Learning to Drive Autonomously a Race Car

João Gomes de Oliveira Pinho
joao.g.pinho@tecnico.ulisboa.pt

Instituto Superior Técnico, Universidade de Lisboa, Lisboa, Portugal

September 2021

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

I. Introduction

Autonomous driving (AD) has been an increasingly active field of research for academia and the industry, especially in the last two decades, with pioneering events such as the 2005 DARPA Grand Challenge [1]. This thesis focus on Autonomous Racing, a subfield of Autonomous Driving that aims to contribute to the broader problem by introducing innovations in autonomous technology through sport [2]. This kind of synergy is well established between Formula One and the automotive industry.

[3] surveyed the state of the art on planning and control algorithms for AVs in the urban setting. Several controllers that resort to a kinematic bicycle model have been designed. For instance, the Stanley [1] controller. To handle more demanding driving manoeuvres, more complex controllers must be designed.

Advances in computing hardware and mathematical programming algorithms have made Model Predictive Control feasible for real-time use in AD [5]. This model-based technique relies on a sufficiently accurate state transition model whose complexity can only grow while the real-time feasibility of the online optimization framework is ensured.

Machine Learning techniques have been used to improve the formulation of the MPC using collected data [6]. This paper's pivot will be Learning-based Model Predictive Control. Most research has focused on using Machine Learning tools as a data-based adaptation of the prediction model or uncertainty description - *Model Learning*. Notwithstanding, learning has also targeted an MPC controller's parameterisation, *e.g.* the cost function, horizon length, or terminal components.

Several LMPC architectures have been proposed. Zeilinger's research group introduced a cautious Learning-based Model Predictive Controller that combines a nominal model with Gaussian Process Regression techniques to model the unknown dynamics [7]. It has been shown to increase safety and performance and has been applied to trajectory tracking with a robotic arm [8]. Schoellig's Dynamic Systems Lab proposed using Bayesian Linear Regression to model the unknown dynamics [9]. These researchers argue that this simple model is more accurate in estimating the mean behaviour and model uncertainty than GPR and generalizes to novel operating conditions with little or no tuning.

We target the 10-lap trackdrive event of the Formula Student Driverless autonomous racing competition. Formula Student (FS) is a student engineering design competition where participating university teams design, build, test and compete with a single seat formula racecar. The autonomous driving competition environment is a rather controlled one. For instance, no other agents, such as other vehicles or pedestrians, are immediately near the track. The track is composed of blue and yellow cones on the left and right borders, respectively. Exit and entry lanes are marked with small orange cones, while big orange cones are used in the start, finish and timekeeping lines.

AMZ Driverless, an FSD team, used a learning-based controller to tackle the issue relevant to autonomous racing that accurate vehicle models that cover the entire performance envelope are highly nonlinear and difficult to identify [10]. The proposed formulation considers a simple nominal vehicle model where GPR models residual model uncertainty. The approach is based on Model Predictive Contouring Control (MPCC) [11] and cautions MPC [7]. This framework was tested on an FSD prototype, achieving lap-time improvements of 10%.

The main contribution of this work is the adaptation of Professor Borrelli's MPC Lab terminal component learning LMPC architecture to the FSD context and the computational improvement of its implementation. Moreover, we further improve the ML technique that had been used for the model learning process. We show the architecture is able to improve the lap time by 12% over the course of the event.

II. Theoretical Background

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

A. Model Predictive Control

The idea of Receding Horizon Control is that an infinite horizon sub-optimal controller can be designed by repeatedly solving Finite-Time Constrained Optimal Control (FTCOC) problems in a receding horizon fashion [4]. At each sampling time, starting at the current state, an open-loop optimal control problem is solved over a finite horizon. The computed optimal manipulated input signal is applied to the process only during the following sampling interval $[t,\ t+1]$. At the next time-step $t+1$ a new optimal control problem based on new measurements of the state is solved over a shifted horizon.

MPC is a Receding Horizon Control problem where the FTCOC problem with a prediction horizon of N is computed by solving the following optimization problem online:

$$J^j_{t \to t+N}(x^j_t) =$$

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

$s.t.$

$$x_{k+1|t} = Ax_{k|t} + Bu_{k|t} \quad \forall k \in [t, \cdots, t+N-1] \quad \text{(1b)}$$

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

$$x_{t+N|t} \in \mathcal{X}_f \quad \text{(1d)}$$

$$x_{t|t} = x(t) \quad \text{(1e)}$$

where x is the state and u the control input. The subscript $k|t$ represents a given quantity in the prediction horizon with respect to time t. $x_{t|t}$ and $x_{t+N|t}$ represent the initial and terminal state of the system starting at time t, respectively. Equation (1e) imposes the current system state to be the initial condition of the generic FTCOC problem. Equation (1b) represents the discrete-time linear time-invariant system dynamics. State and input constraints are given by Equation (1c). The terminal constraint is given by Equation (1d) which forces the terminal state $x_{t+N|t}$ into some set $\mathcal{X}_f$. The stage $q(\cdot,\cdot)$ and terminal cost $p(x_{t+N|t})$ are any arbitrary continuous, strictly positive functions.

B. Learning-based Model Predictive Control

[12] first proposed the LMPC architecture which this work builds upon. This is a reference-free iterative control strategy able to learn from previous iterations. At each iteration, the initial condition, the constraints, and the objective function do not change. The authors show how to design a terminal safe set - $\mathcal{SS}$ - and a terminal cost function - Q-function - such that the following theoretical guarantees hold:

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

This framework's main contribution is to learn terminal constraints rather than model learning. Particularly, the terminal cost is given by the Q-function: $p(x_{t+N|t}) = Q^{j-1}(x_{t+N|t})$; whereas the terminal constraint corresponds to the terminal safe set $\mathcal{SS}$: $\mathcal{X}_f = \mathcal{SS}^{j-1}$.

$$\mathbf{x}^{*,j}_{t:t+N|t} = \left[x^{*,j}_{t|t}, \cdots, x^{*,j}_{t+N|t} \right] \quad \text{(2a)}$$

$$\mathbf{u}^{*,j}_{t:t+N|t} = \left[u^{*,j}_{t|t}, \cdots, u^{*,j}_{t+N-1|t} \right] \quad \text{(2b)}$$

The Equations (2a) and (2b) are the optimal state and control solution at time t of iteration j, respectively. At this instance, the control input applied to the system is the first element of $\mathbf{u}^{*,j}_{t:t+N|t}$:

$$u^j_t = u^{*,j}_{t|t} \quad \text{(3)}$$

At the jth iteration, the inputs applied to the system and the corresponding state evolution are collected in the vectors given by Equation (4b) and Equation (4a), respectively.

$$\mathbf{x}^j = \left[x^j_0, x^j_1, \cdots, x^j_t, \cdots \right] \quad \text{(4a)}$$

$$\mathbf{u}^j = \left[u^j_0, u^j_1, \cdots, u^j_t, \cdots \right] \quad \text{(4b)}$$

The safe set $\mathcal{SS}^j$, given by Equation (5) is the collection of all state trajectories at iteration i for $i \in M^j$ - the set of indexes k corresponding to the iterations that successfully steered the system to the final point x_F.

$$\mathcal{SS}^j = \left\{ \bigcup_{i \in M^j} \bigcup_{t=0}^{\infty} x^i_t \right\} \quad \text{(5)}$$

The Q^j function, defined in Equation (6), assigns to every point in the sampled safe set the minimum cost-to-go along the trajectories therein.

$$\forall x \in \mathcal{SS}^j, Q^j(x) = J^{i^*}_{t^* \to \infty}(x) = \sum_{k=t^*}^{\infty} q(x^{i^*}_k, u^{i^*}_k) \quad \text{(6)}$$

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

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

C. Gaussian Processes Regression

Gaussian Processes is a non-parametric, probabilistic Machine Learning approach to learning in *kernel* machines. By focusing on Gaussian processes, the problem becomes computationally tractable. Furthermore, it provides a fully probabilistic predictive distribution, including estimates of

the uncertainty of the predictions. For a detailed description of GPR, the author refers the reader to the book Gaussian Processes for Machine Learning by [13].

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

$$\mathbf{y}_k = \mathbf{g}(\mathbf{z}_k) + \mathbf{w}_k \tag{7}$$

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

$$\mathcal{D} = \Big\{ \mathbf{Y} = [\mathbf{y}_1^T; \cdots; \mathbf{y}_n^T] \in R^{n \times n_g}, \tag{8}$$

$$\mathbf{Z} = [\mathbf{z}_1^T; \cdots; \mathbf{z}_n^T] \in R^{n \times n_z} \Big\} \tag{9}$$

Assuming a Gaussian prior on $\mathbf{g}$ in each output dimension $d \in \{1, \cdots, n_g\}$, such that they can be treated independently, the posterior distribution in dimension d at an evaluation point $\mathbf{z}$ has mean and variance given by Equations (10a) and (10b), respectively. Further, in this situation, one refers to $\mathbf{Y}$ as $\mathbf{y}$. That is, there is a collection of n_d n-dimensional vectors $\mathbf{y}^d$.

$$\mu^d(\mathbf{z}) = \mathbf{k}_{\mathbf{zZ}}^d \big(\mathbf{K}_{\mathbf{ZZ}}^d + \mathbf{I}\sigma_d^2\big)^{-1} \mathbf{y}^d \tag{10a}$$

$$\Sigma^d(\mathbf{z}) = k_{\mathbf{zz}}^d - \mathbf{k}_{\mathbf{zZ}}^d \big(\mathbf{K}_{\mathbf{ZZ}}^d + \mathbf{I}\sigma_d^2\big)^{-1} \mathbf{k}_{\mathbf{Zz}}^d \tag{10b}$$

where $\mathbf{K}_{\mathbf{ZZ}}^d$ is the Gram matrix, *i.e.* $[\mathbf{K}_{\mathbf{ZZ}}^d]_{ij} = k^d(\mathbf{z}_i, \mathbf{z}_j)$, $[\mathbf{k}_{\mathbf{Zz}}^d]_j = k^d(\mathbf{z}_j, \mathbf{z})$, $\mathbf{k}_{\mathbf{Zz}}^d = (\mathbf{k}_{\mathbf{zZ}}^d)^{\mathrm{T}}$ and $k_{\mathbf{zz}}^d = k^d(\mathbf{z}, \mathbf{z})$ corresponds to the kernel function used.

D. Sparse Approximations for Gaussian Process Regression

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

A host of sparse approximation techniques have been proposed to allow the application of GPs to large problems in Machine Learning [14]. An additional set of $m < n$ latent variables $\bar{\mathbf{g}} = [\bar{g}_1, \cdots, \bar{g}_m]$, which are called inducing variables or support points, are used to approximate Equation (10). These are values of the Gaussian Process evaluated at the inducing inputs $\mathbf{Z}_{ind} = [\bar{\mathbf{z}}_0^T; \cdots; \bar{\mathbf{z}}_m^T]$. The latent variables are represented as $\bar{\mathbf{g}}$ rather than $\bar{\mathbf{y}}$ as they are not real observations. Thus, it does not make sense to include a noise variance.

The simplest sparse approximation method is the Subset of Data (SoD) approximation, *i.e.* solves Equation (10) by substituting $\mathbf{Z}$ by $\mathbf{Z}_{ind}$. It is often used as a baseline for sparse approximations. The computational complexity is reduced to $\mathcal{O}(m^3)$ for training; and $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ for the mean and variance, respectively. In order to improve the chances of good performance, rather than selecting the m points randomly, researchers have designed methods to select which points are included in the active set [15].

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

The Fully Independent Training Conditional (FITC) [16] assumes all the training datapoints are independent. The computational complexity is $\mathcal{O}(nm^2)$ initially, and $\mathcal{O}(m)$ and $\mathcal{O}(m^2)$ per test case for the predictive mean and variance, respectively. FITC can be viewed as a standard GP with a particular non-stationary covariance function parameterized by the pseudo-inputs. The mean and variance are given by:

$$\mu^d(\mathbf{z}) = \mathbf{k}_{\mathbf{zZ}_{ind}}^d \Theta \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}} \Lambda^{-1} y^d = \mathbf{k}_{\mathbf{zZ}_{ind}}^d i^d \tag{11a}$$

$$\Sigma^d(\mathbf{z}) = \mathbf{k}_{\mathbf{zZ}_{ind}}^d \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{Z}_{ind}}^{-1} \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}} - \mathbf{K}_{\mathbf{zZ}_{ind}} \Theta \mathbf{K}_{\mathbf{Z}_{ind}\mathbf{z}} \tag{11b}$$

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

III. Methods

A. LMPC for Autonomous Racing

Rosolia and Borrelli first introduced in [17] the adaptation of the core LMPC architecture to the autonomous racing problem. This is formulated as a *minimum time problem*, where an iteration j corresponds to a lap. Therefore, the stage cost is given as follows:

$$q(x_k, u_k) = \begin{cases} 1 & \text{if} \quad x_k \notin \mathcal{L} \\ 0 & \text{if} \quad x_k \in \mathcal{L} \end{cases} \tag{12}$$

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

The vehicle's dynamics are represented by the states and inputs vector quantities in Equation (13a) and Equation (13b), respectively.

$$\mathbf{x} = [s, e_y, e_\psi, v_x, v_y, r] \tag{13a}$$

$$\mathbf{u} = [a, \delta] \tag{13b}$$

where s is the distance along the track's centerline; e_y and e_ψ are the lateral distance and heading angle errors between the vehicle and the centerline, respectively. These quantities are given in the curvilinear abscissa reference frame, see Figure 1, also known as the Frenet reference frame. In particular, a given track is defined by the curvature $k(s)$ and maximum admissible lateral error $e_y^{max}(s)$ along the track's centerline. v_x and v_y are the longitudinal and lateral vehicle velocities, respectively, while r is the vehicle's yaw rate. The inputs are the longitudinal acceleration a and the steering angle δ.

[18] further extended this architecture by proposing a local LMPC that significantly reduced the computational burden by using a subset of the stored data. In particular, the local convex safe set $\mathcal{CS}_l^j$ is built around the candidate terminal state c_t using the N_p^{SS}-nearest neighbours from each of the previous

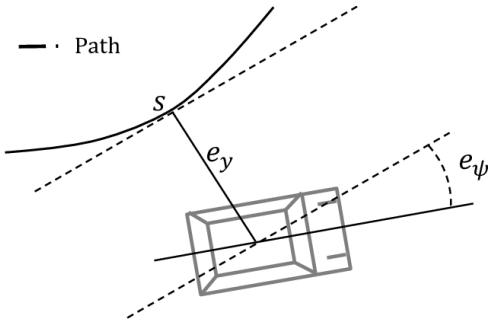

Fig. 1: Frenet Frame [reprinted from [17]]

N_l^{SS} laps. Notice that $N_l^{SS} = j - l$. These points are collected in the matrix $\mathbf{D}_l^j$, defined in Equation (14), which is updated at each time step. The candidate terminal state c_t is the estimated value for $\mathbf{x}_{t+N|t}$, calculated at time $t-1$. The approximation of the cost-to-go is computed using the costs associated with the selected states in $\mathbf{D}_l^j$.

$$\mathbf{D}_l^j = [\mathbf{x}_{t_1}^l, \cdots, \mathbf{x}_{t_{N_p^{SS}}}^l, \cdots, \mathbf{x}_{t_1}^j, \cdots, \mathbf{x}_{t_{N_p^{SS}}}^j] \tag{14}$$

The original LMPC architecture represents the vehicle's pose in the local coordinate frame, Equation (13a). However, we found that the discretization method used that assumes constant track curvature κ within a sampling period fails to properly describe complex tracks. In the FSG track, within a single meter, the curvature can change as much $0.14\,\mathrm{m}^{-1}$ which is almost half of the track's maximum curvature. At a control frequency of $10\,\mathrm{Hz}$ it takes a velocity of just $10\,\mathrm{m\,s}^{-1}$ to travel $1\,\mathrm{m}$. Thus, proving that this modelling strategy is unfit. The trajectory behaviour at a given corner becomes very sharp rather than smooth. I argue that the authors in [18], [19] might have not faced this issue because they have shown results for constant curvature corners.

Therefore, I have decided to change the vehicle model's pose states to global coordinates to fix this issue. s and e_y are still calculated because they provide immediate information regarding the track but they are not used to characterize the vehicle's pose dynamics. The longitudinal control input is $P \in [-1, 1]$ which represents a pedal setpoint. This corresponds to the normalized acceleration and brake pedal travel. This is the way the actual prototype is controlled both in simulation and reality. Thus, for our application, Equation (13) becomes:

$$\mathbf{x} = [x,\ y,\ \psi,\ v_x,\ v_y,\ r] \tag{15a}$$

$$\mathbf{u} = [P,\ \delta] \tag{15b}$$

where $\mathbf{x}$ is the vector of states that describe the vehicle's movement and $\mathbf{u}$ is the vector of control inputs.

The cost function has five main parts. First, the derivative terms apply a penalty on the squared change of a given quantity between consecutive steps along the prediction horizon, both for dynamic state and inputs. This enables one to control how aggressively the controller behaves and obtain smooth trajectories. A quadratic cost is applied on v_y to act as a regularization cost which forces the vehicle into its stable domain and helps convergence. A quadratic cost is applied on the lag error which measures the accuracy of the global to local coordinate transformation [11].

The fourth part concerns the soft constraints on the states. Bounds on the states should not be implemented as hard constraints since one cannot exclude that the real system moves outside the constraint range due to, for instance, model mismatch which would render the problem infeasible [4]. Thus, the bound on a given state $x \leq x_{max}$ can be approximated by $x \leq x_{max} + \epsilon$ where $\epsilon \geq 0$ and a term $l(\epsilon)$ is added to the cost functional. It can be shown that $l(\epsilon) = u\epsilon + v\epsilon^2$ with a sufficiently high u and $v > 0$ ensures that no constraint violation occurs provided there exists a feasible input. The bounded states are v_x and e_y. e_y needs to be bounded to stay within the track width. Finally, a longitudinal and lateral velocity ellipse is implemented to ensure the vehicle remains within its physical limits.

Finally, the last two terms constitute the penalty on the terminal components of the MPC. A linear cost is applied to the product of α_i - the coefficients of the local safe set's $\mathbf{D}_l^j$ convex hull - and $\mathbf{Q}^j$ - the cost-to-go of each point in the safe set (Equation (6)). Moreover, a quadratic penalty is applied to the error between the terminal state and the safe set points $\mathbf{D}_l^j$, *i.e.* $\mathbf{x}_{t+N|t} - \alpha_i \mathbf{D}_i^j(c_t)$, for each point i in the safe set. While this slack term ensures the terminal state lies within the convex hull, the terminal cost favours points in the safe set that resulted in faster laps.

Bounds on both inputs and their rate are applied. These bounds may be more restrictive than the actual physical limits imposed by the robotic platform. The pedal setpoint has no relevant rate physical limit but imposing this bound ensures wheel slippage is avoided in case the derivative cost did not achieve this already. The steering setpoint has a maximum angular velocity constraint which results from the servomotor limits.

$\mathbf{u}_t^j$ is the control actuation that is to be applied at the current sampling time. It corresponds to the previous sampling time solution shifted by one step, Equation (16). This delay applied to the system aims to account for solver processing time - which is significant in real-time and may be inconsistent across the experiment - and to keep a constant node rate.

$$\mathbf{u}_{t|t} = \mathbf{u}_t^j = \mathbf{u}_{t|t-1}^{*,j} \tag{16}$$

1) Vehicle Model: The system dynamics in Equation (1b) are given in its continuous-time form by Equation (17) which is the sum of an *a priori physics-based model* f_t and a term to model the *unknown dynamics* g_t, *i.e.* those not represented by f_t.

$$\mathbf{x}_{t+1} = h_t(\mathbf{x}_t, \mathbf{u}_t) = f_t(\mathbf{x}_t, \mathbf{u}_t) + g_t(\mathbf{x}_t, \mathbf{u}_t) \tag{17}$$

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

$$\dot{x} = v_x \cos(\psi) - v_y \sin(\psi) \tag{18a}$$

$$\dot{y} = v_x \sin(\psi) + v_y \cos(\psi) \tag{18b}$$

$$\dot{\psi} = r \tag{18c}$$

The dynamic part of the vehicle model, *i.e.* related to Newton's first law, has been modelled with a dynamic bicycle model - Figure 2. This model is frequently used in automotive control algorithms [21] and is given as follows:

$$\begin{bmatrix} \dot{v}_x \\ \dot{v}_y \\ \dot{r} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}(F_x - F_{F,y}\sin\delta + m v_y r) \\ \frac{1}{m}(F_{R,y} + F_{F,y}\cos\delta - m v_x r) \\ \frac{1}{I_z}(F_{F,y} l_F \cos\delta - F_{R,y} l_R) \end{bmatrix} \tag{19}$$

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

$$F_{a,y} = -2D_a \sin\left(C_a \arctan(B_a \alpha_a)\right) \tag{20}$$

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

$$\alpha_F = \arctan\left(\frac{v_y + l_F r}{v_x}\right) - \delta \tag{21a}$$

$$\alpha_R = \arctan\left(\frac{v_y - l_R r}{v_x}\right) \tag{21b}$$

The longitudinal force F_x is given as follows:

$$F_x = 2 \cdot \frac{T_{max} \cdot GR \cdot P}{r_{wheel}} - C_{roll} \cdot m \cdot g + \frac{1}{2}\rho \cdot C_d \cdot A_f \cdot v_x^2 \tag{22}$$

where T_{max} is the maximum available torque at each of the two rear axle in-wheel motors whose maximum practical value is 21 Nm but a smaller value may be used for safety reasons. GR is the transmission's gear ratio and C_{roll} is the roll resistance factor. Concerning the aerodynamic drag force, ρ is the air density, A_f is the vehicle's frontal area used as a reference for the force calculation and C_d is the drag coefficient.

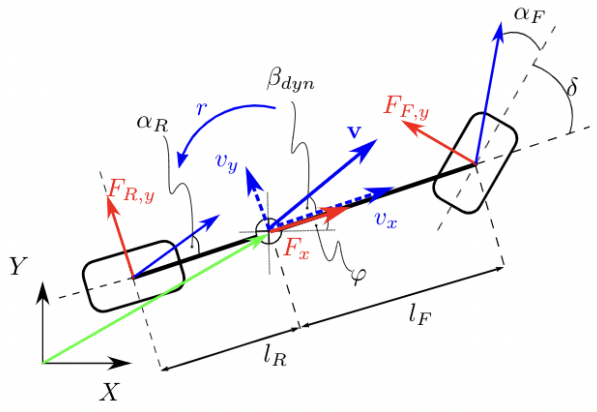

Fig. 2: Dynamic Bicycle Model: position vectors are in depicted in green, velocities in blue, and forces in red. [reprinted from [20]]

B. Gaussian Processes Regression

GPR is used to predict the error between the vehicle model - Section III-A1 - and the available measurements, *i.e.* estimate g_t in Equation (17). One assumes that the modelling error only affects the dynamic part of the first-principle model, *i.e.* the velocity states. Therefore, the training outputs are given by the difference between the measurement $\mathbf{x}_{k+1}$ and the nominal model predictions:

$$\mathbf{y}_k = \mathbf{B}_d^{\dagger}\big(\mathbf{x}_{k+1} - f(\mathbf{x}_k, \mathbf{u}_k)\big) = \mathbf{g}(\mathbf{z}_k) + \mathbf{w}_k \tag{23}$$

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

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

The covariance function used is the squared-exponential kernel with the independent measurement noise component - $\sigma_{n,d}^2 \delta_{\mathbf{z}\bar{\mathbf{z}}}$:

$$k_{SE}^d(\mathbf{z}, \bar{\mathbf{z}}) = \sigma_{f,d}^2 \exp\left(-\frac{1}{2}\frac{(\mathbf{z} - \bar{\mathbf{z}})^{\mathrm{T}}(\mathbf{z} - \bar{\mathbf{z}})}{l_d^2}\right) + \sigma_{n,d}^2 \delta_{\mathbf{z}\bar{\mathbf{z}}} \tag{24}$$

Two sparse approximations have been explored: *i)* SoD and *ii)* FITC. The first approximation method is essentially a full GP that does not use all data available. This means the computations for the error prediction in Equation (17) are those of an exact GP given by Equation (10a). While for the second approach the computations are those of Equation (11a).

Finally, it should be noted that some quantities can be precomputed which otherwise could prevent real-time feasibility. Particularly, $\left(\mathbf{K}_{\mathbf{ZZ}}^d + \mathbf{I}\sigma_d^2\right)^{-1}\mathbf{y}^d$ in Equation (10a) only needs to be recomputed whenever the training data $\mathcal{D}$ is changed - this corresponds to `training`. In the SoD offline fashion, it is only computed once before the controller is launched. `Inference` corresponds to the rest of the computation of Equation (10a).

In the FITC approximation, the `training` part corresponds to the determination of the information vector which only requires recomputation when either the training dataset $\mathcal{D}$ or the inducing points Z_{ind} are changed. In our application, however, the inducing points are updated at each sampling time. They are equally distributed along the last sampling time's shift predicted trajectory. This is a sensible placement of the inducing points since the new test cases are expected to

be proximate as the trajectory does not change significantly at consecutive sampling times. This means the online adaptation of the dictionary $\mathcal{D}$ is immediately possible.

GPR is generally susceptible to outliers, which can hinder the model error learning performance [10]. Moreover, large and sudden changes in the GP predictions can lead to erratic driving behaviour. To attenuate these effects, one only includes datapoints in the dictionary $\mathcal{D}$ (both online and offline) whose measurements fall within predefined bounds $\pm\mathbf{y}_{lim}$, defined from physical considerations and empirical knowledge. This bound is also enforced to the GPR predictions.

C. Control Architecture

There are two variations of the LMPC architecture given by Algorithm 1. They differ only in Line 9. In the pre-computed version, the model error predictions are fed directly to the solver based on the previous sampling time's shifted trajectory. While in the solver-embedded version the model error predictions are calculated at each solver's iteration and only the information vector $\mathbf{i}^d$ is pre-computed.

Algorithm 1: LMPC Architecture

```
1  Initializations;
2  while Event not finished do
3  |  Update car state from SLAM data;
4  |  Publish control command;
5  |  Convert to local coordinates;
6  |  if Lap finished then
7  |  |  Add closed-loop data to safe set;
8  |  if Model learning active then
9  |  |  Compute GPR;
10 |  Find local safe set;
11 |  Solve nonlinear optimization;
12 |  Store measurement data;
13 |  if Online learning active then
14 |  |  Update dictionary
```

IV. Implementation

A. Simulation Platform

FSSIM[1] is the vehicle simulator used to test the controllers. AMZ Driverless developed this vehicle simulator dedicated to the FSD competition and released it open-source to other teams. This team reported 1% lap-time accuracy compared with their FSG 2018 trackdrive run [20]. FSSIM comes with the standard Acceleration and Skidpad competition tracks. Additionally, it includes track layout data mapped from the 2018 official FS events of Italy and Germany.

Due to real-time requirements, this simulator does not simulate raw sensor data, *e.g.* camera or LiDAR data. Instead, cone observations around the vehicle are simulated using a given cone-sensor model, that yields different probabilities of detecting a cone and correctly identifying its class based on the distance from the vehicle.

FSSIM simulates the vehicle dynamics using a model that blends a dynamic and a kinematic bicycle model [20]. The dynamic bicycle model is ill-defined for slow velocities due to the tire slip angles considered to compute the tire forces. In a racing application, most of the track is spent at high-speeds where this model accurately represents the vehicle behaviour. At low speeds, *e.g.* at race start or in sharp corners, the kinematic bicycle model constitutes a faithful description of the vehicle dynamics.

B. Controller Implementation

We have started the development of this thesis by testing some of the available open-source LMPC packages developed by the MPC Lab researchers. In the implementations in Python[2] [18] and Julia[3] [19], it did not seem possible to increase the node's frequency from 10 to 20 Hz nor increase the prediction horizon to more than $N = 12$ which results in a lookahead time of just 1.2 s. From experience, the combination of these two factors is clearly insufficient for an autonomous racing application.

Hence, I decided to develop a custom C++ implementation of the LMPC architecture using FORCESPRO [22], [23] - a solver designed for embedded solving of MPCs - to solve the optimization problem. With these changes, the controller is able to run at 20 Hz with $N > 20$. The controller - Algorithm 1 - is implemented in ROS/C++.

The coordinate conversion (Line 5) to the Frenet frame resorts to a library[4] [11] that fits the track centerline with splines. This library enables finding the track progress s and lateral deviation e_y from the centerline given a location (x, y) and conversely.

The computations associated with GPR model error prediction described in Section III-B resort to the C++ open-source `albatross`[5] library developed by Swift Navigation.

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

The hyperparameter estimation procedure resorts to the Matlab function `fitrgp`[6]. For a particular kernel function and sparse approximation, given the original dataset of size n, this function outputs the hyperparameters and the active set of size m that maximize the marginal logarithmic likelihood. The active set choice is of particular relevance so as to maximize the information provided by the subset of the original dataset. We use the differential-entropy based selection method. We use the active set of the SoD approximation as the training set for the FITC approximation, *i.e.* $n_{FITC} = m_{SoD}$.

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

[2] https://github.com/MPC-Berkeley/barc/tree/devel-ugo

[3] https://github.com/MPC-Berkeley/barc/tree/LMPC-Shuqi

[4] https://github.com/alexliniger/MPCC/

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

[6] https://www.mathworks.com/help/stats/fitrgp.html

V. Simulation Results

In Section V-A, we show results for both sparse approximations by varying the size m of their latent variables. There, we aim to study the influence of these parameters in the ability to predict the nominal model error. We test both sparse models in an offline and online learning fashion. In the former, the training dataset is not updated online with current measurements while in the latter it is. We also analyse how the computational cost of GPR prediction evolves for these strategies. The results shown there correspond to the average over 10 laps.

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

In Section V-B, we use the best performing models found in Section V-A to evaluate the complete LMPC architecture. We evaluate its performance with the lap times over the 10-lap trackdrive event. We show results in the FSG track and change the controller parameters to achieve a more aggressive controller. Finally, we also compare the FITC pre-computed and embedded versions of the architecture - Algorithm 1.

A. Model Learning

For the Subset of Data approximation, we use the tuning scheme described in Section IV-B to find the active set of different sizes m_{SoD} for each model from a dictionary of $n_{SoD} = 43329$.

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

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

TABLE I: SoD Active Set Size Analysis

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

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

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

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

Although in a comparable order of magnitude, the offline version exhibits better model error fitting ability. I once again argue that this is due to the dictionary update procedure. Therefore, we have increased the training dataset to $n_{FITC} = 400$. The best performing model with $m_{FITC} = 10$ yields model learning performance comparable to that of the SoD

approximation. The corrected dynamics average error is 0.12 slightly above of the SoD benchmark value of 0.09.

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

TABLE II: FITC Inducing Points Strategy Analysis

B. Learning-based Model Predictive Control

In Table III, we show the lap times along the 10-lap trackdrive event and average model error for the FSG track. The initial safe set was collected using the nominal controllers. It is composed of four laps with lap times of around 28.8 s. The controllers herein have a prediction horizon of $N = 20$ which corresponds to a look-ahead time of 1 s, until stated otherwise. The LMPC results without model learning prove the iterative improvement character of the architecture. The first lap is immediately 33% faster compared to the path-following controller. Equivalently, the last lap is 39% faster. Furthermore, the last lap corresponds to a 10% improvement compared to the first LMPC lap.

Figure 3 shows the FSG trackdrive trajectories. The finish line is at the origin and the vehicle runs clockwise. It can be seen that the LMPC exploits the track layout to improve performance measured by lap time. Nevertheless, the third column of Table III exhibits severe model mismatch which causes the vehicle to break the track constraint. See, for instance, the exit of the hairpin or the first corner where the trajectory is on top of the track boundary which entails a cone was hit since the trajectory corresponds to the centre of mass. The car starts at the origin. The hairpin is the sharp corner on the rectangle region given by the top left corner of $(10, -65)$ and the bottom right corner of $(30, -75)$. Furthermore, the approach to the slalom segment is not optimal per empirical vehicle dynamics standards. The vehicle is braking too late which leads to a slower slalom with greater steering actuation required. The slalom segment is given by the corners $(-10, -15)$ and $(0, -45)$. In this case, I reckon it is due to a combination of a relatively short prediction horizon and model mismatch.

Let one now analyse the performance of the LMPC when the GP model learning scheme is deployed. Table III shows that the last lap is 42 and 12% faster when compared to the path-following lap and the first lap, respectively. These results correspond to the online SoD model with $m_{SoD} = 200$. Figure 3 shows the corresponding FSG trackdrive trajectories. It is clear the reduced model mismatch prevents the vehicle from disrespecting the track constraint. However, the slalom trajectory does not yet look optimal.

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

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

TABLE III: LMPC Lap Times and Model Error

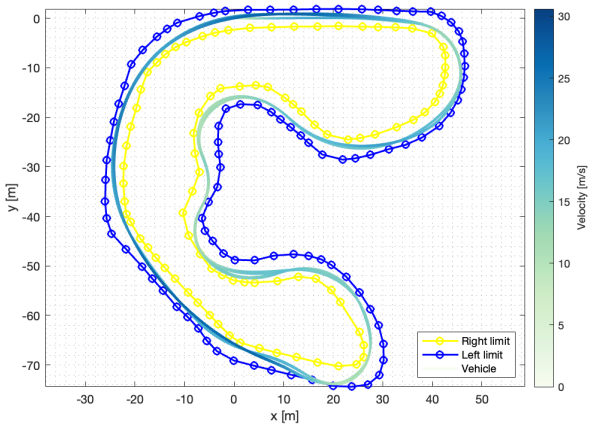

Fig. 3: FSG Trackdrive Trajectory of LMPC without Model Learning

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

TABLE IV: LMPC Lap Times and Model Error - FITC Approximation

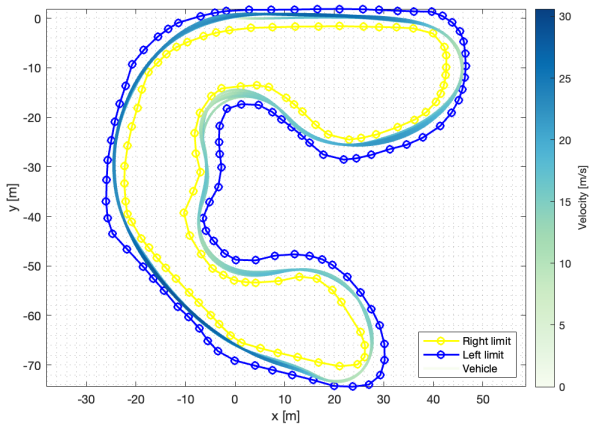

Fig. 4: FSG Trackdrive Trajectory of LMPC with Model Learning [$N = 20$]

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

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

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

TABLE V: LMPC Lap Times and Model Error [$N = 30$]

Both models used the offline version of the SoD approximation with $m_{SoD} = 600$. Table V further corroborates these claims. For instance, the nominal model average error on the first lap of the default controller with $N = 30$ of 0.27 is substantially higher than those with $N = 20$ which is on average 0.21. The increased driving behaviour aggressiveness is verified by the larger nominal model errors of around 0.37. The model learning scheme is able to significantly reduce the model mismatch to enable safe aggressive racing. For the first case, the corrected dynamics model average error is around 0.08 which corresponds to a reduction of about 70%. While for the aggressive controller, the final model mismatch is on average 0.15, a reduction of 60%. The model learning scheme is able to keep an acceptable value of corrected dynamics model mismatch even in the case of offline model learning on the aggressive controller. It should be noted that such high nominal model errors data were not present in the original training dataset.

VI. Conclusions & Future Work

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

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

In order to overcome this issue, we use Gaussian Processes Regression to predict the nominal model error. The GPs are able to successfully model this error such that the corrected dynamics exhibit reasonably low model errors - model error reduction of at least 65% and by as much as 75%. This error remains about constant as the LMPC strives for faster laps in which the nominal model error increases to prohibitively high errors.

We have shown how the model error fitting ability changes with varying parameters for the sparse GP approximations. We have concluded that it is more important to enable online learning, *i.e.* adaptation of the training dataset with data collected on a given run. In the SoD approximation, this comes with a considerable computational cost burden due to training which only allows relatively small active sets whilst still abiding by the real-time requirement. With the naïve dictionary update technique implemented, which stores the most recent datapoints, the small active set size does not yield significant prediction improvements. I argue that a better

selection procedure should improve results although the active set size would still be rather short.

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

Subsequently, we have demonstrated that the full architecture, *i.e.* LMPC with model learning, outperforms the one without model learning in the metrics considered.

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

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

The greatest focus of the academia that studies learning model-based controllers has been on reducing model mismatch. First, one should start by testing the Automatic Relevance Determination (ARD) kernel. It gets its name from having individual length-scales which enables identifying irrelevant inputs - those with a very large length-scale where the covariance function becomes effectively independent of that input. Other ML techniques should also be considered such as the use of Bayesian Linear Regression or Neural Networks. Finally, one has already claimed that a dictionary data selection criterion that considers the information each data point conveys should improve prediction accuracy considerably.

In this LMPC architecture, the Q-function could be improved. The minimum-time stage cost should be augmented such that points that yield long-term benefits are more favoured. The state uncertainty measurement which is inherent with the use of Gaussian Processes should be considered in a Robust Learning-based Model Predictive Control framework. Finally, learning of other MPC parameters should also be targeted. For instance, one could explore methods to automatically adjust the MPCs parameters using some kind of reward function exploited by a Reinforcement Learning algorithm.

REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

[14] J. Quiñonero-Candela, C. Rasmussen, and C. Williams. Approximation methods for Gaussian process regression. 4 2007.

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

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

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

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

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

[20] J. Kabzan, M. Valls, V. Reijgwart, H. Hendrikx, C. Ehmke, M. Prajapat, A. B uhler, N. Gosala,M. Gupta, R. Sivanesan, A. Dhall, E. Chisari, N. Karnchanachari, S. Brits, M. Dangel, I. Sa,R. Dub e, A. Gawel, M. Pfeiffer, A. Liniger, J. Lygeros, and R. Siegwart. Amz driverless: The fulla utonomous racing system. Journal of Field Robotics, 2020.

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

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

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