GP-based Model Predictive Control*

Dimitrios Gkoutzos[1], Luzia Knödler[2] and Lucas Rath[3]

***Abstract*— The performance of Model Predictive Control (MPC) depends highly on how well the model captures the dynamics of the plant. But the identification of such models can be challenging due to complex dynamics and unknown or changing parameters. Therefore, an approach is to use a simple and fixed nominal model and learn the unknown deviation between the nominal model and the true plant dynamics. The deviation/disturbance can be learned using Gaussian Process (GP) Regression. In this report an introduction to learning-based MPC using GPs is given as well as an illustrative and an application-oriented example are discussed. The implementation of the latter has shown that hyper-parameter optimization (GP training) plays an important role in the final controller performance. An efficient MPC formulation and an efficient GP Regression implementation were realized to improve the computational performance. For both examples a significant improvement of the performance was achieved by learning the unmodeled dynamics with GP Regression and using this knowledge in the MPC predictions.**

I. INTRODUCTION

Model predictive control (MPC) is a popular control strategy which uses a model of the plant dynamics to obtain the control input that optimizes future reactions of the plant [4]. The performance of MPC depends highly on how well the model captures the dynamics of the plant [3]. But the identification of such an *a priori* model can be challenging due to complex dynamics and unknown parameters and the dynamics of the plant could also change during the application [3], [7]. Moreover, a complex model can lead to computational intractability [3]. Therefore, a simple and fixed nominal model of the plant can be used in combination with a learned disturbance model, which captures the unmodeled dynamics of the plant. The disturbance model represents the error between the observed behaviour of the plant and the behaviour of the nominal model [7]. It can be modelled using Gaussian Process (GP) Model which is a probabilistic, non-parametric model [4]. GPs have the advantage of characterizing the prediction uncertainties additional to the prediction mean [4]. GP Regression can also be used to model the full dynamics of the plant and not only the model error. This approach was applied to a cart pole swing-up environment and an autonomous racing task in [10]. To reduce high computational costs sparse spectrum GPs are chosen. Kocijan et al. [4] use an offline-identified GP model instead. Another alternative is the generation of local GPs (LGPs) where for each subspace of the GP input space different GPs are identified. While Nguyen-Tuong et al. [6] and Meier et al. [5] identify many LGPs, Ostafew et al. [7] compute one single LGP based on data within a sliding window.

In this report we present the results of our project within the course "Statistical Learning and Stochastic Control". First, the notation is defined. Then, introductions to the theory of GPs as well as MPC are given in section III. In section IV GP Regression and MPC are combined to GP-based MPC. Additionally, important aspects for the successful implementation of GP-based MPC are described in section IV-B. Then, two examples are introduced in section V and the results are discussed in section VI. The first of the two examples is illustrative and is used to explain and clarify the theory (See section V-A). It is a cart pole system which is later also referred to as inverted pendulum. The second example is application-oriented and more complex (See section V-B). It treats an autonomous racing problem using a single track model. For the implementation of this example the procedure introduced in Kabzan et al. [3] was followed. Finally, our conclusions are presented in section VII.

II. NOTATION

Bold lowercase letters are used for vectors $\boldsymbol{x} \in \mathbb{R}^n$ and bold uppercase letters for matrices $\boldsymbol{X} \in \mathbb{R}^{n \times m}$. The j-th column of a matrix $\boldsymbol{X}$ is denoted by $[\boldsymbol{X}]_{.,j}$ and the element in the i-th row and j-th column is $[\boldsymbol{X}]_{ij}$. A diagonal matrix $\boldsymbol{X}$ with the diagonal elements $x_{11}, x_{22}, \cdots, x_{nn}$ is represented by $\text{diag}(x_{11}, x_{22}, \cdots, x_{nn})$. The notation $\text{blkdiag}(x_{11}, x_{22}, \cdots, x_{nn})$ represents a block diagonal matrix with the elements $x_{11}, x_{22}, \cdots, x_{nn}$. If $w \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ is given, w is normal distributed with mean $\boldsymbol{\mu}$ and variance $\boldsymbol{\Sigma}$.

III. BACKGROUND

A. Gaussian Process (GP) Regression

The main goal of the GP regression is to identify an unknown function $d_{true} : \mathbb{R}^{n_z} \to \mathbb{R}$:

$$y = d_{true}(\boldsymbol{z_k}) + w$$

where the process noise $w \sim \mathcal{N}(0, \sigma_n)$ is a i.i.d. Gaussian noise and $\boldsymbol{z_k} \in \mathbb{R}^{n_z}$ are relevant features to be used in the regression.

*Project within the course Statistical Learning and Stochastic Control, University of Stuttgart, 4. Februar 2020.

[1]Dimitrios Gkoutzos is a student of the Master study programm Engineering Cybernetics, University of Stuttgart, dimitrios.gk@gmx.de

[2]Luzia Knödler is a student of the Master study program Engineering Cybernetics, University of Stuttgart, luzia.knoedler@web.de

[3]Lucas Rath is a student of the Master study program Engineering Cybernetics, University of Stuttgart, and of the Master study program Systems, Control and Mechatronics, Chalmers University of Technology, lucasrm25@gmail.com

We model the prior $\boldsymbol{d_{true}}$ as a Gaussian Process

$$\boldsymbol{d_{true}}|\boldsymbol{z} \sim \mathcal{GP}(m(\boldsymbol{z}), k(\boldsymbol{z}, \boldsymbol{z'}))$$

where $m(\boldsymbol{z})$ and $k(\boldsymbol{z}, \boldsymbol{z'})$ are any valid functions that evaluate the mean and covariance of d_{true}.

Given the GP prior and a training data dictionary $\mathcal{D} = \{(\boldsymbol{z_i}, \boldsymbol{y_i})|i = 1, \ldots, m\}$ with m observation pairs $(\boldsymbol{z_i}, \boldsymbol{y_i})$ such that

$$\boldsymbol{Z} = [\boldsymbol{z_1}, \ldots, \boldsymbol{z_m}] \in \mathbb{R}^{n_z \times m},$$
$$\boldsymbol{Y} = [\boldsymbol{y_1}, \ldots, \boldsymbol{y_m}] \in \mathbb{R}^{n_d \times 1},$$

the posterior distribution of the unknown function evaluated at a test point $\boldsymbol{z}$ is also Gaussian [11] and given by:

$$d_{GP}(\boldsymbol{z}) := d_{true}|\boldsymbol{z}, \boldsymbol{Z}, \boldsymbol{Y} \sim \mathcal{N}(\mu^d(\boldsymbol{z}), \Sigma^d(\boldsymbol{z}))$$

where

$$\mu^d(\boldsymbol{z}) = m(\boldsymbol{z}) + K(\boldsymbol{z}, \boldsymbol{Z})(K(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I})^{-1}(\boldsymbol{Y} - m(\boldsymbol{Z}))$$
$$\Sigma^d(\boldsymbol{z}) = K(\boldsymbol{z}, \boldsymbol{z}) - K(\boldsymbol{z}, \boldsymbol{Z})(K(\boldsymbol{Z}, \boldsymbol{Z}) + \sigma_n^2 \boldsymbol{I})^{-1} K(\boldsymbol{Z}, \boldsymbol{z}).$$

with $[K(\boldsymbol{Z}, \boldsymbol{Z'})]_{ij} = k(\boldsymbol{z_i}, \boldsymbol{z'_j})$.

Here, we make use of the zero mean function $m(z) = 0$ and *Square Exponential Kernel* (SE) given by:

$$k(\boldsymbol{z}, \boldsymbol{z'}) = \sigma_f^2 exp(-0.5(\boldsymbol{z} - \boldsymbol{z'})^{\mathsf{T}} M^{-1}(\boldsymbol{z} - \boldsymbol{z'}))$$

where $\sigma_f^2 \in \mathrm{R}^{1 \times n_z}$ and $M \in \mathrm{R}^{n_z \times n_z}$ are the squared output variance and the length-scale covariance matrix, respectively. We choose to parameterize the length scale as diagonal positive semi-definite resulting in

$$\boldsymbol{M} = \begin{bmatrix} l_1 & 0 & \cdots & 0 \\ 0 & l_2 & & \vdots \\ \vdots & & \ddots & 0 \\ 0 & \cdots & 0 & l_{n_z} \end{bmatrix},$$

with $l_i \geq 0$, $\forall i \in 1, \cdots, n_z$. Thus, the free parameters of the kernel, the so called hyper-parameters, are given by

$$\theta = [l_1, \ldots, l_{n_z}, \sigma_f^2, \sigma_n^2].$$

The evaluation noise variance $\sigma_n^2 \in \mathbb{R}$ is not part of the kernel, but still a free parameter. How the hyper-parameters were chosen, including the usage of hyper-parameter optimization, is described in section IV-B.

So far, we have described GP regression for one dimensional outputs. Generally, one strategy to model GP with multivariate outputs is to treat each output dimension as an independent GP. Note that the hyper-parameters do not have to be equal and might be optimized independently. Therefore, combining the GPs of each dimension results in the multivariate GP approximation given by

$$\boldsymbol{d_{\mathcal{GP}}}(\boldsymbol{z}) \sim \mathcal{N}(\boldsymbol{\mu^d}(\boldsymbol{z}), \boldsymbol{\Sigma^d}(\boldsymbol{z})),$$

where the mean $\boldsymbol{\mu^d}(\boldsymbol{z}) \in \mathbb{R}^{n_d}$ and the variance $\boldsymbol{\Sigma^d}(\boldsymbol{z}) \in \mathbb{R}^{n_d \times n_d}$ are given by

$$\boldsymbol{\mu^d}(\boldsymbol{z}) = [\mu^{d,1}(\boldsymbol{z}), \cdots, \mu^{d,n_d}(\boldsymbol{z})]^T,$$
$$\boldsymbol{\Sigma^d}(\boldsymbol{z}) = \mathrm{diag}([\Sigma^1(\boldsymbol{z}), \cdots, \Sigma^{n_d}(\boldsymbol{z})]).$$

B. Standard MPC Formulation

Model predictive control (MPC), receding horizon control or moving horizon control are all names for a control strategy which predicts the future dynamic behaviour within a finite prediction horizon N and chooses the control input such that a performance functional is minimized [1]. Since the predicted behaviour is not equal to the system behaviour, due to disturbances and model-plant mismatch, only the first input of the computed control input sequence $\boldsymbol{u^*_{0:N-1}}$ is applied [1]. The discrete-time finite-horizon control problem of standard Nonlinear MPC (NMPC) with terminal cost is given by

$$\begin{aligned} \boldsymbol{x^*_{0:N}}, \boldsymbol{u^*_{0:N-1}} = \underset{\boldsymbol{x_{0:N}}, \boldsymbol{u_{0:N-1}}}{\arg\min} \quad & \sum_{k=0}^{N-1} f_o(\boldsymbol{x_k}, \boldsymbol{u_k}) + \phi(\boldsymbol{x_N}) \\ s.t. \quad & \boldsymbol{x_{k+1}} = f_{nom}(\boldsymbol{x_k}, \boldsymbol{u_k}) \\ & \boldsymbol{x_0} = \bar{\boldsymbol{x}} \\ & \boldsymbol{x_k} \in \mathcal{X}(\boldsymbol{x_k}) \\ & \boldsymbol{x_N} \in \mathcal{X}_f \\ & \boldsymbol{u_k} \in \mathcal{U}(\boldsymbol{x_k}), \end{aligned} \tag{1}$$

where f_{nom} refers to the discrete-time dynamics of the nominal model of the plant. The nominal dynamics are a simplified version of the true dynamics $\boldsymbol{x_{k+1}} = f_{true}(\boldsymbol{x_k}, \boldsymbol{u_k})$. The input and state constraint sets $\mathcal{U}$, $\mathcal{X}$ and $\mathcal{X}_f$ can for example be defined as box constraints [1]

$$\mathcal{U} = \{u \in \mathbb{R}^p | \boldsymbol{u_{min}} \leq \boldsymbol{u} \leq \boldsymbol{u_{max}}\},$$
$$\mathcal{X} = \{x \in \mathbb{R}^n | \boldsymbol{x_{min}} \leq \boldsymbol{x} \leq \boldsymbol{x_{max}}\}.$$

The cost function f_o and the final cost $\phi(x_N)$ are defined according to the considered control problem and are further described for each example in Section V.
When using MPC each sampling instance the following steps are repeated:

1) Measure (estimate) current state $\bar{x}$
2) Solve the open-loop discrete-time finite-horizon optimal control problem (1)
3) Apply the first portion of the optimal control sequence $\boldsymbol{u_{NMPC}} = \boldsymbol{u_0^*}$

IV. GP-BASED MPC

A. Framework

In this section the framework for GP-based MPC is introduced.

As stated before, unmodeled dynamics might harm the performance of the MPC, since predictions might not represent the true dynamics of the system. To this end, a Gaussian Process can be used to learn unmodeled dynamics and therefore enhance predictions of the MPC. The new learning-based prediction can be formulated as:

$$\boldsymbol{x_{k+1}} = f_{est}(\boldsymbol{x_k}, \boldsymbol{u_k}) = f_{nom}(\boldsymbol{x_k}, \boldsymbol{u_k}) + \boldsymbol{B_d}(\boldsymbol{d_{\mathcal{GP}}}(\boldsymbol{z_k}) + \boldsymbol{w}), \tag{2}$$

where $\boldsymbol{z_k} = [\boldsymbol{B_{z_x}} \boldsymbol{x_k}; \boldsymbol{B_{z_u}} \boldsymbol{u_k}]$ are the selected entries of the state $\boldsymbol{x_k}$ and the input $\boldsymbol{u_k}$ which are assumed to affect the model error. On the other side, the matrix $\boldsymbol{B_d}$ picks the states of $\boldsymbol{x_{k+1}}$ which are affected by the model error. By introducing the matrices $\boldsymbol{B_d} \in \mathbb{R}^{n \times n_d}$, $\boldsymbol{B_{z_x}} \in \mathbb{R}^{n_{z_x} \times n}$ and $\boldsymbol{B_{z_u}} \in \mathbb{R}^{n_{z_x} \times p}$ the dimensionality of the learning problem can be reduced. Below we refer to $\boldsymbol{z_k} \in \mathbb{R}^{n_z}$ as regression features. The process noise $\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{\Sigma^w}) = \mathcal{N}(0, I\sigma_n^2)$.

This allows us to define a Learning-Based MPC scheme as follows:

$$\begin{aligned} \boldsymbol{\mu^{*x}_{0:N}}, \boldsymbol{u^*_{0:N-1}} = \underset{\boldsymbol{\mu^x_{0:N}}, \boldsymbol{u_{0:N-1}}}{\arg\min} & \sum_{k=1}^{N-1} f_o(\boldsymbol{\mu^x_k}, \boldsymbol{u_k}) + \phi(\boldsymbol{\mu^x_N}) \\ s.t. \quad & \boldsymbol{\mu^x_0} = \bar{\boldsymbol{x}}, \quad \boldsymbol{\Sigma^x_0} = 0 \\ & \boldsymbol{\mu^x_{k+1}} = f_{nom}(\boldsymbol{\mu^x_k}, \boldsymbol{u_k}) + \boldsymbol{B_d} \boldsymbol{\mu^d_k} \\ & \boldsymbol{\mu^x_k} \in \mathcal{X}, \\ & \boldsymbol{u_k} \in \mathcal{U}, \end{aligned} \tag{3}$$

Note that due to the Gaussian Process $d_{\mathcal{GP}}$, the predictions are now random variables. It turns out, however, that evaluation of (2) is intractable. One quick solution is to use the Extended Kalman Filter approach to first linearize (2) w.r.t. the random variables

$$\begin{aligned} & f_{est}(x_k, u_k, d_{GP}, w) = f_{nom}(x_k, u_k) + B_d(d_{\mathcal{GP}}(z_k) + w) \\ & \approx f_{est}(\mu^x_k, u_k, \mu^d(\mu^x_k), \mu^w) \\ & \quad + \left(\nabla_{\{x_k, d_{GP}, w\}} f_{est}(\mu^x_k, u_k, \mu^d(\mu^x_k), \mu_w)\right)^\mathsf{T} \begin{bmatrix} x_k - \mu^x_k \\ d_{\mathcal{GP}} - \mu^d_k \\ w - \mu^w \end{bmatrix} \end{aligned}$$

the result of the linearized transformation is Gaussian distributed, with the following mean $E[x_{k+1}] := \mu^x_{k+1}$ and the variance $\mathbb{V}ar[x_{k+1}] := \Sigma^x_{k+1}$

$$\mu^x_{k+1} \approx f_{nom}(\mu^x_k, u_k) + B_d\, \mu^d(\mu^x_k, u_k)$$

$$\Sigma^x_{k+1} \approx \begin{bmatrix} \nabla_{x_k} f_{nom}(\mu^x_k, u_k) \\ B_d \\ B_d \end{bmatrix}^\mathsf{T} \mathbb{V}ar \begin{bmatrix} x_k \\ d_{\mathcal{GP}} \\ w \end{bmatrix} \begin{bmatrix} \nabla_{x_k} f_{nom}(\mu^x_k, u_k) \\ B_d \\ B_d \end{bmatrix}$$

Note that, as in [2], we assumed that $d_{\mathcal{GP}}(x_k)$ and x_k are independent. Thus, we consider

$$\mathbb{V}ar[x_k, d, w] = blkdiag(\Sigma^x_k, \Sigma^d_k(\mu^x_k), \sigma_n^2 I)$$

Although this assumption might eventually deteriorate the prediction quality, this is a simple and computationally cheap approach, allowing a tractable MPC formulation.

For sake of simplicity, in IV, we also assume that at each time-step the measurement is perfect and therefore $\bar{x}$ is deterministic. For this reason $\boldsymbol{\mu^x_0} = \bar{\boldsymbol{x}}, \boldsymbol{\Sigma^x_0} = 0$.

An overview on all components of GP-based MPC is given in Figure 1.

B. Important aspects for the implementation

In this section important aspects for the successful implementation of our considered examples are described. We start by introducing different aspects for an efficient MPC implementation. Next, different GP enhancements are explained, which are beyond what was considered in the lecture. These enhancements include an efficient GP Regression implementation, a sparse realization of collecting training data and hyper-parameter optimization.

The MPC formulation can be made more efficient by removing constraints. Input constraints given by:

$$u_k \in \mathcal{U} = \{u : g_u \leq \lambda_u\}$$

are usually considered hard-constraints and occur due to actuation limits of the physical system. Such constraints can be removed by replacing them by barrier-functions of the form

$$B(u) = q_u \left(-log\left(\lambda_u - g_u(u)\right).\right)$$

The barrier function $B(u)$ is not defined for $g_u(u) > \lambda_u$, as seen in Figure 2.

In a similar way, state constraints given by

$$\begin{aligned} x_k \in \mathcal{X} &= \{x : g_x(x) \leq \lambda_x\} \\ x_N \in \mathcal{X} &= \{x : g_{x_N}(x) \leq \lambda_{x_N}\} \end{aligned}$$

can also be removed. However, in this work we transform them into soft-constraints and incorporate them as relaxed barrier functions $B_r(x)$ into the cost function, in order to always ensure feasibility of the controller. A relaxed barrier function $B_r(x)$ is given by

$$\begin{aligned} B_r(x) &= \frac{q_x}{2}\left(\sqrt{\frac{(4 + \gamma(\lambda - x)^2)}{\gamma}} - (\lambda - x)\right) \\ &\approx q_x \frac{(|\lambda - x| - (\lambda - x))}{2} \end{aligned}$$

and depicted in Figure 2.

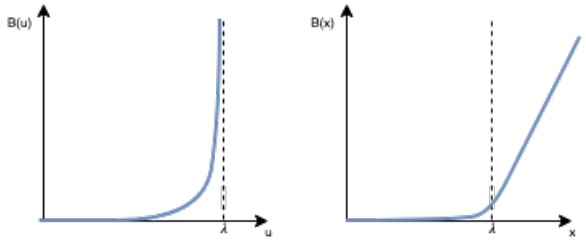


Fig. 2: Barrier functions. Input and state constraints can be removed by introducing either a barrier function $B(u)$ (left) or a relaxed barrier function $B_r(x)$ (right) into the cost function. This improves the optimizer performance.

The final result is that the GP-MPC problem (3) can be rewritten as an unconstrained optimization problem, reducing drastically the computation needed for optimization.

Another approach to decrease the computational complexity is to limit the number of data points m within the dictionary $\mathcal{D}$. If adding a new point results in $m > m_{max}$, one data pair is removed. There exist different approaches on how the number of data points can be kept low, while ensuring a good prediction. For example, the closest point to the new point in

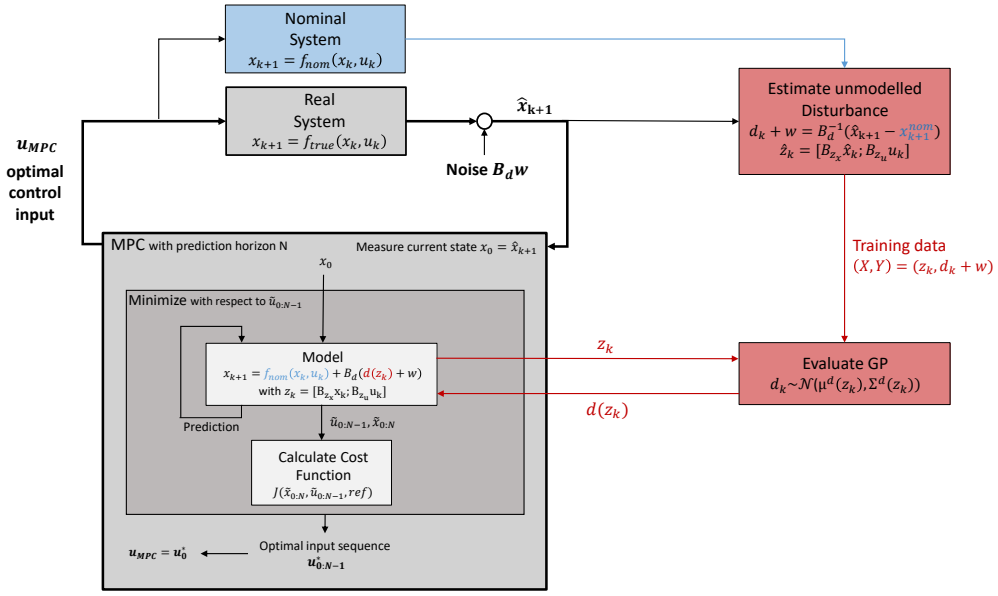


Fig. 1: Diagram showing the components of a control loop using Gaussian Process (GP)-based Model Predictive Controller (MPC). The plant is represented by the block "Real System". The MPC algorithm is illustrated within the "MPC" block. It uses a nominal model of the plant with an additional learned disturbance. The disturbances represents the model-plant mismatch and is learned using GP regression. Training data (Z,Y) (within the image (X,Y)) is generated by estimating the unmodeled disturbance. Components of GP-bases MPC corresponding to the GP are shown in red. The nominal model is depicted in blue.

Euclidean space could be removed. Alternatively, the oldest point or the point with the lowest function covariance can also be selected to be discarded.

As described in the lecture, one way of choosing the hyper-parameters θ is to set them to fixed values by either having an insight into the considered problem or by picking them based on previously seen data. But if it is not possible to pick hyper-parameters such that good predictions are achieved, hyper-parameters optimization is the way out. This happened for our second example, where in the beginning the GP did not learn the disturbance. By using hyper-parameter optimization, we were able to achieve good predictions.

The idea behind using the Maximum Likelihood method is to find the hyper-parameters θ^*, which make the observed data the most likely [9]. From

$$y = d_{\mathcal{GP}}(Z) + w, \quad d_{\mathcal{GP}} \sim \mathcal{GP}(0, k(z, z')), \; w \sim \mathcal{N}(0, \sigma_n^2 I)$$

follows that the marginal likelihood is given by

$$Y|Z, \theta \sim \mathcal{N}(0, \underbrace{K(Z,Z) + \sigma_n^2 I}_{K_y}),$$

where θ is the previously defined hyper-parameter vector $\theta = [l_1, \ldots, l_{n_z}, \boldsymbol{\sigma_f^2}, \boldsymbol{\sigma_n^2}]$. One may optimize the GP hyper-parameters by maximizing the Log Likelihood given by

$$\log p(Y|Z,\theta) = -\frac{1}{2} y^T K_y^{-1} y - \frac{1}{2}\log|K_y| - \frac{n}{2}\log(2\pi) \quad (4)$$

and the optimal hyper-parameters θ^* are given by

$$\theta^* = \arg\max_{\theta} \; \log \, p(Y|Z,\theta).$$

which allows local optimization of the hyper-parameters, since (4) is nonconvex. We independently perform one optimization for each GP output dimension. We use the gradient-free tool `fmincon` from Matlab, which showed to be efficient. Alternatively, the gradient of (4) could have been easily derived and used in the optimizer, as shown in [11].

V. Examples

Two examples were considered during this project. An inverted pendulum and an autonomous racing problem based on a single-track model. The inverted pendulum example is constructed such that we are able to illustrate important aspects of GP-based MPC. The autonomous racing example is application oriented. Both examples are introduced in this chapter.

A. Inverted Pendulum

The objective of the considered inverted pendulum example is to achieve an upright pendulum position of the pole by applying force to the cart. A schematic drawing of the inverted pendulum including its parameters can be seen in Figure 3. The example is constructed such that it is possible to display the resulting GP in 3D. Thus, the true model which is actually defined by

$$\begin{aligned} \boldsymbol{x_{k+1}} &= f_{true}(\boldsymbol{x_k}, \boldsymbol{u_k}) \\ &= f_{nom}(\boldsymbol{x_k}, \boldsymbol{u_k}) + \boldsymbol{B_d}(\boldsymbol{d_{true}}(\boldsymbol{z_k}) + \boldsymbol{w}) \end{aligned}$$

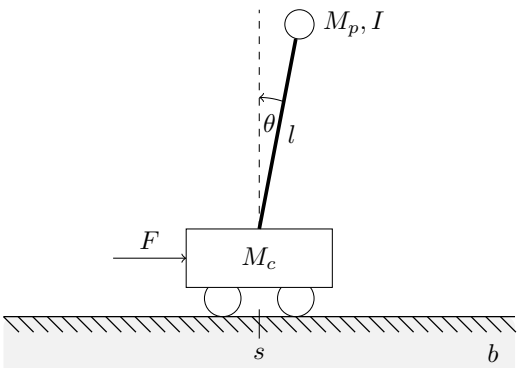


Fig. 3: A schematic drawing of the inverted pendulum on a carriage. The mass of the carriage and the pole are given by M_c and M_p, respectively. The pole is defined by its length l and its moment of inertia I. b is the friction coefficient between the carriage and the floor. The states of the state-space model are the carriage position s, its derivative $\dot{s}$, the pole angle with the vertical θ and its derivative $\dot{\theta}$. F is the applied force.

is chosen, with $\boldsymbol{z_k} = [\theta\ \dot{\theta}]^T$ and $\boldsymbol{B_d} = [0\ 0\ 1\ 0]$. This means that the disturbance between the nominal and the true model only affects the third state (θ) and depends on the pole angle with the vertcial θ as well as its derivative $\dot{\theta}$. Thus, the size of the GP input space and output space are $n_z = 2$ and $n_d = 1$, respectively.

The nominal model of the 2-dimensional inverted pendulum is based on the following equations of motions:

$$(M_c + M_p)\ddot{x} + b\dot{x} + \frac{1}{2}M_p l\ddot{\theta}\cos\theta - \frac{1}{2}M_p l\dot{\theta}^2\sin\theta = F$$
$$\left(I + M_p\left(\frac{l}{2}\right)^2\right)\ddot{\theta} - \frac{1}{2}M_p g l\sin\theta + M_p l\ddot{x}\cos\theta = 0.$$

The standard gravity on the surface of the earth g is set to be $g = 9.8$. After establishing a nonlinear state space model with the states $[x, \dot{x}, \theta, \dot{\theta}]$ and input $u = F$, followed by discretization using Runge-Kutta 4th order the nominal model is given by

$$\boldsymbol{x_{k+1}} = f_{nom}(\boldsymbol{x_k}, \boldsymbol{u_k}).$$

Here, the explicit equations are not mentioned since they are not relevant for further explanations. The true disturbance d_{true}, introduced above, represents a defect in the joint connecting the cart and the pole and is explicitly given by

$$\begin{aligned} d_{true} &= 0.1\theta - 0.01\dot{\theta} + 0.0524 \\ &= 0.1z(1) - 0.01z(2) + 0.0524, \end{aligned}$$

with $\boldsymbol{z} = \boldsymbol{B_{z_x}}\boldsymbol{x} + B_{z_u}u$ and

$$\boldsymbol{B_{z_x}} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix},\ B_{z_u} = 0.$$

In our example the parameters of the inverted pendulum were set to $M_c = 5$, $M_p = 2$, $I = 0.6$, $l = 3$ and $b = 0.1$. The implemented MPC has a prediction horizon $N = 10$ and the number of iterations to optimize the cost function at each time step is limited to 15. The control formulation aims to stabilize the pole at $\theta = 0$ with $\dot{\theta} = 0$ and $\dot{s} = 0$. Therefore, the cost function f_o is defined to

$$f_o(\boldsymbol{x}, u) = (\boldsymbol{C}\boldsymbol{x} - \boldsymbol{r})^T\boldsymbol{Q}(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{r}) + Ru^2,$$

where $\boldsymbol{Q}$ and R are weight matrices/values penalizing the deviation between a selection of states $\boldsymbol{C}\boldsymbol{x_k}$ and the reference signal r as well as the amplitude of the input force F, respectively. The reference $\boldsymbol{r}$ is defined to $\boldsymbol{r} = [0\ 0\ 0]$ and the states $\dot{s}$, θ and $\dot{\theta}$ are picked by $\boldsymbol{C}$
The terminal cost is defined by

$$\phi(\boldsymbol{x}) = (\boldsymbol{C}\boldsymbol{x} - \boldsymbol{r})^T\boldsymbol{Q}(\boldsymbol{C}\boldsymbol{x} - \boldsymbol{r}).$$

The disturbance within the estimated dynamics (see (2)) is learned using the squared exponential kernel. The hyper-parameters were preset to $\sigma_f = 0.03$, $\boldsymbol{M} = \text{diag}(0.1, 0.1)^2)$ and $\sigma_n^2 = 10^{-8}$. Later, the hyper-parameters were optimized with the above values as initial values. For a comparison between the learned disturbance with and without hyper-parameter optimization see section VI. The maximum number of points in the dictionary was set to $m_{max} = 100$. Since the simulation time was only 7s and the simulation step size 0.1s the maximum number of data points was not reached, when starting without offline training.

B. Autonomous Racing

This example considers an autonomous vehicle with the aim to drive around a defined race track as fast as possible without leaving the track. The true model is simulated using a non-linear single-track model with non-linear wheel dynamics following the Pacejka formula. In order to explore the potential of learning-based MPC, we consider that the nominal vehicle has been modeled as a similar non-linear single-track vehicle model but with linear wheel dynamics and wrong model parameters. The GP will then be responsible for capturing these unmodeled-dynamics/deviation between the true and the nominal model.

A schematic drawing of the single-track model used in this example including the relevant parameters can be seen in Figure 4. We follow the approach presented in [3].

For both models the state-space representation of the single-track model with the states $x = [I_x\ I_y\ \psi\ V_{x'}\ V_{y'}\ \dot{\psi}\ d_{track}]^T$ is given by

$$\dot{x}(x,u) = \begin{bmatrix} V_{x'}\cos(\psi) - V_{y'}\sin(\psi) \\ V_{x'}\sin(\psi) + V_{y'}\cos(\psi) \\ \dot{\psi} \\ 1/M(F_{r,x'} + F_{f,x'} * \cos(\delta) - F_{f,y'}\sin(\delta) + V_{y'}\dot{\psi}) \\ 1/M(F_{r,y'} + F_{f,x'}\sin(\delta) + F_{f,y'}\cos(\delta) - V_{y'}\dot{\psi}) \\ 1/I(F_{f,y'}L_f\cos(\delta) + F_{f,x'}L_f\sin(\delta) - F_{r,y'}L_r) \\ v_{track} \end{bmatrix},$$

with input vector $u = [\delta, T, v_track]$, where $\delta \in [-30°, 30°]$ is the steering angle, $T \in [0, 1]$ is the acceleration/brake pedal and v_{track} the track center line velocity. d_{track} and

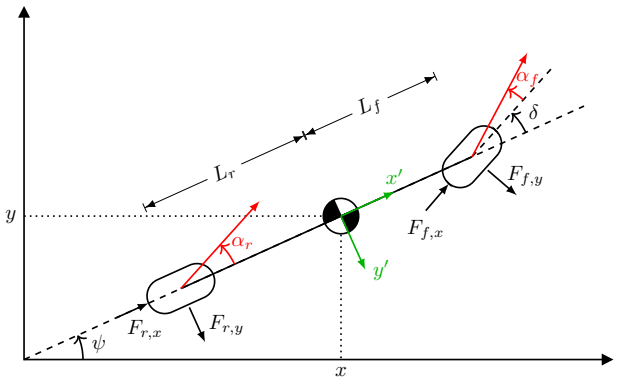


Fig. 4: A schematic drawing of the single-track model. The parameters are vehicle mass M, vehicle moment of inertia I, distance from the front wheel as well as from the rear wheel to the center of mass are (L_f and L_f). The forces which act on the rear and front wheel in longitudinal and lateral direction are named $F_{r,x'}$, $F_{r,y'}$, $F_{f,x'}$ and $F_{f,y'}$. x' and y' define the vehicle coordinate system. The rear-slip and front-slip angle are α_r and α_f, respectively.

v_{track} are virtual state and input and their meaning are explained later.

The longitudinal wheel forces in vehicle coordinates are modeled simply as proportional to the acceleration/brake pedal T and the torque distribution ξ by:

$$\begin{aligned} F_W &= T((T>0)4000 + (T<0)8000 sign(V_{x'})) \\ F_{r,x'} &= \zeta F_W \\ F_{f,x'} &= (1-\zeta) F_W \end{aligned}$$

For the true model we use nonlinear lateral wheel dynamics according to the Pacejka Magic formula [8], given by

$$\begin{aligned} F_{r,y'} &= D_r \sin\Big[C_r \arctan\Big(B_r\alpha - E_r\big(B_r\alpha - \arctan(B_r\alpha)\big)\Big)\Big] \\ F_{f,y'} &= D_f \sin\Big[C_f \arctan\Big(B_f\alpha - E_f\big(B_f\alpha - \arctan(B_f\alpha)\big)\Big)\Big], \end{aligned}$$

with stiffness factor B, peak factor D and shape factors C and E.

On the other hand, the nominal model assumes linear tyre force characteristics given by a linear relation between lateral wheel force $F_{y'}$ and wheel slip angle α

$$F_{r,y'} = C_{l,r}\alpha_r F_{f,y'} = C_{l,f}\alpha_f,$$

where $C_{l,r}$ and $C_{l,f}$ are the rear and front cornering stiffness.

For both cases, the wheel slip angles α_r and α_f are defined by

$$\begin{aligned} \alpha_r &= \text{atan2}(V_{y'} - L_r\dot{\psi}, V_{x'}) \\ \alpha_f &= \text{atan2}(V_{y'} + L_f\dot{\psi}, V_{x'}) - \delta. \end{aligned}$$

In order to make the difference between true and nominal model even more distinct, we consider very wrong estimations of the cornering stiffness. The lateral rear wheel force as a function of the slip angle for the true and nominal lateral wheel dynamics are shown in Figure 5.

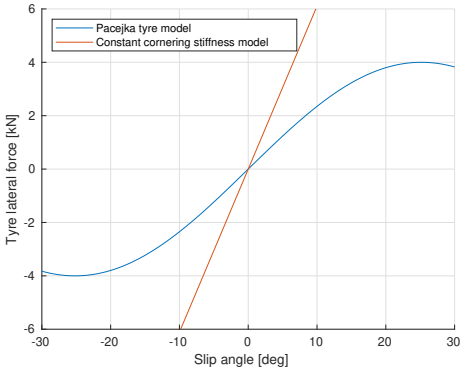


Fig. 5: Rear wheel lateral force for true (Pacejka) and nominal model

Finally, the last deviation between the two models consists of different model hyper-parameters. The nominal model overestimates the mass and the yaw-inertia in 50%. With those absurd differences, we intended to expose the weakness of MPC towards very wrong or unmodeled dynamics.

After having specified the true and nominal model, it is time to define the cost function for the MPC. In this example, we follow a similar strategy as used in Model Predictive Contouring Controller (MPCC) [3]. The center line of the race track is parameterized by $s \in [0, s_{max}]$, which represents the traveled distance during the current lap. Given an exemplary s, the center line position $[X_c(s),\ Y_c(s)]$ and orientation $\psi(s)$ can be evaluated. The added state d_{track} represents the approximately traveled distance along the center line.

Moreover, with the virtual input v_{track} we predict positions on the center line, the so called *optimal projected points*. Each of those points is linked to the correspondent prediction of the vehicle position, which does not necessarily lie on the center line. The idea is to maximize the progress of the *optimal projected points* on the center line while keeping each correspondent predicted vehicle position close. We ensure that they are close by penalizing the lag error e_l, contour error e_c, orientation error e_o and offset error e_{off}, as illustrated in Figure 6.

This results in a min max problem, where the aim is to maximize the progression while keeping the vehicle close to the center line. The corresponding optimal cost is defined by

$$J(\bar{x}, u^*_{0:N-1}) = \sum_{k=0}^{N-1} \min_{u(1:2)} \max_{v_{track}} \big[f_o(\mu^x_k, u_k) + q_d \cdot v_{track}\big],$$

where f_o is a weighted function of the squared errors

$$f_o(\mu^x_k, u_k) = q_c\, e_c^2 + q_l\, e_l^2 + q_e\, e_o^2 + q_{off} e_{off}^2.$$

For sake of simplify we will not fully describe the cost function here. For more information please refer to our implementation.

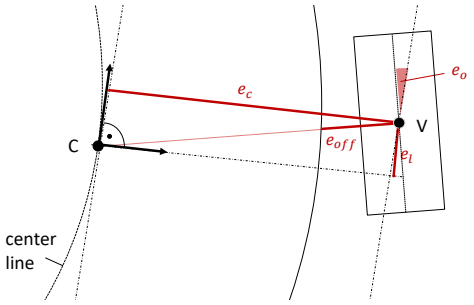


Fig. 6: Lag-, contour, orientation- and offset error. Lag error e_l, contour error e_c, orientation error e_o and offset error e_{off} are defined between the actual vehicle position V and the the vehicle position C on the center line corresponding to the state d_{track}. The offset error e_{off} is 0 when the vehicle is within a circle with radius $R/2$ around C. Here, R is the race track width.

VI. Results

In this section we present and discuss the results of both examples. Both examples were implemented in MATLAB. First, the inverted pendulum results are presented, followed by the results of the autonomous racing example.

The initial condition of the inverted pendulum was set to $x_0 = [0,\ 0,\ 5°,\ 0]$. Four simulations are depicted in Figure 8. First, we run the standard-MPC using the nominal model, while the GP is deactivated, but accumulating data. In the next run, GP is activated, using the knowledge from the last simulation and also accumulating new data from the current run. Further, we perform hyper-parameter optimization using the data collected and run the same experiment again. Lastly, the fourth run shows the standard-MPC with the perfect knowledge of the underlying true dynamics, as a reference for the other cases.

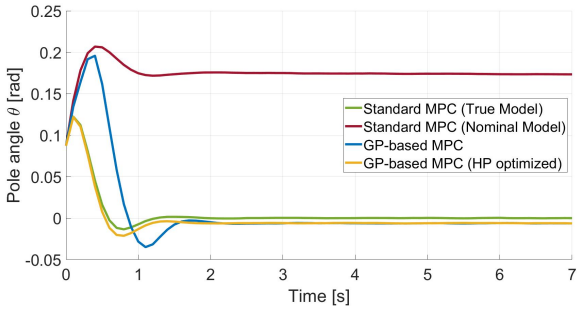
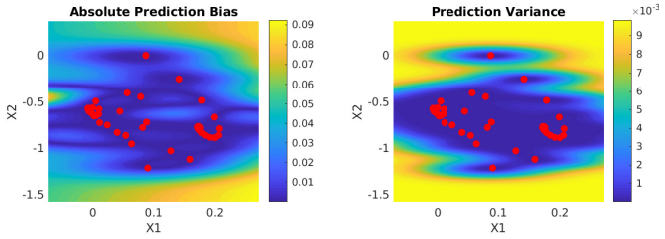


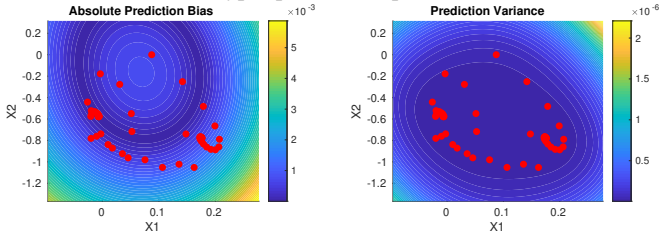
Fig. 8: Angle over time. Data for GP is collected offline and further added online.

The standard-MPC using the nominal model shows a constant offset in θ and drives the cart to infinity in order to keep the pole in that position. It is not able to successfully stabilise the pole. Using the GP results in a stabilisation. Hyper-parameter optimization clearly improves its performance and leads to a faster stabilzation.

Having an analytical expression for the true unmodeled dynamics allows us to evaluate the learning performance of the Gaussian Progress regression model. The mean of the true and the learned unmodeled dynamics are depicted in Figure 7a and Figure 7b, respectively. As can be seen, the initial hyper-parameter values chosen before optimization led to an overfitting of the data. The model is able to make accurate predictions for the training data but is not able to generalize. After hyper-parameter optimization (training), the prediction bias is decreased (as seen in Figures 9a and 9b), which confirms that training is highly beneficial for the model.



(a) Before hyper-parameter optimization



(b) After hyper-parameter optimization

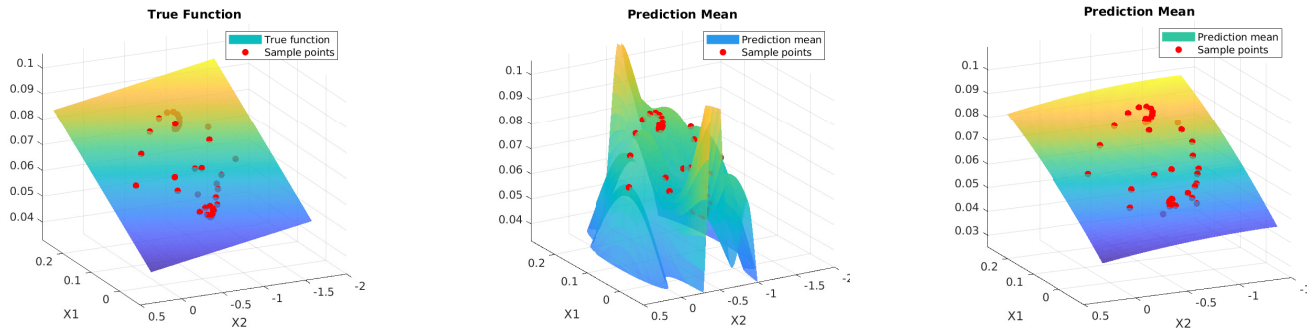
Fig. 9: Absolute Prediction Bias (left) and Prediction Variance (right). On the $X1$ and $X2$ axis θ_k and $\dot{\theta}_k$ are depicted, respectively

For the autonomous racing example we evaluated the lap time as a performance criterium. The GP was trained offline with data accumulated using the nominal model and the hyper-parameters were optimized. In Table I the lap times using the nominal and the estimated model are depicted. The lap time of the first round was neglected. It can be seen that using GP-based MPC results in lower lap times for all laps. The mean lap time was also decreased by 18.6% compared to standard MPC.

Lap	Lap time [s]
1	15.30
2	15.75
3	14.25
4	15.15
avg.	**15.11**

Lap	Lap time [s]
1	12.45
2	11.85
3	12.75
4	12.15
avg.	**12.3**

TABLE I: Lap times when using the nominal (left table) and the estimated model (right table) within the MPC formulation. For both cases the average lap time was calculated.

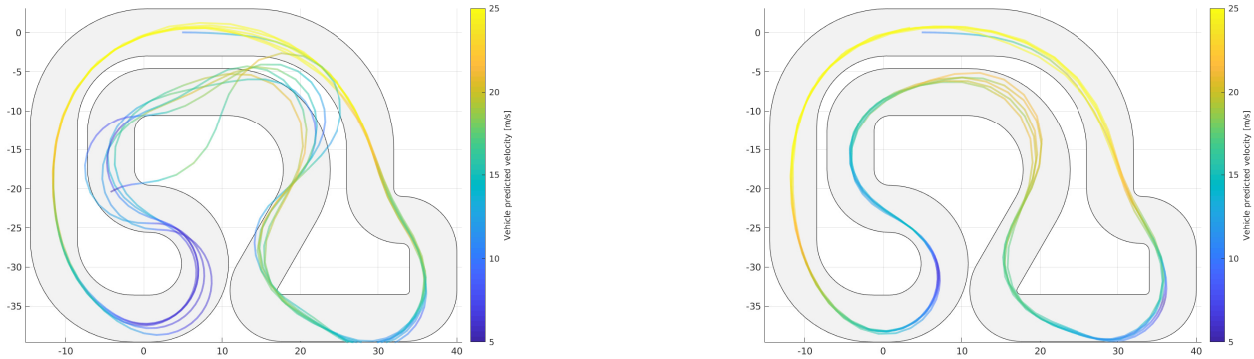


(a) True disturbance

(b) Estimated disturbance without hyper-parameter optimization

(c) Estimated disturbance with optimized hyper-parameters

Fig. 7: True and learned unmodeled dynamics mean of the Gaussian Process regression. On the $X1$ and $X2$ axis θ_k and $\dot{\theta}_k$ are depicted, respectively.



(a) Trace when using Standard MPC

(b) Trace when using GP-based MPC

Fig. 10: Difference of controller performance when unmodeled dynamics are estimated using a Gaussian Process

In Figure 10a the trace of the vehicle and its velocity in color is depicted for 5 full rounds. The standard MPC is unable to keep the vehicle inside the race track. This is because the standard MPC uses a too simplified prediction model, which also overestimates its lateral wheel force available for a certain slip angle. Here, we refer again to Figure 5. In Figure 10b the trace and velocity of a GP-based MPC after hyper-parameter optimization is displayed. The vehicle stays within the race track boundaries at all times. The traces of all 5 rounds are more consistent and for the most part identical compared to the previous case. Without hyper-parameter optimization, the MPC was still unable to complete the task and stay in bound even though GP was active and learning. Finding good estimations for the hyper-parameters can be quite challenging as in this case and hyper-parameter optimization had to be utilized.

VII. CONCLUSIONS

This paper provides an introduction on how to combine GP Regression with MPC. The advantage of using GP Regression to learn the unmodeled dynamics is, that no precise system model has to be derived. Deriving a such a precise system model is hard due to complex dynamics relations. Thus, a nominal model of the dynamics can be used in combination with the learned unmodeled dynamics.

For both presented examples a significant improvement of the performance was achieved by adding the learned unmodeled dynamics. Their realization included hyper-parameter optimization as well as an efficient implementation of MPC and GP Regression, which reduced the computational complexity. Since finding good estimates for the hyper-parameters without prior knowledge is difficult, hyper-parameter optimization (GP training) played an important role and was able to substantially improve the final controller performance for both examples.

One also has to take into account that MPC itself can yield good performance for a small model mismatch, due to its robustness. Therefore the design process of both examples included increasing the model mismatch such that GP regression was beneficial. However, in real life scenarios the true dynamics are usually more complex, resulting in a larger model mismatch. For example, we used a single-track model to describe the true dynamics within the second example,

which is actually not fully representative.

The advantage of GP Regression is, that the residual uncertainty of the prediction model is also estimated. Although we propagated the uncertainty, we did not use the resulting variance, since the computation would become intractable. It is noteworthy that using the propagated variance is a big advantage of GP Regression. By considering it in the cost function or introducing it as a constraint, the performance and robustness can be improved.

REFERENCES

[1] Frank Allgöwer, Rolf Findeisen, Christian Ebenbauer, et al. Nonlinear model predictive control.

[2] Lukas Hewing, Juraj Kabzan, and Melanie N. Zeilinger. Cautious Model Predictive Control using Gaussian Process Regression. *arXiv:1705.10702 [cs, math]*, November 2018. arXiv: 1705.10702.

[3] Juraj Kabzan, Lukas Hewing, Alexander Liniger, and Melanie N Zeilinger. Learning-based model predictive control for autonomous racing. *IEEE Robotics and Automation Letters*, 4(4):3363–3370, 2019.

[4] Juš Kocijan, Roderick Murray-Smith, Carl Edward Rasmussen, and Agathe Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American control conference*, volume 3, pages 2214–2219. IEEE, 2004.

[5] Franziska Meier, Philipp Hennig, and Stefan Schaal. Efficient bayesian local model learning for control. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2244–2249. IEEE, 2014.

[6] Duy Nguyen-Tuong, Jan R Peters, and Matthias Seeger. Local gaussian process regression for real time online model learning. In *Advances in Neural Information Processing Systems*, pages 1193–1200, 2009.

[7] Chris J Ostafew, Angela P Schoellig, and Timothy D Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4029–4036. IEEE, 2014.

[8] Hans Pacejka. *Tire and vehicle dynamics*. Elsevier, 2005.

[9] Sebastian Trimpe. *Lecture Notes Statistical Learning and Stochastic Control*. Max Planck Institute for Intelligent Systems, November 2019.

[10] Benjamin Van Niekerk, Andreas Damianou, and Benjamin S Rosman. Online constrained model-based reinforcement learning. 2017.

[11] Christopher KI Williams and Carl Edward Rasmussen. *Gaussian processes for machine learning*, volume 2. MIT press Cambridge, MA, 2006.