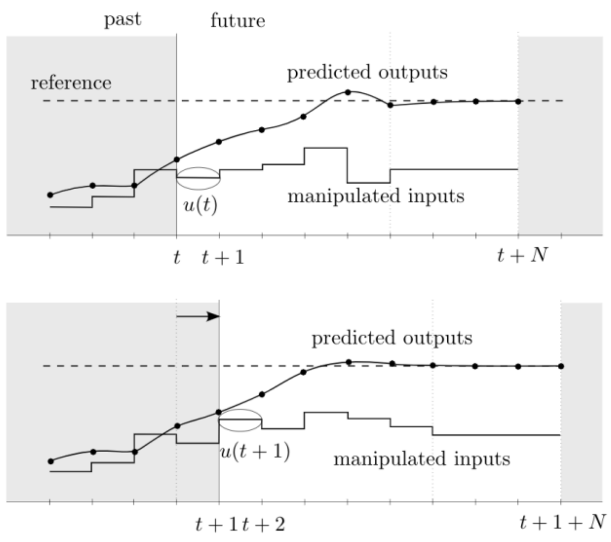


Figure 2.1: Receding Horizon principle [47].

is to solve at each iteration the set of future input variables that, when applied to the system, the system behaves in the desired manner. MPC accomplishes this by predicting future system states through a system's dynamic prediction model. As such, the main idea of MPC is to take advantage of mathematical models developed for a given system, with the purpose of controlling that system.

This controller's scheme can be further explained by implementing the following general steps:

1. At a given sampling instant, t, predict the N future states of the system, $\mathbf{x}_{t+k}$, $k = 1, ..., N$, by using the system's model as a function of the states at time instant t and future system inputs, $\mathbf{u}_{t+k-1}$, $k = 1, ..., N$;
2. Define a cost function as well as constraint function(s) based of the system's states and inputs and optimize it with respect to future inputs, $\mathbf{u}_{t+k-1}$, $k = 1, ..., N$;
3. Apply the first input as computed by the optimization problem, $\mathbf{u}_t^*$, to the system;
4. Repeat this procedure at the next sampling instant.

As stated previously, in step 2, the optimization problem solved by MPC is able to take into account constraints regarding system limitations [48]. These constraints can be, for example, constraints regarding input actuation limitations, safe working of the system, and many others. When comparing MPC to classical control techniques, it is noticeable that taking the system's constraints into the control algorithm is one of the greatest advantages of MPC. The main downside of MPC is the computational power required to compute the optimal solution: for a nonlinear problem, high computational power is required for solving the MPC problem, thus, the applicability of MPC is heavily dependent on the available hardware to control the system.

2.1.1 Optimization Problem

As stated in the previous section, a correct formulation of the MPC optimization problem is a fundamental step to develop a MPC to achieve the desired system behavior. Consider a generic nonlinear discrete-time system with the following state transition dynamics:

$$\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \tag{2.1}$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is C^1 and the states, $\mathbf{x}$ and inputs, $\mathbf{u}$ are subject to constraints:

$$\mathbf{x}_k \in \mathbb{X} \subseteq \mathbb{R}^n, \quad \mathbf{u}_k \in \mathbb{U} \subseteq \mathbb{R}^m, \quad \forall k \in \mathbb{Z}^+ \tag{2.2}$$

Considering that the controller predicts $N > 0$ system steps into the future i.e. the prediction horizon is N, and $J : \mathbb{R}^n \times \mathbb{R}^{m \times N} \to \mathbb{R}$ the cost function, generally defined as:

$$J_{t \to t+N} = l_{t+N}(\mathbf{x}_{t+N}) + \sum_{k=t}^{t+N-1} l_k(\mathbf{x}_k, \mathbf{u}_k) \tag{2.3}$$

where l_k represents a general stage cost at time instant $k \in [t, t+N]$. Note that the cost of the last stage, l_{t+N} is considered to only depend on $\mathbf{x}_{t+N}$ since the input at the prediction horizon, $\mathbf{u}_{t+N}$, would only have an impact at the time instant $t + N + 1$, which is outside the considered prediction horizon.

From above mentioned definitions, it is possible to write the general MPC optimization problem as follows:

$$\min_{\mathbf{u}_k, \forall k \in [t, t+N-1]} \quad J_{t \to t+N} \tag{2.4a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k) \quad \forall k \in [t, ..., t+N-1] \tag{2.4b}$$

$$\mathbf{x}_t = \mathbf{x}(t) \tag{2.4c}$$

$$\mathbf{u}_k \in \mathbb{U} \quad \forall k \in [t, ..., t+N-1] \tag{2.4d}$$

$$\mathbf{x}_k \in \mathbb{X} \quad \forall k \in [t, ..., t+N] \tag{2.4e}$$

where $\mathbf{u}_k$, $k = t, ..., t+N-1$ represents the sequence of the next N optimal inputs to be determined. Equation (2.4b) represents an equality constraint which translates the state prediction model used by the MPC. Equation (2.4c) represents the initialization of the MPC problem: the current measured state of the system, $\mathbf{x}(t)$, is assigned to the first element of the sequence of N predicted states of the system. Equations (2.4d) and (2.4e) represent the (possibly nonlinear) equality or inequality constraints of the inputs and states of the system, respectively.

After solving this optimization problem, the optimal inputs, $\mathbf{u}_k^*$, $k = t, ..., t+N-1$, are obtained, but only the first optimal input, $\mathbf{u}_t^*$, is applied to the system until a new sampling instant occurs.

2.2 Supervised Learning

Supervised Learning (SL) is inserted in the broad topic of Machine Learning (ML). In Supervised Learning, the learning process relies on a training set which consists of a set of labeled examples. This technique is most commonly associated with classification, regression, and ranking problems [49]. In this thesis, Supervised Learning will be used in the context of regression.

Further expliciting the Supervised Learning general problem, consider the training set with N training examples of the form $\{(\mathbf{v}_1, \mathbf{y}_1), ..., (\mathbf{v}_N, \mathbf{y}_N)\}$. In this training set $\mathbf{v}_i$ is commonly known as the feature vector of the i^{th} example in the set and $\mathbf{y}_i$ the corresponding target output vector. The objective of SL is to find a learner function $f : \mathbb{R}^{n_\mathbf{v}} \to \mathbb{R}^{n_\mathbf{y}}$, $n_\mathbf{v}$ being the dimension of the feature vector and $n_\mathbf{y}$ being the dimension of the target vector, that learns the training set. In other words, the objective is to find the function f such that $\hat{\mathbf{y}}_i \approx \mathbf{y}_i$, being $\hat{\mathbf{y}}_i = f(\mathbf{v}_i)$ the learner function output for a given feature vector in the training set.

In order to find the learner function, f, an algorithm for minimizing a loss function over the training set is employed. The loss function can be generally defined as a function $L : \mathbb{R}^{n_\mathbf{y}} \times \mathbb{R}^{n_\mathbf{y}} \to \mathbb{R}$ that approaches a minimum value as the predicted output, $\hat{\mathbf{y}}_i$, converges to the target output, $\mathbf{y}_i$, i.e. $\hat{\mathbf{y}}_i \to \mathbf{y}_i$ and tends to infinity as the predicted output diverges from the the target output.

As such, the learner function is generally obtained from solving an optimization problem which minimizes the loss function over the training data, obtaining the trained learner function, f^*, as the solution of:

$$f^* = \arg\min_f \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, f(\mathbf{v}_i)) \tag{2.5}$$

The minimization problem presented in equation (2.5) is commonly referred to in the literature as *Empirical Risk Minimization* (ERM) [49].

Regarding model selection, the act of choosing the inner workings and the structure of the learner function, f, one can consider two major model categories to choose from: parametric models and non parametric models.

- **Parametric models:** are characterized by having a fixed finite number of parameters;
- **Non parametric models:** are characterized by having an infinite number of parameters.

The main advantage of non parametric models over parametric models is that non parametric models typically offer higher flexibility, in the sense that they are capable of fitting a large number of function forms. However, non parametric models typically require more data to train [50].

For the purpose of model correction learning, presented in section 3.6 - Error Correction Model, parametric models were chosen for the reasons presented therein. Moreover, Artificial Neural Networks (ANN) were chosen as the learner function. The next section introduces the general definition of an ANN architecture, then, Elliptical Basis Function Networks (EBFNs), a type of ANN, are introduced.

2.2.1 Artificial Neural Networks

In this section, single-layer Artificial Neural Networks (ANNs) will be introduced. Artificial Neural Networks are inspired by biological neural networks as seen in the neurological system of animals, which consist of a highly connect neuron network [51]. Similar to biological neural networks, ANNs are also composed of several artificial neurons which are connected to each other. An illustrative example of a shallow Artificial Neural Network architecture can be seen in figure 2.2. In this figure, neurons are represented by the blue dots and their connections are represented by gray lines.

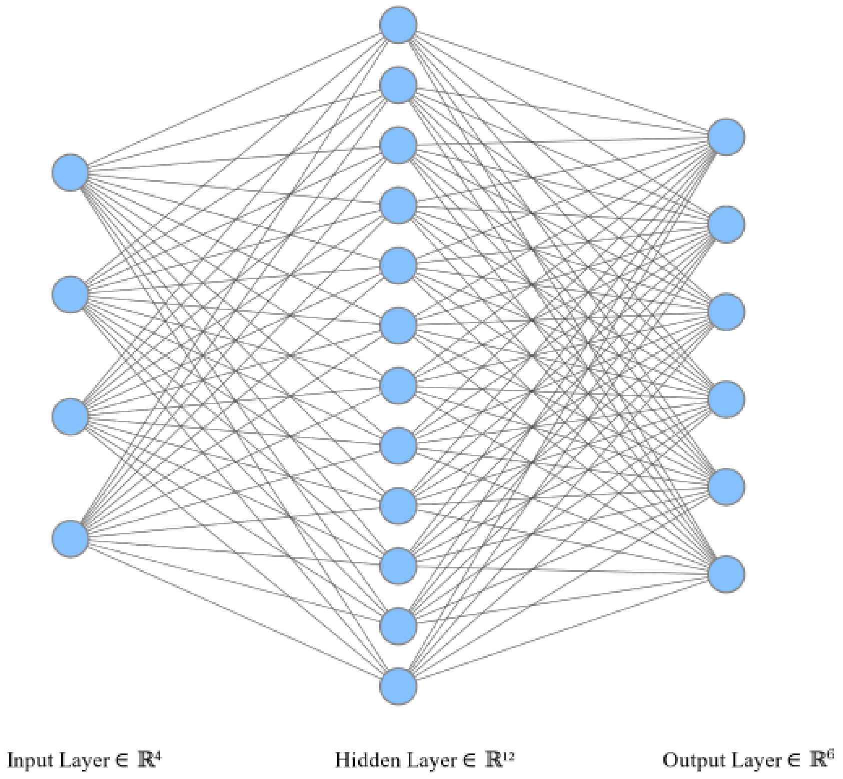

Input Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^{12}$
Output Layer $\in \mathbb{R}^6$

Figure 2.2: Example neural network with 4 input features, 12 hidden neurons and 6 outputs.

As suggested by figure 2.2, these networks consist of three main layers:

- **The input layer**: where the feature vector is fed as an input to the network;
- **The hidden layer**: where the neuron activation function is some nonlinear function of feature vector;
- **The output layer**: where the output of the network is, generally, computed as a biased linear combination of the hidden layer activations.

In general, the connections between these neurons are mathematically represented by a weight and the neurons themselves represent a nonlinear function of the previous layer. The ANN architecture chosen for this dissertation is Elliptical Basis Function Networks which will be introduced in the following section.

2.2.2 Elliptical Basis Function Networks

Elliptical Basis Function Networks (EBFNs) can be described as a generalization of the more commonly known Radial Basis Function Networks (RBFNs) [52]. These networks, which have an input layer, one hidden layer, and an output layer are closely related to clustering: a neuron i in the hidden layer represents a data cluster that is parameterized by its cluster center and its variance. The activation function of the hidden neurons is radial in the RBFN case and elliptical in the EBFN case. Mathematically, consider the previously mentioned $\mathbf{v}$ as the input vector, $\boldsymbol{\mu}_i$ the cluster center and $\boldsymbol{\Sigma}_i^{-1}$ the inverse of the positive definite covariance matrix of neuron i in the hidden layer (i.e. the covariance matrix with respect to the cluster center $\boldsymbol{\mu}_i$). The activation of neuron i in the hidden layer, h_i, is computed through the following equation:

$$h_i = \exp\Big(-\left(\mathbf{v}-\boldsymbol{\mu}_i\right)^T \boldsymbol{\Sigma}_i^{-1}\left(\mathbf{v}-\boldsymbol{\mu}_i\right)\Big) \quad , \quad i = 1, ..., n_h \tag{2.6}$$

As a further simplification, the covariance matrix of neuron i, $\boldsymbol{\Sigma}_i$ is approximated by a diagonal matrix, i.e. $\boldsymbol{\Sigma}_i \approx \operatorname{diag}\left(\sigma_{i,1}, ..., \sigma_{i,n_v}\right)$. In other words, this approximation considers that the components of $\mathbf{v}-\boldsymbol{\mu}_i$ are independent, since the off diagonal terms of this matrix are zero. For notation purposes, consider the matrix $\boldsymbol{\Omega}_i$ as the inverse of $\boldsymbol{\Sigma}_i$, commonly referred as the precision matrix. Given that $\boldsymbol{\Sigma}_i$ is diagonal, $\boldsymbol{\Omega}_i$ is also approximated by a diagonal matrix, $\boldsymbol{\Omega}_i \approx \operatorname{diag}\left(\omega_{i,1}, ..., \omega_{i,n_v}\right)$. Also given that $\boldsymbol{\Sigma}_i$ is positive definite, its inverse, $\boldsymbol{\Omega}_i$ must also be positive definite. As a result, the eigenvalues of $\boldsymbol{\Omega}_i$ must be positive, i.e. $\omega_{i,l} > 0$, $l = 1, ..., n_v$, $i = 1, ..., n_h$. For further notation, the subset of the parameter vector that regards the eigenvalues of the clusters' precision matrices will be denoted as $\boldsymbol{\omega} \subset \mathbf{p}$. This simplification can be introduced in equation (2.6) resulting in:

$$h_i = \exp\Big(-\left(\mathbf{v}-\boldsymbol{\mu}_i\right)^T \boldsymbol{\Omega}_i\left(\mathbf{v}-\boldsymbol{\mu}_i\right)\Big) \quad , \quad i = 1, ..., n_h \tag{2.7}$$

Regarding the output layer, the output is composed of a linear combination of the hidden layer activations. Mathematically, consider the previously mentioned output of the hidden layer, h_i, in its vector format, $\mathbf{h}$, where component i of this vector corresponds to activation i in the hidden layer, h_i, and $\mathbf{w}_j$ the weights' vector of neuron j in the output layer. Then, output j, e_j, is given by:

$$e_j = \mathbf{w}_j \cdot \mathbf{h} \quad , \quad j = 1, ..., n_e \tag{2.8}$$

Note that, taking into consideration equation (2.7), it is possible to intuitively understand how a cluster affects the behavior of these networks. For an input vector near a cluster center, $\mathbf{v} \approx \mu_i$, the activation value of that neuron converges to its maximum value, $h_i \to 1$. In other words, the hidden layer neurons are more active when the input vector is near the cluster centers and inactive far from these. As such, if a given input vector is far from all cluster centers, all neuron activations tend to zero, i.e., $h_i \to 0$, $i = 1, ..., n_h$. Further taking into account equation (2.8), as the activations tend to zero, the error estimation (output) of the network will also tend to zero, $h_i \to 0 \implies e_i \to 0$. In other words, such network architecture ensures that the error is not extrapolated for input data regions where no learning occurred.

Note also that the $\mathbf{\Omega}_i$ matrix is responsible for shaping the ellipse according to its eigenvalues. This matrix is the main difference between RBFN and EBFN: as the name suggests, RBFN has a circular neuron activation region instead of an elliptical neuron activation region, which in turn allows EBFN to potentially ensure better data fitting at the cost of some more parameters when compared to RBFN.

Further analyzing the number of required parameters to model a network with n_v input features, n_h neurons in the hidden layer, and n_e outputs, it can be observed that a neuron in the hidden layer needs n_v inverse covariance matrix parameters and n_v cluster center parameters, thus $2 \cdot n_v$ parameters, and a neuron in the output layer needs n_h parameters for the output weights. Finally, given the number of input features, neurons in the hidden layer, and outputs, the total number of parameters required by this type of network is given by: $n_p = 2 \cdot n_h \cdot n_v + n_e \cdot n_h$.

Regarding the training of these networks, Man-Wai Mak and Sun-Yuan Kung [53] propose a method which, in summary, applies a clustering algorithm to the input data, estimates the variance of the data with respect to the cluster centers, and extracts the diagonal of $\mathbf{\Sigma}_i$, $i = 1, ..., n_h$. With the hidden layer parameters estimated, linear least squares are used to estimate the output layer parameters given the linear relation between the network output and the hidden layer activations, as per equation (2.8).

2.2.3 Online Gradient Descent

In this section, the Online Gradient Descent training algorithm will be introduced. However, in order to introduce this training algorithm, the more commonly known Gradient Descent (GD) algorithm will be firstly presented. Consider that the adaptive learning function, f, which is dependent on parameters represented by a parameter vector, $\mathbf{p}$. Using this notation, the predicted output of the learner function given parameters $\mathbf{p}$ and feature vector $\mathbf{v}$ is computed as $\hat{\mathbf{y}} = f_{\mathbf{p}}(\mathbf{v})$. Recall the ERM problem presented in equation (2.5). Considering a fixed structure of the learner function, $f_{\mathbf{p}}$, for example, by using an EBFN as previously presented, this problem can now be modified in order to optimize the learning parameters, $\mathbf{p}$, as follows:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} \frac{1}{N} \sum_{i=1}^{N} L\left(\mathbf{y}_i, f_{\mathbf{p}}\left(\mathbf{v}_i\right)\right) \tag{2.9}$$

Consider $R_{\mathbf{p}}$ the *Empirical Risk* of the previous equation, i.e. the term that is optimized:

$$R_{\mathbf{p}} = \frac{1}{N} \sum_{i=1}^{N} L\left(\mathbf{y}_i, f_{\mathbf{p}}\left(\mathbf{v}_i\right)\right) \tag{2.10}$$

The GD algorithm's approach considers a learning step that is attained by computing the gradient of the ERM problem. In other words, given parameters $\mathbf{p}$, the GD algorithm will take a learning step according to the steepest direction with the purpose of minimizing $R_{\mathbf{p}}$. This direction can be computed by taking the gradient of $R_{\mathbf{p}}$ with respect to the parameter vector, $\mathbf{p}$. This direction is mathematically defined as $-\nabla_{\mathbf{p}} R_{\mathbf{p}}$, where the operator $\nabla_{\mathbf{p}}$ represents the gradient operation with respect to $\mathbf{p}$. The components of the gradient operator are computed as $\left[\nabla_{\mathbf{p}}\right]_i \equiv \frac{\partial}{\partial p_i}$, i.e., the i^{th} component of the gradient is computed as a derivative with respect to component i of the parameter vector. As such, the parameter update equation of the GD algorithm is defined as:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \eta \nabla_{\mathbf{p}} R_{\mathbf{p}_k} \tag{2.11}$$

In this equation, the index k represents the current time instant and $k+1$ the next time instant. As such, the terms $\mathbf{p}_k$ and $\mathbf{p}_{k+1}$ can be read as the current and next parameter vectors, respectively. Furthermore, the parameter $\eta \geq 0$ represents the learning rate. This parameter controls the step size of the learning algorithm. The choice of η often depends on the problem, but, in general, a large value of the learning rate results in faster learning but coarser parameter tuning, and a small value results in slower learning but finer parameter tuning.

Note, however, that when large training sets are used, i.e., N is large, computing the *Empirical Risk* gradient becomes computationally exhaustive as N increases, as shown by the following equation:

$$\nabla_{\mathbf{p}} R_{\mathbf{p}} = \nabla_{\mathbf{p}} \frac{1}{N} \sum_{i=1}^{N} L\left(\mathbf{y}_i, f_{\mathbf{p}}\left(\mathbf{v}_i\right)\right) = \frac{1}{N} \sum_{i=1}^{N} \nabla_{\mathbf{p}} \left[L\left(\mathbf{y}_i, f_{\mathbf{p}}\left(\mathbf{v}_i\right)\right)\right] \tag{2.12}$$

Furthermore, in an online learning scenario, the training data becomes available iteratively, i.e., as time passes, more and more data becomes available leading to N increasing. For this reason, the Online Gradient Descent algorithm approximates the gradient of the *Empirical Risk* as the gradient of the loss function with respect to the most recent data point. As such, the OGD parameter update equation is defined as follows:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \eta \nabla_{\mathbf{p}} \left[L\left(\mathbf{y}_k, f_{\mathbf{p}_k}\left(\mathbf{v}_k\right)\right)\right] \tag{2.13}$$

Algorithm 1 summarizes the idea behind the OGD algorithm.

Algorithm 1 Online Gradient Descent update step

```
1: procedure OGD($\mathbf{v}_k$, $\mathbf{y}_k$, $\mathbf{p}_k$, $\eta$, $f_{\mathbf{p}}(\cdot)$, $L(\cdot)$)    ▷ Input data for the method
2:     $\hat{\mathbf{y}} \leftarrow f_{\mathbf{p}_k}(\mathbf{v}_k)$    ▷ Predict the expected target
3:     $\Delta\mathbf{p} \leftarrow -\eta \nabla_{\mathbf{p}} \left[L(\mathbf{y}_k, \hat{\mathbf{y}})\right]$    ▷ Compute parameter increment
4:     $\mathbf{p}_{k+1} \leftarrow \mathbf{p}_k + \Delta\mathbf{p}$    ▷ Compute new parameters as per equation (2.13)
5:     return $\mathbf{p}_{k+1}$    ▷ Output next parameter vector
6: end procedure
```

2.2.4 Levenberg-Marquardt Algorithm

In this section, the Levenberg-Marquardt algorithm will be introduced. The Levenberg-Marquardt (LM) algorithm, also known as damped least-squares is an optimization algorithm that was first suggested by Levenberg (1944) and by Marquardt (1963) in the context of nonlinear least-squares optimization. This algorithm interpolates between other known algorithms known as the Gauss-Newton algorithm and the previously mentioned Gradient Descent (GD) algorithm. Since this method is used for solving nonlinear least squares function, the ERM problem, as per equation (2.9), must be in general expressed as a sum of squares. In order to introduce one possible ERM problem that may be solved using the LM algorithm, consider a target in the training set, $\mathbf{y}_i$, and the corresponding prediction, $\hat{\mathbf{y}}_i = f_{\mathbf{p}}\left(\mathbf{v}_i\right)$, where the index i ranges for every training set data point, i.e., $i = 1, ..., N$. The sum squared error (SSE) of a target in the training set is defined as:

$$\mathrm{SSE}_i = \sum_{j=1}^{n_e} \left([\mathbf{y}_i]_j - [\hat{\mathbf{y}}_i]_j \right)^2 \tag{2.14}$$

where n_e is the dimension of vector $\mathbf{y}_i$, or, in other words, the dimension of the targets in the training set. With the SSE defined, the ERM problem to be considered can be applied. The *Empirical Risk*, R, can then be defined as:

$$R_{\mathbf{p}} = \sum_{i=1}^{N} \mathrm{SSE}_i = \sum_{i=1}^{N} \sum_{j=1}^{n_e} \left([\mathbf{y}_i]_j - [\hat{\mathbf{y}}_i]_j \right)^2 \tag{2.15}$$

Similar to the previously presented algorithm, the Online Gradient Descent (OGD) algorithm in section 2.2.3, the LM's objective is to find the optimal parameters, $\mathbf{p}^*$, as the solution of the following optimization problem:

$$\mathbf{p}^* = \arg\min_{\mathbf{p}} R_p \tag{2.16}$$

To further define the equations used by the LM algorithm, consider the jacobian matrix, $\mathbf{J}_f^i$, of the learner function referring to data point $i = 1, ..., N$, which component $\left[\mathbf{J}_f^i\right]_{l,j}$, where the indexes $l = 1, ..., n_p$ and $j = 1, ..., n_e$ represent a row and a column, respectively, can be computed as $\left[\mathbf{J}_f^i\right]_{l,j} = \frac{\partial [f_{\mathbf{p}}(\mathbf{v}_i)]_j}{\partial p_l}$. Note also that, considering a small parameter variation, $\Delta\mathbf{p}$, the new value of the ERM problem with new parameters added by this variation, $\mathbf{p} + \Delta\mathbf{p}$, can be approximated by a linearization given by:

$$R_{\mathbf{p}+\Delta\mathbf{p}} \approx \sum_{i=1}^{N} \sum_{j=1}^{n_e} \left([\mathbf{y}_i]_j - [f_{\mathbf{p}}(\mathbf{v}_i)]_j - \left[\mathbf{J}_f^i\right]^T \Delta\mathbf{p} \right)^2 \tag{2.17}$$

Taking the derivative of $R_{\mathbf{p}+\Delta\mathbf{p}}$ with respect to the parameter increment, $\Delta\mathbf{p}$, and setting the result to be a null vector, results in the Gauss-Newton algorithm equation:

$$\left(\mathbf{J}_R \mathbf{J}_R{}^T \right) \Delta\mathbf{p} = \mathbf{J}_R \left(\mathbf{y}_R - \hat{\mathbf{y}}_R \right) \tag{2.18}$$

where $\mathbf{J}_R$ represents the jacobian matrix of the ERM problem. The vectors $\mathbf{y}_R$ and $\hat{\mathbf{y}}_R$ result from the concatenation of the training set targets and current predictions as computed by f_p, respectively. Mathematically, these variables are assembled as follows:

$$\mathbf{J}_R = \begin{bmatrix} \mathbf{J}_f^1 & \cdots & \mathbf{J}_f^N \end{bmatrix} \tag{2.19a}$$

$$\mathbf{y}_R = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \tag{2.19b}$$

$$\hat{\mathbf{y}}_R = \begin{bmatrix} \hat{\mathbf{y}}_1 \\ \vdots \\ \hat{\mathbf{y}}_N \end{bmatrix} \tag{2.19c}$$

The main difference between the Gauss-Newton algorithm is that the LM algorithm adds a *damping* term to the Gauss-Newton equation, presented in equation (2.18). Thus, the LM algorithm equation can be written as follows:

$$\left(\mathbf{J}_R{\mathbf{J}_R}^T + \lambda\mathbf{I}\right)\Delta\mathbf{p} = \mathbf{J}_R\left(\mathbf{y}_R - \hat{\mathbf{y}}_R\right) \tag{2.20}$$

The learning parameter $\lambda > 0$ is known as the damping parameter in the LM algorithm. A larger value of λ results in taking a step in the steepest direction, i.e., the algorithm behaves like the Gradient Descent (GD) algorithm and a smaller value of λ results in taking a step towards the Gauss-Newton step. Note that for large values of λ the damping parameter term $\lambda\mathbf{I}$ becomes much larger than $\mathbf{J}_R{\mathbf{J}_R}^T$. As a result, $\mathbf{J}_R{\mathbf{J}_R}^T + \lambda\mathbf{I} \approx \lambda\mathbf{I}$ and by of solving equation (2.20), $\Delta\mathbf{p} \approx \frac{1}{\lambda}\mathbf{J}_R\left(\mathbf{y}_R - \hat{\mathbf{y}}_R\right)$. Thus, the LM's damping parameter is analogous to the inverse of the learning rate parameter of the GD's algorithm.

Furthermore, note that in equation (2.20), the matrix $\lambda\mathbf{I} + \mathbf{J}_R{\mathbf{J}_R}^T$ is guaranteed to be positive definite as $\lambda\mathbf{I} > 0$ and $\mathbf{J}_R{\mathbf{J}_R}^T \geq 0$. Justifying, $\lambda\mathbf{I}$ is positive definite as all the eigenvalues of this diagonal matrix are $\lambda > 0$ and $\mathbf{J}_R{\mathbf{J}_R}^T$ is positive semi definite as, given a non null vector $\mathbf{v}$, it can be concluded that $\mathbf{v}^T\left(\mathbf{J}_R{\mathbf{J}_R}^T\right)\mathbf{v} = \left({\mathbf{J}_R}^T\mathbf{v}\right)^T\left({\mathbf{J}_R}^T\mathbf{v}\right) \geq 0$ [54]. Thus, this linear system of equations can be solved using a Cholesky decomposition, with a computational complexity of $O(n^3/3)$ [55], which is computationally more efficient than inverting $\lambda\mathbf{I} + \mathbf{J}_R{\mathbf{J}_R}^T$, which has a computational complexity of $O(n^3)$.

The new parameters, $\mathbf{p}_{k+1}$, are obtained by solving for $\Delta\mathbf{p}$ equation (2.20) and adding this parameter increment to the previous parameter vector as follows:

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta\mathbf{p} \tag{2.21}$$

Algorithm 2 summarizes the idea behind the LM algorithm for minimizing the ERM problem presented in equation (2.15).

2.3 Reinforcement Learning

Reinforcement Learning (RL) is inserted in the broad topic of machine learning. Unlike the previously mentioned learning scheme of Supervised Learning, introduced in section 2.2, the learner does not passively receive a labeled data set. Instead, it collects information through a course of actions by interacting with the environment. As a result of performing an action, the learner, also known as the *agent* in RL, generally receives two types of information: its current state in the environment and a reward, which is specific to the learner's task or goal [49]. Figure 2.3 represents the general learning scheme of RL.

Algorithm 2 Levenberg-Marquardt update step

```
 1: procedure LM($\mathbf{v}_{t=1,...,N}$, $\mathbf{y}_{t=1,...,N}$, $\mathbf{p}_k$, $N$, $n_p$, $n_e$, $\lambda$, $f_{\mathbf{p}}(\cdot)$)    ▷ Input data for the method
 2:     for integer $t = 1$ to $t = N$ do    ▷ Iterate over training set
 3:         $\hat{\mathbf{y}}_t \leftarrow f_{\mathbf{p}_k}(\mathbf{v}_t)$    ▷ Predict the expected target
 4:         initialize $\mathbf{J}_f^t \in \mathbb{R}^{n_p \times n_e}$    ▷ Allocate memory space for the learner function jacobian
 5:         for integer $i = 1$ to $i = n_p$ do
 6:             for integer $j = 1$ to $i = n_e$ do
 7:                 $\left[\mathbf{J}_f^t\right]_{i,j} \leftarrow \frac{\partial\left[f_{\mathbf{p}_k}(\mathbf{v}_t)\right]_j}{\partial p_i}$    ▷ Compute the jacobian components
 8:             end for
 9:         end for
10:     end for
11:     $\mathbf{J}_R \leftarrow \begin{bmatrix}\mathbf{J}_f^1 & \cdots & \mathbf{J}_f^N\end{bmatrix}$    ▷ Assemble *Empirical Risk* jacobian as per equation (2.19a)
12:     $\mathbf{y}_R \leftarrow \begin{bmatrix}\mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N\end{bmatrix}$    ▷ Assemble target's vector as per equation (2.19b)
13:     $\hat{\mathbf{y}}_R \leftarrow \begin{bmatrix}\hat{\mathbf{y}}_1 \\ \vdots \\ \hat{\mathbf{y}}_N\end{bmatrix}$    ▷ Assemble prediction's vector as per equation (2.19c)
14:     $\Delta\mathbf{p} \leftarrow$ Cholesky solve $\left(\mathbf{J}_R{\mathbf{J}_R}^T + \lambda\mathbf{I}\right)\Delta\mathbf{p} = \mathbf{J}_R(\mathbf{y}_R - \hat{\mathbf{y}}_R)$    ▷ Obtain $\Delta\mathbf{p}$ as per equation (2.20)
15:     $\mathbf{p}_{k+1} \leftarrow \mathbf{p}_k + \Delta\mathbf{p}$    ▷ Compute new parameters as per equation (2.21)
16:     return $\mathbf{p}_{k+1}$    ▷ Output next parameter vector
17: end procedure
```

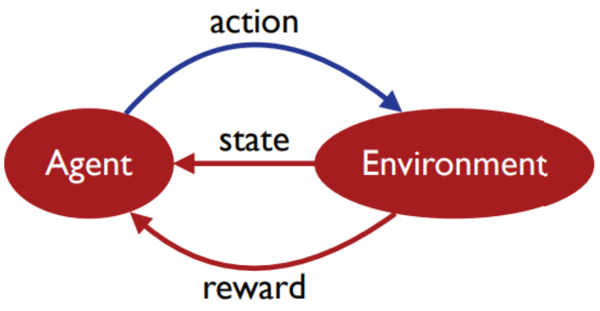


Figure 2.3: General learning scheme of Reinforcement Learning [49].

In RL, the learner's objective is to maximize its received reward, determining which is the best course of action, or *policy*, to achieve that objective.

The learner is faced with the dilemma between exploring unknown states and actions to gain more information about the environment and the achievable rewards, and exploiting the already gathered information to optimize its reward. In RL this is known as the exploration versus exploitation trade-off.

In order to further define Reinforcement Learning, the model of Markov Decision Processes (MPDs), a model of the environment and interactions with the environment, will be introduced. MDPs represent a wide range of models which can represent various real world tasks. RL algorithms are based on discrete time MDPs for a time horizon of $t \in \{0, ..., T\}$ which can essentially be defined by the:

- Set of states, $\mathbb{S}$, possibly infinite;
- Initial state, $\mathrm{s}_0 \in \mathbb{S}$;

- Set of actions, $\mathbb{A}$, possibly infinite;
- Transition probability, $\mathrm{P}\left(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t\right)\ \forall \mathbf{a}_t \in \mathbb{A}$, $\mathbf{s}_t$, $\mathbf{s}_{t+1} \in \mathbb{S}$: distribution over future states, $\mathbf{s}_{t+1} = \delta\left(\mathbf{s}_t,\mathbf{a}_t\right)$;
- Reward probability, $\mathrm{P}\left(r_{t+1}|\mathbf{s}_t,\mathbf{a}_t\right)\ \forall \mathbf{a}_t \in \mathbb{A}$, $\mathbf{s}_t$, $\mathbf{s}_{t+1} \in \mathbb{S}$: distribution over expected future rewards, $r_{t+1} = r\left(\mathbf{s}_t,\mathbf{a}_t\right)$.

At time t the state $\mathbf{s}_t$ is observed by the agent and an action, $\mathbf{a}_t$, is taken. At the next time instant the state $\mathbf{s}_{t+1}$, associated with probability $\mathrm{P}\left(\mathbf{s}_{t+1}|\mathbf{s}_t,\mathbf{a}_t\right)$, is observed and a reward, $r_{t+1} \in \mathbb{R}$, is received, also associated with probability $\mathrm{P}\left(r_{t+1}|\mathbf{s}_t,\mathbf{a}_t\right)$. This process is illustrated by figure 2.4.

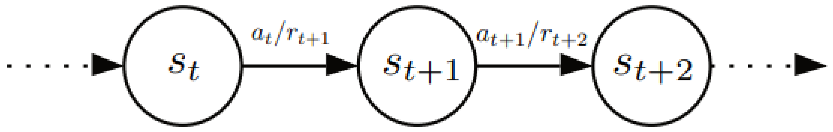


Figure 2.4: Scheme of states and transitions of a MDP at different times [49].

The main challenge for an agent in a MDP environment is to determine the action to take at each state. Formally, a policy is a mapping $\pi_t : \mathbb{S} \to \Delta\left(\mathbb{A}\right)$, where $\Delta\left(\mathbb{A}\right)$ is the set of probability distributions over $\mathbb{A}$. A policy π_t is deterministic if an unique $\mathbf{a}_t \in \mathbb{A}$ exists such that $\pi_t\left(\mathbf{s},\mathbf{a}\right) = 1$. In that case, π_t can be identified with a mapping from $\mathbb{S}$ to $\mathbb{A}$. As mentioned previously, the agent's objective is to find a policy that maximizes its expected reward, called the optimal policy.

The value of a policy, V_π, at state $\mathbf{s} \in \mathbb{S}$ for a finite horizon, T, along a specific sequence of states $\mathbf{s}_0, ..., \mathbf{s}_T$ is generally defined as the expected reward returned when starting at $\mathbf{s}_0 = \mathbf{s}$ and following policy π:

$$V_\pi\left(\mathbf{s}\right) = \mathbb{E}\left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \pi\left(\mathbf{s}_t\right)\right)\middle|\mathbf{s}_0 = s\right] \tag{2.22}$$

The definitions regarding RL presented in this section are enough for understanding the methods developed in this dissertation. For further reading on this topic, please refer to the book by Mohri et al. [49].

Classic RL algorithms are, for example, Q-Learning, Policy iteration, Value Iteration, as well as Monte Carlo algorithms [56]. Nevertheless, gradient based [57] and evolutionary [58] methods have been employed in a RL framework. In the next section an evolutionary optimization method - the Genetic Algorithm (GA) - will be presented in a RL problem solving framework.

2.3.1 Genetic Algorithm

In this section, the Genetic Algorithm (GA) will be presented while focusing on using GAs for solving a Reinforcement Learning problem.

GAs are derivative free metaheuristic algorithms based on the mechanics of natural selection and natural genetics. These algorithms combine survival of the fittest among a population with a structured yet randomized information exchange for a search algorithm. The main idea behind this algorithm is that at each new generation, a new set of individuals (denoted by population) is generated using bits and pieces of the fittest individuals of the previous generation, hoping that the new generation contains an overall fitter population when compared to the previous generation individuals. An individual is essentially represented by a string of genetic code that, in turn, translates to certain traits and fitness. The new generation individuals can be generated according to three main genetic operations [59]:

1. **Reproduction**: some individuals of the previous generation are randomly selected according to their fitness values. If an individual is selected, its genetic code is passed directly onto the new generation without changes in its genetic code;
2. **Crossover**: pairs of individuals of the previous generation are randomly selected and parts of their genetic code are swapped, generating two new children that are passed onto the new generation;
3. **Mutation**: some individuals of the previous generation are randomly selected according to their fitness values. If an individual is selected, part of its genetic code is randomly changed and the mutated individual is then passed onto the new generation.

These three genetic operations have three different distinct purposes. Moreover, the reproduction operation aims at retaining the, so far, best individuals of the previous generation so that their genetic code does not get randomly lost in case the newly generated individuals are genetically worse than the previous generation individuals. The crossover operation aims at generating new individuals that retain portions of the genetic code of each parent, hoping that the generated children's genetic code is better than the individual parent's genetic code. The mutation operation consists of generating new individuals by randomly mutating a piece of an individual's genetic code, expecting that the mutated individual is genetically better than the original one. It can be further noted that the crossover operation generally exploits the genetic code space whereas the mutation operation generally explores the genetic code space. In other words, crossovers tend to improve (potentially local) optimality points whereas mutations can escape local optimality points. Figure 2.5 schematically represents these three genetic operations, in this case, using binary strings that represent the individuals' genetic code.

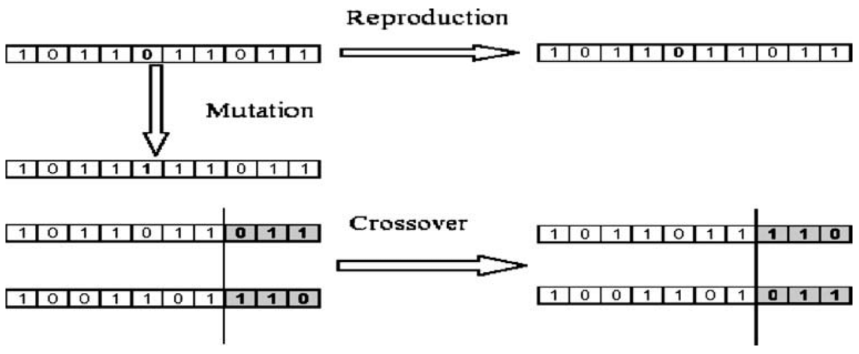


Figure 2.5: Visual representation of the three different genetic operations [60].

To mathematically define the inner workings of the Genetic Algorithm, consider an optimization problem that, for example, reflects the reward maximization problem presented in the previous section regarding Reinforcement Learning (RL), section 2.3. One can generally define this problem as follows:

$$\max_{\mathbf{d}} r\left(\mathbf{d}\right) \tag{2.23}$$

Equation (2.23) reflects an optimization problem which purpose is to maximize a reward function denoted as r. This reward function is slightly different from the expected reward function defined in equation (2.22) in the previous section, as the reward function presented in equation (2.23) reflects "how good" a genetic string $\mathbf{d} \in \mathbb{R}^{n_d}$ is for a given RL problem. For purposes that will be further explained consider a designed reward function that respects $r\left(\mathbf{d}\right) \geq 0 \; \forall \mathbf{d} \in \mathbb{R}^{n_d}$. This can be considered as the reward function is defined by the user according to the desired RL task. Furthermore, a population of N individuals can be defined by a matrix, $\mathbf{D} \in \mathbb{R}^{n_d \times N}$, in which component $d_{i,j}$ represents gene $i = 1, ..., n_d$ of individual $j = 1, ..., N$. As such, the columns of matrix $\mathbf{D}$ contain the population individuals. Given this, the population matrix can be defined as follows:

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \dots & \mathbf{d}_N \end{bmatrix} \tag{2.24}$$

The GA starts by defining an initial randomly generated population of N individuals. In other words, the initial generation matrix components are randomly generated over a given search space. The GA then obtains a reward value for each individual in the initial generation, i.e., let $\mathbf{r}$ represent the reward vector, its components $r_j \equiv r\left(\mathbf{d}_j\right)$ represent the reward obtained by using the individual's $j = 1, ..., N$ genes in the learning process. With the reward vector computed, the next task is to find a mechanism that maps the reward of each individual to the probability of the individuals being selected for a genetic operation. In other words, an individual is likely to be selected if its reward is a large value and unlikely to be selected if its reward value is small. Given this, a common mapping method to attain a selection probability distribution over each individual is the relative fitness. The relative fitness, $\mathbf{f}$, is essentially a normalization of the individual's reward. Note that the relative fitness of an individual represents its probability of being selected for a given genetic operation, thus, the components of $\mathbf{f}$, denoted by f_j, must respect the following properties:

$$0 \leq f_j \leq 1 \quad , \quad j = 1, ..., N \tag{2.25a}$$

$$\sum_{j=1}^{N} f_j = 1 \tag{2.25b}$$

Equation (2.25a) denotes that the probability of an individual being picked must be some value between it always being picked, 1, and never being picked, 0. Equations (2.25b) represent the probability of the whole set, i.e., an individual must always be picked from the population set.

In order to compute the relative fitness of each individual, a common approach is to map the reward of each individual to its relative fitness by normalizing the reward vector to the sum of its components. As such, the relative fitness components can be computed as follows:

$$f_j = \frac{r_j}{\sum_{n=1}^{N} r_n} \quad , \quad j = 1, ..., N \tag{2.26}$$

Note that the mapping method presented in equation (2.26) respects the properties in equation (2.25) since the reward function was previously considered to respect $r(\mathbf{d}) \geq 0 \; \forall \mathbf{d} \in \mathbb{R}^{n_d}$.

With the relative fitness of each individual defined, the number of each genetic operation to perform must be selected. Typically, the number of each operation to perform is also randomly selected, and, as a parameter to the Genetic Algorithm, there is typically a user defined probability associated with each operation. Let R, C and M denote the reproduction, crossover and mutation operations, respectively. The probability of each operation being picked must also be some value between zero and one, i.e., $0 \leq \mathrm{P}(\mathrm{R}) \leq 1$, $0 \leq \mathrm{P}(\mathrm{C}) \leq 1$, and $0 \leq \mathrm{P}(\mathrm{M}) \leq 1$. Moreover, an operation must always be picked between the three operations. As such, $\mathrm{P}(\mathrm{R}) + \mathrm{P}(\mathrm{C}) + \mathrm{P}(\mathrm{M}) = 1$. Further note that the balance between the exploitation and exploration problem inherent to the RL problem can be tuned by manipulating $\mathrm{P}(\mathrm{C})$ and $\mathrm{P}(\mathrm{M})$, as increasing $\mathrm{P}(\mathrm{C})$ and decreasing $\mathrm{P}(\mathrm{M})$ increases the exploitation behavior of the algorithm whereas decreasing $\mathrm{P}(\mathrm{C})$ and increasing $\mathrm{P}(\mathrm{M})$ increases the exploration behavior of the algorithm. After selecting N operations according to these probabilities and applying these operations to the initial population, a new generation is generated with new individuals. This process is then recursively repeated with the new generation, forming populations that, statistically, should be better than the previous ones.

Note, however, that the number of crossover operations selected at each algorithm iteration must be even, since a pair of parents is always required to generate the new children. Furthermore, the two selected parents must not be the same individual, since this would result in two children with the same genetic code as the parent. This would essentially be equivalent to a double reproduction of an individual in the old generation.

Algorithm 4 in appendix A summarizes the inner workings of the GA. This algorithm also presents a simple solution to the issue of the number of crossovers to be performed, presented in the last paragraph. This solution consists of, if the number of crossovers is odd, subtracting one to it and adding one to either the number of reproductions or mutations. For further insight on Genetic Algorithms and their applications, the author recommends the book by Goldberg [59].

Chapter 3

Vehicle Models

In this chapter, vehicle models that will be used in the MPC algorithm will be introduced.

Vehicles in general have complex underlying dynamics due to the presence of multiple subsystems. Representing these complex dynamics and the relationships between them is not an easy task. Furthermore, in a racing competition environment, the vehicle is expected to perform high-speed and high-acceleration maneuvers which may lead to certain simplifications being invalid, leading to more complex models than, for example, a regular traffic-driving scenario. In conclusion, modeling race cars is indeed a challenging task.

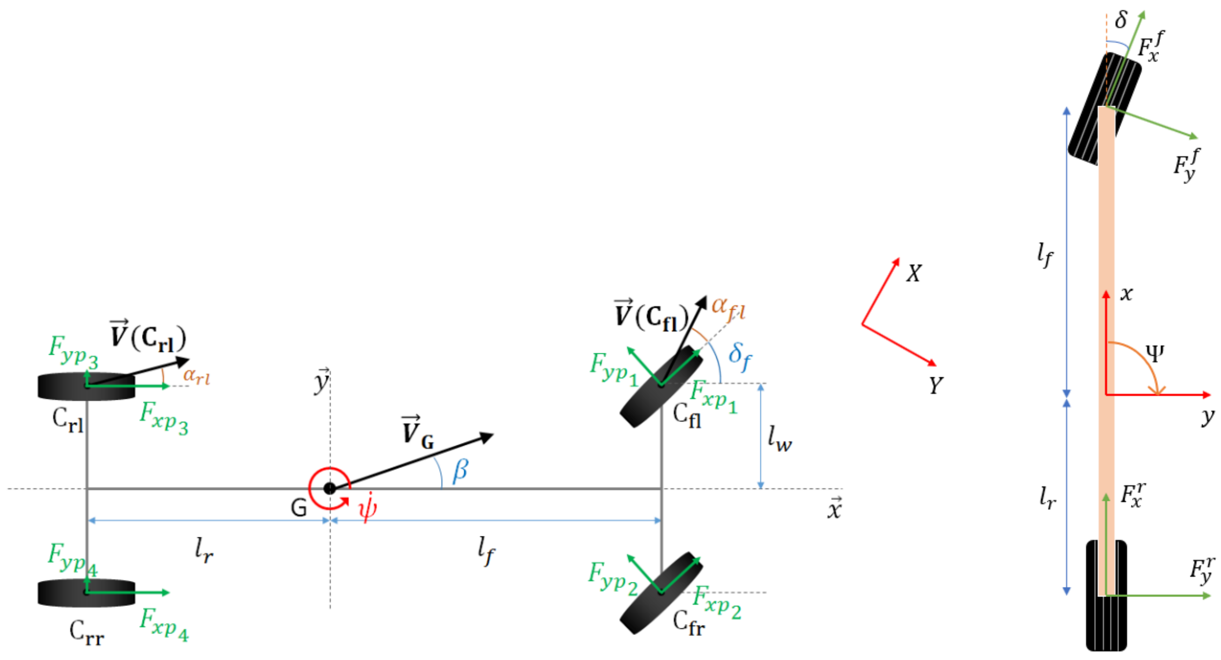
Despite the already mentioned complexity of modeling a racing car, the design of appropriate models represents the fundamental initial step of the design of a MPC. The degree of complexity and how well the model represents the real system dynamics will directly influence the performance of the controller. However, the use of higher complexity models is not always desired: in general, a well-modeled higher-order model better describes the system dynamics. Nevertheless, by including more variables than a simpler model, the MPC optimization problem will become more complex, resulting in a higher computation cost.

3.1 Vehicle Description

3.1.1 Vehicle Geometry

The vehicle that is to be modeled has two axles with two wheels each. Propulsion is achieved by two electric motors that apply torque to the rear axle wheels, thus accelerating the vehicle forward. Moreover, to steer the vehicle, the front wheels typically move according to Ackermann's steering geometry. A general free-body diagram that represents the main geometry of such a vehicle can be seen in figure 3.1a.

Despite being a better representation of the real vehicle to be modeled, the geometry shown in figure 3.1a will add unwanted complexity for control purposes. For this reason, a model with the geometry shown in figure 3.1b will be used. Such a model is commonly known as a bicycle model. The main purpose of this model is to represent the two wheels of the front and the two wheels of the rear of the real vehicle by a single wheel at the front and another at the rear. The steering angle of the front wheel is represented by δ. The distance from the Center of Gravity (CG) of the vehicle to the rear axle is denoted by l_r. Similarly, the distance from the CG to the front axle is denoted by l_f. The wheelbase of the vehicle can therefore be written as $L = l_r + l_f$.



(a) Full vehicle geometry.

(b) Bicycle model geometry.

Figure 3.1: Vehicle geometry comparison.

Note that in the bicycle model, in figure 3.1b, two frames of reference are represented: (X, Y) and (x, y) which respectively refer to the global fixed frame of reference i.e. a frame of reference that is fixed to the ground and a local frame of reference which is fixed to the vehicle's center of mass with the x coordinate aligned with the vehicle's longitudinal direction and the y coordinate aligned with the vehicle's lateral direction, respectively. Moreover, the orientation of the vehicle relative to the global frame of reference is represented by Ψ, often referred to as the vehicle Yaw angle. Ψ can be seen as the angle formed by the global X direction and the vehicle's heading direction, x. In order to rotate some vector, $\mathbf{v}$, from one frame to the other, a rotation matrix is employed. Moreover, to transform a vector in the vehicle's frame of reference, $\mathbf{v}_{(x,y)}$, to the global frame of reference, $\mathbf{v}_{(X,Y)}$, the following transformation can be used:

$$\mathbf{v}_{(X,Y)} = \mathbf{R}_{(x,y)}^{(X,Y)} \mathbf{v}_{(x,y)} \tag{3.1}$$

where:

$$\mathbf{R}_{(x,y)}^{(X,Y)} = \begin{bmatrix} \cos(\Psi) & -\sin(\Psi) \\ \sin(\Psi) & \cos(\Psi) \end{bmatrix} \tag{3.2}$$

As suggested from equation (3.1), $\mathbf{R}_{(x,y)}^{(X,Y)} \in \mathbb{R}^{2\times 2}$ is a rotation matrix that can be interpreted as a matrix that when multiplied on the left side of a vector in frame (x, y) it rotates that vector to frame (X, Y).

As this matrix is a rotation matrix, the inverse transformation, i.e. the matrix $\mathbf{R}_{(X,Y)}^{(x,y)}$ that transforms a vector expressed in frame (X, Y) to frame (x, y) can be easily computed by transposing $\mathbf{R}_{(x,y)}^{(X,Y)}$. Mathematically:

$$\mathbf{R}_{(X,Y)}^{(x,y)} = \left[\mathbf{R}_{(x,y)}^{(X,Y)}\right]^{-1} = \left[\mathbf{R}_{(x,y)}^{(X,Y)}\right]^{T} \tag{3.3}$$

Thus, the inverse transformation of equation (3.1) can be expressed as follows:

$$\mathbf{v}_{(x,y)} = \left[\mathbf{R}_{(x,y)}^{(X,Y)}\right]^{T} \mathbf{v}_{(X,Y)} \tag{3.4}$$

3.1.2 Lateral Tire Forces

The lateral forces that act on the lateral tire direction, represented in figure 3.1b by F_y^f and F_y^r for the front and rear wheels, respectively, are responsible for steering the vehicle and keeping the vehicle on track. To analyze these forces, the tire-slip angle definition is employed. This angle is defined as the difference between the wheel's steering angle and the wheel's direction of travel. A visual representation of this angle can be seen in figure 3.2.

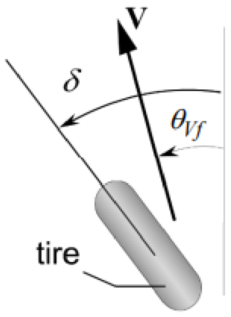


Figure 3.2: tire-slip model [61].

The tire-slip angle, α, can then be computed as follows:

$$\alpha = \theta_{Vf} - \delta \tag{3.5}$$

Using the previous equation, the front and rear wheels' slip angle can be computed as follows in equations (3.6), where α_f and α_r denote the tire-slip angles of the front and rear wheels [61]. In these equations, the angle θ_{Vf} can be computed from the velocities in the vehicle's frame of reference, (x, y), where v_x and v_y represent the longitudinal and lateral velocities of the vehicle in x and y, respectively, as well as the rate of change of the orientation of the vehicle, denoted as $\dot{\Psi}$. Note that, the rear wheels of the vehicle are considered not to steer, which results in $\delta_r = 0$.

$$\alpha_f = \text{arctan}\left(\frac{v_y + \dot{\Psi} l_f}{v_x}\right) - \delta \tag{3.6a}$$

$$\alpha_r = \text{arctan}\left(\frac{v_y - \dot{\Psi} l_r}{v_x}\right) \tag{3.6b}$$

With the tire-slip angles computed, a model for the lateral tire forces on each wheel can be applied as follows in equations (3.7) [42]. In this model, $D_{f/r}$ represents the maximum value of the lateral force for one tire, $C_{f/r}$ is a shape factor, $B_{f/r}$ is the stiffness factor of the tire and $F_y^{f/r}$ are the resultant forces applied on the front (f) and rear (r) wheels. These forces are applied contrary to the movement direction, hence the negative sign. Moreover, note also that the coefficient 2 that appears in equations (3.7) denotes the existence of two wheels in the front axle as well as two wheels in the rear axle, as in the real vehicle's geometry.

$$F_y^f = -2D_f \sin\left(C_f \text{arctan}\left(B_f \cdot \alpha_f\right)\right) \tag{3.7a}$$

$$F_y^r = -2D_r \sin\left(C_r \text{arctan}\left(B_r \cdot \alpha_r\right)\right) \tag{3.7b}$$

3.1.3 Longitudinal Forces

Regarding forces that are applied in the longitudinal axis of the vehicle, x, three main forces are considered: motor's propulsion, rolling resistance, and aerodynamic drag.

The motor's propulsion is the force that allows the vehicle to accelerate in its longitudinal direction, as it is commonly known. This propulsion force is provided by the torque generated by electric motors, which is then transferred to the wheels via the drivetrain. The electric motors, gearbox, and transmission are complex subsystems that require extensive knowledge about the internal workings of such components. Nevertheless, these components can be assumed to be controlled by a low-level controller. For this purpose, to model the propulsion force provided by each motor, equation (3.8) is used. In this equation, η_{motor} represents the efficiency of the motors, $T_{\max}$ the maximum torque one motor can provide, r_{wheel} the radius of the wheels and GR the gear ratio of the drivetrain assembly. The variable d represents the normalized throttle input. This variable, $d \in [-1, 1]$, represents the typical throttle the driver can input to the car through the pedals i.e. the situation of $d = 1$ translates to maximum acceleration, and for $d = -1$ a full breaking situation.

$$F_{\text{motor}} = \eta_{\text{motor}} \frac{T_{\max} \cdot GR}{r_{\text{wheel}}} d \tag{3.8}$$

Since there are two motors located in the rear axle of the considered FST10d Formula Student vehicle, the total propulsion force, F_{prop}, is given by the following equation:

$$F_{\text{prop}} = 2F_{\text{motor}} = 2\eta_{\text{motor}} \frac{T_{\max} \cdot GR}{r_{\text{wheel}}} d \tag{3.9}$$

As for the resistive forces, i.e. the rolling resistance and aerodynamic drag, the rolling resistance, F_{roll}, can be modeled as follows:

$$F_{\text{roll}} = C_r \cdot m \cdot g \tag{3.10}$$

where C_r represents the fraction of the vehicle's weight that contributes to rolling resistance and g the gravitational acceleration.

As for the aerodynamic drag force, it can be computed as follows:

$$F_{\text{drag}} = \frac{1}{2}\rho C_d A_F {v_x}^2 \tag{3.11}$$

where ρ is the density of air, C_d the drag coefficient of the vehicle, and A_F the frontal area of the vehicle.

3.1.4 Summation of Forces and Moments Around the CG

With the forces acting on the vehicle and its subsystems computed in the previous sections, the summation of forces and moments with respect to the vehicle's CG can be computed as in equations (3.12). As the vehicle is driven by the rear wheels, it is considered that the forces acting on the longitudinal direction of the tires, represented in figure 3.1b by F_x^f and F_x^r are zero and equal to the propulsion force, respectively, i.e. $F_x^f = 0$ and $F_x^r = F_{\text{prop}}$.

$$F_x = F_{\text{prop}} - F_{\text{drag}} - F_{\text{roll}} - F_y^f \sin(\delta) \tag{3.12a}$$

$$F_y = F_y^f \cos(\delta) + F_y^r \tag{3.12b}$$

$$M_z = F_y^f \cos(\delta) \cdot l_f - F_y^r \cdot l_r \tag{3.12c}$$

3.2 Kinematic Bicycle Model

The kinematic bicycle model is commonly used for modeling vehicles in urban driving scenarios. It provides a mathematical description of the vehicle without considering the forces acting on it. For this reason, the equations that lead to this model are purely based on geometric relations that govern the system. Consider again the previously shown bicycle model geometry represented in figure 3.1b.

An important consideration taken by this model is that the vehicle is assumed to have planar motion, not taking vehicle pitch and roll effects under consideration. For this reason, three coordinates are considered in this model: X, Y and Ψ. X and Y are inertial coordinates of the location of the CG of the vehicle while Ψ represents the orientation of the vehicle, also know as the vehicle's yaw angle. The velocity at the CG of the vehicle is denoted by V which direction makes an angle β, called the slip angle of the vehicle, with the longitudinal axis of the vehicle.

The major assumption that this model takes into account is that the velocity vectors of the rear and front wheel are aligned with the orientation of the rear and front wheels, respectively. In other words, this model assumes that the front and rear tires do not slip with respect to the ground, which is a reasonable assumption for low speeds and accelerations of the vehicle. By algebraic manipulation of the kinematic relations given by the model described in figure 3.1b, it is possible to write the set of differential equations [61] that govern this system, as shown in equations (3.13). It can also be noted that these equations result in a multi-input vehicle model that takes the vehicle velocity, V, and the steering angle, δ, as control inputs.

$$\dot{X} = V\cos(\Psi + \beta) \tag{3.13a}$$

$$\dot{Y} = V\sin(\Psi + \beta) \tag{3.13b}$$

$$\dot{\Psi} = \frac{V\cos(\beta)}{l_r + l_f}\tan(\delta) \tag{3.13c}$$

where the slip angle of the vehicle, β, can be computed as shown in equation (3.14).

$$\beta = \text{arctan}\left(\frac{l_f}{l_f + l_r}\tan(\delta)\right) \tag{3.14}$$

Considering the vehicle's frame of reference, in which the x axis is aligned with its longitudinal direction and y with the lateral direction and centered at the CG of the vehicle, one can denote the velocities of the vehicle in this frame of reference as projections of the CG velocity, V, as follows:

$$v_x = V\cos(\beta) \tag{3.15a}$$

$$v_y = V\sin(\beta) \tag{3.15b}$$

Since it is desired to have the throttle, t, as the input to the system the following dynamic equation for the derivative of the longitudinal velocity is introduced:

$$\dot{v}_x = \frac{F_{\text{prop}}(d) - F_{\text{drag}}(v_x) - F_{\text{roll}}}{m} \tag{3.16}$$

where m represents the vehicle's mass. From a velocity triangle, it is also possible to write a static relation for the lateral vehicle velocity v_y as follows:

$$v_y = \dot{\Psi}\tan(\delta)\frac{l_r}{l_r + l_f} \tag{3.17}$$

As a summary, the kinematic bicycle model results in the following set of equations, regarding the quantities expressed in the vehicle's frame of reference.

$$\dot{v}_x = \frac{F_{\text{prop}}(d) - F_{\text{drag}}(v_x) - F_{\text{roll}}}{m} \tag{3.18a}$$

$$\dot{\Psi} = \frac{v_x}{l_r + l_f} \tan(\delta) \tag{3.18b}$$

$$v_y = \dot{\Psi} l_r = v_x \tan(\delta) \frac{l_r}{l_r + l_f} \tag{3.18c}$$

To obtain the global coordinates of the vehicle, (X, Y), as a function of the local vehicle velocities, v_x and v_y, the transformation shown in equation (3.1) from section 3.1.1 is employed, resulting in the following dynamic equation:

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \end{bmatrix} = \mathbf{R}_{(x,y)}^{(X,Y)} \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} v_x \cos(\Psi) - v_y \sin(\Psi) \\ v_x \sin(\Psi) + v_y \cos(\Psi) \end{bmatrix} \tag{3.19}$$

Using equations (3.18) and (3.19), a state space model formulation of this system of equations can be written: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, where $\mathbf{x} = [X, Y, \Psi, v_x]^T$ is the state vector and $\mathbf{u} = [d, \delta]^T$ is the input vector.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \\ \dot{v_x} \end{bmatrix} = \begin{bmatrix} v_x \cos(\Psi) - v_y \sin(\Psi) \\ v_x \sin(\Psi) + v_y \cos(\Psi) \\ \frac{v_x}{l_r + l_f} \tan(\delta) \\ \frac{F_{\text{prop}}(d) - F_{\text{drag}}(v_x) - F_{\text{roll}}}{m} \end{bmatrix} \tag{3.20}$$

For purposes of notation and further use, let $r \equiv \dot{\Psi}$. In other words, r will be introduced to represent the time variation of the yaw angle. Then, r and v_y can be computed using the relation presented in equations (3.18b) and (3.18c), respectively.

In order to implement these dynamics in a MPC, a discretization of the continuous model presented in equation (3.20) must be computed. Using a discretization method a discrete state space model can be obtained as $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$, where $\mathbf{x}_k = [X_k, Y_k, \Psi_k, v_{x_k}]^T$ is the state vector and $\mathbf{u}_k = [d_k, \delta_k]^T$ is the input vector. The employed discretization method can be further consulted in section 3.4.

3.3 Dynamic Bicycle Model

Despite the model presented in the previous section being a fair model for describing a vehicle at relatively low speeds, for the reasons presented in this chapter, it is clear that the no slip assumption of the rear and front wheels of the kinematic bicycle model is not valid considering a racing scenario, which requires the vehicle to perform high acceleration and velocity maneuvers. For the purpose of developing a model that better describes the tire-slip dynamics of the vehicle, consider again the model described in figure 3.1b.

Similarly to the kinematic bicycle model, assuming a planar motion of the vehicle and neglecting the effects caused by the pitch and roll of the vehicle, for a frame of reference in which axis x and y are aligned with the longitudinal and lateral directions of the vehicle, respectively, the following dynamics equations can be written with respect to this frame of reference, resulting in equations (3.21) [61].

$$F_x = m\dot{v}_x - m\dot{\Psi}v_y \tag{3.21a}$$

$$F_y = m\dot{v}_y + m\dot{\Psi}v_x \tag{3.21b}$$

$$M_z = \ddot{\Psi}I_z \tag{3.21c}$$

where m and I_z represent the mass of the vehicle and the inertia moment with respect to its z axis. v_x and v_y represent the velocity of the CG of the vehicle expressed in the vehicle's frame of reference. F_x and F_y represent the sum of forces being applied to the CG of the vehicle expressed in the vehicle's frame of reference. M_z represents the sum of moments being applied to the CG of the vehicle with respect to its z axis. These quantities can be computed using the equations presented in section 3.1.

Similarly to the kinematic bicycle model, it is possible to express the global coordinates of the vehicle as a function of the local vehicle velocities as shown by equation (3.19) from the previous section.

The dynamic bicycle model can be obtained by algebraic manipulation of equations (3.21) and (3.19) and written in a state space format: $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$, where $\mathbf{x} = [X, Y\Psi, v_x, v_y, r]^T$ is the state vector and $\mathbf{u} = [d, \delta]^T$ is the input vector, as in equation (3.22). Note that, similarly to the kinematic bicycle model, $\ddot{\Psi}$ was rewritten as $\dot{r}$ i.e. $\ddot{\Psi} \equiv \dot{r}$, where r represents the time variation of the yaw angle, Ψ.

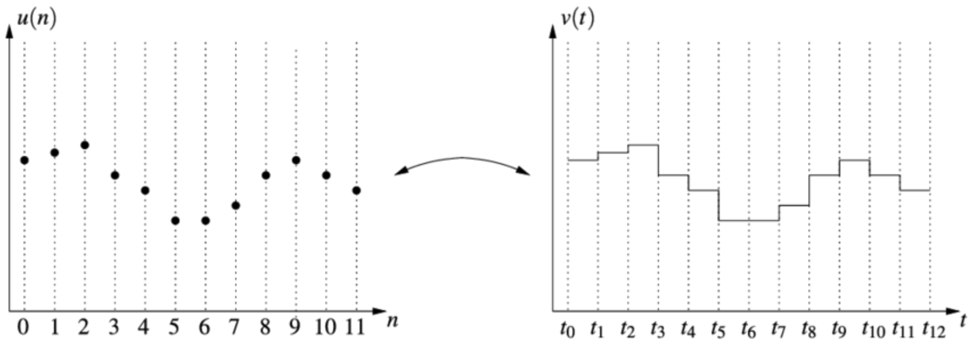
$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x\cos(\Psi) - v_y\sin(\Psi) \\ v_x\sin(\Psi) + v_y\cos(\Psi) \\ r \\ \frac{1}{m}\left(F_{\text{prop}}(d) - F_{\text{drag}}(v_x) - F_{\text{roll}} - F_y^f\sin(\delta)\right) + v_y r \\ \frac{1}{m}\left(F_y^f\cos(\delta) + F_y^r\right) + v_x r \\ \frac{1}{I_z}\left(F_y^f\cos(\delta)\cdot l_f - F_y^r\cdot l_r\right) \end{bmatrix} \tag{3.22}$$

3.4 Model Discretization

For the purpose of implementing the two previously presented models in a Model Predictive Controller, a discretization method must be employed. Consider a continuous-time dynamical system described by the following state space equation:

$$\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.23}$$

The first consideration one must take into account is how the control actions are intended to be applied. As the vehicle will be controlled by a digital system, discrete control inputs at a fixed time rate are expected. Similarly, the MPC assumes that the control actions are kept constant between two sampling instants. For this reason, a zero order hold input method for the control actions is chosen. A scheme of this input method can be seen in figure 3.3



u(n)
0 1 2 3 4 5 6 7 8 9 10 11 n
v(t)
t0 t1 t2 t3 t4 t5 t6 t7 t8 t9 t10 t11 t12 t

Figure 3.3: Zero order hold input scheme illustration [62].

Mathematically, a zero order hold input method consists in applying a piecewise constant control discretization between two time instants throughout the MPC prediction horizon, N, as suggested by the following equation:

$$\mathbf{u}(t) = \mathbf{u}_k \quad \forall t \in [t_k, t_{k+1}] \quad , \quad k = i, ..., i + N - 1 \tag{3.24}$$

Focusing on the discretization of the state trajectory, a multiple shooting discretization method was chosen. This method works as follows: consider the time interval between two sampling instants, $[t_k, t_{k+1}]$. Given an initial state $\mathbf{x}_k \equiv \mathbf{x}(t_k)$, one can apply a numerical integration method in order to obtain $\mathbf{x}_{k+1}$ as suggested by the following equation:

$$\mathbf{x}_{k+1} = \Phi_k(\mathbf{x}(t), \mathbf{u}(t)) \tag{3.25}$$

where Φ_k is a suitable integrator [63]. A frequent choice for Φ_k is a Runge-Kutta 4th order (RK4) approximation, since this method is reasonably simple and robust and is a good general candidate for numerical solution of differential equations. As an input to this method, a step size, h, such that $0 < h \leq T_s$ and $nh = T_s$ for some $n \in \mathbb{N}$ is chosen as the interval between two integration nodes, called the *shooting nodes*. In general, h is chosen to be equal to the sampling time T_s, however, if better accuracy is required, h should be chosen to be smaller than the sampling time. Then, algorithm 3 is applied.

Algorithm 3 Runge-Kutta 4th order

```
1: procedure RK4($h, T_s, \mathbf{x}_k, \mathbf{u}_k, f(\cdot)$)          ▷ Input data for the method
2:     $\mathbf{x}_0 \leftarrow \mathbf{x}_k$                                        ▷ Assign initial state
3:     for integer $n = 0$ to $n = \frac{T_s}{h} - 1$ do              ▷ Integrate between shooting nodes
4:         $k_1 \leftarrow f(\mathbf{x}_n, \mathbf{u}_k)$
5:         $k_2 \leftarrow f(\mathbf{x}_n + \frac{h}{2}k_1, \mathbf{u}_k)$
6:         $k_3 \leftarrow f(\mathbf{x}_n + \frac{h}{2}k_2, \mathbf{u}_k)$
7:         $k_4 \leftarrow f(\mathbf{x}_n + hk_3, \mathbf{u}_k)$
8:         $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$
9:     end for
10:    $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_{\frac{T_s}{h}}$       ▷ The next state corresponds to the last integration node state
11:    return $\mathbf{x}_{k+1}$                                      ▷ Output next state
12: end procedure
```

With the discretization procedure defined, for control purposes it is desired to discretize the state vector as $\mathbf{x}_k = [X_k, Y_k, \Psi_k, v_{x_k}, v_{y_k}, r_k]^T$, the input vector as $\mathbf{u}_k = [d_k, \delta_k]^T$, and define the discrete state transition function as $\mathbf{x}_{k+1} = f(\mathbf{x}_k, \mathbf{u}_k)$, where, for notation purposes, f, refers to the discretization of the continuous model dynamics using algorithm 3.

For the discretization of the kinematic bicycle model, the following approach was adopted. Since, in this model, the state vector is defined as $\mathbf{x}^{\text{kin}} = [X, Y, \Psi, v_x]^T$, the state space model presented in equation (3.20) is discretized using the procedure in algorithm 3, obtaining a discrete time dynamic model: $\mathbf{x}^{\text{kin}}_{k+1} = f_{\text{kin\_partial}}(\mathbf{x}_k, \mathbf{u}_k)$. Then, to compute the remaining states, $v_{y_{k+1}}$ and r_{k+1} the kinematic relations of equations (3.18b) and (3.18c) are employed as follows:

$$r_{k+1} = \frac{v_{x_{k+1}}}{l_r + l_f} \tan(\delta_k) \tag{3.26a}$$

$$v_{y_{k+1}} = r_{k+1} l_r = v_{x_{k+1}} \tan(\delta_k) \frac{l_r}{l_r + l_f} \tag{3.26b}$$

Finally, a discrete state space model for the kinematic bicycle, $\mathbf{x}_{k+1} = f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k)$, is obtained by annexing the states v_{y_k} and r_k to the state space vector. Resulting in the desired state space vector:

$$\mathbf{x}_k = \begin{bmatrix} \mathbf{x}^{\text{kin}}_k \\ v_{y_k} \\ r_k \end{bmatrix} \implies \mathbf{x}_{k+1} = \begin{bmatrix} f_{\text{kin\_partial}}(\mathbf{x}_k, \mathbf{u}_k) \\ v_{y_{k+1}} \\ r_{k+1} \end{bmatrix} = f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k) \tag{3.27}$$

The discretization of the dynamic bicycle model is made simpler by the fact that this model already utilizes the desired state space vector. Thus, a direct discretization of equation (3.22) using algorithm 3 is employed, resulting in the discrete state space model for the dynamic bicycle, $\mathbf{x}_{k+1} = f_{\text{dyn}}(\mathbf{x}_k, \mathbf{u}_k)$.

3.5 Blended Bicycle Model

In the previous sections, two methods were presented for modeling the dynamics of the vehicle: the kinematic bicycle model and the dynamic bicycle model. Compared to the dynamic bicycle model, the kinematic model is a simpler model that does not include tire-slip dynamics, which are important to model for vehicles that perform high acceleration and high velocity maneuvers, as in the case of a Formula Student vehicle. The importance of modeling such dynamics for these vehicles may lead to, naively, exclusively using the bicycle dynamic model as the model to use in the control algorithm. However, it is easy to notice that, for low velocities, the tire-slip angle as computed in equations (3.6) in section 3.1.2 is ill defined for low velocities. This may result in unrealistically high lateral forces acting on the vehicle when the vehicle is moving at low velocities or even stopped.

For the previously mentioned reasons, it is noticeable that the kinematic bicycle model is not valid for high velocities, where tire-slip dynamics are not neglectable, but valid for low velocities, and that the dynamic bicycle model is desirable for high velocities but unreliable for low velocities due to the ill definition of the tire-slip angle. Attempting to solve this problem, this section aims at developing a model valid for all velocity regions using a similar blending method to [42].

Taking into account the discretizations developed for the kinematic and dynamic bicycle models, presented in section 3.4, given by $\mathbf{x}_{k+1} = f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k)$ and $\mathbf{x}_{k+1} = f_{\text{dyn}}(\mathbf{x}_k, \mathbf{u}_k)$, respectively. The state vector, $\mathbf{x}_k = [X_k, Y_k, \Psi_k, v_{x_k}, v_{y_k}, r_k]^T$, represents the state variables and $\mathbf{u}_k = [d_k, \delta_k]^T$ the input variables at time instant k. By this definition, the functions f_{kin} and f_{dyn} relate the states of the system at time instant k, to the next time instant, $k+1$. The discrete state transition function of the blended bicycle model is expressed in equation (3.28).

$$\mathbf{x}_{k+1} = \lambda_k f_{\text{dyn}}(\mathbf{x}_k, \mathbf{u}_k) + (1-\lambda_k) f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k) = f_{\text{blend}}(\mathbf{x}_k, \mathbf{u}_k) \tag{3.28}$$

where the variable $\lambda_k \in [0,1]$ represents the relevance of the dynamic bicycle model. In other words, for $\lambda_k = 1$, the dynamic model is used as the state transition function, whereas for $\lambda_k = 0$, the kinematic model is used. Consequently, λ_k is computed based on the vehicle's velocity, $V_k = \sqrt{{v_{x_k}}^2 + {v_{y_k}}^2}$, as follows.

$$\lambda_k = \min\left(\max\left(\frac{V_k - V_{\text{blend}_{\text{min}}}}{V_{\text{blend}_{\text{max}}} - V_{\text{blend}_{\text{min}}}}, 0\right), 1\right) \tag{3.29}$$

where $V_{\text{blend}_{\text{min}}}$ and $V_{\text{blend}_{\text{max}}}$ define the velocity boundaries at which either only the kinematic or dynamic model is used, i.e. for $V_k \geq V_{\text{blend}_{\text{max}}}$ the dynamic bicycle model is used and for $V_k \leq V_{\text{blend}_{\text{min}}}$ the kinematic bicycle model is used. If the velocity, V_k, lies between these boundaries, $V_{\text{blend}_{\text{min}}} < V_k < V_{\text{blend}_{\text{max}}}$, a linear combination of both models will be used as expressed in equation (3.28).

Furthermore, the parameter values relating to the FST10d vehicle are summarized in table B.1 in appendix B.

3.6 Error Correction Model

So far, the previous models are built based on first principles, more specifically, based on Newtonian mechanics. These models describe a fair representation of the physics of a race car. However, several considerations were taken when developing these models which may result in inaccuracies when comparing predictions from the bicycle models to the actual real vehicle dynamics. Furthermore, not only do these considerations affect the disparity between the real car and the car model, but also uncontrollable and unpredictable phenomena. Consider the following example: if the tire parameters were taken with reference to ideal track conditions, in dry asphalt, there is no guarantee that the road is dry at the time of the race. If the road is wet, it is expected that the tires do not have an ideal grip, thus leading to greater disparity between the bicycle model and the real vehicle dynamics. Other unpredictable phenomena that may incur might be related to wind conditions, a slight malfunction in one of the vehicle motors, etc. For a personal vehicle, in a normal traffic scenario, this disparity may not be so important since such vehicle is not subject to high accelerations and overall aggressive driving. However, for race cars such as the ones Formula Student teams develop, precision in the control of these vehicles is fundamental in order to ensure that not only the vehicle does not crash but also to achieve the highest score possible in the competitions.

For the reasons mentioned in the previous paragraph, a machine learning technique that minimizes the disparity between the vehicle model and the real vehicle was developed. The focus of this technique is to learn some function e that is able to model the error between the real vehicle dynamics and the blended bicycle model presented in the previous section by introducing the error correction on the state transition function, as suggested in equation (3.30).

$$\mathbf{x}_{k+1}^{\text{real}} \approx f_{\text{blend}}(\mathbf{x}_k, \mathbf{u}_k) + e_{\Psi_k}^{(X,Y)}(\mathbf{p}_k, \mathbf{v}_k) \tag{3.30}$$

In this equation, $\mathbf{x}_{k+1}^{\text{real}}$ is the real measured and/or estimated states of the vehicle in the next time instant, $f_{\text{blend}}(\mathbf{x}_k, \mathbf{u}_k)$ is the state transition function given by the blended bicycle model from the previous section and $e_{\Psi_k}^{(X,Y)}(\mathbf{p}_k, \mathbf{v}_k)$ is the error between the real vehicle dynamics and the bicycle model. The parameter vector, $\mathbf{p}$, is the vector of parameters that shape the error function, e, and $\mathbf{v}$ is the feature vector which consists of a subset of the vehicle states and control actions, $\mathbf{v} \subseteq \{\mathbf{x}, \mathbf{u}\}$. The superscript (X, Y) refers tho the global frame of reference and the subscript Ψ_k refers to the vehicle's orientation at time instant k. The meaning of this superscript and subscript will be further explained in the training paragraph.

Note that the exact real vehicle states at the next time instant, $\mathbf{x}_{k+1}^{\text{real}}$, are unknown at time instant k. Furthermore, in order to introduce the learning scheme used to learn the error function, consider the same equation as 3.30, but applied to the previous time instant, as in equation (3.31).

$$\mathbf{x}_{k}^{\text{real}} \approx f_{\text{blend}}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + e_{\Psi_{k-1}}^{(X,Y)}(\mathbf{p}_{k-1}, \mathbf{v}_{k-1}) \tag{3.31}$$

In this equation, every state and control actions are known at time instant k: $\mathbf{x}_k^{\text{real}}$ is the real vehicle current state measure/estimation and $\mathbf{x}_{k-1}$ and $\mathbf{u}_{k-1}$ were the real vehicle state and control actions in the previous time instant, $k-1$.

Further defining the error function, $e : \mathbb{R}^{n_p} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n_e}$ is described as a function with n_p shaping parameters, n_f input features and n_e outputs, equal to the dimension of the state vector.

For the purpose of learning the error function, e, a parametric model was chosen, since not only it requires less data to train but also makes it possible to input the fixed number of model parameters into the MPC solver, which will be further detailed in section 6.3.1. As such, single-layer neural networks were chosen to learn the error function, namely Elliptical Basis Function Networks (EBFNs). For an introduction to these neural networks recall sections 2.2.1 and 2.2.2.

3.6.1 Training and Feature Selection

The objective of employing this model correction method is to improve the vehicle model's accuracy online, as the vehicle drives itself. In other words, consider the previously mentioned equation (3.31). As the vehicle drives, at time instant k, the state vector, $\mathbf{x}_k^{\text{real}}$ is known: either by direct sensor measurement or by some state estimation technique. At the previous time instant, $k-1$, the blended bicycle vehicle model predicted that by applying control actions $\mathbf{u}_{k-1}$, the next time instant (instant k) state vector would be $\mathbf{x}_k^{\text{blend}}$. However, due to the reasons mentioned before, $\mathbf{x}_k^{\text{real}}$ may be different from $\mathbf{x}_k^{\text{blend}}$. The error correction term, $\mathbf{e}_k = e(\mathbf{p}_{k-1}, \mathbf{v}_{k-1})$, is then added to $\mathbf{x}_k^{\text{blend}}$ such that it closely matches $\mathbf{x}_k^{\text{real}}$. As such, at time instant k, it is desired that the discrepancy between the real state vector $\mathbf{x}_k^{\text{real}}$ and the corrected model, i.e., $\mathbf{x}_k^{\text{blend}}$ added by $\mathbf{e}_k$ is minimized. The discrepancy at time instant k, $\boldsymbol{\epsilon}_k$, is defined as follows:

$$\boldsymbol{\epsilon}_k^{(X,Y)} = \mathbf{x}_k^{\text{real}} - \Big(f_{\text{blend}}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}) + e_{\Psi_{k-1}}^{(X,Y)}(\mathbf{p}_{k-1}, \mathbf{v}_{k-1}) \Big) \tag{3.32}$$

In order to achieve the best possible next time instant prediction, as in equation (3.30), it is necessary that the discrepancy's components converge to zero. Thus, it is necessary to find a parameter learning algorithm that minimizes the discrepancy over time, as the vehicle gathers more data. The indexes (X, Y) that are present in equation (3.32) in the discrepancy term mean that, in this equation, the discrepancy spatial coordinate (i.e., X and Y) values are expressed in the global frame of reference. For further clarification on the global and local frames of reference refer to section 3.1.1.

Feature selection

The feature vector, $\mathbf{v}$, was introduced at the beginning of section 3.6 as a vector which may contain some components of the state vector, $\mathbf{x}$, and some components of the input vector, $\mathbf{u}$, i.e., as a subset of these two vectors: $\mathbf{v} \subseteq \{\mathbf{x}, \mathbf{u}\}$. Feature selection can be defined as finding this subset such that it maximizes the learner's ability to classify patterns. It may bring several benefits such as reducing overfitting, improving model accuracy, and training time reduction [64]. For the purpose of learning the parameters of the error function, $\mathbf{p}$, local features, i.e., related with the local frame of reference were chosen. Recalling equation (3.22) from the dynamic bicycle model, note that the vehicle's global coordinates, X and Y, and the vehicle's orientation, Ψ, are essentially obtained by an integration of the local states, v_v, v_y and r. If the X, Y, and Ψ components were used, the learning would be based on the geometry of the track itself, which is not desired. To illustrate this, one can consider a track with two or more identical sections, for example, consider the track illustrated in figure 3.4. The presented track contains two identical curves and two identical straight lines in different locations. While driving through, for example, the two curves, the local states are expected to have similar values despite the global coordinates and vehicle's orientations being different. Thus, introducing the global coordinates and orientation variables in the feature vector would result somewhat in the learning of the track while potentially disregarding similar driving conditions, i.e., similar local states.

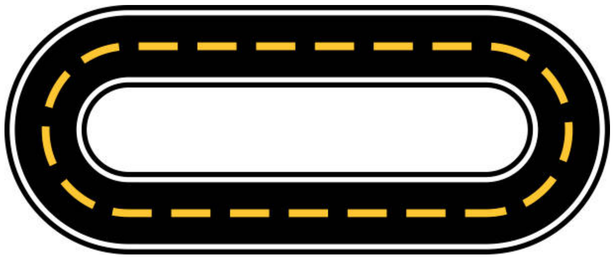


Figure 3.4: Example track with two similar curves and two similar straight lines.

Regarding the control actions, both the steering angle and throttle can be chosen to be inserted in the feature vector since these quantities are also locally related to the vehicle.

In summary, the feature vector, $\mathbf{v}$, was initially chosen to have the local vehicle states as well as control actions, mathematically, $\mathbf{v}_k = \left[v_{x_k}, v_{y_k}, r_k, d_k, \delta_k\right]^T$.

However, using data from previous simulations and performing a correlation analysis between the previously mentioned feature vector variables, it was observed that v_y and r were highly correlated. This can be intuitively noted in the kinematic bicycle model equations. Equation (3.18c) states that $v_y = \dot{\Psi} l_r \leftrightarrow v_y = r l_r$, showing that the kinematic bicycle model, indeed, fully correlates the vehicle's lateral velocity with the yaw rate. This behavior is not predicted by the dynamic bicycle model since this model allows for the vehicle to drift sideways. Nevertheless, FS vehicles are not designed to drift sideways as such technique is inefficient energy wise. If, however, the vehicles drifts, it is typically for short periods of time. As such, since, statistically and also mathematically, these variables offer the same information, one of them was removed in order to simplify the error correction model. Thus, the yaw rate, r, was chosen to be kept in the feature vector, removing the lateral velocity, v_y. Finally, the feature vector was considered to be defined as $\mathbf{v}_k = \left[v_{x_k}, r_k, d_k, \delta_k\right]^T$.

Training algorithm

For the purpose of fitting the error correction function, e, a training algorithm is proposed which involves two consecutive general steps:

1. A pre-training phase, in which offline data previously collected allows for a prior estimation of the parameter vector, $\mathbf{p}$;
2. An online training phase, in which the parameter vector, $\mathbf{p}$, is fine tuned in order to minimize the model discrepancy, $\boldsymbol{\epsilon}$, as the vehicle drives itself.

Regarding the pre-training phase, as suggested by Man-Wai Mak and Sun-Yuan Kung [53], previously collected data is used to find the EBFN's cluster centers, $\boldsymbol{\mu}_i$ and variance, $\boldsymbol{\Sigma}_i$, $i = 1, ..., n_h$. This technique consists in applying the fuzzy c-means algorithm to cluster the feature space. Then, the cluster's variance are found using the variance estimator with respect to each cluster center, as follows in equation (3.33). In this equation, N_i represents the number of data points that belong to cluster i. As such, N_i is found by counting how many data points in the training set have their maximum membership degree in cluster i using the partition matrix that the fuzzy c-means algorithm outputs.

$$\boldsymbol{\Sigma}_i = \frac{1}{N_i} \sum_{j=1}^{N_i} \left(\mathbf{v}_j - \boldsymbol{\mu}_i\right) \left(\mathbf{v}_j - \boldsymbol{\mu}_i\right)^T \quad , \quad i = 1, ..., n_h \tag{3.33}$$

Then, the diagonal terms of $\boldsymbol{\Sigma}_i$ are extracted and the precision matrix $\boldsymbol{\Omega}_i \approx \operatorname{diag}\left(\omega_{i,1}, ..., \omega_{i,n_v}\right)$ is obtained by the inverse of the diagonal components of $\boldsymbol{\Sigma}_i$: $\omega_{i,j} = 1/\sigma_{i,j}$, $j = 1, ..., n_v$ where $\sigma_{i,j}$ represents the j^{th} diagonal components of cluster's i covariance matrix. As for the output layer weights, these are then initialized as small random numbers which will be tuned in the online learning phase that will be described next.

Recall the definition of the discrepancy vector expressed in the global frame of reference, $\boldsymbol{\epsilon}_k^{(X,Y)}$, as written in equation (3.32). For notation purposes, the sampling index k will be omitted for the following expressions. Further analyzing the individual components of this vector, $\boldsymbol{\epsilon}^{(X,Y)} = \left[\epsilon_X, \epsilon_Y, \epsilon_\Psi, \epsilon_{v_x}, \epsilon_{v_y}, \epsilon_r\right]^T$, it can be observed that the first two components, regarding the spatial coordinates discrepancy, ϵ_X and ϵ_Y are expressed in the global frame of reference, as was also stated before. By the definition presented in equation (3.32) there is no guarantee that ϵ_X and ϵ_Y are independent of the vehicle's orientation, Ψ. However, as justified before in the feature selection paragraph, learning local features is desired. On the other hand, if local features are learned and the vehicle's orientation, Ψ, is not fed as an input feature, directly outputting the correction for X and Y, i.e., e_X and e_Y is not trivial since e_X and e_Y may depend on the vehicle's orientation. As such, in order to keep the vehicle's orientation out of the feature vector such that the geometry of the track is not learned, as explained in the feature selection paragraph, a transformation such that ϵ_X, ϵ_Y, e_X and e_Y are converted into a local frame of reference, obtaining ϵ_x, ϵ_y, e_x and e_y, is employed. For the purpose of explaining this transformation consider the illustration from figure 3.5.

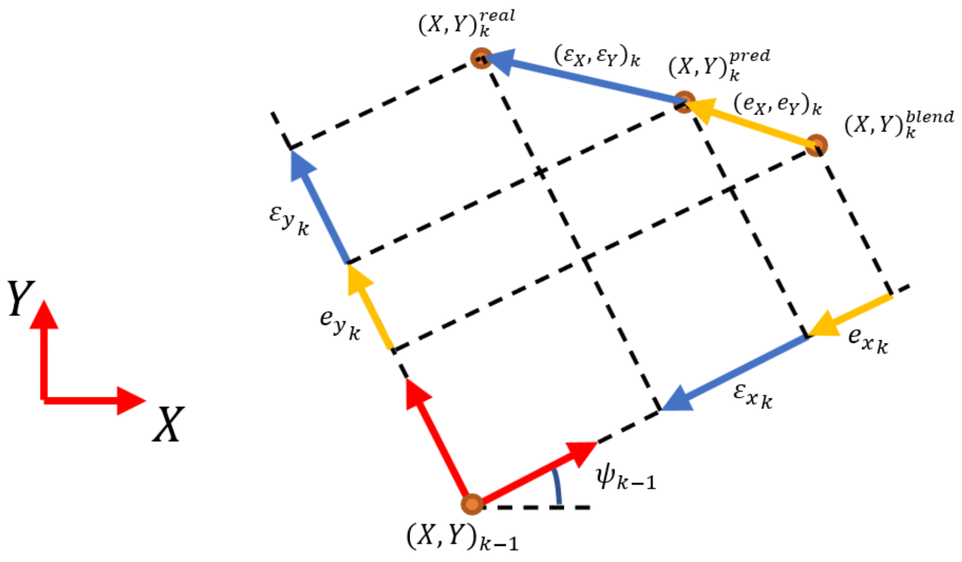


Figure 3.5: Discrepancy and error transformation to the previous local frame of reference

Consider the previous time instant $k-1$. At this time instant the vehicle was oriented according to Ψ_{k-1} and a state vector prediction of the current time instant k was made according to $\mathbf{x}_{k-1}$, $\mathbf{u}_{k-1}$ and $\mathbf{v}_{k-1}$. This prediction is denoted by $\mathbf{x}_k^{\text{pred}}$. However, at the current time instant k the vehicle states were measured/estimated to be $\mathbf{x}_k^{\text{real}}$. The discrepancy can then be computed as $\boldsymbol{\epsilon}_k^{(X,Y)} = \mathbf{x}_k^{\text{real}} - \mathbf{x}_k^{\text{pred}}$, with, as mentioned before, its first two components, ϵ_X and ϵ_Y, expressed in the global frame of reference. Since Ψ_{k-1} is known, these two quantities can be projected, i.e. rotated, to the the previous local frame of reference, obtaining ϵ_x and ϵ_y. Furthermore, since it is desired to learn these quantities expressed in the local frame of reference, the error correction function, $e^{(x,y)}$, will also have its first two components, e_x and e_y, expressed with respect to the previous local frame of reference. These quantities can easily be transformed to the local frame of reference using the rotation matrix $\mathbf{R}_{(x,y)}^{(X,Y)}$ presented in equation (3.2) in section 3.1.1. Recalling some properties of this matrix, given some vector with its components expressed in the local frame of reference, left-multiplying this matrix by that vector will rotate that vector to the global frame of reference. Recall also that the inverse transformation, $\mathbf{R}_{(X,Y)}^{(x,y)}$ can be obtained by $\mathbf{R}_{(X,Y)}^{(x,y)} = \left[\mathbf{R}_{(x,y)}^{(X,Y)}\right]^T$. Contrary to $\mathbf{R}_{(x,y)}^{(X,Y)}$, its inverse transformation, $\mathbf{R}_{(X,Y)}^{(x,y)}$, rotates a vector expressed in the global frame of reference into a local frame of reference. Then, the following equations can be used to transform $\boldsymbol{\epsilon}^{(X,Y)}$ into $\boldsymbol{\epsilon}^{(x,y)}$ and $e^{(x,y)}$ into $e^{(X,Y)}$.

$$\boldsymbol{\epsilon}^{(x,y)} = \mathbf{A}^T\left(\Psi_{k-1}\right)\boldsymbol{\epsilon}^{(X,Y)} \tag{3.34a}$$

$$e_{\Psi_{k-1}}^{(X,Y)} = \mathbf{A}\left(\Psi_{k-1}\right)e^{(x,y)} \tag{3.34b}$$

where the $\mathbf{A} \in \mathbb{R}^{6\times 6}$ matrix is defined as:

$$\mathbf{A}\left(\Psi\right) = \begin{bmatrix} \mathbf{R}_{(x,y)}^{(X,Y)}\left(\Psi\right) & 0_{2\times 4} \\ 0_{4\times 2} & \mathbf{I}_{4\times 4} \end{bmatrix} \tag{3.35}$$

Further specifying the vehicle model equation as well as introducing the state transition function that represents the vehicle model that will be used in the MPC, $f_{\text{vehicle}}\left(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k\right)$, and the local discrepancy equation by introducing the previous transformations:

$$\mathbf{x}_{k+1}^{\text{pred}} = f_{\text{vehicle}}\left(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_k\right) = f_{\text{blend}}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{A}(\Psi_k)e^{(x,y)}(\mathbf{p}_k, \mathbf{v}_k) \tag{3.36a}$$

$$\boldsymbol{\epsilon}_k^{(x,y)} = \mathbf{A}^T\left(\Psi_{k-1}\right)\left(\mathbf{x}_k^{\text{real}} - \mathbf{x}_k^{\text{pred}}\right) \tag{3.36b}$$

Note that Ψ_k and $\mathbf{v}_k$ are not inputs of the blended bicycle vehicle model. However, recall that these values are a subset of $\mathbf{x}_k$ and $\mathbf{u}_k$, thus, the vehicle's orientation and feature vector can be reconstructed from the state vector and input vector.

With the local discrepancy defined, $\boldsymbol{\epsilon}^{(x,y)}$, as well as the transformations mentioned before it comes naturally, from equation (3.36b) as well as from observing figure 3.5, that minimizing the discrepancy components over time improves the prediction accuracy, i.e., $\boldsymbol{\epsilon}^{(x,y)} \to 0 \implies \mathbf{x}^{\text{real}} \approx \mathbf{x}^{\text{pred}}$. Since the discrepancy is a vector which components are desired to be minimized, the mean squared error of the discrepancy vector will be used to design a cost function. The mean squared error of the discrepancy vector is then defined as follows:

$$\text{MSE}_k = \frac{1}{n_e}\sum_{j=1}^{n_e}\left(\epsilon_{k,j}\right)^2 \tag{3.37}$$

where, for simplicity, ϵ_j corresponds to component j of the local discrepancy vector, $\boldsymbol{\epsilon}_k^{(x,y)}$. Since there are six state variables, $n_e = 6$.

Considering a learning batch of size n_{batch}, meaning that both feature vectors, $\mathbf{v}_t$, and discrepancy vectors, $\boldsymbol{\epsilon}_t^{(x,y)}$ are stored up to n_{batch} instants before instant k, i.e. computationally, the feature vectors and discrepancies are stored for $t = k - n_{\text{batch}} + 1, ..., k$. At each time step k, the developed algorithm takes a step towards minimizing the following empirical risk cost function:

$$R = \frac{1}{\Gamma} \sum_{t=k-n_{\text{batch}}+1}^{k} \gamma^{k-t} \cdot \mathrm{MSE}_t \tag{3.38}$$

where the learning parameter $\gamma \in]0,1]$ refers to the forgetting factor. The forgetting factor, γ, is introduced due to the way a learning data point is treated at each sampling instant k. To better understand this, considered that at instant $k-1$, the mean squared error was measured to be MSE_{k-1}. With this measurement, the error function parameters were updated towards reducing MSE_{k-1}. However, since this data point cannot be resampled at time instant k, due to the vehicle being in different driving conditions, the previous time instant mean squared error, MSE_{k-1} has to be estimated at the current instant k. As such, a new value for the previous instant mean squared error value is considered, given by $\gamma\,\mathrm{MSE}_{k-1}$. Further analyzing how the value of γ affects equation (3.38), a forgetting rate close to one, $\gamma \to 1$, represents a high contribution of the previous samples on the cost function whereas a forgetting factor close to zero, $\gamma \to 0$, represents a low contribution of the previous samples on the cost function. Furthermore, the parameter Γ is a normalization parameter defined on the geometric series of the forgetting factor, γ. This term can be computed using equation (3.39).

$$\Gamma = \sum_{n=0}^{n_{\text{batch}}-1} \gamma^n = \frac{1-\gamma^{n_{\text{batch}}}}{1-\gamma} \tag{3.39}$$

Regarding the online learning algorithm, which minimizes R, several options were considered. The developed algorithm is an online version of the commonly known Levenberg-Marquardt algorithm. This algorithm merges two famous optimization algorithms for training neural networks:

1. **Online Gradient Descent (OGD)**: a first order method in which the parameter vector increment is made using the gradient with respect to the most recent data point;

2. **Levenberg-Marquardt algorithm (LM)**: an algorithm that interpolates between the Gauss-Newton algorithm and gradient descent.

The "Online Levenberg-Marquardt" (OLM) algorithm's objective is to capture the benefits of OGD and LM by combining them, taking the online training scheme of OGD and applying the LM hessian matrix approximation in order to improve the learning step direction of OGD. For the purpose of explaining this algorithm, the cost function Jacobian must be specified first. Recall from equations (3.36), that the discrepancy vector, $\boldsymbol{\epsilon}_k^{(x,y)}$, can be expressed as:

$$\boldsymbol{\epsilon}_k^{(x,y)} = \mathbf{A}^T\left(\Psi_{k-1}\right)\left(\mathbf{x}_k^{\text{real}} - f_{\text{blend}}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1})\right) - e^{(x,y)}(\mathbf{p}_{k-1}, \mathbf{v}_{k-1}) \tag{3.40}$$

As such, the Jacobian matrix of the discrepancy vector at instant k with respect to the parameter vector, $\mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_k}$, where component $\left[\mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_k}\right]_{i,j}$, in which index i represents a row and index j a column, can be computed as:

$$\left[\mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_k}\right]_{i,j} = \frac{\partial \epsilon_{k,j}}{\partial p_i} = -\frac{\partial e_j(\mathbf{p}, \mathbf{v}_{k-1})}{\partial p_i} \tag{3.41}$$

For the purpose of this thesis, the derivatives presented in equation (3.41) are computed using automatic differentiation tools, namely MATLAB's Symbolic Toolbox[1]. Nevertheless, these derivatives can be computed mainly resourcing to the chain rule, as shown in [65]. Note that the parameter vector, written as $\mathbf{p}$, refers to the current parameters, i.e., in this equation, the derivative is always evaluated with respect to the most recent parameters used at instant k despite using the corresponding features previous to instant k. Furthermore, the gradient of the mean squared error at instant k with respect to the discrepancy vector, $\nabla_{\boldsymbol{\epsilon}}\mathrm{MSE}_k$, can be computed as:

$$[\nabla_{\boldsymbol{\epsilon}}\mathrm{MSE}_k]_j = \frac{\partial\,\mathrm{MSE}_k}{\partial \epsilon_{k,j}} = \frac{2}{n_e}\cdot \epsilon_{k,j} \implies \nabla_{\boldsymbol{\epsilon}}\mathrm{MSE}_k = \frac{2}{n_e}\cdot \boldsymbol{\epsilon}_k \tag{3.42}$$

The gradient of the learning cost function with respect to the parameter vector, $\nabla_{\mathbf{p}}R$, can be computed as follows:

$$\nabla_{\mathbf{p}}R = \frac{1}{\Gamma}\sum_{t=k-n_{\text{batch}}+1}^{k} \gamma^{k-t}\cdot \mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_t}\cdot \nabla_{\boldsymbol{\epsilon}}\mathrm{MSE}_t = \frac{2}{n_e\cdot\Gamma}\sum_{t=k-n_{\text{batch}}+1}^{k} \gamma^{k-t}\cdot \mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_t}\cdot \boldsymbol{\epsilon}_t \tag{3.43}$$

The expression in equation (3.43) could be used if a gradient descent method, like OGD, was used. However, for the Levenberg-Marquardt update step, a Jacobian matrix of the whole cost function, $\mathbf{J}_{\mathbf{p}}$, and a discrepancy vector of the data in the learning batch, $\boldsymbol{\epsilon}^{\text{batch}}$, are required. As such, the expression in equation (3.43) for $\nabla_{\mathbf{p}}R$ can be alternatively rewritten in a matrix format as follows:

$$\nabla_{\mathbf{p}}R = \mathbf{J}_{\mathbf{p}}\cdot \boldsymbol{\epsilon}^{\text{batch}} \tag{3.44}$$

where the Jacobian matrix of the whole cost function, $\mathbf{J}_{\mathbf{p}}$, is computed by horizontally concatenating the discrepancy Jacobians of each data point in the learning batch, and the discrepancy vector of the data in the learning batch, $\boldsymbol{\epsilon}^{\text{batch}}$, is also computed by vertically concatenating the discrepancy vectors of each data point with the forgetting factor applied, as in the following equations.

$$\mathbf{J}_{\mathbf{p}} = \begin{bmatrix}\mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_k} & \mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_{k-1}} & \cdots & \mathbf{J}_{\mathbf{p}}^{\boldsymbol{\epsilon}_{k-n_{\text{batch}}+1}}\end{bmatrix} \tag{3.45a}$$

$$\boldsymbol{\epsilon}^{\text{batch}} = \frac{2}{n_e\cdot\Gamma}\begin{bmatrix}\boldsymbol{\epsilon}_k \\ \gamma\cdot\boldsymbol{\epsilon}_{k-1} \\ \vdots \\ \gamma^{n_{\text{batch}}-1}\cdot\boldsymbol{\epsilon}_{k-n_{\text{batch}}+1}\end{bmatrix} \tag{3.45b}$$

With the cost function Jacobian, $\mathbf{J}_{\mathbf{p}}$, and batch discrepancy, $\boldsymbol{\epsilon}^{\text{batch}}$, defined, the parameters are updated as in equation (2.21) introduced in section 2.2.4 - Levenberg-Marquardt Algorithm. Where $\Delta\mathbf{p}$ is obtained by solving the following linear system of equations:

$$\left(\lambda I + \mathbf{J}_{\mathbf{p}}{\mathbf{J}_{\mathbf{p}}}^{T}\right)\Delta\mathbf{p} = -\mathbf{J}_{\mathbf{p}}\boldsymbol{\epsilon}^{\text{batch}} \tag{3.46}$$

[1]`https://www.mathworks.com/products/symbolic.html`

Furthermore, in order to prevent overfitting as well as controlling the learning step size, a schedule based on the mean squared error for the damping factor was implemented. This schedule's purpose is to achieve a faster learning rate, i.e. a coarser but faster parameter tuning, when the mean squared error is high and a slower learning rate, i.e. a finer and slower parameter tuning, when the mean squared error is low. However, upon performing various simulations it was verified that the mean squared error is a noisy signal. Thus, to capture the average mean squared error, a first order low pass filter was implemented, resulting in a smoothed mean squared error. Figure 3.6 contains an example of the raw mean squared error in red and the low pass filtered mean squared error in blue for each sampling instant.

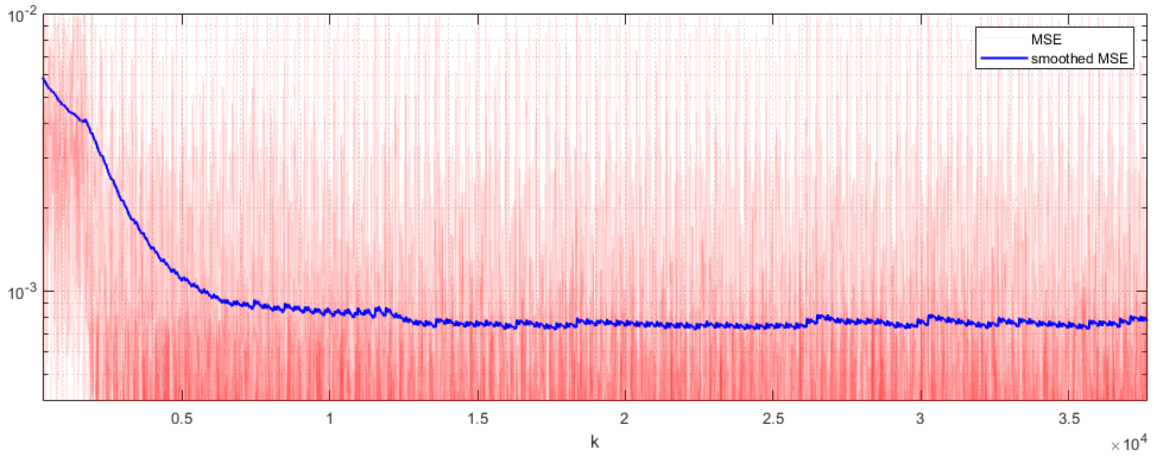


Figure 3.6: Example mean squared error signal vs. smoothed mean squared error

The smoothed mean squared error (SMSE) can be obtained through the general equation of a digital low pass filter as follows:

$$\mathrm{SMSE}_k = \lambda_{\mathrm{MSE}} \cdot \mathrm{MSE}_k + (1 - \lambda_{\mathrm{MSE}}) \cdot \mathrm{SMSE}_{k-1} \tag{3.47}$$

The parameter $\lambda_{\mathrm{MSE}} \in [0, 1]$ controls the filter's cutoff frequency, f_c, of the low pass filter. The filter's cutoff frequency can be computed as $f_c = \frac{\lambda_{\mathrm{MSE}}}{(1-\lambda_{\mathrm{MSE}})\cdot 2\pi \cdot T_s}$ [Hz] where T_s represents the controller's sampling time. Further analyzing λ_{MSE}, a value of λ_{MSE} close to one results in a high cutoff frequency, i.e. the raw mean squared error is less filtered while achieving faster response times, and a value of λ_{MSE} close to zero results in a low cut off frequency, i.e. a smoother signal, while achieving slower filter response times.

The damping parameter schedule is then defined as a function of the smoothed mean squared error as represented in figure 3.7. This figure represents a template function of the damping parameter which reflects the desired learning behavior of the algorithm. For high values of SMSE, i.e. for $\mathrm{SMSE} > \mathrm{SMSE}_{\text{high}}$, the damping parameter, λ, takes the value of λ_{high}, the lowest value λ can take, which is analogous to a high learning rate, meaning that when the smoothed mean squared error is high the algorithm learns faster in order to minimize the discrepancy as fast as possible such that the vehicle model is adapted quickly. For intermediate values of SMSE, i.e. for $\mathrm{SMSE}_{\text{low}} \leq \mathrm{SMSE} \leq \mathrm{SMSE}_{\text{high}}$, the damping parameter λ starts to linearly increase as the SMSE decreases, which is analogous to decreasing the learning rate as the smoothed mean squared error starts to decrease, meaning that, in

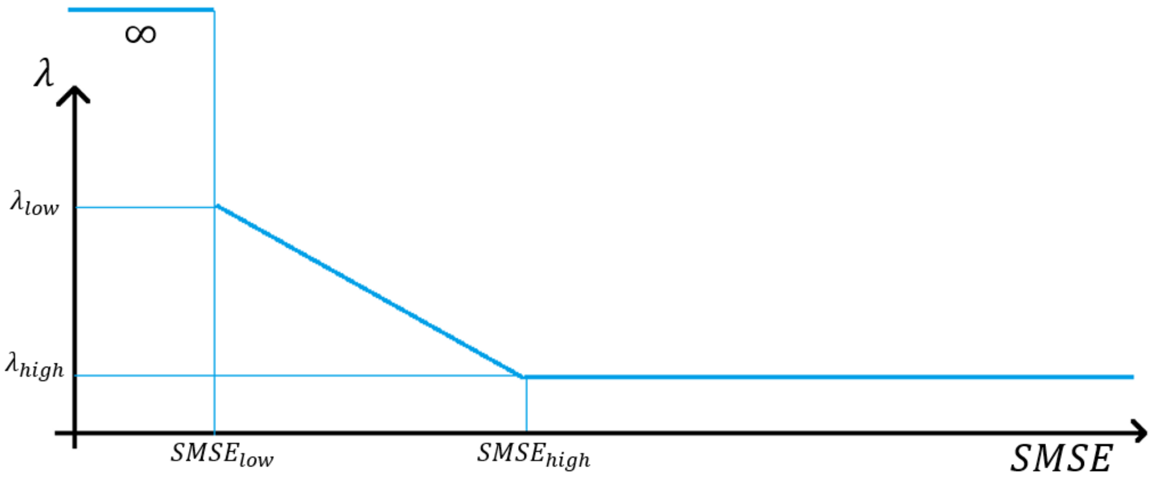


Figure 3.7: Schedule template function of the damping parameter, λ.

this region, the learning rate is decreased in order to fine tune the parameter vector. For low values of SMSE, i.e., for $0 \leq \mathrm{SMSE} < \mathrm{SMSE}_{\text{low}}$ the damping parameter takes an infinite value, which is equivalent of turning off the learning algorithm as, by recalling equation (3.46), $\Delta \mathbf{p} = 0$ in order to prevent overfitting. Mathematically, the schedule function for the damping parameter can be defined as follows in equation (3.48).

$$\lambda\left(\mathrm{SMSE}\right) = \begin{cases} +\infty & , 0 \leq \mathrm{SMSE} < \mathrm{SMSE}_{\text{low}} \\ \frac{\lambda_{\text{high}} - \lambda_{\text{low}}}{\mathrm{SMSE}_{\text{high}} - \mathrm{SMSE}_{\text{low}}}\,\mathrm{SMSE} + \lambda_{\text{low}} & , \mathrm{SMSE}_{\text{low}} \leq \mathrm{SMSE} \leq \mathrm{SMSE}_{\text{high}} \\ \lambda_{\text{high}} & , \mathrm{SMSE} > \mathrm{SMSE}_{\text{high}} \end{cases} \tag{3.48}$$

In summary, the Online Levenberg-Marquardt learning algorithm for the blended model error correction can be described as follows in algorithm 5 in appendix A and the learning parameter values can be consulted in table B.2 in appendix B. The parameter update algorithm takes the learning batch discrepancies and feature vectors as well as other relevant parameters as inputs and computes the current local discrepancy. After this, the current and smoothed mean squared errors are computed. If the smoothed mean squared error is above a threshold, the damping factor is computed according to the previously defined learning schedule and the Levenberg-Marquardt parameter update step is applied. It is important to note that as EBFNs are used, an additional step is added to ensure that the subset of the parameter vector regarding the eigenvalues of Ω_i, $i = 1, ..., n_h$ remains positive, i.e. to ensure that $\omega_{i,l} > 0$, $l = 1, ..., n_v$, $i = 1, ..., n_h$.

Chapter 4

MPC Formulation

In this chapter, the designed MPC architecture will be explained in detail. First, the concept of track progress with respect to the track's center line will be introduced, then the MPC optimization problem will be presented. This formulation was strongly influenced by Liniger et al. [31], in which a reference-free MPC based on the vehicle's track progress for autonomous racing is developed.

4.1 Track Progress

With the objective of developing a reference-free MPC, a common approach to design the MPC's cost function, J, to be optimized is to base this function on a measure of track progress. A natural way of measuring the track progress of a race vehicle is to project the vehicle's position at a given instant onto the race track center line, obtaining the length of the center line from the beginning of the track up to that projected point [31, 42, 66]. An illustration of this process can be seen in figure 4.1.

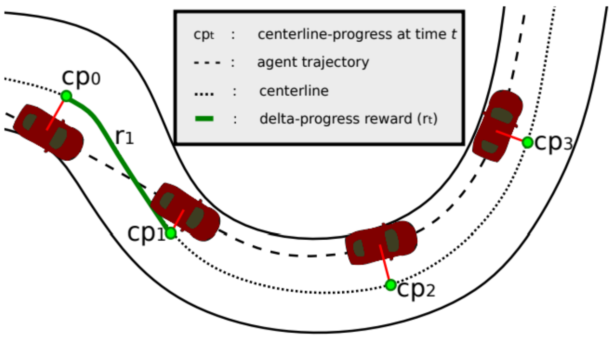

cp$_t$: centerline-progress at time t
- - - : agent trajectory
.... : centerline
▬ : delta-progress reward (r_t)
cp$_0$
r$_1$
cp$_1$
cp$_2$
cp$_3$

Figure 4.1: Track progress based on the track's center line [31].

With the track progress defined, it is natural to think of what are the optimal control actions to be applied to the vehicle at a given instant in order to maximize its track progress or the track progress velocity for a given prediction horizon. As such, a Model Predictive Controller based on the maximization of the track progress at a given time instant will be designed.

Note that, to use this center line projection to measure track progress, the track itself must be known prior to the race, which is generally the case for FS driverless competitions. Consider that the center line of the track is parameterized as a function of its length. In other words, consider the center line to be parameterized by a two dimensional vector, $\mathbf{g}$, which is a function of the center line length, s.

$$\mathbf{g}(s) = \begin{bmatrix} g_X(s) \\ g_Y(s) \end{bmatrix} \tag{4.1}$$

where the components $g_X(s)$ and $g_Y(s)$ are C^2 and express the coordinates of a center line point in the global frame of reference, (X, Y). With such parameterization of the center line, it is also possible to retrieve its direction at a given center line length by computing the derivative of $\mathbf{g}$ with respect to the center line length, s, denoted as $\frac{\partial \mathbf{g}}{\partial s}$. Furthermore, to compute the curvature radius of the track at a given center line length, the second derivatives of $\mathbf{g}$, denoted as $\frac{\partial^2 \mathbf{g}}{\partial s^2}$, are required. The mathematical definition of both these quantities is expressed in equations (4.2).

$$\frac{\partial \mathbf{g}}{\partial s}(s) = \begin{bmatrix} \frac{\partial g_X}{\partial s}(s) \\ \frac{\partial g_Y}{\partial s}(s) \end{bmatrix} \tag{4.2a}$$

$$\frac{\partial^2 \mathbf{g}}{\partial s^2}(s) = \begin{bmatrix} \frac{\partial^2 g_X}{\partial s^2}(s) \\ \frac{\partial^2 g_Y}{\partial s^2}(s) \end{bmatrix} \tag{4.2b}$$

Since $\mathbf{g}$ is parameterized by its length, $\frac{\partial \mathbf{g}}{\partial s}$ represents a unitary vector in the direction of the center line at a given length, s [67].

In order to find the center line parameterization, $\mathbf{g}$, cubic splines are used for $\mathbf{g}_X$ and $\mathbf{g}_Y$. In practice, using MATLAB and MATLAB's Symbolic Toolbox, a script was developed that takes a track as the input and computes the spline functions for $\mathbf{g}_X$ and $\mathbf{g}_Y$ and also its derivatives $\frac{\partial g_X}{\partial s}$ and $\frac{\partial g_Y}{\partial s}$, all function of the center line length, s.

Figure 4.2 shows an example track from the Formula Student Germany (FSG) competition and has represented the center line of the track in blue, the direction of the center line in black arrows and the left and right limits of the race track represented in green and red colors, respectively.

As stated previously, to compute the track progress, the position of the vehicle in the global frame of reference must be projected onto the center line of the track. Since the center line parameterization function, $\mathbf{g}(s)$, is considered to be C^2, the track progress at a given vehicle position, (X_k, Y_k), can be retrieved as the center line length value, s, for which the distance from the corresponding center line point to the vehicle position is minimal. This idea is equivalently formulated in the following equation:

$$s_k = \arg\min_s \sqrt{(X_k - g_X(s))^2 + (Y_k - g_Y(s))^2} \tag{4.3}$$

Another important measurement that can be taken from the previous idea is the distance to the center line itself, represented by e_{CL}. The distance of the vehicle to the center line is important to ensure that the car stays within the limits of the race track. This distance can be computed as follows:

$$e_{CL_k} = \min_s \sqrt{(X_k - g_X(s))^2 + (Y_k - g_Y(s))^2} \tag{4.4}$$

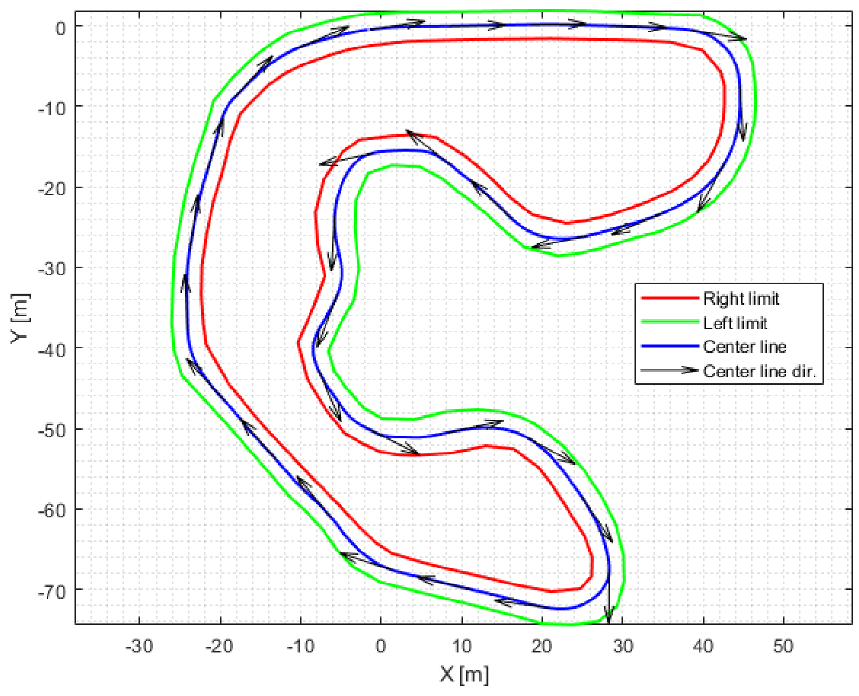


Figure 4.2: Race track from FSG.

Recall that $\frac{\partial \mathbf{g}}{\partial s}$ is a unitary vector with direction aligned with the track direction. The center line progress velocity can be computed by the dot product between the vehicle velocity expressed in the global frame of reference, $[\dot{X}_k, \dot{Y}_k]^T$, and the direction of the center line, $\frac{\partial \mathbf{g}}{\partial s}$, at the corresponding track progress, s_k. Consider the rotation matrix, $\mathbf{R}_{(x,y)}^{(X,Y)}$, from equation (3.2) in section 3.1.1. This rotation matrix transforms the velocity of the vehicle in its local frame of reference, $[v_{x_k}, v_{y_k}]^T$, to the global frame of reference, the center line progress velocity can be expressed as in the following equation.

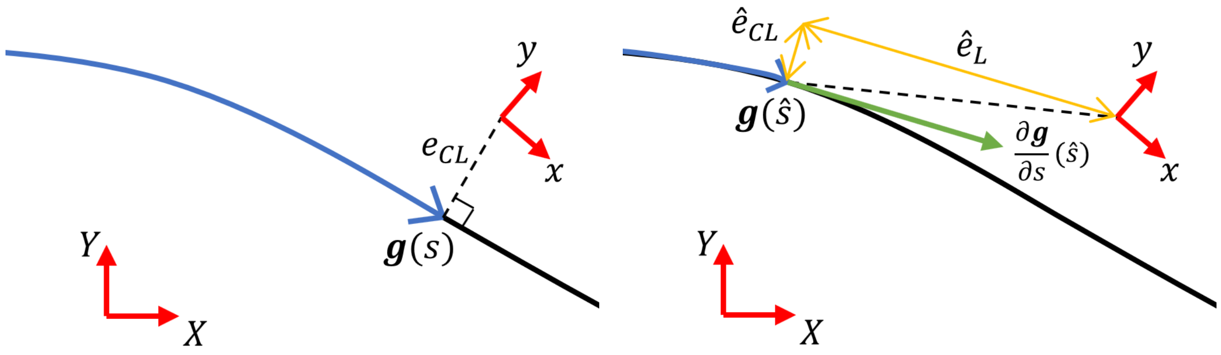
$$\dot{s}_k = \frac{\partial \mathbf{g}}{\partial s}^T \mathbf{R}_{(x,y)}^{(X,Y)} (\Psi_k) \begin{bmatrix} v_{x_k} \\ v_{y_k} \end{bmatrix} \tag{4.5}$$

Despite equations (4.3) and (4.4) representing an exact manner of computing s_k and e_{CL_k}, respectively, these equations do not guarantee explicit expressions for s_k and e_{CL_k}. As a result, the exact solution for s_k and e_{CL_k} would have to be retrieved by solving the respective optimization problems shown in these equations at each time step. However, solving an optimization problem within the optimal control problem of the MPC is not desired due to employing a high computational cost. For this reason, as suggested in [31, 42], a method for approximating the values of s_k and e_{CL_k} is employed. Figure 4.3 represents a visualization of the mechanisms involved in this approximation. The variable $\hat{e}_{CL}$ represents an approximation of e_{CL}, and $\hat{e}_L$ represents the *lag error*.

At a given estimated track length, $\hat{s}$, the values for $\hat{e}_{CL}$ and $\hat{e}_L$ can be explicitly expressed as follows:

$$\hat{e}_{CL}(\hat{s}) = \frac{\partial g_Y}{\partial s}(\hat{s}) \cdot (X - g_X(\hat{s})) - \frac{\partial g_X}{\partial s}(\hat{s}) \cdot (Y - g_Y(\hat{s})) \tag{4.6a}$$

$$\hat{e}_{CL}(\hat{s}) = -\frac{\partial g_X}{\partial s}(\hat{s}) \cdot (X - g_X(\hat{s})) - \frac{\partial g_Y}{\partial s}(\hat{s}) \cdot (Y - g_Y(\hat{s})) \tag{4.6b}$$



(a) True projection on the center line.

(b) Approximated projection on the center line.

Figure 4.3: Approximation mechanism for the track progress.

Note that, in order to obtain a good estimation of $s \approx \hat{s}$ and $e_{CL} \approx \hat{e}_{CL}$, both $\hat{e}_{CL}$ and $\hat{e}_L$ should take a minimum absolute value. For this reason, $\hat{e}_{CL}$ and $\hat{e}_L$ can be included in the MPC's cost function in order to compute an approximation of the center line progress at any prediction stage, s_k.

4.2 MPC Based On Track Progress

In order to develop a reference-free MPC based on the track progress, the formulation presented in equations (4.7) was developed.

$$
\begin{aligned}
\min_{\substack{\mathbf{u}_k, \forall k \in [t+1,\dots,t+N-1] \\ s_k \forall k \in [t,\dots,t+N]}} \quad & \alpha_{CL}\hat{e}^n_{CL_t} + \alpha_L \hat{e}^2_{L_t} + \\
& \sum_{k=t+1}^{t+N-1} \left(q_{v_y} {v_{y_k}}^2 + \alpha_{CL}\hat{e}^n_{CL_k} + \alpha_L \hat{e}^2_{L_k} + \beta_\delta(\delta_k - \delta_{k-1})^2 + e^{q_{v\text{max}}\left(v_{x_k} - v_{\text{max}}\right)} \right) \\
& - \lambda_s s_{t+N} + q_{v_y} {v_{y_{t+N}}}^2 + \alpha_{CL}\hat{e}^n_{CL_{t+N}} + \alpha_L \hat{e}^2_{L_{t+N}} + e^{q_{v\text{max}}\left(v_{x_{t+n}} - v_{\text{max}}\right)} && \text{(4.7a)} \\
\text{s.t.} \quad & \mathbf{x}_{k+1} = f_{\text{vehicle}}\left(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_t\right) \quad \forall k \in [t, \dots, t+N-1] && \text{(4.7b)} \\
& \mathbf{x}_t = \mathbf{x}(t) && \text{(4.7c)} \\
& \mathbf{u}_t = \mathbf{u}(t) && \text{(4.7d)} \\
& \mathbf{s}_{t-1} = \mathbf{s}(t-1) && \text{(4.7e)} \\
& -\Delta\mathbf{u}_{\max} \le \mathbf{u}_k - \mathbf{u}_{k-1} \le \Delta\mathbf{u}_{\max} \quad \forall k \in [t+1, \dots, t+N-1] && \text{(4.7f)} \\
& \mathbf{u}_{\min} \le \mathbf{u}_k \le \mathbf{u}_{\max} \quad \forall k \in [t+1, \dots, t+N-1] && \text{(4.7g)} \\
& -\hat{e}_{CL_{\max}} \le \hat{e}_{CL_k} \le \hat{e}_{CL_{\max}} \quad \forall k \in [t+1, \dots, t+N] && \text{(4.7h)} \\
& \Delta s_{\min} \le s_k - s_{k-1} \le \Delta s_{\max} \quad \forall k \in [t, \dots, t+N] && \text{(4.7i)}
\end{aligned}
$$

The main objective of this formulation is to obtain the set of control actions at time instant t that maximize the track progress (i.e. center line progress) at the prediction horizon instant, $t+N$. This behavior is reflected by the final stage cost term: $-\lambda_s s_{t+N}$ in the MPC's objective function as per equation (4.7a), where $\lambda_s > 0$ is the weight of the track progress at the prediction horizon, $t+N$. A value of λ_s close to zero translates to a controller that does not prioritize as much the track progress, while a large value for λ_s translates to a controller that greatly prioritizes the track progress at the prediction horizon.

Furthermore, in the cost function, the parameters that control the approximation of the track progress at a given prediction instant, s_k, $k \in [t, ..., t+N]$, are denoted by $\alpha_{CL} > 0$ and $\alpha_L > 0$. Recall that, as suggested from equations (4.6) from section 4.1, for a good approximation of $s \approx \hat{s}$ and $e_{CL} \approx \hat{e}_{CL}$, the absolute values of $\hat{e}_{CL}$ and $\hat{e}_L$ should be minimum. For this reason, the terms $\alpha_{CL}\hat{e}^n_{CL_k} + \alpha_L \hat{e}^2_{L_k}$, $k \in [t, ..., t+N]$, $n \in \{2, 4, 6, 8, ...\}$ (positive even number), are introduced in the cost function. Since, as suggested from figure 4.3, for a value of $\hat{s}$ close to the real track progress value, s, $\hat{e}_L$ is more sensitive to small variations of $\hat{s}$ than $\hat{e}_{CL}$ and also since it is not necessarily desired to penalize the center line error, $\hat{e}_{CL}$, in general a choice of α_{CL} and α_{CL} such that $\alpha_L \gg \alpha_{CL}$ is preferred. For the same reason, n, is chosen to be a positive even number since, for small values of center line error, a higher exponent will result in a lower cost as suggested in figure 4.4. Thus, by tuning both α_{CL} and n, the controller is able to get good approximations for the track progress at a given prediction stage, s_k, while still computing optimal trajectories that do not heavily penalize the center line error. It can be further noted that, in the cost function, at the current time instant t the only terms being penalized are these center line projection terms, as the states and control actions at this time instant are considered to be fixed. For this reason, cost function terms regarding the vehicle states and control actions are introduced only between $t+1$ and $t+N$.

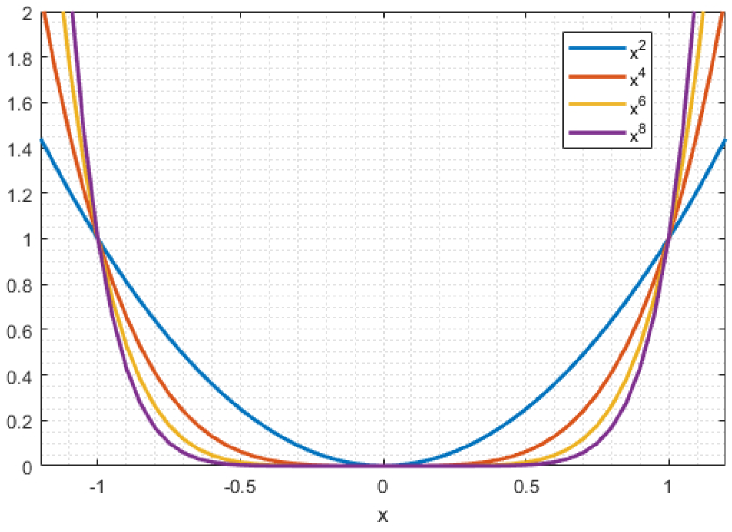


Figure 4.4: Effect of the exponent of the center line error cost.

Still related to the cost function, to limit the vehicle velocity, either to a given maximum velocity related to safety or to the vehicle's physical maximum velocity, the term $e^{q_{v_{\max}}\left(v_{x_k} - v_{\max}\right)}$, $k = t+1, .., t+N$ is added. This term translates to a soft constraint on the vehicle's maximum velocity, parameterized by $v_{\max}$. The parameter $q_{v_{\max}} > 0$ controls the steepness of the exponential function: a larger value of $q_{v_{\max}}$ results in a steeper exponential function, i.e., the constraint gets harder and smaller value of $q_{v_{\max}}$ results in a flatter exponential function, i.e., the constraint gets softer. To better understand the effect of

$q_{v_{\text{max}}}$ on the maximum velocity soft constraint cost refer to figure 4.5. Prior to using this soft constraint, a hard constraint was being used for the same purpose, however, upon performing simulations while using the hard constraint it was verified that jerky control actions, both in steering and throttle, were being applied when the vehicle's velocity was close to v_{max}. It was also verified that this happened due to model mismatch: since there was a slight model mismatch, the vehicle's velocity would slightly surpass v_{max}, as such, the control system would be at an unfeasible point, and, as a result, the control actions to be applied were the ones that would slow the vehicle down as much as possible, resulting in the observed jerky control actions. Hence, a soft constraint is preferred over a hard constraint for the maximum velocity.

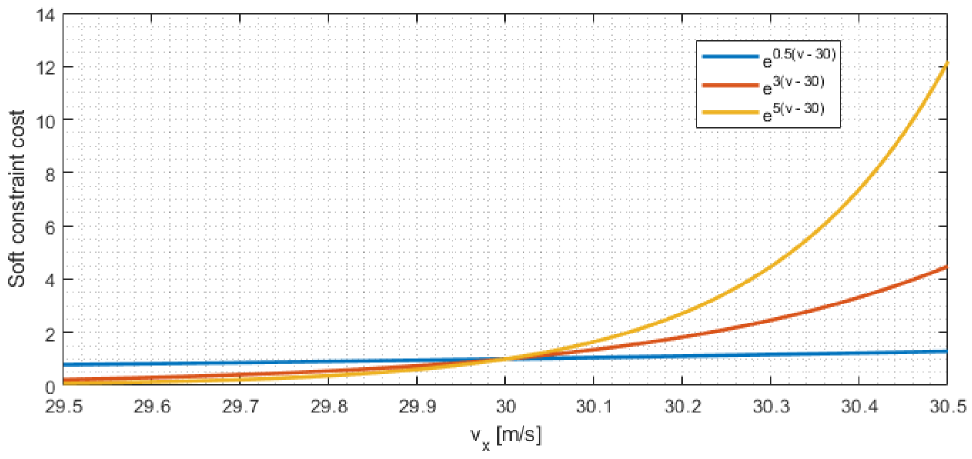


Figure 4.5: Effect of $q_{v_{\text{max}}}$ in the maximum velocity soft constraint cost.

The remaining parameters in the cost function of the MPC problem serve to control the aggressiveness of the MPC. Moreover, the parameter β_δ acts on the steering variation (i.e. the *steering rate*) between two instants to ensure a smooth steering control action. The parameter q_{v_y} penalizes high lateral velocities. This parameter is especially important to control possible sideways drifting of the vehicle as the vehicle may lose control when drifting if the model mismatch is high. All parameters that refer to the MPC cost function should take positive values.

As for the constraints, equation (4.7b) is the vehicle model state transition function constraint. Note that the model correction parameter vector, $\mathbf{p}_t$ is considered to be constant throughout the horizon as at time instant t this term includes the most up to date parameters. For further information about the vehicle model refer to chapter 3. Equations (4.7c) to (4.7e) are the initial constraints, i.e., the states measured at time instant t, the control actions being applied at time instant t, and the previous instant track progress, respectively. Equation (4.7f) represents control action variation constraints to ensure a realistic smooth control action, where $\Delta\mathbf{u}_{\max} = [\Delta d_{\max}, \Delta\delta_{\max}]^T$. Equation (4.7g) represents control action constraints representative of the maximum and minimum throttle values as well as the maximum and minimum steering angles of the vehicle, which values can be specified by $\mathbf{u}_{\min} = [-1, -\delta_{\max}]$ and $\mathbf{u}_{\max} = [d_{\max}, \delta_{\max}]^T$. Note that the minimum allowed throttle is -1 and the maximum is $d_{\max}$ since, in general, a deceleration is safer than accelerating, as a deceleration ultimately leaves the vehicle stationary. For this reason, full deceleration, $d = -1$, is allowed whereas the vehicle acceleration is limited by $d_{\max} \in]0, 1]$ in order to control the vehicle's maximum acceleration. Finally, $\delta_{\max}$ represents the physical steering limits of the vehicle. Equation (4.7h) is the track constraint that ensures that the vehicle stays within a fixed distance from the center line. On average, the track width of FS competitions is 2 meters. Equation (4.7i) is a constraint that forces neighboring track progress values to be within a given range of each other. Recall that the main objective of this controller is to maximize the track progress at the prediction horizon. As the center line parameterization function, $\mathbf{g}$, is periodic, with a period equal to the center line length, function $\mathbf{g}$ verifies the following: $\mathbf{g}(s) = \mathbf{g}(s + i \cdot \text{center\_line\_length})$, $\forall i \in \mathbb{Z}$. Thus, without this constraint, the optimization algorithm would always be able to find a greater value, more precisely a multiple of the track length apart, for the final stage track progress, not converging.

As such, this formulation employs a set of run time parameters and a set of fixed parameters. The run time parameters, denoted by the variables α_{CL}, $d_{\max}$, q_{v_y}, n and β_δ, are meant to be tuned for a given track as the vehicle drives itself. These parameters will be learned by the method presented in chapter 5. The fixed parameters, mainly determined by the physical properties of the vehicle are summarized in table B.3 in appendix B. The parameters $\Delta s_{\min}$ and $\Delta s_{\max}$ were obtained by assuming that the vehicle can follow the center line with a velocity between 2 and 30 m/s, respectively, by multiplying these velocities by the sampling time, T_s. The prediction horizon was chosen such that the vehicle is able to react to an incoming obstacle when traveling at maximum velocity. In other words, it is desired that the spatial prediction horizon is greater or equal to the maximum braking distance - equivalent to fully braking at the maximum speed. The maximum braking distance is roughly 40 m, as such, using $N = 40$ at $T_s = 0.05$ s, the spatial prediction at maximum velocity is $40 \cdot 0.05 \cdot 30 = 60$ m, which is greater than 40 m.

Furthermore, note that in order to obtain the MPC's optimization problem solution some solver must be used in order to solve the MPC's problem in real time. However, due to the solver's complex computations, the solution to the MPC optimization problem is only available some time after the sampling instant t. As such, the solution will only be available some time between t and $t+1$. For this reason, it would be impossible to get the current states, solve the MPC's problem and apply the control action corresponding to the sampling instant t, as that would require some method to instantly obtain the optimal solution. To overcome this issue, the next sampling instant control action $\mathbf{u}_{t+1}$, as computed based on the known $\mathbf{x}_t$ and $\mathbf{u}_t$ and determined between t and $t+1$, is applied at the next sampling instant, $t+1$, i.e. at the corresponding sampling instant. A disadvantage of this method is that this is equivalent to adding an input time delay equal to T_s in the control loop. Despite this disadvantage, it was found to be the next best solution to overcome the computation time issue. This control loop technique is summarized by algorithm 6 in appendix A. Furthermore, in section 6.3.1, the employed MPC solver will be introduced in which details of some of the terms used in algorithm 6 will be given.

Chapter 5

Controller Parameter Learning

In this chapter, a method for automatically tuning the parameters of the MPC presented in chapter 4 - MPC Formulation is developed. The inner workings of this method are based on Reinforcement Learning (RL) resourcing to the Genetic Algorithm (GA), which has been introduced in section 2.3.

The developed MPC's cost function and constrains are parameterized by parameters that directly relate to safety and aggressiveness of controller. As such, in summary, this method aims at maximizing a reward function that is based on the Trackdrive event lap times by testing several combinations of these parameters. The smaller the obtained corrected lap time, the higher the reward and vice versa. The term ”corrected lap time” refers to the raw lap time, i.e., the time between two start line crossings, plus possible penalties, namely cones that were hit by the vehicle.

5.1 Reward System

This section's goal is to present a comprehensive and detailed explanation behind the design of the parameter learning reward function for learning the MPC design presented in chapter 4 in a Reinforcement Learning (RL) environment. Designing a reward function is in general task-specific for a given RL problem. In general, the behavior of a reward function should reflect what is desired for the agent to learn.

For the purpose of learning the controller's design in the autonomous racing context, one can design a reward system based on the time the vehicle takes at driving in track segments. These track segments are divided by check points marked along the track. Given this, a reward system can be developed such that if the time the vehicle takes traveling a track segment is large the reward should be small. Otherwise, if the time the vehicle takes traveling that track segment is small, the reward should be large. This reward value would then be linked to the respective MPC parameters, denoted as $\mathbf{d}$, that were being applied to the MPC when driving through that particular track segment. However, due to model mismatch, there can be some parameters that may result in the vehicle colliding with cones. As cone collisions result in penalties in FS competitions, this is not desired. Consequently, the reward function should also reflect whether cones were hit in a given track section while using some parameters, $\mathbf{d}$, or not. The reward obtained in each segment would then be compared in order to retain the parameters that maximize the reward.

Note, however, that dividing a closed loop track into different segments is not a trivial task. For example, a track section characterized by a straight line is not comparable to a track section described by a sharp curve. This could be solved if the track was segmented into a single segment (i.e. the segment would be the track itself) as testing different parameters in the same track is comparable. However, this would result in a slow learning process as the new parameters to be tested would be applied to the controller at the beginning of the track and the respective reward obtained after the vehicle completes that lap. Nevertheless, if there was a mechanism that would make different track segments comparable, more than one parameter vector could be tested within the same lap, accelerating the learning process. With the objective of finding such a mechanism, the following method is proposed for making different track sections comparable. This method is based on two principles: an estimation of where the check points should be located such that the expected time the vehicle takes at driving each segment is equal between all segments and a normalization of the times obtained in each segment.

5.1.1 Vehicle Point Mass Model

The method presented in this section aims at estimating the check point locations along the track such that the expected time the vehicle takes at driving each segment is equal between all segments. This method is based on a simple point mass model of the vehicle which only purpose is to estimate these check point locations. This model takes the track and vehicle parameters as input and is based on the following principles:

1. The vehicle travels along the track's center line always heading towards the center line direction;
2. The vehicle always travels at its maximum speed, which is either limited by the maximum allowed centripetal force or the vehicle's physical maximum velocity.

Given these principles, the mathematical equation that describes the vehicle's maximum velocity at a given center line length, $v\left(s\right)$, is expressed in equation (5.1). In this equation, $D_{f/r}$ is the maximum value of the lateral force in the front and rear tires, respectively, as introduced in section 3.1.2 - Lateral Tire Forces. Moreover, m stands for the mass of the vehicle and $v_{\max}$ the maximum velocity the vehicle can achieve in a straight line.

$$v\left(s\right) = \min\left(v_{\max}, \sqrt{R\left(s\right)\frac{2D_f + 2D_r}{m}}\right) \tag{5.1}$$

Furthermore, $R\left(s\right)$ represents the track's curvature radius at a given center line length. The curvature radius can be computed from the previously defined center line parameterization function derivatives, which are defined in equations (4.2) in page 49 from section 4.1. The mathematical definition of the curvature radius obtained from these derivatives is expressed in equation (5.2) [68].

$$R(s) = \frac{1}{\left|\frac{\partial g_X}{\partial s}(s)\cdot\frac{\partial^2 g_Y}{\partial s^2}(s) - \frac{\partial g_Y}{\partial s}(s)\cdot\frac{\partial^2 g_Y}{\partial s^2}(s)\right|} \tag{5.2}$$

Note that, when the track's center line is a straight line, the second derivatives tend to zero, as such, the radius tends to infinity. In this case, the vehicle's maximum achievable velocity is its maximum straight line velocity as per equation (5.1). Otherwise, its velocity will be limited by the maximum allowable lateral centripetal force, which is dependent on the vehicle's mass and tire parameters.

Given a track and the required vehicle parameters, one can plot the maximum achievable velocity as seen in figure 5.1 represented by the green line.

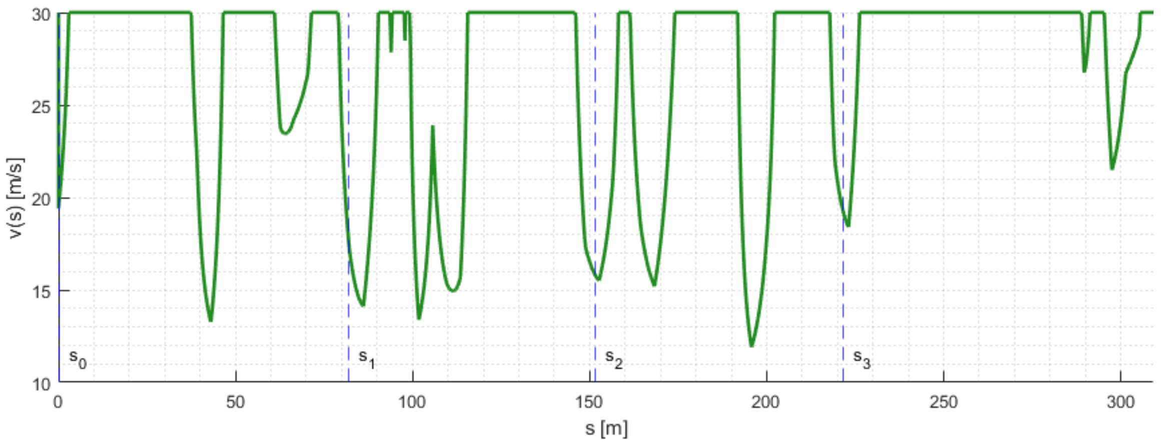


Figure 5.1: Maximum achievable velocity and track segmentation using 4 check points for the track shown in figure 4.2.

Still referring to figure 5.1, the blue dashed lines represent the check point locations that, according to the point mass model, require the same time for the vehicle to travel each segment. The first check point, s_0, coincides with the track starting line, i.e., $s_0 = 0$. Considering L as the track length for the given track, an estimation of the lowest achievable lap time, T_{lap}, can be obtained as follows:

$$T_{\text{lap}} = \int_0^L \frac{1}{v(s)}\,ds \tag{5.3}$$

Then, considering the number of desired check points for track segmentation, $N_{\text{CP}} \in \mathbb{N}$, one can obtain the remaining check point locations by solving for $s_i \;\forall\, i = 1, ..., N_{\text{CP}}-1$ the following system of equations:

$$\int_{s_{i-1}}^{s_i} \frac{1}{v(s)}\,ds = \frac{T_{\text{lap}}}{N_{\text{CP}}} \quad , \quad s_0 = 0 \quad , \quad i = 1, ..., N_{\text{CP}} - 1 \tag{5.4}$$

One could argue that another method for check point placement would be to place these check points equally spaced throughout the track. However, this check point placement method aims at placing the check points equally separated in time. The reason behind this is that a curved segment of the track becomes more comparable with a straight line segment, as by using track segments that, in theory, require the same time for the vehicle to travel, the parameters being tested in each segment are also tested for an equal time amount. This behavior can be observed in figure 5.1: the last track segment defined by s_3 and the end of the track is longer than, for example, the second segment defined by s_1 and

s_2 as, in this segment, on average, the vehicle velocity is lower than in the last segment. In other words, the second segment contains more curves than the last segment, nevertheless, both segments are expected to take the same time to travel. Furthermore, if the track segments would require different times to travel, these time measurements themselves could be affected due to the discrete time resolution used to time the segments.

5.1.2 Reward Function

The designed reward function is built using the track segmentation method presented in section 5.1.1. The reward function, $r : \mathbb{R}^+ \times \mathbb{Z}_0^+ \to \mathbb{R}^+$, is mathematically defined as follows:

$$r\left(T_{i,k}, n_i^{\text{cones}}\right) = \exp\left(-k_T\left(T_{i,k} - T_{i,k}^{\text{avg}} + k_c n_i^{\text{cones}}\right)\right) \tag{5.5}$$

where $T_i \in \mathbb{R}^+$ is the time measurement of segment i, i.e., the time the vehicle needed for traveling segment i and n_i^{cones} the number of cones that were hit in segment i. Furthermore, $T_{i,k}^{\text{avg}}$ represents the average time measurement of segment i, which will be further detailed. The parameters $k_T > 0$ and $k_c > 0$ are meant to shape the reward function. To understand the idea behind the development of this reward function, consider the time difference of $T_{i,k} - T_{i,k}^{\text{avg}}$ as shown in equation (5.5), with no cones being hit in segment i, $n_i^{\text{cones}} = 0$. Note that, being k_T a positive number, if the time measured in segment i is larger than the average time taken in that segment, $T_{i,k} - T_{i,k}^{\text{avg}} > 0$, the reward function will take a small value. Otherwise, if the time measured in segment i is smaller than the average time taken in that segment, $T_{i,k} - T_{i,k}^{\text{avg}} < 0$, the reward function takes a larger value. In other words, if the current controller parameters being tested in segment i resulted in a smaller travelling segment time than the average, those parameters are attributed a higher reward value, as they resulted in a faster than average lap time, otherwise, they are attributed a smaller reward. In addition to this, if the vehicle hits cones in segment i, the number of cones that were hit translates into a time penalty, which will further reduce the received reward value. As such, the parameter k_c can be read as the time penalty per cone hit in a given segment. Regarding k_T, this parameter mainly affects the slope of the exponential function. The reward function will become more sensitive to the measured time difference as k_T increases and less sensitive as it decreases.

Regarding the segment's average time, $T_{i,k}^{\text{avg}}$, one could attain this value using several methods. For this purpose, it was chosen to use a low pass filter to retain the average time of each segment. This is accomplished using the following equation:

$$T_{i,k+1}^{\text{avg}} = \lambda_T T_{i,k} + \left(1 - \lambda_T\right) T_{i,k-1}^{\text{avg}} \quad , \quad i = 1, ..., N_{\text{CP}} \tag{5.6}$$

As such, $T_{i,k}$ can be read as the current time measurement, with index k, of segment i. The indexes $k+1$ and $k-1$ refer to the next and previous time average values. The parameters $\lambda_T \in [0, 1]$ regulates the intensity of the filter, similarly to the λ_{MSE} parameter introduced in equation (3.47) in page 46. Then, a value of λ_T close to zero results in a smoother T^{avg} but with slower response times, and a value of λ_T

close to one results in a less filtered T^{avg} but with faster response times.

Note that, according to the point mass model presented in the previous section, the average times across all segments, $T^{\text{avg}}_{i,k}$, $i = 1, ..., N_{\text{CP}}$, should all be equally valued due to this method's check point placement. However, in practice, the vehicle model is much more complex than the point mass model presented previously, thus, the average times across all segments may be different. Hence the necessity of using an average time for each segment in the reward function.

5.2 Genetic Algorithm for Learning the Controller Design

In this section, the Genetic Algorithm (GA) will be applied in a Reinforcement Learning (RL) environment for learning the controller design.

In order to apply the GA, one must first define the genetic code string. The MPC defined in chapter 4 is mainly parameterized by the following performance affecting set of parameters: α_{CL}, α_L, $d_{\max}$, q_{v_y}, n, β_δ, $q_{v_{\max}}$, $v_{\max}$ and λ_s. For an explanation of the effect of each of these parameters on the controller recall section 4.2. Some of these parameters are related to the vehicle center line projection or to the vehicle's capabilities, like its maximum velocity. As such, α_L, $q_{v_{\max}}$ and $v_{\max}$ are fixed parameters which are not meant to be tuned, since $q_{v_{\max}}$ and $v_{\max}$ directly relate to vehicle properties and α_L only affects the vehicle's projection mechanism on the center line. Furthermore, λ_s was chosen to be a fixed parameter as well since this parameter may be considered as a redundant parameter, as increasing or decreasing this parameter is equivalent to decreasing or increasing the remaining parameters in the optimization problem. For these reasons, the remaining parameters are classified as tunable parameters, as such, the MPC parameter vector is described as $\mathbf{d} = \begin{bmatrix}\alpha_{CL}, d_{\max}, q_{v_y}, n, \beta_\delta\end{bmatrix}^T$, which will also be used as the genetic code string for the GA. Recall that these parameters are positive real numbers except for n, which should be a positive even number.

Considering a population with N_{pop} individuals, the population matrix, $\mathbf{D}$, which columns represent an individual and rows a given gene, can be defined as:

$$\mathbf{D} = \begin{bmatrix}\mathbf{d}_1 & \dots & \mathbf{d}_N\end{bmatrix} \tag{5.7}$$

Given the population matrix and the reward values of each individual, one can generate a new population, i.e., new generation resourcing to genetic operations: reproductions, crossovers, and mutations. The following sections will define these genetic operations in detail.

5.3 Genetic Operations

This section will define the genetic operations used for the GA aiming at learning the controller design.

5.3.1 Reproductions

Given an individual randomly chosen from the population with a probability according to the fitness vector, referred to as individual i, $i = 1, ..., N_{\textbf{pop}}$, the individual passes directly to the new generation. This operation aims at retaining the characteristics of the best individuals in the case the generated individuals of the remaining operations are worse than the original population. Mathematically, this is accomplished by passing directly a column of the old population matrix to the new population matrix as suggested by the following equations:

$$\mathbf{d}_i^{\text{old}} = \left[\alpha_{CL}^i, d_{\max}^i, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.8a}$$

$$\mathbf{d}_i^{\text{new}} = \left[\alpha_{CL}^i, d_{\max}^i, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.8b}$$

5.3.2 Crossovers

Given two individuals, referred to as ”parent” individuals, randomly chosen from the population with probability according to the fitness vector, i.e., two different columns chosen from the population matrix, $\mathbf{D}$, the reproduction is an operation with exploitative characteristics aiming at generating two new individuals, referred to as ”children” individuals, that take the best characteristics of each parent, resulting in two new improved individuals. For this purpose, consider the two parents i and j, $i \neq j$, $i, j = 1, ..., N_{\text{pop}}$, characterized by their genetic code:

$$\mathbf{d}_i^{\text{parent}} = \left[\alpha_{CL}^i, d_{\max}^i, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.9a}$$

$$\mathbf{d}_j^{\text{parent}} = \left[\alpha_{CL}^j, d_{\max}^j, q_{v_y}^j, n^j, \beta_\delta^j\right]^T \tag{5.9b}$$

The reproduction operation starts by randomly choosing a gene index from a uniform discrete distribution between the second gene index in the genetic code, 2, and the last gene index in the genetic code, in this case, 5. In other words, this index can be any given integer number between 2 and 5 with probability $1/4$. For illustration purposes, considered that the index 3 was selected. Then, the newly generated children would be characterized as follows in equations (5.10), where child i contains the genetic code of parent i until the second gene and the remaining genetic code of parent j. On the contrary, child j contains the genetic code of parent j until the second gene and the remaining genetic code of parent i. In summary, the children are generated by swapping the parents' genetic code between the second and third gene:

$$\mathbf{d}_i^{\text{children}} = \left[\alpha_{CL}^i, d_{\max}^i, q_{v_y}^j, n^j, \beta_\delta^j\right]^T \tag{5.10a}$$

$$\mathbf{d}_j^{\text{children}} = \left[\alpha_{CL}^j, d_{\max}^j, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.10b}$$

Note that if the selected index corresponded to the first gene, the children would have the same genetic code as their parents. Hence the index selection starting at the second gene.

5.3.3 Mutations

Given an individual randomly chosen from the population with probability according to the fitness vector, referred to as individual i, $i = 1, ..., N_{\text{pop}}$, the mutation is characterized by randomly changing one gene on the individual and inserting the newly mutated individual in the new generation. This operation reflects an exploratory character by randomly trying a new gene combination hoping that the mutation results in a better performing individual than the original individual. Mathematically, this operation is characterized by randomly choosing a gene index from a uniform discrete distribution between the first gene, 1, and the last gene index, in this case, 5. In other words, this index can be any given integer number between 1 and 5 with probability $1/5$. For illustration purposes, assume that the second index is selected. Then, the newly mutated individual would be generated as follows:

$$\mathbf{d}_i^{\text{old}} = \left[\alpha_{CL}^i, d_{\max}^i, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.11a}$$

$$\mathbf{d}_i^{\text{new}} = \left[\alpha_{CL}^i, d_{\max}^i + m, q_{v_y}^i, n^i, \beta_\delta^i\right]^T \tag{5.11b}$$

where m represents a random variable obtained by some probability distribution function. For the case of the selected index affecting a real valued parameter, like α_{CL}, $d_{\max}$, q_{v_y}, or β_δ, as is the case of the example given in equations (5.11), m is obtained by a continuous uniform distribution between some predefined values m^{low} and m^{high}, which can be defined by the probability density function f_{real}:

$$f_{\text{real}}\left(m, m^{\text{low}}, m^{\text{high}}\right) = \begin{cases} \frac{1}{m^{\text{high}} - m^{\text{low}}} & , m^{\text{low}} \leq m \leq m^{\text{high}} \\ 0 & , \text{otherwise} \end{cases} \tag{5.12}$$

For the case of selecting the index affecting the n parameter - an even number - m is obtained through a uniform discrete probability density function which can make n jump to the next even number or to the previous even number. This probability density function, denoted as f_n, is defined as follows:

$$f_n(m) = \begin{cases} 0.5 & , m = 2 \\ 0.5 & , m = -2 \\ 0 & , \text{otherwise} \end{cases} \tag{5.13}$$

5.4 Algorithm Procedure

In this section, a general explanation of how the previous methods are used in the GA will be given. Recall that the track is segmented into N_{CP} segments (which are divided by N_{CP} check points) and consider a population of N_{pop} individuals.

When the vehicle is racing and crosses a check point, segment i corresponding to that check point is timed, resulting in $T_{i,k}$ and n_i^{cones}. The reward value for that timing and number of cones hit is computed as per equation (5.5). At the same time, the average time of that same segment is adjusted as per equation (5.6). With these quantities computed, as the next segment begins, the MPC parameters are updated to the next column of the population matrix, D. When the vehicle crosses the next checkpoint, this process is repeated. When the reward that corresponds to the last population individual, i.e., the final column of the population matrix, is computed, every individual in the population will have its respective reward value, r_j, $j = 1, ..., N_{\text{pop}}$. As such, a new population can be generated.

To generate a new population, the fitness vector must firstly be computed. As the reward function is always positive, $r > 0$, the fitness vector, $\mathbf{f}$, can be generated from the normalization of the reward value, as follows.

$$f_j = \frac{r_j}{\sum_{n=1}^{N_{\text{pop}}} r_n} \quad , \quad j = 1, ..., N_{\text{pop}} \tag{5.14}$$

With the fitness vector generated, the number of each genetic operation to perform is generated. Let R, C and M denote the reproduction, crossover and mutation operations, respectively. The number of reproductions, N_{R}, crossovers, N_{C}, and mutations, N_{M}, can be stochastically generated according to a discrete probability distribution function with probabilities $\mathrm{P}\left(\mathrm{R}\right)$, $\mathrm{P}\left(\mathrm{R}\right)$ and $\mathrm{P}\left(\mathrm{M}\right)$. In other words, being O a random variable that represents one genetic operation, O can be generated according to the following probability density function, $f_{\text{operation}}$:

$$f_{\text{operation}}\left(O\right) = \begin{cases} \mathrm{P}\left(\mathrm{R}\right) & , O = \mathrm{R} \\ \mathrm{P}\left(\mathrm{R}\right) & , O = \mathrm{C} \\ \mathrm{P}\left(\mathrm{M}\right) & , O = \mathrm{M} \\ 0 & , \text{otherwise} \end{cases} \tag{5.15}$$

After selecting N_{pop} operations, N_{R}, N_{C} and N_{M} are defined by recursively using the probability function of equation (5.15) N_{pop} times. Note that one must ensure that the N_{C} is an even number since the crossover operation requires the use of pairs of parents. For the purpose of this work, N_{C} is checked for being even. If this number is odd, N_{C} is subtracted by one and one is added to the number of mutations, N_{M}. Then, the genetic operations are performed by selecting individuals stochastically according to their fitness value. As such, considering individual $j \in \{1, ..., N_{\text{pop}}\}$ as the j^{th} column of the population matrix, D, as explicit in equation (5.7), for the reproduction and mutation operations, individuals are selected according to the following probability function, $f_{\text{selection}}$:

$$f_{\text{selection}}\left(j\right) = \begin{cases} f_j & , j = 1, ..., N_{\text{pop}} \\ 0 & , \text{otherwise} \end{cases} \tag{5.16}$$

Regarding crossovers, a similar method is used with a slight modification to ensure that the two selected parents are not the same individual. As such, the first parent $p^1 \in \{1, ..., N_{\text{pop}}\}$ is selected from the probability function of equation (5.16). Then, one must rearrange the fitness vector such that

p^1 is not selected again. As such, given the first parent index, p^1, the second parent index, p^2, can be selected according to This can be done according to the following probability function, $f_{\text{parent\_2}}$:

$$f_{\text{parent\_2}}\left(p^2|p^1\right) = \begin{cases} \frac{r_{p^2}}{\left(\sum_{n=1}^{N_{\text{pop}}} r_n\right) - r_{p^1}} & , p^2 \in \{1, ..., N_{\text{pop}}\} \setminus \{p^1\} \\ 0 & , \text{otherwise} \end{cases} \tag{5.17}$$

Note that this probability function, in equation (5.17), is defined as if a new fitness vector was defined after considering that the reward of p^1 is zero.

After performing every genetic operation using the original population, the new generation's individuals are created. These individuals are then shuffled such that each individual receives a random track segment. Finally, they are inserted into the new population matrix which describes the new generation. This whole process is then repeated for the new generation. As the number of generations increases the individuals' genetic code should, statistically, result in iteratively better MPC parameters.

As a further note regarding the mutation operation, an extra step was added to ensure that the parameters remain valid. Recall that mutations are responsible for exploring the parameter space, however, for safety purposes and to ensure the validity of the parameters, the parameter exploration space was limited such that $\mathbf{d}^{\min} \leq \mathbf{d} \leq \mathbf{d}^{\max}$. In other words, a constraint is added such that each parameter in the parameter vector, $\mathbf{d}$, must be contained within an interval, $\left[\mathbf{d}^{\min}\right]_i \leq d_i \leq \left[\mathbf{d}^{\max}\right]_i, \forall i = 1, ..., 5$. For notation purposes, to refer to the parameters, for example, $d^{\min}_{\alpha_{CL}}$ and $d^{\max}_{\alpha_{CL}}$ will refer to the minimum and maximum values that α_{CL} can take after being mutated, respectively, $d^{\min}_{d_{\max}}$ and $d^{\max}_{d_{\max}}$ will refer to the minimum and maximum values that $d_{\max}$ can take after being mutated, respectively, and so on for the remaining q_{v_y}, n and β_δ variables.

Moreover, the mutation ranges as per equation (5.12), m_{low} and m_{high}, for the real valued parameters, i.e., for α_{CL}, $d_{\max}$, q_{v_y}, or β_δ are also defined for each one of these parameters. In other words, for the α_{CL} variation range should be between $m^{\text{low}}_{\alpha_{CL}}$ and $m^{\text{high}}_{\alpha_{CL}}$, the $d_{\max}$ variation range should be between $m^{\text{low}}_{d_{\max}}$ and $m^{\text{high}}_{d_{\max}}$ and so on for the remaining q_{v_y} and β_δ parameters. By customizing this parameter variation for each one of the real valued parameters one can manipulate both the expected value and the standard deviation of the variation of each parameter. This may be useful to accelerate learning as the starting MPC parameters, i.e. the initial population, should be formed by conservative parameters that do not result in an aggressive control of the vehicle. In other words, the initial parameters should reflect high values of α_{CL}, q_{v_y} and β_δ and low values of $d_{\max}$ and n in order to ensure the vehicle starts in a safe controller configuration. With such parameters, it is expected that by slightly decreasing α_{CL}, q_{v_y} and β_δ or increasing $d_{\max}$ and n, the vehicle is able to drive faster as such variation would result in more aggressive parameters than the initial ones. As such, by manipulating the expected value of the α_{CL}, q_{v_y} and β_δ variations to be negative and the expected value of the $d_{\max}$ and n variations to be positive, on average, a mutation would result in slightly more aggressive parameters. Furthermore, one can define maximum and minimum variation values to be positive and negative, respectively, in order to allow a mutation to escape a parameter that is too aggressive, i.e., for the population to move to a safer parameter configuration if needed.

In summary, the new generation can be computed using a similar procedure to algorithm 4 presented in appendix A. Instead, an algorithm for the reward and average segment times computation will be presented to clarify this idea - algorithm 7 in appendix A. This algorithm should be applied the instant the vehicle crosses a new check point. Finally, the GA algorithm parameters can be consulted in table B.4 and the initial generation parameters in table B.5, both in appendix B. Furthermore, note that N_{CP} and N_{pop} can be different positive integer numbers, however, if one was to choose some $N_{\mathrm{CP}} < N_{\text{pop}}$ the time measurements of each individual would be influenced by the time measurements of other individuals in the same population since there are more individuals than track segments. In other words, if two individuals in the same population share the same segment, the average segment time of the second individual would be biased by the segment time of the first individual. Since this is not desired, the author recommends a choice of N_{pop} and N_{CP} such that $N_{\text{pop}} \leq N_{\mathrm{CP}}$, as this ensures that a segment is never shared between two individuals in the same population.

Finally, figure 5.2 contains simulation data showing that by using track segments instead of using the whole track as one single segment, for the same N_{pop}, using various segments may indeed lead to faster learning.

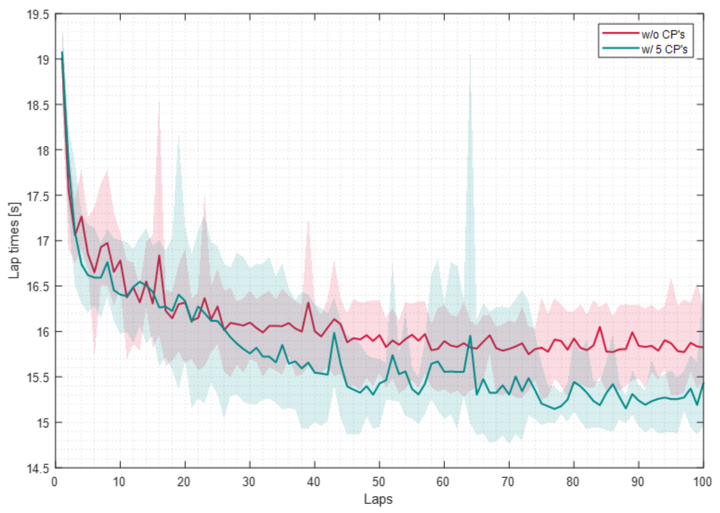


Figure 5.2: Example lap time reduction by using check points vs. not using check points.

Chapter 6

Implementation

In this chapter, details regarding the controller implementation will be described. Firstly, the simulation environment in which the controller was implemented and tested will be described. Then, the tracks used for performing simulation tests will be presented. Finally, the details of the learning MPC implementation are given.

6.1 Simulation Environment

The simulation of choice of FST Lisboa is FSSIM[1]. FSSIM is a high fidelity simulator developed by the AMZ Driverless FS team from ETH Zürich. This simulator was developed with the purpose of testing the vehicle pipeline in a virtual environment, which in turn allows for the adjustment of these algorithms prior to the implementation in the real vehicle. This team reported 1% lap time accuracy compared with their FSG 2018 Trackdrive run [42].

The simulator includes the standard Acceleration and Skidpad competition tracks. It also includes tracks that were mapped from the 2018 FS events from Italy and Germany. The simulator also features systems required for racing in the real competitions such as the Remote Emergency Stop (RES) activation when the vehicle's four wheels leave the track, time penalization when hitting a track cone, and a lap time counter.

To simulate raw sensor data, the simulator also features a cone sensor model that simulates the cone detections around the vehicle. Moreover, the employed vehicle model is a complex model which also results from a blend of a kinematic model for low velocities and a dynamic model for high velocities. The kinematic vehicle model used by the simulator is similar to the one presented in section 3.2. However, the dynamic model used by the simulator deeply details the various subsystems present in FS vehicles. Namely, as an example, the simulator models all the vertical and horizontal loads applied to all four tires, as well as the angular velocity of each wheel. This complex vehicle model is responsible for allowing a high fidelity match between the real vehicle and the simulated one.

By default, FSSIM contains the parameters of the AMZ's Gotthard vehicle. However, these param-

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

eters can be manually defined, allowing for the introduction of the FST10d parameters previously presented in table B.1 in appendix B. Furthermore, with the purpose of testing the controller's robustness, simulations were also performed using AMZ's Gotthard parameters. In order to compare both vehicles, some of the parameters of AMZ's Gotthard model are present in table B.6 in appendix B.

6.2 Tracks

As mentioned in the previous section, the FSSIM simulator includes various tracks from FS competitions. Namely, three of the included tracks will be used for simulation and testing. The track files contain essentially the coordinates of the various cones that form the track. Recall that orange cones mark the track's starting line which is also used for timing and the right and left limits of the track are marked by yellow and blue cones respectively. The tracks are mainly composed of sections designed to test the vehicle for a given driving scenario, namely slaloms - sequence of consecutive short left and right turns, hairpins - sharp curves, thin sections - where the track width is reduced, and straight lines. The layout of these tracks can be consulted in figure 6.1.

Regarding the first track, which will be referred to as Track 1, is the track from the FSG 2018 Trackdrive event. This Track, which can be seen in figure 6.1a is mainly characterized by a slalom around $X = -4$ m and $Y = -34$ m and a hairpin, around $X = 28$ m and $Y = -70$ m. Moreover, Track 2, from a FS Italy Trackdrive event is mostly characterized by hairpins as can be seen in figure 6.1c. Finally, Track 3, presented in figure 6.1b and nicknamed "thin" is not from a FS competition but is still relevant since this track, as its nickname suggests, contains track sections with reduced track width combined with hairpins and slaloms.

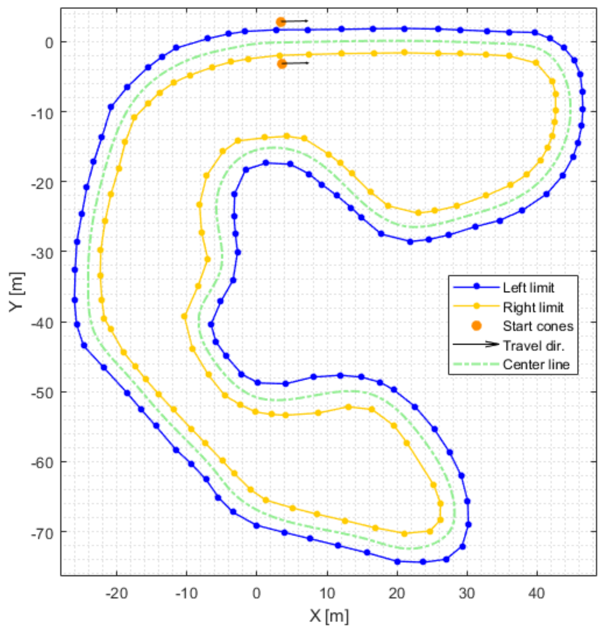
6.3 Controller Implementation

As referred previously in chapter 1 - Introduction - the developed controller should be implemented in the current FST Lisboa autonomous systems' software pipeline. As such, this section aims at clarifying the procedure involved in such implementation.

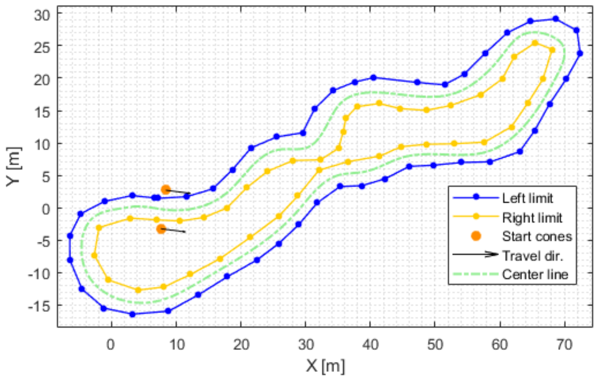
Overall, FST Lisboa autonomous systems' software pipeline is written in Python, C and C++ while resourcing to the Robot Operating System[2] (ROS). ROS is an open source set of software libraries and tools made for robotic applications. ROS mainly supports two programming languages: Python and C++. Due to the real time requirements, C++ was chosen since it is a compiled language that is capable of creating powerful and light weight software. For this implementation, three ROS nodes were created for: model correction learning, controller parameter learning, and another for the model predictive controller itself. A scheme of these nodes and their communications can be seen in figure 6.2.

Furthermore, note that the modularity offered by the control system in figure 6.2 allows for the activation or deactivation of the auxiliary MPC nodes if needed. For example, the control system can work

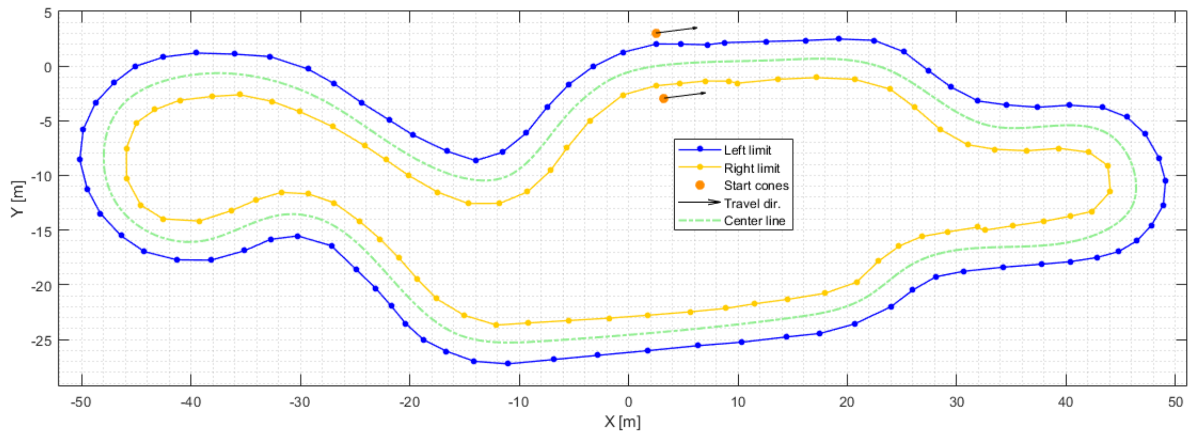
[2]https://www.ros.org/



(a) Track 1.



(b) Track 3.



(c) Track 2.

Figure 6.1: Tracks to be used in simulation and testing.

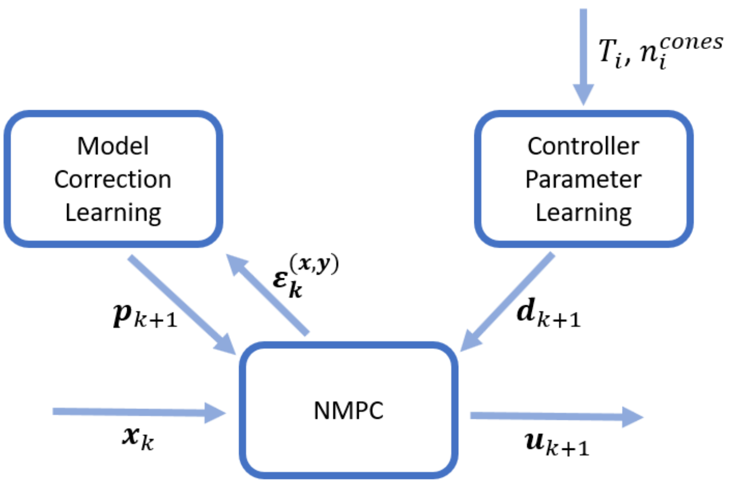


Figure 6.2: ROS nodes for the developed control architecture.

with the MPC node alone, the model correction learning node with the MPC, the controller parameter learning node with the MPC, or with all these nodes, as intended.

6.3.1 MPC Solver

For solving the MPC optimization problem a commercial solver was employed. FORCESPRO [69, 70], designed by Embotech AG, enables users to generate tailor-made solvers from a high-level mathematical description of an optimization problem. This commercial solver generates an optimized C code library which solves the MPC optimization problem formulated in equations (4.7) in chapter 4 - MPC Formulation. This C library, which is callable from C++, can then be embedded on many hardware platforms, and, moreover, on FST's autonomous systems' pipeline. FORCESPRO achieves this by extracting the MPC optimization problem's structure and automatically applying a suitable optimization algorithm, outputting the optimized and compact solver. Due to its compact size and optimized code, the generated solver is appropriate for solving the MPC optimization problem in real time.

The MPC's structure and the track are defined using FORCESPRO's MATLAB high-level interface[3]. The previously mentioned MPC optimization problem is a Nonlinear Programming (NLP) optimization problem, which can be solved using FORCESPRO NLP for non-convex finite-time non-linear optimal control problems. Additionally, using MATLAB's Symbolic Toolbox, the track spline function is generated, containing the center line coordinates and the center line direction as a function of the track length, and automatically embedded into the MPC problem. Furthermore, this interface also allows the algorithm to use real time parameters, i.e. allowing for the update of the vehicle model correction parameters as well as the MPC parameters themselves.

Additionally, these solvers can be generated with several user selected options. Namely, due to the real time solving requirement, the solver's timeout option can be exploited. The timeout option allows the user to add a time limit for the solving process. In other words, since the vehicle is being controlled at a fixed sampling time of T_s one can define the solver's timeout option to be slightly less than the sampling time (slightly less in order to allow other less time consuming operations to proceed). As such, this option ensures that the solver does not take over T_s to find the next control actions and that a new problem is ready to be solved at the next time instant. Furthermore, the solver also returns several flags which, in turn, have three different meanings: *flag 1* - the solver reached the optimal solution within the timeout, *flag 2* - the solver did not reach the optimal solution within the timeout and returned the best feasible solution found within the timeout, and *flag 3* - the solver was unable to find a feasible solution within the timeout. In order to correct for the possibility of the solver returning an infeasible solution (*flag 3*), a previously computed feasible control action is applied, i.e., the corresponding time instant control action solved at some previous time instant at which the solver returned *flag 1* or *flag 2*.

Other options that were used relate to the code optimization, which were set to optimize the code for the target platform as much as possible as well as enabling parallel computing, which allows for the selection of the number of Processing Unit (PU) threads dedicated to the solver in order to split the workload among the selected number of threads.

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

Figure 6.3 contains an example of the solver's computation time required obtained in the simulations. These results were obtained with four CPU threads dedicated to solving the MPC problem in a machine equipped with an Intel Core i7-3610QM, a quad core processor with eight threads that was released in the second quarter of 2012. Recall that the selected sampling time is $T_s = 50$ ms. As such, a timeout of 48 ms was selected, which explains the bump seen at the histogram's rightest values. The timeout value is slightly less than the sampling time in order to allow for other lightweight finishing operations to take place. Furthermore, the FST10d vehicle is equipped with an Intel i7-8700T, a six core processor with twelve threads that was released in the second quarter of 2018 which has much higher processing capabilities than the processor used for performing the simulations. Thus, the vehicle's PU should be able to achieve better solve times than the ones presented in figure 6.3.

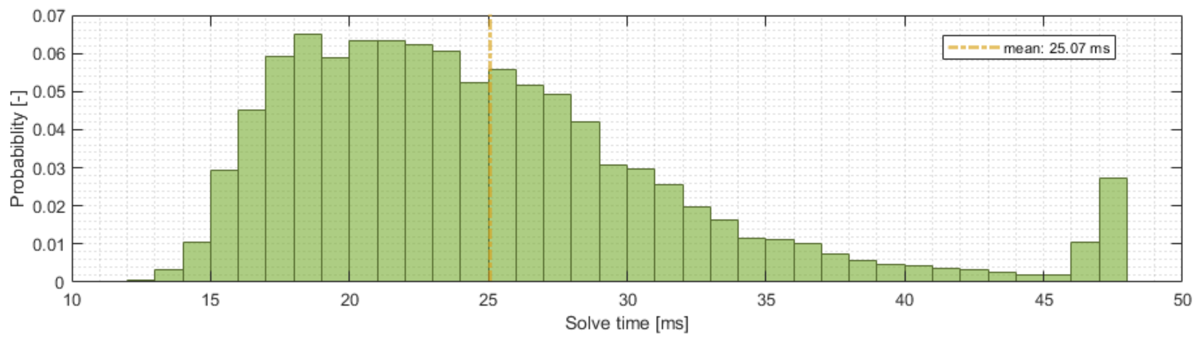


Figure 6.3: Example of the computation time histogram.

6.3.2 Model Correction

A neural network library for the application of the vehicle correction model, as in section 3.6, was developed. This library creates a MATLAB function that takes the neural network features and parameters as input and outputs the model corrections. Due to being a MATLAB function, the neural network model can be embedded directly onto the FORCESPRO solver using, once again, the high-level interface, which in turn allows for taking advantage of the FORCESPRO code generation to generate the optimized code for the full vehicle model, i.e., the blended model plus the correction model. Furthermore, as the training algorithm is made explicitly in C++, the developed MATLAB library also generates C++ code for the computation of the neural network jacobian, by taking advantage of MATLAB's Symbolic Toolbox for automatic differentiation. This is accomplished by, once again, generating an optimized MATLAB function and then automatically translating the MATLAB code into C++ code while resourcing to the Eigen[4] library.

Overall, to use this developed library, the user simply has to select the number of clusters, i.e., the number of neurons in the hidden layer of the Elliptical Basis Function Network. Then, the necessary code for implementation in FST Lisboa's pipeline is generated.

[4]https://eigen.tuxfamily.org/index.php?title=Main_Page

6.3.3 Controller Parameter Learning

Regarding learning the MPC parameters, the procedure described in chapter 5 - Controller Parameter Learning was directly implemented in the ROS node using C++. Moreover, this node is activated when the vehicle crosses a check point. As explained previously, this results in a given segment time and a number of cones that were hit. After this, it computes the individual's reward value and returns the next individual parameters to the MPC node, as per figure 6.2.

This module relies on the previously mention Eigen C++ library as well as the C++ standard library for random number generation. Recall that the Genetic Algorithm heavily relies on random numbers due to the stochastic genetic operations. Given this, various tests/simulations of this algorithm should be performed in order to prove some statistical validity of the algorithm. Contrary to the model correction learning node, this node does not require heavy algebraic operations. Nevertheless, the Eigen library was used for variable organization, for example, for keeping the population organized in the population matrix.

Chapter 7

Results

In this chapter, results obtained from simulations will be presented. Firstly results obtained with the simulator containing the FST10d parameters will be shown and discussed. Then, results using AMZ's Gotthard parameters will be shown in order to test the algorithm's robustness.

The results presented in this chapter are, in general, based on three different tests of the MPC: a test using the model correction node, a test using the controller parameter node learning, and a test using both the model correction and controller parameter learning nodes. Each test is composed of 100 laps performed in Track 1. Moreover, five simulations for each test were performed. As such, each line corresponds to the average of all five simulations and the colored area surrounding the respective line corresponds to the extreme values, minimums and maximums, obtained in each test.

Furthermore, the control algorithm is using the previous parameters presented in the previous tables - table B.3 contains the parameters used in the MPC, table B.2 the parameters for learning the model correction and tables B.5 and B.4 contain the initial parameter generation and the parameters used for learning the control design, respectively. All tables can be found in appendix B.

7.1 FST10d Simulations

In this section, results obtained in simulation by using the FST10d vehicle parameters in the simulator are shown and discussed. In other words, the simulator is using similar vehicle parameters to the ones in table B.1.

Figure 7.1 shows the lap times obtained for each lap for three different tests of the MPC: a test using the model correction node (in blue), a test using the controller parameter node learning (in green), and a test using both the model correction and controller parameter learning nodes (in red).

One of the first noticeable facts in figure 7.1 is that the blue line, corresponding to the model correction node plus the MPC, is basically constant throughout each lap. Moreover, this line shows negligible variance with respect to the average line. This happens for two main reasons: the controller parameters are kept constant throughout the test since there is no learning regarding the MPC parameters and that the simulator model closely matches the controller's predictive model. As such, the model correction node is pretty much inactive as the blended bicycle model describes the vehicle dynamics with enough accuracy. Recall that the tunable MPC parameters that are used in this test, expressed in table B.5, are parameters that are considered to be safe, i.e. parameters that do not result in overall aggressive control of the vehicle, like high lateral and longitudinal accelerations and decelerations.

The other two lines, regarding using the controller parameter node learning plus the MPC (in green) and using both the model correction and controller parameter learning nodes plus the MPC (in red), are observed to have similar behaviors. Approximately at lap 50, the results roughly achieve steady lap times: the green test achieves a 15.52 s steady state lap time and the red test a 15.27 s lap time. These steady lap times are rather close to each other, which is explained once again by the blended bicycle model matching well the simulator's model. Despite this, the steady lap time that uses both the controller parameter learning and model correction is slightly smaller than the one that uses the controller parameter learning alone. This happens for the following reason: as the controller parameters move to a more aggressive state, the vehicle mismatch becomes clearer as higher accelerations and decelerations start to take place. At this point, the model correction is able to learn this slight mismatch, resulting in more precise control of the vehicle which in turn also achieves faster lap times. Furthermore, note that these two lines are shown to have a higher variance when compared to the blue line. Recall that the controller parameter's learning algorithm, the Genetic Algorithm, heavily relies on randomly generated numbers, whereas the model correction node alone does not employ any stochastic algorithm. Thus, it is expected for these two lines to show higher variance than the blue line due to the stochastic nature of the Genetic Algorithm.

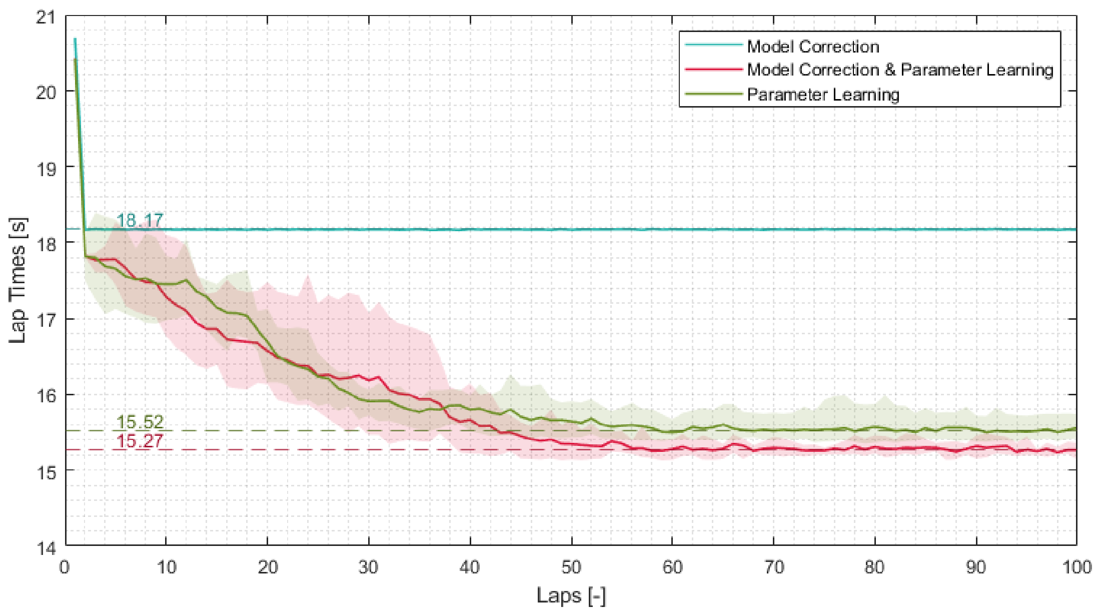
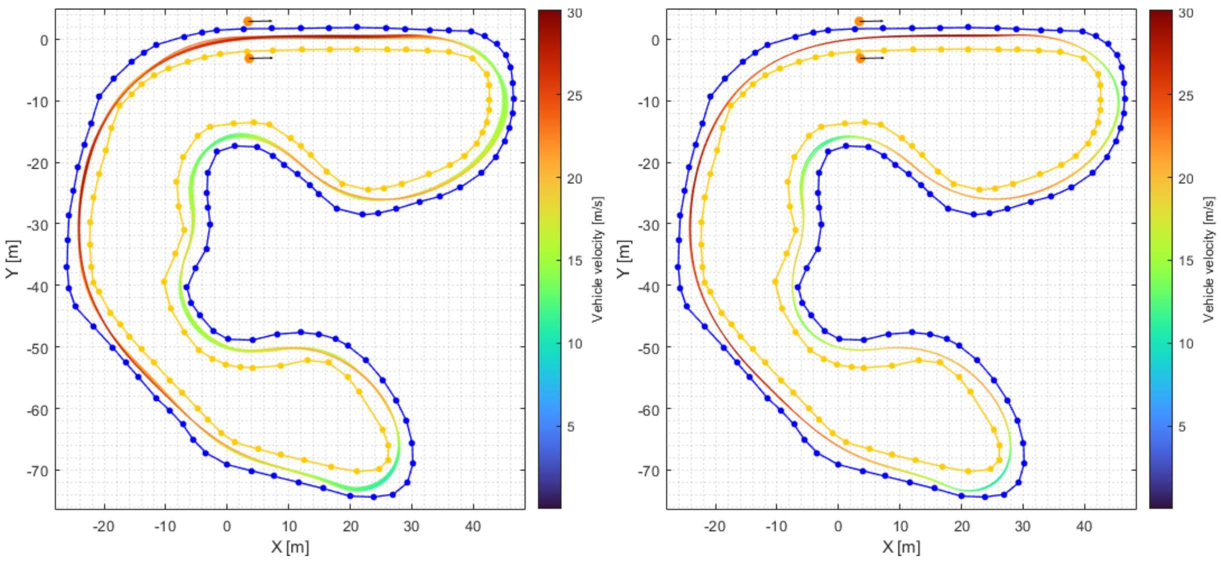


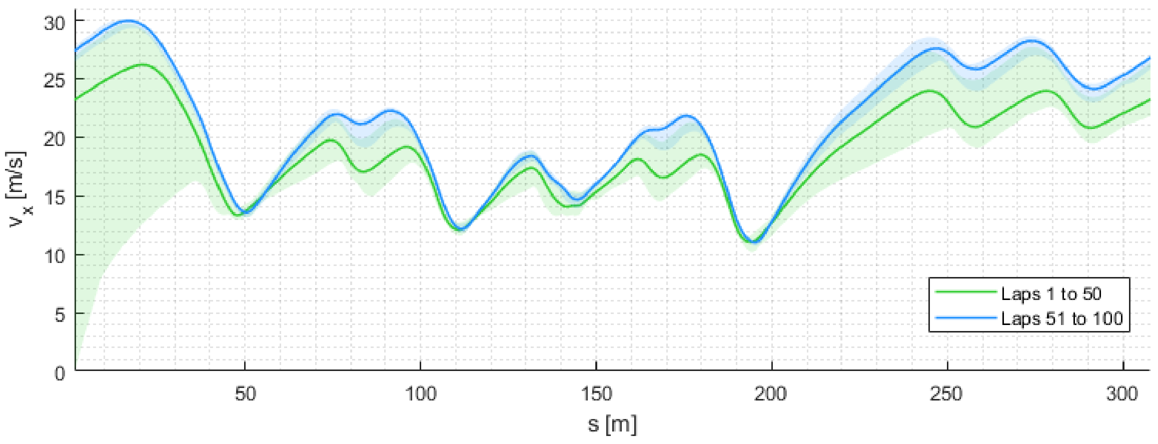
Figure 7.1: Lap time simulation results for the FST10d vehicle in Track 1.

Focusing now on the full algorithm test, the test using the model correction and controller parameter learning nodes plus the MPC, corresponding to the red curve in figure 7.1. By observing the vehicle's trajectory between lap 1 and lap 50, and between lap 51 and lap 100, as in figure 7.2, it can be observed that during the learning period - laps 1 to 50, figure 7.2a - the trajectory varies (although slightly) as the algorithm learns the controller design and the model correction, as the trajectory line on the left figure is thicker than the one on the right. On the other hand, as the algorithm reaches a learning steady state - laps 51 to 100, figure 7.2b - the trajectory line becomes fixed as can be observed by the thinner line on the right figure. In other words, the trajectory presented in 7.2b can be thought out as the optimal trajectory as found by the algorithm that, by being followed, achieves the fastest lap time possible for this track. In figure 7.1c the average longitudinal velocity between laps 1 to 50 and 51 to 100 is displayed. As expected from this figure, there is a significant increase in velocity throughout the track.



(a) laps 1 to 50.

(b) laps 51 to 100.



(c) Longitudinal velocity evolution.

Figure 7.2: FST10d simulation trajectory for Track 1 using the full algorithm.

Still focusing on the full algorithm test, the controller parameter learning behavior can be analyzed in figure 7.3. Once again, this figure shows a line corresponding to the average parameters across the five simulations with the corresponding extreme values area, denoting the minimums and maximums across the five simulations. Furthermore, the parameter evolution is shown as a function of each generation of the Genetic Algorithm. Recalling the parameter description in section 4.2 - MPC Based On Track Progress - this figure shows, as expected, an overall increase in the aggressiveness of the parameters as new generations are created. It can be further noted that the track was segmented into five segments and the that the population is also composed of five individuals. Therefore, a new generation is created at each new lap, hence, for this particular figure 7.3, the generations can be seen as laps as well.

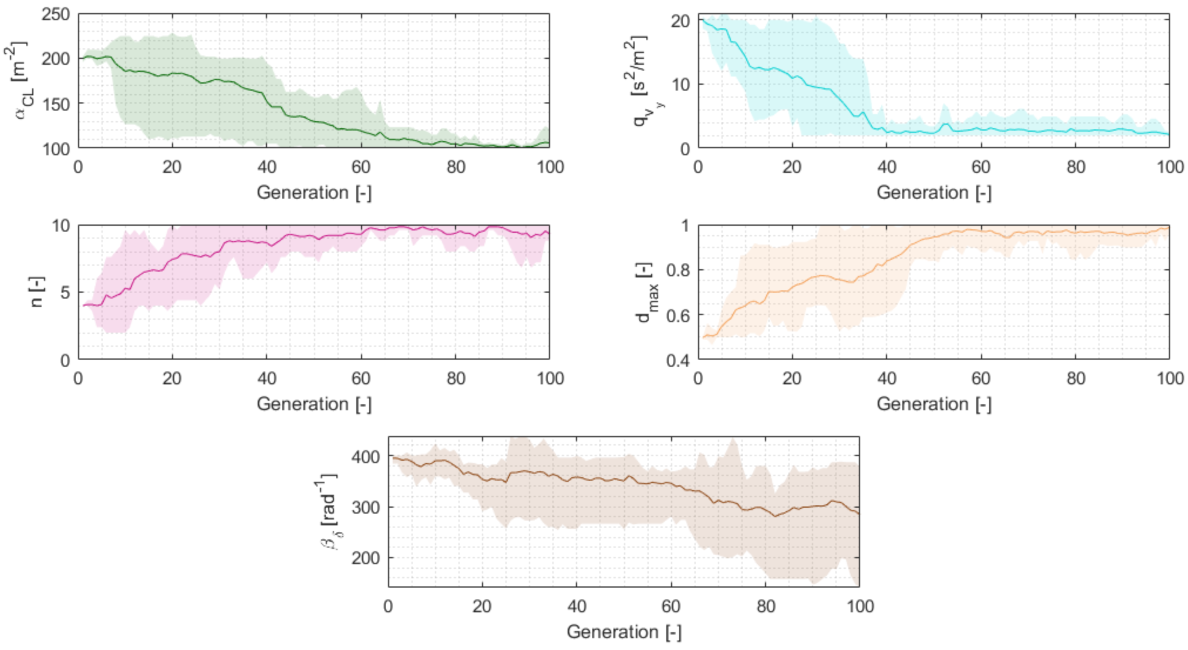


Figure 7.3: Controller parameter for the FST10d vehicle in Track 1.

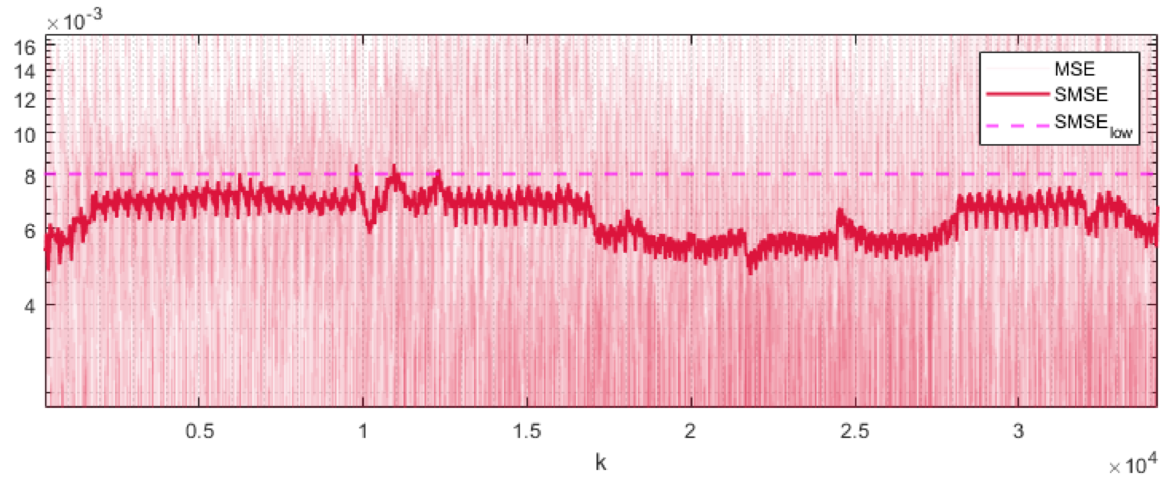
Regarding the model correction learning, figure 7.4 contains the values for the mean squared error (MSE) and for the smoothed mean squared error (SMSE) for each sampling instant k. Moreover, figure 7.4a contains data related to one of the simulations of the full algorithm test, corresponding to the red line of figure 7.1 whereas figure 7.4b contains data related to one of the simulations of the model correction node plus the MPC test, corresponding to the blue line of figure 7.1. Recall the values for the model correction learning algorithm in table B.2, namely, for $\mathrm{SMSE}_{\text{low}}$ and $\mathrm{SMSE}_{\text{high}}$. Recall also that if the SMSE is smaller than $\mathrm{SMSE}_{\text{low}}$ the model learning algorithm is inactive. As shown in 7.4b, the value of SMSE remained below this threshold throughout this simulation, therefore, no model correction learning occurred, which explains why the blue line of figure 7.1 remains constant. On the other hand, in figure 7.4a it can be seen that this threshold was crossed at some instants, as controller design learning starts to explore MPC parameters that are more aggressive than the starting ones.

Finally, more tests regarding the two other tracks, Track 2 and Track 3 were performed for FST10d. Figures C.1 and C.2 from appendix C contain the equivalent results for the full algorithm for Track 2 and Track 3, respectively.

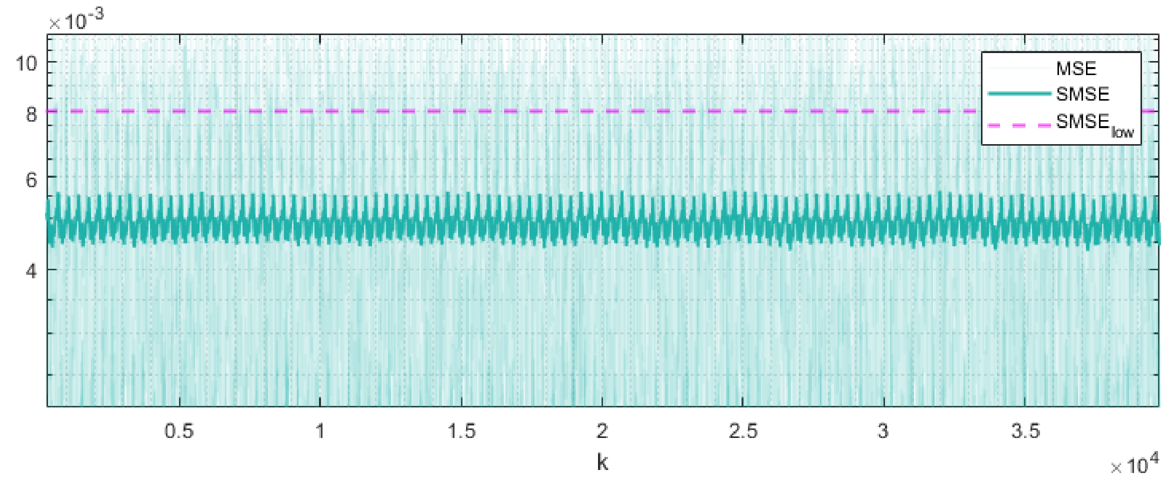
7.2 AMZ's Gotthard Simulations

In this section, results obtained in simulation by using AMZ's Gotthard vehicle parameters in the simulator are shown and discussed. In other words, the simulator is using vehicle parameters that differ from the FST10d's model that is used in the MPC. AMZ's Gotthard parameters are shown in table B.6.

Similarly to the last section, figure 7.5 contains the lap times obtained for, in this case, two different



(a) Model correction data for the full algorithm.



(b) Model correction data for the model correction algorithm alone.

Figure 7.4: MSE and SMSE for each sampling instant for FST10d.

tests for the MPC: a test using the model correction node (in blue) and a test using both the model correction and controller parameter learning nodes (in red).

The tests involving using the controller parameter learning node with the MPC alone were not performed since due to the high model mismatch the vehicle crashes somewhere along the 100 laps, being unable to finish the test. In fact, the learning correction node becomes much more relevant when the controller uses a different model from the simulator, as this node directly reduces model mismatch. In figure 7.5, this becomes clear as by using the model correction node alone the lap time is reduced from 17.89 s to 16.56 s. A rather noticeable difference when compared to the same test with the FST10d vehicle, as shown in figure 7.1, where using the model correction node alone did not result in a lap time improvement. Further using the controller parameter learning node, the lap times can be further reduced from 16.56 s to 14.93 s.

Focusing now on the full algorithm test, the test using the model correction and controller parameter learning nodes plus the MPC, corresponding to the red curve in figure 7.5. Once again, by observing the vehicle's trajectory between lap 1 and lap 50, and between lap 51 and lap 100, as in figure 7.6, the same conclusions from the last section can be withdrawn: during the learning period - laps 1 to 50 -

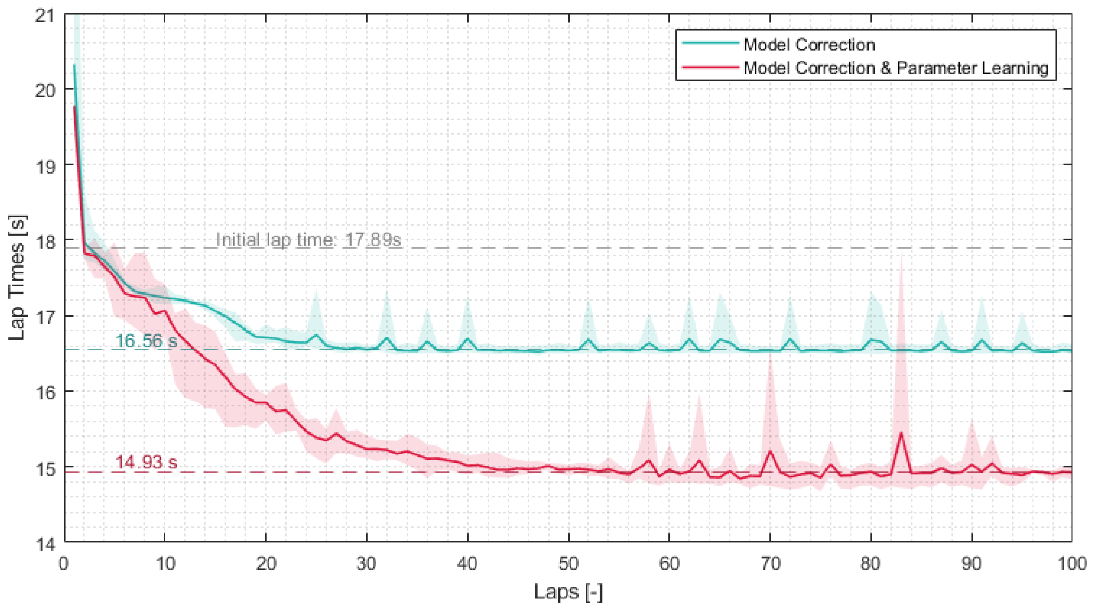


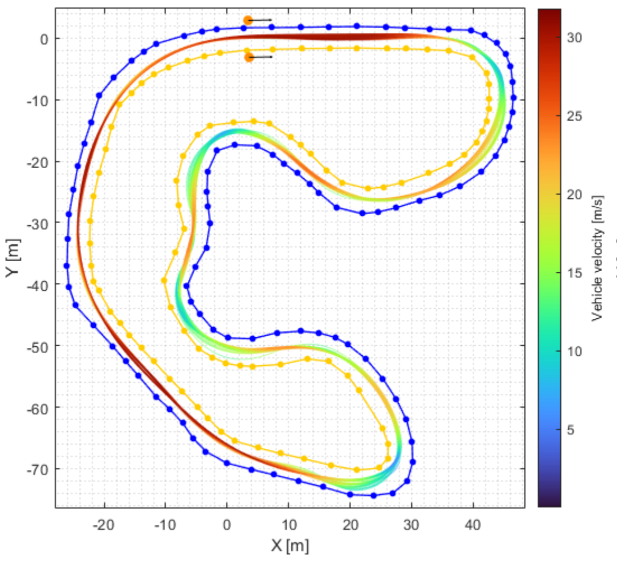
Figure 7.5: Lap time simulation results for AMZ's Gotthard in Track 1.

slightly different trajectories are explored, and the longitudinal velocities are in general lower, whereas during the steady period - laps 51 to 100 - the trajectories become more consistent and the longitudinal velocities higher.

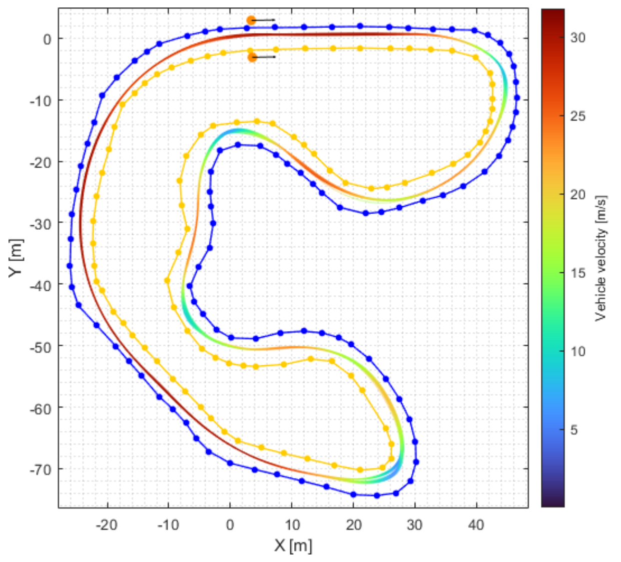
For the MPC's parameter evolution, figure 7.7 shows similar parameter evolution results to the ones for FST10d, as in figure 7.3.

Regarding model correction learning, figure 7.8 contains the values for the mean squared error (MSE) and for the smoothed mean squared error (SMSE) for each sampling instant k. Moreover, figure 7.8a contains data related to one of the simulations of the full algorithm test, corresponding to the red line of figure 7.5 whereas figure 7.8b contains data related to one of the simulations of the model correction node plus the MPC test, corresponding to the blue line of figure 7.5. For the case of AMZ's Gotthard, figure 7.8b expectedly reflects the perceived model mismatch from the learning period of the blue curve from figure 7.5, as in the first sampling instants, the SMSE is above the learning threshold, $\mathrm{SMSE}_{\text{low}}$, and is then reduced by the model correction algorithm. After the SMSE crosses the learning threshold, on average, it remains constant. On the other hand, in figure 7.4a it can be seen that this threshold was crossed more than one instant. This happens because, as controller design learning starts to explore MPC parameters that are more aggressive than the starting ones, i.e., as new driving scenarios are explored. As prior to exploring these learning scenarios no learning occurred in that feature space, the SMSE rises and is then attenuated by the model correction node.

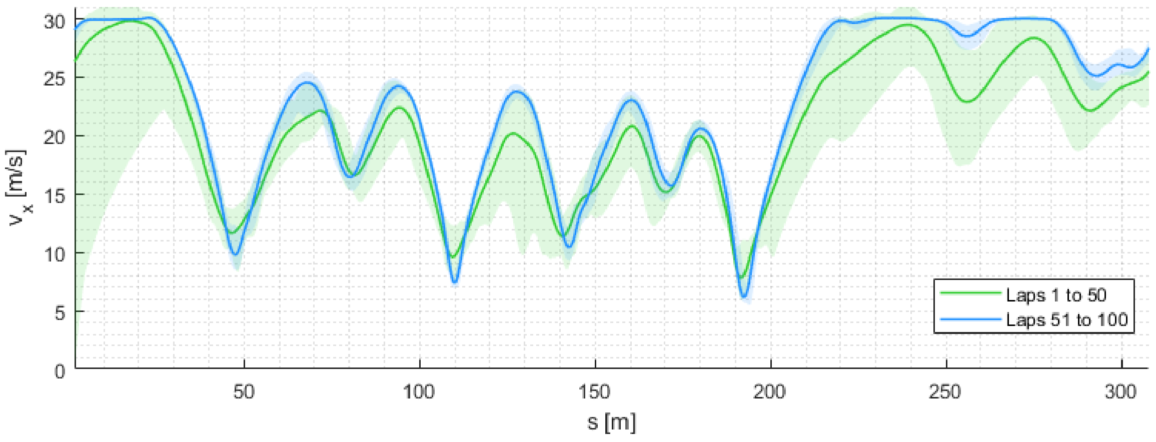
For reference, in AMZ's most recent research paper by Srinivasan et al. [43], it is claimed that the optimal lap time in Track 1 while subjected to the vehicle constraints mentioned in that paper is 18.0 s. Nevertheless, the tests presented in that paper were performed using AMZ's full pipeline. At the moment of writing this dissertation, FST Lisboa's autonomous systems' pipeline was not fully developed yet. However, depending on the rest of the pipeline, the path planning and control algorithm developed in this dissertation may potentially reach faster lap times than the ones presented in AMZ's paper.



(a) laps 1 to 50.



(b) laps 51 to 100.



(c) Longitudinal velocity evolution

Figure 7.6: AMZ's Gotthard simulation trajectory for Track 1 using the full algorithm.

Finally, the results obtained for Track 2 and 3 for AMZ's Gotthard can be seen in figures C.3 and C.4 in appendix C.

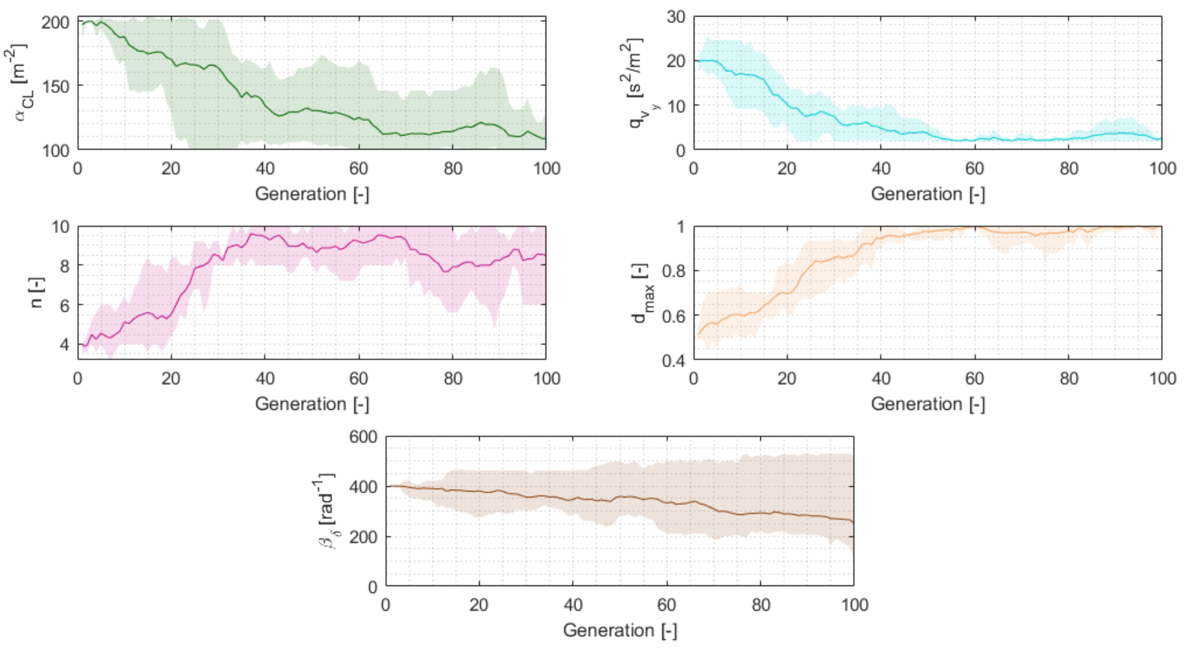
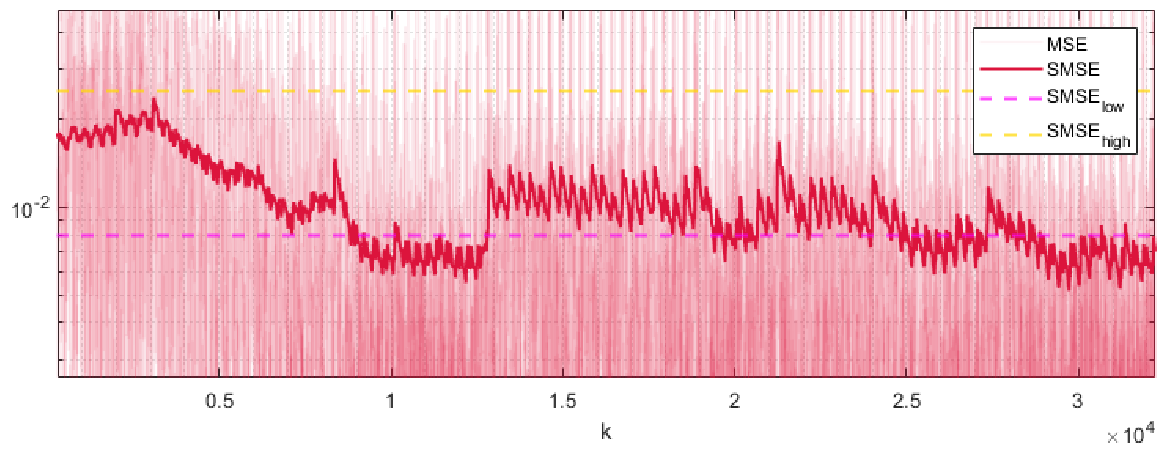
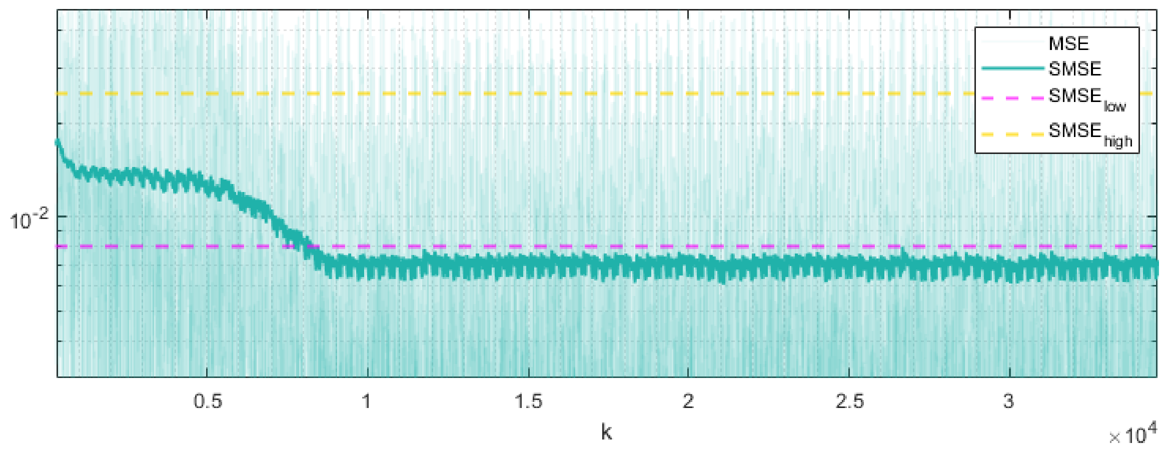


Figure 7.7: Controller parameter for AMZ's Gotthard vehicle in Track 1.



(a) Model correction data for the full algorithm.



(b) Model correction data for the model correction algorithm alone.

Figure 7.8: MSE and SMSE for each sampling instant for AMZ's Gotthard.

Chapter 8

Conclusions

8.1 Main Conclusions

A Learning Model Predictive Controller was designed with two learning principles in play: learning the dynamic model mismatch such that it is iteratively corrected as the vehicle drives itself and learning the controller design by developing an algorithm that automatically tunes the MPC parameters in a Reinforcement Learning background.

The results obtained in simulation have proven that, indeed, the developed methods effectively tend to maximize the vehicle's performance as the vehicle drives itself through the track. Namely, these methods have allowed, for Track 1, lap time reductions up to 16.5% when compared to the initial lap, achieving a lap time of 14.93 s for AMZ's Gotthard. As such, the objectives previously mentioned in section 1.3 - Objectives and Deliverables - were achieved. Moreover, these results were shown to be competitive towards other approaches developed by other FS teams, such as the team from ETH Zürich - AMZ Driverless.

The methods developed in this dissertation have some advantages and disadvantages. Namely, the developed MPC formulation, as mentioned before, requires previously collected data of the track itself. This might be a disadvantage if previously collected track data is not allowed. Nevertheless, FS competitions often allow teams to collect track data prior to the race. Track data offers the developed controller the advantage of being able to predict far ahead into the track for possible future obstacles and act accordingly. If, however, track data is not previously available, to overcome this issue, the race's first lap could be used to collect the required track data while the vehicle is controlled by a simpler controller. After this first lap, the developed LMPC could be activated since the required track data has been gathered. Another drawback of the developed approach is that it requires for the vehicle to drive roughly 40 laps until a minimum lap time is achieved. As such, to mitigate this issue, the parameters can be tuned in simulation, as the employed simulator was proven to obtain similar results to the real vehicle. Then, one could retrieve the final parameters from the simulations and then implement these parameters in the real vehicle as its initial parameters. The real vehicle should then drive some more laps in order to fine tune these parameters.

8.2 Future Work

Unfortunately, due to the work in progress in the vehicle's earlier pipeline stages, such as the SLAM algorithm for pose estimation, it was not possible to validate the developed control system in the real FST10d vehicle. For this reason, the obtained results should be replicated on the real vehicle to validate the results withdrawn from the simulations.

As expressed in [22], the research on model based controllers has been towards reducing model mismatch. However, instead of using the concept of adding a correction learning model to the nominal model, one could experiment with an online parameter estimation of the nominal model. Often, the vehicle parameters are estimated with some uncertainty, given this, by employing an online parameter estimation technique, the vehicle parameters could be adapted as the vehicle drives itself such that model mismatch is reduced. Moreover, more complex models than the blended bicycle models can be used for this purpose. This online system identification concept has been explored in research papers like [71, 72].

Contrary to the previous idea, one could employ a full machine learning model for learning the system's dynamics. For example, in [73], deep auto encoder networks have been used for learning the system's dynamics. Moreover, deep auto encoders aim at finding some coordinate transformation that transforms nonlinear system dynamics into linear dynamics. With such transformation, one could then apply classic well known linear control theory methods with the objective of controlling the nonlinear system through the coordinate transformation.

Furthermore, one could experiment ideas such as the ones in [74, 75]. In these research papers, evolutionary algorithms, such as Genetic Algorithms, are used to learn the vehicle's controller. Recall that FS vehicles drive in a track delimited by cones. If, for instance, the distance of these cones to the vehicles is measured one could estimate the position of the vehicle relative to the track's left and right limits. Then, using these measurements as an input to a learner function, such as an Artificial Neural Network, one could apply Reinforcement Learning techniques to estimate the learner function parameters that are able to drive the car. Nevertheless, this may be rather challenging to implement, as, in general, a lot of training is required to obtain a good controller. However, if succeeded, this would result in a fast computing nonlinear controller for the vehicle.

Finally, a mentioned downside to the implementation of the control algorithm in this dissertation is that it requires previously collected track data, such as the locations of the cones. To overcome this, one could modify this controller, or use similar techniques, to control the vehicle by using what the car ”sees”. In other words, as the car detects new cones ahead, the track path is estimated until the last detectable cone, then, a MPC could be designed to control the vehicle using this track path estimation.

Bibliography

[1] M. Hirz and B. Walzel. Sensor and object recognition technologies for self-driving cars. *Computer-Aided Design and Applications*, 15:1–8, 01 2018. doi: 10.1080/16864360.2017.1419638.

[2] M. Schwall, T. Daniel, T. Victor, F. Favaro, and H. Hohnhold. Waymo public road safety performance data. *arXiv preprint arXiv:2011.00038*, 2020.

[3] Tesla Vehicle Safety Report, Jan. 2019. URL `https://www.tesla.com/VehicleSafetyReport`.

[4] M. Dikmen and C. M. Burns. Autonomous driving in the real world: Experiences with tesla autopilot and summon. In *Proceedings of the 8th international conference on automotive user interfaces and interactive vehicular applications*, pages 225–228, 2016.

[5] T. B. Lee. Tesla has a self-driving strategy other companies abandoned years ago, Mar. 2019. URL `https://arstechnica.com/cars/2019/03/teslas-self-driving-strategy-is-outdated-and-possibly-dangerous/`.

[6] C. Amsel and D. Stapel. *Compact radar sensors enabling autonomous driving*, pages 843–849. 04 2017. ISBN 978-3-658-16987-9. doi: 10.1007/978-3-658-16988-6_67.

[7] A. Uçar, Y. Demir, and C. Güzeliş. Object recognition and detection with deep learning for autonomous driving applications. *Simulation*, 93(9):759–769, 2017.

[8] 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.

[9] G. Ros, S. Ramos, M. Granados, A. Bakhtiary, D. Vazquez, and A. M. Lopez. Vision-based offline-online perception paradigm for autonomous driving. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 231–238, 2015. doi: 10.1109/WACV.2015.38.

[10] M. Al-Qizwini, I. Barjasteh, H. Al-Qassab, and H. Radha. Deep learning algorithm for autonomous driving using googlenet. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 89–96, 2017. doi: 10.1109/IVS.2017.7995703.

[11] S. Thrun. Probabilistic robotics. *Communications of the ACM*, 45(3):52–57, 2002.

[12] M. Montemerlo and S. Thrun. *FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics*, volume 27. Springer, 2007.

[13] T. Suzuki, Y. Amano, and T. Hashizume. Development of a sift based monocular ekf-slam algorithm for a small unmanned aerial vehicle. In *SICE Annual Conference 2011*, pages 1656–1659. IEEE, 2011.

[14] H.-j. Wang, J. Wang, L. Yu, and Z.-y. Liu. A new slam method based on svm-aekf for auv. In *OCEANS'11 MTS/IEEE KONA*, pages 1–6. IEEE, 2011.

[15] T. Chong, X. Tang, C. Leng, M. Yogeswaran, O. Ng, and Y. Chong. Sensor technologies and simultaneous localization and mapping (slam). *Procedia Computer Science*, 76:174–179, 2015.

[16] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–15, 2020. doi: 10.1109/TITS.2020.3012034.

[17] J. Li, W. Zhan, Y. Hu, and M. Tomizuka. Generic tracking and probabilistic prediction framework and its application in autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, 21 (9):3634–3649, 2020. doi: 10.1109/TITS.2019.2930310.

[18] N. Muhammad and B. Åstrand. Predicting agent behaviour and state for applications in a roundabout-scenario autonomous driving. *Sensors*, 19(19), 2019. ISSN 1424-8220. doi: 10.3390/s19194279. URL `https://www.mdpi.com/1424-8220/19/19/4279`.

[19] K. Chu, M. Lee, and M. Sunwoo. Local path planning for off-road autonomous driving with avoidance of static obstacles. *IEEE Transactions on Intelligent Transportation Systems*, 13(4):1599–1616, 2012. doi: 10.1109/TITS.2012.2198214.

[20] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. *Ann Arbor*, 1001(48105):18–80, 2008.

[21] W. Farag. Complex trajectory tracking using pid control for autonomous driving. *International Journal of Intelligent Transportation Systems Research*, 18(2):356–366, 2020.

[22] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, et al. Towards fully autonomous driving: Systems and algorithms. In *2011 IEEE intelligent vehicles symposium (IV)*, pages 163–168. IEEE, 2011.

[23] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli. Kinematic and dynamic vehicle models for autonomous driving control design. In *2015 IEEE Intelligent Vehicles Symposium (IV)*, pages 1094–1099, 2015. doi: 10.1109/IVS.2015.7225830.

[24] S. Lefèvre, A. Carvalho, and F. Borrelli. A learning-based framework for velocity control in autonomous driving. *IEEE Transactions on Automation Science and Engineering*, 13(1):32–42, 2016. doi: 10.1109/TASE.2015.2498192.

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

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

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

[28] J. Pinho. Learning to drive autonomously a race car. Master's thesis, Instituto Superior Técnico, 2021.

[29] A. Passo. Path following and control for autonomous driving of a formula student car. Master's thesis, Instituto Superior Técnico, 2021.

[30] R. C. Coulter. Implementation of the pure pursuit path tracking algorithm. Technical report, Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

[31] 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, 2015.

[32] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger. Learning-based model predictive control: Toward safe learning in control. *Annual Review of Control, Robotics, and Autonomous Systems*, 3(1):269–296, 2020. doi: 10.1146/annurev-control-090419-075625.

[33] A. Nagabandi, I. Clavera, S. Liu, R. S. Fearing, P. Abbeel, S. Levine, and C. Finn. Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *arXiv preprint arXiv:1803.11347*, 2018.

[34] S. Piche, B. Sayyar-Rodsari, D. Johnson, and M. Gerules. Nonlinear model predictive control using neural networks. *IEEE Control Systems Magazine*, 20(3):53–62, 2000.

[35] J. Kabzan, L. Hewing, A. Liniger, and M. N. Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, 2019. doi: 10.1109/LRA.2019.2926677.

[36] L. Hewing, J. Kabzan, and M. N. Zeilinger. Cautious model predictive control using gaussian process regression. *IEEE Transactions on Control Systems Technology*, 28(6):2736–2743, 2020. doi: 10.1109/TCST.2019.2949757.

[37] H. Boubertakh, M. Tadjine, P.-Y. Glorennec, and S. Labiod. Tuning fuzzy pd and pi controllers using reinforcement learning. *ISA Transactions*, 49(4):543–551, 2010. ISSN 0019-0578. doi: https://doi.org/10.1016/j.isatra.2010.05.005. URL `https://www.sciencedirect.com/science/article/pii/S0019057810000601`.

[38] S. Gros and M. Zanon. Data-driven economic nmpc using reinforcement learning. *IEEE Transactions on Automatic Control*, 65(2):636–648, 2019.

[39] F. Leung, H. Lam, S. Ling, and P. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE Transactions on Neural Networks*, 14(1):79–88, 2003. doi: 10.1109/TNN.2002.804317.

[40] J. Zhang, J. Zhuang, H. Du, and S. Wang. Self-organizing genetic algorithm based tuning of pid controllers. *Information Sciences*, 179(7):1007–1018, 2009. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2008.11.038. URL `https://www.sciencedirect.com/science/article/pii/S0020025508004970`.

[41] A. H. Wright. Genetic algorithms for real parameter optimization. volume 1 of *Foundations of Genetic Algorithms*, pages 205–218. Elsevier, 1991. doi: https://doi.org/10.1016/B978-0-08-050684-5.50016-1. URL `https://www.sciencedirect.com/science/article/pii/B9780080506845500161`.

[42] J. Kabzan, M. I. Valls, V. J. Reijgwart, H. F. Hendrikx, C. Ehmke, M. Prajapat, A. Bühler, N. Gosala, M. Gupta, R. Sivanesan, and et al. Amz driverless: The full autonomous racing system. *Journal of Field Robotics*, 2020. doi: 10.1002/rob.21977.

[43] S. Srinivasan, S. N. Giles, and A. Liniger. A holistic motion planning and control solution to challenge a professional racecar driver. *IEEE Robotics and Automation Letters*, 6(4):7854–7860, 2021.

[44] O. Dünkel, L. Ohnemus, and J. Kaths. Virtual test drives for the development of the kit driverless 18d. *ATZextra worldwide*, 23(1):62–65, 2018.

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

[46] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks. a data-driven control framework. *IEEE Transactions on Automatic Control*, 63(7):1883–1896, 2017.

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

[48] L. Ljung. *Control theory: multivariable and nonlinear methods*. Taylor & Francis, 2000.

[49] M. Mohri, A. Rostamizadeh, and A. Talwalkar. *Foundations of machine learning*. MIT press, 2018.

[50] L. Hana. Parametric and nonparametric machine learning algorithms, Aug 2020. URL `https://lamiae-hana.medium.com/parametric-and-nonparametric-machine-learning-algorithms-ec9a21f25705`.

[51] J. M. Keller, D. Liu, and D. B. Fogel. *Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation*. John Wiley & Sons, 2016.

[52] RBF and EBF Models. URL `https://abaqus-docs.mit.edu/2017/English/IhrUserMap/ihr-c-Reference-RBFandEBF.htm#GeneralizingRBFtoEBF`.

[53] Man-Wai Mak and Sun-Yuan Kung. Estimation of elliptical basis function parameters by the EM algorithm with application to speaker verification. *IEEE Transactions on Neural Networks*, 11(4): 961–969, July 2000. ISSN 10459227. doi: 10.1109/72.857775. URL `http://ieeexplore.ieee.org/document/857775/`.

[54] R. Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[55] R. A. van de Geijn. Notes on cholesky factorization. *The University of Texas at Austin Austin*, 2011.

[56] C. Szepesvári. Algorithms for reinforcement learning. *Synthesis lectures on artificial intelligence and machine learning*, 4(1):1–103, 2010.

[57] J. Baxter and P. L. Bartlett. Direct gradient-based reinforcement learning. In *2000 IEEE International Symposium on Circuits and Systems (ISCAS)*, volume 3, pages 271–274. IEEE, 2000.

[58] S. Khadka and K. Tumer. Evolutionary reinforcement learning. *arXiv preprint arXiv:1805.07917*, 2018.

[59] D. E. Goldberg. Genetic algorithms in search, optimization, and machine learning. addison. *Reading*, 1989.

[60] S. R, R. Sundaresan, N. ebénézer, and R. Natarajan. Evolutionary bi-criteria optimum design of robots based on task specifications. *International Journal of Advanced Manufacturing Technology*, 41:386–406, 03 2009. doi: 10.1007/s00170-008-1483-8.

[61] R. Rajamani. *Vehicle dynamics and control*. Springer, 2012.

[62] L. Grüne and J. Pannek. *Discrete Time and Sampled Data Systems*, pages 13–42. 04 2011. ISBN 978-0-85729-500-2. doi: 10.1007/978-0-85729-501-9_2.

[63] G. Sánchez, M. Murillo, L. Genzelis, N. Deniz, and L. Giovanini. Mpc for nonlinear systems: A comparative review of discretization methods. In *2017 XVII Workshop on Information Processing and Control (RPIC)*, pages 1–6, 2017. doi: 10.23919/RPIC.2017.8214333.

[64] R. Shaikh. Feature Selection Techniques in Machine Learning with Python, Oct. 2018. URL `https://towardsdatascience.com/feature-selection-techniques-in-machine-learning-with-python-f24e7da3f36e`.

[65] T. H. Park and P. Cook. Radial/elliptical basis function neural networks for timbre classification. *Proceedings of the Journées d'Informatique Musicale, Paris*, 2005.

[66] F. Fuchs, Y. Song, E. Kaufmann, D. Scaramuzza, and P. Duerr. Super-human performance in gran turismo sport using deep reinforcement learning. *arXiv preprint arXiv:2008.07971*, 2020.

[67] J. H. (https://math.stackexchange.com/users/114036/john hughes). Why arc-length parametrized curves has unit tangent vector? Mathematics Stack Exchange. URL `https://math.stackexchange.com/q/2122376`. URL:https://math.stackexchange.com/q/2122376 (version: 2017-01-31).

[68] Curvature and Radius of Curvature. URL https://math24.net/curvature-radius.html.

[69] A. Domahidi and J. Jerez. Forces professional. Embotech AG, 2014–2019. URL https://embotech.com/FORCES-Pro.

[70] 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*, pages 1–17, 2017.

[71] M. Boufadene, A. Rabhi, M. Belkheiri, and A. Elhajjaji. Vehicle online parameter estimation using a nonlinear adaptive observer. In *2016 American Control Conference (ACC)*, pages 1006–1010, 2016. doi: 10.1109/ACC.2016.7525046.

[72] G. Reina, M. Paiano, and J. L. Blanco. Vehicle parameter estimation using a model-based estimator. *Mechanical Systems and Signal Processing*, 87, 07 2016. doi: 10.1016/j.ymssp.2016.06.038.

[73] G. Beintema, R. Toth, and M. Schoukens. Nonlinear state-space identification using deep encoder networks. In *Learning for Dynamics and Control*, pages 241–250. PMLR, 2021.

[74] H. Marques, J. Togelius, M. Kogutowska, O. Holland, and S. Lucas. Sensorless but not senseless: Prediction in evolutionary car racing. pages 370 – 377, 05 2007. ISBN 1-4244-0701-X. doi: 10.1109/ALIFE.2007.367819.

[75] Y. Sáez, D. Perez Liebana, O. Sanjuán, and P. Isasi. Driving cars by means of genetic algorithms. pages 1101–1110, 09 2008. doi: 10.1007/978-3-540-87700-4_109.

Appendix A

Algorithms

In this appendix, various algorithms for further understanding of the methods developed in this dissertation are presented.

Algorithm 4 Genetic Algorithm new generation step

```
1: procedure GA($\mathbf{D}_k$, $N$, $n_d$, $\mathrm{P}(\mathrm{R})$, $\mathrm{P}(\mathrm{C})$, $\mathrm{P}(\mathrm{M})$, $r(\cdot)$)    ▷ Input data for the method
2:     $\mathbf{r}, \mathbf{f} \in \mathbb{R}^N$    ▷ Allocate memory space for the reward and relative fitness vector
3:     for integer $j = 1$ to $j = N$ do    ▷ Compute the reward for each individual
4:         $r_j \leftarrow r(\mathbf{d}_j)$    ▷ Recall that individuals genetic code can be retrieved from the columns of $\mathbf{D}_k$
5:     end for
6:     for integer $j = 1$ to $j = N$ do    ▷ Compute the fitness for each individual
7:         $f_j \leftarrow \frac{r_j}{\sum_{n=1}^{N} r_n}$    ▷ Fitness is mapped according to equation (2.26)
8:     end for
9:     $n_\mathrm{R}, n_\mathrm{C}, n_\mathrm{M} \leftarrow \mathrm{distribution}(\mathrm{P}(\mathrm{R}), \mathrm{P}(\mathrm{C}), \mathrm{P}(\mathrm{M}), N)$    ▷ Select the $N$ operations to the probabilities of each operation
10:    if $n_\mathrm{C}$ is odd then    ▷ Check if the number of crossovers is odd
11:        $n_\mathrm{C} \leftarrow n_\mathrm{C} - 1$    ▷ Subtracts one from $n_\mathrm{C}$ so that $n_\mathrm{C}$ is even
12:        chose one: $n_\mathrm{R} \leftarrow n_\mathrm{R} + 1$ or $n_\mathrm{M} \leftarrow n_\mathrm{M} + 1$    ▷ Add one to $n_\mathrm{R}$ or $n_\mathrm{M}$ such that $n_\mathrm{R} + n_\mathrm{C} + n_\mathrm{M} = N$
13:    end if
14:    $\mathbf{D_{k+1}} \in \mathbb{R}^{n_d \times N}$    ▷ Allocate memory space for the new generation
15:    $t \leftarrow 0$    ▷ Initialize a counter variable
16:    for integer $n = 1$ to $n = n_\mathrm{R}$ do    ▷ Perform reproductions
17:        $t \leftarrow t + 1$
18:        column $t$ of $\mathbf{D}_{k+1} \leftarrow \mathrm{select\_one}(\mathbf{D}_k, \mathbf{f})$
19:    end for
20:    for integer $n = 1$ to $n = \frac{n_\mathrm{C}}{2}$ do    ▷ Perform crossovers
21:        $\mathbf{p}_1, \mathbf{p}_2 \leftarrow \mathrm{select\_two}(\mathbf{D}_k, \mathbf{f})$    ▷ Select two different parents
22:        $\mathbf{c}_1, \mathbf{c}_2 \leftarrow \mathrm{crossover}(\mathbf{p}_1, \mathbf{p}_2)$    ▷ Get two children from a crossover of the parents
23:        $t \leftarrow t + 1$
24:        column $t$ of $\mathbf{D}_{k+1} \leftarrow \mathbf{c}_1$
25:        $t \leftarrow t + 1$
26:        column $t$ of $\mathbf{D}_{k+1} \leftarrow \mathbf{c}_2$
27:    end for
28:    for integer $n = 1$ to $n = n_\mathrm{M}$ do    ▷ Perform mutations
29:        $\mathbf{d} \leftarrow \mathrm{select\_one}(\mathbf{D}_k, \mathbf{f})$
30:        $\mathbf{d} \leftarrow \mathrm{mutation}(\mathbf{d})$    ▷ perform mutation on selected individual
31:        $t \leftarrow t + 1$
32:        column $t$ of $\mathbf{D}_{k+1} \leftarrow \mathbf{d}$
33:    end for
34:    return $\mathbf{D}_{k+1}$    ▷ Return the next generation
35: end procedure
```

Algorithm 5 Online Levenberg-Marquardt parameter update algorithm for the vehicle model

```
 1: procedure OLM($\epsilon^{(x,y)}_{t=k-n_{\text{batch}}+1,\ldots,k-1}$, $\mathbf{v}_{t=k-n_{\text{batch}}+1,\ldots,k}$, $\mathbf{p}_k$, $\Psi_{k-1}$, $\mathbf{x}^{\text{real}}_k$, $\mathbf{x}^{\text{pred}}_k$, $\gamma$, $\lambda_{\text{MSE}}$,
        $n_{\text{batch}}$, $\text{SMSE}_{k-1}$, $n_s$, $n_p$, $\text{SMSE}_{\text{low}}$, $\Gamma$)    ▷ Input data for the method
 2:     $\mathbf{A} \leftarrow \mathbf{A}(\Psi_{k-1})$    ▷ Compute the transformation matrix as per equation (3.35)
 3:     $\epsilon^{(x,y)}_k \leftarrow \mathbf{A}^T \left(\mathbf{x}^{\text{real}}_k - \mathbf{x}^{\text{pred}}_k\right)$    ▷ Compute the current discrepancy as per equation (3.36b)
 4:     $\text{MSE}_k \leftarrow 0$    ▷ Initialize current mean squared error as zero
 5:     for integer $j = 1$ to $j = n_s$ do    ▷ Compute mean squared error as per equation (3.37)
 6:         $\text{MSE}_k \leftarrow \text{MSE}_k + \frac{(\epsilon_{k,j})^2}{n_s}$
 7:     end for
 8:     $\text{SMSE}_k = \lambda_{\text{MSE}} \cdot \text{MSE}_k + (1 - \lambda_{\text{MSE}}) \cdot \text{SMSE}_{k-1}$    ▷ Compute smoothed mean squared error as
    per equation (3.47)
 9:     if $\text{SMSE}_k \geq \text{SMSE}_{\text{low}}$ then    ▷ Update is only required if above a threshold, recall figure 3.7
10:         $\lambda \leftarrow \lambda(\text{SMSE}_k)$    ▷ Compute lambda according to the schedule defined by equation (3.48)
11:         for integer $t = k - n_{\text{batch}} + 1$ to $t = k$ do    ▷ Iterate over the learning batch
12:             initialize $J^{\epsilon_t}_{\mathbf{p}} \in \mathbb{R}^{n_p \times n_s}$    ▷ Allocate memory space for the discrepancy jacobian matrix
13:             for integer $i = 1$ to $i = n_p$ do    ▷ Iterate over the number of network parameters
14:                 for integer $j = 1$ to $i = n_s$ do    ▷ Iterate over the number of network outputs
15:                     $\left[J^{\epsilon_t}_{\mathbf{p}}\right]_{i,j} \leftarrow -\frac{\partial e_j(\mathbf{p}_k, \mathbf{v}_t)}{\partial p_i}$    ▷ Assign element $i$,$j$ to the discrepancy jacobian
16:                 end for
17:             end for
18:         end for
19:         $J_{\mathbf{p}} \leftarrow \begin{bmatrix} J^{\epsilon_k}_{\mathbf{p}} & J^{\epsilon_{k-1}}_{\mathbf{p}} & \cdots & J^{\epsilon_{k-n_{\text{batch}}+1}}_{\mathbf{p}} \end{bmatrix}$    ▷ Obtain cost function jacobian as per equation (3.45a)
20:         $\epsilon^{\text{batch}} = \frac{2}{n_s \cdot \Gamma} \begin{bmatrix} \epsilon_k \\ \gamma \cdot \epsilon_{k-1} \\ \vdots \\ \gamma^{n_{\text{batch}}-1} \cdot \epsilon_{k-n_{\text{batch}}+1} \end{bmatrix}$    ▷ Obtain batch discrepancy as per equation (3.45b)
21:         $\Delta\mathbf{p} \leftarrow$ Cholesky solve $\Delta\mathbf{p} : \left(\lambda I + J_{\mathbf{p}} J^T_{\mathbf{p}}\right) \Delta\mathbf{p} = -J_{\mathbf{p}} \epsilon^{\text{batch}}$    ▷ Obtain $\Delta\mathbf{p}$ as per equation
    (3.46)
22:         $\mathbf{p}_{k+1} \leftarrow \mathbf{p}_k + \Delta\mathbf{p}$
23:         if The selected neural network is EBFN then
24:             for parameter $p_{k+1}$ in $\omega \subset \mathbf{p}_{k+1}$ do    ▷ Iterate over each parameter regarding each
    cluster's precision matrix, recall section 2.2.2
25:                 if $p_{k+1} \leq 0$ then
26:                     $p_{k+1} \leftarrow \delta$  ▷ Ensure that each eigen value remains positive being $\delta$ a small positive
    value
27:                 end if
28:             end for
29:         end if
30:     else
31:         $\mathbf{p}_{k+1} \leftarrow \mathbf{p}_k$    ▷ Parameter vector remains the same
32:     end if
33:     return $\mathbf{p}_{k+1}$, $\epsilon^{(x,y)}_k$, $\text{SMSE}_k$    ▷ Output new parameters, current discrepancy vector and current
    smoothed mean squared error
34: end procedure
```

Algorithm 6 Controller loop scheme

```
1: $\mathbf{u}^* \leftarrow 0$    ▷ Apply null throttle and steering at the beginning
2: **integer** $\text{fail\_counter} \leftarrow 0$
3: **while** Event not finished **do**
4:     $t_{\text{initial}} \leftarrow \text{current\_time}$    ▷ Get loop starting time
5:     **apply** $\mathbf{u}^*$    ▷ Apply control actions computed at the previous time instant
6:     $\mathbf{x} \leftarrow \mathbf{x}_t$    ▷ Update vehicle states
7:     $\mathbf{p}^{\text{model}} \leftarrow \mathbf{p}_t^{\text{model}}$    ▷ Update vehicle model correction parameters (if needed)
8:     $\mathbf{p}^{\text{MPC}} \leftarrow \mathbf{p}_t^{\text{MPC}}$    ▷ Update MPC parameters (if needed)
9:     $\text{flag}, \mathbf{u}_k, s_k, \forall k \in [t, ..., t+N-1] \leftarrow \text{FORCESPRO\_Solve}\left(s_{t-1}, \mathbf{x}, \mathbf{u}^*, \mathbf{p}^{\text{model}}, \mathbf{p}^{\text{MPC}}, \text{timeout}\right)$    ▷ Solve the MPC optimization problem with FORCESPRO's solver
10:    **if** $\text{flag} = 3$ **then**    ▷ Check if the obtained solution is unfeasible
11:        $\text{fail\_counter} \leftarrow \text{fail\_counter} + 1$
12:        $\mathbf{u}^* \leftarrow \mathbf{u}^{\text{safe}}_{\text{fail\_counter}+1}$
13:    **else**    ▷ Else, if solution is feasible
14:        $\text{fail\_counter} \leftarrow 0$
15:        $\mathbf{u}^* \leftarrow \mathbf{u}_{t+1}$
16:        $\mathbf{u}^{\text{safe}} \leftarrow \mathbf{u}_k, \forall k \in [t, ..., t+N]$    ▷ Store the control action sequence
17:    **end if**
18:    $t_{\text{final}} \leftarrow \text{current\_time}$    ▷ Get loop finishing time
19:    **wait** $T_s - (t_{\text{final}} - t_{\text{initial}})$    ▷ Wait for the next sampling instant
20: **end while**
```

Algorithm 7 Reward and average segment time computation

```
1: **procedure** GA($T_{i,k}$, $n_i^{\text{cones}}$, $T_{i,k}^{\text{avg}}$, $T_{i,k-1}^{\text{avg}}$, $i$, $j$, $\mathbf{r}$, $N_{\text{CP}}$, $N_{\text{pop}}$)    ▷ Input data for the method
2:     $r_j \leftarrow r\left(T_{i,k}, n_i^{\text{cones}}\right)$    ▷ compute reward for individual $j$ as per equation (5.5)
3:     $T_{i,k+1}^{\text{avg}} \leftarrow \lambda_T T_{i,k} + (1-\lambda_T)\, T_{i,k-1}^{\text{avg}}$    ▷ update average segment time, as per equation (5.6)
4:     $j \leftarrow j + 1$    ▷ Select next individual of the population
5:     **if** $j > N_{\text{pop}}$ **then**
6:         $\mathbf{D}_{k+1} \leftarrow \text{new\_generation}\left(\mathbf{D}_k, \mathbf{r}\right)$    ▷ If the previous individual was the last individual on the population, create new generation
7:         $j \leftarrow 1$    ▷ Reset population counter
8:     **else**
9:         $\mathbf{D}_{k+1} \leftarrow \mathbf{D}_k$    ▷ Else, move on to the next individual in the current generation
10:    **end if**
11:    $\mathbf{d}_{k+1} \leftarrow$ **column** $j$ **of** $\mathbf{D}_{k+1}$    ▷ Select next parameters to apply on the next segment
12:    $i \leftarrow i + 1$    ▷ Increment segment counter
13:    **if** $i > N_{\text{CP}} - 1$ **then**
14:        $i \leftarrow 1$    ▷ If the previous segment was the last segment, restart the segment counter
15:    **end if**
16:    **return** $\mathbf{d}_{k+1}$, $\mathbf{D}_{k+1}$, $T_{i,k+1}^{\text{avg}}$, $i$, $j$, $\mathbf{r}$    ▷ Return parameters to be applied in the next segment and other variables
17: **end procedure**
```

Appendix B

Tables of parameters

In this appendix, various tables containing various parameters used for the development and implementation of the methods in this dissertation are presented.

Table B.1: FST10d's model parameters for the kinematic, dynamic, and blended models.

Parameter	Value	Unit
m	250	kg
I_z	80	kg m^2
l_f	0.832	m
l_r	0.708	m
C_d	1.2	-
A_F	1.18	m^2
GR	15.74	-
r_{wheel}	0.23	m
$T_{\max}$	21	Nm
η_{motor}	0.9	-
C_r	0.092	-
$B_{f/r}$	10	-
$C_{f/r}$	138	-
$D_{f/r}$	1500	N
ρ	1.18	kg m^{-3}
g	9.81	m/s^2
$V_{\text{blend}_{\min}}$	2	m/s
$V_{\text{blend}_{\max}}$	5	m/s
T_s	50	ms
h	10	ms

Table B.2: Correction model parameters.

Parameter	Value
n_{batch}	300
γ	0.98
$\mathrm{SMSE}_{\text{high}}$	0.025
$\mathrm{SMSE}_{\text{low}}$	0.008
λ_{high}	5
λ_{low}	50
λ_{MSE}	0.005
n_v	4
n_h	12
n_e	6

Table B.3: Fixed MPC parameters.

Parameter	Value	Unit
λ	200	m^{-1}
α_L	1000	m^{-2}
$q_{v_{\max}}$	5	s/m
$v_{\max}$	30	m/s
$\Delta d_{\max}$	0.2	-
$\Delta \delta_{\max}$	0.075	rad
$\delta_{\max}$	0.47	rad
$\Delta s_{\min}$	0.1	m
$\Delta s_{\max}$	1.5	m
$\hat{e}_{CL_{\max}}$	1	m
T_s	50	ms
N	40	-

Table B.4: Genetic algorithm parameters.

Parameter	Value	Unit
$\mathrm{P}\,(\mathrm{R})$	0.1	-
$\mathrm{P}\,(\mathrm{C})$	0.5	-
$\mathrm{P}\,(\mathrm{M})$	0.4	-
λ_T	0.3	-
N_{pop}	5	-
N_{CP}	5	-
$m_{\alpha_{CL}}^{\text{high}}$	20	m^{-n}
$m_{\alpha_{CL}}^{\text{low}}$	-40	m^{-n}
$m_{d_{\max}}^{\text{high}}$	0.2	-
$m_{d_{\max}}^{\text{low}}$	-0.1	-
$m_{q_{v_y}}^{\text{high}}$	5	s^2/m^2
$m_{q_{v_y}}^{\text{low}}$	-10	s^2/m^2
$m_{\beta_\delta}^{\text{high}}$	-80	rad^{-2}
$m_{\beta_\delta}^{\text{low}}$	40	rad^{-2}
k_T	4	s^{-1}
k_c	0.5	s/cone
$d_{\alpha_{CL}}^{\max}$	500	m^{-n}
$d_{\alpha_{CL}}^{\min}$	100	m^{-n}
$d_{d_{\max}}^{\max}$	1	-
$d_{d_{\max}}^{\min}$	0.1	-
$d_{q_{v_y}}^{\max}$	40	s^2/m^2
$d_{q_{v_y}}^{\min}$	2	s^2/m^2
$d_{n}^{\max}$	10	-
$d_{n}^{\min}$	2	-
$d_{\beta_\delta}^{\max}$	800	rad^{-2}
$d_{\beta_\delta}^{\min}$	100	rad^{-2}

Table B.5: Initial generation parameters.

Parameter	Value	Unit
α_{CL}	200	m^{-n}
$d_{\max}$	0.5	-
q_{v_y}	20	s^2/m^2
n	4	-
β_δ	400	rad^{-2}

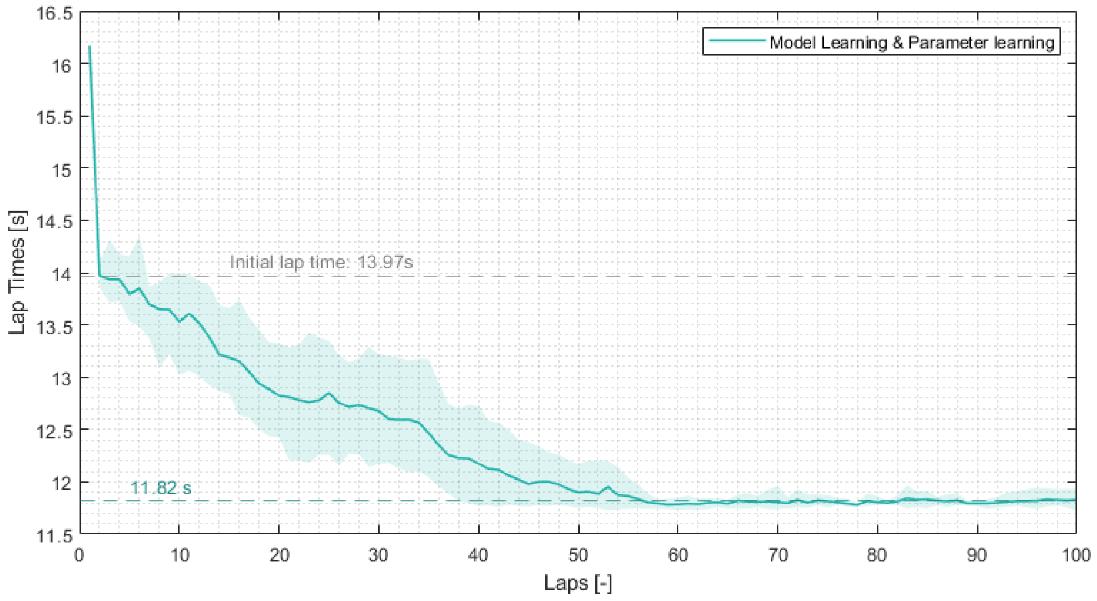
Table B.6: Some of AMZ's Gotthard model parameters for the simulator.

Parameter	Value	Unit
m	190	kg
I_z	110	kg m^2
l_f	0.765	m
l_r	0.765	m
C_d	1.0	-
A_f	1.4	m^2

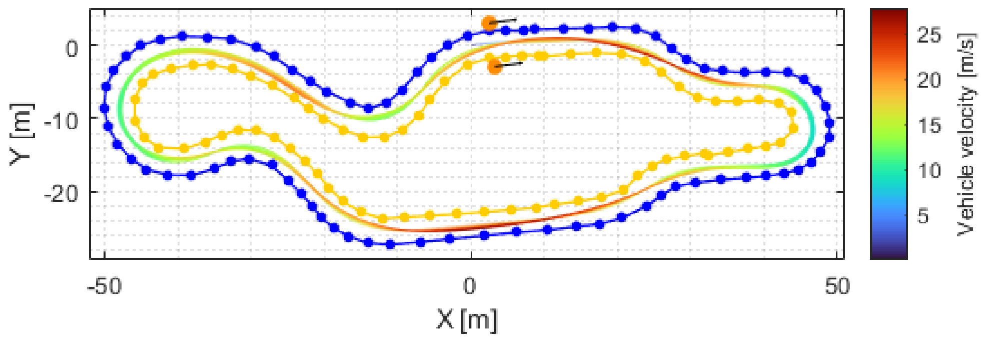
Appendix C

Results for Track 2 and 3

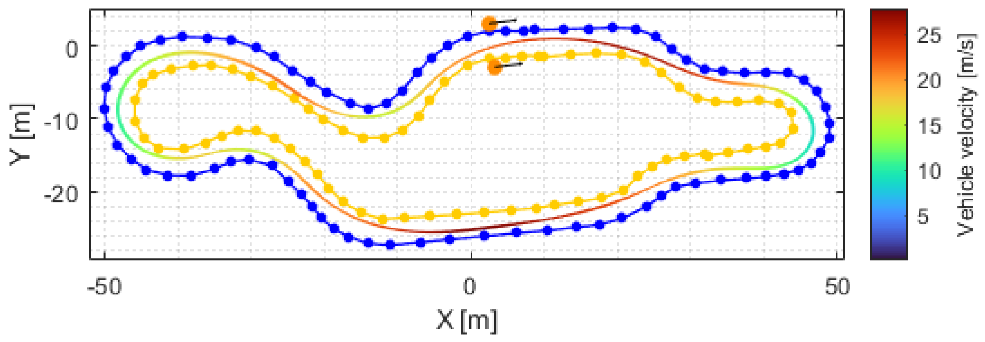
In this appendix, obtained results in simulation are presented for Track 2 and 3 for both vehicles - FST10d and AMZ's Gotthard.



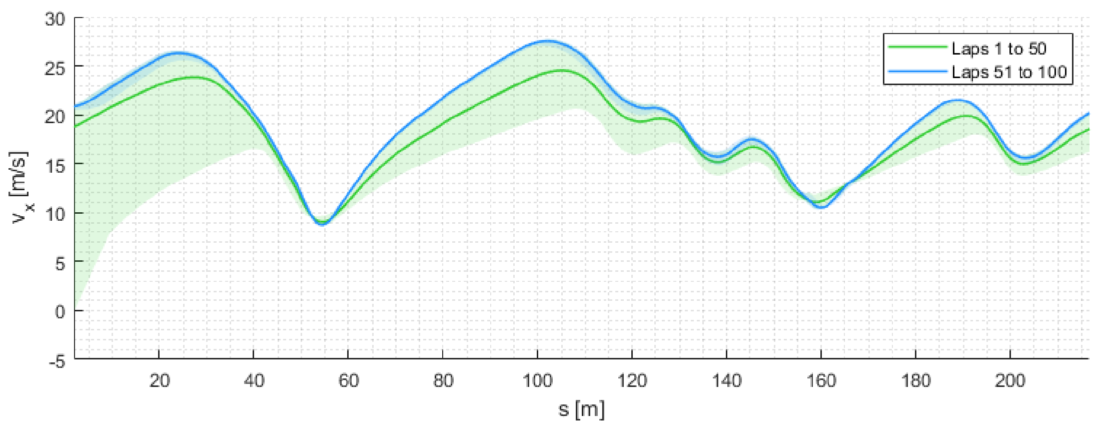
(a) Lap time evolution.



(b) Laps 1 to 50.

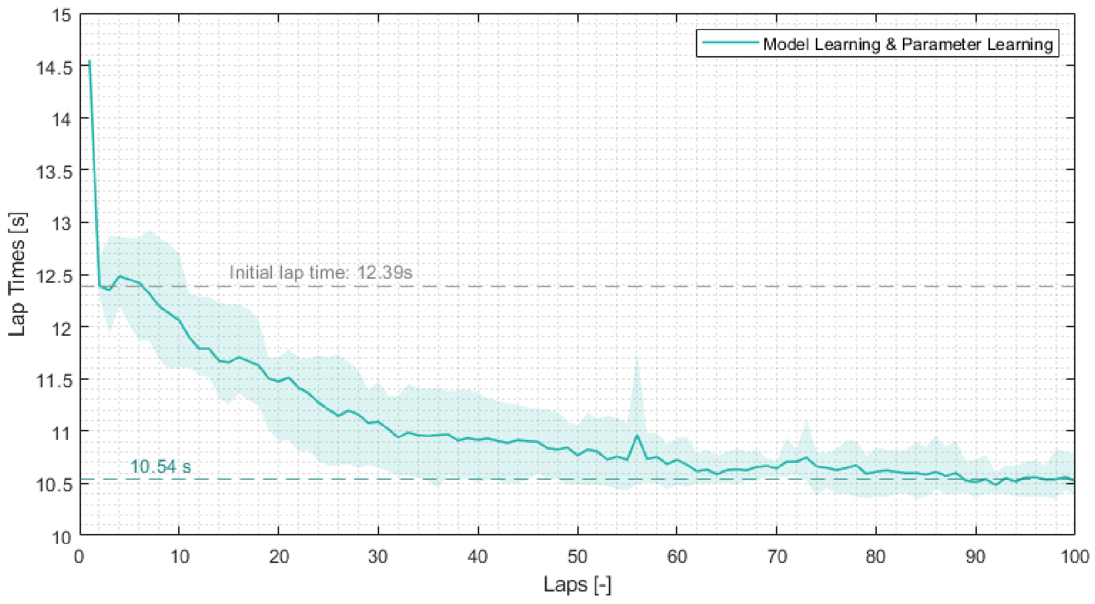


(c) Laps 51 to 100.

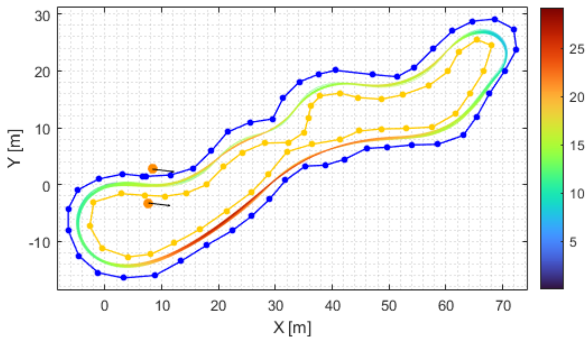


(d) Longitudinal velocity evolution.

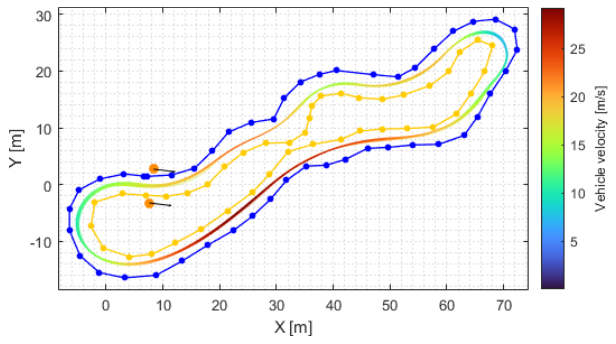
Figure C.1: FST10d simulation results for Track 2 using the full algorithm.



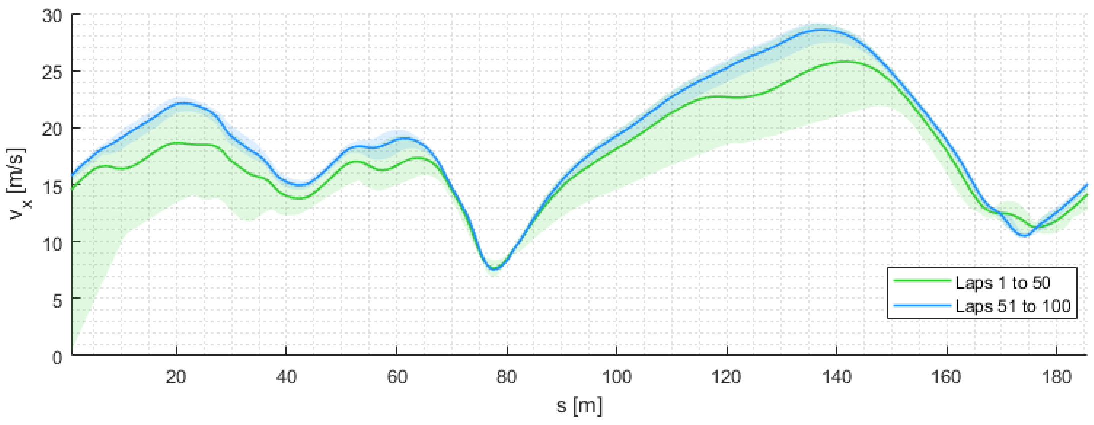
(a) Lap time evolution.



(b) Laps 1 to 50.

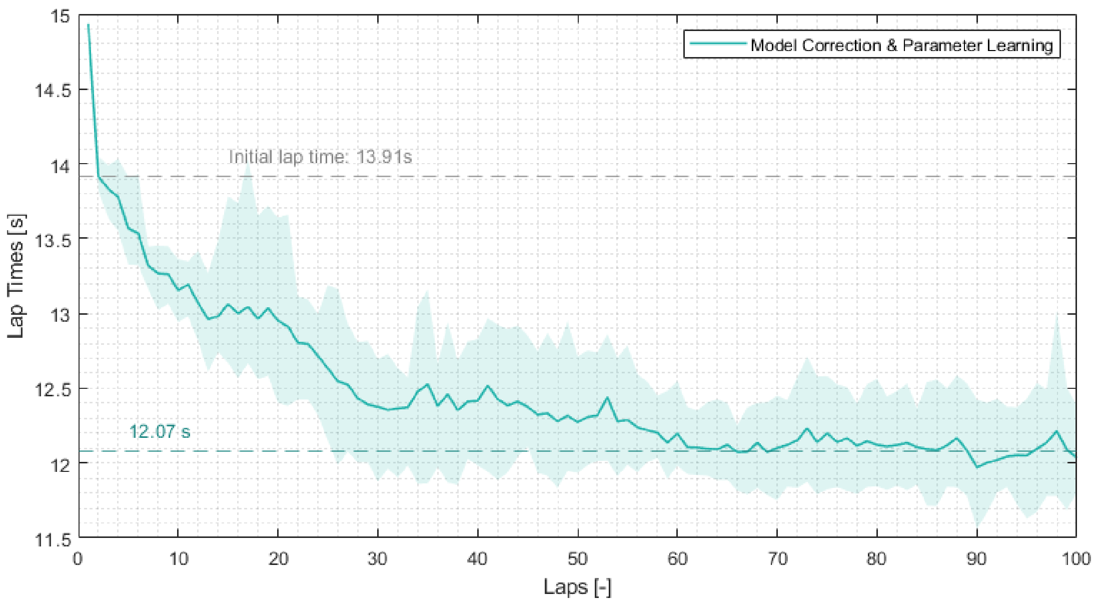


(c) Laps 51 to 100.

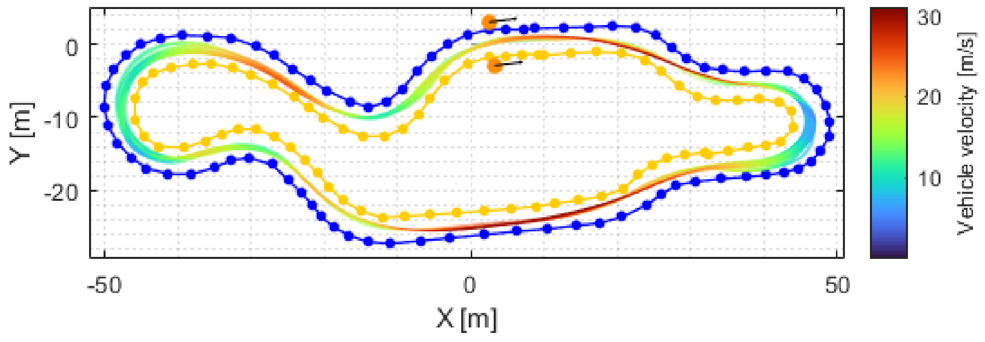


(d) Longitudinal velocity evolution.

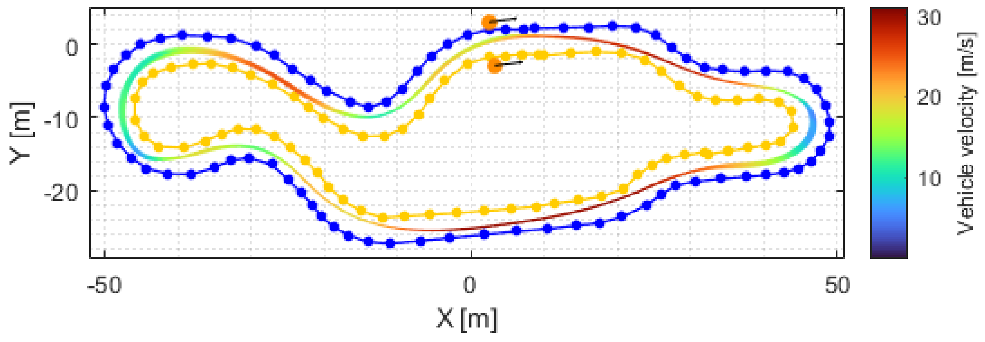
Figure C.2: FST10d simulation results for Track 3 using the full algorithm.



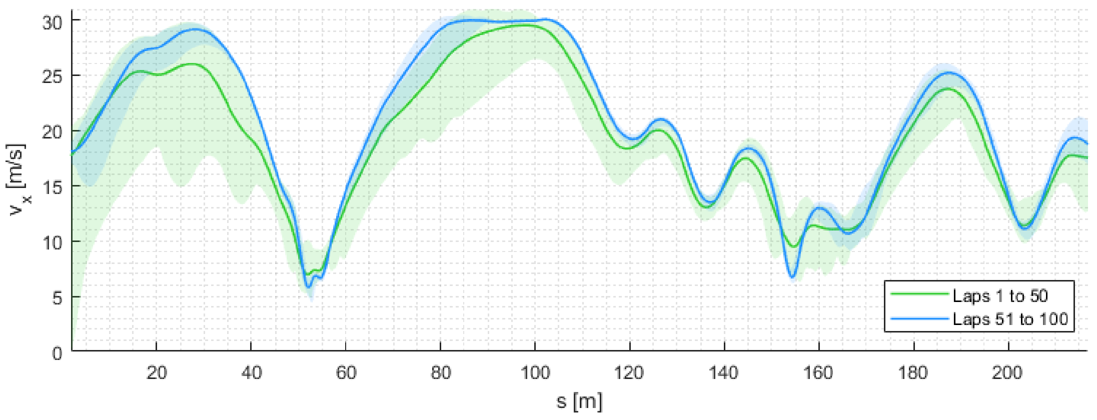
(a) Lap time evolution.



(b) Laps 1 to 50.

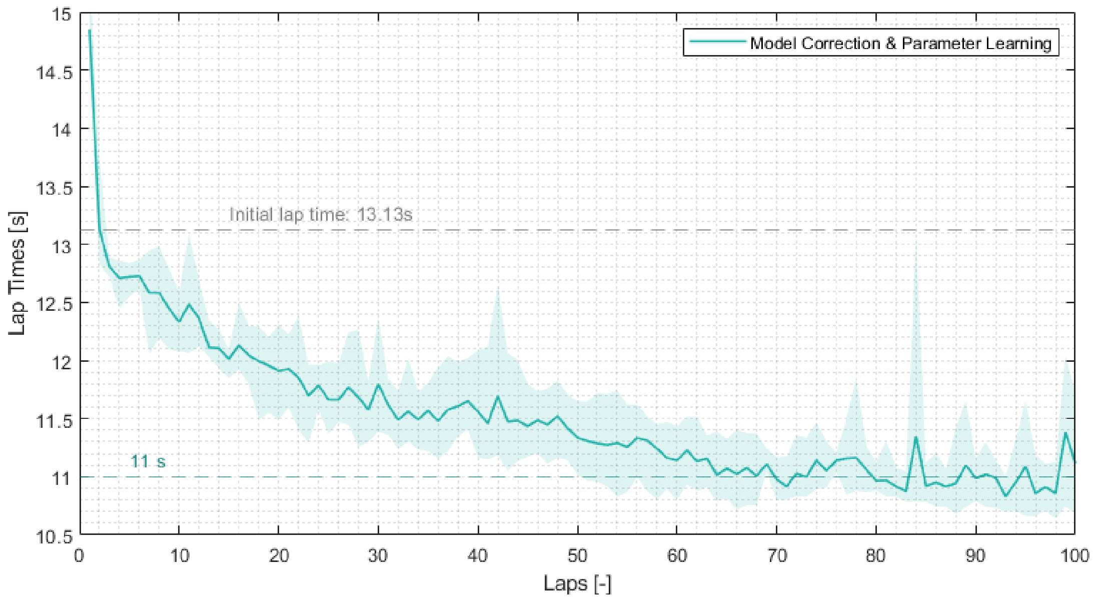


(c) Laps 51 to 100.

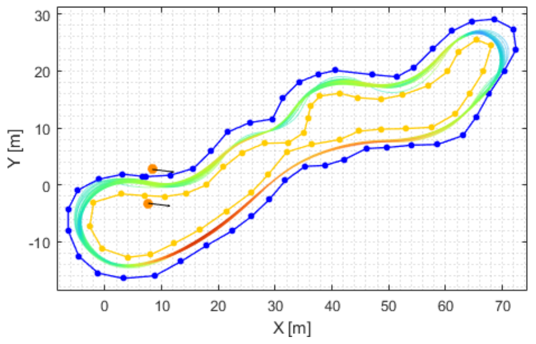


(d) Longitudinal velocity evolution.

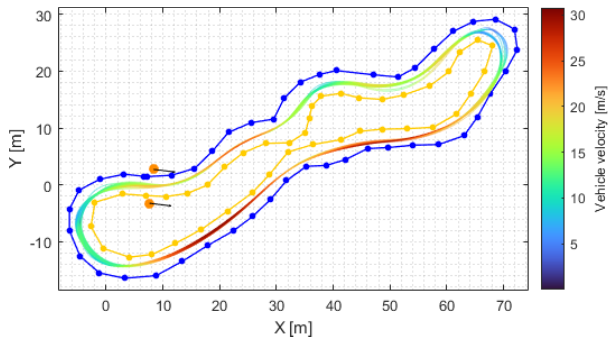
Figure C.3: AMZ's Gotthard simulation results for Track 2 using the full algorithm.



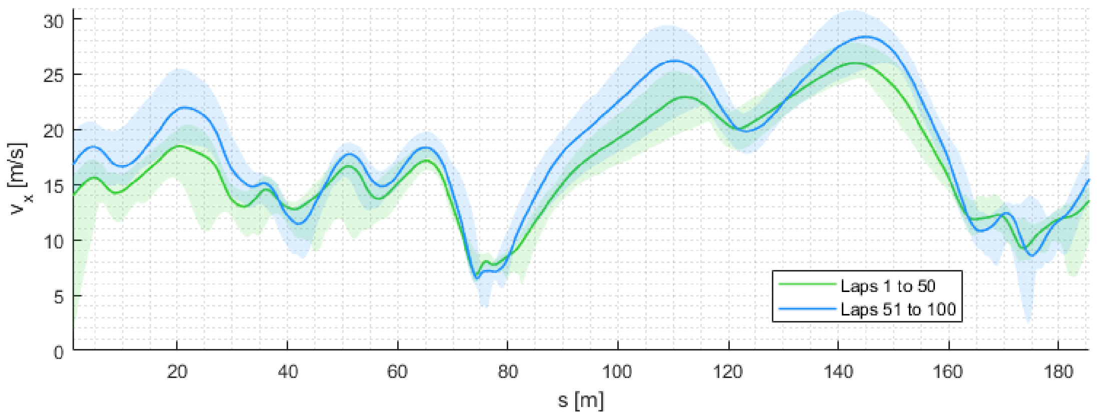
(a) Lap time evolution.



(b) Laps 1 to 50.



(c) Laps 51 to 100.



(d) Longitudinal velocity evolution.

Figure C.4: AMZ's Gotthard simulation results for Track 3 using the full algorithm.