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

Lukas Schwenkel\textsuperscript{1,2}, Meriem Gharbi\textsuperscript{1}, Sebastian Trimpe\textsuperscript{2}, and Christian Ebenbauer\textsuperscript{1}

\textsuperscript{1}Institute for Systems Theory and Automatic Control, University of Stuttgart, Pfaffenwaldring 9, 70550 Stuttgart, Germany
\textsuperscript{2}Intelligent Control Systems Group, Max Planck Institute for Intelligent Systems, Heisenbergstr. 3, 70569 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. The suboptimality of the applied control input resulting from the real-time requirements is shown to vanish over time as more and more data is collected. 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 recover the value function on the limit set of the closed-loop state trajectory. Simulative examples show that existing real-time MPC schemes can be improved by storing data and the proposed learning 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,\cite{3}. As a consequence, an input sequence for the whole prediction horizon and its resulting state trajectory are computed at every sampling instant thus continuously generating a lot 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 memorized. Our daily life is full of receding horizon decision making problems and, there, we often memorize decisions and successively improve them in recurrent situations. For instance, in the daily drive to work, we optimize our driving style by memorizing and successfully improving the route, gear shifting, etc. Motivated by this, the key question addressed in this article is how to leverage on optimization data and how to introduce memory and online learning in real-time MPC algorithms in order to improve their performance.

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 in the long run, 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 is retained.

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 lack of computation time the MPC optimization problem can not 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 straightforward 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
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 \([10] , [23] , [25] , [30] , and [38]\), but also online, \([3] , [8] , [13] , [28]\), or first learned offline and then refined online \([20] , [26]\). Nevertheless, only few of these works, namely \([3] , [10] , [13] , and [26]\), 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 \([1] , [12] , [16] , [21] , [22] , [29] , [31] , and [41]\), while support vector machines are used in \([11]\), and the learning problem is formulated as quadratically constrained quadratic program in \([15]\). 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 this 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. Only the works \([15] \) and \([21]\) can guarantee stability.

The paper \([33]\) belongs to the third category \((iii)\). The applied 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.

This article proposes a novel way of leveraging data in MPC, which does not match any of the aforementioned categories. 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 make use of the following notation: \(\mathbb{R}_+ , \mathbb{R}_{++} \) and \(\mathbb{N}_+\) 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 \(\mathcal{S}_n^+\). For \(M \in \mathbb{S}_{++}\), we define \(\|x\|_M := (x^T M x)^{1/2}\). Further, \(B^p_\mathcal{E}(x)\) is an open \(n\)-dimensional Euclidean ball with radius \(\varepsilon \in \mathbb{R}_{++}\) around \(x\) and \(\overline{B}^p_\mathcal{E}(x)\) is its
closure. A function $\alpha : \mathbb{R}_+ \to \mathbb{R}_+$ with $\alpha(0) = 0$ is a $\mathcal{K}$-function if it is continuous and strictly increasing; it is a $\mathcal{K}_\infty$-function if $\alpha \in \mathcal{K}$ and in addition $\alpha(s) \to \infty$ as $s \to \infty$. The interior of a set $A \subset \mathbb{R}^n$ is denoted int$A$ and the convex hull is conv$A$. For a function $f : \mathbb{R}^n \to \mathbb{R}$ the epigraph is
\[ \text{epi } f = \{(x, \mu) \in \mathbb{R}^n \times \mathbb{R} | f(x) \leq \mu \}. \] (1)

2. Problem formulation: 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
\[ x(k+1) = f(x(k), u(k)) + w(k) \] (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 $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. 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
\[
J_N(U,x(k)) = \sum_{j=0}^{N-1} l(x_j, u_j) + F(x_N) \\
J_N^*(x(k)) = \min_U J_N(U,x(k)) \tag{3}
\]
s.t. $x_{j+1} = f(x_j, u_j), \quad j = 0, \ldots, N-1$ \\
$x_0 = x(k)$

where $U = [u_0^T, \ldots, u_{N-1}^T]^T \in \mathbb{R}^{Nm}$ denotes the stacked control inputs over the finite prediction horizon $N \in \mathbb{N}_+$, and $l : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}_+$ and $F : \mathbb{R}^n \to \mathbb{R}_+$ denote the stage and terminal costs, respectively. 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
\[
U(k+1) = \Phi^{\tau}(k) \Psi_U(U(k), x(k)) \tag{4a} \\
u(k+1) = \Pi_U U(k+1) \tag{4b}
\]
where $\Phi^{\tau}(k) : \mathbb{R}^{Nm} \times \mathbb{R}^n \to \mathbb{R}^{Nm}$ represents the optimization algorithm and $\Pi_U = [I_m 0 \ldots 0] \in \mathbb{R}^{m \times Nm}$ is a projection matrix selecting the input to be actually applied. We divide the optimization algorithm $\Phi^{\tau}(k)$ into a warm start operator $\Psi_w : \mathbb{R}^{Nm} \times \mathbb{R}^n \to \mathbb{R}^{Nm}$ and an optimization update operator $\Psi_o : \mathbb{R}^{Nm} \times \mathbb{R}^n \to \mathbb{R}^{Nm}$, which is iterated $\tau(k) \in \mathbb{N}_+$ times
\[
\Phi^0(U,x) = \Psi_w(U,x) \tag{5a} \\
\Phi^i(U,x) = \Psi_o(\Phi^{i-1}(U,x), f(x, \Pi_U U)). \tag{5b}
\]
In this way, an input for the nominal next state $f(x, \Pi_U U)$ 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 $\Psi_w$ as temporal warm start operator. If the closed loop is stable for any number of optimization algorithm iterations $\tau(k)$, the real-time scheme is also called anytime MPC, [5], [19]. A generic result on closed-loop stability of an anytime iteration scheme is proven in [19] and stated in the following theorem.

**Theorem 2.1.** 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 $\underline{\alpha}, \overline{\alpha} \in \mathcal{K}_\infty$ such that
\[ \underline{\alpha}(\|\Pi_U U\|_2) \leq J_N(U,x) \leq \overline{\alpha}(\|\Pi_U U\|_2) \] (6)
and that the warm start and the optimization update operator fulfill
\[
J_N(\Psi_w(U,x), f(x, \Pi_U U)) - J_N(U,x) \leq -l(x, \Pi_U U) \tag{7a} \\
J_N(\Psi_o(U,x)) - J_N(U,x) \leq -\gamma(U,x) \tag{7b}
\]
for all $(U,x) \in \mathbb{R}^{Nm} \times \mathbb{R}^n$, where $\gamma : \mathbb{R}^{Nm} \times \mathbb{R}^n \to \mathbb{R}_+$ is a non-negative continuous function with $\gamma(U,x) = 0$ if $J_N(U,x) = J_N^*(x)$. Then, for any sequence $\{\tau(0), \tau(1), \ldots\}$, 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. So \( t_1(k) \) is typically small, and 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 computed input sequences, which are calculated at previous time steps and which are usually discarded (except the last input sequence generating a temporal warm start). The core idea we pursue is rather straightforward. 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. A main challenge of this approach is to make ‘close to a point’ mathematically precise, i.e. which previous input sequence (or combinations of multiple ones) to take as warm start solution at a given location and to show that the learning scheme asymptotically recovers the value function and 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 \( D(k) \) by \( \Psi_{sw}(\cdot, D(k)) : \mathbb{R}^n \to \mathbb{R}^{N_m} \) and call it spatial to distinguish it from the temporal one. The data \( D(k) \) we store is the set containing the input sequence \( U(j) \) for each time instant \( 1 \leq j \leq k \), 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
\[
D(k) = D(k-1) \cup \{ (z_k, J_N(z_k)) \}, \quad D(0) = \emptyset \tag{8a}
\]
with
\[
z_k = (U(k), f(x(k-1), \Pi_0 U(k-1))). \tag{8b}
\]
For the subsequent analysis, we introduce the following convenient notation
\[
D_k := \{ x \in \mathbb{R}^n \mid \exists (U,x,J) \in D \} \tag{8c}
\]
\[
D_{kl} := \{ (x,J) \in \mathbb{R}^n \times \mathbb{R}_+ \mid (U,x,J) \in D \} \tag{8d}
\]
\[
D_U := \{ U \in \mathbb{R}^{N_m} \mid \exists (U,x,J) \in D \} \tag{8e}
\]
and we denote an approximation of \( J_N^*(x) \) based on the collected data \( D(k) \) by \( J_N^*(x, D(k)) \).

In a nominal MPC stabilization problem, we cannot expect that the system state repeatedly arrives at the same points in the state space except for the origin, for which the optimal input is trivial. In a real-world scenario, however, there are disturbances, periodic operation conditions, set point changes, or reference signals that lead to other points than the origin being visited several times. Thus, learning the optimal control at these points is a meaningful task. We generically model such situations with the signal \( w(k) \), and we assume that \( w(k) \) is such that the state of the closed loop \( (U(k),x(k)) \) stays bounded. This ensures that the omega limit set of the sequence of points for which an input is calculated,
\[
\Omega = \{ y \in \mathbb{R}^n \mid \exists k_i \to \infty : f(x(k_i), \Pi_0 U(k_i)) \to y \} \tag{9}
\]
is nonempty due to the Bolzano-Weierstrass theorem and that \( \Omega \) is approached by \( f(x(k), \Pi_0 U(k)) \) as \( k \to \infty \). A point in \( \Omega \) is called limit or recurrent point and is basically a point that 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 \( \Psi_{sw} \) such that it will converge to the optimal policy for all \( x \in \Omega \) as \( k \to \infty \), i.e.
\[
\forall x \in \Omega : \lim_{k \to \infty} J_N(\Psi_{sw}(x, D(k)), x) = J_N^*(x). \tag{10}
\]
Hence, we want to learn the value function \( J_N^* \) on \( \Omega \) 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 known 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
\[
\Phi^0(U,x,D) = \begin{cases} \Psi_{tw}(U,x) & \text{if } J_{Ns} < J_{Ns} \\ \Psi_{sw}(f(x, \Pi_0 U), D) & \text{else,} \end{cases} \tag{11a}
\]
where
\[
J_{Ns} = J_N(\Psi_{sw}(U(x), f(x, \Pi_0 U))) \tag{11b}
\]
\[
J_{Ns} = J_N(\Psi_{sw}(f(x, \Pi_0 U, D), f(x, \Pi_0 U))). \tag{11c}
\]
Through this straightforward approach, the resulting online learning scheme is inherently stable, which we state in the following corollary.

**Corollary 2.1.** 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 possible spatial warm start operators \( \Psi_{sw} \).

Overall, the proposed MPC iteration scheme measures at every time instant the current system state, generates the warm starts, and chooses the better one. Then it proceeds with the optimization iteration and collects its data, before it finally applies the first portion of the input sequence to the system. A pseudocode description of this procedure is given in Algorithm 1. The main focus of this article is line 4 and 14 of the algorithm. More specifically, we will design in Section 3 and 4 spatial warm start operators for linear and nonlinear MPC, respectively. Through-out the article, we assume that a temporal warm start satisfying the conditions in Theorem 2.1 is given.
Algorithm 1 Real-time MPC scheme with learning

1: for \( k = 1, 2, \ldots \) do
2: \hspace{1em} measure the current system state:
3: \hspace{1em} obtain \( x(k) \)
4: \hspace{1em} generate warm starts:
5: \hspace{2em} temporal: \( U_i = \Psi_{sw}(U(k), x(k)) \)
6: \hspace{2em} spatial: \( U_s = \Psi_{sw}(f(x(k), \Pi_0 U(k)), D(k)) \)
7: if \( J_{Nt} < J_{Na} \) according to (11) then
8: \hspace{3em} \( U_t = U_i \)
9: else
10: \hspace{3em} \( U_t = U_s \)
11: end if
12: end for
13: \( U(k+1) = U_t \)  \hspace{1em} memorize data:
14: \( D(k+1) = D(k) \cup \{ x(k), J_N(x(k)) \} \) as per (8)
15: apply first part of input sequence:
16: \( U(k+1) = \Pi_0 U(k+1) \)

3. Leveraging data in real-time linear MPC

In this section, we assume \( f \) in (2) is linear in the state and the input

\[
 f(x, u) = Ax + Bu, \tag{12}
\]

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

\[
 l(x, u) = \|x\|^2_Q + \|u\|^2_R + \epsilon \bar{B}_x(x) + \epsilon \bar{B}_u(u), \hspace{1em} \tag{13}
\]

\[
 F(x) = \|x\|^2_R, \hspace{1em} \tag{14}
\]

with the design parameters \( \epsilon \in \mathbb{R}_+, Q \in S^n_+, R \in S^m_+ \), as well as \( P \in S^n_+ \) resulting from \( \epsilon, Q, R \) 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 2.1 are satisfied. Moreover, it has been shown in [19] that constraint satisfaction can be guaranteed with a finite barrier parameter \( \epsilon \) and suitable assumptions. We will refer to this scheme as linear anytime 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, discuss an algorithmic implementation and demonstrate the method in an example.

3.1. A spatial warm start that guarantees convergence

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 \( \bar{x} \in \text{conv} D_x(k) \) by

\[
 (\Psi_{sw}(\bar{x}, D), \bar{x}, J_N(\bar{x}, D)) = \arg\min\limits_{(U, x) \in \text{conv} D} J \tag{15a}
\]

\[
 J_N(\bar{x}, D) = \min_{x = \bar{x}} \min\limits_{(U, x) \in \text{conv} D} J. \tag{15b}
\]

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

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

In the remainder of this section we will show that this spatial warm start converges to the optimal policy at recurrent points (9) by the following steps: First, we show that \( J_N(x, D(k)) \) upper bounds \( J_N(\Psi_{sw}(x, D(k)), x) \) if \( J_N \) is convex (Lemma 3.1); second, we show that \( J_N \) is convex (Lemma 3.2); and finally, we show that the functions \( J_N(\cdot, D(k)) \) are decreasing with every new data point and converging to the value function at recurrent points (Theorem 3.1).

Lemma 3.1. Let \( J_N : \mathbb{R}^{nm} \times \mathbb{R}^n \to \mathbb{R}_+ \) be convex with respect to \((U, x)\). Further, let \( \varepsilon_j \in \mathbb{R}^{nm} \times \mathbb{R}_+^n \), \( j \in \mathbb{N} \) be any sequence,
and let \( D(j) \subset \mathbb{R}^{Nm} \times \mathbb{R}^n \times \mathbb{R}_+ \) be the corresponding sequence of data sets defined in (8a). Moreover, let \( \Psi_{sw} (\cdot, D(j)) : \text{conv} \, D_x(j) \to \mathbb{R}^{Nm} \) and \( J_N^r (\cdot, D(j)) : \text{conv} \, D_x(j) \to \mathbb{R}_+ \) be defined in (15) and let \( k \in \mathbb{N}_+ \), then

(i) \( J_N (\Psi_{sw}(x, D(k)), x) \leq J_N^r (x, D(k)) \).

(ii) \( J_N^r (\cdot, D(k)) \) is convex.

Proof. (i) Since \( J_N \) is convex and the spatial warm start defined in (15) is a convex combination of data points \((U_i, x_i, J_i) \in D(k)\), we have

\[
(\Psi_{sw}(x, D(k)), x, J_N^r (x, D(k))) = \sum_{i=1}^{K} \lambda_i (U_i, x_i, J_i)
\]

and further

\[
J_N (\Psi_{sw}(x, D(k)), x) = J_N \left( \sum_{i=1}^{K} \lambda_i (U_i, x_i) \right)
\]

\[
\leq \sum_{i=1}^{K} \lambda_i J_N (U_i, x_i)
\]

\[
= \sum_{i=1}^{K} \lambda_i J_i = J_N^r (x, D(k)).
\]

(ii) We slightly reformulate (15)

\[
J_N (x, D(k)) = \min_{(z,J) \in \text{conv} \, D_x(k)} J
\]

and show that the epigraph of \( J_N^r (\cdot, D(k)) : \text{conv} \, D_x(k) \to \mathbb{R}_+ \) is convex:

\[
\text{epi} J_N^r (\cdot, D(k)) = \{(x, J) \mid J \geq \min_{(z,J) \in \text{conv} \, D_x(k)} J \}
\]

\[
= \{(x, J) \mid J \geq \min_{(z,J) \in \text{conv} \, D_x(k)} J \}
\]

\[
= \{(x, J + \alpha) \mid \alpha \geq 0 \land (x, J) \in \text{conv} \, D_x(k) \}
\]

\[
= \text{conv} \, D_x(k) \oplus \{(0) \times [0, \infty)\}
\]

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

\( \square \)

In order to use Lemma 3.1, the following lemma states that \( J_N \) given by the linear anytime MPC is indeed convex.

**Lemma 3.2.** The cost function \( J_N : \mathbb{R}^{Nm} \times \mathbb{R}^n \to \mathbb{R}_+ \) of linear anytime MPC, i.e. (3) with (12), (13), (14) is convex in \((U, x)\).

**Proof.** It is shown in [19] that the cost function \( J_N \) can be written under these assumptions as

\[
J_N (U, x) = \frac{1}{2} U^\top H U + x^\top F U + \frac{1}{2} x^\top Y x + \epsilon \hat{B}_{sw} (U, x)
\]

where \( H, F \) and \( Y \) can be computed from \( Q, R, P, A \) and \( B \) and account for the terms \( \|x\|_2^2 \), \( \|u\|_2^2 \) and \( \|x\|_2^2 \) in \( l \) and \( F \). Furthermore, the relaxed logarithmic barrier function \( \hat{B}_{sw} (U, x) \) is shown in [19] to be a weighted sum over convex functions whose argument depends affinely on \((U, x)\). Therefore \( \hat{B}_{sw} (U, x) \) 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]. \( \square \)

In view of Lemma 3.1 and Lemma 3.2, we can show the following convergence theorem, which establishes the desired result.

**Theorem 3.1.** Consider the system (12) controlled by (3)–(8), (11), (13)–(15) and \( \Omega \) from (9). Let \( x \in \Omega \) and assume that \( x \in \text{int conv} \, D_x(k_0) \) for some \( k_0 \in \mathbb{N} \). Then

\[
\lim_{k \to \infty} J_N (\Psi_{sw}(x, D(k)), x) = J_N^r (x).
\]

**Proof.** Step 1) \( J_N^r (\cdot, D(k)) \) converges: For \( k \geq k_0 \), it holds \( \text{conv} \, D_x(k_0) \subseteq \text{conv} \, D_x(k) \) and hence \( J_N^r \) is defined on \( \text{conv} \, D_x(k_0) \) for all \( k \geq k_0 \). Further, since \( \text{conv} \, D_x(k) \supseteq \text{conv} \, D_x(k-1) \) and the minimum over a larger set can only be smaller, we can conclude that

\[
J_N^r (y, D(k)) \leq J_N^r (y, D(k-1))
\]

for all \( y \in \text{conv} \, D_x(k_0), k \geq k_0 \). Due to (i) in Lemma 3.1 and the optimality of \( J_N^r \) we have

\[
J_N (y) \leq J_N (\Psi_{sw}(y, D(k)), y) \leq J_N^r (y, D(k))
\]

for all \( y \in \text{conv} \, D_x(k_0), k \geq k_0 \). Hence, \( J_N^r (\cdot, D(k)) \) is non-increasing and bounded from below on \( \text{conv} \, D_x(k_0) \) and thus converges pointwise

\[
\lim_{k \to \infty} J_N^r (\cdot, D(k))
\]

on \( \text{conv} \, D_x(k_0) \). Due to (ii) in Lemma 3.1, this is a sequence of convex functions \( J_N (\cdot, D(k)) \), which has a convex limit \( J_N^r (\cdot) \). Every convex function is locally Lipschitz continuous on open subsets (see e.g. [7]), hence \( J_N^r (\cdot) \) is locally Lipschitz continuous on \( \text{int conv} \, D_x(k_0) \).

Step 2) \( J_N^r (x, D(k)) \) converges to \( J_N^r (x) \): Let \( x \in \Omega \cap \text{int conv} \, D_x(k_0) \) and let \( C \subset \text{int conv} \, D_x(k_0) \) with \( \text{int} \, C \supseteq x \) be compact, then there exists a sequence \( k_i \to \infty \) such that

\[
\xi_i = f(x(k_i), \Pi_0 U(k_i)) \to x,
\]

\( \xi_i \in C \).

Let \( L_k \) be the Lipschitz constant of \( J_N^r (\cdot, D(k)) \) on \( C \) and \( L_\infty \) for \( J_N^r (\cdot) \), respectively, which are finite since \( C \) is compact and the functions are locally Lipschitz. Hence, \( L_k \to L_\infty \) as \( k \to \infty \) is a real valued converging sequence and is therefore upper bounded by some \( M \in \mathbb{R} \). It follows

\[
|J_N^r (\xi_i, D(k_i + 1)) - J_N^r (\xi_i, D(k_i))| \leq 2M ||\xi_i - x|| + |J_N^r (x, D(k_i + 1)) - J_N^r (x, D(k_i))| \to 0.
\]
Thus the following chain of inequalities

\[
\begin{align*}
J_N^k(\xi_i, D(k_i + 1)) &\leq J_N(U(k_i + 1), \xi_i) \\
&\leq J_N(\Phi^1(U(k_i), x(k_i), D(k_i)), \xi_i) \\
&\leq J_N(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) \\
&\leq J_N(\Psi_{sw}(\xi_i, D(k_i)), \xi_i) \\
&\leq J_N^k(\xi_i, D(k_i))
\end{align*}
\]

(19)

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 3.1 (i) in this specific order. Therefore, as \(i \to \infty\) the decrease of the optimizer update operator (7b)

\[
\gamma(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) \to 0,
\]

which is due to \(\gamma(U, x) = 0 \Leftrightarrow J_N(U, x) = J_N^k(x)\) only possible if

\[
J_N(\Phi^0(U(k_i), x(k_i), D(k_i)), \xi_i) - J_N^k(\xi_i) \to 0.
\]

This is due to (19) equivalent to

\[
J_N^k(\xi_i, D(k_i)) - J_N^k(\xi_i) \to 0.
\]

Since \(J_N^k\) is convex, it is also locally Lipschitz continuous on \(\text{int}\,\text{conv}\, D_N(k_0)\) and without loss of generality we can assume that \(M\) is the Lipschitz constant of \(J_N^k\) on \(C\) (if not choose \(M\) large enough). Hence, it follows

\[
\begin{align*}
|J_N^k(x, D(k_i)) - J_N^k(x)| &\leq |J_N^k(\xi_i, D(k_i)) - J_N^k(\xi_i)| \\
&+ |J_N^k(x, D(k_i)) - J_N^k(\xi_i, D(k_i)) + J_N^k(\xi_i) - J_N^k(\xi_i)| \\
&\leq |J_N^k(\xi_i, D(k_i)) - J_N^k(\xi_i)| + 2M|x - \xi_i| \to 0.
\end{align*}
\]

Thus, we showed that \(J_N^k(x, D(k)) \to J_N^k(x)\) which implies due to (18) the desired result (17). \(\Box\)

**Remark 3.1.** The assumption that there exists a \(k_0 \in \mathbb{N}\) such that \(x \in \text{int}\,\text{conv}\, D_N(k_0)\) is rather technical. Still, it is possible that a recurrent point never lies in the interior of \(\text{conv}\, D_N(k)\) even as \(k \to \infty\). There is a straightforward solution to this problem, which lies in initializing the data set \(D(0)\) with some points \((x_i, U_i, J_N(U_i, x_i)), i = 1, \ldots, K\) beforehand. If \(x \in \text{int}\,\text{conv}\{x_1, \ldots, x_K\}\), this solves the issue even for arbitrary \(U_i\). A good choice for the points \(x_i\), if no previous knowledge about the recurrent points exists, is to take \(x_i\) as the extreme points of the polytopic feasible set, then \(\text{conv}\, D_N(k)\) will always contain the whole feasible set. Another point that should be included in the initialization of \(D(0)\) is \((0, 0, 0)\), because we know that the optimal input at the origin is trivial and has zero costs. This initialization also ensures that on the whole feasible set a spatial warm start can be provided. If nevertheless a spatial warm start for a point outside \(\text{conv}\, D_N(k)\) is required, we could provide a surrogate spatial warm start by taking a point inside \(\text{conv}\, D_N(k)\) that lies close to it or by using some extrapolation method like an affine combination instead of a convex one.

### 3.2. Algorithm and implementation

For online learning, it is crucial to solve (15) or its reformulation (16) in an efficient way, since a warm start solution must be provided within one sampling period. As discussed, \(J_N^k(\cdot, D(k))\) gives rise to a triangulation of \(\text{conv}\, D_N(k)\) 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 \(\text{conv}\, D_N(k)\) must be computed by a convex hull algorithm that determines the facets of the lower boundary of the convex hull \(\text{conv}\, D_N(k)\). The lower boundary of \(\text{conv}\, D_N(k)\) is the graph of \(J_N^k(\cdot, D(k))\), 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 \(\text{conv}\, D_N(k)\). 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. [36]) 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 \(x\). A detailed description of the algorithm that we use for the following examples can be found in the Appendix A.

### 3.3. Academic Example

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

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

(20)

with sampling time \(T_s = 0.1\). Input and state constraint sets \(\mathcal{U}\) and \(\mathcal{X}\) are defined as \(\mathcal{U} = \{u \in \mathbb{R} : |u| \leq 1\}\) and \(\mathcal{X} = \{x \in \mathbb{R}^2 : -2 \leq x_1 \leq 3, |x_2| \leq 1\}\), respectively, and are incorporated into
the cost function with help of relaxed logarithmic barrier functions (see [19], [18]) with parameters $\delta = \varepsilon = 10^{-3}$. Prediction horizon and weight matrices for the quadratic stage costs (13) are chosen as $N = 30$, $Q = \text{diag}(1,0.1)$ and $R = 0.1$. A backtracking line search as described in [19] with the gradient descent search direction and the parameters $\rho = 0.5$, $c_1 = 10^{-3}$ and $c_2 = 0.999$ is used for the optimization. The external disturbance is assumed to be quasiperiodic

$$w(k) = 0.09 \begin{bmatrix} \sin(kT_s) & \cos(kT_s) \end{bmatrix}^T. \tag{21}$$

For two optimization iterations $i_T(k) \equiv i_T = 2$ and a randomly chosen initial condition at $x(0) \approx [0.9 - 0.9]^T$, we have simulated the closed loop behaviour with and without learning from $k = 1$ until $k = N_{\text{sim}} = 3000$. By MPC ‘with learning’ we refer to Algorithm 1 and by MPC ‘without learning’ to the standard anytime MPC [19], i.e. to Algorithm 1 without lines 4, 5, 7-9, 14. The data was initialized with $D(0) = \{(0,0,0)\}$. In the left subplot of Fig. 3, the state trajectory in the phase portrait until $k = 700$ is depicted together with the limit cycle of the optimally controlled system, where the optimization problem (3) is solved with MATLAB’s $\text{fminunc}$. As we can see in Fig. 3, the learning significantly improves the performance since its trajectory is not only much closer to the optimal one, but also has significantly better constraint satisfaction. In the beginning, both trajectories are equal since the spatial warm start is never used because it first needs data to provide a reasonable warm start. By design, the spatial warm start operator performs best inside $\text{conv} D_s(k)$. This can be seen in the simulation where as soon as the trajectory completed its second loop and enters the interior of $\text{conv} D_s(k)$, it deviates from the one without learning and approaches the optimal one.

In the middle subplot, the difference between the costs of both warm starts to the optimal costs for the learning based MPC is depicted over time. As was to be expected, the spatial warm start is in the beginning worse than the temporal one, but then improves over time since more and more data is collected until it is always preferred. We can also see that, although the learning scheme achieves good results, it is limited in speed by the optimization update operator since only two gradient descent iterations per time step results in a rather slow learning progress for this problem.

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. Therefore, we run the simulation for different numbers of optimization iterations $i_T(k)\equiv i_T = 1,\ldots,10$ and for ten different initial conditions. For each $i_T$ and each initial condition, we sum up the cost function values from $k = 1$ until $k = N_{\text{sim}}$ and average these accumulated costs over the different initial conditions. In addition, we also average the computation time per simulation over the different initial conditions and divide it by $N_{\text{sim}}$. Hence, we obtain for each $i_T$ a point in the ‘computation time’ vs. ‘accumulated cost’ plot, depicted on the right of Fig. 3. This is done for the learning based MPC scheme as well as the one that uses only the temporal warm start, such that we can compare curves for the two methods with each other and with the costs resulting from the optimal controller. We can see that for the same computation time, the closed loop with the learning scheme is way closer to the optimal performance than without learning.
4. Leveraging data in real-time nonlinear MPC

In the general case of the nonlinear MPC scheme (2)–(8), (11), we can not 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 leverages Lipschitz continuity.

4.1. A spatial warm start that guarantees convergence

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

$$
(\Psi_{sw}(\bar{x}, D), x^*(\bar{x}, D), J^*(\bar{x}, D)) = \arg\min_{(U, x) \in \mathcal{D}} J + L(U) \|x - \bar{x}\| \quad (22a)
$$

$$
J_N^*(\bar{x}, D) = \min_{(U, x) \in \mathcal{D}} J + L(U) \|x - \bar{x}\| \quad (22b)
$$

where $L(U) \in \mathbb{R}_{++}$ is a Lipschitz parameter that might depend on $U$ and $J_N^*(\cdot, D)$ is an approximation of the value function $J_N$ based on the data $\mathcal{D}$. An illustration of the spatial warm start (22) is given in Fig. 4. If we assume $J_N(U, \cdot)$ to be Lipschitz continuous with constant $L(U)$, then $J_N^*(\cdot, D)$ 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 [39].

**Assumption 4.1.** Assume that $J_N(U, \cdot)$ is Lipschitz continuous with constant $L(U)$, i.e. that for all $x, y \in \mathbb{R}^n$

$$
|J_N(U, x) - J_N(U, y)| \leq L(U) \|x - y\|. \quad (23)
$$

**Lemma 4.1.** Let Assumption 4.1 hold, then

$$
J_N(\Psi_{sw}(x, D), x) \leq J_N^*(x, D) \quad (24)
$$

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

**Proof.** In view of (23) we have

$$
|J_N(\Psi_{sw}(x, D), x) - J_N(\Psi_{sw}(x, D), x^*(x, D))| \
\leq L(\Psi_{sw}(x, D)) \|x^*(x, D) - x\|
$$

and further

$$
J_N(\Psi_{sw}(x, D), x) \leq J_N(\Psi_{sw}(x, D), x^*(x, D)) \
+ L(\Psi_{sw}(x, D)) \|x^*(x, D) - x\| \
= J_N^*(x, D).
$$

To prove the second statement, we use (22a) with $\bar{x} = y \in \mathbb{R}^n$ and (22b) with $\bar{x} = x \in \mathbb{R}^n$ to see

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

which implies

$$
J_N^*(x, D) - J_N^*(y, D) \leq L(\Psi_{sw}(y, D))((\|x - x^*(y, D)\| - \|y - x^*(y, D)\|) \leq L_D \|x - y\|.\quad (25)
$$

Since $x$ and $y$ are interchangeable, it follows

$$
|J_N^*(x, D) - J_N^*(y, D)| \leq L_D \|x - y\|. \quad \Box
$$

In view of Lemma 4.1, we obtain a convergence result similar to Theorem 3.1.

**Theorem 4.1.** Consider the nonlinear MPC scheme presented in (2)–(9), (11) with the spatial warm start operator (22), further let Assumption 4.1 hold, and let $\forall k \geq 1 : L(U(k)) \leq M \in \mathbb{R}$ be upper bounded. Then for all $k \in \Omega$

$$
\lim_{k \to \infty} J_N(\Psi_{sw}(x(k), D(k)), x) = J_N^*(x). \quad (25)
$$

**Proof.** Step 1) $J_N^*(x, D(k))$ converges: In view of Lemma 4.1, we have

$$
J_N^*(x) \leq J_N(\Psi_{sw}(x, D(k)), x) \leq J_N^*(x, D(k)).
$$

Thus by showing $J_N^*(x, D(k)) \to J_N^*(x)$ for $x \in \Omega$ we will prove the theorem. We also see from this inequality that $J_N^*(x, D(k))$ is bounded from below by $J_N^*(x)$. Further $J_N^*(x, D(k))$ is decreasing since the minimum over a larger set $D(k + 1) \supseteq D(k)$ is smaller or equal and thus $J_N^*(x, D(k))$ must converge to some value.

Step 2) $J_N^*(x, D(k))$ converges to $J_N^*(x)$: This step is analogous to step 2 in the proof of Theorem 3.1, but for the sake of completeness we will denote it in full detail. For $x \in \Omega$, there
exists a sequence \( k_i \to \infty \) such that \( \xi_i = f(x(k_i), \Pi_0 U(k_i)) \to x \).
It follows
\[
|J^*_N(\xi_i, D(k_i+1)) - J^*_N(\xi_i, D(k_i))| \leq 2M \|\xi_i - x\|
+ |J^*_N(x, D(k_i+1)) - J^*_N(x, D(k_i))| \to 0.
\]
Thus the following chain of inequalities
\[
J^*_N(\xi_i, D(k_i+1)) \leq J_N(U(k_i+1), \xi_i) \\
\leq J_N(\Phi^U(U(k_i), x(k_i), D(k_i)), \xi_i) \\
\leq J_N(\Phi^U(U(k_i), x(k_i), D(k_i)), \xi_i) \\
\leq J_N(\Psi_{sw}(\xi_i, D(k_i)), \xi_i) \\
\leq J^*_N(\xi_i, D(k_i)),
\]
is a chain of equalities in the limit, where the first inequality holds due to (8) and (22) and the other inequalities due to (7b), (7b), (11) and Lemma 4.1 in this specific order. Therefore as \( i \to \infty \) the decrease of the optimizer update operator (7b)
\[
\gamma(\Phi^D(U(k_i), x(k_i), D(k_i)), \xi_i) \to 0,
\]
which is only possible if
\[
J_N(\Phi^D(U(k_i), x(k_i), D(k_i)), \xi_i) - J^*_N(\xi_i) \to 0.
\]
This is due to (26) equivalent to
\[
J^*_N(\xi_i, D(k_i)) - J_N(U(k_i+1), \xi_i) \to 0.
\]
\( J^*_N \) 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
\[
|J_N^*(x, D(k)) - J^*_N(\xi_i)| \leq |J^*_N(\xi_i, D(k_i)) - J^*_N(\xi_i)| \\
+ |J^*_N(x, D(k_i)) - J^*_N(\xi_i, D(k_i)) + J^*_N(\xi_i, D(k_i)) - J^*_N(\xi, D(k))| \\
\leq |J^*_N(\xi_i, D(k_i)) - J^*_N(\xi_i)| + 2M \|x - \xi\| \to 0
\]
which is with (18) what we wanted. \( \square \)

**Remark 4.1.** Even tough Assumption 4.1 might be restrictive, if we assume that \( x(k) \) does not grow unbound, but stays in some compact region \( C_x \subset \mathbb{R}^n \), then the Lipschitz constants \( L(U(k)) \) do not need to apply globally, but only on \( C_x \) in order to obtain the same result. If further \( U(k) \) stays in some compact set \( C_U \subset \mathbb{R}^{dm} \), then the assumption \( L(U(k)) \leq M \) is also satisfied with \( M = \max_{U \in C_U} L(U) < \infty \) if \( J^*_N \) is continuous in \( U \).

**Remark 4.2.** 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 (22). 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 4.3.** If in a specific set up the cost function \( J_N \) is nevertheless known to be convex, then the learning scheme of Section 3 can also be used and probably leads to a better upper bound \( J^*_N \), see Fig. 4 where the convex hull over the data points would result in a lower upper bound \( J^*_N(\cdot, D) \) inside \( \text{conv } D \). On the other hand, if one faces a linear MPC problem with a nonconvex cost function \( J_N \), due to nonconvex stage or terminal costs, then the learning scheme from this section can still be applied.

### 4.2. Academic Example

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

\[
x = f_c(x, u) = \begin{bmatrix} u_1 \cos(x_3) \\ u_1 \sin(x_3) \\ u_2 \end{bmatrix}
\]

with state and input vectors \( x = [x_1, x_2, x_3], u = [u_1, u_2] \) respectively. The system is discretized using the forward Euler method and a sampling time of \( T_s = 0.1 \). We obtain the following nonlinear discrete-time system

\[
x(k+1) = x(k) + T_s f_c(x(k), u(k)) + w(k),
\]

where we also added the external signal \( w(k) \). The control task is to drive the unicycle to \((x_1, x_2) = (0, 0)\) facing into positive \( x_1 \) direction, i.e. \( x_3 = 2\pi z \) for some \( z \in \mathbb{Z} \) while keeping \( \|u\|_\infty < 1 \).

Therefore the stage cost in (3) is chosen as

\[
l(x, u) = 0.1 \sin \left( \frac{x_3}{2} \right)^2 + \sqrt{1 + x_1^2 + x_2^2} - 1 + u_1^2 + u_2^2.
\]

Note that \( l \) is positive definite with respect to \((x_1, x_2, x_3) = (0, 0, 2\pi z), u = 0 \) and shoots up as \( \|u\|_\infty > 1 \). Further, the 4th root over \( 1 + x_1^2 + x_2^2 \) prevents this term from getting too steep, while the shift of 1 ensures differentiability at the origin. The terminal costs is defined as \( F(x) = 100l(x, 0) \).

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 B.

We choose the temporal warm start as

\[
\psi_{\text{sw}}(U, x) = [U_1^\top, \ldots, U_{N-1}^\top, 0]^\top
\]

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. We simulate the system with a constant number of optimization iterations \( t_f(k) = 2 \). We consider iterative learning in a repetitive and fast process, which is a common task in iterative learning control (see [37]). Whenever 120 time steps are passed, the state is set
Figure 5. Left: Several parts of the closed-loop state trajectory of the unicycle example from Section 4.2 when applying \(i_T(k) = 2\) gradient descent steps and where the system state is restarted from the initial condition every 120 time steps. We call the part of the trajectory in between two restarts a run. The optimal trajectory is computed with the global optimization procedure patternsearch from MATLAB. Right: Accumulated cost per run over the number of the runs. Additionally, the optimal cost is depicted.

back to the initial condition \(x_0 = [1, 1, 1 + \pi/2]^{\top}\) to restart the process, i.e. the external signal can be interpreted as

\[
w(k) = \begin{cases} x_0 - x(k) - T_s f_r(x(k), u(k)) & k + 1 \in 120\mathbb{N} \\ 0 & \text{else.} \end{cases}
\]

Although the temporal warm start (30) might not satisfy (7a), the closed loop performance is still satisfying as we can see in the simulation results depicted in Fig. 5. The closed loop trajectory in the phase portrait in the left subplot approaches the origin in every run, and with increasing number of runs the trajectory converges to the optimal one. The performance is almost optimal after 20 runs. We can also see that in the first run, the unicycle starts driving forward, although the optimal solution starts with driving backwards. Nevertheless, starting in the right direction is learned quite fast in between the first and the fifth run since driving back and forth is expensive near the initial point and far away from the origin. Also the first run approaches the origin from the right driving backwards while the optimal trajectory approaches from left driving forward. This is learned later between 10 and 20 runs. The performance improvement can also be seen in the right subplot where the accumulated costs per run are depicted over the number of the run. The costs improve with every run and approach the optimal costs.

5. Application

We have already seen two examples for the linear and the nonlinear case, which provided insight into the proposed MPC scheme and confirmed the results concerning stability and convergence. However, we have not considered the actual time available for the computation yet. In fact, the learning update may not always be computable within one sampling period \(T_s\), especially when considering systems with higher dimensions. Therefore, we will outsource the learning in the following linear example onto a server that communicates with the controller. In particular, we will implement a linear realtime MPC scheme with the following characteristics. First, the computation of the convex hull update is executed in parallel to the spatial warm start generation and the optimization. Second, whenever the convex hull update is completed, the latest data point is added next, even tough there might be some points skipped in between the last added point and this one. Third, instead of using a predefined constant number of optimization iterations \(i_T(k)\) as before, we use in this example the more realistic approach of doing as many iterations as possible until 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 2.1 and Theorem 3.1 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 model

\[
\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -\frac{k_g}{J_c} & -\frac{\beta_L}{J_c} & \frac{k_g}{J_c} & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_g}{pJ_M} & 0 & -\frac{k_g}{p^2J_M} & -\frac{\beta_M + K_e^2/R}{J_M} \end{bmatrix} x + \begin{bmatrix} 0 \\ 0 \\ 0 \\ \frac{K_e}{J_M} \end{bmatrix} u \tag{31}
\]

and the parameters, which can be found in Table 1. The state vector \(x = [\theta_L \dot{\theta}_L \theta_M \dot{\theta}_M]^{\top}\) consists of the load angle \(\theta_L\) and the motor angle \(\theta_M\) as well as their time derivatives. The input \(u\) corresponds to the DC voltage. The reference tracking task of this system is included in the collection of benchmark MPC problems given in [24]. The model is discretized using zero-order hold on the input and the sampling time \(T_s\) given in Table 1. Further the system has to satisfy the state and input
In addition, we added the constraints

$$\left[ \begin{array}{cc} k_\theta & 0 \\ \frac{k_\theta}{\rho} & 0 \end{array} \right] x \leq T_{max},$$

(32a)

$$|u| \leq V_{max}. \quad (32b)$$

In addition, we added the constraints

$$|x_1| \leq 2, \quad |x_2| \leq 2, \quad |x_2 + x_4| \leq 40 \quad (33)$$

to make the problem more difficult and to obtain a polytopic feasible set.

We further need to specify the reference tracking task and how to handle it. The goal is that the angle of the load $\theta_L$ follows the periodic step reference signal

$$r(k) = \begin{cases} 1 & \text{if } \exists z \in \mathbb{Z} : 1 \leq k - 200z \leq 100, \\ 0 & \text{otherwise}. \end{cases} \quad (34)$$

To achieve this we see that

$$x_s(r_k) := [1 \ 0 \ \rho \ 0]^T r_k, \quad u_s(r_k) := 0 \quad (35)$$

is a steady state of (31) that produces exactly the constant reference $r_k$ for arbitrary $r_s \in \mathbb{R}$. Therefore we will define the cost function with respect to $x - x_s(r(k))$ and $u - u_s(r(k))$ except for the part that incorporates the constraints. That is, (13) and (14) become

$$l(x, r, u) = ||x - x_s(r)||_Q^2 + ||u - u_s(r)||_R^2 + \epsilon \hat{B}_x(s) + \epsilon \hat{B}_u(u), \quad (36)$$

$$F(x, r) = ||x - x_s(r)||_p^2, \quad (37)$$

and we choose

$$Q = \text{diag}(\{10 \ 0.1 \ 10 \ 0.1\}), \quad R = 0.01 \quad (38)$$

and the barrier function parameters as $\epsilon = \delta = 10^{-3}$ (see [18] for details on relaxed logarithmic barrier functions). 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 $D(0)$ will be initialized with the extreme points of the feasible set as described in Remark 3.1. As input sequences for these points, we let the optimization iteration run for one sampling period at each point starting from $U = 0$.

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 mean reference tracking error of the optimally controlled system. The standard anytime MPC scheme without a spatial warm start and thus without learning does not improve over time and the mean reference tracking error stays as high as in the beginning. In this scheme without learning, there is time for an average of $I_1(k) \approx 39.4$ iterations while in the scheme with learning $I_1(k) \approx 37.6$ optimization iterations take place. So the computation time invested into the spatial warm start generation costs on average 1.8 optimization iterations. Therefore, the scheme without learning results in a lower mean tracking error for the very first period. However, already in the second period, the scheme with learning achieves better tracking performance.
Throughout the whole simulation, the constraints were always satisfied. As we can see in the right subplot of Fig. 6, the input constraint (32b) is active for the optimal input in the first few time instants after each step occurs. The suboptimal inputs do not exploit the whole range of feasible inputs for the first 120 periods, but afterwards the controller learned it.

There is a computational aspect that we have not discussed yet. On the one hand, the more data is collected, the better the warm start gets. On the other hand, the required memory capacity and the computation time for updating the convex hull and for the spatial warm start generation increases with the size of $D(k)$. Hence, when we keep adding data, we will eventually hit the point where no spatial warm start solution could be provided within one sampling period. This would mean that the temporal warm start solution must be used as input without any further optimization since no more computation time is left. We were able to prevent this, by only adding data points $(U, x, J)$ to $D(k)$ if the improvement that will be made from $J_N^k(x, D(k))$ to $J = J_N^k(x, D(k + 1))$ is larger than some constant, which we chose as $10^{-2}$. In this way, we will only add points that lead to noticeable improvements and skip data points in regions where we already have good knowledge of the value function. In this example, the points generated between five and ten as well as between 15 and 20 seconds are almost similar. Instead of adding lots of these data points that do not lead to any further improvement but increase the complexity of the convex hull, it is better to discard these points and save memory and computation time for more interesting points. Overall, for this example, we added 4206 points to the convex hull object after 30000 time steps, while we skipped 21164 because the improvement was too small. Another 4630 points were omitted because it was not possible to update the convex hull within one sampling time.

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 example. 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 by including some mechanisms to forget expired data points. Since the proposed online learning scheme guarantees stability independently of the learning method and memory capacity, it would be interesting to compare classical machine learning function approximators (e.g. neural networks or Gaussian processes) to the methods developed in this work that exploit the special properties of the MPC optimization problem.

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. Alamir. From certification of algorithms to certified MPC: The missing links. In Proceedings of the 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 International Conference on Robotics and Automation, pages 279–284, May 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 Transactions on Automatic Control*, 62(1):135–148, Jan 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 *Annual American Control Conference (ACC)*, pages 1520–1527, June 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, May 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 Conference on Decision and Control and European Control Conference*, pages 513–519, Dec 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 set formulations for linear model predictive control. In *2015 American Control Conference (ACC)*, pages 1647–1652, July 2015.

[18] C. Feller and C. Ebenbauer. Relaxed logarithmic barrier function based model predictive control of linear systems. *IEEE Transactions on Automatic Control*, 62(3):1223–1238, March 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, July 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] J. Kocijan, R. Murray-Smith, C. E. Rasmussen, and A. Girard. Gaussian process model based predictive control. In *Proceedings of the 2004 American Control Conference*, volume 3, pages 2214–2219, June 2004.

[24] 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, July 2015.

[25] I. Lenz, R. Knepper, and A. Saxena. DeepMPC: Learning deep latent features for model predictive control. In *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.

[26] D. Limon, J. Calliess, and J. Maciejowski. Learning-based nonlinear model predictive control. *IFAC-PapersOnLine*, 50(1):7769–7776, 2017. 20th IFAC World Congress.

[27] L. K. McGovern and E. Feron. Closed-loop stability of systems driven by real-time, dynamic optimization algorithms. In *Proceedings of the 38th IEEE Conference on Decision and Control*, volume 4, pages 3690–3696, Dec 1999.

[28] C. J. Ostařew, 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, May 2014.

[29] T. Parisini and R. Zoppoli. A receding-horizon regulator for nonlinear systems and a neural approximation. *Automatica*, 31(10):1443–1451, 1995.

[30] 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.

[31] 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.

[32] J. B. Rawlings, D. Q. Mayne, and M. M. Diehl. *Model Predictive Control: Theory and Design*. Nob Hill Publishing, 2017.
A. Convex hull algorithm

The purpose of the convex hull algorithm presented in this section is to solve (15) as discussed in Section 3.2. 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 \( CH \). The properties of the convex hull object are listed in Table 2. References to a property of the convex hull object are denoted with a dot, e.g.

| name | description |
|------|-------------|
| \( D_x \) | list of previous data points in \( \mathbb{R}^n \) |
| \( D_u \) | list of previous input sequences in \( \mathbb{R}^{N_u} \) |
| \( D_y \) | list of previous cost function values in \( \mathbb{R}_+ \) |
| \( CH_{\delta_x} \) | list that contains all facets of the lower half of conv \( D_x \) as lists of the indices of their extreme points in \( D_{\delta_x} \) |
| \( CH_x \) | list that contains all facets of conv \( D_x \) as lists of the indices of their extreme points in \( D_x \) |
| \( c_x \) | center point of conv \( D_x \) |
| \( c_{\delta_x} \) | center point of conv \( D_{\delta_x} \) |
| \( o_x \) | list of deleted facets in \( CH_x \) that can be overwritten |
| \( o_{\delta_x} \) | list of deleted facets in \( CH_{\delta_x} \) that can be overwritten |
| \( G_x \) | list that contains for each point in \( D_x \) a list of facets in \( CH_x \) that are attached to this point |
| \( G_{\delta_x} \) | list that contains for each point in \( D_{\delta_x} \) a list of facets in \( CH_{\delta_x} \) that are attached to this point |

\( CH\cdot D_x \). In addition to these listed properties we will use \( CH\cdot D \) and \( CH\cdot D_{\delta} \) to denote the combined lists of the data points, so the \( k \)th element of \( CH\cdot D \) is for example a triple containing the \( k \)th elements of \( CH\cdot D_u \), \( CH\cdot D_x \) and \( CH\cdot D_y \).

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 conv \( D_x(k) \) 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

\[ D_x \]
graph \( CH, G_{\epsilon} \), which needs to be updated whenever the convex hull is.

**A.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 \( CH \) 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 \( a \in CH.D_{\epsilon} \) that is close to \( x \). Assume the index \( i \) of \( a \) in \( CH.D_{\epsilon} \) is given, then the routine takes the inputs \( CH, x \) and \( i \). As outputs it makes sense to not only return the spatial warm start \( \Psi_{sw} = \Psi_{sw}(x, CH.D) \) but also the index \( F \) of the facet of \( CH.CH_{\epsilon} \) that contains \( x \) and the index \( i \) of the next initial guess. If \( x \) is not contained in any facet, i.e. if \( x \notin conv CH.D_{\epsilon} \), then let \( F \) be negative, in fact let it be \( F = -F_{k} \) where \( F_{k} \) is the index of a facet of \( CH.CH_{\epsilon} \) that excludes \( x \) by means of \( x \) lies outside the supporting halfspace to \( conv D_{\epsilon} \) at this facet \( F_{k} \). Summing, the routine can be invoked by \((\Psi_{sw}, F, i) = \text{generateSW}(CH, x, i)\). 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 \( conv CH.D_{\epsilon} \), then \( x \notin conv CH.D_{\epsilon} \) and we can not provide a spatial warm start as described in (15) at \( a \), but we can at the point where we stepped out of \( conv CH.D_{\epsilon} \) 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 \( CH.CH_{\epsilon} \) 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 \( conv CH.D_{\epsilon} \), \( a \) must lie on the boundary and hence \( x \notin conv CH.D_{\epsilon} \). So there must exist at least one facet of \( CH.CH_{\epsilon} \) 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 \( conv CH.D_{\epsilon} \).

e) (35:–41:) If we stepped out before \( s = 1 \), then find the index of facet of \( CH.CH_{\epsilon} \) 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 \( \Psi_{sw} \).

f) (42:–46:) Else we have found index \( F \) of the facet of \( CH.CH_{\epsilon} \) 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 \( i \).

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.

- \( b = \text{pointsInFacet}(CH, x, F, i) \) checks if the vector \( v = x - CH.D_{\epsilon}(i) \) starting from \( i \) points inside the facet of \( CH.CH_{\epsilon} \) 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. \( x = CH.D_{\epsilon}(i) + v \) need not lie inside \( F \) for \( b \) being true, it is enough if there exists \( \epsilon > 0 \) such that \( CH.D_{\epsilon}(i) + \epsilon v \) 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 \( M \in \mathbb{R}^{p \times p} \) into an orthogonal matrix \( Q = Q^{-1} \) and an upper triangular matrix \( R \) where all diagonal entries of \( R \) are positive.

- \( f = \text{getFacets}(CH, E, \text{flag}) \) gives out all facets of \( CH.CH_{\epsilon} \) 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.

- \( f = \text{findIntersection}(CH, a, v, s, F, E) \) 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 \( s \) > 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.

- \( f = \text{incConvComb}(CH, f, x) \) calculates the weights \( c \) such that \( \sum_{j=1}^{f} c(j)CH.D_{\epsilon}(f(j)) = x \) 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.

**A.2. Updating the convex hull**

Second we consider updating the convex hull, therefore let \( CH \) 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 \( CH.CH_{\epsilon} \) that contains \( x \) from
Algorithm 2 Spatial warm start generation

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

**Input:** convex hull object \(CH\), evaluation point \(x\) and initial guess \(i\)

**Output:** spatial warm start \(\Psi_{sw}\) at \(x\), index \(F\) of facet \(CH, CH_\Delta\) that contains \(x\) and next initial guess \(i\). If \(x \notin \text{conv}(CH, D_x)\), then \(F\) is negative and \(|F|\) is the index of a facet in \(CH, CH_x\) that excludes \(x\).

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

Algorithm 3 Check if vector points in facet

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

**Input:** convex hull object \(CH\), vector \(x\), facet \(F\) and index \(i\) of starting point

**Output:** boolean \(b\), true if \(v = x - CH.D_x(i)\) points in \(F\) starting from \(i\).

1. \(E = CH.CH_\Delta(F)\)
2. for \(j = 1, \ldots, n + 1\)
3. if \(E(j) \neq i\) then
4. \(\bar{E} = E\) without \(E(j)\)
5. \(d = \text{normal}(CH.D_\Psi(\bar{E}), CH.D_x(E(j)))\)
6. if \(d^\top (x - CH.D_x(i)) > 0\) 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 \(j = 1, \ldots, n - 1\)
2. \(A(j) = A(j) - A(n)\)
3. end for
4. \(A(n) = x - A(n)\)
5. QR-decomposition \(QR = [A(1), A(2), \ldots, A(n)]\)
6. return \(d = -\) last column of \(Q\)

Algorithm 5 Get facets attached to set of extreme points

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

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

**Output:** list \(f\) of facets of \(CH, CH_\text{flag}\) that contain all extreme points given in \(E\)

1. \(f = CH.G_\text{flag}(E(1))\)
2. for \(p \in E\) without \(E(1)\) do
3. \(f = \text{common elements of} \ f \text{ and} \ CH.G_\text{flag}(p)\)
4. end for
5. return \(f\)
Algorithm 6: Find intersection of line and facet boundary

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

1. \( p = \text{CH.CH}_3(F) \) without \( E \)
2. for \( j = 1, 2, \ldots, n \)
3. \( \tilde{E}(j) = E \) without \( E(j) \) but with \( p \)
4. \( d = \text{normal}(\text{CH.D}_x(E), \text{CH.D}_x(E(j))) \)
5. \( \tilde{s}(j) = d^\top(\text{CH.D}_x(p) - a)/(d^\top v) \)
6. if \( \tilde{s}(j) \leq s \) then \( \tilde{s}(j) = \infty \)
7. end for
8. \( \tilde{j} = \arg\min \tilde{s}(j) \)
9. return \((s, E) = (\tilde{s}(j), \tilde{E}(j))\)

Algorithm 7: Find convex combination

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

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

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 \( \text{CH} \), hence it can be invoked by \( \text{CH} = \text{updateCH}(\text{CH}, x, U, F) \). The basic idea of updating \( \text{CH} \) is that we start at the facet \( F \), which must be for 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 the 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 \( x \notin \text{convCH.D}_x \), i.e. \( F < 0 \), then we need to find all facets of \( \text{CH.D}_x \) that exclude \( x \) and check that facets of \( \text{CH.D}_x \) attached to them if they must be updated. In this case we must also update \( \text{CH.D}_x \), which we do by the same procedure, starting from \(-F\) search for neighboring facets that must to be updated. In fact all facets of \( \text{CH.D}_x \) that must to be updated are facets that exclude \( x \) and the other way round, so we search them, update them and initialize with the facets of \( \text{CH.D}_x \) attached to them the search for updates in \( \text{CH.D}_x \). 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 \( \text{CH.D} \).

b) (6:--24:) If \( F < 0 \) then \( x \notin \text{convCH.D}_x \) and we need to update \( \text{CH.D}_x \) 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 \( \text{CH.D}_x \). Whenever we remove a facet we also add it to the list of edges we need to check for updating \( \text{CH.D}_x \).

c) (25:--28) If \( F > 0 \) then we remove this facet from \( \text{CH.D}_x \) 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 \( \text{CH.D}_x \).

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 \( \text{CH.D}_x \) we of course also have to update the graph \( \text{CH.G}_{x/x} \). 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 \( \text{CH.D}_x \) and also removes its appearance in \( \text{CH.G}_{x/x} \). When removing \( F \) we still need to keep the slot \( F \) in \( \text{CH.D}_x \) because if we would delete it, all facets coming after \( F \) in the list \( \text{CH.D}_x \) would get their index decremented and we would need to change the whole \( \text{CH.G}_{x/x} \). Hence it is easier to add \( F \) to a list \( \text{CH.D}_x \) of open slots that can be overwritten, thereby keeping all other indices as they are. If it happens for \( \text{flag} = xJ \) that there is 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 \( \text{CH.G}_{x/x}(p) = -k \). 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 can not 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.
Algorithm 8 Update convex hull object
\[ \text{\textbf{CH} = updateCH(CH,x,U,F)} \]
\textbf{Input:} convex hull object \( \text{CH} \), new point \( x \), input sequence \( U \), index \( F \) of facet in \( \text{CH} \) that contains \( x \), negative if \( x \notin \text{convCH}_x \), then \( \vert F \vert \) is facet of \( \text{CH} \) that excludes \( x \).
\textbf{Output:} updated convex hull object \( \text{CH} \) that contains the new point

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

- \( \text{CH} = \text{addFacet}(\text{CH},f,\text{flag}) \) adds the facet with extreme points listed in \( f \) to \( \text{CH} \) and updates the graph \( \text{CH} \). If there is an open slot \( \text{CH}_o \) in \( \text{CH} \), then we can overwrite it, otherwise we append the new facet to the list \( \text{CH} \).

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 straightforward approach is to wait until the \( n+1 \) data points are available and initialize the convex hull \( \text{CH}_x \) with the first and only facet [1 \ldots n] as well as the convex hull \( \text{CH}_x \) with all \( n+1 \) possible combinations of \( n \) points out of these \( n+1 \) points.

Algorithm 9 Remove facet from convex hull object
\[ \text{\textbf{CH} = removeFacet(CH,F,k,flag)} \]
\textbf{Input:} convex hull object \( \text{CH} \), facet \( F \) of \( \text{CH}_o \), current index \( k \) and flag that is either \( x \) or \( x \) indicating the convex hull
\textbf{Output:} updated convex hull object \( \text{CH} \) without \( F \)

1: for \( p \in \text{CH}_o(\text{flag}(p)) \) do
2: remove \( F \) from \( \text{CH}_o \)
3: if \( \text{CH}_o(\text{flag}(p)) \) is empty and flag = \( x \) then
4: \( \text{CH}_o(\text{flag}(p)) = -k \)
5: end if
6: end for
7: add \( F \) to \( \text{CH}_o \)
8: return \( \text{CH} \)

Algorithm 10 Add facet to convex hull object
\[ \text{\textbf{CH} = addFacet(CH,f,flag)} \]
\textbf{Input:} convex hull object \( \text{CH} \), list of extreme points of facet that is added to \( \text{CH} \), current index \( k \) and flag that is either \( x \) or \( x \) indicating the convex hull
\textbf{Output:} updated convex hull object \( \text{CH} \) with facet \( F \)

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

$$L(U) = \sum_{i=0}^{N} (L_i + \delta_{Ni}(L_F - L_i)) \prod_{j=0}^{i-1} L_j(\Pi_j U)$$  \hspace{1cm} (39)

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

$$L_i = \sqrt{\frac{13}{50}} \quad L_F = 100L_i$$  \hspace{1cm} (40a)

$$L_j(u) = \sqrt{1 + T_s |u_1| \sqrt{1 + T_s^2 u_1^2 / 4 + T_s^2 u_1^2 / 2}}. \quad (40b)$$

by the following calculations

$$\left( \frac{\partial l}{\partial x_3} (x,u) \right)^2 = \left( 0.1 \sin \left( \frac{x_3}{2} \right) \cos \left( \frac{x_3}{2} \right) \right)^2 \leq 0.1^2$$

$$\left( \frac{\partial l}{\partial x_{1,2}} (x,u) \right)^2 = \left( \frac{2x_{1,2}}{4 (1+x_1^2+x_2^2)^{3/2}} \right)^2$$

$$= \frac{1}{4} \sqrt{\frac{x_{1,2}^4}{(1+x_{1,2}^2)^3}}$$

$$\leq \frac{1}{4} \sqrt{\frac{x_{1,2}^4}{1+3x_{1,2}^2+3x_{1,2}^4+x_{1,2}^6}} \leq \frac{1}{8}$$

$$\left\| \frac{\partial l}{\partial x} (x,u) \right\| \leq \sqrt{0.1^2 + \frac{1}{4}} = \sqrt{\frac{13}{50}}$$

and

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

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

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

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

20