Online learning of MPC for autonomous racing

Gabriel Alexandre Francisco Costa
gabrielafcosta@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

December 2021

Abstract

In this paper, 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

1. Introduction

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.

Various control algorithms are being researched for the application of autonomous driving. These controllers widely range in terms of complexity. This spectrum ranges from simple and well studied controllers like Proportional-Integral-Derivative (PID) controllers [1] up to more complex optimal based controllers like Model Predictive Control (MPC) [2].

Moreover, [3] overviewed some of this recent research regarding Learning-based Model Predictive Control (LMPC). 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.

Following this increasing research, 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.

In this paper, a nonlinear MPC for FST Lisboa's FST10d driverless vehicle is developed for the tasks of trajectory planning and control. The controller mainly adopts techniques from the two first topics previously mentioned: learning the system dynamics and learning the controller design. Learning the system dynamics is accomplished by adding an Artificial Neural Network (ANN) model to the blended bicycle model suggested in [4] - a model built on Newtonian mechanics, leading to a gray-box system identification method. Learning the controller design is accomplished by an algorithm that automatically tunes the parameters used in the MPC's cost function and constraints in a Reinforcement Learning (RL) environment based on time measurements between given track segments divided by check points.

Moreover, the MPC's cost function and constraints are based on the idea of [5] which couples the vehicle's path planning and control tasks into one single task, allowing for the development of a control method that is based on the track length maximization for a given prediction horizon.

2. Theoretical Background

As a 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

Model Predictive Control (MPC) is based on the idea of Receding Horizon Control (RHC), which 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 [6].

The general framework of such controller in typically achieved by implementing the following general steps:

1. At a given sampling instant, t, predict over the prediction horizon, N, the 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 such, at each sampling instant a potentially nonlinear optimization problem is solved in order to find the optimal control actions to apply at time instant t, $\mathbf{u}_t^*$. The definition of this optimization problem is typically dependent on the given application, however, such problem can be generally represented by the following equations:

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

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

$$\mathbf{x}_t = \mathbf{x}(t) \tag{1c}$$

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

$$\mathbf{x}_k \in \mathbb{X} \quad \forall k \in [t, ..., t+N] \tag{1e}$$

Where $\mathbf{u}_k$, $k = t, ..., t+N-1$ represents the sequence of the next N optimal inputs to be determined. Equation (1b) represents an equality constraint which translates the state prediction model used by the MPC. Equation (1c) 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 (1d) and (1e) represent the (possibly non-linear) equality or inequality constraints of the inputs and states of the system, respectively.

2.2. Supervised Learning

Supervised Learning (SL) is inserted in the broad topic of machine learning. 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 [7]. In this paper, 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 vectors 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.

The learner function, f, is generally obtained by solving an optimization problem which is, in general, based on the average of the loss function over the training data, obtaining the trained learner function, f^*, as the solution of the following Empirical Risk Minimization (ERM) problem.

$$f^* = \arg\min_f \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, f(\mathbf{v}_i)) \tag{2}$$

2.3. 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) [8]. 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 based in the RBFN case and elliptical based in the EBFN case. Mathematically, consider the previously mentioned input feature vector, $\mathbf{v}$, with n_v features, and $\boldsymbol{\mu}_i$ and $\boldsymbol{\Omega}_i$ the cluster center and the positive definite precision matrix of neuron i in the hidden layer, respectively. Given n_h hidden neurons, the activation of neuron i in the hidden layer, h_i, is computed through following equation.

$$h_i = \exp\left(-(\mathbf{v} - \boldsymbol{\mu}_i)^T \boldsymbol{\Omega}_i (\mathbf{v} - \boldsymbol{\mu}_i)\right) \ , \ i = 1, ..., n_h \tag{3}$$

Moreover, as a simplification, the precision matrix of each cluster $\boldsymbol{\Omega}_i$ is considered to be a diagonal positive definite matrix: $\boldsymbol{\Omega}_i \approx \text{diag}(\omega_{i,1}, ..., \omega_{i,n_v})$, $\omega_{i,l} > 0$, $l = 1, ..., n_v$, $i = 1, ..., n_h$. The network output is then 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:

$$e_j = \mathbf{w}_j \cdot \mathbf{h} \quad , \quad j = 1, ..., n_e \tag{4}$$

Regarding the training of these networks, [9] proposes a method which, in summary, applies a clustering algorithm in the input data and estimates the covariance matrix of the data with respect to the cluster centers. Then, the values for $\omega_{i,l}$ are computed by inverting the diagonal terms of each clusters' covariance matrix. 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 (4).

2.4. Online Gradient Descent

Consider $R_\mathbf{p}$ the Empirical Risk term in equation (5).

$$R_\mathbf{p} = \frac{1}{N} \sum_{i=1}^{N} L(\mathbf{y}_i, f_\mathbf{p}(\mathbf{v}_i)) \tag{5}$$

The Online Gradient Descent's (OGD's) objective is to minimize the Empirical Risk, as shown in the previous equation by using the most recent data point. In other words, the parameters are updated by computing the gradient of the loss function of the most recent data point alone:

$$\mathbf{p}_{k+1} = \mathbf{p}_k - \eta \nabla_{\mathbf{p}} \left[L\left(\mathbf{y}_k, f_{\mathbf{p}_k}\left(\mathbf{v}_k\right)\right)\right] \tag{6}$$

Where index k denotes the most recent value, $\eta > 0$ the learning rate and $f_{\mathbf{p}_k}$ the learner function as parameterized by the current parameter vector, $\mathbf{p}_k$.

2.5. Levenberg-Marquardt Algorithm

Consider an ERM problem written as the sum of sum squared errors of the learning batch, as follows.

$$R_{\mathbf{p}} = \sum_{i=1}^{N} \sum_{j=1}^{n_e} \left([\mathbf{y}_i]_j - [\hat{\mathbf{y}}_i]_j \right)^2 \tag{7}$$

In the previous equation, N represents the number of samples in the training set and n_e the number of outputs of the learner function. Moreover, $[\mathbf{y}_i]_j$ represents the j^{th} component of the i^{th} target vector in the training set. Finally, $\hat{\mathbf{y}}_i$ is the output of the learner function for feature vector i in the training set: $\hat{\mathbf{y}}_i = f_{\mathbf{p}}\left(\mathbf{v}_i\right)$.

The Levenberg-Marquardt (LM) algorithm requires the computation of the learner function jacobian with respect to its parameters. This jacobian is computed as follows:

$$\left[\mathbf{J}_f^i\right]_{l,j} = \frac{\partial \left[f_{\mathbf{p}}\left(\mathbf{v}_i\right)\right]_j}{\partial p_l} \ , \ l = 1, ..., n_p, \ , \ j = 1, ..., n_e \tag{8}$$

Then, the LM parameter update variation is computed by solving the following linear system of equations.

$$\left(\mathbf{J}_R {\mathbf{J}_R}^T + \lambda \mathbf{I}\right) \Delta \mathbf{p} = \mathbf{J}_R \left(\mathbf{y}_R - \hat{\mathbf{y}}_R\right) \tag{9}$$

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:

$$\mathbf{J}_R = \begin{bmatrix} \mathbf{J}_f^1 & \cdots & \mathbf{J}_f^N \end{bmatrix} \tag{10a}$$

$$\mathbf{y}_R = \begin{bmatrix} \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_N \end{bmatrix} \tag{10b}$$

$$\hat{\mathbf{y}}_R = \begin{bmatrix} \hat{\mathbf{y}}_1 \\ \vdots \\ \hat{\mathbf{y}}_N \end{bmatrix} \tag{10c}$$

After solving the linear system of equations (9), the parameter update is computed as follows.

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \Delta \mathbf{p} \tag{11}$$

The $\lambda > 0$ parameter seen in equation (9) is commonly referred to as the LM's damping parameter. Moreover, it can be mathematically proved that matrix $\mathbf{J}_R {\mathbf{J}_R}^T + \lambda \mathbf{I}$ is positive definite. As such, this system of equations can be efficiently solved resourcing to a Cholesky decomposition.

2.6. Reinforcement Learning

Reinforcement Learning (RL) is inserted in the broad topic of machine learning. Unlike Supervised Learning, 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 [7].

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.

RL algorithms are based on discrete time Markov Decision Processes (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, $\mathbf{s}_0 \in \mathbb{S}$;
- 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 $\mathrm{P}\left(r_{t+1} | \mathbf{s}_t, \mathbf{a}_t\right)$.

The main problem 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:

$$V_\pi\left(\mathbf{s}\right) = \mathbb{E}\left[\sum_{t=0}^{T} r\left(\mathbf{s}_t, \pi\left(\mathbf{s}_t\right)\right) \middle| \mathbf{s}_0 = s \right] \tag{12}$$

2.7. Genetic Algorithm

Genetic Algorithms (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 idea behind these algorithms is to create new generations better than the previous ones based on a desired purpose. Generations are

composed by various 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 [10]:

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.

To mathematically define the inner workings of the GA, consider an optimization problem that, for example, reflects the reward maximization problem presented in the previous section regarding RL. One can generally define this problem as follows.

$$\max_{\mathbf{d}} r\left(\mathbf{d}\right) \tag{13}$$

The reward function, $r \geq 0$, depends on the vector $\mathbf{d} \in \mathbb{R}^{n_d}$ which, in turn, represents a genetic string. 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. $\mathbf{D}$ can be defined as follows.

$$\mathbf{D} = \begin{bmatrix} \mathbf{d}_1 & \dots & \mathbf{d}_N \end{bmatrix} \tag{14}$$

New generations are stochastically created through the previously mentioned genetic operations. As such, there is a probability associated with each individual which is computed by its reward value as follows, obtaining each individuals relative fitness:

$$f_j = \frac{r_j}{\sum_{n=1}^{N} r_n} \quad , \quad j = 1, ..., N \tag{15}$$

As such, f_j represents the probability of picking individual j for performing a genetic operation when creating the new generation.

For further insight on Genetic Algorithms and their applications, the author recommends the book [10].

3. Vehicle Models

The employed nominal vehicle model built on Newtonian mechanics is based on two well known vehicle models: the kinematic and dynamic bicycle models. These are well known vehicle models which can be found in books such as [11]. Moreover, the geometry representation of these models can be seen in figure 1.

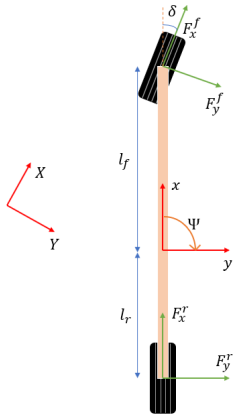


Figure 1: Bicycle model geometry.

These models should predict the evolution of the vehicle state vector, described as $\mathbf{x} = [X, Y, \Psi, v_x, v_y, r]^T$. Where X and Y refer to the vehicle global position coordinates, Ψ is the vehicle's orientation, v_x and v_y correspond to the vehicle's longitudinal and lateral velocities, respectively, and r the time derivative of the vehicle's orientation, $r \equiv \dot{\Psi}$. Moreover, the input vector, containing the vehicle's normalized throttle, d, and the steering angle, δ, is defined as $\mathbf{u} = [d, \delta]^T$.

3.1. Kinematic Bicycle Model

The continuous time state space equations of the kinematic bicycle model can be consulted in the following equation.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \\ \dot{v_x} \end{bmatrix} = \begin{bmatrix} v_x \cos(\Psi) - v_y \sin(\Psi) \\ v_x \sin(\Psi) + v_y \cos(\Psi) \\ \frac{v_x}{l_r + l_f} \tan(\delta) \\ \frac{F_x}{m} \end{bmatrix} \tag{16}$$

These equations do not model the full state vector desired for the six vehicle states. However, the remaining, states, v_y and r, can be retrieved from the following kinematic relations:

$$r = \frac{v_x}{l_r + l_f} \tan(\delta) \tag{17a}$$

$$v_y = r l_r = v_x \tan(\delta) \frac{l_r}{l_r + l_f} \tag{17b}$$

Where F_x represents the longitudinal forces being applied to the vehicle. This component is modeled by a sum of forces such as the propulsion force, aerodynamic drag and rolling resistance, and can be computed as:

$$F_x = 2\eta_{\text{motor}} \frac{T_{\max} \cdot GR}{r_{\text{wheel}}} d - \frac{1}{2} \rho C_d A_F {v_x}^2 - C_r m g \tag{18}$$

3.2. Dynamic Bicycle Model

The main difference between the kinematic and dynamic models is that the latter also takes into account the lateral forces applied by the tires. These forces are modeled using the following equations [4].

$$F_y^f = -2D_f \sin\left(C_f \arctan\left(B_f \cdot \alpha_f\right)\right) \tag{19a}$$

$$F_y^r = -2D_r \sin\left(C_r \arctan\left(B_r \cdot \alpha_r\right)\right) \tag{19b}$$

Where the indexes f and r denote the front and rear wheels respectively. Moreover, the tire slip angles of the front and rear wheels, α_f and α_r, can be computed as follows.

$$\alpha_f = \arctan\left(\frac{v_y + \dot{\Psi} l_f}{v_x}\right) - \delta \tag{20a}$$

$$\alpha_r = \arctan\left(\frac{v_y - \dot{\Psi} l_r}{v_x}\right) \tag{20b}$$

The continuous time state space equations of the dynamic bicycle model can be consulted in the following equation.

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\Psi} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} v_x \cos(\Psi) - v_y \sin(\Psi) \\ v_x \sin(\Psi) + v_y \cos(\Psi) \\ r \\ \frac{1}{m}\left(F_x - F_y^f \sin(\delta)\right) + v_y r \\ \frac{1}{m}\left(F_y^f \cos(\delta) + F_y^r\right) + v_x r \\ \frac{1}{I_z}\left(F_y^f \cos(\delta) \cdot l_f - F_y^r \cdot l_r\right) \end{bmatrix} \tag{21}$$

3.3. Blended Bicycle Model

The two previously mentioned nonlinear models are discretized using a Runge-Kutta 4$^{\text{th}}$ order integrator. As such, two discrete time models, $\mathbf{x}_{k+1} = f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k)$ and $\mathbf{x}_{k+1} = f_{\text{dyn}}(\mathbf{x}_k, \mathbf{u}_k)$, are obtained for the kinematic and dynamic bicycle models, respectively.

As the kinematic model is reliable for low speed and unrealiable for high speeds and the dynamic model is reliable for high speeds and unreliable for low speeds, these models are then fused together using a similar approach to [4].

$$\begin{aligned} \mathbf{x}_{k+1} &= \lambda_k f_{\text{dyn}}(\mathbf{x}_k, \mathbf{u}_k) + (1 - \lambda_k) f_{\text{kin}}(\mathbf{x}_k, \mathbf{u}_k) \\ &= f_{\text{blend}}(\mathbf{x}_k, \mathbf{u}_k) \end{aligned} \tag{22}$$

Where $\lambda_k \in [0, 1]$ 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 kinetic 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.

$$\lambda_k = \min\left(\max\left(\frac{V_k - V_{\text{blend}_{\min}}}{V_{\text{blend}_{\max}} - V_{\text{blend}_{\min}}}, 0\right), 1\right) \tag{23}$$

3.4. Error Correction Model

Consider the definition of discrepancy in equation (24).

$$\boldsymbol{\epsilon}_k^{(X,Y)} = \mathbf{x}_k^{\text{real}} - f_{\text{vehicle}}(\mathbf{x}_{k-1}, \mathbf{u}_{k-1}, \mathbf{p}_{k-1}) \tag{24}$$

The discrepancy vector measures the difference between the vehicle model prediction and the real measured/estimated vehicle states. Moreover, the vehicle state transition function is defined by the previously introduced blended bicycle model added with an EBFN learning model for model correction purposes, as follows in equation (25).

$$f_{\text{vehicle}}(\mathbf{x}, \mathbf{u}, \mathbf{p}) = f_{\text{blend}}(\mathbf{x}, \mathbf{u}) + e_{\Psi}^{(X,Y)}(\mathbf{p}, \mathbf{v}) \tag{25}$$

Where $\mathbf{v} \subseteq \{\mathbf{x}, \mathbf{u}\}$ is the feature vector, a subset of the state vector and control actions, and $\mathbf{p}$ is the parameter vector of the EBFN model.

3.4.1 Feature Selection

The feature vector, $\mathbf{v}$, is composed by the local vehicle states and the control actions. The local vehicle states are v_x, v_y and r as these quantities are locally related to the vehicle's frame of reference. In short, local features are used since these should not allow for the EBFN to learn track related features. Nevertheless, as suggested by equation (17b), in the kinematic bicycle model section, v_y and r are highly correlated. Thus, only one of these quantities is present in the feature vector. Hence, the feature vector is defined as $\mathbf{v} = [v_x, r, d, \delta]^T$.

3.4.2 Training Algorithm

The initial parameter vector of the EBFN is computed according to a similar technique to the one presented in [9]. Then, in simulation, these parameters are fine tuned using the "Online Levenberg-Marquardt" (OLM) algorithm, which will be further described. This 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.

To understand the developed training algorithm, one should first analyze the discrepancy vector previously mentioned: $\boldsymbol{\epsilon}^{(X,Y)} = \left[\epsilon_X, \epsilon_Y, \epsilon_\Psi, \epsilon_{v_x}, \epsilon_{v_y}, \epsilon_r\right]^T$. This vector's first two components, regarding the spatial coordinates discrepancy, ϵ_X and ϵ_Y are expressed in the global frame of reference, (X, Y). By the definition presented in equation (24) 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 section, 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. For the purpose of explaining this transformation consider the illustration in figure 2.

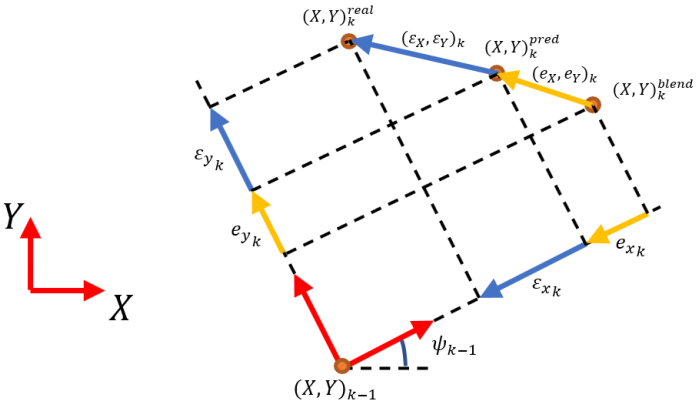


Figure 2: Discrepancy and error transformation to the previous local frame of reference

Given this, the following equations can be used to transform $\boldsymbol{\epsilon}^{(X,Y)}$ into $\boldsymbol{\epsilon}^{(x,y)}$ and $e^{(x,y)}$ into $e^{(X,Y)}$.

$$\boldsymbol{\epsilon}^{(x,y)} = \mathbf{A}^T\left(\Psi_{k-1}\right)\boldsymbol{\epsilon}^{(X,Y)} \tag{26a}$$

$$e^{(X,Y)}_{\Psi_{k-1}} = \mathbf{A}\left(\Psi_{k-1}\right)e^{(x,y)} \tag{26b}$$

Where the $\mathbf{A} \in \mathbb{R}^{6\times6}$ matrix is defined as:

$$\mathbf{A}\left(\Psi\right) = \begin{bmatrix} \mathbf{R}^{(X,Y)}_{(x,y)}\left(\Psi\right) & 0_{2\times4} \\ 0_{4\times2} & \mathbf{I}_{4\times4} \end{bmatrix} \tag{27}$$

Moreover, $\mathbf{R}^{(X,Y)}_{(x,y)} \in \mathbb{R}^{2\times2}$ is a rotation matrix that when left multiplied by a vector with components expressed in the vehicle's local frame of reference, rotates this vector to the global frame of reference. As such, this rotation matrix is defined as follows in equation (28).

$$\mathbf{R}^{(X,Y)}_{(x,y)} = \begin{bmatrix} \cos(\Psi) & -\sin(\Psi) \\ \sin(\Psi) & \cos(\Psi) \end{bmatrix} \tag{28}$$

With the local discrepancy defined, one can define an Empirical Risk Minimization (ERM) problem to minimize the discrepancy components. Consider the mean squared error of the discrepancy vector as follows.

$$\text{MSE}_k = \frac{1}{n_e}\sum_{j=1}^{n_e}\left(\epsilon_{k,j}\right)^2 \tag{29}$$

Where, for simplicity, ϵ_j corresponds to component j of the local discrepancy vector, $\boldsymbol{\epsilon}^{(x,y)}_k$. 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, $\boldsymbol{\epsilon}^{(x,y)}_t$ 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:

$$R = \frac{1}{\Gamma}\sum_{t=k-n_{\text{batch}}+1}^{k}\gamma^{k-t}\cdot\text{MSE}_t \tag{30}$$

Where the learning parameter $\gamma \in]0, 1]$ refers to the forgetting factor and Γ is a normalization parameter defined on the geometric series of the forgetting factor, γ. This term can be computed using equation (31).

$$\Gamma = \sum_{n=0}^{n_{\text{batch}}-1}\gamma^n = \frac{1-\gamma^{n_{\text{batch}}}}{1-\gamma} \tag{31}$$

To apply the LM algorithm's update step, one must first compute the jacobian of the learner function, denoted as $\mathbf{J}^{\boldsymbol{\epsilon}_k}_{\mathbf{p}}$, where component $\left[\mathbf{J}^{\boldsymbol{\epsilon}_k}_{\mathbf{p}}\right]_{i,j}$, in which index i represents a row and index j a column, can be computed as follows in equation (32). Moreover, the index k refers to the discrepancy obtained at instant k.

$$\left[\mathbf{J}^{\boldsymbol{\epsilon}_k}_{\mathbf{p}}\right]_{i,j} = \frac{\partial\epsilon_{k,j}}{\partial p_i} = -\frac{\partial e_j(\mathbf{p}, \mathbf{v}_{k-1})}{\partial p_i} \tag{32}$$

Then, the jacobian and discrepancy of the entire learning batch are computed by concatenating the jacobians and discrepancy vectors of each data point in the learning batch, as follows.

$$\mathbf{J}_{\mathbf{p}} = \begin{bmatrix} \mathbf{J}^{\boldsymbol{\epsilon}_k}_{\mathbf{p}} & \mathbf{J}^{\boldsymbol{\epsilon}_{k-1}}_{\mathbf{p}} & \cdots & \mathbf{J}^{\boldsymbol{\epsilon}_{k-n_{\text{batch}}+1}}_{\mathbf{p}} \end{bmatrix} \tag{33a}$$

$$\boldsymbol{\epsilon}^{\text{batch}} = \frac{2}{n_e\cdot\Gamma}\begin{bmatrix} \boldsymbol{\epsilon}_k \\ \gamma\cdot\boldsymbol{\epsilon}_{k-1} \\ \vdots \\ \gamma^{n_{\text{batch}}-1}\cdot\boldsymbol{\epsilon}_{k-n_{\text{batch}}+1} \end{bmatrix} \tag{33b}$$

Finally, the linear system of equations (34) is solved for $\Delta\mathbf{p}$, and the parameters are updated as in equation (11).

$$\left(\lambda I + \mathbf{J}_{\mathbf{p}}{\mathbf{J}_{\mathbf{p}}}^T\right)\Delta\mathbf{p} = -\mathbf{J}_{\mathbf{p}}\boldsymbol{\epsilon}^{\text{batch}} \tag{34}$$

4. MPC Formulation

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 [12, 5, 4]. An illustration of this process can be seen in figure 3.

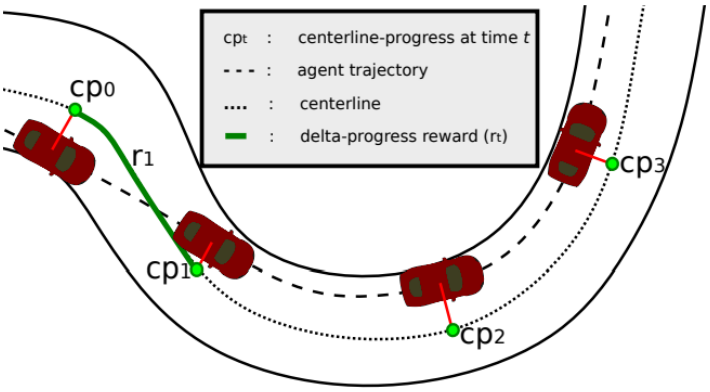


Figure 3: Track progress based on the track's center line [5].

Furthermore, 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.

$$\mathbf{g}(s) = \begin{bmatrix} g_X(s) \\ g_Y(s) \end{bmatrix} \tag{35}$$

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 (36).

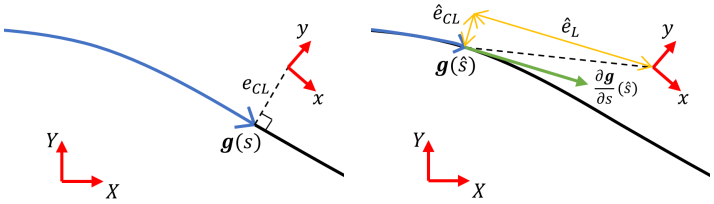
$$\frac{\partial\mathbf{g}}{\partial s}(s) = \begin{bmatrix} \frac{\partial g_X}{\partial s}(s) \\ \frac{\partial g_Y}{\partial s}(s) \end{bmatrix} \tag{36a}$$

$$\frac{\partial^2 \mathbf{g}}{\partial s^2}(s) = \begin{bmatrix} \frac{\partial^2 g_X}{\partial s^2}(s) \\ \frac{\partial^2 g_Y}{\partial s^2}(s) \end{bmatrix} \tag{36b}$$

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.

$$s_k = \arg\min_s \sqrt{(X_k - g_X(s))^2 + (Y_k - g_Y(s))^2} \tag{37}$$

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 [4, 5], a method for approximating the values of s_k and e_{CL_k} is employed. Figure 4 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*.



(a) True projection on the center line.

(b) Approximated projection on the center line.

Figure 4: Approximation mechanism for the track progress.

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

$$\hat{e}_{CL}(\hat{s}) = \frac{\partial g_Y}{\partial s}(\hat{s}) \cdot (X - g_X(\hat{s})) - \frac{\partial g_X}{\partial s}(\hat{s}) \cdot (Y - g_Y(\hat{s})) \tag{38a}$$

$$\hat{e}_{CL}(\hat{s}) = -\frac{\partial g_X}{\partial s}(\hat{s}) \cdot (X - g_X(\hat{s})) - \frac{\partial g_Y}{\partial s}(\hat{s}) \cdot (Y - g_Y(\hat{s})) \tag{38b}$$

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

The optimization problem formulated in equations (39) was used and was influenced on the idea of [5].

$$\begin{aligned}
\min_{\substack{\mathbf{u}_k, \forall k \in [t+1,\dots,t+N-1] \\ s_k \forall k \in [t,\dots,t+N]}} \quad & \alpha_{CL} \hat{e}^n_{CL_t} + \alpha_L \hat{e}^2_{L_t} + \\
& \sum_{k=t+1}^{t+N-1} \left(q_{v_y} {v_{y_k}}^2 + \alpha_{CL} \hat{e}^n_{CL_k} + \alpha_L \hat{e}^2_{L_k} + \beta_\delta (\delta_k - \delta_{k-1})^2 + e^{q_{v_{\max}}(v_{x_k} - v_{\max})} \right) \\
& - \lambda_s s_{t+N} + q_{v_y} {v_{y_{t+N}}}^2 + \alpha_{CL} \hat{e}^n_{CL_{t+N}} + \alpha_L \hat{e}^2_{L_{t+N}} + e^{q_{v_{\max}}(v_{x_{t+n}} - v_{\max})} && (39a) \\
\text{s.t.} \quad & \mathbf{x}_{k+1} = f_{\text{vehicle}}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{p}_t) \quad \forall k \in [t, \dots, t+N-1] && (39b) \\
& \mathbf{x}_t = \mathbf{x}(t) && (39c) \\
& \mathbf{u}_t = \mathbf{u}(t) && (39d) \\
& \mathbf{s}_{t-1} = \mathbf{s}(t-1) && (39e) \\
& -\Delta \mathbf{u}_{\max} \le \mathbf{u}_k - \mathbf{u}_{k-1} \le \Delta \mathbf{u}_{\max} \quad \forall k \in [t+1, \dots, t+N-1] && (39f) \\
& \mathbf{u}_{\min} \le \mathbf{u}_k \le \mathbf{u}_{\max} \quad \forall k \in [t+1, \dots, t+N-1] && (39g) \\
& -\hat{e}_{CL_{\max}} \le \hat{e}_{CL_k} \le \hat{e}_{CL_{\max}} \quad \forall k \in [t+1, \dots, t+N] && (39h) \\
& \Delta s_{\min} \le s_k - s_{k-1} \le \Delta s_{\max} \quad \forall k \in [t, \dots, t+N] && (39i)
\end{aligned}$$

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 (39a), 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.

Overall, 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 relate to the aggressiveness of the controller. Moreover, $\alpha_{CL} > 0$ penalizes the vehicle's center line distance, $d_{\max} > 0$ limits the maximum allowed vehicle acceleration to $d_{\max}$, $q_{v_y} > 0$ penalizes high lateral velocities which are characteristic of a sideways drifting scenario which may lead to a crash, $n \in \{2, 4, 6, \dots\}$ is the exponent of the center line penalization term, i.e. shaping this term, and $\beta_\delta > 0$ controls the smoothness of the steering angle.

5. Controller Parameter Learning

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.

5.1. 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 times. 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, $v(s)$, is expressed in equation (40). 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 3.2. Moreover, m stands for the mass of the vehicle and $v_{\max}$ the maximum velocity the vehicle can achieve in a straight line.

$$v(s) = \min\left(v_{\max}, \sqrt{R(s)\,\frac{2D_f + 2D_r}{m}}\right) \tag{40}$$

Furthermore, $R(s)$ 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 (36) from section 4.1. The mathematical definition of the curvature radius obtained from these derivatives is expressed in equation (41) [13].

$$R(s) = \frac{1}{\left|\frac{\partial g_X}{\partial s}(s)\cdot\frac{\partial^2 g_Y}{\partial s^2}(s) - \frac{\partial g_Y}{\partial s}(s)\cdot\frac{\partial^2 g_Y}{\partial s^2}(s)\right|} \tag{41}$$

Given a track and the required vehicle parameters, one can plot the maximum achievable velocity as seen in figure 5 represented by the green line.

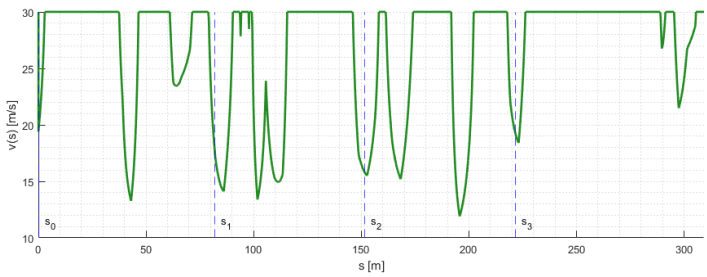


Figure 5: Maximum achievable velocity and track segmentation using 4 check points for the track shown in figure 8.

Still referring to figure 5, 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:

$$T_{\text{lap}} = \int_0^L \frac{1}{v(s)}\,ds \tag{42}$$

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:

$$\int_{s_{i-1}}^{s_i} \frac{1}{v(s)}\,ds = \frac{T_{\text{lap}}}{N_{\text{CP}}} \quad , \quad s_0 = 0 \quad , \quad i = 1, ..., N_{\text{CP}} - 1 \tag{43}$$

5.2. Reward Function

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

$$r\left(T_{i,k}, n_i^{\text{cones}}\right) = \exp\left(-k_T\left(T_{i,k} - T_{i,k}^{\text{avg}} + k_c n_i^{\text{cones}}\right)\right) \tag{44}$$

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.

The segment's average time, $T_{i,k}^{\text{avg}}$, is attained from a low pass filter. This is accomplished using the following equation.

$$T_{i,k+1}^{\text{avg}} = \lambda_T T_{i,k} + (1 - \lambda_T)\,T_{i,k-1}^{\text{avg}} \quad , \quad i = 1, ..., N_{\text{CP}} \tag{45}$$

5.3. Genetic Algorithm for Learning the Controller Design

Consider the set of run time MPC parameters previously presented in section 4.2: $\mathbf{d} = \left[\alpha_{CL}, d_{\max}, q_{v_y}, n, \beta_\delta\right]^T$. Here, $\mathbf{d}$ represents a genetic code string for the GA. 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:

$$\mathbf{D} = \begin{bmatrix}\mathbf{d}_1 & \dots & \mathbf{d}_N\end{bmatrix} \tag{46}$$

The GA is applied as follows. 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 (44). At the same time, the average time of that same segment is adjusted as per equation (45). 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. Therefore, 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 is computed according to equation (15). Then, 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 P (R), P (R) and P (M). Afterward, the selected genetic operations are applied to the population. When performing a genetic operation, the individuals are randomly selected according to a discrete probability distribution function according to the individuals' fitness vector. Upon completing every genetic operation, a new population matrix is obtained and the whole process is repeated.

6. Implementation

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 [4].

6.2. Controller 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 lightweight 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.

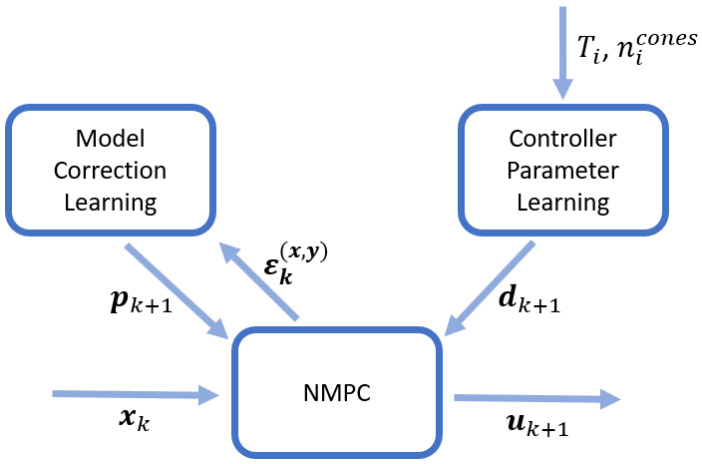


Figure 6: ROS nodes for the developed control architecture.

6.2.1 Controller Implementation

For solving the MPC optimization problem a commercial solver was employed. FORCESPRO [14, 15], 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 (39).

6.2.2 Model Correction

A neural network library for the application of the vehicle correction model 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[3] library.

6.2.3 Controller Parameter Learning

Regarding learning the MPC parameters, the procedure described in section 5 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.

7. Results

In this section, simulation results of the designed controller system are shown. For testing the controller robustness, the results shown were obtained using FST10d's - FST Lisboa's autonomous vehicle - parameters in the MPC dynamic vehicle model while using AMZ's Gotthard - the FS team from ETH Zürich - parameters in the simulator.

Figure 7 contains the lap times obtained for, in this case, two different tests for the MPC were performed: a test using the model correction node (in blue) and a test using both the model correction and controller parameter learning nodes (in red).

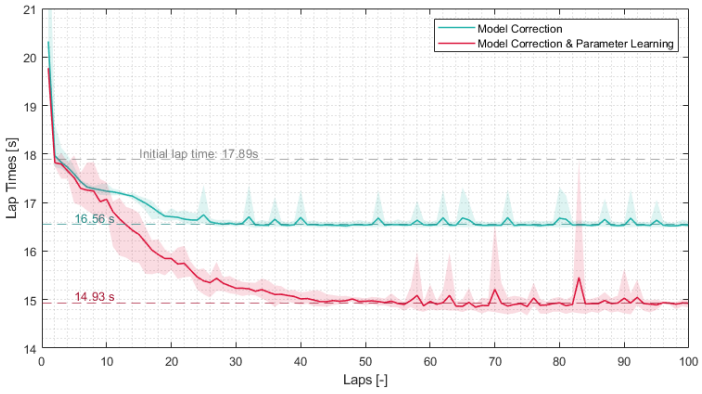


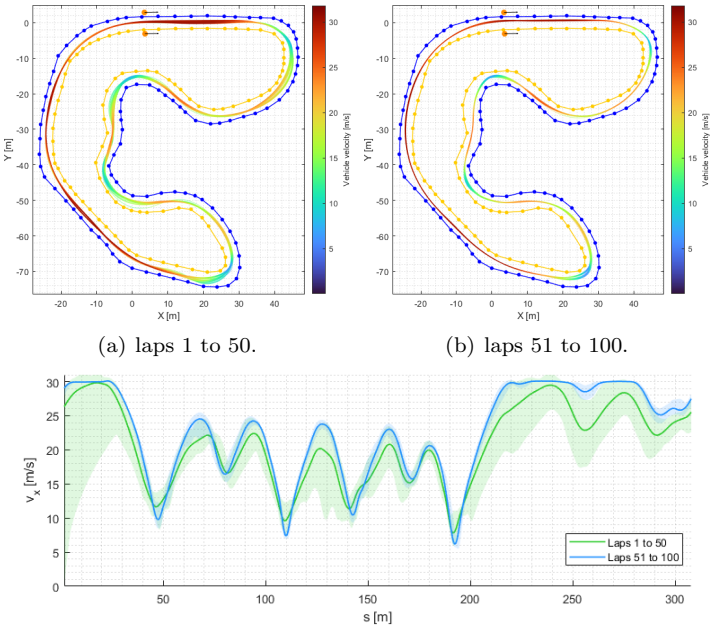
Figure 7: Lap time simulation results for AMZ's Gotthard in FSG.

As can be seen from this figure, by employing the model learning node alone, the lap times are reduced from 17.89 s to 16.56 s, a 7.43% reduction. By further employing the parameter learning algorithm, the lap times are reduced to 14.93 s, a total reduction of 16.5%.

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. Observing the vehicle's trajectory between lap 1 and lap 50, and

[1] https://github.com/AMZ-Driverless/fssim
[2] https://www.ros.org/
[3] https://eigen.tuxfamily.org/index.php?title=Main_Page

between lap 51 and lap 100, as in figure 8, it can be seen that during the learning period - laps 1 to 50 - 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.



(a) laps 1 to 50.

(b) laps 51 to 100.

(c) Longitudinal velocity evolution

Figure 8: AMZ's Gotthard simulation trajectory for a FSG 2018 track using the full algorithm.

For reference, in AMZ's most recent research paper [16], it is claimed that the optimal lap time in the FSG 2018 track 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 paper, 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 paper may potentially reach faster lap times than the ones presented in AMZ's paper.

8. 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 the FSG 2018 track, 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 of the methods developed in this paper 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 paper 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.

References

[1] W. Farag, "Complex trajectory tracking using pid control for autonomous driving," *International Journal of Intelligent Transportation Systems Research*, vol. 18, no. 2, pp. 356–366, 2020.

[2] 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)*, 2015, pp. 1094–1099.

[3] L. Hewing, K. P. Wabersich, M. Menner, and M. N. Zeilinger, "Learning-based model predictive control: Toward safe learning in control," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 3, no. 1, p. 269–296, 2020.

[4] 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.

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

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

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

[8] "RBF and EBF Models." [Online]. Available: https://abaqus-docs.mit.edu/2017/English/IhrUserMap/ihrc-Reference-RBFandEBF.htmGeneralizingRBFtoEBF

[9] 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*, vol. 11, no. 4, pp. 961–969, Jul. 2000.

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

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

[12] 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.

[13] "Curvature and Radius of Curvature." [Online]. Available: https://math24.net/curvature-radius.html

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

[15] 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*, pp. 1–17, 2017.

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