

Learning Based Approximate Model Predictive Control for Nonlinear Systems [★]

D. Gálgó ^{*}, T. Péni ^{*} R. Tóth ^{**}

^{*} *Systems and Control Laboratory, Institute for Computer Science and Control, Hungarian Academy of Sciences, H-1111 Bp. Kende u. 13-17.
(e-mail: gango.daniel@sztaki.mta.hu, peni.tamas@sztaki.mta.hu).*

^{**} *Eindhoven University of Technology, P.O. Box 513, 5600 MB Eindhoven, The Netherlands. (e-mail r.toth@tue.nl)*

Abstract: The paper presents a systematic design procedure for approximate explicit model predictive control for constrained nonlinear systems described in *linear parameter-varying (LPV)* form. The method applies a *Gaussian process (GP)* model to learn the optimal control policy generated by a recently developed fast *model predictive control (MPC)* algorithm based on an LPV embedding of the nonlinear system. By exploiting the advantages of the GP structure, various active learning methods based on information theoretic criteria, gradient analysis and simulation data are combined to systematically explore the relevant training points. The overall method is summarized in a complete synthesis procedure. The applicability of the proposed method is demonstrated by designing approximate predictive controllers for constrained nonlinear mechanical systems.

© 2019, IFAC (International Federation of Automatic Control) Hosting by Elsevier Ltd. All rights reserved.

Keywords: model predictive control; Gaussian process; linear parameter-varying systems; machine learning.

1. INTRODUCTION

Model predictive control (MPC) has several attractive features that make it an important control technology for engineering applications (Rakovic and Levine, 2019). For example, MPC can naturally handle strict state and input constraints and various performance specifications can be easily added to the design process. However, the price to be paid for these advantages is the high computational effort needed to obtain the control input: at every time instant a constrained optimization task has to be performed to get the next control action. This computational demand makes MPC less attractive for systems with fast dynamical components, e.g. mobile robots, aerospace applications and automotive systems. In order to apply predictive controllers to these model classes, the computational time has to be significantly decreased.

One approach to speed up the MPC is to compute the optimal control input as a function of the measured variables (e.g. as a function of the state if the state is available for measurement), store this function in memory and simply evaluate it during the control process. This strategy is called explicit MPC. If the optimization problem is linear or quadratic (this is the case if the system to be controlled is linear, the constraints are linear and the cost is linear or quadratic), the procedure that can be used to construct the parametric control function is *multiparametric linear or quadratic programming (mpLP, mpQP)*, see e.g. Borrelli et al. (2019) for more details.

[★] This work was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and the ÚNKP-18-4 New National Excellence Program of the Ministry of Human Capacities. It was also supported by the research program titled "Exploring the Mathematical Foundations of Artificial Intelligence (2018-1.2.1-NKP-00008)".

For nonlinear problems, the multiparametric optimization is currently under development (Johansen, 2003; Dominguez et al., 2010), however, reliable solvers are not available yet.

This is where approximate solutions come into the picture. Here the goal is to approximate the optimal control input up to some predefined tolerance. Theoretically, any function approximation method can be used, e.g., piecewise linear functions (Johansen, 2003), set membership methods (Canale et al., 2010), neural networks and machine learning (ML) solutions (Parisini and Zoppoli, 1995), (Csekő et al., 2015), (Hertneck et al., 2018). The latter methods are especially promising, because the resulting control laws can be efficiently implemented by using recently developed parallel software and hardware architectures, enabling applicability even for large-scale systems. As the approximation is independent of the underlying MPC algorithm, (it sees only the training data), this approach can be used for nonlinear MPC (NMPC) problems as well. Motivated by these attractive features of ML methods, the paper proposes a practical procedure for learning based approximate NMPC design. Compared to the other approaches, we put our focus on numerical efficiency: first, a fast nonlinear MPC algorithm based on *linear parameter-varying (LPV)* embedding is applied to speed up the training point computation, second, a systematic procedure is proposed for exploring the most relevant training samples in order to improve the accuracy of the approximation while keeping the training set at a manageable size.

The paper is organized as follows. In Section 2 the LPV embedding based NMPC algorithm is introduced, while in Section 3, the concept of Gaussian process model based policy approximation is summarized. These are the

core components of the approximate MPC design method presented and analyzed in Section 4. The applicability of the proposed procedure is demonstrated in Section 5 on two application examples. The paper is finished with conclusions from the results achieved.

2. NONLINEAR MPC BASED ON LPV EMBEDDING

This section presents the nonlinear MPC algorithm developed by H. Werner and his co-authors in Cisneros et al. (2016). This algorithm is used to compute the optimal state feedback control policy that is then learned by a function approximator. The NMPC method is based on embedding of the nonlinear dynamics in an LPV model and applies iterative *quadratic programming (QP)* to solve the nonlinear optimization problem associated with the MPC synthesis. The algorithm is highly efficient as it has been demonstrated on moderate scale practical problems, although the convergence of the QP iteration has not been completely proven.

To begin, consider a nonlinear, discrete-time system represented in an LPV form:

$$x_{k+1} = A(\rho(x_k))x_k + B(\rho(x_k))u_k \quad (1)$$

where k is the time index, $x_k \in \mathbb{R}^{n_x}$ is the state, $u \in \mathbb{R}^{n_u}$ is the control input and $\rho(\cdot)$ denotes the state-dependent scheduling parameter. For simplicity, we assume that the state is available for measurement. The control goal is to perform the standard regulation task, i.e., to steer the state from some initial value $x_0 \neq 0$ to the origin while minimizing a quadratic cost and satisfying a set of linear constraints prescribed for the state and input trajectory. In the MPC setting this can be formalized by the following nonlinear optimization problem that has to be solved at each time instant to obtain the next control action u_k :

$$\min_{\substack{u_{k|k}, \dots \\ u_{k+N-1|k}}} \sum_{i=0}^{N-1} x_{k+i|k}^\top Q x_{k+i|k} + u_{k+i|k}^\top R u_{k+i|k} + x_{k+N|k}^\top W x_{k+N|k} \quad (2a)$$

$$x_{k+i+1|k} = A(\rho(x_{k+i|k}))x_{k+i|k} + B(\rho(x_{k+i|k}))u_{k+i|k} \quad (2b)$$

$$x_{k|k} = x_k \quad (2c)$$

$$F x_{k+i|k} + G u_{k+i|k} \leq h \quad (2d)$$

$$x_{k+N|k} \in X_T \quad (2e)$$

Here $x_{k+i|k}$, $i = 0 \dots N$ are the predictions of the future states based on the actual measurement x_k and the control input sequence $u_{k|k}, \dots, u_{k+N-1|k}$. Inequalities (2d) represent the constraints, $x^\top W x$ and X_T are the terminal cost and terminal set, respectively. The terminal ingredients used to ensure stability guarantees can be constructed by one of the several methods available in the literature. A commonly used approach is to linearize the dynamics around the origin and construct the maximal ellipsoidal controlled invariant set for the linear system obtained. The details of this procedure are described, e.g., in Cannon et al. (2011).

To solve (2) Cisneros et al. (2016) proposes an iterative algorithm. In each iteration the parameter trajectory is fixed by using the state sequence obtained in the previous

step. This simplifies the dynamic model (2b) to a *linear time-varying (LTV)* system so the optimization problem can be solved by quadratic programming. The result is a new control input sequence. By applying this sequence on the nonlinear model, the next prediction of the state trajectory is obtained. The method is summarized in Algorithm 1.

Algorithm 1 NMPC by LPV embedding

Input: State vector x , control horizon N .

Output: Control input u

- 1: Let $\bar{x}_0 = \bar{x}_1 = \dots = \bar{x}_{N-1} = x$
 - 2: **while** the state and input trajectories do not converge **do**
 - 3: Compute $\bar{\rho}_i = \rho(\bar{x}_i)$, $i = 0, \dots, N - 1$.
 - 4: Solve (2) with the LTV dynamics
 $x_{i+1} = A(\bar{\rho}_i)x_i + B(\bar{\rho}_i)u_i$, $x_0 = x$.
 The result is the optimal u_0^*, \dots, u_{N-1}^* sequence.
 - 5: Compute the state response of the nonlinear plant for u_0^*, \dots, u_{N-1}^*
 $\bar{x}_{i+1} = A(\rho(\bar{x}_i))\bar{x}_i + B(\rho(\bar{x}_i))u_i^*$,
 for $i = 0, \dots, N - 1$, $\bar{x}_0 = x$.
 - 6: Let $u = u_0^*$
-

It has been shown in Cisneros et al. (2018) that the iteration converges very quickly in practice: the stopping criterion is reached typically after 5-10 iterations. Since the procedure is based on quadratic programming, it requires much less computation time than a general nonlinear solver. This represents a serious advantage of Algorithm 1, compared to other NMPC methods in training set generation, where Algorithm 1 has to be performed several times with different initial states.

3. GAUSSIAN PROCESS BASED FUNCTION APPROXIMATION

Next, to obtain an explicit form of the control law represented in Algorithm 1, we apply a *Gaussian process (GP)* to approximate the general optimal control policy. GP, which represents a single layer based neural network, is chosen for this task, because it has a simple, expressive structure, that depends on relatively few tuning parameters and its training is fast and efficient. Moreover, GP provides information on the reliability of the approximation, which can be used to systematically select relevant training samples (active learning). A deep theoretical analysis of Gaussian processes can be found in Rasmussen and Williams (2006) while various engineering applications exploiting the attractive features of this structure are presented e.g. in (Liu et al., 2018), (Darwish, 2017), (Sharif, 2018), respectively.

From mathematical point of view, a GP is an infinite dimensional extension of the multivariate Gaussian distribution. Formally, a Gaussian process $\mathcal{GP} : \mathbb{R}^n \rightarrow \mathbb{R}$ is a mapping that assigns to every point $x \in \mathbb{R}^n$ a random variable $\mathcal{GP}(x) \in \mathbb{R}$ such that for any finite set $x^{(1)} \dots x^{(M)}$ the joint probability distribution of $\mathcal{GP}(x^{(1)}), \dots, \mathcal{GP}(x^{(M)})$ is Gaussian with mean m and covariance K , where:

$$m = [m(x^{(1)}), \dots, m(x^{(M)})]^T \quad (3)$$

$$[K]_{ij} = \kappa(x_i, x_j). \quad (4)$$

Here $[\cdot]_{ij}$ denotes the (i, j) -th entry of a matrix and κ is a suitable kernel function. (In the paper, we often use $\kappa(\cdot)$ with matrix arguments, so we may write $K = \kappa(X, X)$).

Both $m(\cdot)$ and $\kappa(\cdot)$ depend on additional tuning variables, denoted by θ . These are called the *hyperparameters* of the model. For given $m(\cdot)$ and $\kappa(\cdot)$ the sampling of the GP means sampling the Gaussian random variables at all $x \in \mathbb{R}^n$. The samples define a (deterministic) function $g : \mathbb{R}^n \rightarrow \mathbb{R}$, so the Gaussian process can be interpreted as a distribution over functions as well. If GP is used for regression, the goal is to learn a continuous function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ by using a training set composed of $(x, f(x))$ tuples. The training is based on assuming that f is a sample of a GP and the goal is to find the most probable GP that can generate the training set. For this, the first step is the selection of the mean and kernel functions (model selection). By correcting the training data with its mean, $m(\cdot) \equiv 0$ can be chosen in general. It is thus enough to focus on the selection of the kernel function. The kernel is the core of the GP model. It determines the function class the GP is able to approximate. If the function to be learned is smooth and its characteristic length is almost constant, a simple Squared Exponential (SE) kernel is a good choice. On the other hand, if fast changes and discontinuities are expected, a more complex kernel, e.g. the Matern class kernel has to be chosen. Further kernel functions with the related modeling capabilities are discussed in detail in Rasmussen and Williams (2006).

The next phase of the training is the tuning of the hyperparameters θ . The most common learning rule is obtained by maximizing the marginal likelihood of the training samples. Specifically, if $\mathcal{T} = \{(x^{(1)}, \bar{y}^{(1)}), \dots, (x^{(M)}, \bar{y}^{(M)})\}$ is the training set and $p(y|X, \theta)$ with $X = [x^{(1)} \dots x^{(M)}]$ denotes the M dimensional joint Gaussian distribution of $\mathcal{GP}(x^{(1)}) \dots \mathcal{GP}(x^{(M)})$, then the goal is to maximize the *marginal likelihood* $\log p(\bar{y}|X, \theta)$ where $\bar{y} = [\bar{y}^{(1)} \dots \bar{y}^{(M)}]^\top$. Since the gradient of $\log p(\bar{y}|X, \theta)$ in θ can be easily evaluated, a simple gradient ascent algorithm can be applied.

Now, assume that the GP has already been trained. An approximation of f at a test point $x_* \in \mathbb{R}^n$ is obtained by taking the $M + 1$ dimensional joint distribution $p([y^{(1)}, \dots, y^{(M)}, y] | [X, x_*], \theta)$ and computing the one dimensional conditional distribution $p(y|x_*, y^{(1)} = \bar{y}^{(1)}, \dots, y^{(M)} = \bar{y}^{(M)}, X, \theta)$. The mean m_* of this distribution is considered to be the approximation for $f(x_*)$, while the variance σ_* provides information on the uncertainty of the regression. As all distributions above are Gaussian, the evaluation of a GP requires only elementary matrix manipulations so it can be performed efficiently.

4. ACTIVE LEARNING OF APPROXIMATE MPC POLICY

4.1 Outline of the concept

In many safety critical mechatronic and automotive applications, it is highly important to ensure deterministic computation time of the control law under a fast sampling rate. Even if Algorithm 1 has one of the fastest solution times, its rate of convergence to an optimum is problem dependent. Hence, to enable reliable real-time execution of the LPV MPC method, we intend to construct an approximate controller that is much faster to evaluate online and has a deterministic execution time. For this, a training set $\mathcal{T} = \{(x^{(1)}, u^{(1)}), \dots, (x^{(M)}, u^{(M)})\}$ is generated by performing Algorithm 1 M times with initial

values $x^{(1)} \dots x^{(M)}$ and then a GP is trained by using this data to learn the optimal control policy.

A crucial part of training is to select the most relevant training samples and keep the size M of the training set minimal. This gives rise to the need for a systematic method for exploring the most informative training points that help to reduce the approximation error.

Along with the results presented in Krause et al. (2008), Brochu et al. (2010) and Boef, den P. (2019) we apply the following method for training point selection. First the training set is initialized, then active learning methods are applied to select additional training points. Finally, the training set is refined by controlling the system such that both the NMPC and the approximated control inputs are simultaneously computed at each time instant. The points where the GP performs poorly compared to the NMPC are added to the training set. Note that the simultaneous control can be implemented in simulation, or on a real plant provided that a suitably powerful computing device is available to run the NMPC and GP in real time. Of course, after the required level of precision is reached, the online NMPC can be removed from the loop. This can be seen as a procedure of tuning in a laboratory environment before the approximate MPC can be deployed on the real system.

4.2 Step I: Initial training set generation

Let (\mathbb{X}, \mathbb{U}) denote the admissible input and state space. Moreover, let $\mathcal{V} \subset \mathbb{X}$ denote a set of discrete samples, the potential places of training points. A straightforward way to specify \mathcal{V} is to generate a dense, equally spaced grid over the state space, but it can be also generated by random sampling as well. The initial training set \mathcal{T}_0 can be determined in two different ways. One approach is to randomly draw entries from \mathcal{V} , and calculate their corresponding input values using the NMPC algorithm. The other, more structured method is to get the instances of \mathcal{V} which lay on a sparse, equally spaced grid of the state space. In the sequel, we will denote by \mathcal{A} all the points of \mathcal{V} that are present in the training set \mathcal{T} . The complementer set $\mathcal{V} \setminus \mathcal{A}$, collecting the free points is denoted by $\bar{\mathcal{A}}$.

4.3 Step II: Active learning based training point generation

Once the initial training set \mathcal{T}_0 has been generated, additional training points are selected from $\bar{\mathcal{A}}$, to reduce the approximation error of the GP. In this phase, we want to ensure that each point selected decreases the most the global approximation error reduction of the GP with respect to the control policy. For this purpose, 3 training point selection methods are proposed. They can be used individually, or at the cost of computational complexity can be combined to achieve better performance.

Maximum gradient: It is straightforward to assume that the more abruptly the approximated function changes, the more training points are needed for its accurate approximation. Therefore, the gradient of the mean function of the GP can be used as a query function to the training point selection. In case of one test point x_* , the gradient of the mean function is

$$\nabla \bar{f}_{x_*} = \nabla \mathbf{k}_*^\top K_{\bar{\mathcal{A}}}^{-1} \mathbf{y}, \quad (5)$$

where $X_{\mathcal{A}} = [x^{(1)} \ x^{(2)} \ \dots \ x^{(M)}]^\top$ is a matrix assembled from the elements of \mathcal{A} , $y = [y^{(1)} \ y^{(2)} \ \dots \ y^{(M)}]^\top$ is the vector of input values corresponding to the elements of \mathcal{A} , $k_* = \kappa(X_{\mathcal{A}}, x_*)$ is the vector of covariances between the test point x_* and the points in \mathcal{A} , and $K_{\mathcal{A}} = \kappa(X_{\mathcal{A}}, X_{\mathcal{A}})$ is the covariance matrix of the points in \mathcal{A} . After the gradient has been calculated for each potential training point in $\bar{\mathcal{A}}$, we calculate the norm of the gradients, and choose the point with the highest gradient norm to be added to the training set.

Maximum variance: Besides the predictive mean, the variance can be also calculated for any x_* test point. The variance is an indicator of the uncertainty of the GP, and thus is a suitable query function for training point selection. According to Rasmussen and Williams (2006), for a single test point the variance can be written as

$$\sigma_{x_*|\mathcal{A}}^2 = \kappa(x_*, x_*) - k_*^\top K_{\mathcal{A}}^{-1} k_*. \quad (6)$$

Similarly as before, the potential training points can be ranked based on their variance values, and the one with the highest variance is added to the training set.

Mutual information: In the training point selection process, our goal is to select the element of \mathcal{A} such that it reduces the uncertainty of the predictions in $\bar{\mathcal{A}}$ the most. According to Krause et al. (2008), this is equivalent to finding the set \mathcal{A} that maximizes the mutual information $\text{MI}(\mathcal{A}) = I(\mathcal{A}, \bar{\mathcal{A}})$. Using a greedy selection approach, we sequentially choose the points which maximize the increment of the mutual information

$$\arg \max_{x \in \bar{\mathcal{A}}} \text{MI}(x \cup \mathcal{A}) - \text{MI}(\mathcal{A}), \quad (7)$$

This can be simplified to

$$\arg \max_{x \in \bar{\mathcal{A}}} H(x|\mathcal{A}) - H(x|\bar{\mathcal{A}}), \quad (8)$$

where $H(x|\mathcal{A})$ is the entropy of x conditioned on the elements of \mathcal{A} , and can be expressed as a function of the variance,

$$H(x|\mathcal{A}) = \frac{1}{2} \log(\sigma_{x|\mathcal{A}}^2) + \frac{1}{2} (\log(2\pi) + 1). \quad (9)$$

The calculation is analogous for $H(x|\bar{\mathcal{A}})$ with the replacement of \mathcal{A} with $\bar{\mathcal{A}}$. For the details of the derivation and the technical aspects of the selection algorithm see (Krause et al., 2008).

Note that it is possible to use these selection concepts by themselves, but the combined use of them with appropriately chosen weights (w_v, w_g, w_{mi}) ≥ 0 , $w_v + w_g + w_{mi} = 1$ can be also beneficial. The pseudo-code of the active learning algorithm is summarized in Algorithm 2. A comparison of different selection methods is shown in Figure 1.

4.4 Step III: Control refinement in closed loop

After the augmentation of the training set it is still possible that at certain points of the state space the approximation error of the GP exceeds the allowed tolerance. (In our algorithm, the threshold is set a-priori by analysing the robustness properties of the NMPC algorithm.) To ensure that all the control relevant points are included in the training set, we propose the following refinement strategy. First, a large number of initial states are generated randomly within the bounds of the state constraint. For each initial

Algorithm 2 Training point selection by using active learning

Input: Gaussian process model GP, training set \mathcal{T} , set of potential and current training point locations $\bar{\mathcal{A}}, \mathcal{A}$, weights w_v, w_g, w_{mi} , number of training points to be added j

Output: GP, \mathcal{T}

- 1: **for** $i = 1$ to j **do**
- 2: **for** $x \in \bar{\mathcal{A}}$ **do**
- 3: $\delta_x^v \leftarrow \sigma_{x|\mathcal{A}}^2$
- 4: $\delta_x^g \leftarrow \|\nabla f_x\|$
- 5: $\delta_x^{mi} \leftarrow \sigma_{x|\mathcal{A}}^2 / \sigma_{x|\bar{\mathcal{A}}}^2$
- 6: $\delta_{\max}^v \leftarrow \max_{x \in \bar{\mathcal{A}}} \delta_x^v$
- 7: $\delta_{\max}^g \leftarrow \max_{x \in \bar{\mathcal{A}}} \delta_x^g$
- 8: $\delta_{\max}^{mi} \leftarrow \max_{x \in \bar{\mathcal{A}}} \delta_x^{mi}$
- 9: $x_* \leftarrow \arg \max_{x \in \bar{\mathcal{A}}} (w_v \delta_x^v / \delta_{\max}^v + w_g \delta_x^g / \delta_{\max}^g + w_{mi} \delta_x^{mi} / \delta_{\max}^{mi})$
- 10: $\mathcal{A} \leftarrow \mathcal{A} \cup x_*$
- 11: $\bar{\mathcal{A}} \leftarrow \bar{\mathcal{A}} \setminus x_*$
- 12: $u_{\text{MPC}} \leftarrow \text{calculate optimal MPC input for } x_*$
- 13: $\mathcal{T} \leftarrow \mathcal{T} \cup (x_*, u_{\text{MPC}})$
- 14: GP \leftarrow retrain GP with \mathcal{T}

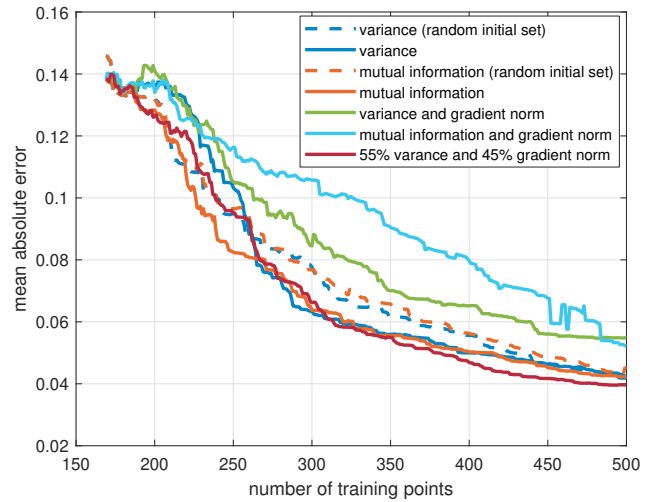


Fig. 1. The comparison of various active learning methods and their effect on the global absolute mean approximation error computed on \mathcal{V} . The criterion functions to be maximized are indicated in the legend for each curve. The curves with solid lines share the same initial training set with training points placed on an equidistant grid, whereas the curves with dashed lines also share an initial training set with randomly placed training points.

state, the system is simulated until it reaches the terminal set. During the simulation, both the NMPC algorithm and the GP are evaluated to calculate both the optimal input value u_{MPC} and its approximation u_{GP} . At each step the difference of the optimal and the approximated input is checked: if it exceeds a certain threshold ε , then the current state and optimal input pair (x, u_{MPC}) is included in the training set and the GP is retrained. After a sufficient number of simulations, this method should result in a GP model with a finalized training set, which is able to fulfill the control task with near-optimal control inputs (within the specified tolerance bounds). Guaranteed robustness against the approximation error can be achieved by choosing a suitable small tolerance ε and applying constraint

tightening in the MPC design. One possible procedure is proposed in Hertneck et al. (2018). The pseudo-code of the refinement method is summarized in Algorithm 3.

Algorithm 3 Control oriented learning with simulation data

Input: Gaussian process model GP, training set \mathcal{T} , system matrices A and B , terminal set, error tolerance ε , number of simulations j
Output: GP, \mathcal{T}

```

1: for  $i = 1$  to  $j$  do
2:    $x \leftarrow$  assign random value within constraints
3:    $\mathcal{T}_+ \leftarrow$  initialize empty set
4:   while  $x$  is outside the terminal set do
5:      $u_{\text{MPC}} \leftarrow$  calculate optimal MPC input
6:      $u_{\text{GP}} \leftarrow$  infer GP at test point  $x$ 
7:     if  $|u_{\text{MPC}} - u_{\text{GP}}| \leq \varepsilon$  then
8:        $u \leftarrow u_{\text{GP}}$ 
9:     else
10:       $\mathcal{T}_+ \leftarrow \mathcal{T}_+ \cup (x, u_{\text{MPC}})$ 
11:       $u \leftarrow u_{\text{MPC}}$ 
12:       $x \leftarrow Ax + Bu$ 
13:     $\mathcal{T} \leftarrow \mathcal{T} \cup \mathcal{T}_+$ 
14:    GP  $\leftarrow$  retrain GP with the new training set

```

5. APPLICATION EXAMPLE: CONTROL DESIGN FOR A SIMPLIFIED HELICOPTER MODEL

In this section the approximate NMPC design procedure is presented on an application example.

Consider the simplified 2D helicopter model from Cannon et al. (2011).

$$\ddot{y} = (u_1 + g) \sin(\alpha), \quad \ddot{z} = (u_1 + g) \cos(\alpha) - g, \quad \ddot{\alpha} = u_2 \quad (10)$$

where y , z and α denote the position and orientation of the helicopter. The net thrust and moment acting on the helicopter are proportional to the inputs u_1 and u_2 , respectively. The control task is to drive the helicopter from any arbitrary initial hovering state $x_0 = [y_0 \ z_0 \ 0 \ 0 \ 0 \ 0]^\top$ to the terminal set placed at the origin. The constraints of the system are $|y| \leq 1$, $|z| \leq 1$, $|\dot{y}| \leq 0.5$, $|\dot{z}| \leq 0.5$, $|\alpha| \leq 0.3$, $|\dot{\alpha}| \leq 0.5$, $|u_1| \leq 10$, $|u_2| \leq 10$, with cost weight matrices $Q = \text{diag}(0.1, 0.1, 1, 1, 10, 1)$, $R = \text{diag}(10^{-4}, 10^{-3})$. The LPV form of the system is formulated as

$$\begin{bmatrix} \dot{y} \\ \dot{z} \\ \dot{y} \\ \dot{z} \\ \dot{\alpha} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & g\rho_1 & 0 \\ 0 & 0 & 0 & 0 & g\rho_2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} y \\ z \\ \dot{y} \\ \dot{z} \\ \alpha \\ \dot{\alpha} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \rho_3 & 0 \\ \rho_4 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \quad (11)$$

$$= A(\rho)x + B(\rho)u,$$

$$\rho_1 = \frac{\sin(\alpha)}{\alpha}, \quad \rho_2 = \frac{\cos(\alpha) - 1}{\alpha}, \quad \rho_3 = \sin(\alpha), \quad \rho_4 = \cos(\alpha).$$

For control design, the continuous model was discretized by Euler's method:

$$x_{k+1} = (I + hA(\rho_k))x_k + hBu_k, \quad (12)$$

where $h = 0.1$ s is the sampling time. In the GP model, due to the expected discontinuities of the explicit MPC law, a *Matern* class kernel was used (Rasmussen and Williams (2006)), namely

$$\kappa(x_p, x_q) = \sigma_f^2 (1 + \sqrt{3}r) \exp(-\sqrt{3}r), \quad (13)$$

$$r = (x_p - x_q)^\top M (x_p - x_q),$$

where $M = \text{diag}(l_i^{-2})$, $i = 1, \dots, 6$ and $\sigma_f, l_1, l_2, \dots, l_6$ are the hyperparameters of the model.

Since the system has 2 inputs, two separate GP models were required for the approximation, however, during our calculations we used one common training set for the two models.

First, a validation set was constructed to be used for qualifying the approximation. For this, we calculated the optimal input values on an equidistant grid with 0.1 spacing from -1 to 1 in y and z , and with constant 0 in the other 4 state variables. We also included in the validation set all the states along the predicted trajectories with their corresponding input values on an $N = 30$ prediction horizon obtained from the NMPC algorithm. This resulted in a set that fairly represents all the states the system can reach while executing the control task.

Then the set of potential training point locations \mathcal{V} was obtained by gridding the whole state space (with 0.2 spacing along all 6 dimensions) and selecting only those points that lay inside the convex hull of the points in the validation set. The initial training set was assembled by using a sparser grid with 0.5 spacing along y and z and 0.2 spacing along the remaining 4 dimensions. This resulted in 257 initial training points. Afterwards, the active learning algorithm increased the number of training points to 945. The selection criteria were the maximum variance and the maximum gradient norm, with weights $w_v = 0.55$ and $w_g = 0.45$, respectively. In each iteration, one point was selected for each GP, and both points (if different), were added to the same, shared training set. Finally, the refinement process with tolerance $\varepsilon = 0.1$ increased the number of training points to 990. Figure 2 shows the average absolute error of the approximations during the training process, compared to the validation set described earlier. Figure 3 shows that the GP models were able to learn the control input. In the figure, 500 trajectories are shown which were generated by controlling the system with the fully trained GPs alone.

During the simulation the average runtime of the 2 GP's inference together was 0.2321 s, whereas the average runtime of the NMPC algorithm was 1.1880 s. This shows, that after a certain level of system complexity, the real time evaluation of the NMPC algorithm becomes intractable, while the time required to GP inference does not increase significantly.

Finally, the GP regression was compared with linear interpolation. The interpolation was evaluated on the same dataset over which the GP is defined. By examining the control performance from the 500 initial conditions depicted in Fig. 3, we found that GP performed better in several aspects: first, the computation of the control input by interpolation was approximately 20 times slower compared to the GP (we used the built-in interpolation routines of Matlab). Second, the dataset was proved to be inadequate for linear regression: in some cases the linear interpolation produced high approximation error resulting in poor control performance or unstable closed-loop behavior.

6. CONCLUSION

In this paper a practical approach is presented for constructing a learning based approximate explicit LPV

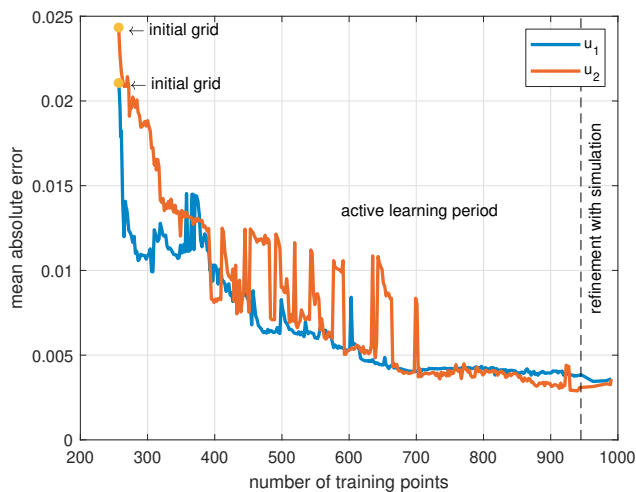


Fig. 2. The average absolute error of the approximation of the optimal input for the helicopter, compared to the validation set. The jumps in the average absolute error of u_2 are the result of the parameters in the GP model converging to different local optima during the training.

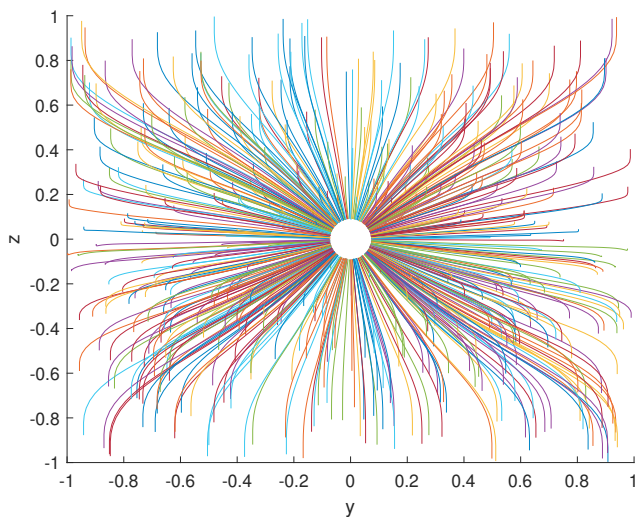


Fig. 3. Trajectories of the helicopter model controlled using the GP based explicit controller from random initial states $x_0 = [y_0 \ z_0 \ 0 \ 0 \ 0 \ 0]^T$ to the terminal set.

model predictive controller for nonlinear systems. Special attention is paid to the systematic collection of relevant training samples. It has been shown that active learning methods and closed loop simulations can be successfully blended in an efficient training point selection algorithm. The applicability of the procedure has been demonstrated by designing an approximate predictive controller for a nonlinear mechanical system.

REFERENCES

- Boef, den P. (2019). *Frequency Domain LPV System Identification Using Local Data*. Master’s thesis, Eindhoven University of Technology.
- Borrelli, F., Bemporad, A., and Morari, M. (2019). *Predictive Control for Linear and Hybrid Systems*. Cambridge University Press.
- Brochu, E., Cora, V.M., and Freitas, N.D. (2010). A tutorial on Bayesian optimization of expensive cost

- functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv:1012.2599*.
- Canale, M., Fagiano, L., and Milanese, M. (2010). Efficient model predictive control for nonlinear systems via function approximation techniques. *IEEE Transactions on Automatic Control*, 55(8), 1911–1916.
- Cannon, M., Buerger, J., Kouvaritakis, B., and Rakovic, S. (2011). Robust tubes in nonlinear model predictive control. *IEEE Transactions on Automatic Control*, 56(8), 1942–1947.
- Cisneros, P.S.G., Sridharan, A., and Werner, H. (2018). Constrained Predictive Control of a Robotic Manipulator using quasi-LPV Representations. *IFAC-PapersOnLine*, 51(26), 118–123.
- Cisneros, P.S.G., Voss, S., and Werner, H. (2016). Efficient Nonlinear Model Predictive Control via quasi-LPV Representation. In *Proceedings of Conference on Decision and Control*, 3216–3221.
- Cseko, L.H., Kvasnica, M., and Lantos, B. (2015). Explicit MPC-Based RBF Neural Network Controller Design With Discrete-Time Actual Kalman Filter for Semiactive Suspension. *IEEE Transactions on Control Systems Technology*, 23(5), 1736–1753.
- Darwish, M.A.H. (2017). *Bayesian identification of linear dynamic systems*. Ph.D. thesis, Eindhoven University of Technology.
- Dominguez, L.F., Narciso, D.A., and Pistikopoulos, E.N. (2010). Recent advances in multiparametric nonlinear programming. *Computers & Chemical Engineering*, 34(5), 707–716.
- Hertneck, M., Kohler, J., Trimpe, S., and Allgower, F. (2018). Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3), 543–548.
- Johansen, T.A. (2003). Approximate explicit receding horizon control of constrained nonlinear systems. *Automatica*, 40(2), 293–300.
- Krause, A., Singh, A., and Guestrin, C. (2008). Near-optimal sensor placements in Gaussian processes: Theory, efficient algorithms and empirical studies. *Journal of Machine Learning Research*, 9(Feb), 235–284.
- Liu, M., Chowdhary, G., da silva, B.C., Liu, S., and How, J.P. (2018). Gaussian Processes for Learning and Control - A tutorial with examples. *IEEE Control Systems Magazine*, 38(5), 53 – 86.
- Parisini, T. and Zoppoli, R. (1995). A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10), 1443–1451.
- Rakovic, S.V. and Levine, W.S. (eds.) (2019). *Handbook of Model Predictive Control*. Birkhuser.
- Rasmussen, C.E. and Williams, C.K.I. (2006). *Gaussian Processes for Machine Learning*. MIT Press.
- Sharif, B. (2018). *Linear Parameter Varying Control of Nonlinear Systems*. Master’s thesis, Eindhoven University of Technology.