Methods

Maximilian Pierer von Esch*, Andreas Völz and Knut Graichen

Sensitivity-based distributed model predictive control: synchronous and asynchronous execution compared to ADMM

Sensitivitätsbasierte verteilte modellprädiktive Regelung: Synchrone und Asynchrone Ausführung im Vergleich zu ADMM

https://doi.org/10.1515/auto-2023-0050
Received March 31, 2023; accepted November 30, 2023

Abstract: This paper presents synchronous and asynchronous formulations of a sensitivity-based algorithm for solving distributed continuous-time nonlinear optimal control problems with a neighbor-affine structure. Sensitivities are defined in an optimal control context and utilized to ensure coordination between agents. By using delayed data in the asynchronous formulation, idle times of agents are reduced and faster execution of the algorithm is achieved. The convergence behavior w.r.t. topology and coupling strength is investigated in a numerical simulation and the execution time is evaluated on distributed hardware with communication over TCP in comparison to the alternating direction method of multipliers (ADMM) algorithm.

Keywords: multi-agent systems; distributed model predictive control; distributed optimization; sensitivities; asynchronous optimization

Zusammenfassung: In diesem Beitrag werden synchrone und asynchrone Formulierungen eines sensitivitätsbasierten Algorithmus zur Lösung verteilter zeitkontinuierlicher nichtlinearer Optimalsteuerungsprobleme mit nachbaraffiner Struktur vorgestellt. Sensitivitäten werden für verteilte Optimalsteuerungsprobleme definiert und genutzt, um die Koordination zwischen den Agenten sicherzustellen. Durch die Verwendung verzögerter Daten in einer asynchronen Formulierung werden die Wartezeiten der Agenten reduziert und eine schnellere Ausführung des Algorithmus erreicht. Das Konvergenzverhalten in Abhängigkeit von Topologie und Kopplungsstärke wird in einer numerischen Simulation untersucht und die Ausführungszeit auf verteilter Hardware mit Kommunikation über TCP im Vergleich zum Alternating Direction Method of Multipliers (ADMM)-Algorithmus bewertet.

Schlagwörter: multiagenten Systeme; verteilte modellprädiktive Regelung; verteilte Optimierung; Sensitivitäten; asynchrone Optimierung

1 Introduction

In model predictive control (MPC) [1, 2], the system input is obtained by solving an optimal control problem (OCP) in each sampling step and applying the input to the system until the next measurement is available. MPC is a widely used modern control technique due to its ability to handle constrained nonlinear systems in real-time within many application areas [3, 4].

MPC is usually performed in a centralized fashion meaning that the complete system is considered and all the inputs are calculated within a single OCP. For large-scale, distributed systems, this approach can become difficult to realize due to a high number of states and controls in the central OCP which hinders the repeated real-time solution of the optimization problem. The idea of distributed model predictive control (DMPC) [5, 6] is to split the central OCP into local OCPs of reduced order which can be solved locally and in parallel. This is especially useful if the central OCP is sparse and the local problems only contain a fraction of the overall variables.

A popular approach to DMPC is to solve the OCPs iteratively in the MPC loop via distributed optimization algorithms such as the alternating direction method of multipliers (ADMM) [7–9], dual decomposition [10] or ALADIN

***Corresponding author: Maximilian Pierer von Esch**, Chair of Automatic Control, Friedrich-Alexander-Universität Erlangen-Nürnberg, 91058 Erlangen, Germany, E-mail: maximilian.v.pierer@fau.de
Andreas Völz and Knut Graichen, Chair of Automatic Control, Friedrich-Alexander-Universität Erlangen-Nürnberg, 91058 Erlangen, Germany, E-mail: andreas.voelz@fau.de (A. Völz), knut.graichen@fau.de (K. Graichen)

[11]. An extensive compendium of DMPC algorithms can be found in [12]. In this paper, a sensitivity-based DMPC algorithm is investigated which is based on the works of [13–15] and uses so-called interaction operators or sensitivities [16] to achieve coordination among agents. Sensitivity-based DMPC algorithms have been utilized mostly in the context of linear systems for process control [17], building automation [18] or chemical reactors [15] but only sporadically for nonlinear systems [14]. Furthermore, sensitivities have been used in distributed optimization together with the ADMM algorithm to reduce the computational burden of the individual sub-problems [19].

The aforementioned algorithms have in common that some kind of information exchange is necessary to ensure that the local solutions converge to the global one. Communication is a vital aspect of DMPC as previous studies have reported that communication time can take around 80 % of the overall computation time [9, 20]. In combination with heterogeneous solution times of the local OCPs, this can lead to large waiting times in the network.

Asynchronous formulations of distributed optimization algorithms mitigate this effect by allowing the use of outdated information in the solution of the local OCPs. This reduces waiting times and lowers the overall execution time but affects convergence properties negatively. Various methods have been proposed and analyzed to cope with delayed information exchange in the context of DMPC including asynchronous ADMM [21, 22] as well as Newton methods [23] or event triggered control schemes [24].

In this paper, the algorithm developed in [13] is extended to constrained neighbor-affine nonlinear systems and formulated in an asynchronous fashion to reduce network waiting times. The needed sensitivities can be efficiently computed via nonlinear optimal control theory. In addition, it is demonstrated that if the algorithm is convergent, then the optimality conditions of the central and distributed problem correspond. Finally, different key figures such as convergence w.r.t. network topology, coupling strength, or execution times on distributed hardware are evaluated and compared to the (asynchronous) ADMM algorithm.

The paper is structured as follows. Firstly, the problem description is stated in Section 2, before the sensitivities are derived for neighbor-affine problems in an optimal control context and synchronous as well as asynchronous versions of the sensitivity-based and ADMM algorithm are presented in Section 3. The modular implementation as well as a comparison via simulation of all algorithms is presented in Section 4 before the paper is summarized in Section 5.

Throughout this text, several conventions are used. The stacking and reordering of individual vectors $\boldsymbol{x}_i \in \mathbb{R}^n$, $i \in \mathcal{V}$ from a set $\mathcal{V}$ is denoted as $\boldsymbol{x}_\mathcal{V}$. The standard p-norms of vectors $\|\boldsymbol{x}\|_p = \left(\sum_i^n |x_i|^p\right)^{\frac{1}{p}}$ as well as the weighted squared norm $\|\boldsymbol{x}\|_{\boldsymbol{P}} = \sqrt{\boldsymbol{x}^\mathrm{T}\boldsymbol{P}\boldsymbol{x}}$ for some positive definite weighting matrix $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ are utilized. Furthermore, the superscript k denotes an iteration step of an algorithm. The partial derivatives of a function $\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ are denoted as $\partial_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})$ and the abbreviation for the evaluation at step k: $\partial_{\boldsymbol{x}}\boldsymbol{f}^{[k]} = \partial_{\boldsymbol{x}}\boldsymbol{f}(\boldsymbol{x},\boldsymbol{y})|_{\boldsymbol{x}^{[k]},\boldsymbol{y}^{[k]}}$ is used. For a functional $F(\boldsymbol{x},\boldsymbol{y})\colon \mathcal{X}\times\mathcal{X}\to\mathbb{R}$ with the Hilbert space $\mathcal{X}$, the first-order Gâteaux derivatives in directions $\delta\boldsymbol{x}$ and $\delta\boldsymbol{y}$ are denoted as $\delta_{\boldsymbol{x}}F(\boldsymbol{x},\boldsymbol{y})(\delta\boldsymbol{x})$ and $\delta_{\boldsymbol{y}}F(\boldsymbol{x},\boldsymbol{y})(\delta\boldsymbol{y})$, respectively. The explicit time dependency of trajectories, e.g. $\boldsymbol{x}(t)$, as well as multivariable dependencies are typically omitted for the sake of readability.

2 Problem formulation

In the context of DMPC, multi-agent systems are usually described by a directed graph $\mathcal{G} = (\mathcal{V},\mathcal{E})$ with the vertices $\mathcal{V}$ and edges $\mathcal{E}$. Each vertex, referred to as an agent, represents a subsystem. An edge stands for a directed coupling between the corresponding agents. The set $\mathcal{N}_i$ defines the neighborhood of agent $i \in \mathcal{V}$ and comprises all agents $j \in \mathcal{V}$ directly coupled with agent $i \in \mathcal{V}$. This set can be further refined into the sending neighbors $\mathcal{N}_i^{\leftarrow}$ that influence agent $i \in \mathcal{V}$ and receiving neighbors $\mathcal{N}_i^{\rightarrow}$ that are influenced by agent $i \in \mathcal{V}$. Note that a neighbor j can be an element of $\mathcal{N}_i^{\rightarrow}$ as well as $\mathcal{N}_i^{\leftarrow}$. Using these notations, the central OCP is given by

$$\min_{\boldsymbol{u}_i, i\in\mathcal{V}} \quad J(\boldsymbol{u}) = \sum_{i\in\mathcal{V}} J_i(\boldsymbol{u}_i, \bar{\boldsymbol{v}}_i) \tag{1a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) + \sum_{j\in\mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{x}_j,\boldsymbol{u}_j), \tag{1b}$$

$$\boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \tag{1c}$$

with the states $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$ and controls $\boldsymbol{u}_i \in \mathbb{R}^{m_i}$ of agent i as well as the states $\boldsymbol{x}_j \in \mathbb{R}^{n_j}$ and controls $\boldsymbol{u}_j \in \mathbb{R}^{m_j}$ of neighbor $j \in \mathcal{N}_i^{\leftarrow}$. The local cost functional is defined as

$$J_i(\boldsymbol{u}_i,\bar{\boldsymbol{v}}_i) := V_i(\boldsymbol{x}_i(T)) + \sum_{j\in\mathcal{N}_i^{\leftarrow}} V_{ij}(\boldsymbol{x}_i(T),\boldsymbol{x}_j(T)) + \int_0^T l_i(\boldsymbol{x}_i,\boldsymbol{u}_i) + \sum_{j\in\mathcal{N}_i^{\leftarrow}} l_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{x}_j,\boldsymbol{u}_j)\,\mathrm{d}t. \tag{2}$$

The notation $\bar{\boldsymbol{v}}_i := (\boldsymbol{x}_j, \boldsymbol{u}_j / j \in \mathcal{N}_i^{\leftarrow})$, $i \in \mathcal{V}$ refers to the stacked vectors of states $\boldsymbol{x}_j$ and controls $\boldsymbol{u}_j$ of all

sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$ while $\boldsymbol{v}_i := (\boldsymbol{x}_i, \boldsymbol{u}_i)$, $i \in \mathcal{V}$ describes the local variables. The extension on how to incorporate input/state constraints in the problem description is given in the Appendix.

Furthermore, the dynamics function (1b) as well as the cost function (2) of an agent are given in the so-called neighbor-affine form, meaning that the costs and dynamics consist of the local costs l_i and dynamics $\boldsymbol{f}_i$ which only depend on the states and controls of the agent, and additive (nonlinear) coupled costs l_{ij} and dynamics $\boldsymbol{f}_{ij}$ that depend on the states and controls of the agent and one neighbor j. The local terminal cost is given by V_i and the additive coupled terminal cost by V_{ij}. This allows for a modular description of the multi-agent system while covering most of the relevant system classes and problem descriptions, for example setpoint and trajectory tracking of dynamically coupled systems or problems with coupled cost functions such as synchronization problems, formation control or leader-follower scenarios.

Moreover, it is assumed that the system dynamics (1b) and cost terms in (2) are sufficiently continuously differentiable w.r.t. their arguments on $t \in [0, T]$ and that an optimal solution to the OCP (1) exists. It is assumed that the system dynamics (1b) have a unique and bounded solution $\boldsymbol{x}_i(t)$, $i \in \mathcal{V}$ for each continuous control input $\boldsymbol{u}_i(t)$ uniquely defined for almost all t, $t \in [0, T]$, $i \in \mathcal{V}$ such that the cost (1a) can be regarded as the functional $J(\boldsymbol{u})$.

3 Distributed model predictive control

For large-scale systems, the repeated real-time solution of OCP (1) leads to a significant computational effort for standard MPC algorithms due to the high number of states and controls in (1). DMPC algorithms provide a remedy by distributing the computational effort to the agents while solving reduced sub-problems with fewer variables which are computationally less expensive.

In order for these sub-problems to converge to the central solution of OCP (1), coordination between the agents is necessary, usually by exchanging information regarding their respective solutions. Therefore, DMPC algorithms typically consist of local computation and communication steps. This higher algorithmic complexity compared to MPC algorithms is justified by a parallel operation which is expected to outperform a central MPC approach if the number of agents is high.

3.1 Sensitivity-based DMPC formulation

In a distributed setting, agent i has no direct access to the variables $\bar{\boldsymbol{v}}_i$ of its sending neighbors but can only rely on states and controls $\bar{\boldsymbol{v}}_i^{[k]}$ which were communicated by the sending neighbors at some iteration k of the distributed algorithm. Then, agent $i \in \mathcal{V}$ could solve the parameterized OCP

$$\min_{\boldsymbol{u}_i} \quad J_i(\boldsymbol{u}_i, \bar{\boldsymbol{v}}_i^{[k]}) \tag{3a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j^{[k]}, \boldsymbol{u}_j^{[k]}) \tag{3b}$$

$$\boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0}. \tag{3c}$$

However, in this way agent i does not take into account that its own variables $\boldsymbol{x}_i, \boldsymbol{u}_i$ also influence the local problems of its neighbors.

The solution and main idea of sensitivity-based DMPC is to augment the cost functional (3a) by a functional Taylor expansion of the neighbor's cost functional at $\boldsymbol{v}_j^{[k]}$ up to the first-order term

$$J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]} + \delta\bar{\boldsymbol{v}}_j) \approx J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]}) + \delta_{\bar{\boldsymbol{v}}_j} J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]})(\delta\bar{\boldsymbol{v}}_j) \tag{4}$$

with the Gâteaux derivative $\delta_{\bar{\boldsymbol{v}}_j} J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]})(\delta\bar{\boldsymbol{v}}_j)$ in direction $\delta\bar{\boldsymbol{v}}_j := \bar{\boldsymbol{v}}_j - \bar{\boldsymbol{v}}_j^{[k]}$. Using this approximation, the local control problem (3) is modified to

$$\min_{\boldsymbol{u}_i} \quad J_i(\boldsymbol{u}_i, \bar{\boldsymbol{v}}_i^{[k]}) + \sum_{j \in \mathcal{N}_i^{\rightarrow}} \bar{J}_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]} + \delta\bar{\boldsymbol{v}}_j) \tag{5a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j^{[k]}, \boldsymbol{u}_j^{[k]}) \tag{5b}$$

$$\boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0}. \tag{5c}$$

Note that the augmented cost functional is given by

$$\bar{J}_j(\boldsymbol{u}_j, \bar{\boldsymbol{v}}_j^{[k]}) := J_j(\boldsymbol{u}_j, \bar{\boldsymbol{v}}_j^{[k]}) + \sum_{s \in \mathcal{N}_j^{\rightarrow}} \bar{J}_s(\boldsymbol{u}_s^{[k]}, \bar{\boldsymbol{v}}_s^{[k]} + \delta\bar{\boldsymbol{v}}_s) \tag{6}$$

and considers the Taylor expansion of the neighbors since the respective approximations (4) are added to the cost functionals of all agents. Furthermore, the part $J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]})$ in (4) can be neglected in the local OCP (5) as it does not depend on the variables of agent i. With the same reasoning, the linear part $\delta_{\bar{\boldsymbol{v}}_j} J_j(\boldsymbol{u}_j^{[k]}, \bar{\boldsymbol{v}}_j^{[k]})(\delta\bar{\boldsymbol{v}}_j)$ reduces to $\delta_{\boldsymbol{v}_i} \bar{J}_j(\boldsymbol{u}_j^{[k]}, \boldsymbol{v}_i^{[k]})(\delta\boldsymbol{v}_i)$. To simplify the calculation of the reduced Gâteaux derivative, it is convenient to define the augmented local cost as

$$\bar{J}_j(\boldsymbol{u}_j, \bar{\boldsymbol{v}}_j^{[k]}) = \bar{V}_j(\boldsymbol{x}_j) + \sum_{s\in\mathcal{N}_j^{\leftarrow}} V_{js}(\boldsymbol{x}_j, \boldsymbol{x}_s^{[k]}) + \int_0^T \bar{l}_j(\boldsymbol{x}_j, \boldsymbol{u}_j) + \sum_{s\in\mathcal{N}_j^{\leftarrow}} l_{js}(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{x}_s^{[k]}, \boldsymbol{u}_s^{[k]})\,\mathrm{d}t \tag{7}$$

with the modified local terminal and integral costs $\bar{V}_j(\boldsymbol{x}_j(T))$ and $\bar{l}_j(\boldsymbol{x}_j, \boldsymbol{u}_j)$ which additionally incorporate the linear terms which result from adding $\delta_{\boldsymbol{v}_j}\bar{J}_s(\boldsymbol{u}_s^{[k]}, \boldsymbol{v}_j^{[k]})(\delta\boldsymbol{v}_j)$ in (6). Then, $\delta_{\boldsymbol{v}_i}\bar{J}_j(\boldsymbol{u}_j^{[k]}, \boldsymbol{v}_i^{[k]})(\delta\boldsymbol{v}_i)$ can be compactly computed via the local Hamiltonian $\bar{H}_j$ to OCP (5) (compare [25], Section 3.6])

$$\bar{H}_j := \bar{l}_j(\boldsymbol{x}_j, \boldsymbol{u}_j) + \boldsymbol{\lambda}_j^{\mathrm{T}}\boldsymbol{f}_j(\boldsymbol{x}_j, \boldsymbol{u}_j) + \sum_{s\in\mathcal{N}_j^{\leftarrow}} H_{js} \tag{8}$$

with the abbreviation

$$H_{js} := l_{js}(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{x}_s^{[k]}, \boldsymbol{u}_s^{[k]}) + \boldsymbol{\lambda}_j^{\mathrm{T}}\boldsymbol{f}_{js}(\boldsymbol{x}_j, \boldsymbol{u}_j, \boldsymbol{x}_s^{[k]}, \boldsymbol{u}_s^{[k]}), \tag{9}$$

where $\boldsymbol{\lambda}_j \in \mathbb{R}^{n_j}$ is the solution to the adjoint system

$$\dot{\boldsymbol{\lambda}}_j = -\partial_{\boldsymbol{x}_j}\bar{H}_j, \quad \boldsymbol{\lambda}_j(T) = \partial_{\boldsymbol{x}_j}\bar{V}_j + \sum_{s\in\mathcal{N}_j^{\leftarrow}} \partial_{\boldsymbol{x}_j} V_{js} \tag{10}$$

associated with the local OCPs (5). This leads to the expression

$$\delta_{\boldsymbol{v}_i}\bar{J}_j(\boldsymbol{u}_j^{[k]}, \boldsymbol{v}_i^{[k]})(\delta\boldsymbol{v}_i) = \left(\partial_{\boldsymbol{x}_i}\bar{V}_j^{[k]} + \sum_{s\in\mathcal{N}_j^{\leftarrow}} \partial_{\boldsymbol{x}_i} V_{js}^{[k]}\right)^{\mathrm{T}} \delta\boldsymbol{x}_i(T) + \int_0^T \left(\partial_{\boldsymbol{u}_i}\bar{H}_j^{[k]}\right)^{\mathrm{T}}\delta\boldsymbol{u}_i + \left(\partial_{\boldsymbol{x}_i}\bar{H}_j^{[k]}\right)^{\mathrm{T}}\delta\boldsymbol{x}_i\,\mathrm{d}t, \tag{11}$$

which formally defines the sensitivity of the augmented cost functional of some neighbor $j \in \mathcal{N}_i^{\rightarrow}$ w.r.t. the states and controls of agent i. The variations in this context are defined as the deviation of the current state or control to the respective one in the previous iteration, i.e. $\delta\boldsymbol{x}_i := \boldsymbol{x}_i - \boldsymbol{x}_i^{[k]}$, $\delta\boldsymbol{u}_i := \boldsymbol{u}_i - \boldsymbol{u}_i^{[k]}$ and $\delta\boldsymbol{x}_i(T) := \boldsymbol{x}_i(T) - \boldsymbol{x}_i^{[k]}(T)$.

Due to the neighbor-affine structure of cost function and dynamics, the partial derivatives of the Hamiltonian and terminal cost w.r.t. $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$ in (11) simplify to

$$\partial_{\boldsymbol{u}_i}\bar{H}_j^{[k]} = \partial_{\boldsymbol{u}_i} l_{ji}^{[k]} + \left(\partial_{\boldsymbol{u}_i}\boldsymbol{f}_{ji}^{[k]}\right)^{\mathrm{T}}\boldsymbol{\lambda}_j^{[k]} \tag{12a}$$

$$\partial_{\boldsymbol{x}_i}\bar{H}_j^{[k]} = \partial_{\boldsymbol{x}_i} l_{ji}^{[k]} + \left(\partial_{\boldsymbol{x}_i}\boldsymbol{f}_{ji}^{[k]}\right)^{\mathrm{T}}\boldsymbol{\lambda}_j^{[k]} \tag{12b}$$

$$\partial_{\boldsymbol{x}_i}\bar{V}_j^{[k]} + \sum_{s\in\mathcal{N}_i^{\leftarrow}} \partial_{\boldsymbol{x}_i} V_{js}^{[k]} = \partial_{\boldsymbol{x}_i} V_{ji}^{[k]} \tag{12c}$$

as the remaining partial derivatives evaluate to zero

$$\partial_{\boldsymbol{u}_i} l_{js} = 0, \quad \partial_{\boldsymbol{x}_i} l_{js} = 0, \quad s \in \mathcal{N}_j^{\leftarrow}\backslash\{i\} \tag{12d}$$

$$\partial_{\boldsymbol{u}_i}\boldsymbol{f}_{js} = 0, \quad \partial_{\boldsymbol{x}_i}\boldsymbol{f}_{js} = 0, \quad s \in \mathcal{N}_j^{\leftarrow}\backslash\{i\} \tag{12e}$$

$$\partial_{\boldsymbol{x}_i}\bar{V}_j = 0, \quad \partial_{\boldsymbol{x}_i} V_{js} = 0, \quad s \in \mathcal{N}_j^{\leftarrow}\backslash\{i\}. \tag{12f}$$

An essential takeaway from (12a)–(12f) is that agents can calculate the sensitivities (11) locally and only need the states $\boldsymbol{x}_j$, controls $\boldsymbol{u}_j$ and adjoint states $\boldsymbol{\lambda}_j$ from their receiving neighbors if the agent has access to the functions l_{ji} and $\boldsymbol{f}_{ji}$.

This is a key advantage of neighbor-affine systems to the more general system description used in [13], in which the partial derivatives (12a)–(12c) needed to be calculated by the neighbors and communicated to the agents as (12d)–(12f) do not evaluate to zero in the general nonlinear case.

This is also in line with the results for linear systems where only the Lagrangian multipliers and optimization variables need to be communicated to neighboring agents [15]. Thus, the neighbor-affine system definition allows a description of the majority of nonlinear distributed systems and retains the advantage of only needing the states, controls and adjoint states of the respective neighbor similar to the linear case.

The relationship of coupling topology, neighborhoods and exchanged information is visualized in Figure 1 for an example system. The directions of the black arrows correspond to sending and receiving agents while the red arrows indicate the direction of communication.

3.2 Sensitivity-based DMPC algorithm

The preceding considerations regarding the calculation of the sensitivities, solution of the local OCP, and exchange of information are summarized in Algorithm 1. The two calculation steps 2 and 3 can be computed in parallel by each

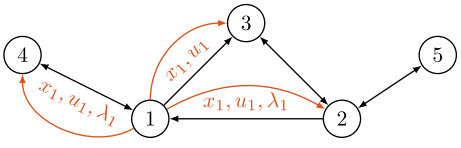

Figure 1: Coupling topology (black) and exchanged variables (red) for an example system of five agents. The neighborhood of agent 1, $\mathcal{N}_1 = \{2, 3, 4\}$, comprises the sending neighbors $\mathcal{N}_1^{\leftarrow} = \{2, 4\}$ and the receiving neighbors $\mathcal{N}_1^{\rightarrow} = \{3, 4\}$. The outgoing communication from agent 1 is visualized in red.

Algorithm 1: Sensitivity-based distributed optimal control.

Initialize $\boldsymbol{u}_i^{[0]}, \boldsymbol{x}_i^{[0]} = \boldsymbol{x}_{i,0}, \lambda_i^{[0]}$, set $k \leftarrow 0$, and send initial trajectories $\boldsymbol{u}_i^{[0]}, \boldsymbol{x}_i^{[0]}$ to all neighbors $j \in \mathcal{N}_i$ and $\lambda_i^{[0]}$ to sending neighbors $\mathcal{N}_i^{\rightarrow}$

1: **While** $k < k_{\max}$ **do** in parallel for each $i \in \mathcal{V}$

2: Calculate partial derivatives for all receiving neighbors $j \in \mathcal{N}_i^{\rightarrow}$ according to (12a)–(12c)

3: Compute $\boldsymbol{u}_i^{[k+1]}, \boldsymbol{x}_i^{[k+1]}, \lambda_i^{[k+1]}$ by solving the OCP with sensitivities (11)

$$\begin{aligned} \min_{\boldsymbol{u}_i} \quad & J_i\left(\boldsymbol{u}_i, \bar{\boldsymbol{v}}_i^{[k]}\right) + \sum_{j \in \mathcal{N}_i^{\rightarrow}} \delta_{\boldsymbol{v}_i} \bar{J}_j\left(\boldsymbol{u}_j^{[k]}, \boldsymbol{v}_i^{[k]}\right)(\delta \boldsymbol{v}_i) \\ \text{s.t.} \quad & \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}\left(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j^{[k]}, \boldsymbol{u}_j^{[k]}\right), \\ & \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \end{aligned} \tag{14}$$

4: Send $\boldsymbol{u}_i^{[k+1]}$ and $\boldsymbol{x}_i^{[k+1]}$ to all neighbors $j \in \mathcal{N}_i$

5: Send $\lambda_i^{[k+1]}$ to all sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$

6: Set $k \leftarrow k+1$

agent $i \in \mathcal{V}$, whereas step 3 involves the solution of a local OCP.

After the calculations are finished by each agent, the resulting trajectories $\boldsymbol{x}_i$ and $\boldsymbol{u}_i$ are communicated to all neighbors in the set $\mathcal{N}_i$ since both receiving and sending neighbors require this information for the evaluation of the system dynamics (5b) and the calculation of the partial derivatives (12a) and (12b), respectively. The adjoint states λ_i only need to be communicated in step 5 to the sending neighbors for the computation of partial derivatives of the Hamiltonian $\bar{H}_j$ in (11). In practice, it is more suitable to concatenate the communication steps 4 and 5 into one step and send the adjoint states to all agents to reduce the communication effort. It is therefore notable that this algorithm only requires one communication step in each iteration. The algorithm is executed until a maximum number of iterations $k_{\max}$ have been reached or a suitable convergence criterion is satisfied.

It should be pointed out that the algorithm ensures scalability w.r.t. the number of agents as only neighbor-to-neighbor communication and no central communication step is necessary. Thus, the computation and communication effort is decoupled from the overall system size and is bounded by the cardinality of an agent's neighborhood.

Furthermore, it is possible to enhance the convergence behavior of the Algorithm 1 by adding certain convexification terms to the local cost function (2). If the local cost function (2) is extended by the convexification terms

$$\int_0^T \frac{1}{2}\|\boldsymbol{x}_i - \boldsymbol{x}_i^{[k]}\|_{\boldsymbol{C}_x}^2 + \frac{1}{2}\|\boldsymbol{u}_i - \boldsymbol{u}_i^{[k]}\|_{\boldsymbol{C}_u}^2 \, \mathrm{d}t \tag{13}$$

with the positive definite weighting matrices $\boldsymbol{C}_x \in \mathbb{R}^{n_i \times n_i}$ and $\boldsymbol{C}_u \in \mathbb{R}^{m_i \times m_i}$, the local minimization step in (14) is similar to a local application of the proximity operator in each algorithm step k. A convexification of the form (13) typically improves the convergence behavior and robustness of Algorithm 1 under the aspect that nonlinear optimal control problems, like (1), are in general nonconvex optimization problems within function spaces.

In the following, the equivalence of the optimality conditions of the centralized OCP (1) and those of the local OCP (14) in Algorithm 1 is investigated. To this end, the Hamiltonian of the central OCP (1)

$$H(\boldsymbol{x}, \boldsymbol{u}, \lambda) = \sum_{i \in \mathcal{V}} \left[l_i + \lambda_i^{\mathrm{T}} \boldsymbol{f}_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \left(l_{ij} + \lambda_i^{\mathrm{T}} \boldsymbol{f}_{ij} \right) \right] \tag{15}$$

with the stacked notation $\boldsymbol{x} = \boldsymbol{x}_{\mathcal{V}}, \boldsymbol{u} = \boldsymbol{u}_{\mathcal{V}}, \lambda = \lambda_{\mathcal{V}}$ is needed.

Theorem 1. *Under the assumption that Algorithm 1 is convergent, the iterates $\left(\boldsymbol{x}_i^k, \boldsymbol{u}_i^k\right)$ of the local OCP (14) satisfy the first-order optimality conditions of the central OCP (1) in the limit.*

Proof. Suppose that $\boldsymbol{u}_i^*(t), i \in \mathcal{V}$ is a (local) solution of OCP (1) and let $\boldsymbol{x}_i^*(t), i \in \mathcal{V}$ denote the optimal response of the system. Then, there exist continuously differentiable vector functions $\lambda_i^*(t), i \in \mathcal{V}$ such that the canonical equations

$$\dot{\boldsymbol{x}}_i^* = \partial_{\lambda_i} H(\boldsymbol{x}^*, \boldsymbol{u}^*, \lambda^*), \quad \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \tag{16a}$$

$$\dot{\lambda}_i^* = -\partial_{\boldsymbol{x}_i} H(\boldsymbol{x}^*, \boldsymbol{u}^*, \lambda^*), \quad \lambda_i(T) = \lambda_{i,T} \tag{16b}$$

with $\lambda_{i,T} = \partial_{\boldsymbol{x}_i} \sum_{i \in \mathcal{V}} \left[V_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} V_{ij} \right]$ and the stationarity condition

$$0 = \partial_{\boldsymbol{u}_i} H(\boldsymbol{x}^*, \boldsymbol{u}^*, \lambda^*) \tag{16c}$$

are satisfied for all agents $i \in \mathcal{V}$ and $t \in [0, T]$.

In the following, it will be shown that (16a)–(16c) are equivalent to the local optimality conditions of the individual agents. To this end, define the local Hamiltonian $\bar{H}_i$ for agent $i \in \mathcal{V}$ according to $\bar{H}_j$ in (8)

$$\begin{aligned} \bar{H}_i = {} & l_i + \lambda_i^{\mathrm{T}} \boldsymbol{f}_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \left(l_{ij} + \lambda_i^{\mathrm{T}} \boldsymbol{f}_{ij} \right) \\ & + \sum_{j \in \mathcal{N}_i^{\rightarrow}} \left(\partial_{\boldsymbol{u}_i} \bar{H}_j^{[k]} \right)^{\mathrm{T}} \delta \boldsymbol{u}_i + \left(\partial_{\boldsymbol{x}_i} \bar{H}_j^{[k]} \right)^{\mathrm{T}} \delta \boldsymbol{x}_i. \end{aligned} \tag{17}$$

Recall the definition of the variations as $\delta \boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_i^{[k]}$ and $\delta \boldsymbol{u}_i = \boldsymbol{u}_i - \boldsymbol{u}_i^{[k]}$. The optimality conditions for the solution $\boldsymbol{u}_i^{[k+1]}, \boldsymbol{x}_i^{[k+1]}$ of the local problem then read

$$\dot{\boldsymbol{x}}_i^{[k+1]} = \partial_{\lambda_i} \bar{H}_i, \quad \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \tag{18a}$$

$$\dot{\lambda}_i^{[k+1]} = -\partial_{x_i} \bar{H}_i, \quad \lambda_i(T) = \bar{\lambda}_{i,T} \tag{18b}$$

$$0 = \partial_{u_i} \bar{H}_i \tag{18c}$$

with the terminal condition $\bar{\lambda}_{i,T} = \partial_{x_i} \bar{V}_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \partial_{x_i} V_{ij}$. Note that $\bar{\lambda}_{i,T} = \lambda_{i,T}$ in the limit.

It is easily shown that the dynamic Equations (16a) and (18a) of central and distributed problem are identical due to

$$\partial_{\lambda_i} H = \partial_{\lambda_i} \bar{H}_i = \boldsymbol{f}_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}. \tag{19}$$

To show that (16b) and (18b) are equivalent, a closer look is taken at the partial derivative in (16b)

$$\begin{aligned} \partial_{x_i} H = \partial_{x_i} l_i + \left(\partial_{x_i} \boldsymbol{f}_i\right)^{\mathrm{T}} \lambda_i^* + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \partial_{x_i} l_{ij} + \left(\partial_{x_i} \boldsymbol{f}_{ij}\right)^{\mathrm{T}} \lambda_i^* \\ + \sum_{j \in \mathcal{N}_i^{\rightarrow}} \partial_{x_i} l_{ji} + \left(\partial_{x_i} \boldsymbol{f}_{ji}\right)^{\mathrm{T}} \lambda_j^*. \end{aligned} \tag{20}$$

These terms are identical to the terms of $\partial_{x_i} \bar{H}_i$ in the local optimality condition (18b)

$$\begin{aligned} \partial_{x_i} \bar{H}_i = \partial_{x_i} l_i + \left(\partial_{x_i} \boldsymbol{f}_i\right)^{\mathrm{T}} \lambda_i^{[k+1]} + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \partial_{x_i} l_{ij} + \left(\partial_{x_i} \boldsymbol{f}_{ij}\right)^{\mathrm{T}} \lambda_i^{[k+1]} \\ + \partial_{x_i} \sum_{j \in \mathcal{N}_i^{\rightarrow}} \left(\partial_{u_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{u}_i + \left(\partial_{x_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{x}_i, \end{aligned} \tag{21}$$

since the parts involving the receiving neighbors $\mathcal{N}_i^{\rightarrow}$ are equal as well

$$\begin{aligned} \partial_{x_i} \sum_{j \in \mathcal{N}_i^{\rightarrow}} \left(\partial_{u_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{u}_i + \left(\partial_{x_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{x}_i \\ = \sum_{j \in \mathcal{N}_i^{\rightarrow}} \partial_{x_i} \bar{H}_j^{[k]} = \sum_{j \in \mathcal{N}_i^{\rightarrow}} \partial_{x_i} l_{ji} + \left(\partial_{x_i} \boldsymbol{f}_{ji}\right)^{\mathrm{T}} \lambda_j^{[k]}, \end{aligned} \tag{22}$$

which demonstrates the equivalence of both optimality conditions. Note that in the limit, under the assumption of convergence, iterates at $k+1$ and k are identical.

The same applies to the stationarity conditions (16c) and (18c) as again the terms involving the sending neighbors are equal, i.e.

$$\begin{aligned} \partial_{u_i} \sum_{j \in \mathcal{N}_i^{\rightarrow}} \left(\partial_{u_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{u}_i + \left(\partial_{x_i} \bar{H}_j^{[k]}\right)^{\mathrm{T}} \delta \boldsymbol{x}_i \\ = \sum_{j \in \mathcal{N}_i^{\rightarrow}} \partial_{u_i} \bar{H}_j^{[k]} = \sum_{j \in \mathcal{N}_i^{\rightarrow}} \partial_{u_i} l_{ji} + \left(\partial_{u_i} \boldsymbol{f}_{ji}\right)^{\mathrm{T}} \lambda_j^{[k]}, \end{aligned} \tag{23}$$

which demonstrates that the set of optimality conditions (16) and (18) correspond. □

The strong assumption of convergence of Algorithm 1 is required to show the equivalency of the optimality conditions since this can only hold in the limit, i.e. $k \to \infty$. Convergence has been proven for linear discrete-time systems in [15], for nonlinear systems under convexity assumptions with a not fully distributed algorithm variant considering the complete dynamics on agent level [26] and in particular for Algorithm 1 although with some restrictions on the dynamics [27].

If the local problems are additionally strictly convex, the converged iterates of the distributed solution correspond to the central solution. Furthermore, (22) and (23) indicate that the sensitivities can also be interpreted as the needed terms to extend the Hamiltonian in such a way that the first-order optimality conditions of central and distributed problem correspond. Additionally, Theorem 1 is still valid if the term (13) is added to the local cost function (5a) as it evaluates to zero if $\boldsymbol{x}_i^{[k]}$ and $\boldsymbol{u}_i^{[k]}$ approach the optimal solution. With the focus of this work on the derivation and real-time capable implementation of Algorithm 1, Theorem 1 is also important from a practical viewpoint as one can be sure that if the algorithm converges, then the solution trajectories fulfill the central optimality conditions and one has solved the same problem.

3.3 ADMM algorithm

In the following, the ADMM algorithm [8], which is widely used in distributed optimization and DMPC, is shortly recapitulated to highlight the differences to the sensitivity-based algorithm. Note that there exist many different ADMM variants for various application areas even in the context of DMPC. That is why the following considerations are restricted to the ADMM formulation from [9] which possesses similar structural characteristics (e.g. solution of a local OCP, neighbor-to-neighbor communication, fully distributed with no central computation steps) as Algorithm 1 and enables a fair comparison. As the derivation and implementation of the ADMM algorithm for continuous-time, nonlinear systems was already extensively covered in [9], only the main computation steps are presented here.

The ADMM algorithm solves OCP (1) in a distributed manner by introducing local copies $\bar{\boldsymbol{x}}_{ji} \in \mathbb{R}^{n_j}$ and $\bar{\boldsymbol{u}}_{ji} \in \mathbb{R}^{m_j}$ of the variables $\boldsymbol{x}_j$ and $\boldsymbol{u}_j$, $j \in \mathcal{N}_i^{\leftarrow}$, which serve as additional optimization variables and are used to evaluate the coupled dynamics and costs. The summarized optimization vector is denoted as $\boldsymbol{w}_i = \left[\boldsymbol{u}_i^{\mathrm{T}} \ \bar{\boldsymbol{x}}_{\mathcal{N}_i^{\leftarrow}}^{\mathrm{T}} \ \bar{\boldsymbol{u}}_{\mathcal{N}_i^{\leftarrow}}^{\mathrm{T}}\right]^{\mathrm{T}}$. The equality constraint that local copies and original variables must coincide in the optimal solution is enforced through local coupling variables $\boldsymbol{z}_i \in \mathbb{R}^{n_i + m_i}$. This constraint is adjoined to the cost function (2) utilizing Lagrangian multipliers $\boldsymbol{\mu}_i = \left[\boldsymbol{\mu}_{ii}^{\mathrm{T}} \ \boldsymbol{\mu}_{\mathcal{N}_i^{\leftarrow}}^{\mathrm{T}}\right]^{\mathrm{T}}$ and a quadratic penalty term with penalty factor $\rho > 0$ to form the augmented Lagrangian cost functional

Algorithm 2: ADMM Algorithm.

Initialize $\boldsymbol{w}_i^{[0]}, \boldsymbol{z}_i^{[0]}$ and $\boldsymbol{\mu}_i^{[0]}$, choose ρ, set $k \leftarrow 0$
1: **While** $k < k_{\max}$ **do** in parallel for each $i \in \mathcal{V}$
2: Compute local variable $\boldsymbol{w}_i^{[k+1]}$:

$$\begin{aligned} \min_{\boldsymbol{w}_i} \quad & L_i\left(\boldsymbol{w}_i, \boldsymbol{\mu}_i^{[k]}, \boldsymbol{z}_i^{[k]}, \boldsymbol{z}_{\mathcal{N}_i^{\leftarrow}}^{[k]}\right) \\ \text{s.t.} \quad & (1b) - (1c) \end{aligned} \tag{25}$$

3: Send local copies $\bar{\boldsymbol{x}}_{ji}^{[k+1]}, \bar{\boldsymbol{u}}_{ji}^{[k+1]}$ to neighbors $j \in \mathcal{N}_i^{\leftarrow}$
4: Compute coupling variable $\boldsymbol{z}_i^{[k+1]}$:

$$\min_{\boldsymbol{z}_i} \sum_{i \in \mathcal{V}} L_i\left(\boldsymbol{w}_i^{[k+1]}, \boldsymbol{\mu}_i^{[k]}, \boldsymbol{z}_i, \boldsymbol{z}_{\mathcal{N}_i^{\leftarrow}}^{[k]}\right) \tag{26}$$

5: Send coupling variables $\boldsymbol{z}_i^{[k+1]}$ to neighbors $j \in \mathcal{N}_i^{\rightarrow}$
6: Update Lagrangian multipliers $\boldsymbol{\mu}_i^{[k+1]}$:
(27)

$$\boldsymbol{\mu}_{ii}^{[k+1]} = \boldsymbol{\mu}_{ii}^{[k]} + \rho\left(\boldsymbol{z}_i^{[k+1]} - \begin{bmatrix} \boldsymbol{x}_i^{[k+1]} \\ \boldsymbol{u}_i^{[k+1]} \end{bmatrix}\right) \tag{27}$$

$$\boldsymbol{\mu}_{ji}^{[k+1]} = \boldsymbol{\mu}_{ji}^{[k]} + \rho\left(\boldsymbol{z}_j^{[k+1]} - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji}^{[k+1]} \\ \bar{\boldsymbol{u}}_{ji}^{[k+1]} \end{bmatrix}\right), \quad j \in \mathcal{N}_i^{\leftarrow} \tag{28}$$

7: Send Lagrangian multipliers $\boldsymbol{\mu}_i{}^{[k+1]} \boldsymbol{\mu}_i$ to neighbors $j \in \mathcal{N}_i^{\leftarrow}$
8: Set $k \leftarrow k + 1$

$$\begin{aligned} & L_i\left(\boldsymbol{w}_i, \boldsymbol{\mu}_i, \boldsymbol{z}_i, \boldsymbol{z}_{\mathcal{N}_i^{\leftarrow}}\right) \\ & = J_i(\boldsymbol{w}_i) + \int_0^T \boldsymbol{\mu}_{ii}^{\mathrm{T}}\left(\boldsymbol{z}_i - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix}\right) + \frac{\rho}{2}\left\| \boldsymbol{z}_i - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} \right\|^2 \\ & \quad + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{\mu}_{ji}^{\mathrm{T}}\left(\boldsymbol{z}_j - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix}\right) + \frac{\rho}{2}\left\| \boldsymbol{z}_j - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix} \right\|^2 \mathrm{d}t. \end{aligned} \tag{24}$$

The notation $\boldsymbol{\mu}_{\mathcal{N}_i^{\leftarrow}}$ refers to the stacked vectors of Lagrangian multipliers $\boldsymbol{\mu}_{ji} \in \mathbb{R}^{n_j + m_j}$. The resulting min-max problem is solved in an alternating fashion by Algorithm 2. It should be pointed out that each step in Algorithm 2 is fully decoupled for each agent. Hence, the algorithm is fully distributed and the computational effort is spread over all agents.

The algorithm consists of the three computation steps 2, 4, and 6 of which the first one solves a local OCP similar to step 3 of Algorithm 1. The other two steps can be carried out analytically and are therefore less computationally expensive. Convergence of the ADMM algorithm in the context of DMPC for nonlinear continuous-time systems has been shown under certain assumptions in [28].

While Algorithm 1 directly considers the states of the neighbors in the previous iteration k to evaluate the coupled dynamics $\boldsymbol{f}_{ij}$ and costs l_{ij}, Algorithm 2 relies on the local copies to compute the influence of $\boldsymbol{f}_{ij}$ and l_{ij}. Therefore, the ADMM algorithm has to solve local OCPs of higher dimensionality than Algorithm 1 for the same problem. Furthermore, three communication steps are necessary to provide the neighbors with information for their computation steps, compared to the one communication step of Algorithm 1.

3.4 Asynchronous execution

DMPC algorithms are usually executed synchronously meaning that a central entity, often called coordinator, triggers each algorithm step to ensure that all the agents are at the same iteration k. This has the disadvantage that every agent needs to wait for the slowest agent in the network to continue its calculations. The heterogeneity in execution time might arise from different computational burdens of the individual subproblems or discrepancies in communication time and is especially dominant on distributed hardware with non-negligible communication times [20, 21].

To mitigate these effects, one can allow the agents to perform the algorithm steps asynchronously in the sense that one agent might be at iteration $k+1$ while its neighbor is still at iteration k. Therefore, a delay d_{ji}

$$k_{\max} > d_{ji} \geq 0, \quad i \in \mathcal{V},\ j \in \mathcal{N}_i \tag{29}$$

between the algorithm steps is introduced [29]. This has the consequence that the agents use outdated information, for example, to calculate the sensitivities of their neighbors, which is expected to reduce the convergence speed as no new information is processed. On the other hand, the agents are now able to effectively use their waiting times as they can directly proceed with the next algorithm step. The idea is that this trade-off is beneficial for the real-time capability of a DMPC scheme on distributed platforms.

To formalize this concept, let the counter k_i denote the iteration $k_i \leq k$ of the agent i w.r.t. the global counter k and drop the distinction between sending and receiving neighbors for the sake of readability. Then, the delay can be defined as difference between the current iteration k_i of agent i and some previous iteration k_j of its neighbor $j \in \mathcal{N}_j$, i.e.

$$d_{ji} = k_i - k_j, \quad i \in \mathcal{V},\ j \in \mathcal{N}_i. \tag{30}$$

Furthermore, define $\mathcal{N}_i^{[k_i]}$ as the set of neighbors that sent their trajectories during the time step k_i and $\bar{\mathcal{N}}_i^{[k_i]}$ as the set of neighbors that did not send their trajectories. Note that $\mathcal{N}_i^{[k_i]} \cap \bar{\mathcal{N}}_i^{[k_i]} = \emptyset$. Let $\boldsymbol{t}_j^{[k_j]}$ be some trajectory, for example a state or control trajectory, that the agent receives

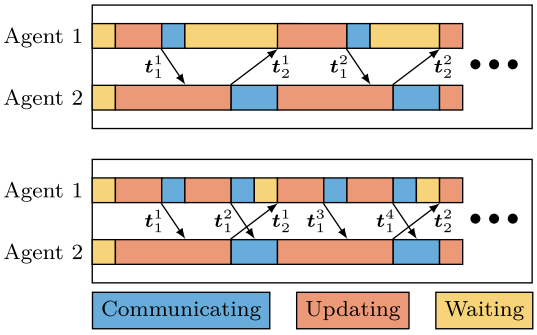

Figure 2: Synchronous (top) and asynchronous (bottom) execution of Algorithm 1. The different colors visualize the respective state of the agent.

from its neighbor. Then, the agent updates its version of the quantity by performing the update

$$\boldsymbol{t}_j^{[k_i]} \leftarrow \begin{cases} \boldsymbol{t}_j^{[k_j]}, & \forall j \in \mathcal{N}_i^{[k_i]}, \\ \boldsymbol{t}_j^{[k_i-1]}, & \forall j \in \bar{\mathcal{N}}_i^{[k_i]}. \end{cases} \tag{31}$$

The definition (30) can be written in the same fashion as an update rule such that k_j does not need to be transmitted

$$d_{ji} \leftarrow \begin{cases} 0, & \forall j \in \mathcal{N}_i^{[k_i]}, \\ d_{ji} + 1, & \forall j \in \bar{\mathcal{N}}_i^{[k_i]}. \end{cases} \tag{32}$$

In other words, if a neighbor sends his trajectories on time, the respective delay is set to zero for that neighbor, otherwise d_{ji} is increased by one. In this context, the global counter is increased if one agent $i \in \mathcal{V}$ performs all its algorithm steps. The delay d_{ji} is bounded from above

$$d_{ji} \leq d_{\max} \tag{33}$$

with a maximum delay $d_{\max} \geq 0$, as an unbounded delay would lead to the divergence of the algorithm since in the worst case no information would be exchanged.

This is known as the partially asynchronous model [29], meaning that each neighbor must send its needed variables within the period $[k_i - d_{\max}, k_i]$. Otherwise, that agent must wait. Note that while a single delay is sufficient for the sensitivity-based algorithm, the ADMM algorithm requires three delays $d_{ji,u}, d_{ji,z}$ and $d_{ji,\mu}$, as three communication steps exist that possibly introduce delayed data.

The situation of the synchronous and asynchronous execution of Algorithm 1 is depicted in Figure 2 for the example of two neighboring agents for a maximum tolerable delay of $d_{\max} = 1$. The corresponding communicated quantities $\boldsymbol{x}_i, \boldsymbol{u}_i$ and λ_i are stacked to $\boldsymbol{t}_i^{\mathrm{T}} = [\boldsymbol{x}_i^{\mathrm{T}}\ \boldsymbol{u}_i^{\mathrm{T}}\ \lambda_i^{\mathrm{T}}]$. The diagram for the asynchronous execution of the ADMM

Algorithm 3: Asynchronous sensitivity-based distributed optimal control.

Initialize $\boldsymbol{u}_i^{[0]}, \boldsymbol{x}_i^{[0]} = \boldsymbol{x}_{i,0}, \lambda_i^{[0]}$, set $k_i \leftarrow 0$, and send initial trajectories, set $d_{ji} = 0,\ j \in \mathcal{N}_i$

1: **Wait** until $\left(\boldsymbol{x}_j^{[k_j]}, \boldsymbol{u}_j^{[k_j]}\right),\ j \in \mathcal{N}_i$ and $\lambda_j^{k_j},\ j \in \mathcal{N}_i^{\rightarrow}$ have been received such that $d_{ji} \leq d_{\max}$

2: Update $\boldsymbol{x}_j^{[k_i]}, \boldsymbol{u}_j^{[k_i]}$ and $\lambda_j^{[k_i]}$ according to (31)

3: Calculate partial derivatives for all receiving neighbors $j \in \mathcal{N}_i^{\rightarrow}$ according to (12a)–(12c)

4: Compute $\boldsymbol{u}_i^{[k_i+1]}, \boldsymbol{x}_i^{[k_i+1]}, \lambda_i^{[k_i+1]}$ by solving the OCP (14)

5: Update delay of neighbors according to the rule (32)

6: Send $\left(\boldsymbol{u}_i^{[k_i+1]}, \boldsymbol{x}_i^{[k_i+1]}\right)$ to neighbors $j \in \mathcal{N}_i$ and $\lambda_i^{[k_i+1]}$ to sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$

7: Repeat with $k_i \leftarrow k_i + 1$ as long as $k_i < k_{\max}$

algorithm would have the same structure as the sensitivity-based algorithm in Figure 2 albeit with three update and communication steps.

In view of the preceding considerations, the required steps to execute Algorithm 1 asynchronously are summarized in Algorithm 3 describing the asynchronous sensitivity-based algorithm for distributed optimal control. The computation steps 3 and 4 are the same as in the synchronous case, however, using delayed trajectories of the neighbors.

To ensure that the information is not too outdated, each agent keeps track of the relative difference between its iterations and those of the neighbors by performing the delay update rule (32) in step 5 and checking the respective delay in step 1 before continuing with its calculations. This allows the agents to independently perform their computation steps without needing to wait for the slowest agent while ensuring that the iterations are at maximum $d_{\max}$ iterations apart.

Theorem 1 holds in the same way as in the synchronous case, as indicated by Theorem 2.

Theorem 2. *Under the assumption that Algorithm 3 is convergent and that the delay is bounded, i.e.* $d_{ji} < d_{\max}$, *the iterates* $(\boldsymbol{x}_i^{[k]}, \boldsymbol{u}_i^{[k]})$ *of the local OCP* (14) *in Algorithm 3 satisfy the optimality conditions of the central OCP* (1) *in the limit.*

Proof. The chain of reasoning follows the proof of the synchronous case. The only difference is that the extended local Hamiltonian $\tilde{H}_j$ in (17), l_{ij} as well as $\boldsymbol{f}_{ij}$ use the delayed trajectories at step k_i instead of k. The optimality conditions (16) and (18) must hold regardless of the delay and therefore (22) and (23) are valid which shows that if the Algorithm 3 converges, which is certainly a stronger assumption than in the synchronous case, that the iterates satisfy the central optimality conditions. □

Algorithm 4: Asynchronous ADMM (a-ADMM).

Initialize $\boldsymbol{w}_i^{[0]}$, $\boldsymbol{z}_i^{[0]}$ and $\boldsymbol{\mu}_i^{[0]}$, choose ρ, set $k_i \leftarrow 0$, set $d_{ji,\mu} = d_{ji,u} = d_{ji,z} = 0$
1: **While** $k_i < k_{\max}$ **do** in parallel for each $i \in \mathcal{V}$
2: **Wait** until $\boldsymbol{\mu}_{ij}^{[k_j]}$, $j \in \mathcal{N}_i^{\rightarrow}$ have been received such that $d_{ji,\mu} \leq d_{\max}$
3: Update $\boldsymbol{\mu}_{ij}^{[k_j]}$ according to (31) and compute local variable $\boldsymbol{w}_i^{[k_i+1]}$ according to (25)
4: Send local copies $\left(\bar{\boldsymbol{x}}_{ji}^{[k_i+1]}, \bar{\boldsymbol{u}}_{ji}^{[k_i+1]}\right)$ to sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$ and set delay $d_{ji,u}$ according to (32)
5: **Wait** until $\left(\bar{\boldsymbol{x}}_{ij}^{[k_j]}, \bar{\boldsymbol{u}}_{ij}^{[k_j]}\right)$, $j \in \mathcal{N}_i^{\rightarrow}$ have been received such that $d_{ji,u} \leq d_{\max}$
6: Update $\bar{\boldsymbol{x}}_{ij}^{[k_j]}$ and $\bar{\boldsymbol{u}}_{ij}^{[k_j]}$ according to (31) and compute coupling variables $\boldsymbol{z}_i^{[k_i+1]}$ according to (26)
7: Send coupling variables $\boldsymbol{z}_i^{[k_i+1]}$ to receiving neighbors $j \in \mathcal{N}_i^{\rightarrow}$ and set delay $d_{ji,z}$ according to (32)
8: **Wait** until $\boldsymbol{z}_j^{[k_j]}$, $j \in \mathcal{N}_i^{\leftarrow}$ have been received such that $d_{ji,z} \leq d_{\max}$
9: Update $\boldsymbol{z}_j^{[k_j]}$ according to (31) and compute Lagrangian multipliers $\boldsymbol{\mu}_i^{[k_i+1]}$ by maximization of the augmented Lagrangian (24) via steepest ascent
10: Send Lagrangian multipliers $\boldsymbol{\mu}_i^{[k_i+1]}$ to sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$ and set delay $d_{ji,\mu}$ according to (32)
11: Set $k_i \leftarrow k_i + 1$

Similar to the synchronous formulation, the converged iterates of Algorithm 3 correspond to the central solution if the local problems are assumed to be strictly convex. In the same manner, the synchronous ADMM algorithm can be adapted to consider delayed information similar as in [21]. The respective steps of the asynchronous ADMM (a-ADMM) algorithm are summarized in Algorithm 4. Convergence of a similar asynchronous ADMM algorithm variant in the context of DMPC for nonlinear continuous-time systems is shown under certain assumptions in [30].

Since Algorithm 2 requires three communication steps, the agents must check the delays of their neighbors before each computation step to verify that condition (33) is satisfied. It can be expected that the effect on the saved waiting time of the asynchronous ADMM algorithm will be greater than that of the sensitivity-based algorithm, as waiting time can be reduced in three communication steps compared to one.

4 Implementation and numerical evaluation

This section aims to explain the implementation structure which is needed to realize the distributed controller architecture as well as to provide a numerical evaluation of the presented algorithms which is divided into two parts. First, the synchronous sensitivity-based and ADMM algorithms are compared in terms of the influence of network topology, the coupling strength between agents, and the impact of nonconvex optimal control problems. Then, the synchronous and asynchronous versions of the algorithms are evaluated on distributed hardware in order to characterize computational effort, communication, and waiting times in a real distributed optimization setting using the TCP/IP protocol for communication. The evaluation hardware consists of four Raspberry Pi 3B+, running Raspberry Pi OS.

4.1 Modular structure and framework

The C++ toolbox GRAMPC-D is a DMPC framework that is structured in a modular fashion to achieve a scalable and flexible implementation [9]. It already utilizes the synchronous ADMM Algorithm 2 as a distributed optimization algorithm to achieve coordination among agents. In the following, the main additions to the structure of GRAMPC-D are laid out which are needed to implement Algorithms 1, 3, and 4 in the same modular fashion. Figure 3 visualizes the distributed control structure. Each agent only has access to its local variables and communication is required to acquire data from other agents. In a distributed setup each agent creates its own instance of the communication interface that enables data transfer over a network. Currently, TCP/IP is used as the communication protocol but the modular design allows the implementation of further protocols. Depending on which algorithm is chosen, a local solver is initialized which holds an abstract implementation of the local OCPs and the implementation of the chosen algorithm. The optimal control toolbox GRAMPC [31] is used to solve the respective local OCPs.

Whether the algorithms are executed synchronously or asynchronously is encapsulated in the step selector as it ensures that the agent is always at the correct step in the algorithm. A coordinator performs higher-level algorithm steps like registering agents, initiating and terminating the algorithm, or in case of a synchronous execution triggering the individual algorithm steps. Note that the coordinator does not perform any optimization steps. Therefore, the optimization is performed fully distributed with no central communication. The last module is a simulator which enables a simulation of the overall system independent of the chosen solver.

GRAMPC-D considers a more general OCP than considered in (1) which additionally incorporates input box constraints, (coupled) nonlinear equality constraints, and (coupled) nonlinear inequality constraints. Similar to how coupled dynamics and costs influence the sensitivities, coupled constraints also need to be accounted for in the calculation

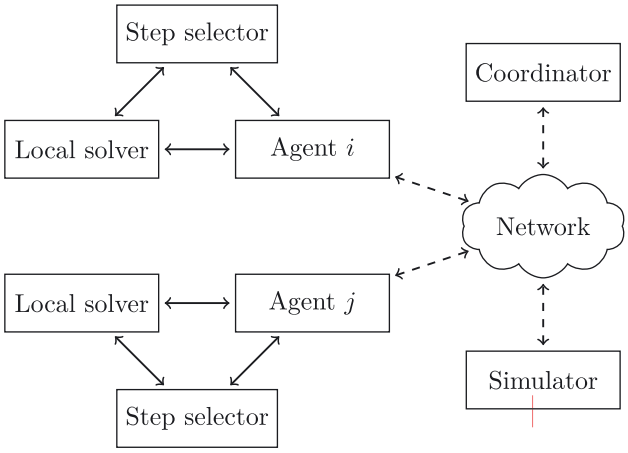

Figure 3: Each agent has its own local solver which is initialized depending on which algorithm is chosen. Each local solver has its own step selector which executes the steps of the corresponding algorithm in the correct order. A coordinator initializes, monitors, and terminates the algorithm. Communication between neighboring agents and the coordinator is enabled with a communication interface. A simulator handles the simulation of the overall system.

of the sensitivities as well as in the solution of OCP (14). Details in that regard are presented in the Appendix. This shows that generic DMPC applications with constraints can be handled by the sensitivity-based algorithm. The synchronous and asynchronous versions of the sensitivity-based algorithm as well as the asynchronous ADMM algorithm are implemented within GRAMPC-D, which is licensed under the Berkeley Software Distribution 3-clause version (BSD-3) license. The complete source code is available at GitHub https://github.com/grampc/grampc-d.

4.2 Investigation of topology, coupling strength and scalability

The influence of network topology and the coupling strength on the convergence behavior of the algorithms presents an interesting aspect of distributed optimal control algorithms and has been analyzed for linear systems for various DMPC algorithms in [32]. In this paper, it is analyzed for a nonlinear benchmark system of ten coupled Van-der-Pol oscillators which are described by the following differential equation

$$\ddot{\phi}_i = (1-\phi_i^2)\dot{\phi}_i - \phi_i + u_i + \sum_{j\in\mathcal{N}_i^{\leftarrow}} \sigma(\phi_i - \phi_j) \tag{34}$$

with the states $\boldsymbol{x}_i = \begin{bmatrix}\phi_i & \dot{\phi}_i\end{bmatrix}^{\mathrm{T}}$, input u_i, and the coupling strength $\sigma > 0$. The quadratic formulations

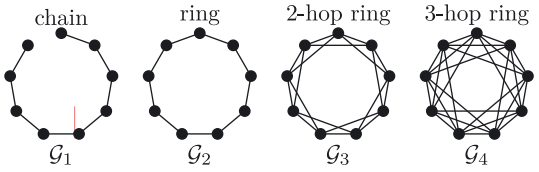

Figure 4: Different graph topologies.

$$\begin{aligned} l_i(\boldsymbol{x}_i, \boldsymbol{u}_i) &= \frac{1}{2}\left(\|\boldsymbol{x}_i\|_{\boldsymbol{Q}_i}^2 + R_i u_i^2\right), \\ V_i(\boldsymbol{x}_i(T)) &= \frac{1}{2}\|\boldsymbol{x}_i(T)\|_{\boldsymbol{P}_i}^2 \end{aligned} \tag{35}$$

are used for the integral and terminal costs. The OCP is parameterized by

$$\boldsymbol{Q}_i = \mathrm{diag}(1,1), \quad R_i = 0.1, \quad \boldsymbol{P}_i = \mathrm{diag}(1,1) \tag{36}$$

and the horizon is set to $T = 0.75$ s with a sampling time of $\Delta t = 0.1$ s. The initial condition is randomly distributed on the interval $\boldsymbol{x}_{i,0} \in [0.25, 1] \times [0, 0]$ while the control is constrained to $u_i \in [-1, 1]$.

The number of iterations required to achieve convergence in the first DMPC time step are now compared for varying coupling strengths and topologies. The different investigated graph topologies are visualized in Figure 4. To ensure comparable stopping criteria for the sensitivity-based and ADMM algorithm, the absolute difference from the global optimal cost $\left|J(\boldsymbol{x}^*, \boldsymbol{u}^*) - \sum_{i\in\mathcal{V}} J_i^{[k]}(\cdot)\right| < \epsilon_{\mathrm{abs}}$ as well as the relative difference between two iterations $\left|\sum_{i\in\mathcal{V}}\left(J_i^{[k+1]}(\cdot) - J_i^{[k]}(\cdot)\right)\right| < \epsilon_{\mathrm{rel}}$ are used as a common stopping criterion. The optimal global cost is obtained by solving the central OCP (1). The thresholds are set to $\epsilon_{\mathrm{abs}} = 5 \cdot 10^{-2}$ and $\epsilon_{\mathrm{rel}} = 5 \cdot 10^{-3}$.

The result of varying coupling strength is plotted in Figure 5. While the required number of ADMM iterations scales with increasing coupling strength, the iterations of Algorithm 1 stay almost constant even in highly coupled systems. The same effect can be observed when moving from loose topologies to more coupled ones for which the required ADMM iterations strongly increase compared to the sensitivity-based algorithm iterations.

The reasoning behind these observations is that the ADMM algorithm is a consensus algorithm in which neighboring agents need to agree upon their choice of optimization variables, which gets increasingly difficult for highly coupled systems or a large number of neighbors. Similar observations have been made for DMPC of linear systems [32]. The sensitivity-based algorithm, however, does not suffer from this drawback as it directly considers the change

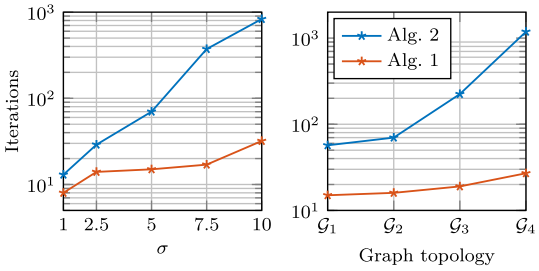

Figure 5: Number of iterations depending on the coupling strength σ for a chain topology $\mathcal{G}_1$ (left); number of iterations depending on the topology for a given $\sigma = 5$ (right).

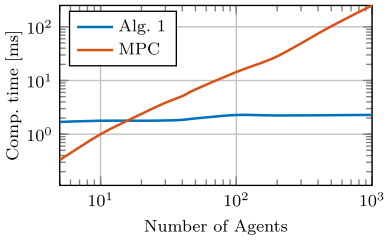

Figure 6: Scalability of the DMPC approach.

of the neighbors cost in the local cost functional and no consensus needs to be achieved.

The advantage of the distributed approach can be seen in Figure 6. It shows the average computation time per MPC time step with a constant number of ten sensitivity-based algorithm iterations compared to the central MPC approach. The agents are arranged in the ring topology $\mathcal{G}_2$ and the required computation time is averaged using the 40 measurements arising from simulating the system for 4 s. While the central MPC approach increases linearly with the number of agents, the computation time per agent is nearly independent of the system size for the sensitivity-based algorithm. This demonstrates the capability of the proposed algorithm to control high-scaled systems.

4.3 Nonconvex optimal control problems

The main advantage of the ADMM algorithm is its robustness since its augmented Lagrangian formulation is favorable for solving nonconvex optimization problems [33]. This is beneficial in optimal control as the general nonlinear OCP is nonconvex even for convex cost functions. Thus, the ability to cope with nonconvex optimal control problems is essential for nonlinear DMPC algorithms.

Algorithm 1 does not have this advantage, but the following example demonstrates that it is able to converge to a (local) stationary point even for nonconvex optimal control problems if suitable convexification terms are present. To this end, consider a system of agents arranged in a chain

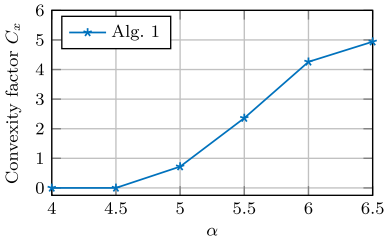

Figure 7: Influence of the factor α on the required minimum convexification factor C_x.

topology with the dynamic equations

$$\dot{x}_i = u_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \alpha x_i x_j \tag{37}$$

and parameter $\alpha > 0$. The same quadratic formulations as in (35) are utilized for the cost function which is characterized by the weights

$$Q_i = 1, \quad R_i = 0.1, \quad P_i = 1. \tag{38}$$

Furthermore, the horizon is set to $T = 0.75$ s with a sampling time of $\Delta t = 0.1$ s and the controls are constrained to $u_i \in [-1, 1]$. To show that the parameter α influences the convexity of the OCP, consider the central Hamiltonian

$$\begin{aligned} H = \frac{1}{2}\left(x_1^2 + x_2^2 + 0.1u_1^2 + 0.1u_2^2\right) \\ + \lambda_1\left(u_1 + \alpha x_1 x_2\right) + \lambda_2\left(u_2 + \alpha x_2 x_1\right) \end{aligned} \tag{39}$$

for the case of two agents. It is known that an OCP is convex if the Hamiltonian is jointly convex in the states $\boldsymbol{x}$ and controls $\boldsymbol{u}$, i.e $\partial_{\boldsymbol{uu}} H$ as well as $\partial_{\boldsymbol{xx}} H$ must be positive semi-definite [34]. Clearly, $\partial_{\boldsymbol{uu}} H$ is positive definite and the Hessian w.r.t. $\boldsymbol{x}$ is given by

$$\partial_{\boldsymbol{xx}} H = \begin{bmatrix} 1 & \alpha\lambda_1 \\ \alpha\lambda_2 & 1 \end{bmatrix}, \tag{40}$$

which is negative definite for $\frac{1}{\lambda_1\lambda_2} < \alpha^2$. This indicates that the problem is nonconvex for a sufficiently large α. In order to mitigate the effect of this nonconvex influence on the OCP, the convexification term $C_x(x_i - x_i^{[k]})^2$, compare to (13), is added to the integral cost function (35). A sufficiently high factor C_x for a given α can always be found such that $(1 + C_x)^2 > \alpha^2\lambda_1\lambda_2$ leading to strict convexity of the OCP. Figure 7 depicts this need of an increasing convexity factor C_x for a higher α for an example of ten agents ordered in a chain topology.

While it is possible to determine a sufficiently large C_x for this simple system, this is in general more difficult or even impossible for more complex systems. As this parameter is a design parameter, a suitable choice must be found

manually for each individual problem, which is a drawback of the sensitivity-based algorithm. The penalty parameter in the ADMM algorithm, which is also a tuning parameter, has a similar influence as the convexification parameter although various dynamic update heuristics exist to eliminate the design choice [8, 9]. The penalty parameter ρ is tuned more easily compared to the convexification factor as the ADMM algorithm still converges with a badly chosen penalty parameter ρ in this example while the sensitivity-based algorithm diverges.

4.4 Distributed hardware

Additionally, it is of great interest how the distributed DMPC algorithms perform on real distributed hardware with non-negligible communication as well as possible heterogeneous computation times. It is explored if the asynchronous versions of the sensitivity-based and ADMM algorithms indeed execute the steps of the algorithm faster due to skipped waiting times.

To investigate these aspects, the execution time is split into idle time, in which an agent does not perform any computation steps, and active time, in which the agent runs calculations but also for example prepares data for sending. Both times are measured for the four different algorithm variants. The same setup as in Section 4.2 is used but now evaluated on individual Raspberry Pi devices. The simulation is run for 25 algorithm steps.

The impact of the maximum tolerable delay $d_{\max}$ on the average idle time of the network is depicted in Figure 8. The idle time is already drastically reduced for $d_{\max} = 1$ when compared to the synchronous execution. Higher tolerable delays are hardly beneficial for reducing the idle time in this case and will only degrade the convergence further as more outdated data is used. Therefore, the maximum tolerable delay is set to $d_{\max} = 1$ for the following numerical evaluations.

The discrepancy between the idle time of the sensitivity-based and ADMM algorithms is explained by the fact that the ADMM algorithm requires three communication steps compared to one in the sensitivity-based algorithm and thus the agents need to wait longer on average for their relevant data. In the considered example, higher tolerable delays ($d_{\max} > 1$) do not further decrease the idle time as the heterogeneity in computation and communication time between the agents is rather low. For larger networks a higher $d_{\max}$ might be necessary. The required total time to perform the 25 algorithm iterations in the first MPC step and the relative cost function error compared to the central solution are shown in Table 1.

Table 1: Average total time and relative cost function error for 25 algorithm iterations in the first MPC time step for the chain (top) and ring (bottom) topology.

Algorithm	Alg. 1	Alg. 2	Alg. 3	Alg. 4
Total time	117.84 ms	185.60 ms	91.93 ms	99.60 ms
Rel. cost err.	0.01 %	0.13 %	0.34 %	4.16 %
Total time	124.77 ms	199.70 ms	100.16 ms	100.94 ms
Rel. cost err.	0.03 %	2.99 %	0.81 %	9.72 %

In the synchronous case, the sensitivity-based algorithm is about 36 % faster than the ADMM algorithm and has a smaller relative error with the same number of iterations. The asynchronous formulations reduce the required execution time for the chain topology by 22 %

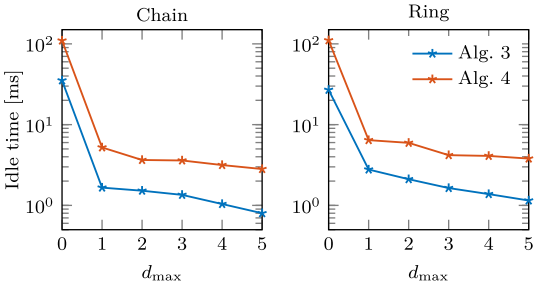

Figure 8: Influence of the maximum delay $d_{\max}$ on idle time.

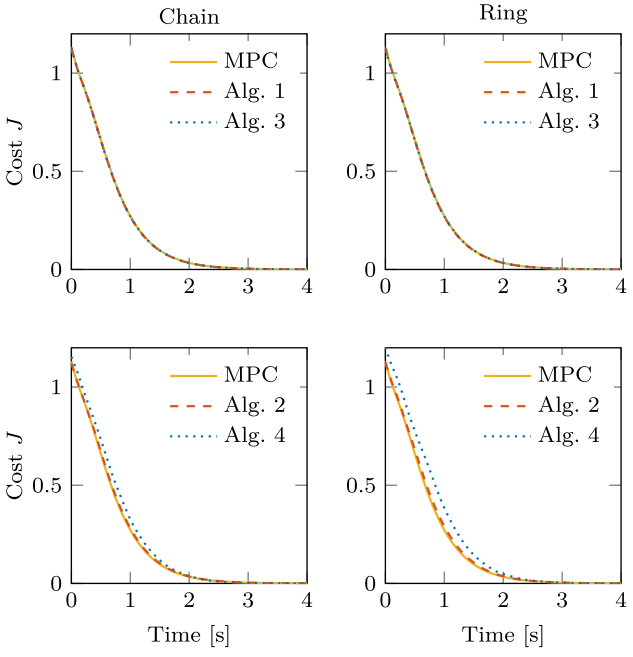

Figure 9: The asynchronous formulations produce a converging MPC scheme although slightly more sub-optimal than their synchronous counterparts.

and 46 %, respectively. Similar values of 20 % and 49 % are obtained for the ring topology showcasing that by running the same algorithm asynchronously, a significant execution time improvement can be expected.

Figure 9 shows the resulting cost trajectories with a minimal difference between synchronous and asynchronous versions. In the context of sub-optimal real-time capable MPC, the trade-off between less accuracy in the iterative solution per MPC time step for a faster computation time is favorable.

The difference in communication effort in the synchronous case is seen in Figure 10, in which the average communication time of 25 iterations is displayed. Since the sensitivity-based algorithm only needs one communication step compared to the three of the ADMM algorithm, its communication time is also about one third compared to the ADMM algorithm. The communication time increases only slightly for a more connected topology since more data are communicated through the network.

The distribution of the total needed time, from Table 1, in active and idle network time is displayed in Figure 11 in which can be observed that the asynchronous formulation indeed increases the active time of the agents, improving the high fraction of waiting times encountered in the synchronous formulations [9, 20]. As conjectured in Section 3.4, the a-ADMM algorithm is more effective compared to the asynchronous sensitivity-based algorithm in reducing the idle time since it involves more communication steps.

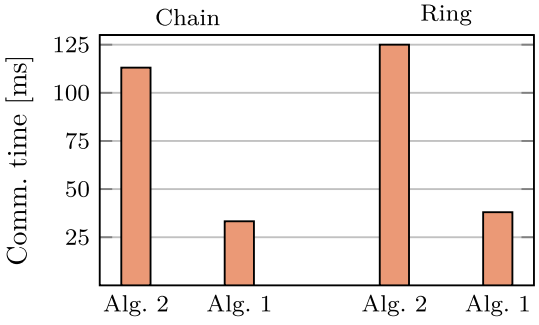

Figure 10: The network communication time is greatly reduced by the sensitivity-based algorithm compared to the ADMM algorithm.

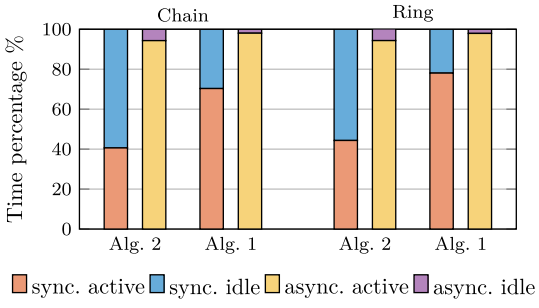

Figure 11: Average network active and idle time for the (a) synchronous versions of the sensitivity-based and ADMM algorithm.

5 Conclusions

This paper presented synchronous and asynchronous formulations of a sensitivity-based algorithm to solve nonlinear, constrained optimal control problems in a fully distributed fashion with only inter-agent communication. The algorithm is based on augmenting the cost function of an agent by the Taylor approximation of its neighbors' cost functional.

It is shown that if the algorithm is convergent, the optimality conditions of the distributed and the central OCP correspond. Furthermore, asynchronous formulations of the sensitivity-based and ADMM algorithm are presented to reduce the idle times of agents.

Numerical evaluations show that the sensitivity-based algorithm can cope better with highly coupled systems than the ADMM algorithm. For the same number of iterations, the sensitivity-based algorithm was up to 37 % faster than the ADMM algorithm in a distributed setting. The asynchronous formulation drastically reduced the idle time and overall execution time of the agents by up to 50 %. This trade-off between the accuracy of the solution and the computation time is favorable in the context of sub-optimal (D) MPC, where the computation must be finished within the sampling time.

Future research concerns proving the convergence of the proposed asynchronous sensitivity-based algorithm for nonlinear continuous-time OCPs in a general setting. The interpretation of the sensitivities as a Taylor approximation suggests that considering higher-order terms could be beneficial to the convergence behavior. Furthermore, switching to communication over the User Datagram Protocol (UDP) or other communication structures without explicit handling of the present packet loss provides an interesting prospect to further reduce communication time.

Research ethics: Not applicable.
Author contributions: The author(s) have (has) accepted responsibility for the entire content of this manuscript and approved itssubmission.
Competing interests: The author(s) state(s) no conflict of interest.
Research funding: This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under project grant number 3870/6-1.

Data availability: The raw data can be obtained upon reasonable request from the corresponding author.

Appendix

Algorithm 1 must be adapted if the states and controls in OCP (1) are subject to various constraints. At first, the case of pure box input constraints

$$\boldsymbol{u}_i \in [\boldsymbol{u}_{i,\min}, \boldsymbol{u}_{i,\max}], \tag{41}$$

which often occurs in MPC, is examined. While this constraint must be considered in the local solution of OCP (14), for example via the projected gradient method [31], the sensitivity-based algorithm does not need to be modified since the constraint does not depend on states and controls of any neighbors. However, the more general case in which OCP (1) is subject to local and coupled equality constraints

$$\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = 0 \tag{42a}$$

$$\boldsymbol{g}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j, \boldsymbol{u}_j) = 0,\ j \in \mathcal{N}_i^{\leftarrow} \tag{42b}$$

or local and coupled inequality constraints

$$\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq 0 \tag{43a}$$

$$\boldsymbol{h}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j, \boldsymbol{u}_j) \leq 0,\ j \in \mathcal{N}_i^{\leftarrow} \tag{43b}$$

requires further attention. As the local OCP (14) is now subject to coupled constraints (42b) and (43b), the calculation of the sensitivities must accommodate for the additional constraints as well.

To this end, equality and inequality constraints are adjoined to the Hamiltonian via the Lagrangian multipliers, $\boldsymbol{\mu}_{j,g} \in \mathbb{R}^{n_{g_j}}$, $\boldsymbol{\mu}_{j,h} \in \mathbb{R}^{n_{h_j}}$, $\boldsymbol{\mu}_{js,g} \in \mathbb{R}^{n_{g_{js}}}$, and $\boldsymbol{\mu}_{js,h} \in \mathbb{R}^{n_{h_{js}}}$, to form the extended Hamiltonian (compare [35], Section 11])

$$\hat{H}_j = \bar{H}_j + \boldsymbol{\mu}_{j,g}^{\mathrm{T}} \boldsymbol{g}_j + \boldsymbol{\mu}_{j,h}^{\mathrm{T}} \boldsymbol{h}_j + \sum_{s \in \mathcal{N}_j^{\leftarrow}} \boldsymbol{\mu}_{js,g}^{\mathrm{T}} \boldsymbol{g}_{js} + \boldsymbol{\mu}_{js,h}^{\mathrm{T}} \boldsymbol{h}_{js} \tag{44}$$

to calculate the extended partial derivatives

$$\partial_{\boldsymbol{u}_i} \hat{H}_j = \partial_{\boldsymbol{u}_i} \bar{H}_j + \left(\partial_{\boldsymbol{u}_i} \boldsymbol{g}_{ji}\right)^{\mathrm{T}} \boldsymbol{\mu}_{ji,g} + \left(\partial_{\boldsymbol{u}_i} \boldsymbol{h}_{ji}\right)^{\mathrm{T}} \boldsymbol{\mu}_{ji,h} \tag{45a}$$

$$\partial_{\boldsymbol{x}_i} \hat{H}_j = \partial_{\boldsymbol{x}_i} \bar{H}_j + \left(\partial_{\boldsymbol{x}_i} \boldsymbol{g}_{ji}\right)^{\mathrm{T}} \boldsymbol{\mu}_{ji,g} + \left(\partial_{\boldsymbol{x}_i} \boldsymbol{h}_{ji}\right)^{\mathrm{T}} \boldsymbol{\mu}_{ji,h}, \tag{45b}$$

which replace the respective partial derivatives in (12a) and (12b). Note that again only coupled constraints similar to coupled dynamics (1b) or costs (2) need to be regarded in the calculation of the sensitivities. For this purpose, the constraint functions must be sufficiently continuously differentiable w.r.t. their arguments on $t \in [0, T]$. The required Lagrangian multipliers in each algorithm iteration can be usually obtained from the solution of the local OCP (14) in which the constraints (42a) and (43b) are for example considered by an augmented Lagrangian framework and the multipliers are calculated via steepest ascent [31]. Note that Theorem 1 is not applicable in the same manner as the optimality conditions (16) have changed and it is not straightforward to show equality of the central and distributed optimality conditions under general constraints without any further assumptions.

References

[1] L. Grüne and J. Pannek, *Nonlinear Model Predictive Control: Theory and Algorithms*, London, Springer, 2011.

[2] S. V. Raković and W. S. Levine, *Handbook of Model Predictive Control*, Basel, Springer International Publishing, 2019.

[3] R. Findeisen, K. Graichen, and M. Mönnigmann, "Eingebettete Optimierung in der Regelungstechnik — Grundlagen und Herausforderungen," *Automatisierungstechnik*, vol. 66, no. 11, pp. 877—902, 2018.

[4] S. J. Qin and T. A. Badgwell, "An overview of nonlinear model predictive control applications," in *Nonlinear Model Predictive Control*, Basel, Springer International Publishing, 2000, pp. 369—392.

[5] E. Camponogara, D. Jia, B. H. Krogh, and S. Talukdar, "Distributed model predictive control," *IEEE Control Syst. Mag.*, vol. 22, no. 1, pp. 44—52, 2002.

[6] R. Negenborn and J. Maestre, "Distributed model predictive control: an overview and roadmap of future research opportunities," *IEEE Control Syst.*, vol. 34, no. 4, pp. 87—97, 2014.

[7] A. Bestler and K. Graichen, "Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM," *Optim. Control Appl. Methods*, vol. 40, no. 1, pp. 1—23, 2019.

[8] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1—122, 2010.

[9] D. Burk, A. Völz, and K. Graichen, "A modular framework for distributed model predictive control of nonlinear continuous-time systems (GRAMPC-D)," *Optim. Eng.*, vol. 23, pp. 771—795, 2021.

[10] P. Giselsson, M. D. Doan, T. Keviczky, B. De Schutter, and A. Rantzer, "Accelerated gradient methods and dual decomposition in distributed model predictive control," *Automatica*, vol. 49, no. 3, pp. 829—833, 2013.

[11] B. Houska and Y. Jiang, "Distributed optimization and control with ALADIN," in *Recent Advances in Model Predictive Control: Theory, Algorithms, and Applications*, Cham (Switzerland), Springer, 2021, pp. 135—163.

[12] G. Stomberg, A. Engelmann, and T. Faulwasser, "A compendium of optimization algorithms for distributed linear-quadratic MPC," *Automatisierungstechnik*, vol. 70, no. 4, pp. 317—330, 2022.

[13] H. Huber and K. Graichen, "A sensitivity-based distributed model predictive control algorithm for nonlinear continuous-time systems," in *Proc. of the Conference on Control Technology and Applications (CCTA)*, 2021, pp. 195—201.

[14] H. Scheu, J. Busch, and W. Marquardt, "Nonlinear distributed dynamic optimization based on first order sensitivities," in *Proc. of the American Control Conference (ACC)*, 2010, pp. 1574—1579.
[15] H. Scheu and W. Marquardt, "Sensitivity-based coordination in distributed model predictive control," *J. Process Control*, vol. 21, no. 5, pp. 715—728, 2011.
[16] G. Cohen, "On an algorithm of decentralized optimal control ," *J. Math. Anal. Appl.*, vol. 59, no. 2, pp. 242—259, 1977.
[17] I. Alvarado, D. Limon, D. M. De La Peña, et al, "A comparative analysis of distributed MPC techniques applied to the HD-MPC four-tank benchmark," *J. Process Control*, vol. 21, no. 5, pp. 800—815, 2011.
[18] T. Darure, V. Puig, J. Yamé, F. Hamelin, and Y. Wang, "Distributed model predictive control applied to a VAV based HVAC system based on sensitivity analysis," *IFAC-PapersOnLine*, vol. 51, no. 20, pp. 259—264, 2018.
[19] D. Krishnamoorthy and V. Kungurtsev, "A sensitivity assisted alternating directions method of multipliers for distributed optimization," in *Proc. of the Conference on Decision and Control (CDC)*, 2022, pp. 295—300.
[20] G. Stomberg, H. Ebel, and T. Faulwasser, "Cooperative distributed MPC via decentralized real-time optimization: Implementation results for robot formations," *Control Engineering Practice*, vol. 138, 2023.
[21] D. Burk, A. Völz, and K. Graichen, "Towards asynchronous ADMM for distributed model predictive control of nonlinear systems," in *Proc. of the European Control Conference (ECC)*, 2021, pp. 1957—1962.
[22] T.-H. Chang, M. Hong, W.-C. Liao, and X. Wang, "Asynchronous distributed ADMM for large-scale optimization—part I: algorithm and convergence analysis," *IEEE Trans. Signal Process.*, vol. 64, no. 12, pp. 3118—3130, 2016.
[23] A. Maffei, L. Iannelli, L. Glielmo, and F. Borrelli, "Asynchronous cooperative method for distributed model predictive control," in *Proc. of the Conference on Decision and Control (CDC)*, 2016, pp. 6946—6951.
[24] H. Wei, K. Zhang, and Y. Shi, "Self-triggered min—max DMPC for asynchronous multiagent systems with communication delays," *IEEE Trans. Ind. Inf.*, vol. 18, no. 10, pp. 6809—6817, 2021.
[25] B. Chachuat, "Nonlinear and dynamic optimization: from theory to practice," *Technical Report; Laboratoire d'Automatique, École Polytechnique Fédérale de Lausanne*, pp. 1—214, 2007.
[26] H. Huber, D. Burk, and K. Graichen, "Comparison of sensitivity-based and ADMM-based DMPC applied to building automation," in *Proc. of the Conference on Control Technology and Applications (CCTA)*, 2022, pp. 546—553.
[27] M. Pierer von Esch, A. Völz, and K. Graichen, "A fixed-point iteration scheme for sensitivity-based distributed optimal control," *IEEE Trans. Automat. Control*, 2023, Submitted for publication.
[28] S. Hentzelt, "A modular approach for distributed predictive control," Ph.D. thesis, Universität Ulm, Ulm, 2021.
[29] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Belmont, Athena Scientific, 2015.
[30] M. Pierer von Esch, A. Völz, and K. Graichen, "Asynchronous ADMM for nonlinear continuous-time systems," *Optim. Control Appl. Methods*, 2023, Submitted for publication.
[31] T. Englert, A. Völz, F. Mesmer, S. Rhein, and K. Graichen, "A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC)," *Optim. Eng.*, vol. 20, no. 3, pp. 769—809, 2019.
[32] G. Costantini, R. Rostami, and D. Görges, "Decomposition methods for distributed quadratic programming with application to distributed model predictive control," in *Proc. of the Allerton Conference on Communication, Control, and Computing*, 2018, pp. 943—950.
[33] R. T. Rockafellar, "Augmented Lagrange multiplier functions and duality in nonconvex programming," *SIAM J. Control*, vol. 12, no. 2, pp. 268—285, 1974.
[34] A. Seierstad and K. Sydsaeter, "Sufficient conditions in optimal control theory," *Int. Econ. Rev.*, vol. 18, no. 2, pp. 367—391, 1977.
[35] M. Papageorgiou, M. Leibold, and M. Buss, *Optimierung*, Berlin, Heidelberg, Springer, 2015.

Bionotes

Maximilian Pierer von Esch
Chair of Automatic Control,
Friedrich-Alexander-Universität
Erlangen-Nürnberg, 91058 Erlangen,
Germany
maximilian.v.pierer@fau.de

Maximilian Pierer von Esch is a research assistant at the Chair of Automatic Control at Friedrich-Alexander-Universität Erlangen-Nürnberg. His research focuses on distributed optimization, distributed nonlinear model predictive control, and communication aspects in distributed control.

Andreas Völz
Chair of Automatic Control,
Friedrich-Alexander-Universität
Erlangen-Nürnberg, 91058 Erlangen,
Germany
andreas.voelz@fau.de

Dr.-Ing. Andreas Völz is a lecturer at the Chair of Automatic Control at Friedrich-Alexander-Universität Erlangen-Nürnberg. His research focuses on local optimization methods for robotic applications, collision-free motion planning and nonlinear model predictive control.

Knut Graichen
Chair of Automatic Control,
Friedrich-Alexander-Universität
Erlangen-Nürnberg, 91058 Erlangen,
Germany
knut.graichen@fau.de

Prof. Dr.-Ing. Knut Graichen is head of the Chair of Automatic Control at Friedrich-Alexander-Universität Erlangen-Nürnberg. Main research areas: model predictive and distributed control as well as machine learning methods for control with applications in mechatronics, robotics, and energy systems. Knut Graichen is Editor-in-Chief of Control Engineering Practice.