

Online learning with stability guarantees: A memory-based real-time model predictive controller*

Lukas Schwenkel¹, Meriem Gharbi¹, Sebastian Trimpe^{2,3}, and Christian Ebenbauer¹

¹*Institute for Systems Theory and Automatic Control, University of Stuttgart, Germany*

²*Institute for Data Science in Mechanical Engineering, RWTH Aachen University, Germany*

³*Intelligent Control Systems Group, Max Planck Institute for Intelligent Systems, Stuttgart, Germany*

Abstract. We propose and analyze a real-time model predictive control (MPC) scheme that utilizes stored data to improve its performance by learning the value function online with stability guarantees. For linear and nonlinear systems, a learning method is presented that makes use of basic analytic properties of the cost function and is proven to learn the MPC control law and the value function on the limit set of the closed-loop state trajectory. The main idea is to generate a smart warm start based on historical data that improves future data points and thus future warm starts. We show that these warm starts are asymptotically exact and converge to the solution of the MPC optimization problem. Thereby, the suboptimality of the applied control input resulting from the real-time requirements vanishes over time. Numerical show that existing real-time MPC schemes can be improved by storing optimizating the proposed ng scheme.

1. Introduction

Model predictive control (MPC) is a control strategy that solves at each sampling instant a finite horizon open-loop optimal control problem (see [33]). As a consequence, at every sampling instant an input sequence and its resulting state trajectory are computed for the whole prediction horizon, thus continuously generating large amounts of optimization data. However, only the first portion of the computed input sequence is applied to the system and typically the remaining part is not stored. This is wasteful and contradicts our human intuition to memorize our decisions when we are facing recurring problems or tasks

* This article is an extended version of [35] including all proofs, an application example, and a detailed description of the used algorithm.

Email addresses: schwenkel@ist.uni-stuttgart.de (Lukas Schwenkel), gharbi@ist.uni-stuttgart.de (Meriem Gharbi), trimpe@dsme.rwth-aachen.de (Sebastian Trimpe), ce@ist.uni-stuttgart.de (Christian Ebenbauer). The authors thank the International Max Planck Research School for Intelligent Systems (IMPRS-IS) for supporting Lukas Schwenkel. This work was supported in part by the Cyber Valley Initiative and the Max Planck Society.

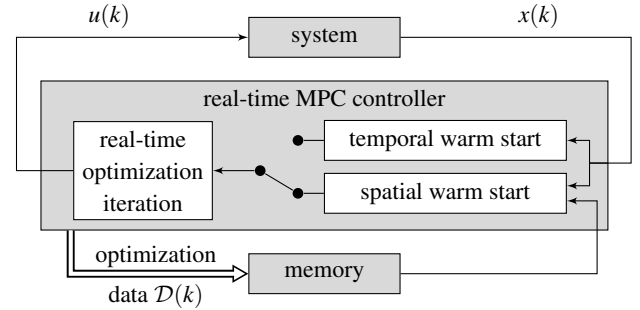


Figure 1. Illustration of the proposed online-learning real-time MPC scheme. The standard real-time MPC controller consists of a (temporal) warm start solution and an optimization iteration. To improve control performance, past optimization data of the MPC are stored in memory. From these, a spatial warm start is constructed, which improves with more data. By taking in each iteration the warm start that results in a lower cost function value, the stability properties of the original real-time MPC are retained.

in order to successively improve them. Motivated by this, the key question addressed in this article is how to leverage on optimization data by introducing memory and online learning in real-time MPC algorithms in order to improve their performance.

In this work, we will store the previously computed input sequences and utilize them to learn the solution of the MPC optimization problem online, see Fig. 1. The proposed approach specifically targets real-time MPC where, due to a lack of computation time, the MPC optimization problem cannot be solved exactly and thus, suboptimal solutions are applied to the system. Hence, there is a need to learn the optimal solution.

The proposed online learning is based on the idea to store optimization data and leverage it for subsequent optimizations: whenever a new input sequence has to be computed close to a point that was visited before, the optimization iteration is initialized with the previously computed input sequence. This

way, we can use the available computation time to refine this suboptimal input sequence instead of starting the optimization from scratch. At recurrent points, we expect to approach the optimal solution since we improve a suboptimal one over and over again. By combining these data-based improvements with the theory of real-time MPC, we will be able to show that online learning is feasible for recurrent points with inherent stability guarantees.

Contributions The main contributions of this work are as follows. We present a real-time MPC scheme based on [19] that allows for online learning to improve control performance and that comes with inherent stability guarantees independent of the chosen learning method. Further, we introduce novel learning methods tailored to the real-time MPC scheme for both linear and nonlinear systems. The learning methods exploit analytic properties of the cost function, namely convexity in the linear case and Lipschitz continuity in the nonlinear case, in order to upper bound the value function. This bound is shown to converge to the value function on the ω -limit set of the closed-loop state trajectory. Thereby, the input applied to the system converges to the optimal feedback policy. The result is an online method that learns the value function and the optimal MPC law asymptotically over the ω -limit set. Moreover, we present several examples, which illustrate and confirm the results and are used to discuss practical issues and how to circumvent them. In particular, we demonstrate how learning and applying the learned control law can be parallelized and executed on different time scales, whereby learning itself does not have to be executed in real time.

Related work This work combines real-time MPC with data-based approaches, both of which being active areas of research. The proposed online learning approach builds on top of a real-time MPC scheme and especially its stability properties. In real-time MPC, there are several results on stability, which all take the suboptimality of the applied input sequence explicitly into account. The work [36] seeks for a feasible solution without executing any optimization steps, which makes this method unsuitable as basis for learning the optimal solution from its data. The works [2], [5], and [28] optimize the input sequence until a certain accuracy is achieved. Hence, to improve their performance by including learning, we would need to adjust this accuracy according to the learning progress, which is rather difficult to do. That is why we assume a method for the proposed learning approach that optimizes until a certain computation time is reached like those in [14], [19], and [41]. In particular, the real-time MPC scheme presented in [19] will be the basis for the online learning method we develop herein. In [19], the cost function is established as a Lyapunov function for the coupled system optimizer dynamic, which allows for a straightforward extension of the scheme to learning while preserving stability.

Learning approaches in MPC are becoming increasingly popular. Different ways for incorporating data in MPC have been suggested in literature and can be summed up in two main

categories: (i) system input-output data to learn the model; and (ii) sampling data of the MPC policy to approximate it with an explicit control law. Model learning (i) is an important topic of current research since the accuracy of the model has significant impact on the control performance. The model can be learned offline, as for example in [10], [24], [26], [31], and [39], but also online, [3], [8], [13], [29], or first learned offline and then refined online [20], [27]. Nevertheless, only few of these works, namely [3], [10], [13], and [27], establish stability results for their methods. The learning of an explicit MPC control law (ii) is done offline in order to substitute the online optimization in MPC with an online evaluation of the explicit control law. The majority of works in this direction employ neural networks as approximate controllers [1], [12], [16], [21], [22], [30], [32] and [42], while support vector machines are used in [11], and the learning problem is formulated as quadratically constrained quadratic program in [15]. Out of these works, only [15] and [21] can guarantee stability. Since the evaluation of e.g. a neural network is typically much faster than solving the MPC optimization problem, these methods are suited for real-time applications. In contrast to our work, these methods learn the control law offline. This implies that they have a fixed approximation error and cannot improve online, while our method approaches optimality by learning online. By their design, offline methods cannot adapt to changes and need training data beforehand, while our method can be started without prior data.

There are few other works on learning in MPC that do not fit in the two categories. One of them is [34], in which past inputs and the past state trajectory are used for iterative tasks to learn a terminal set and terminal cost of a finite horizon MPC controller such that infinite horizon performance is optimized while stability is ensured. In [23], a neural network is trained offline to identify active constraints in order to warm start an active set algorithm in embedded MPC while guaranteeing online stability. Although this work also considers learning warm starts, it is conceptually different from ours, since it learns offline and does not improve performance by decreasing the cost.

This article proposes a novel way of leveraging data in MPC, which does not match any of the aforementioned categories and works. Here, we use internal data of the *real-time* MPC algorithm, specifically predicted state and input sequences. With this data, we improve the suboptimal real-time controller over time by providing better warm start solutions based on past optimization solutions. To the best of the authors' knowledge, this is the first work on learning in real-time MPC that utilizes this internal data.

Outline This article continues with introducing preliminary results, the problem formulation, and the core idea of this work in Section 2. Sections 3 and 4 then contain the main results of this article: the online learning method is developed for linear and nonlinear MPC, respectively, and their properties are analyzed in theory and illustrated through numerical examples. Section 5 underlines the relevance of the results by an application example. Our conclusions, stated in Section 6, complete

the article.

Notation Throughout the article, we will use \mathbb{R}_+ , \mathbb{R}_{++} and \mathbb{N}_+ to denote the sets of non-negative real, strictly positive real, and strictly positive natural numbers. The set of positive definite matrices of dimension $n \in \mathbb{N}_+$ is denoted with \mathbb{S}_{++}^n . For $M \in \mathbb{S}_{++}$, we define $\|x\|_M := (x^\top M x)^{1/2}$. A continuous function $\alpha : \mathbb{R}_+ \rightarrow \mathbb{R}_+$ with $\alpha(0) = 0$ is a \mathcal{K}_∞ -function if it is strictly increasing and $\alpha(s) \rightarrow \infty$ as $s \rightarrow \infty$. The interior of a set $A \subset \mathbb{R}^n$ is denoted $\text{int}A$ and the convex hull is $\text{conv}A$. For a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ the epigraph is

$$\text{epi } f = \{(x, \mu) \in \mathbb{R}^n \times \mathbb{R} \mid f(x) \leq \mu\}. \quad (1)$$

2. Problem setting: Learning in real-time MPC

In this section, we set the stage for developing the main results of this article. We first introduce the real-time MPC framework [19], in which our learning method will be embedded. We then present the main idea of how to leverage data in this setting, make the learning problem precise, and provide preliminary stability results.

2.1. Real-time MPC framework

We consider the control of time-invariant discrete-time systems of the form

$$x(k+1) = f(x(k), u(k)) + w(k) \quad (2)$$

where $x(k) \in \mathbb{R}^n$ denotes the state vector, $u(k) \in \mathbb{R}^m$ the control input vector, and $w(k) \in \mathbb{R}^n$ some external signal vector, all at time instant $k \in \mathbb{N}$. We will consider both linear and nonlinear system dynamics $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^n$. In MPC, a typical control task is regulation of the system state to the origin while minimizing some cost function. This is handled by solving at each time instant k an open-loop optimal control problem of the form

$$\begin{aligned} J_N(U, x(k)) &= \sum_{j=0}^{N-1} l(x_j, u_j) + F(x_N) \\ J_N^*(x(k)) &= \min_U J_N(U, x(k)) \\ \text{s.t. } x_{j+1} &= f(x_j, u_j), \quad j = 0, \dots, N-1 \\ x_0 &= x(k) \end{aligned} \quad (3)$$

where $U = [u_0^\top, \dots, u_{N-1}^\top]^\top \in \mathbb{R}^{Nm}$ denotes the stacked control inputs over the finite prediction horizon $N \in \mathbb{N}_+$, and $l : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}_+$ and $F : \mathbb{R}^n \rightarrow \mathbb{R}_+$ denote the stage and terminal costs, respectively. Since the external signal $w(k)$ is unknown over the prediction horizon, we assume $w(k) = 0$ and use the nominal system to predict x_j . We will call J_N^* the *value function* and J_N the *cost function*. We assume that state and input constraints are taken into account by a barrier or penalty term in the stage and terminal cost.

Usually in MPC, it is assumed that the system dynamics and the optimization algorithm evolve on different time scales such that the convergence time of the algorithm can be neglected and an instantaneous solution to (3) is available. In real-time iteration schemes, this often unrealistic assumption is dropped and the optimization algorithm is interpreted as a system with its own dynamics. These are coupled with (2) and are given by

$$U(k+1) = \Phi^{i_T(k)}(U(k), x(k)) \quad (4a)$$

$$u(k+1) = \Pi_0 U(k+1) \quad (4b)$$

where $\Phi^{i_T(k)} : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}^{Nm}$ represents the optimization algorithm and $\Pi_0 = [I_m \ 0 \ \dots \ 0] \in \mathbb{R}^{m \times Nm}$ is a projection matrix selecting the input to be actually applied. We divide the optimization algorithm $\Phi^{i_T(k)}$ into a warm start operator $\Psi_{\text{tw}} : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}^{Nm}$ and an optimization update operator $\Psi_o : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}^{Nm}$, which is iterated $i_T(k) \in \mathbb{N}_+$ times

$$\Phi^0(U, x) = \Psi_{\text{tw}}(U, x) \quad (5a)$$

$$\Phi^i(U, x) = \Psi_o(\Phi^{i-1}(U, x), f(x, \Pi_0 U)). \quad (5b)$$

In this way, an input for the nominal next state $f(x, \Pi_0 U)$ is determined.

The warm start operator is typically generated by a time shift of the previous input sequence appended with some local control law [19]. Throughout the article, we will therefore refer to Ψ_{tw} as *temporal* warm start operator. If the closed loop is stable for *any* number of optimization algorithm iterations $i_T(k)$, the real-time scheme is also called *anytime* MPC, [5], [19]. A generic result on nominal closed-loop stability of an anytime iteration scheme is proven in [19] and stated in the following theorem.

Theorem 1. *Consider the real-time MPC scheme introduced in (2)–(5) with $w(k) = 0$ and assume that the stage and terminal costs l and F are positive definite. Further assume the existence of $\underline{\alpha}, \bar{\alpha} \in \mathcal{K}_\infty$ such that*

$$\underline{\alpha}(\|(U, x)\|) \leq J_N(U, x) \leq \bar{\alpha}(\|(U, x)\|) \quad (6)$$

and assume that Ψ_{tw} and Ψ_o fulfill

$$J_N(\Psi_{\text{tw}}(U, x), f(x, \Pi_0 U)) - J_N(U, x) \leq -l(x, \Pi_0 U) \quad (7a)$$

$$J_N(\Psi_o(U, x), x) - J_N(U, x) \leq -\gamma(U, x) \quad (7b)$$

for all $(U, x) \in \mathbb{R}^{Nm} \times \mathbb{R}^n$, where $\gamma : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ is a continuous function with $\gamma(U, x) = 0 \Leftrightarrow J_N(U, x) = J_N^*(x)$. Then, for any sequence $\{i_T(0), i_T(1), \dots\}$, the origin $(U, x) = (0, 0)$ is globally asymptotically stable.

This theorem is stated in [19] for linear systems using a relaxed barrier function formalism to ensure the assumptions. For nonlinear systems, this generic result still applies, however, it is much more challenging to ensure (6) and (7) in nonlinear MPC. In [14], a weaker stability result for a slightly different nonlinear real-time MPC framework is presented, yet without satisfying the assumptions (6) and (7).

2.2. Problem formulation and main idea

When the system dynamics and the optimization algorithm operate on similar time scales, there is typically only time for a few optimization iterations until the system requires the next input, i.e. $i_T(k)$ is small. Hence the input might be far from optimality resulting in unsatisfying controller performance. To solve this issue, we will leverage the data of *all* the previously computed input sequences, which are usually discarded (except the last input sequence generating a temporal warm start). We store all past input sequences, and if the system state arrives close to a point in the state space that has been visited before, we use this previously calculated input sequence to generate a warm start solution. The main challenges of this approach are to make ‘close to a point’ mathematically precise and to prove convergence. Hence, we have to choose which stored input sequence (or combinations of multiple ones) to take as warm start solution at a given location such that the learning scheme asymptotically recovers the optimal policy at recurrent points of the closed loop trajectory.

In more detail, we will denote the warm start generated from the optimization data $\mathcal{D}(k)$ by $\Psi_{\text{sw}}(\cdot, \mathcal{D}(k)) : \mathbb{R}^n \rightarrow \mathbb{R}^{N_m}$ and call it *spatial* to distinguish it from the temporal one. The stored data $\mathcal{D}(k)$ includes: the input sequence $U(j)$ for each time $1 \leq j \leq k$, the point in state space at which it was calculated $f(x(j-1), \Pi_0 U(j-1))$, and the cost function value that was achieved $J_N(U(j), f(x(j-1), \Pi_0 U(j-1)))$; that is

$$\mathcal{D}(k) = \mathcal{D}(k-1) \cup \{(z_k, J_N(z_k))\}, \quad \mathcal{D}(0) = \emptyset \quad (8a)$$

with $z_k = (U(k), f(x(k-1), \Pi_0 U(k-1)))$. For the subsequent analysis, we introduce the notation

$$\mathcal{D}_x := \{x \in \mathbb{R}^n \mid \exists (U, x, J) \in \mathcal{D}\} \quad (8b)$$

$$\mathcal{D}_{xJ} := \{(x, J) \in \mathbb{R}^n \times \mathbb{R}_+ \mid \exists (U, x, J) \in \mathcal{D}\} \quad (8c)$$

$$\mathcal{D}_U := \{U \in \mathbb{R}^{N_m} \mid \exists (U, x, J) \in \mathcal{D}\} \quad (8d)$$

and we denote an approximation of $J_N^*(x)$ based on the collected data $\mathcal{D}(k)$ by $J_N^a(x, \mathcal{D}(k))$.

In a nominal MPC stabilization problem, we cannot expect that the system state repeatedly arrives at the same points in the state space except at the origin, where the optimal input is trivial. In a real-world scenario, however, there are disturbances, periodic operation conditions, set point changes, or reference signals such that more points can be visited several times. Thus, *learning* the optimal control at these points is a meaningful task. We generically model such situations with the signal $w(k)$, which influences the shape of the ω -limit set of the sequence of points for which an input is computed,

$$\Omega = \{y \in \mathbb{R}^n \mid \exists k_i \rightarrow \infty : f(x(k_i), \Pi_0 U(k_i)) \rightarrow y\}. \quad (9)$$

A point in Ω is called *limit* or *recurrent* point and is reached infinitely often arbitrarily closely. Only at such points can we expect learning the optimal policy to be possible because the optimization iteration usually converges asymptotically to the

optimum and thus needs an infinite number of iterations in general.

With this, we can now make the objective of this work precise. We aim to design the spatial warm start operator Ψ_{sw} such that it will converge to the optimal policy for all $x \in \Omega$ as $k \rightarrow \infty$, i.e.

$$\forall x \in \Omega : \lim_{k \rightarrow \infty} J_N(\Psi_{\text{sw}}(x, \mathcal{D}(k)), x) = J_N^*(x). \quad (10)$$

Hence, we want to learn the value function J_N^* on Ω and the corresponding optimal control input.

While learning the optimal control, we do not want to jeopardize stability. Hence, we need to make sure that the warm start solution that is used to initialize the optimization iteration satisfies (7a). We can achieve stability by exploiting the temporal warm start (which is shown to be stabilizing) and by only applying the spatial warm start when it yields a lower cost. That is, we replace (5a) with the new warm start operator

$$\Phi^0(U, x, \mathcal{D}) = \begin{cases} \Psi_{\text{tw}}(U, x) & J_{Nt} < J_{Ns} \\ \Psi_{\text{sw}}(f(x, \Pi_0 U), \mathcal{D}) & \text{else,} \end{cases} \quad (11a)$$

$$\text{with } J_{Nt} = J_N(\Psi_{\text{tw}}(U, x), f(x, \Pi_0 U)) \quad (11b)$$

$$J_{Ns} = J_N(\Psi_{\text{sw}}(f(x, \Pi_0 U), \mathcal{D}), f(x, \Pi_0 U)). \quad (11c)$$

Through this intuitive approach, the resulting online learning scheme is inherently stable as stated in the following corollary.

Corollary 2. *Consider the real-time MPC scheme introduced in (2)–(5), (8), (11) with $w(k) = 0$ and assume (6), (7). Then the origin $(U, x) = (0, 0)$ is globally asymptotically stable for all spatial warm start operators Ψ_{sw} .*

Remark 3. The stability result considers the nominal system with $w(k) = 0$ since stability of the origin can not be achieved with a non-vanishing $w(k)$. The learning problem, however, is trivial for $\Omega = \{0\}$, thus, we require $w(k_i) \neq 0$ for some $k_i \rightarrow \infty$ to render $\Omega \neq \{0\}$. Nevertheless, nominal stability is meaningful and essential for the learning task.

- As a first example, consider a scenario as typically studied in iterative learning control, where, after some finite time period, the system is reset to a new initial state. Let the set of initial states be \mathbb{X}_0 . Then, if we model $w(k_i) = -f(x_{k_i}, u_{k_i}) + \bar{x}$, $\bar{x} \in \mathbb{X}_0$, for a sequence of reset times $\{k_i\} \rightarrow \infty$ and $w(k) = 0$ for $k \neq k_i$, this would correspond to an iterative learning scenario for multiple initial states in \mathbb{X}_0 , where the goal is to steer $\bar{x} \in \mathbb{X}_0$ close to the origin. While executing an iteration it is $w(k) = 0$, hence, global asymptotic stability of the nominal closed loop system is essential to guarantee that the controller does the right thing (not growing unbounded and approaching the origin). This iterative learning interpretation is also discussed in the unicycle example in Section 4 of the paper.
- As a second example, let $w(k)$ be Gaussian noise. Then $w(k)$ would render the set of recurrent points nonempty

(presumably the whole state space). Nevertheless, the stability result still makes sense in terms of expectations. Since a complex stochastic analysis is beyond the scope of this work, we do not explicitly consider this case. Still, we can think of $w(k)$ as a deterministic noise signal or a realization of a stochastic process. In general, feedback stabilization is essentially only meaningful in the presence of disturbances. Often, in nominal MPC, disturbances are associated to nonzero initial states.

- As a third example, consider a tracking of references or set points. These are often based on a nominal stabilizing controller, which is used to stabilize different set points. When the set point changes, this can be interpreted as a new initial condition and modeled by the signal $w(k)$ as in the first example. This scenario is treated in an Application example in Section 5.

In general, we assume that $w(k)$ is of such nature that the closed-loop trajectory $(x(k), U(k))$ stays bounded. This ensures that Ω is nonempty due to the Bolzano-Weierstrass theorem and that Ω is approached by $f(x(k), \Pi_0 U(k))$ as $k \rightarrow \infty$.

Overall, the proposed MPC iteration scheme measures at every time instant the current system state, generates the warm starts, chooses the better one, executes the optimization iteration, and stores its data before it finally applies the first portion of the input sequence to the system. A pseudo-code description of this procedure is given in Algorithm 1. The main focus of this article is line 4 and 10 of the algorithm. More specifically, we will design in Section 3 and 4 spatial warm start operators for linear and nonlinear MPC, respectively. Throughout the article, we assume that a temporal warm start satisfying the conditions in Theorem 1 is given.

Algorithm 1 Real-time MPC scheme with learning

```

1: for  $k = 1, 2, \dots$  do
2:   obtain current state:  $x(k)$ 
   generate warm starts:
3:    $U_t = \Psi_{tw}(U(k), x(k))$  (temporal)
4:    $U_s = \Psi_{sw}(f(x(k), \Pi_0 U(k)), \mathcal{D}(k))$  (spatial)
5:    $U_+ = U_t$  if  $J_{Nt} < J_{Ns}$  as per (11) else  $U_+ = U_s$ 
   real-time optimization iteration:
6:   for  $i = 1, 2, \dots, i_T(k)$  do optimizer update
7:      $U_+ = \Psi_o(U_+, f(x(k), \Pi_0 U(k)))$ 
8:   end for
9:    $U(k+1) = U_+$ 
   memorize data:
10:   $\mathcal{D}(k+1) = \mathcal{D}(k) \cup \{(z_k, J_N(z_k))\}$  as per (8)
11:  apply first input:  $u(k+1) = \Pi_0 U(k+1)$ 
12: end for

```

3. Leveraging data in real-time linear MPC

In this section, we assume (2) to be linear

$$f(x, u) = Ax + Bu, \quad (12)$$

with $A \in \mathbb{R}^{n \times n}$, $B \in \mathbb{R}^{n \times m}$, the stage and terminal cost are quadratic, and the polytopic state and input constraints are incorporated in the costs via relaxed logarithmic barrier functions $\hat{B}_x : \mathbb{R}^n \rightarrow \mathbb{R}_+$, $\hat{B}_u : \mathbb{R}^m \rightarrow \mathbb{R}_+$ (see [17] or [18] for definition). Under these assumptions, l and F in (3) become

$$l(x, u) = \|x\|_Q^2 + \|u\|_R^2 + \varepsilon \hat{B}_x(x) + \varepsilon \hat{B}_u(u), \quad (13)$$

$$F(x) = \|x\|_P^2, \quad (14)$$

with the design parameters $\varepsilon \in \mathbb{R}_{++}$, $Q \in \mathbb{S}_{++}^n$ and $R \in \mathbb{S}_{++}^m$, as well as $P \in \mathbb{S}_{++}^n$ resulting from ε , Q , R and the constraints (see [18] for details). In [19], a temporal warm start and an optimization update operator are defined such that the conditions (6) and (7) of Theorem 1 are satisfied. Moreover, it has been shown in [19] that constraint satisfaction can be guaranteed with a finite barrier parameter ε if the relaxation of the logarithmic barrier functions and the initialization of the scheme are chosen suitably. We will refer to this scheme as linear any-time MPC. For this setting, we are going to present a method to learn the optimal policy and the value function in the sense of (10), analyze its convergence properties, and demonstrate the method in an example.

3.1. A spatial warm start based on convexity

The main idea for the spatial warm start generation is to exploit convexity of the cost function and use convex combinations of past data points. More formally, we define the spatial warm start at $\bar{x} \in \text{conv } \mathcal{D}_x(k)$ by

$$(\Psi_{sw}(\bar{x}, \mathcal{D}), \bar{x}, J_N^a(\bar{x}, \mathcal{D})) = \arg \min_{\substack{(U, x, J) \in \text{conv } \mathcal{D} \\ x = \bar{x}}} J. \quad (15)$$

Hereby, we dropped the time dependency of the data $\mathcal{D} = \mathcal{D}(k)$ from (8) for the sake of clarity. Furthermore, \bar{x} denotes the point at which (3) is to be solved and $J_N^a(\cdot, \mathcal{D}(k))$ denotes a convex approximation of the value function $J_N^*(x)$ based on the data $\mathcal{D}(k)$. A graphical illustration of (15) is given in Fig. 2. Notice that (15) is only feasible for $\bar{x} \in \text{conv } \mathcal{D}_x(k)$ and thus the domain of $J_N^a(\cdot, \mathcal{D}(k))$ is $\text{conv } \mathcal{D}_x$. For $\bar{x} \notin \text{conv } \mathcal{D}_x(k)$, we cannot compute a spatial warm start in this fashion and have to take the temporal one.

Geometrically, J_N^a is a piecewise affine function and partitions $\text{conv } \mathcal{D}_x(k)$ into n -simplices, on which it is affine. We will refer to this partition as triangulation of $\text{conv } \mathcal{D}_x(k)$ inspired by topology. For computing the spatial warm start at x , one has to find the n -simplex that contains x , construct the convex combination of its extreme points that leads to x , and combine the inputs corresponding to the extreme points in the same way.

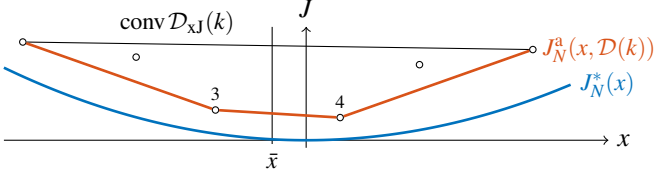


Figure 2. Illustration of the spatial warm start (15) in the (x, J) -domain with data points $D_{xj}(k)$ (\circ). The lower boundary (in orange) of their convex hull is an upper bound $J_N^a(x, D(k))$ of the value function $J_N^*(x)$ (blue), since J_N^* is convex and optimal. The spatial warm start solution at \bar{x} is a convex combination of the inputs from points 3 and 4.

In the remainder of this section we will show that this spatial warm start converges to the optimal policy at recurrent points (9) by showing that $J_N^a(\cdot, D(k))$ upper bounds $J_N(\Psi_{\text{sw}}(x, D(k)), x)$ if J_N is convex (Lemma 4), that J_N is convex (Lemma 5), and that $J_N^a(\cdot, D(k))$ converge to the value function (Theorem 6).

Lemma 4. Let $J_N : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ be convex with respect to (U, x) . Further, let $z_j \in \mathbb{R}^{Nm} \times \mathbb{R}^n$, $j \in \mathbb{N}$ be any sequence, and let $\mathcal{D}(j) \subset \mathbb{R}^{Nm} \times \mathbb{R}^n \times \mathbb{R}_+$ be the corresponding sequence of data sets defined in (8a). Moreover, let $\Psi_{\text{sw}}(\cdot, \mathcal{D}(j)) : \text{conv } \mathcal{D}_x(j) \rightarrow \mathbb{R}^{Nm}$ and $J_N^a(\cdot, \mathcal{D}(j)) : \text{conv } \mathcal{D}_x(j) \rightarrow \mathbb{R}_+$ be defined in (15) and let $k \in \mathbb{N}_+$. Then (i) $J_N(\Psi_{\text{sw}}(x, D(k)), x) \leq J_N^a(x, D(k))$ and (ii) $J_N^a(\cdot, D(k))$ is convex.

For a proof, see Appendix A. In order to apply Lemma 4, convexity of J_N is needed, which is established for the linear anytime MPC by the following lemma.

Lemma 5. The cost function $J_N : \mathbb{R}^{Nm} \times \mathbb{R}^n \rightarrow \mathbb{R}_+$ of linear anytime MPC, i.e. (3) with (12), (13), (14) is convex in (U, x) .

The proof is given in Appendix A. In view of Lemma 4 and Lemma 5, we can show the following convergence theorem, which establishes that we can indeed asymptotically learn the value function and the MPC law in Ω .

Theorem 6. Consider the system (12) controlled by (3)–(8), (11), (13)–(15) and Ω from (9). Then for all $x \in \Omega$ with $x \in \text{int conv } \mathcal{D}_x(k_0)$ for some $k_0 \in \mathbb{N}$ it holds

$$\lim_{k \rightarrow \infty} J_N(\Psi_{\text{sw}}(x, D(k)), x) = J_N^*(x). \quad (16)$$

Proof. Step 1) $J_N^a(\cdot, D(k))$ converges: For $k \geq k_0$, it holds $\text{conv } \mathcal{D}(k_0) \subseteq \text{conv } \mathcal{D}(k)$ and hence J_N^a is defined on $\text{conv } \mathcal{D}(k_0)$ for all $k \geq k_0$. Further, since $\text{conv } \mathcal{D}(k) \supseteq \text{conv } \mathcal{D}(k-1)$ and the minimum over a larger set can only be smaller, we can conclude that $J_N^a(y, D(k)) \leq J_N^a(y, D(k-1))$ for all $y \in \text{conv } \mathcal{D}_x(k_0)$, $k \geq k_0$. In order to apply Lemma 4, J_N has to be convex, which is shown in Lemma 5. Due to (i) in Lemma 4 and the optimality of J_N^* we have

$$J_N^*(y) \leq J_N(\Psi_{\text{sw}}(y, D(k)), y) \leq J_N^a(y, D(k)) \quad (17)$$

for all $y \in \text{conv } \mathcal{D}_x(k_0)$, $k \geq k_0$. Hence, $J_N^a(\cdot, D(k))$ is nonincreasing and bounded from below on $\text{conv } \mathcal{D}_x(k_0)$ and thus converges pointwise $\lim_{k \rightarrow \infty} J_N^a(\cdot, D(k)) = J_N^\infty(\cdot)$ on $\text{conv } \mathcal{D}_x(k_0)$. Due to (ii) in Lemma 4, this is a sequence of convex functions $J_N(\cdot, D(k))$, which has a convex limit $J_N^\infty(\cdot)$. Every convex function is locally Lipschitz continuous on open subsets (see e.g. [7]), hence $J_N^\infty(\cdot)$ is locally Lipschitz continuous on $\text{int conv } \mathcal{D}_x(k_0)$.

Step 2) $J_N^a(x, D(k))$ converges to $J_N^*(x)$: Let $x \in \Omega \cap \text{int conv } \mathcal{D}_x(k_0)$ and let $\mathcal{C} \subset \text{int conv } \mathcal{D}_x(k_0)$ with $\text{int } \mathcal{C} \ni x$ be compact, then there exists a sequence $k_i \rightarrow \infty$ such that $\xi_i = f(x(k_i), \Pi_0 U(k_i)) \rightarrow x$, $\xi_i \in \mathcal{C}$. Let L_k be the Lipschitz constant of $J_N^a(\cdot, D(k))$ on \mathcal{C} and L_∞ for $J_N^\infty(\cdot)$, respectively, which are finite since \mathcal{C} is compact and the functions are locally Lipschitz. Hence, $L_k \rightarrow L_\infty$ as $k \rightarrow \infty$ is a real valued converging sequence and is therefore upper bounded by some $M \in \mathbb{R}$. It follows

$$\begin{aligned} |J_N^a(\xi_i, D(k_i+1)) - J_N^a(\xi_i, D(k_i))| &\leq 2M \|\xi_i - x\| \\ &+ |J_N^a(x, D(k_i+1)) - J_N^a(x, D(k_i))| \rightarrow 0. \end{aligned}$$

Thus the following chain of inequalities

$$\begin{aligned} J_N^a(\xi_i, D(k_i+1)) &\leq J_N(U(k_i+1), \xi_i) \\ &\leq J_N(\Phi^1(U(k_i), x(k_i), D(k_i)), \xi_i) \\ &\leq J_N(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) \\ &\leq J_N(\Psi_{\text{sw}}(\xi_i, D(k_i)), \xi_i) \leq J_N^a(\xi_i, D(k_i)) \end{aligned} \quad (18)$$

is a chain of equalities in the limit, where the first inequality holds due to (8) and (15) and the other inequalities due to (7b), (7b), (11) and Lemma 4 (i) in this specific order. Therefore as $i \rightarrow \infty$ the decrease of the optimizer update operator (7b) $\gamma(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) \rightarrow 0$, which is due to $\gamma(U, x) = 0 \Leftrightarrow J_N(U, x) = J_N^*(x)$ only possible if

$$J_N(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) - J_N^*(\xi_i) \rightarrow 0.$$

This is due to (18) equivalent to $J_N^a(\xi_i, D(k_i)) - J_N^*(\xi_i) \rightarrow 0$. Since J_N^* is convex, it is also locally Lipschitz continuous on $\text{int conv } \mathcal{D}_x(k_0)$ and we assume that M is a Lipschitz constant of J_N^* on \mathcal{C} (if not choose M large enough). Hence, it follows

$$\begin{aligned} |J_N^a(x, D(k_i)) - J_N^*(x)| &\leq |J_N^a(\xi_i, D(k_i)) - J_N^*(\xi_i)| \\ &+ |J_N^a(x, D(k_i)) - J_N^a(\xi_i, D(k_i)) + J_N^*(\xi_i) - J_N^*(x)| \\ &\leq |J_N^a(\xi_i, D(k_i)) - J_N^*(\xi_i)| + 2M \|x - \xi_i\| \rightarrow 0. \end{aligned}$$

Thus, we showed that $J_N^a(x, D(k)) \rightarrow J_N^*(x)$ which implies due to (17) the desired result (16). \square

Remark 7. The assumption $x \in \text{int conv } \mathcal{D}_x(k_0)$ for some $k_0 \in \mathbb{N}$ is rather technical. Still, it is possible that a recurrent point never lies in the interior of $\text{conv } \mathcal{D}_x(k)$ as $k \rightarrow \infty$. A straightforward solution to this problem is to initialize the data set $\mathcal{D}(0)$ with some points $(x_i, U_i, J_N(U_i, x_i))$, $i = 1, \dots, K$. This solves the issue for all $x \in \text{int conv } \{x_1, \dots, x_K\}$ even for arbitrary U_i .

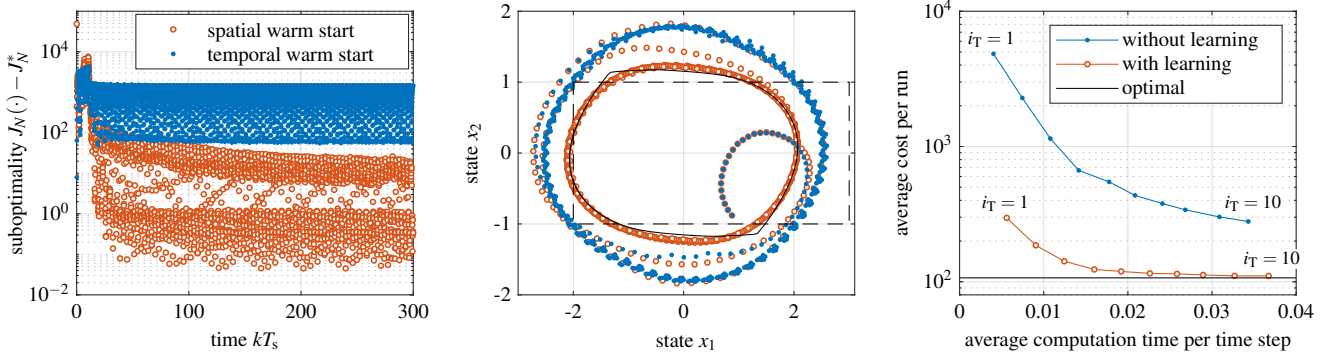


Figure 3. Simulation results of the double integrator example from Section 3.3. *Left:* The suboptimality of the costs for the spatial and temporal warm start $J_N(\Psi_{s/tw}(k), f(x(k), \Pi_0 U(k))) - J_N^*(f(x(k), \Pi_0 U(k)))$ over time when applying $i_T(k) = 2$ gradient descent steps per optimization. *Middle:* Closed-loop trajectory (○) compared to the original non-learning method (•) starting from the same initial condition. State constraints (dashed) and optimal limit cycle (solid) are also depicted. *Right:* Performance vs. computation time comparison for $i_T(k) = 1, \dots, 10$. Averaged over 10 runs ($N_{\text{sim}} = 3000$) with random initial conditions.

Hence, a good choice for the points x_i are the extreme points of the polytopic feasible set. By adding the point $(0, 0, 0)$ to $\mathcal{D}(0)$ the prior knowledge about the optimal input at the origin can be included.

3.2. Algorithm and implementation

For online learning, it is crucial to solve (15) in an efficient way, since a warm start solution must be provided within one sampling period. The convex hull computation gets increasingly burdensome with more data points and higher state space dimensions. Therefore, we rely on an incremental convex hull algorithm and split the computation in two parts: the convex hull update and the spatial warm start generation. Detailed implementations of both parts can be found in the Appendix B. The update can be performed on a different time scale and does not need to be executed in real time, as is demonstrated in Section 5. Notice that the number of data points that is needed to get a good approximation of the MPC control law scales with the dimension of the ω -limit set Ω , which can be significantly lower than the dimension of the state space. Moreover, the computational complexity of the learning algorithm is independent of the length of the prediction horizon. Finally, we emphasize that stability is always guaranteed. Hence, storage and processing restrictions only limit the performance improvement but do not jeopardize the stability of the controller.

3.3. Example: Double Integrator

We demonstrate the learning scheme presented in this section for a numerical example. We consider the discrete-time double integrator system from [19] of the form

$$x(k+1) = \begin{bmatrix} 1 & T_s \\ 0 & 1 \end{bmatrix} x(k) + \begin{bmatrix} T_s^2 \\ T_s \end{bmatrix} u(k) + w(k) \quad (19)$$

with sampling time $T_s = 0.1$ s. Input and state constraints are inherited from [19] $\mathcal{U} = \{u \in \mathbb{R} : |u| \leq 1\}$ and $\mathcal{X} = \{x \in \mathbb{R}^2 :$

$-2 \leq x_1 \leq 3, |x_2| \leq 1\}$, and likewise the parameters of the cost function and of the backtracking line search optimization with gradient descent search direction. The external disturbance is quasiperiodic $w(k) = 0.09 [\sin(kT_s) \quad \cos(kT_s)]^\top$.

For two optimization iterations $i_T(k) \equiv i_T = 2$ and a randomly chosen initial condition at $x(0) \approx [0.9 \quad -0.9]^\top$, we have simulated the closed-loop behavior of the presented MPC scheme for $N_{\text{sim}} = 3000$ time steps. In the left subplot of Fig. 3, the suboptimality of both warm starts, i.e. the difference of their costs to the value function, is depicted over time. As was to be expected, the spatial warm start performs first poorly since too few data is available, but then improves over time while more and more data is collected until it significantly outperforms the temporal warm start. Further, we can see that the learning rate is limited by the optimization update operator since two gradient descent iterations per time step result in a slow rate of convergence.

In order to highlight the improvements offered by the proposed learning scheme, we compare it to the original anytime MPC [19], i.e. to Algorithm 1 without lines 4 and 10. To this end, the state trajectories of both methods are depicted in the middle subplot of Fig. 3 in the phase portrait until time $k = 700$ together with the limit cycle of the optimally controlled system, where the optimization problem (3) is solved with MATLAB's `fminunc`. As we can see, learning significantly improves the performance and leads to much better constraint satisfaction. First, both trajectories are identical since the spatial warm start is never used, but as more data is collected the trajectories deviate and the one from the proposed scheme approaches the optimal one. In particular, the trajectories deviate as soon as they enter the interior of $\text{conv } \mathcal{D}_x(k)$, where for the proposed scheme the convergence result of Theorem 6 holds.

Thus far, we have not considered the fact that generating the spatial warm start consumes computation time that might be better invested in doing more optimization iterations with the temporal warm start. To investigate this, we run the simulation for different numbers of optimization iterations $i_T(k) \equiv i_T =$

1, ..., 10 and for ten different initial conditions. For each i_T , we average the computation times and the costs to obtain a point in the right subplot of Fig. 3. We can see that the proposed learning scheme is way closer to the optimal performance than the original anytime MPC without learning.

4. Leveraging data in real-time nonlinear MPC

In the general case of the nonlinear MPC scheme (2)–(8), (11), we cannot expect that the cost function J_N is convex. Therefore, we propose in this section a different learning method that does not depend on convexity, but instead exploits Lipschitz continuity.

4.1. A spatial warm start based on Lipschitz continuity

The idea for the spatial warm start generation for $\bar{x} \in \mathbb{R}^n$ is to find an input sequence from the data $(U, x, J) \in \mathcal{D}$ such that x is close to \bar{x} and J is small. This directly leads to the following spatial warm start rule

$$\begin{aligned} (\Psi_{\text{sw}}(\bar{x}, \mathcal{D}), x^*(\bar{x}, \mathcal{D}), J^*(\bar{x}, \mathcal{D})) \\ = \arg \min_{(U, x, J) \in \mathcal{D}} J + L(U) \|x - \bar{x}\| \end{aligned} \quad (20a)$$

$$J_N^a(\bar{x}, \mathcal{D}) = \min_{(U, x, J) \in \mathcal{D}} J + L(U) \|x - \bar{x}\| \quad (20b)$$

where $L(U) \in \mathbb{R}_{++}$ is a Lipschitz parameter that might depend on U , and $J_N^a(\cdot, \mathcal{D})$ is an approximation of the value function J_N^* based on the data \mathcal{D} . An illustration of the spatial warm start (20) is given in Fig. 4. If we assume $J_N(U, \cdot)$ to be Lipschitz continuous with constant $L(U)$, then $J_N^a(\cdot, \mathcal{D})$ upper bounds the value function and the spatial warm start as shown in the following Lemma. Similar ideas to approximate an unknown function based on Lipschitz continuity from sampling data have been proposed in [4], [9], and [40].

Assumption 8. Assume that $J_N(U, \cdot)$ is Lipschitz continuous with constant $L(U)$, i.e. for all $x, y \in \mathbb{R}^n$,

$$|J_N(U, x) - J_N(U, y)| \leq L(U) \|x - y\|. \quad (21)$$

Lemma 9. Let Assumption 8 hold, then

$$J_N(\Psi_{\text{sw}}(x, \mathcal{D}), x) \leq J_N^a(x, \mathcal{D}) \quad (22)$$

and $J_N^a(\cdot, \mathcal{D})$ is also Lipschitz continuous with Lipschitz constant $L_{\mathcal{D}} = \max_{U \in \mathcal{D}_U} L(U)$.

For a proof, see Appendix A. With Lemma 9, we obtain a convergence result similar to Theorem 6.

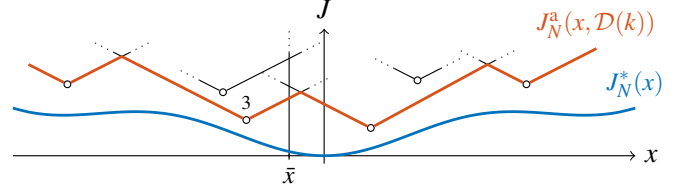


Figure 4. Illustration of the spatial warm start (20) in the (x, J) -domain with data points $(x_i, J_i) \in \mathcal{D}_{xJ}(k)$ (\circ). For each point, the cone $J_i + L(U_i) \|x - x_i\|$ is indicated as well as the minimum over all i for each x given by $J_N^a(x, \mathcal{D}(k))$, which is an upper bound for the value function. The spatial warm start solution at \bar{x} is the input from point 3.

Theorem 10. Consider the nonlinear MPC scheme presented in (2)–(9), (11) with the spatial warm start operator (20), further let Assumption 8 hold, and let $\forall k \geq 1 : L(U(k)) \leq M \in \mathbb{R}$ be upper bounded. Then for all $x \in \Omega$

$$\lim_{k \rightarrow \infty} J_N(\Psi_{\text{sw}}(x, \mathcal{D}(k)), x) = J_N^*(x). \quad (23)$$

Proof. Step 1) $J_N^a(x, \mathcal{D}(k))$ converges: In view of Lemma 9, we have $J_N^*(x) \leq J_N(\Psi_{\text{sw}}(x, \mathcal{D}(k)), x) \leq J_N^a(x, \mathcal{D}(k))$. Thus by showing $J_N^a(x, \mathcal{D}(k)) \rightarrow J_N^*(x)$ for $x \in \Omega$ we will prove the theorem. We also see from this inequality that $J_N^a(x, \mathcal{D}(k))$ is bounded from below by $J_N^*(x)$. Further $J_N^a(x, \mathcal{D}(k))$ is decreasing since the minimum over a larger set $\mathcal{D}(k+1) \supseteq \mathcal{D}(k)$ is smaller or equal and thus $J_N^a(x, \mathcal{D}(k))$ must converge to some value.

Step 2) $J_N^a(x, \mathcal{D}(k))$ converges to $J_N^*(x)$: This step is analogous to step 2) in the proof of Theorem 6 and thus moved to the Appendix A. \square

Remark 11. Even though Assumption 8 might be restrictive, if we assume that $x(k)$ does not grow unbounded, but stays in some compact region $\mathcal{C}_x \subset \mathbb{R}^n$, then the Lipschitz constants $L(U(k))$ do not need to apply globally, but only on \mathcal{C}_x in order to obtain the same result. If further $U(k)$ stays in some compact set $\mathcal{C}_U \subseteq \mathbb{R}^{N_m}$, then the assumption $L(U(k)) \leq M$ is also satisfied with $M = \max_{U \in \mathcal{C}_U} L(U) < \infty$ if J_N is continuous in U .

Remark 12. It can be quite challenging to satisfy (6) and (7) for general nonlinear MPC algorithms. As often done in practice, one can still implement the learning scheme with spatial warm start (20). The controller performance will improve as long as the optimization iteration does reduce the costs, even if (6) and (7) do not hold. However, a prior stability guarantee depends on the nonlinear iteration scheme.

Remark 13. If in a specific setup the cost function J_N is nevertheless known to be convex, then the learning scheme of Section 3 can be used and probably leads to a better upper bound J_N^a ; for example, see Fig. 4 where the convex hull over the data points would result in a lower upper bound $J_N^a(\cdot, \mathcal{D})$ inside $\text{conv } \mathcal{D}_x$. On the other hand, if one faces a linear MPC problem with a nonconvex cost function J_N , due to nonconvex stage or

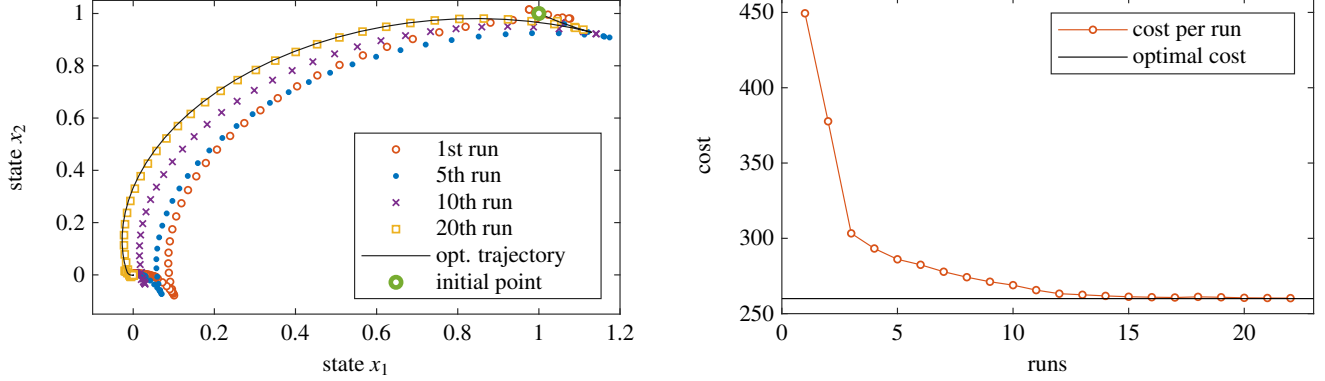


Figure 5. Simulation results of the unicycle example from Section 4.2 controlled by the proposed MPC scheme for nonlinear systems. *Left:* State trajectory for several runs and the optimal trajectory (computed with MATLAB’s `patternsearch`). *Right:* Accumulated cost per run over the number of the runs. Additionally, the optimal cost is depicted.

terminal costs, then the learning scheme from this section can be applied.

4.2. Example: Unicycle

In this section, we implement the nonlinear real-time MPC learning scheme for the unicycle model

$$\dot{x} = f_c(x, u) = [u_1 \cos(x_3) \quad u_1 \sin(x_3) \quad u_2]^\top \quad (24)$$

with state and input vectors $x = [x_1 \ x_2 \ x_3]$, $u = [u_1 \ u_2]$ respectively, as well as the sampling time $T_s = 0.1$ s. The control task is to drive the unicycle to $(x_1, x_2) = (0, 0)$ facing into positive x_1 direction, i.e. $x_3 = 2z\pi$ for some $z \in \mathbb{Z}$ while keeping $\|u\|_\infty < 1$. Therefore, the stage cost in (3) is chosen as $l(x, u) = 0.1 \sin(\frac{x_3}{2})^2 + \sqrt[4]{1 + x_1^2 + x_2^2} - 1 + u_1^8 + u_2^8$ to ensure positive definiteness with respect to $u = 0$, $(x_1, x_2, x_3) = (0, 0, 2z\pi)$, prevent $x_1^2 + x_2^2$ from getting too steep, while ensuring differentiability at the origin, and let u shoot up as $\|u\|_\infty > 1$. The terminal cost is set to $F(x) = 100l(x, 0)$. For this example, the learning scheme from Section 3 cannot be applied since the resulting cost function J_N is not convex. However, the costs turn out to be Lipschitz continuous in x with Lipschitz constant $L(U)$ computed in Appendix C.

We choose $\Psi_{\text{tw}}(U, x) = [U_1^\top, \dots, U_{N-1}^\top, 0^\top]^\top$ as the temporal warm start and the same optimization operator (5b) as for the linear example in Section 3.3, which consists of the gradient descent search direction with backtracking line search. Further, we use a constant number of optimization iterations $i_T(k) = 2$. We consider iterative learning in a repetitive and fast process, which is a common task in iterative learning control (see [38]). The system is repeatedly started from the same initial condition $x_0 = [1, 1, 1 + \pi/2]^\top$, where each run takes 120 time steps. Hence, throughout a run, $w(k)$ is zero and after every 120 time steps it sets $x(k)$ back to x_0 .

Although Ψ_{tw} and Ψ_o might not satisfy (7), the closed loop performance is still satisfying as we can see in the simulation

results depicted in Fig. 5. In the plot, we can see that the optimal behavior is learned within a few runs. After five runs the unicycle has learned to start driving backwards, and after 20 runs it has learned to approach the destination driving forwards and is almost indistinguishable from the optimal trajectory.

5. Application: Servomechanism

In the two previous examples, we have not considered the actual time available for the computation. In fact, the learning update may not always be computable within one sampling period T_s , especially when considering systems with higher dimensions and fast dynamics. To circumvent this issue, we will outsource learning onto a server that communicates with the controller. In particular we assume, that the convex hull update is executed in parallel to the controller and whenever it is completed, one of the latest data points since the last one added is added next. Further, $i_T(k)$ is not predefined but the optimization iteration is stopped as soon as the computation time is exhausted. This does not affect our stability and convergence results as long as at least one optimization iteration is performed since Theorem 1 and Theorem 6 apply for any sequence $i_T(k) \in \mathbb{N}_+$.

In this section, we will implement the learning scheme for a reference tracking task on a simulation model of a servomechanism consisting of a DC-motor, a gear-box, an elastic shaft and a load. This system has already served as an example in [6] from where we have inherited the linear model

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_\theta}{J_L} & -\frac{\beta_L}{J_L} & \frac{k_\theta}{\rho J_L} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_\theta}{\rho J_M} & 0 & -\frac{k_\theta}{\rho^2 J_M} & \frac{-\beta_M R - K_T^2}{J_M R} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{K_T}{R J_M} \end{bmatrix} u \quad (25)$$

and the parameters, which can be found in [6]. The state vector $x = [\theta_L \ \dot{\theta}_L \ \theta_M \ \dot{\theta}_M]^\top$ consists of the load angle θ_L and the motor angle θ_M as well as their time derivatives and the in-

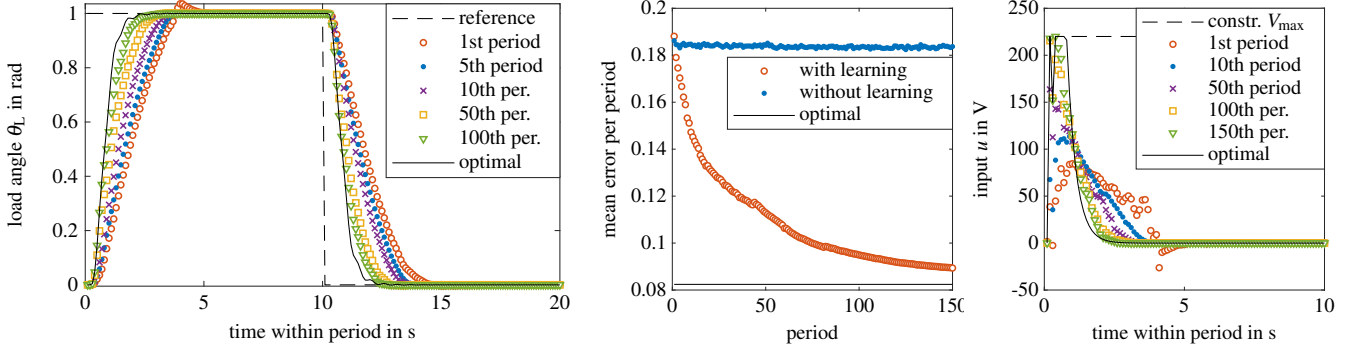


Figure 6. Simulation results of the servomechanism in Section 5. *Left:* The effect of learning is clearly visible in the improved reference tracking behavior over increasing number of periods. *Middle:* The mean tracking error per period of the linear anytime MPC with and without the proposed learning scheme as well as the optimal mean error. *Right:* First ten seconds of the input signal for different periods, as well as the optimal input and the input constraint.

put u corresponds to the DC voltage. The reference tracking task of this system is also included in the collection of benchmark MPC problems given in [25]. The model is discretized using zero-order hold on the input and the sampling time $T_s = 0.1$ s. Further the system has to satisfy the state and input constraints $\| [k_\theta \ 0 \ -k_\theta/\rho \ 0] x \| \leq T_{\max}$, $|u| \leq V_{\max}$, where T_{\max}, V_{\max} can be found in [6]. To make the problem more difficult and to obtain a polytopic feasible set, we added the constraints $|x_1| \leq 2$, $|x_2| \leq 2$, and $|x_2 + x_4| \leq 40$ to the original problem.

The goal is that the angle of the load θ_L follows the periodic step reference signal $r(k)$ with period length 200. To achieve this we see that $x_s(r_s) = [1 \ 0 \ \rho \ 0]^\top r_s$, $u_s = 0$ is a steady state of (25) that produces exactly the constant reference r_s for arbitrary $r_s \in \mathbb{R}$. Therefore we will define the cost function with respect to $x - x_s(r(k))$ and $u - u_s = u$ except for the part that incorporates the constraints. That is, (13) and (14) become

$$l(x, r, u) = \|x - x_s(r)\|_Q^2 + \|u\|_R^2 + \varepsilon \hat{B}(x, u), \quad (26a)$$

$$F(x, r) = \|x - x_s(r)\|_P^2, \quad (26b)$$

with $Q = \text{diag}([10 \ 0.1 \ 10 \ 0.1])$, $R = 0.01$, and $\varepsilon = 10^{-3}$. Further, the relaxed logarithmic barrier function \hat{B} and P are chosen according to [18]. For the optimization update operator, we use the gradient descent backtracking line search as described in [19] with parameters $\rho = 0.5$, $c_1 = 10^{-3}$ and $c_2 = 0.999$. The data set $\mathcal{D}(0)$ will be initialized with the extreme points of the feasible set as described in Remark 7.

The simulation results of the closed loop over $N_{\text{sim}} = 30000$ time steps (150 periods of the reference signal) are shown in Fig. 6. In the left subplot, we can see that the reference tracking performance improves from period to period. This is also visible in the middle subplot, where the mean reference tracking error per period of the learning scheme decreases and converges to the one of the optimally controlled system. The original anytime MPC scheme without a spatial warm start and thus without

learning does not improve over time and the mean reference tracking error stays constant. The scheme with learning is in the first period worse than without learning, because the time to compute the spatial warm start cannot be used for further optimization iterations. In average there is a difference of 1.8 iterations in between the original anytime MPC (≈ 39.4 iterations) and the scheme with learning (≈ 37.6). However, already in the second period the scheme with learning achieves better tracking performance and improves much more over time. After the 120th period the controller has learned to exploit the full range of feasible control inputs as can be seen in the right subplot of Fig. 6.

There is a computational aspect that we have not discussed yet. The required memory capacity and the computation time for updating the convex hull and for the spatial warm start generation increase with the size of $\mathcal{D}(k)$ such that the number of points in the convex hull that can be handled is limited. Hence, to exploit the available capacities we add only data points (U, x, J) to $\mathcal{D}(k)$ that lead to a significant improvement from $J_N^a(x, \mathcal{D}(k))$ to $J = J_N^a(x, \mathcal{D}(k+1))$, here $> 10^{-2}$. In this example, out of 30000 data points 21164 were skipped because the improvement was too small and 4630 because it was not possible to update the convex hull within one sampling time, leading to a total of 4206 data points added to the convex hull.

In summary, the example clearly shows the benefit of the proposed memory-based MPC scheme in improving performance, while maintaining guarantees on stability at all times and being suitable for online implementation.

6. Conclusion and outlook

We presented and analyzed an online learning scheme for the value function and optimal policy in a real-time MPC framework for both linear and nonlinear systems. Even if only one optimization iteration is performed between two consecutive sampling instants, the MPC still approaches optimality in the

long run through the learning scheme. Furthermore, stability of the real-time MPC scheme is retained independent of the type of learning approach. These findings were illustrated in different linear and nonlinear numerical examples. Moreover, we discussed how learning and applying the learned input can be decoupled and parallelized.

An interesting topic for future study is to further improve the presented learning methods by more efficient implementations of the data structure and the algorithms to generate the spatial warm start. In particular, while we have not provided a concrete algorithm for the nonlinear Lipschitz based learning scheme, the ideas of [4] might be helpful for arriving at an efficient implementation. The optimization iteration also leaves room for improvement by exploiting underlying properties or structure. The proposed method can be extended in several ways. For example the online learning scheme could probably be modified to handle time-varying systems or costs by including some mechanisms to forget expired data points. Since our stability result does not depend on the learning method, this opens the possibility to apply classical machine learning function approximators for online learning. A comparison to a function approximation based on offline data to generate a static spatial warm start would also be interesting to investigate.

References

- [1] B. M. Åkesson and H. T. Toivonen. A neural network model predictive controller. *Journal of Process Control*, 16(9):937–946, 2006.
- [2] M. Alami. From certification of algorithms to certified MPC: The missing links. In *5th IFAC Conference on Nonlinear Model Predictive Control*, volume 48, pages 65–72, 2015.
- [3] A. Aswani, H. Gonzalez, S. S. Sastry, and C. Tomlin. Provably safe and robust learning-based model predictive control. *Automatica*, 49(5):1216–1226, 2013.
- [4] G. Beliakov. Interpolation of Lipschitz functions. *Journal of Computational and Applied Mathematics*, 196(1):20 – 44, 2006.
- [5] A. Bemporad, D. Bernardini, and P. Patrinos. A convex feasibility approach to anytime model predictive control. *arXiv*, 1502.07974, 2015.
- [6] A. Bemporad and E. Mosca. Fulfilling hard constraints in uncertain linear systems by reference managing. *Automatica*, 34(4):451–461, 1998.
- [7] J. Borwein and A. Lewis. *Convex analysis and nonlinear optimization: Theory and examples*. Springer, New York, 2006.
- [8] P. Bouffard, A. Aswani, and C. Tomlin. Learning-based model predictive control on a quadrotor: Onboard implementation and experimental results. *IEEE Int. Conf. on Robotics and Automation*, pages 279–284, 2012.
- [9] J. Calliess. *Conservative decision-making and inference in uncertain dynamical systems*. PhD Thesis, University of Oxford, 2014.
- [10] M. Canale, L. Fagiano, and M. Signorile. Nonlinear model predictive control from data: a set membership approach. *International Journal of Robust and Nonlinear Control*, 24(1):123–139, 2014.
- [11] A. Chakrabarty, V. Dinh, M. J. Corless, A. E. Rundell, S. H. ak, and G. T. Buzzard. Support vector machine informed explicit nonlinear model predictive control using low-discrepancy sequences. *IEEE Trans. on Automatic Control*, 62(1):135–148, 2017.
- [12] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari. Approximating explicit model predictive control using constrained neural networks. In *2018 American Control Conf. (ACC)*, pages 1520–1527, 2018.
- [13] G. Chowdhary, M. Mühlegg, J. P. How, and F. Holzapfel. Concurrent learning adaptive model predictive control. In *Advances in Aerospace Guidance, Navigation and Control*, pages 29–47. Springer Berlin Heidelberg, 2013.
- [14] M. Diehl, R. Findeisen, F. Allgöwer, H. G. Bock, and J. P. Schlöder. Nominal stability of real-time iteration scheme for nonlinear model predictive control. *IEE Proceedings - Control Theory and Applications*, 152(3):296–308, 2005.
- [15] A. Domahidi, M. N. Zeilinger, M. Morari, and C. N. Jones. Learning a feasible and stabilizing explicit model predictive control law by robust optimization. In *50th IEEE Conf. on Decision and Control*, pages 513–519, 2011.
- [16] J. Drgoňa, D. Picard, M. Kvasnica, and L. Helsen. Approximate model predictive building control via machine learning. *Applied Energy*, 218:199–216, 2018.
- [17] C. Feller and C. Ebenbauer. Weight recentered barrier functions and smooth polytopic terminal set formulations for linear model predictive control. In *2015 American Control Conference (ACC)*, pages 1647–1652, 2015.
- [18] C. Feller and C. Ebenbauer. Relaxed logarithmic barrier function based model predictive control of linear systems. *IEEE Trans. on Automatic Control*, 62(3):1223–1238, 2017.
- [19] C. Feller and C. Ebenbauer. A stabilizing iteration scheme for model predictive control based on relaxed barrier functions. *Automatica*, 80:328–339, 2017.

- [20] D. Gu and H. Hu. Neural predictive control for a car-like mobile robot. *Robotics and Autonomous Systems*, 39(2):73–86, 2002.
- [21] M. Hertneck, J. Köhler, S. Trimpe, and F. Allgöwer. Learning an approximate model predictive controller with guarantees. *IEEE Control Systems Letters*, 2(3):543–548, 2018.
- [22] N. Hirose, R. Tajima, and K. Sukigara. MPC policy learning using DNN for human following control without collision. *Advanced Robotics*, 32(3):148–159, 2018.
- [23] M. Klaučo, M. Kalúz, and M. Kvasnica. Machine learning-based warm starting of active set methods in embedded model predictive control. *Engineering Applications of Artificial Intelligence*, 77:1 – 8, 2019.
- [24] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard. Gaussian process model based predictive control. In *American Control Conf. (ACC)*, pages 2214–2219, 2004.
- [25] D. Kouzoupis, A. Zanelli, H. Peyrl, and H. J. Ferreau. Towards proper assessment of QP algorithms for embedded model predictive control. In *European Control Conference (ECC)*, pages 2609–2616, 2015.
- [26] I. Lenz, R. Knepper, and A. Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Proceedings of Robotics: Science and Systems*, 2015.
- [27] D. Limon, J. Calliess, and J. Maciejowski. Learning-based nonlinear model predictive control. *IFAC-PapersOnLine*, 50(1):7769–7776, 2017. 20th IFAC World Congress.
- [28] L. K. McGovern and E. Feron. Closed-loop stability of systems driven by real-time, dynamic optimization algorithms. In *38th IEEE Conference on Decision and Control*, volume 4, pages 3690–3696, 1999.
- [29] C. J. Ostafew, A. P. Schoellig, and T. D. Barfoot. Learning-based nonlinear model predictive control to improve vision-based mobile robot path-tracking in challenging outdoor environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4029–4036, 2014.
- [30] T. Parisini and R. Zoppoli. A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10):1443–1451, 1995.
- [31] S. Piche, B. Sayyar-Rodsari, D. Johnson, and M. Gerules. Nonlinear model predictive control using neural networks. *IEEE Control Systems Magazine*, 20(3):53–62, June 2000.
- [32] S. S. Pon Kumar, A. Tulsyan, B. Gopaluni, and P. Loewen. A deep learning architecture for predictive control. *10th IFAC Symposium on Advanced Control of Chemical Processes*, 51(18):512 – 517, 2018.
- [33] J. B. Rawlings and D. Q. Mayne. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2009.
- [34] U. Rosolia and F. Borrelli. Learning model predictive control for iterative tasks. A data-driven control framework. *IEEE Trans. on Automatic Control*, 63(7):1883–1896, 2018.
- [35] L. Schwenkel, M. Gharbi, S. Trimpe, and C. Ebenbauer. Online learning with stability guarantees: A memory-based warm-starting for real-time MPC. *Automatica*, 122:109247, 2020.
- [36] P. O. M. Scokaert, D. Q. Mayne, and J. B. Rawlings. Sub-optimal model predictive control (feasibility implies stability). *IEEE Trans. Automatic Control*, pages 648–654, 1999.
- [37] P. Tøndel, T. A. Johansen, and A. Bemporad. An algorithm for multi-parametric quadratic programming and explicit MPC solutions. *Automatica*, 39(3):489 – 497, 2003.
- [38] Y. Wang, F. Gao, and F. J. Doyle. Survey on iterative learning control, repetitive control, and run-to-run control. *Journal of Process Control*, 19(10):1589 – 1600, 2009.
- [39] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou. Information theoretic MPC for model-based reinforcement learning. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 1714–1721, 2017.
- [40] Z. B. Zabinsky, R. L. Smith, and B. P. Kristinsdottir. Optimal estimation of univariate black-box Lipschitz functions with upper and lower error bounds. *Computers & Operations Research*, 30(10):1539 – 1553, 2003.
- [41] M. N. Zeilinger, D. M. Raimondo, A. Domahidi, M. Morari, and C. N. Jones. On real-time robust model predictive control. *Automatica*, 50(3):683–694, 2014.
- [42] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with MPC-guided policy search. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 528–535, 2016.

A. Proofs

Proof of Lemma 4. (i) Since J_N is convex and the spatial warm start defined in (15) is a convex combination of data points

$(U_i, x_i, J_i) \in \mathcal{D}(k)$, we have

$$(\Psi_{\text{sw}}(x, \mathcal{D}(k)), x, J_N^a(x, \mathcal{D}(k))) = \sum_{i=1}^K \lambda_i (U_i, x_i, J_i)$$

and further

$$\begin{aligned} J_N(\Psi_{\text{sw}}(x, \mathcal{D}(k)), x) &= J_N \left(\sum_{i=1}^K \lambda_i (U_i, x_i) \right) \\ &\leq \sum_{i=1}^K \lambda_i J_N(U_i, x_i) \\ &= \sum_{i=1}^K \lambda_i J_i = J_N^a(x, \mathcal{D}(k)). \end{aligned}$$

(ii) We slightly reformulate (15)

$$J_N^a(x, \mathcal{D}(k)) = \min_{\substack{(z, J) \in \text{conv } \mathcal{D}_{\text{XJ}}(k) \\ z=x}} J \quad (27)$$

and show that the epigraph of $J_N^a(\cdot, \mathcal{D}(k)) : \text{conv } \mathcal{D}_{\text{XJ}}(k) \rightarrow \mathbb{R}_+$ is convex:

$$\begin{aligned} \text{epi } J_N^a(\cdot, \mathcal{D}(k)) &= \left\{ (x, \bar{J}) \mid \bar{J} \geq \min_{\substack{(z, J) \in \text{conv } \mathcal{D}_{\text{XJ}}(k) \\ z=x}} J \right\} \\ &= \{(x, \bar{J}) \mid \exists J \leq \bar{J} : (x, J) \in \text{conv } \mathcal{D}_{\text{XJ}}(k)\} \\ &= \{(x, J + \alpha) \mid \alpha \geq 0 \wedge (x, J) \in \text{conv } \mathcal{D}_{\text{XJ}}(k)\} \\ &= \text{conv } \mathcal{D}_{\text{XJ}}(k) \oplus (\{0\} \times [0, \infty)) \end{aligned}$$

where \oplus denotes the Minkowski sum. Since both $\text{conv } \mathcal{D}_{\text{XJ}}(k)$ and $\{0\} \times [0, \infty)$ are convex, the epigraph $\text{epi } J_N^a(\cdot, \mathcal{D}(k))$ is also. \square

Proof of Lemma 5. It is shown in [19] that the cost function J_N can be written under these assumptions as

$$J_N(U, x) = \frac{1}{2} U^\top H U + x^\top F U + \frac{1}{2} x^\top Y x + \varepsilon \hat{B}_{\text{xu}}(U, x)$$

where H, F and Y can be computed from Q, R, P, A and B and account for the terms $\|x\|_Q^2$, $\|u\|_R^2$ and $\|x\|_P^2$ in l and F . Furthermore, the relaxed logarithmic barrier function $\hat{B}_{\text{xu}}(U, x)$ is shown in [19] to be a weighted sum over convex functions whose argument depends affinely on (U, x) . Therefore $\hat{B}_{\text{xu}}(U, x)$ is convex in (U, x) . The quadratic part of J_N is also convex in (U, x) since it is quadratic and positive definite as proven in [19]. \square

Proof of Lemma 9. In view of (21) we have

$$\begin{aligned} |J_N(\Psi_{\text{sw}}(x, \mathcal{D}), x) - J_N(\Psi_{\text{sw}}(x, \mathcal{D}), x^*(x, \mathcal{D}))| \\ \leq L(\Psi_{\text{sw}}(x, \mathcal{D})) \|x^*(x, \mathcal{D}) - x\| \end{aligned}$$

and further

$$\begin{aligned} J_N(\Psi_{\text{sw}}(x, \mathcal{D}), x) &\leq J_N(\Psi_{\text{sw}}(x, \mathcal{D}), x^*(x, \mathcal{D})) \\ &\quad + L(\Psi_{\text{sw}}(x, \mathcal{D})) \|x^*(x, \mathcal{D}) - x\| \\ &= J_N^a(x, \mathcal{D}). \end{aligned}$$

To prove the second statement, we use (20a) with $\bar{x} = y \in \mathbb{R}^n$ and (20b) with $\bar{x} = x \in \mathbb{R}^n$ to see

$$J_N^a(x, \mathcal{D}) \leq J^*(y, \mathcal{D}) + L(\Psi_{\text{sw}}(y, \mathcal{D})) \|x - x^*(y, \mathcal{D})\|$$

which implies

$$\begin{aligned} J_N^a(x, \mathcal{D}) - J_N^a(y, \mathcal{D}) &\leq L(\Psi_{\text{sw}}(y, \mathcal{D})) (\|x - x^*(y, \mathcal{D})\| - \|y - x^*(y, \mathcal{D})\|) \\ &\leq L_{\mathcal{D}} \|x - y\|. \end{aligned}$$

Since x and y are interchangeable, it follows

$$|J_N^a(x, \mathcal{D}) - J_N^a(y, \mathcal{D})| \leq L_{\mathcal{D}} \|x - y\|. \quad \square$$

Step 2) of the proof of Theorem 10. Step 2) $J_N^a(x, \mathcal{D}(k))$ converges to $J_N^*(x)$: For $x \in \Omega$, there exists a sequence $k_i \rightarrow \infty$ such that $\xi_i = f(x(k_i), \Pi_0 U(k_i)) \rightarrow x$. It follows

$$\begin{aligned} |J_N^a(\xi_i, \mathcal{D}(k_i + 1)) - J_N^a(\xi_i, \mathcal{D}(k_i))| &\leq 2M \|\xi_i - x\| \\ &\quad + |J_N^a(x, \mathcal{D}(k_i + 1)) - J_N^a(x, \mathcal{D}(k_i))| \rightarrow 0. \end{aligned}$$

Thus the following chain of inequalities

$$\begin{aligned} J_N^a(\xi_i, \mathcal{D}(k_i + 1)) &\leq J_N(U(k_i + 1), \xi_i) \\ &\leq J_N(\Phi^1(U(k_i), x(k_i), \mathcal{D}(k_i)), \xi_i) \\ &\leq J_N(\Phi^0(U(k_i), x(k_i), \mathcal{D}(k_i)), \xi_i) \\ &\leq J_N(\Psi_{\text{sw}}(\xi_i, \mathcal{D}(k_i)), \xi_i) \\ &\leq J_N^a(\xi_i, \mathcal{D}(k_i)), \end{aligned} \quad (28)$$

is a chain of equalities in the limit, where the first inequality holds due to (8) and (20) and the other inequalities due to (7b), (7b), (11) and Lemma 9 in this specific order. Therefore as $i \rightarrow \infty$ the decrease of the optimizer update operator (7b)

$$\gamma(\Phi^0(U(k_i), x(k_i), \mathcal{D}(k_i)), \xi_i) \rightarrow 0,$$

which is only possible if

$$J_N(\Phi^0(U(k_i), x(k_i), \mathcal{D}(k_i)), \xi_i) - J_N^*(\xi_i) \rightarrow 0.$$

This is due to (28) equivalent to

$$J_N^a(\xi_i, \mathcal{D}(k_i)) - J_N^*(\xi_i) \rightarrow 0.$$

J_N^* is Lipschitz continuous and without loss of generality we can assume that M is the Lipschitz constant of J_N^* (if not choose M large enough). Hence it follows

$$\begin{aligned} |J_N^a(x, \mathcal{D}(k_i)) - J_N^*(x)| &\leq |J_N^a(\xi_i, \mathcal{D}(k_i)) - J_N^*(\xi_i)| \\ &\quad + |J_N^a(x, \mathcal{D}(k_i)) - J_N^a(\xi_i, \mathcal{D}(k_i)) + J_N^*(\xi_i) - J_N^*(x)| \\ &\leq |J_N^a(\xi_i, \mathcal{D}(k_i)) - J_N^*(\xi_i)| + 2M \|x - \xi_i\| \rightarrow 0 \end{aligned}$$

which is with (17) what we wanted. \square

Table 1. Properties of the convex hull object \mathcal{CH}

name	description
\mathcal{D}_x	list of previous data points in \mathbb{R}^n
\mathcal{D}_U	list of previous input sequences in \mathbb{R}^{Nm}
\mathcal{D}_J	list of previous cost function values in \mathbb{R}_+
\mathcal{CH}_{xJ}	list that contains all facets of the lower half of $\text{conv}\mathcal{D}_{xJ}$ as lists of the indices of their extreme points in \mathcal{D}_{xJ}
\mathcal{CH}_x	list that contains all facets of $\text{conv}\mathcal{D}_x$ as lists of the indices of their extreme points in \mathcal{D}_x
c_x	center point of $\text{conv}\mathcal{D}_x$
c_{xJ}	center point of $\text{conv}\mathcal{D}_{xJ}$
o_x	list of deleted facets in \mathcal{CH}_x that can be overwritten
o_{xJ}	list of deleted facets in \mathcal{CH}_{xJ} that can be overwritten
\mathcal{G}_{xJ}	list that contains for each point in \mathcal{D}_{xJ} a list of facets in \mathcal{CH}_{xJ} that are attached to this point
\mathcal{G}_x	list that contains for each point in \mathcal{D}_x a list of facets in \mathcal{CH}_x that are attached to this point

B. Convex hull algorithm

The purpose of the convex hull algorithm presented in this section is to solve (15). As discussed in Section 3, $J_N^a(\cdot, \mathcal{D}(k))$ gives rise to a triangulation of $\text{conv}\mathcal{D}_x(k)$ and the spatial warm start at x can be computed by seeking for the n -simplex in this triangulation that contains x . Therefore in a first step, this triangulation of $\text{conv}\mathcal{D}_x(k)$ must be computed by a convex hull algorithm that determines the facets of the lower boundary of the convex hull $\text{conv}\mathcal{D}_{xJ}(k)$. The lower boundary of $\text{conv}\mathcal{D}_{xJ}(k)$ is the graph of $J_N^a(\cdot, \mathcal{D}(k))$, see Fig. 2. The extreme points of each of these facets are, after projection onto \mathbb{R}^n , the extreme points of an n -simplex of the triangulation of $\text{conv}\mathcal{D}_x(k)$. As second step, the spatial warm start can be computed from this convex hull by searching the n -simplex that contains x and combine the inputs corresponding to its extreme points to obtain a warm start solution. Hence, the procedure naturally decomposes into two steps:

- i) *Generate spatial warm start* from the convex hull.
- ii) *Update convex hull* with a new data point.

While i) corresponds to line 4 of Algorithm 1, ii) corresponds to line 14 where collecting data is meant as fitting the new point into the data structure. Notice that this decomposition divides the algorithm into learning ii) and applying the learned spatial warm start i) – a property that will be exploited in Section 5 for parallelization.

It is efficient to use an incremental convex hull algorithm that updates the existing convex hull object when a new data point arrives instead of starting the calculation from scratch. Further, the fact that for a point that is added to the convex hull, a spatial warm start was already generated and it has therefore been

located in the triangulation can be leveraged for efficiency. The problem of searching for the n -simplex in the triangulation that contains x is similar to the location problem in explicit MPC and hence these algorithms (see e.g. [37]) can be used. The implementation we used, however, is based on a directed search by tracking neighboring n -simplices starting from an initial guess down to x .

We present a way to implement the two main routines spatial warm start generation `generateSW` and convex hull update `updateCH`. First, we need to define an object that represents the collected data and all we need to know about the convex hull, we call this the convex hull object and denote it with \mathcal{CH} . The properties of the convex hull object are listed in Table 1. References to a property of the convex hull object are denoted with a dot, e.g. $\mathcal{CH}.\mathcal{D}_x$. In addition to these listed properties we will use $\mathcal{CH}.\mathcal{D}$ and $\mathcal{CH}.\mathcal{D}_{xJ}$ to denote the combined lists of the data points, so the k th element of $\mathcal{CH}.\mathcal{D}$ is for example a triple containing the k th elements of $\mathcal{CH}.\mathcal{D}_U$, $\mathcal{CH}.\mathcal{D}_x$ and $\mathcal{CH}.\mathcal{D}_J$.

The way the data is generated allows for efficient tailored implementations of the two routines, if the neighbors of each facet are known.

- i) The facet that contains x can be found by starting from some initial facet and tracking neighboring facets towards x , for example by following $a + s(x - a)$ as s increases from 0 to 1, where a is some point in the facet. A good initial guess can be given by the facet in which the last point was located, if the system state does not change too fast from one time instant to the next. If this point was added to the convex hull, then this facet does not exist anymore. In this case any facet attached to this newly added point can be taken as initial guess. Another more advanced initial guess can be provided by saving for every point in the convex hull its subsequent point. If the subsequent point was not added, then an extreme point of the facet where the subsequent point was located is stored. This information provides for each point in the convex hull a good guess of its subsequent point and this can be used as basis for the initial guess. Thereby, the system dynamic gets also included implicitly in the prediction from the initial guess.
- ii) A data point that needs to be added to the convex hull is already located in the triangulation of $\text{conv}\mathcal{D}_x(k)$ by the previous warm start generation. Hence, by starting from this facet and checking the neighboring facets successively, all facets that need to be updated can be found.

The information about neighboring facets are stored in the graph $\mathcal{CH}.\mathcal{G}_{xJ}$, which needs to be updated whenever the convex hull is.

B.1. Generating the spatial warm start solution from the convex hull object

Let x be the point where we want to generate the warm start solution and \mathcal{CH} the convex hull object. The main part in

the warm start generation is the search for the facet that contains x . As discussed above, it is easy to provide a good guess for a point $a \in \mathcal{CH}.\mathcal{D}_x$ that is close to x . Assume the index i of a in $\mathcal{CH}.\mathcal{D}_x$ is given, then the routine takes the inputs \mathcal{CH} , x and i . As outputs it makes sense to not only return the spatial warm start $\Psi_{\text{sw}} = \Psi_{\text{sw}}(x, \mathcal{CH}.\mathcal{D})$ but also the index F of the facet of $\mathcal{CH}.\mathcal{CH}_{\text{XJ}}$ that contains x and the index i of the next initial guess. If x is not contained in any facet, i.e. if $x \notin \text{conv}\mathcal{CH}.\mathcal{D}_x$, then let F be negative, in fact let it be $F = -F_x$ where F_x is the index of a facet of $\mathcal{CH}.\mathcal{CH}_x$ that excludes x by means of x lies outside the supporting halfspace to $\text{conv}\mathcal{D}_x$ at this facet F_x . Summing, the routine can be invoked by $(\Psi_{\text{sw}}, F, i) = \text{generateSW}(\mathcal{CH}, x, i)$. The basic idea in finding the facet that contains x is to start the search at a and track the facets along the line $a + s(x - a)$, $s \in [0, 1]$ until $s = 1$, then we have reached the facet that contains x . If we sometime step out of $\text{conv}\mathcal{CH}.\mathcal{D}_x$, then $x \notin \text{conv}\mathcal{CH}.\mathcal{D}_x$ and we cannot provide a spatial warm start as described in (15) at x , but we can at the point where we stepped out of $\text{conv}\mathcal{CH}.\mathcal{D}_x$ and use this as spatial warm start solution for x . The procedure can be described by the following steps, where the numbers indicate to which lines of pseudo-code in Algorithm 2 the step corresponds to

- a) (1:–6:) Get the initial point a and a list f of indices of the facets in $\mathcal{CH}.\mathcal{CH}_{\text{XJ}}$ that are attached to it.
- b) (7:–16:) Find the facet $F \in f$ in which the vector $v = x - a$ points starting from a .
- c) (17:–27:) If no such facet exists, then v must point out of $\text{conv}\mathcal{CH}.\mathcal{D}_x$, a must lie on the boundary and hence $x \notin \text{conv}\mathcal{CH}.\mathcal{D}_x$. So there must exist at least one facet of $\mathcal{CH}.\mathcal{CH}_x$ attached to a that excludes x . Find the index of this facet, return its negative as F and return the input used at a as spatial warm start. Also return the index of the extreme point of this facet that is closest to x increased by one as next initial guess i .
- d) (28:–34:) Else track the facets along $a + sv$ until $s \geq 1$ or no further facet exists along the line because we stepped out of $\text{conv}\mathcal{CH}.\mathcal{D}_x$.
- e) (35:–41:) If we stepped out before $s = 1$, then find the index of facet of $\mathcal{CH}.\mathcal{CH}_x$ where we stepped out and return its negative as F , further return the index of the extreme point of this that is closest to x increased by one as next initial guess i and the convex combination of the inputs of this facet as spatial warm start Ψ_{sw} .
- f) (42:–46:) Else we have found index F of the facet of $\mathcal{CH}.\mathcal{CH}_{\text{XJ}}$ that contains x and return it. Also return the convex combination of the inputs at this facet as spatial warm start and the index of the extreme point of this that is closest to x increased by one as next initial guess i .

Algorithm 2 Spatial warm start generation

$(\Psi_{\text{sw}}, F, i) = \text{generateSW}(\mathcal{CH}, x, i)$

Input: convex hull object \mathcal{CH} , evaluation point x and initial guess i

Output: spatial warm start Ψ_{sw} at x , index F of facet $\mathcal{CH}.\mathcal{CH}_{\text{XJ}}$ that contains x and next initial guess i . If $x \notin \text{conv}\mathcal{CH}.\mathcal{D}_x$, then F is negative and $|F|$ is the index of a facet in $\mathcal{CH}.\mathcal{CH}_x$ that excludes x .

```

1:  $f = \mathcal{CH}.\mathcal{G}_{\text{XJ}}(i)$ 
2: while  $f < 0$  do
3:    $i = -f$ 
4:    $f = \mathcal{CH}.\mathcal{G}_{\text{XJ}}(i)$ 
5: end while
6:  $a = \mathcal{CH}.\mathcal{D}_x(i)$ 
7:  $v = x - a$ 
8:  $s = 0$ 
9: for  $F \in f$  do
10:  if  $\text{pointsInFacet}(\mathcal{CH}, x, F, i)$  then
11:     $E = \mathcal{CH}.\mathcal{CH}_{\text{XJ}}(F)$  without  $i$ 
12:     $d = \text{normal}(\mathcal{CH}.\mathcal{D}_x(E), a)$ 
13:     $s = d^\top (\mathcal{CH}.\mathcal{D}_x(E(1)) - a) / (d^\top v)$ 
14:    break for
15:  end if
16: end for
17: if  $s = 0$  then
18:  for  $F \in \mathcal{CH}.\mathcal{G}_x(i)$ 
19:     $d = \text{normal}(\mathcal{CH}.\mathcal{D}_x(\mathcal{CH}.\mathcal{CH}_x(F)), \mathcal{CH}.c_x)$ 
20:    if  $d^\top (x - \mathcal{CH}.\mathcal{D}_x(\mathcal{CH}.\mathcal{CH}_x(F)(1))) > 0$  then
21:      break for
22:    end if
23:  end for
24:   $f = \mathcal{CH}.\mathcal{CH}_x(F)$ 
25:   $i = \arg \min_{j \in f} \|x - \mathcal{CH}.\mathcal{D}_x(j)\|$ 
26:  return  $(\Psi_{\text{sw}}, F, i) = (\mathcal{CH}.\mathcal{D}_U(i), -F, i + 1)$ 
27: end if
28: while true
29:  if  $s \geq 1$  then break while
30:   $f = \text{getFacets}(\mathcal{CH}, E, x)$ 
31:   $F = f$  without  $F$ 
32:  if  $F$  is empty then break while
33:   $(s, E) = \text{findIntersection}(\mathcal{CH}, a, v, s, F, E)$ 
34: end while
35: if  $s < 1$  then
36:   $F = \text{getFacets}(\mathcal{CH}, E, x)$ 
37:   $i = \arg \min_{j \in E} \|x - \mathcal{CH}.\mathcal{D}_x(j)\|$ 
38:   $c = \text{findConvComb}(\mathcal{CH}, E, a + sv)$ 
39:   $\Psi_{\text{sw}} = \sum_{j=1}^n \mathcal{CH}.\mathcal{D}_U(E(j))c(j)$ 
40:  return  $(\Psi_{\text{sw}}, F, i) = (\Psi_{\text{sw}}, -F, i + 1)$ 
41: end if
42:  $f = \mathcal{CH}.\mathcal{CH}_{\text{XJ}}(F)$ 
43:  $i = \arg \min_{j \in f} \|x - \mathcal{CH}.\mathcal{D}_x(j)\|$ 
44:  $c = \text{findConvComb}(\mathcal{CH}, f, x)$ 
45:  $\Psi_{\text{sw}} = \sum_{j=1}^{n+1} \mathcal{CH}.\mathcal{D}_U(f(j))c(j)$ 
46: return  $(\Psi_{\text{sw}}, F, i) = (\Psi_{\text{sw}}, F, i + 1)$ 

```

A detailed description of the steps in pseudo-code can be found in Algorithm 2, still there are needed some minor supporting routines that we will discuss now to complete the whole procedure of generating the warm start.

- $b = \text{pointsInFacet}(\mathcal{CH}, x, F, i)$ checks if the vector $v = x - \mathcal{CH}.\mathcal{D}_x(i)$ starting from i points inside the facet of $\mathcal{CH}.\mathcal{CH}_{xJ}$ with index F , where i has to be an extreme point of F . The length of v does not matter only the direction, i.e. $x = \mathcal{CH}.\mathcal{D}_x(i) + v$ need not lie inside F for b being true, it is enough if there exists $\varepsilon > 0$ such that $\mathcal{CH}.\mathcal{D}_x(i) + \varepsilon v$ lies inside the facet. Pseudo-code is given in Algorithm 3.
- $d = \text{normal}(A, x)$ is the most used subroutine and it calculates for a given set A of p points in \mathbb{R}^p a vector $d \in \mathbb{R}^p$ that is normal to the hyperplane going through all points in A . This hyperplane cuts \mathbb{R}^p into two halfspaces and d points into the halfspace that does not contain the given point $x \in \mathbb{R}^p$, where x must not lie on the this plane. A pseudo-code description of this subroutine is given in Algorithm 4 and uses the well-known QR-decomposition that decomposes an invertible matrix $M \in \mathbb{R}^{p \times p} = QR$ into an orthogonal matrix $Q = Q^{-\top}$ and an upper triangular matrix R where all diagonal entries of R are positive.
- $f = \text{getFacets}(\mathcal{CH}, E, \text{flag})$ gives out all facets of $\mathcal{CH}.\mathcal{CH}_{\text{flag}}$ that share the extreme points specified in E . f can also be only a single facet or even empty, depending on how many facets exists that share the edge E . Pseudo-code is given in Algorithm 5.
- $f = \text{findIntersection}(\mathcal{CH}, a, v, s, F, E)$ takes an edge E of F such that $a + sv$ lies on E and it gives out (s, E) such that $a + sv$ lies on the edge E of F but with output $s > \text{input } s$. In words it takes the edge and point where the line $a + sv$ enters F and computes the edge and point where it leaves F . Pseudo-code is given in Algorithm 6.
- $f = \text{fincConvComb}(\mathcal{CH}, f, x)$ calculates the weights c such that $\sum_{j=1}^{|f|} c(j) \mathcal{CH}.\mathcal{D}_x(f(j)) = x$ and $\sum_{j=1}^{|f|} c(j) = 1$. Therefore it must be verified that it is possible to convex combine the points in f to x . Pseudo-code is given in Algorithm 7.

B.2. Updating the convex hull

Second we consider updating the convex hull, therefore let \mathcal{CH} be the convex hull object and (U, x) the input sequence and point in state space we want to add. In addition we already know the index F of the facet of $\mathcal{CH}.\mathcal{CH}_{xJ}$ that contains x from the previous spatial warm start generation at exactly this point x . This makes up the inputs of the routine and the output is of course the updated convex hull object \mathcal{CH} , hence it can be invoked by $\mathcal{CH} = \text{updateCH}(\mathcal{CH}, x, U, F)$. The basic idea of updating \mathcal{CH} is that we start at the facet F , which must be for

Algorithm 3 Check if vector points in facet

$b = \text{pointsInFacet}(\mathcal{CH}, x, F, i)$

Input: convex hull object \mathcal{CH} , vector x , facet F and index i of starting point

Output: boolean b , true if $v = x - \mathcal{CH}.\mathcal{D}_x(i)$ points in F starting from i .

```

1:  $E = \mathcal{CH}.\mathcal{CH}_{xJ}(F)$ 
2: for  $j = 1, 2, \dots, n + 1$ 
3:   if  $E(j) \neq i$  then
4:      $\tilde{E} = E$  without  $E(j)$ 
5:      $d = \text{normal}(\mathcal{CH}.\mathcal{D}_x(\tilde{E}), \mathcal{CH}.\mathcal{D}_x(E(j)))$ 
6:     if  $d^\top(x - \mathcal{CH}.\mathcal{D}_x(i)) > 0$  then
7:       return  $b = \text{false}$ 
8:     end if
9:   end if
10: end for
11: return  $b = \text{true}$ 
```

Algorithm 4 Calculate normal vector

$d = \text{normal}(A, x)$

Input: list A of n points in \mathbb{R}^n , point $x \in \mathbb{R}^n$

Output: vector d that is normal to the plane spanned by A pointing into the halfspace that does not contain x

```

1: for  $j = 1, \dots, n - 1$ 
2:    $A(j) = A(j) - A(n)$ 
3: end for
4:  $A(n) = x - A(n)$ 
5: QR-decomposition  $QR = [A(1), A(2), \dots, A(n)]$ 
6: return  $d = - \text{last column of } Q$ 
```

Algorithm 5 Get facets attached to set of extreme points

$f = \text{getFacets}(\mathcal{CH}, E, \text{flag})$

Input: convex hull object \mathcal{CH} , list E of extreme points and flag that can either be x or xJ indicating the convex hull

Output: list f of facets of $\mathcal{CH}.\mathcal{CH}_{\text{flag}}$ that contain all extreme points given in E

```

1:  $f = \mathcal{CH}.\mathcal{G}_{\text{flag}}(E(1))$ 
2: for  $p \in E$  without  $E(1)$  do
3:    $f = \text{common elements of } f \text{ and } \mathcal{CH}.\mathcal{G}_{\text{flag}}(p)$ 
4: end for
5: return  $f$ 
```

Algorithm 6 Find intersection of line and facet boundary

 $(s, E) = \text{findIntersection}(\mathcal{CH}, a, v, s, F, E)$ **Input:** convex hull object \mathcal{CH} , start point a , direction vector v , current s , facet F , edge E of F with $a + sv$ lying on this edge**Output:** s and E such that $a + sv$ lies on end E of facet F and output $s > \text{input } s$,

```

1:  $p = \mathcal{CH}.\mathcal{CH}_{xJ}(F)$  without  $E$ 
2: for  $j = 1, 2, \dots, n$ 
3:    $\bar{E}(j) = E$  without  $E(j)$  but with  $p$ 
4:    $d = \text{normal}(\mathcal{CH}.\mathcal{D}_x(\bar{E}), \mathcal{CH}.\mathcal{D}_x(E(j)))$ 
5:    $\bar{s}(j) = d^\top (\mathcal{CH}.\mathcal{D}_x(p) - a) / (d^\top v)$ 
6:   if  $\bar{s}(j) \leq s$  then  $\bar{s}(j) = \infty$ 
7: end for
8:  $\bar{j} = \text{argmin}_j \bar{s}(j)$ 
9: return  $(s, E) = (\bar{s}(\bar{j}), \bar{E}(\bar{j}))$ 

```

Algorithm 7 Find convex combination

 $c = \text{findConvComb}(\mathcal{CH}, E, x)$ **Input:** convex hull object \mathcal{CH} , index array E of points in $\mathcal{CH}.\mathcal{D}_x$, point x **Output:** vector c such that the points in E summed up with the weights in c equal x and sum over elements in c is 1

```

1:  $A = [\mathcal{CH}.\mathcal{D}_x(E)^\top \mathbb{1}^\top]^\top$ 
2:  $b = [x^\top \mathbb{1}^\top]^\top$ 
3: Solve  $Ac = b$  for  $c$ 
4: return  $c$ 

```

sure updated since there is always an improvement in the optimization update operator as long as we have not reached exact optimality and even in that unlikely case the point we want to add lies directly on the facet and it makes no difference to add it and split the facet. Thus we always update the facet F and then we start searching from F for neighboring facets that need to be updated and if we found one, then we also search in its neighbors for facets to update. Since the convex hull was convex before and has to be convex afterwards the set of facets to update must be connected and thus we will find all facets to update by that procedure. A special case is if $x \notin \text{conv } \mathcal{CH}.\mathcal{D}_x$, i.e. $F < 0$, then we need to find all facets of $\mathcal{CH}.\mathcal{CH}_x$ that exclude x and check that facets of \mathcal{CH}_{xJ} attached to them if they must be updated. In this case we must also update $\mathcal{CH}.\mathcal{CH}_x$ which we do by the same procedure, starting from $-F$ search for neighboring facets that must to be updated. In fact all facets of $\mathcal{CH}.\mathcal{CH}_x$ that must be updated are facets that exclude x and the other way round, so we search them, update them and initialize with the facets of $\mathcal{CH}.\mathcal{CH}_{xJ}$ attached to them the search for updates in $\mathcal{CH}.\mathcal{CH}_{xJ}$. In more detail the procedure can be described by the following steps, where the numbers indicate to which lines of pseudo-code in Algorithm 8 the step corresponds to

a) (1:–5:) Append data point to data set $\mathcal{CH}.\mathcal{D}$.

Algorithm 8 Update convex hull object

 $\mathcal{CH} = \text{updateCH}(\mathcal{CH}, x, U, F)$ **Input:** convex hull object \mathcal{CH} , new point x , input sequence U , index F of facet in $\mathcal{CH}.\mathcal{CH}_{xJ}$ that contains x , negative if $x \notin \text{conv } \mathcal{CH}.\mathcal{D}_x$, then $|F|$ is facet of $\mathcal{CH}.\mathcal{CH}_x$ that excludes x .**Output:** updated convex hull object \mathcal{CH} that contains the new point

```

1: Append  $x$  to  $\mathcal{CH}.\mathcal{D}_x$ 
2: Append  $U$  to  $\mathcal{CH}.\mathcal{D}_U$ 
3: Append  $J_N(U, x)$  to  $\mathcal{CH}.\mathcal{D}_J$ 
4:  $z = [x^\top J_N(U, x)]^\top$ 
5:  $k = \text{number of elements in } \mathcal{D}_x$ 
6: if  $F < 0$  then
7:    $\mathcal{E}_{\text{toTestx}} = \text{list of all edges of facet } \mathcal{CH}.\mathcal{CH}_x(-F)$ 
8:    $\mathcal{E}_{\text{toTestxJ}} = \text{list with one element } \mathcal{CH}.\mathcal{CH}_x(-F)$ 
9:    $\mathcal{CH} = \text{removeFacet}(\mathcal{CH}, -F, x)$ 
10:  while  $\mathcal{E}_{\text{toTestx}}$  is not empty do
11:     $F = \text{getFacets}(\mathcal{CH}, \mathcal{E}_{\text{toTestx}}(1), x)$ 
12:     $E = \mathcal{E}_{\text{toTestx}}(1)$ 
13:    if  $F$  is not empty then
14:       $d = \text{normal}(\mathcal{CH}.\mathcal{D}_x(\mathcal{CH}.\mathcal{CH}_x(F)), \mathcal{CH}.\mathcal{C}_x)$ 
15:      if  $d^\top (x - \mathcal{CH}.\mathcal{D}_x(E(1))) > 0$  then
16:        append  $\mathcal{CH}.\mathcal{CH}_x(F)$  to  $\mathcal{E}_{\text{toTestxJ}}$ 
17:         $\mathcal{CH} = \text{removeFacet}(\mathcal{CH}, F, x)$ 
18:        append edges of  $F$  except  $E$  to  $\mathcal{E}_{\text{toTestx}}$ 
19:      else
20:         $\mathcal{CH} = \text{addFacet}(\mathcal{CH}, [E, k], x)$ 
21:      end if
22:    end if
23:    remove  $E$  from  $\mathcal{E}_{\text{toTestx}}$ 
24:  end while
25: else
26:    $\mathcal{E}_{\text{toTestxJ}} = \text{list of all edges of facet } \mathcal{CH}.\mathcal{CH}_{xJ}(F)$ 
27:    $\mathcal{CH} = \text{removeFacet}(\mathcal{CH}, F, xJ)$ 
28:  end if
29:  while  $\mathcal{E}_{\text{toTestxJ}}$  is not empty do
30:     $E = \mathcal{E}_{\text{toTestxJ}}(1)$ 
31:     $F = \text{getFacets}(\mathcal{CH}, E, xJ)$ 
32:    if  $F$  is not empty then
33:       $d = \text{normal}(\mathcal{CH}.\mathcal{D}_{xJ}(\mathcal{CH}.\mathcal{CH}_{xJ}(F)), \mathcal{CH}.\mathcal{C}_{xJ})$ 
34:      if  $d^\top (z - \mathcal{CH}.\mathcal{D}_{xJ}(E(1))) > 0$  then
35:         $\mathcal{CH} = \text{removeFacet}(\mathcal{CH}, F, xJ)$ 
36:        append edges of  $F$  except  $E$  to  $\mathcal{E}_{\text{toTestxJ}}$ 
37:      else
38:         $\mathcal{CH} = \text{addFacet}(\mathcal{CH}, [E, k], x)$ 
39:      end if
40:    end if
41:    remove  $E$  from  $\mathcal{E}_{\text{toTestxJ}}$ 
42:  end while
43:  $\mathcal{CH}.\mathcal{C}_x = \mathcal{CH}.\mathcal{C}_x(k-1)/k + x/k$ 
44:  $\mathcal{CH}.\mathcal{C}_{xJ} = \mathcal{CH}.\mathcal{C}_{xJ}(k-1)/k + z/k$ 
45: return  $\mathcal{CH}$ 

```

- b) (6:–24:) If $F < 0$ then $x \notin \text{convCH}.\mathcal{D}_x$ and we need to update $\text{CH}.\text{CH}_x$ first. We know that $-F$ is one facet that needs to be updated so we remove it and check all neighboring facets if they must also be updated. If so, then we remove them and also add their edges to the list of edges whose neighboring facets need to be checked. If not then we take the edge between the facet that had been removed and the one that stays and add it together with the new point k as new facet to $\text{CH}.\text{CH}_x$. Whenever we remove a facet we also add it to the list of edges we need to check for updating $\text{CH}.\text{CH}_{xJ}$.
- c) (25:–28) If $F > 0$ then we remove this facet from $\text{CH}.\text{CH}_{xJ}$ and initialize the list of edges we need to check with all edges of this facet.
- d) (29:–42) Check edges as long as there are some whether their adjacent facets need to be updated or not, if so remove them and add their edges to the list, if not add the edge with the new point as new facet to $\text{CH}.\text{CH}_{xJ}$.
- e) (43:–45:) Update the center points of the convex hulls and return the updated convex hull object.

Whenever we talked about adding a facet to or removing it from $\text{CH}.\text{CH}_{x/xJ}$ we of course also have to update the graph $\text{CH}.\mathcal{G}_{x/xJ}$. A detailed explanation on the subroutines adding and removing shall be given now.

- $\text{CH} = \text{removeFacet}(\text{CH}, F, k, \text{flag})$ removes the facet F from the $\text{CH}.\text{CH}_{\text{flag}}$ and also removes its appearance in $\text{CH}.\mathcal{G}_{\text{flag}}$. When removing F we still need to keep the slot F in $\text{CH}.\text{CH}_{\text{flag}}$ because if we would delete it, all facets coming after F in the list $\text{CH}.\text{CH}_{\text{flag}}$ would get their index decremented and we would need to change the whole $\text{CH}.\mathcal{G}_{\text{flag}}$. Hence it is easier to add F to a list $\text{CH}.\text{o}_{\text{flag}}$ of open slots that can be overwritten, thereby keeping all other indices as they are. If it happens for $\text{flag} = xJ$ that thereby an extreme point p of F is afterwards not attached to any facet at all, i.e. with adding the new data point p 'moves' from the boundary of the convex hull to its interior, then we store for p the information that it has fallen out of the boundary when updating k by setting $\text{CH}.\mathcal{G}_{xJ}(p) = -k$. This is necessary for the initial guess for the warm start generation where it could happen that for some point the initial guess is p , but if p is not attached to any facet then we cannot start a search from there. Therefore we store that p has been overwritten by k and then we can start the search from a facet attached to k instead.
- $\text{CH} = \text{addFacet}(\text{CH}, f, \text{flag})$ adds the facet with extreme points listed in f to $\text{CH}.\text{CH}_{\text{flag}}$ and updates the graph $\text{CH}.\text{CH}_{\text{flag}}$. If there is an open slot $\text{CH}.\text{o}_{\text{flag}}$ in $\text{CH}.\text{CH}_{\text{flag}}$ then we can overwrite it, otherwise we append the new facet to the list $\text{CH}.\text{CH}_{\text{flag}}$.

Another point we have not discussed here is the initialization of the convex hull. The algorithms presented here only work if there already exists a convex hull object where there are enough points to build the convex hulls and they have to be correct obviously. A straight forward approach is to wait until the $n + 1$ data points are available and initialize the convex hull $\text{CH}.\text{CH}_{xJ}$ with the first and only facet $[1 \dots n + 1]$ as well as the convex hull $\text{CH}.\text{CH}_x$ with all $n + 1$ possible combinations of n points out of these $n + 1$ points.

Algorithm 9 Remove facet from convex hull object

$\text{CH} = \text{removeFacet}(\text{CH}, F, k, \text{flag})$

Input: convex hull object CH , facet F of $\text{CH}.\text{CH}_{\text{flag}}$, current index k and flag that is either x or xJ indicating the convex hull

Output: updated convex hull object CH without F

```

1: for  $p \in \text{CH}.\text{CH}_{\text{flag}}(F)$  do
2:   remove  $F$  from  $\text{CH}.\mathcal{G}_{\text{flag}}(p)$ 
3:   if  $\text{CH}.\mathcal{G}_{\text{flag}}(p)$  is empty and  $\text{flag} = xJ$  then
4:      $\text{CH}.\mathcal{G}_{\text{flag}}(p) = -k$ 
5:   end if
6: end for
7: add  $F$  to  $\text{CH}.\text{o}_{\text{flag}}$ 
8: return  $\text{CH}$ 
```

Algorithm 10 Add facet to convex hull object

$\text{CH} = \text{addFacet}(\text{CH}, f, \text{flag})$

Input: convex hull object CH , list f of extreme points of facet that is added to $\text{CH}.\text{CH}_{\text{flag}}$, current index k and flag that is either x or xJ indicating the convex hull

Output: updated convex hull object CH with facet f

```

1: if  $\text{CH}.\text{o}_{\text{flag}}$  is not empty then
2:    $F = \text{CH}.\text{o}_{\text{flag}}(1)$ 
3:   change  $\text{CH}.\text{CH}_{\text{flag}}(F)$  to  $f$ 
4: else
5:    $F = \text{length of } \text{CH}.\text{CH}_{\text{flag}} + 1$ 
6: end if
7: for  $p \in f$  do
8:   append  $F$  to  $\text{CH}.\mathcal{G}(p)$ 
9: end for
10: remove  $F$  from  $\text{CH}.\text{o}_{\text{flag}}$ 
11: return  $\text{CH}$ 
```

C. Computation of the Lipschitz constant from Section 4.2

The Lipschitz constant $L(U)$ of the cost function $J_N(U, \cdot)$ can be computed as

$$L(U) = \sum_{i=0}^N (L_i + \delta_{Ni}(L_F - L_i)) \prod_{j=0}^{i-1} L_f(\Pi_j U) \quad (29)$$

where δ_{Ni} denotes the Kronecker-delta that is 1 if $i = N$ and 0 else, where L_l , L_F and L_f are the Lipschitz constants of the stage cost l , the terminal cost F and the system dynamic f , respectively and where $\Pi_j = [\dots 0, I_m, 0, \dots] \in \mathbb{R}^{m \times Nm}$ is a projection matrix, that projects U onto its $mj+1$ st till $mj+m$ th component. The Lipschitz constants can be upper estimated as

$$L_l = \sqrt{\frac{13}{50}}, \quad L_F = 100L_l \quad (30a)$$

$$L_f(u) = \sqrt{1 + T_s|u_1| \sqrt{1 + T_s^2 u_1^2/4} + T_s^2 u_1^2/2}. \quad (30b)$$

by the following calculations

$$\begin{aligned} \left(\frac{\partial l}{\partial x_3}(x, u) \right)^2 &= \left(0.1 \sin\left(\frac{x_3}{2}\right) \cos\left(\frac{x_3}{2}\right) \right)^2 \leq 0.1^2 \\ \left(\frac{\partial l}{\partial x_{1,2}}(x, u) \right)^2 &= \left(\frac{2x_{1,2}}{4(1+x_1^2+x_2^2)^{3/4}} \right)^2 \\ &= \frac{1}{4} \sqrt{\frac{x_{1,2}^4}{(1+x_1^2+x_2^2)^3}} \\ &\leq \frac{1}{4} \sqrt{\frac{x_{1,2}^4}{(1+x_{1,2}^2)^3}} \\ &= \frac{1}{4} \sqrt{\frac{x_{1,2}^4}{1+3x_{1,2}^2+3x_{1,2}^4+x_{1,2}^6}} \leq \frac{1}{8} \\ \left\| \frac{\partial l}{\partial x}(x, u) \right\| &\leq \sqrt{0.1^2 + \frac{1}{4}} = \sqrt{\frac{13}{50}} \end{aligned}$$

and

$$\begin{aligned} \frac{\partial f}{\partial x}(x, u) &= \begin{bmatrix} 1 & 0 & -T_s u_1 \sin(x_3) \\ 0 & 1 & T_s u_1 \cos(x_3) \\ 0 & 0 & 1 \end{bmatrix} \\ \Rightarrow \frac{\partial f}{\partial x}(x, u)^\top \frac{\partial f}{\partial x}(x, u) &= \\ &\begin{bmatrix} 1 & 0 & -T_s u_1 \sin(x_3) \\ 0 & 1 & T_s u_1 \cos(x_3) \\ -T_s u_1 \sin(x_3) & T_s u_1 \cos(x_3) & T_s^2 u_1^2 + 1 \end{bmatrix} \end{aligned}$$

which has the eigenvalues 1 and $1 \pm T_s u_1 \sqrt{1 + T_s^2 u_1^2/4} + T_s^2 u_1^2/2$. Thus the maximum absolute eigenvalue is

$$\left\| \frac{\partial f}{\partial x}(x, u) \right\|^2 = 1 + T_s|u_1| \sqrt{1 + T_s^2 u_1^2/4} + T_s^2 u_1^2/2.$$