Optimization and Engineering (2022) 23:771–795
https://doi.org/10.1007/s11081-021-09605-3

RESEARCH ARTICLE

A modular framework for distributed model predictive control of nonlinear continuous-time systems (GRAMPC-D)

Daniel Burk[1] · **Andreas Völz**[1] · **Knut Graichen**[1]

Received: 25 August 2020 / Revised: 15 February 2021 / Accepted: 16 February 2021 /
Published online: 10 March 2021

Abstract
The modular open-source framework GRAMPC-D for model predictive control of distributed systems is presented in this paper. The modular concept allows to solve optimal control problems in a centralized and distributed fashion using the same problem description. It is tailored to computational efficiency with the focus on embedded hardware. The distributed solution is based on the alternating direction method of multipliers and uses the concept of neighbor approximation to enhance convergence speed. The presented framework can be accessed through C++ and Python and also supports plug-and-play and data exchange between agents over a network.

Keywords Distributed model predictive control · Nonlinear model predictive control · Modular framework · Multi-agent systems

1 Introduction

Model predictive control (MPC) is a modern control concept that attained increasing attention during the last decades (Mayne et al. 2000; Allgöwer and Zheng 2012) as it is capable to handle nonlinear systems while considering constraints on both states and controls. It is based on solving an optimal control problem (OCP) on a finite horizon and applying the first part of the control trajectory to the actual plant, corresponding to the sampling time Δ_t of the controller. At the next sampling instant, the horizon is shifted and the OCP is solved again. This iterative scheme is executed repetitively to stabilize the plant on an infinite horizon.

A main difficulty is the computational complexity of solving the OCP in real-time, which in turn requires an efficient implementation of suitable MPC algorithms.

✉ Daniel Burk
daniel.burk@fau.de

[1] Lehrstuhl für Regelungstechnik, Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstraße 7, 91058 Erlangen, Germany

In the recent past, several toolboxes were published that provide adequate software frameworks such as ACADO (Houska et al. 2011) and ACADOS (Verschueren et al. 2019), VIATOC (Kalmari et al. 2015) or GRAMPC (Käpernick and Graichen 2014; Englert et al. 2019). In case of distributed systems with a high number of controls and states, the classic centralized approach is not capable of solving the overall OCP in real-time anymore. Hence, algorithms for distributed model predictive control (DMPC) (Camponogara et al. 2002; Maestre and Negenborn 2014) have been in the focus over the last years. Their basic idea is to decouple the centralized OCP and to split it into multiple local OCPs that can be solved in parallel. The expectation is to compensate the higher computational complexity due to the decoupled formulation as well as the increased communication effort by the parallel structure. There are multiple approaches to distributed algorithms for optimal control problems, such as sensitivity-based algorithms (Scheu and Marquardt 2011), the augmented Lagrangian based alternating direction inexact Newton method (ALADIN) (Houska et al. 2016, 2018) or the alternating direction method of multipliers (ADMM) (Boyd et al. 2011) that is also used in the presented framework.

The difficulty of an efficient implementation is drastically higher in case of DMPC than for classic MPC algorithms, as a potentially high number of subsystems, so-called *agents*, have to be managed. Several toolboxes for DMPC have been published as well. Linear discrete-time systems are considered in the DMPC-Toolbox (Gäfvert 2014) that is implemented in Matlab. The PnPMPC-TOOLBOX (Riverso et al. 2013) focuses on the plug-and-play functionality and provides an implementation in Matlab that considers continuous-time and discrete-time linear systems. Several algorithms are implemented in the Python-Toolbox DISROPT (Farina et al. 2019) regarding distributed optimization problems. ALADIN-α (Engelmann et al. 2020) is the most recent published toolbox that provides a Matlab implementation of the ALADIN algorithm. However, there is a lack of a DMPC framework that provides an implementation tailored to embedded hardware with the focus on real-time capable distributed model predictive control. Many real-world problems such as smart grids or cooperative robot applications are only equipped with weak hardware on the subsystem level that is not able to handle complex computation tasks in an appropriate time. Hence, for realizing distributed controllers on actual plants, an implementation optimized on execution time is required to enable real-time control. Furthermore, providing the possibility of communication between agents over a network is essential for a DMPC framework designed to control actual plants. The restriction to neighbor-to-neighbor-communication decouples the agents communication effort from the overall system size by bounding it to the cardinality of its neighborhood. The focus on real-world plants requires the system class to cover nonlinear dynamics including couplings between the agents in both dynamics and constraints.

The presented framework, in the following GRAMPC-D, provides an open-source C++implementation of the ADMM algorithm (Burk et al. 2019, 2020) that is capable of solving optimal control problems in a distributed manner with a per-agent computation-time in the millisecond range. The underlying minimization problems are solved with the MPC toolbox GRAMPC that is suitable for embedded hardware implementations. However, other toolboxes for solving the local minimization

problem can be used as well. To enable actual distributed optimization, a socket-based TCP communication is provided to allow agents to exchange data over a network. Furthermore, a Python interface is provided in addition to the C++-interface using the software module Pybind11 (Jakob et al. 2017). The Python interface combines both the functionality of Python and the performance of C++as it only wraps the C++interface while the actual code executing the DMPC algorithm is still running in C++. Furthermore, it allows for fast and efficient prototyping when developing a controller for a distributed system as both a centralized as well as a distributed controller can be derived based on the same problem description. The convergence behavior of distributed controllers can be improved by optionally using the concept of neighbor approximation. Thereby, the generated problem description of each agent is adapted to additionally approximate parts of its neighbors OCP and by this to improve the solution of its local OCP in each iteration, leading to an enhanced convergence behavior of the overall algorithm. The modular structure of GRAMPC-D enables modifying the overall system in the sense of plug-and-play by including or removing agents or couplings at run-time. Supporting plug-and-play features is a core functionality for a DMPC framework with focus on embedded systems, as the assumption of a static system description does not hold for a large number of real-world plants.

The paper is structured as follows. Section 2 outlines the considered class of coupled systems and OCP formulation. The DMPC framework GRAMPC-D is introduced in Sect. 3 including the ADMM algorithm as the method of choice for the algorithm. In addition, the concept of neighbor approximation is explained and the implemented algorithm for the crucial task of penalty parameter adaption is presented. The modular structure of the framework is presented in Sect. 4. Finally, simulation examples in Sect. 5 show the effectiveness and modularity of the DMPC framework, before conclusions are drawn in Sect. 6.

Throughout the paper, each vector $\vec{x} \in \mathbb{R}^n$ is written in bold style. Standard p-norms $\|\vec{x}\|_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}$ will be used as well as the weighted squared norm defined by $\|\vec{x}\|_{\vec{P}}^2 = \vec{x}^\mathrm{T} P \vec{x}$ with a positive (semi-)definite square matrix $\vec{P}$. The stacking of individual vectors $\vec{x}_i$, $i \in \mathcal{V}$ from a set $\mathcal{V}$ is denoted by $\vec{x} = \left[\vec{x}_i\right]_{i\in\mathcal{V}}$. As far as time trajectories are concerned, the explicit dependency on time t may be omitted to ease readability. The derivative with respect to time is written using the dot notation $\dot{\vec{x}}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\vec{x}(t)$.

2 Problem description

The presented DMPC framework considers multi-agent systems that can be described by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the sets of edges $\mathcal{E}$ and vertices $\mathcal{V}$. Each vertex represents an agent, while each edge between two vertices stands for a coupling between the corresponding agents. The couplings may be both uni- and bi-directional.

The considered optimal control problem for the coupled nonlinear system is given by

$$\min_{\vec{u}_i, i\in\mathcal{V}} \quad \sum_{i\in\mathcal{V}} J_i(\vec{x}_i, \vec{u}_i) \tag{1a}$$

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

$$\vec{x}_i(0) = \vec{x}_{i,0}, \quad i \in \mathcal{V} \tag{1c}$$

$$\vec{0} = \vec{g}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{1d}$$

$$\vec{0} = \vec{g}_{ij}(\vec{x}_i, \vec{u}_i, \vec{x}_j, \vec{u}_j, t), \quad j \in \mathcal{N}_i^{\leftarrow},\ i \in \mathcal{V} \tag{1e}$$

$$\vec{0} \geq \vec{h}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{1f}$$

$$\vec{0} \geq \vec{h}_{ij}(\vec{x}_i, \vec{u}_i, \vec{x}_j, \vec{u}_j, t), \quad j \in \mathcal{N}_i^{\leftarrow},\ i \in \mathcal{V} \tag{1g}$$

$$\vec{u}_i \in \left[\vec{u}_{i,\min},\ \vec{u}_{i,\max}\right], \quad i \in \mathcal{V} \tag{1h}$$

with

$$J_i(\vec{x}_i, \vec{u}_i) = V_i(\vec{x}_i(T), T) + \int_0^T l_i(\vec{x}_i, \vec{u}_i, t)\, \mathrm{d}t, \tag{2}$$

states $\vec{x}_i(t) \in \mathbb{R}^{n_{x,i}}$, controls $\vec{u}_i(t) \in \mathbb{R}^{n_{u,i}}$ and the horizon length $T > 0$. Each agent may have a general nonlinear cost function $l_i : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R} \to \mathbb{R}$ and terminal cost $V_i : \mathbb{R}^{n_{x,i}} \times \mathbb{R} \to \mathbb{R}$. The overall cost function (1a) is given by the sum over the individual cost functions. The subsystem dynamics (1b) are defined by the functions $\vec{f}_i : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R} \to \mathbb{R}^{n_{x,i}}$ and $\vec{f}_{ij} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{x,j}} \times \mathbb{R}^{n_{u,j}} \times \mathbb{R} \to \mathbb{R}^{n_{x,i}}$. The OCP additionally considers nonlinear equality constraints (1d)–(1e) and inequality constraints (1f)–(1g) with the functions $\vec{g}_i : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R} \to \mathbb{R}^{n_{g,i}}$, $\vec{g}_{ij} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{x,j}} \times \mathbb{R}^{n_{u,j}} \times \mathbb{R} \to \mathbb{R}^{n_{g,ij}}$ and $\vec{h}_i : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R} \to \mathbb{R}^{n_{h,i}}$, $\vec{h}_{ij} : \mathbb{R}^{n_{x,i}} \times \mathbb{R}^{n_{u,i}} \times \mathbb{R}^{n_{x,j}} \times \mathbb{R}^{n_{u,j}} \times \mathbb{R} \to \mathbb{R}^{n_{h,ij}}$ as well as box constraints (1h) for the control input $\vec{u}_i$ of each agent $i \in \mathcal{V}$.

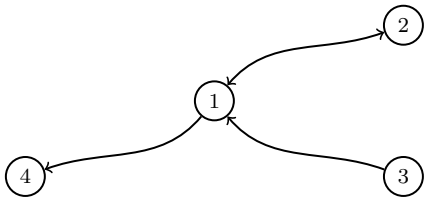

Fig. 1 The neighborhood $\mathcal{N}_1 = \{2, 3, 4\}$ of agent 1 is composed of sending neighbors $\mathcal{N}_1^{\leftarrow} = \{2, 3\}$ and receiving neighbors $\mathcal{N}_1^{\rightarrow} = \{2, 4\}$

The neighborhood $\mathcal{N}_i$ of agent $i \in \mathcal{V}$ is given by two sets that differ in the direction of the coupling, *sending neighbors* $\mathcal{N}_i^{\leftarrow}$ and *receiving neighbors* $\mathcal{N}_i^{\rightarrow}$, see Fig. 1 for an example. States and controls of sending neighbors have an explicit influence on the dynamics of the agent i in form of functions $\vec{f}_{ij}$, see (1b). Receiving neighbors are neighbors of agent i that are explicitly influenced by this agent, hence states or controls of the agent are part of a function $\vec{f}_{ij}$ of receiving neighbors. While a neighbor can be both, receiving and sending, this separation is going to be beneficial in the ADMM algorithm by reducing unnecessary computation and communication effort.

The dynamics (1b) of each agent $i \in \mathcal{V}$ are *neighbor affine* in the sense that the dynamics consists of a function $\vec{f}_i$ that depend only on states and controls of the agent and a sum of functions $\vec{f}_{ij}$ that depend on states and controls of the agent and one neighbor. The constraints (1d)–(1h) on each agent are separated into constraints that depend on states and controls of the agent, given by $\vec{g}_i$, $\vec{h}_i$ and the box constraints (1h), and constraints $\vec{g}_{ij}$ and $\vec{h}_{ij}$ depending on states and controls of the agent and one neighbor, similar to the dynamics.

The considered OCP formulation (1) covers a wide class of distributed systems, e.g. cooperative transport (Hentzelt and Graichen 2013) and scalable systems such as smart grids (Filatrella et al. 2008). This generic system description combined with the focus on a time-efficient implementation opens a wide spectrum of usability for the presented DMPC framework.

3 Distributed model predictive control

Optimal control problems for coupled systems as in (1) contain a large number of states and controls. This leads to a significant computational effort that is challenging for standard MPC algorithms to be handled in real-time. DMPC algorithms instead assume that each of the distributed subsystems are equipped with a dedicated control unit that is capable of solving a reduced optimal control problem. The idea based on this assumption is to decouple the global OCP and spread the computation effort over the set of agents in parallel. Overlying algorithms ensure convergence of the local solutions to an optimal solution for the overall system. While the computational complexity and communication effort of algorithms for DMPC is higher than solving the central problem, the expectation is to compensate this disadvantage by the parallel structure. In the presented DMPC framework, the well-known ADMM algorithm (Boyd et al. 2011) is employed in a continuous-time setting (Bestler and Graichen 2019). Note that the formulation of the algorithm is based on previous work (Burk et al. 2019, 2020).

3.1 ADMM algorithm

The ADMM algorithm enables to spread the computation effort of the global OCP (1) completely on distributed agents. As a starting point, the global OCP (1) is brought into a decoupled form for each agent $i \in \mathcal{V}$ by introducing local copies $\vec{\tilde{x}}_{ji}(t) \in \mathbb{R}^{n_{x,j}}$ and $\vec{\tilde{u}}_{ji}(t) \in \mathbb{R}^{n_{u,j}}$ for the states $\vec{x}_j$ and controls $\vec{u}_j$ of each sending neighbor $j \in \mathcal{N}_i^{\leftarrow}$, i.e.

$$\min_{\vec{w},\vec{z}} \quad \sum_{i\in\mathcal{V}} J_i(\vec{x}_i, \vec{u}_i) \tag{3a}$$

$$\text{s.t.} \quad \vec{x}_i = \vec{f}_i(\vec{x}_i, \vec{u}_i, t) + \sum_{j\in\mathcal{N}_i^{\leftarrow}} \vec{f}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad i \in \mathcal{V} \tag{3b}$$

$$\vec{x}_i(0) = \vec{x}_{i,0}, \ i \in \mathcal{V} \tag{3c}$$

$$\vec{0} = \vec{g}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{3d}$$

$$\vec{0} \geq \vec{h}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{3e}$$

$$\vec{0} = \vec{g}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V} \tag{3f}$$

$$\vec{0} \geq \vec{h}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V} \tag{3g}$$

$$\vec{u}_i \in \left[\vec{u}_{i,\min}, \ \vec{u}_{i,\max}\right], \quad i \in \mathcal{V} \tag{3h}$$

$$\vec{0} = \begin{bmatrix} \vec{z}_{x,i} \\ \vec{z}_{u,i} \end{bmatrix} - \begin{bmatrix} \vec{x}_i \\ \vec{u}_i \end{bmatrix}, \quad i \in \mathcal{V} \tag{3i}$$

$$\vec{0} = \begin{bmatrix} \vec{z}_{x,j} \\ \vec{z}_{u,j} \end{bmatrix} - \begin{bmatrix} \vec{\tilde{x}}_{ji} \\ \vec{\tilde{u}}_{ji} \end{bmatrix}, \quad j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V}. \tag{3j}$$

These local copies $(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji})$ represent new control inputs for the agent i and can be seen as a proposal of agent i for its neighbors $j \in \mathcal{N}_i^{\leftarrow}$. Equivalence between the local copies and the original variables is ensured by introducing the consistency

constraints (3i) and (3j) with the coupling variables $\vec{z}_{x,i}(t) \in \mathbb{R}^{n_{x,i}}$ and $\vec{z}_{u,i}(t) \in \mathbb{R}^{n_{u,i}}$. In (3) and the following, the notation

$$\vec{w}_i = \begin{bmatrix} \vec{u}_i \\ \begin{bmatrix} \vec{x}_{ji} \\ \vec{u}_{ji} \end{bmatrix}_{j \in \mathcal{N}_i^{\leftarrow}} \end{bmatrix}, \quad \vec{z}_i = \begin{bmatrix} \vec{z}_{x,i} \\ \vec{z}_{u,i} \end{bmatrix}, \quad \vec{z}_{-i} = \begin{bmatrix} \vec{z}_{x,j} \\ \vec{z}_{u,j} \end{bmatrix}_{j \in \mathcal{N}_i^{\leftarrow}}, \quad i \in \mathcal{V} \tag{4a}$$

$$\vec{w} = \left[\vec{w}_i\right]_{i \in \mathcal{V}}, \quad \vec{z} = \left[\vec{z}_i\right]_{i \in \mathcal{V}} \tag{4b}$$

is used.

The ADMM method is based on the Augmented Lagrangian formulation (Bestler and Graichen 2019; Bertsekas 1996). Regarding the continuous-time setting used in this paper, the consistency constraints (3i) and (3j) are accounted for in the cost functional

$$\begin{aligned} &J_{\rho,i}(\vec{x}_i, \vec{w}_i, \vec{\mu}_i, \vec{z}_i, \vec{z}_{-i}) \\ &= J_i(\vec{x}_i, \vec{u}_i) \\ &\quad + \int_0^T \begin{bmatrix} \vec{\mu}_{x,ii} \\ \vec{\mu}_{u,ii} \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} \vec{z}_{x,i} \\ \vec{z}_{u,i} \end{bmatrix} - \begin{bmatrix} \vec{x}_i \\ \vec{u}_i \end{bmatrix} \right) + \frac{1}{2} \left\| \begin{bmatrix} \vec{z}_{x,i} \\ \vec{z}_{u,i} \end{bmatrix} - \begin{bmatrix} \vec{x}_i \\ \vec{u}_i \end{bmatrix} \right\|_{\vec{C}_i}^2 \\ &\quad + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \begin{bmatrix} \vec{\mu}_{x,ji} \\ \vec{\mu}_{u,ji} \end{bmatrix}^{\mathsf{T}} \left(\begin{bmatrix} \vec{z}_{x,j} \\ \vec{z}_{u,j} \end{bmatrix} - \begin{bmatrix} \vec{x}_{ji} \\ \vec{u}_{ji} \end{bmatrix} \right) + \frac{1}{2} \left\| \begin{bmatrix} \vec{z}_{x,j} \\ \vec{z}_{u,j} \end{bmatrix} - \begin{bmatrix} \vec{x}_{ji} \\ \vec{u}_{ji} \end{bmatrix} \right\|_{\vec{C}_{ji}}^2 \mathrm{d}t \end{aligned} \tag{5}$$

subject to (3b)–(3h) with the Lagrange multipliers $\vec{\mu}_{x,ii}(t) \in \mathbb{R}^{n_{x,i}}$, $\vec{\mu}_{u,ii}(t) \in \mathbb{R}^{n_{u,i}}$, $\vec{\mu}_{x,ji}(t) \in \mathbb{R}^{n_{x,j}}$, $\vec{\mu}_{u,ji}(t) \in \mathbb{R}^{n_{u,j}}$ and penalty parameters $\vec{\rho}_{x,i}(t) \in \mathbb{R}^{n_{x,i}}$, $\vec{\rho}_{u,i}(t) \in \mathbb{R}^{n_{u,i}}$, $\vec{\rho}_{x,ji}(t) \in \mathbb{R}^{n_{x,j}}$ and $\vec{\rho}_{u,ji}(t) \in \mathbb{R}^{n_{u,j}}$. To ease notations, the multipliers and penalty parameters are stacked according to

$$\vec{\mu}_i = \begin{bmatrix} \vec{\mu}_{x,ii} \\ \vec{\mu}_{u,ii} \\ \begin{bmatrix} \vec{\mu}_{x,ji} \\ \vec{\mu}_{u,ji} \end{bmatrix}_{j \in \mathcal{N}_i^{\leftarrow}} \end{bmatrix}, \ i \in \mathcal{V}, \qquad \vec{\mu} = \left[\vec{\mu}_i\right]_{i \in \mathcal{V}} \tag{6a}$$

$$\vec{C}_i = \operatorname{diag} \begin{bmatrix} \vec{\rho}_{x,i} \\ \vec{\rho}_{u,i} \end{bmatrix}, \ i \in \mathcal{V} \qquad \vec{C}_{ji} = \operatorname{diag} \begin{bmatrix} \vec{\rho}_{x,ji} \\ \vec{\rho}_{u,ji} \end{bmatrix}, \ j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V}. \tag{6b}$$

The corresponding dual problem to (3) can be written as

$$\max_{\vec{\mu}} \min_{\vec{w},\vec{z}} \quad \sum_{i\in\mathcal{V}} J_{\rho,i}(\vec{x}_i, \vec{w}_i, \vec{\mu}_i, \vec{z}_i, \vec{z}_{\text{-}i}) \tag{7a}$$

$$\text{s.t. } \vec{\dot{x}}_i = \vec{f}_i(\vec{x}_i, \vec{u}_i, \tau) + \sum_{j\in\mathcal{N}_i^{\leftarrow}} \vec{f}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \ i \in \mathcal{V} \tag{7b}$$

$$\vec{x}_i(0) = \vec{x}_{i,0}, \quad i \in \mathcal{V} \tag{7c}$$

$$\vec{0} = \vec{g}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{7d}$$

$$\vec{0} \geq \vec{h}_i(\vec{x}_i, \vec{u}_i, t), \quad i \in \mathcal{V} \tag{7e}$$

$$\vec{0} = \vec{g}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V} \tag{7f}$$

$$\vec{0} \geq \vec{h}_{ij}(\vec{x}_i, \vec{u}_i, \vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow}, \ i \in \mathcal{V} \tag{7g}$$

$$\vec{u}_i \in \left[\vec{u}_{i,\min}, \ \vec{u}_{i,\max}\right], \quad i \in \mathcal{V} \tag{7h}$$

with the primal variables $(\vec{w}, \vec{z})$ and the dual variables $\vec{\mu}$. The ADMM algorithm solves the max–min-problem (7) by repetitively executing the three steps

$$\min_{\vec{w}} \quad \sum_{i\in\mathcal{V}} J_{\rho,i}(\vec{x}_i, \vec{w}_i, \vec{\mu}_i^{q-1}, \vec{z}_i^{q-1}, \vec{z}_{\text{-}i}^{q-1}), \quad \text{s.t.}(7b) - (7h) \tag{8a}$$

$$\min_{\vec{z}} \quad \sum_{i\in\mathcal{V}} J_{\rho,i}(\vec{u}_i^{q}, \vec{\mu}_i^{q-1}, \vec{z}_i, \vec{z}_{\text{-}i}; \vec{x}_{i,0}) \tag{8b}$$

$$\vec{\mu}_i^{q} = \vec{\mu}_i^{q-1} + \operatorname{diag}\left(\begin{bmatrix} \vec{C}_i \\ \left[\vec{C}_{ji}\right]_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix}\right) \begin{bmatrix} \vec{z}_i^{q} - \begin{bmatrix} \vec{x}_i^{q} \\ \vec{u}_i^{q} \end{bmatrix} \\ \left[\vec{z}_j^{q} - \begin{bmatrix} \vec{\tilde{x}}_{ji}^{q} \\ \vec{\tilde{u}}_{ji}^{q} \end{bmatrix}\right]_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix}, \quad i \in \mathcal{V} \tag{8c}$$

with the iteration counter q. The minimization with respect to the coupling variables $\vec{z}$ (8ab) can be solved analytically while the steepest ascent is used in (8ac). Important to note is that each step can be subdivided into fully decoupled steps for either agent $i \in \mathcal{V}$. Hence, the algorithm is fully distributable which allows to spread the computation effort over all agents.

Algorithm 1 Alternating direction method of multipliers

Initialize $\boldsymbol{w}_i^0$, $\boldsymbol{z}_i^0$, $\boldsymbol{\mu}_i^0$, choose $\boldsymbol{C}_i$, $\boldsymbol{C}_{ji}$, ϵ, set $q = 1$

1: Compute local variables $\boldsymbol{w}_i^q$ by solving

$$\min_{\boldsymbol{w}_i} \; V_i(\boldsymbol{x}_i(T), T) + \int_0^T l_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t) + \begin{bmatrix} \boldsymbol{\mu}_{x,ii}^{q-1} \\ \boldsymbol{\mu}_{u,ii}^{q-1} \end{bmatrix}^\mathsf{T} \left(\boldsymbol{z}_i^{q-1} - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} \right) + \frac{1}{2} \left\| \boldsymbol{z}_i^{q-1} - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} \right\|_{\boldsymbol{C}_i}^2 \tag{9a}$$
$$+ \sum_{j \in \mathcal{N}_i^{\leftarrow}} \begin{bmatrix} \boldsymbol{\mu}_{x,ji}^{q-1} \\ \boldsymbol{\mu}_{u,ji}^{q-1} \end{bmatrix}^\mathsf{T} \left(\boldsymbol{z}_j^{q-1} - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix} \right) + \frac{1}{2} \left\| \boldsymbol{z}_j^{q-1} - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix} \right\|_{\boldsymbol{C}_{ji}}^2 \mathrm{d}t$$

$$\text{s.t. } \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t) + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}, t), \quad \boldsymbol{x}_i(0) = \boldsymbol{x}_{i,0} \tag{9b}$$

$$\boldsymbol{0} = \boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t), \; \boldsymbol{0} = \boldsymbol{g}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow} \tag{9c}$$

$$\boldsymbol{0} \geq \boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i, t), \; \boldsymbol{0} \geq \boldsymbol{h}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}, t), \quad j \in \mathcal{N}_i^{\leftarrow} \tag{9d}$$

$$\boldsymbol{u}_i \in \left[\boldsymbol{u}_{i,\min}, \; \boldsymbol{u}_{i,\max} \right] \tag{9e}$$

2: Send local copies $\bar{\boldsymbol{x}}_{ji}^q$ and $\bar{\boldsymbol{u}}_{ji}^q$ to sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$

3: Compute coupling variables

$$\boldsymbol{z}_i^q = \frac{1}{1 + |\mathcal{N}_i^{\rightarrow}|} \left(\begin{bmatrix} \boldsymbol{x}_i^q \\ \boldsymbol{u}_i^q \end{bmatrix} - \boldsymbol{C}_i^{-1} \begin{bmatrix} \boldsymbol{\mu}_{x,ii}^{q-1} \\ \boldsymbol{\mu}_{u,ii}^{q-1} \end{bmatrix} + \sum_{j \in \mathcal{N}_i^{\rightarrow}} \begin{bmatrix} \bar{\boldsymbol{x}}_{ij}^q \\ \bar{\boldsymbol{u}}_{ij}^q \end{bmatrix} - \boldsymbol{C}_{ij}^{-1} \begin{bmatrix} \boldsymbol{\mu}_{x,ij}^{q-1} \\ \boldsymbol{\mu}_{u,ij}^{q-1} \end{bmatrix} \right) \tag{10}$$

4: Send coupling variables $\boldsymbol{z}_i^q$ to receiving neighbors $j \in \mathcal{N}_i^{\rightarrow}$

5: Compute Lagrange multipliers

$$\boldsymbol{\mu}_i^q = \boldsymbol{\mu}_i^{q-1} + \operatorname{diag}\left[\boldsymbol{C}_i, \boldsymbol{C}_{ji}\right] \begin{bmatrix} \boldsymbol{z}_i^q - \begin{bmatrix} \boldsymbol{x}_i^q \\ \boldsymbol{u}_i^q \end{bmatrix} \\ \left[\boldsymbol{z}_j^q - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji}^q \\ \bar{\boldsymbol{u}}_{ji}^q \end{bmatrix} \right]_{j \in \mathcal{N}_i \leftarrow} \end{bmatrix} \tag{11}$$

6: Send Lagrange multipliers $\boldsymbol{\mu}_i^q$ to sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$

7: **if** if $q = q_{\max}$ or $\left\| \begin{bmatrix} \boldsymbol{z}_i^q - \boldsymbol{z}_i^{q-1} \\ \boldsymbol{\mu}_i^q - \boldsymbol{\mu}_i^{q-1} \end{bmatrix} \right\| \leq \epsilon, \; \forall \; i \in \mathcal{V}$ **then**

8: STOP

9: **else**

10: set $q = q + 1$ and go to Step 1.

11: **end if**

The resulting ADMM algorithm for each agent is given in Algorithm 1. It consists of the computation steps 1, 3, 5, the communication steps 2, 4, 6, and the evaluation of a convergence criterion in Step 7. The algorithm starts with an initialization of corresponding variables. The local OCP (9) is minimized in Step 1 with respect to the local variables $\vec{w}_i$. This minimization represents the main computation effort of the overall algorithm. In Step 2, the trajectories of the local variables are sent to the sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$ of each agent $i \in \mathcal{V}$. The analytic solution for the minimization with respect to the coupling variables (8ab) is given in Step 3 of the ADMM algorithm, before they are sent to the receiving neighbors $j \in \mathcal{N}_i^{\rightarrow}$ of each

agent in Step 4. The third computation step is given in Step 5 by a maximization with respect to the Lagrange multipliers $\vec{\mu}$. In Step 6, the result of the maximization step is sent to the sending neighbors $j \in \mathcal{N}_i^{\leftarrow}$ of each agent $i \in \mathcal{V}$. A convergence criterion is checked in Step 7. If it is satisfied or the iteration counter has reached its maximum, the algorithm stops and returns the current trajectories. Otherwise, the iteration counter is increased and the algorithm returns to Step 1.

3.2 Neighbor approximation

In practice, the convergence speed of the ADMM algorithm can be enhanced by anticipating the actions of the neighbors in the own agents optimization. The concept of neighbor approximation was introduced in Hentzelt and Graichen (2013) and extended in Burk et al. (2020) and relies on the neighbor affine structure of the dynamics (1b)–(1c) and constraints (1d)–(1h).

The basic idea is to use the already introduced local copies $\vec{\tilde{x}}_{ji}$ and $\vec{\tilde{u}}_{ji}$ to approximate parts of the neighbors OCP. The expectation is that the additional information about the neighborhood improves the local solution of each agent and thus the convergence behavior of the overall algorithm. This also reduces the number of required ADMM iterations until convergence is reached, which has been confirmed in numerical evaluations in Hentzelt and Graichen (2013) and Burk et al. (2020). In practical experience, the reduced number of ADMM iterations can compensate for the increased complexity of the extended OCP which can lead to a significantly decreased computational effort (Burk et al. 2020).

The neighbor approximation implemented in GRAMPC-D is modular in the sense that the neighbors cost, constraints, dynamics and each combination of the three can be considered.

3.2.1 Neighbor cost

The global cost to be mininized (2) consists of the single cost functions of the agents $i \in \mathcal{V}$. The local copies of the neighbor variables $\vec{\tilde{x}}_{ji}$ and $\vec{\tilde{u}}_{ji}$, $j \in \mathcal{N}_i$, can be used to anticipate the neighbors cost $J_j(\vec{\tilde{u}}_{ji}; \vec{x}_{j,0})$ on the local level of agent $i \in \mathcal{V}$, i.e.

$$\tilde{J}_i(\vec{x}_i, \vec{w}_i) = \eta_i J_i(\vec{x}_i, \vec{u}_i) + \sum_{j \in \mathcal{N}_i} \eta_j J_j(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}), \quad i \in \mathcal{V}. \tag{12}$$

The normalization with the factors

$$\eta_i = \frac{1}{1 + |\mathcal{N}_i|}, \quad i \in \mathcal{V} \tag{13}$$

is necessary in order to avoid that the neighbors cost function would appear in the overall cost function multiple times. Approximating the neighbor costs is especially beneficial in examples with a strong dependency on the other agents costs and enables the agent to anticipate the neighbors control action to minimize its local costs.

It is recommended to combine the neighbor cost approximation with the approximation of the neighbor dynamics introduced in the following lines.

3.2.2 Neighbor dynamics

Similar to the neighbor cost consideration, the neighbor affine structure of the dynamics (1b)–(1c) can be exploited to approximate the neighbor dynamics and therefore to improve the quality of the local copies $\vec{\tilde{x}}_{ji}$ and $\vec{\tilde{u}}_{ji}$. To this end, the local dynamics (1b)–(1c) are extended by the approximate neighbor dynamics

$$\dot{\vec{\tilde{x}}}_{ji} = \vec{f}_j(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t) + \vec{f}_{ji}(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, \vec{x}_i, \vec{u}_i, t) + \vec{\tilde{v}}_{ji}, \quad j \in \mathcal{N}_i,\ i \in \mathcal{V} \tag{14}$$

with the initial condition $\vec{\tilde{x}}_{ji}(0) = \vec{x}_{j,0}$. The dependencies of the neighbor's states and controls in $\vec{f}_j$ and $\vec{f}_{ji}$ are decoupled using the local copies $\vec{\tilde{x}}_{ji}$ and $\vec{\tilde{u}}_{ji}$. However, it is not possible to decouple further functions $\vec{f}_{js}$ as these depend on states and controls of agents s for which agent i has in general no local copies. For consistency, the *external influence*

$$\vec{v}_{ij} = \sum_{s \in \mathcal{N}_i^+ \setminus \{j\}} \vec{f}_{is}(\vec{x}_i, \vec{u}_i, \vec{x}_s, \vec{u}_s, t), \quad j \in \mathcal{N}_i,\ i \in \mathcal{V} \tag{15}$$

is introduced with $\vec{v}_{ij}(t) \in \mathbb{R}^{n_{x,i}}$ that captures the remaining terms of the neighbors dynamics. The external influence is considered in the approximated neighbor dynamics (14) by introducing local copies $\vec{\tilde{v}}_{ji}(t) \in \mathbb{R}^{n_{x,j}}$. Thereby, the whole dynamics of neighbor j is approximated in (14). To ensure convergence of the local copies $\vec{\tilde{v}}_{ji}$ to the original variables $\vec{v}_{ij}$, the consistency constraints

$$\vec{z}_{v,ij} = \vec{v}_{ij}, \quad j \in \mathcal{N}_i,\ i \in \mathcal{V} \tag{16a}$$

$$\vec{z}_{v,ji} = \vec{\tilde{v}}_{ji}, \quad j \in \mathcal{N}_i,\ i \in \mathcal{V} \tag{16b}$$

are introduced and replace the consistency constraints in (3i)–(3j) regarding the states. Note that the local copies of the states $\vec{\tilde{x}}_{ji}$ are not considered as control variables anymore, but are determined by the differential equation (14). Instead, the local copies of the external influence $\vec{\tilde{v}}_{ji}$ serve as new local control variables. In summary, the stacked notations (4) and (6) are adapted according to

$$\vec{w}_i = \begin{bmatrix} \vec{u}_i \\ \begin{bmatrix} \vec{\tilde{u}}_{ji} \\ \vec{\tilde{v}}_{ji} \end{bmatrix}_{j \in \mathcal{N}_i} \end{bmatrix},\ i \in \mathcal{V} \quad \vec{z}_i = \begin{bmatrix} \vec{z}_{u,i} \\ \left[\vec{z}_{v,ij}\right]_{j \in \mathcal{N}_i} \end{bmatrix},\ i \in \mathcal{V} \tag{17a}$$

$$\vec{\mu}_i = \begin{bmatrix} \vec{\mu}_{u,ii} \\ \begin{bmatrix} \vec{\mu}_{v,ij} \\ \vec{\mu}_{u,ji} \\ \vec{\mu}_{v,ji} \end{bmatrix}_{j \in \mathcal{N}_i} \end{bmatrix}, \ i \in \mathcal{V} \quad \vec{z}_{\text{-}i} = \begin{bmatrix} \vec{z}_{u,j} \\ \vec{z}_{v,ji} \end{bmatrix}_{j \in \mathcal{N}_i}, \ i \in \mathcal{V} \tag{17b}$$

$$\vec{C}_i = \operatorname{diag}\left[\vec{\rho}_{u,i}\right], \ i \in \mathcal{V} \quad \vec{C}_{ji} = \operatorname{diag}\begin{bmatrix} \vec{\rho}_{v,ij} \\ \vec{\rho}_{u,ji} \\ \vec{\rho}_{v,ji} \end{bmatrix}, \ j \in \mathcal{N}_i, \ i \in \mathcal{V} \tag{17c}$$

and

$$\vec{w} = \left[\vec{w}_i\right]_{i \in \mathcal{V}}, \quad \vec{z} = \left[\vec{z}_i\right]_{i \in \mathcal{V}}, \quad \vec{\mu} = \left[\vec{\mu}_i\right]_{i \in \mathcal{V}} \tag{18}$$

with Lagrangian multipliers $\vec{\mu}_{v,ij}(t) \in \mathbb{R}^{n_{x,i}}$, $\vec{\mu}_{v,ji}(t) \in \mathbb{R}^{n_{x,j}}$, coupling variables $\vec{z}_{v,ij}(t) \in \mathbb{R}^{n_{x,i}}$, and penalty parameters $\vec{\rho}_{v,ij}(t) \in \mathbb{R}^{n_{x,i}}$, $\vec{\rho}_{v,ji}(t) \in \mathbb{R}^{n_{x,j}}$.

3.2.3 Neighbor constraints

In addition to the consideration of the neighbor cost and dynamics within the local OCP of agent $i \in \mathcal{V}$, the constraints (1d)–(1h) of each neighbor $j \in \mathcal{N}_i$ of agent $i \in \mathcal{V}$ can be taken into account by adding

$$\vec{0} = \vec{g}_j(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad \vec{0} = \vec{g}_{ij}(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, \vec{x}_i, \vec{u}_i, t), \quad j \in \mathcal{N}_i, \ i \in \mathcal{V} \tag{19a}$$

$$\vec{0} \geq \vec{h}_i(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, t), \quad \vec{0} \geq \vec{h}_{ij}(\vec{\tilde{x}}_{ji}, \vec{\tilde{u}}_{ji}, \vec{x}_i, \vec{u}_i, t), \quad j \in \mathcal{N}_i, \ i \in \mathcal{V} \tag{19b}$$

$$\vec{\tilde{u}}_{ji} \in \left[\vec{u}_{j,\min}, \ \vec{u}_{j,\max}\right], \quad j \in \mathcal{N}_i, \ i \in \mathcal{V} \tag{19c}$$

to the local OCP (9). Again, the constraints are decoupled from the neighbors states and controls $\vec{x}_j$ and $\vec{u}_i$ by using the local copies $\vec{\tilde{x}}_{ji}$ and $\vec{\tilde{u}}_{ji}$.

As discussed before, this concept is restricted to the agent constraints of each neighbor $j \in \mathcal{N}_i$ and the coupling constraints between neighbors j and agent i, while further coupling constraints between neighbor j and its neighbors $s \in \mathcal{N}_j^{\leftarrow} \setminus \{i\}$ depend on states and controls of agents s for which in general agent i has no local copies.

3.3 Penalty parameter adaption

The update of the penalty parameters in the matrices $\vec{C}_i$ and $\vec{C}_{ji}$ in (7) is crucial for a fast convergence of the ADMM algorithm. The adaptation method implemented in GRAMPC-D follows a proposal in [Boyd et al. (2011), Section 3.4.1] for the optimization problem

$$\min_{\vec{x},\vec{z}} \quad f(\vec{x}) + g(\vec{z}) \tag{20a}$$

$$\text{s.t.} \quad \vec{A}\vec{x} + \vec{B}\vec{z} = \vec{c}\,. \tag{20b}$$

The proposed adaption algorithm is given by

$$\rho^q = \begin{cases} \tau^{\text{incr}}\rho^{q-1} & \text{if } \left\|\vec{r}^{q-1}\right\|_2 > \mu\left\|\vec{s}^{q-1}\right\|_2 \\ \frac{\rho^{q-1}}{\tau^{\text{decr}}} & \text{if } \left\|\vec{s}^{q-1}\right\|_2 > \mu\left\|\vec{r}^{q-1}\right\|_2 \\ \rho^{q-1} & \text{otherwise} \end{cases} \tag{21}$$

with the primal residual $\vec{r}^q = \vec{A}\vec{x}^q + \vec{B}\vec{z}^q - \vec{c}$, the dual residual $\vec{s}^q = \rho\vec{A}^{\mathrm{T}}\vec{B}(\vec{z}^q - \vec{z}^{q-1})$. The basic idea is to keep both within a factor of μ of one another. Following this idea for the OCP (7), the primal and dual residuals are given by

$$\vec{r}_i^q = \begin{bmatrix} \vec{x}_i^q \\ \vec{u}_i^q \\ \begin{bmatrix} \vec{x}_{ji}^q \\ \vec{u}_{ji}^q \end{bmatrix}_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix} - \begin{bmatrix} \vec{z}_{x,i}^q \\ \vec{z}_{u,i}^q \\ \begin{bmatrix} \vec{z}_{x,j}^q \\ \vec{z}_{u,j}^q \end{bmatrix}_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix}, \quad i \in \mathcal{V} \tag{22a}$$

$$\vec{s}_i^q = \operatorname{diag}\left(\begin{bmatrix} \vec{C}_i^{q-1} \\ \left[\vec{C}_{ji}^{q-1}\right]_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix}\right)\begin{bmatrix} \vec{z}_{x,i}^q - \vec{z}_{x,i}^{q-1} \\ \vec{z}_{u,i}^q - \vec{z}_{u,i}^{q-1} \\ \begin{bmatrix} \vec{z}_{x,j}^q - \vec{z}_{x,j}^{q-1} \\ \vec{z}_{u,j}^q - \vec{z}_{u,j}^{q-1} \end{bmatrix}_{j\in\mathcal{N}_i^{\leftarrow}} \end{bmatrix}, \quad i \in \mathcal{V}. \tag{22b}$$

To reduce the number of tuning parameters, $\mu = 1$ is chosen which results in an equality instead of the inequality in (21). To further simplify the implementation, the equality is evaluated element-wise and at each discrete time step $\delta_k = \frac{T}{N-1}$ with N as discretization of the predicted horizon. This results in the condition

$$\left\|\vec{r}_i^q(\delta_k)\right\|_2 \overset{!}{=} \left\|\vec{s}_i^q(\delta_k)\right\|_2 \tag{23}$$

for each discrete time step δ_k and the norm evaluated element-wise. The condition (23) can be reformulated in form of the update law

$$\rho_m^q(\delta_k) = \rho_m^{q-1}(\delta_k)\frac{\left|r_m^q(\delta_k)\right|}{\left|s_m^q(\delta_k)\right|} = \rho_m^{q-1}(\delta_k)\gamma_m^q(\delta_k) \tag{24}$$

with m as index for an arbitrary element in (23). The implementation is presented in Algorithm 2. At first, the division through small numbers, especially zero, is caught to prevent numerical issues. The factor γ^q is computed afterwards and bound between $\gamma_{\min}$ and $\gamma_{\max}$, before the new penalty parameter is calculated by $\rho^q = \gamma^q\rho^{q-1}$.

Algorithm 2 Adaption of penalty parameters

Calculate $|s^q|$ and $|r^q|$

if $|s^q| > \epsilon_0$ **then**
  $\gamma^q = \frac{|r^q|}{|s^q|}$
  if $\gamma^q > \gamma_{\max}$ **then**
    $\gamma^q = \gamma_{\max}$
  else if $\gamma^q < \gamma_{\min}$ **then**
    $\gamma^q = \gamma_{\min}$
  end if
else
  $\gamma^q = 1$
end if
$\rho^q = \gamma^q \rho^{q-1}$

4 Modular framework

GRAMPC-D is implemented in a modular fashion in order to achieve a scalable and flexible implementation. At first, the modular structure is explained before the capability for plug-and-play scenarios is laid out.

4.1 Modular structure

The main parts of GRAMPC-D and their interaction are presented in the following. Due to the modular concept, the implementation of GRAMPC-D can be subdivided into single modules that are composed depending on the chosen type of controller, centralized or distributed, as the structure of GRAMPC-D differs between the two cases. Both are visualized in Fig. 2. Either structure is generated automatically by choosing the corresponding controller type without further required interaction of the user. Both the distributed and centralized structure are scalable due to the modular concept and therefore suitable to handle large or complex systems.

The distributed control structure in the left part of Fig. 2 assumes that each agent only has access to its local variables and communication is required to acquire data from other agents. Thus, the central part in the distributed setup is the communication interface. While it enables to exchange data between agents, the actual implementation depends on the chosen type of communication interface. If the DMPC is simulated on a single processor, there is no need to actually send data over a network. Instead, a central communication interface is provided that exchanges data pointers, which is a significant difference in performance. If the ADMM algorithm is implemented in an actual distributed setup, each agent creates its own local communication interface that enables to exchange data over a network. The corresponding protocol is encapsulated into the local communication interface due to the modular concept that enables implementing multiple protocols and switching between them. In either case, each agent creates a local solver that contains the local OCP

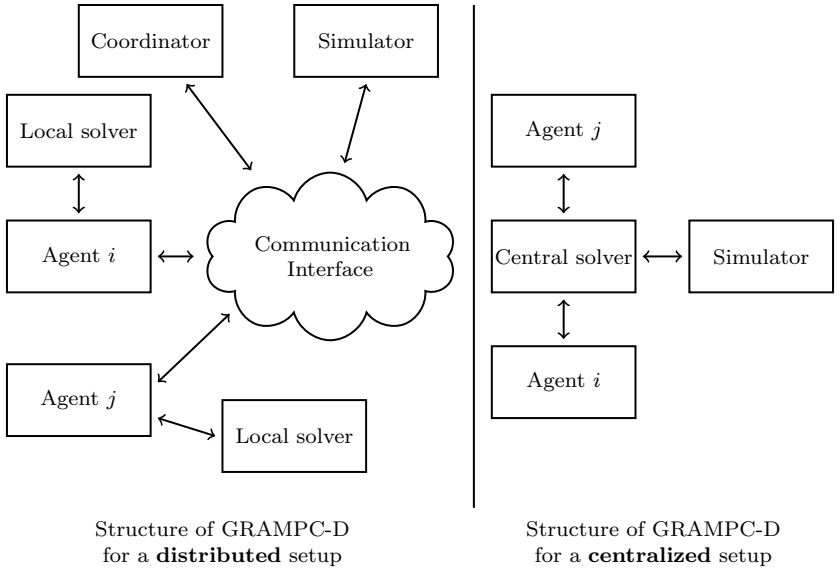

Fig. 2 The communication interface is the central part of GRAMPC-D in case of distributed optimization. Each agent has its own local solver that handles the steps of the ADMM algorithm. The coordinator provides a synchronization of all agents while the simulator handles the simulation of the overall system. If a centralized controller is chosen, GRAMPC-D is centered around the central solver that knows all agents and can access their variables

depending on the neighborhood and the chosen optimization parameters such as neighbor approximation. Hence, tasks like decoupling the global OCP by introducing local copies are done automatically in the background. The ADMM algorithm is implemented inside the local solver with an abstract implementation of the minimization problem with respect to the local variables. This enables implementing multiple solvers and switching between them without changing other parts of the software, although GRAMPC is chosen as default. The remaining two important modules are the coordinator and the simulator. The ADMM algorithm assumes a fully synchronized execution, which has to be guaranteed even in a distributed setup. This synchronization is handled by the coordinator by triggering each step of the algorithm and waiting for a response of each agent before sending the following trigger. The last module is an integrated simulator that enables simulations independent of the chosen controller or the specific system.

A more simple structure is generated if a centralized controller is chosen, see right part of Fig. 2. In this case, the global OCP (1) is solved in a centralized manner including all agents dynamics and having knowledge of all variables. The centralized setup implicitly synchronizes the execution of the algorithm without the need for a coordinator. The only remaining module is the simulator that is used to simulate the overall system.

Each part of GRAMPC-D is interchangeable by alternative software. As already mentioned, the MPC toolbox GRAMPC is used by default to solve both the global OCP (1) in case of a centralized controller and the underlying reduced OCP (9) in

Step 1 of the ADMM algorithm in case of a distributed controller. GRAMPC is tailored to embedded hardware and by this a natural choice, but if another solver is desired, e.g. with a stronger focus on precision instead of computational speed, then the only required change is to overload the class regarding the solver with a new implementation. The same holds for each part of the framework such as the implemented communication protocol. The TCP protocol is provided by default, but alternative protocols can be implemented by overloading the local communication interface.

4.2 Rapid prototyping

Both the local and global OCPs are dynamically generated at run-time based on the same problem description provided by the user, see Fig. 3. In case of a centralized controller, the global OCP is generated by the central solver while in case of a distributed controller, the local OCP is generated for each agent individually based on its neighbors and the optimization parameters. Therein, the corresponding flags for neighbor approximation are defined that state whether additional variables have to be initialized, e.g. local copies $\vec{\bar{x}}_{ji}$ and $\vec{\bar{u}}_{ji}$ or the external influence $\vec{v}_{ij}$, see Sect. 3.2. This results in a convenient prototyping process while designing controllers, as each type of controller can be automatically generated and evaluated based on the same problem description. To further support the efficiency of prototyping, the possibility of multi-threading can be activated to spread the computation effort on each available core of the processor and thus to speed-up the computations.

4.3 Plug-and-play functionality

Plug-and-play is a core feature for the usability of a framework in the field of DMPC, see e.g. Zeilinger et al. (2013), Riverso et al. (2014), Riverso and Ferrari-Trecate

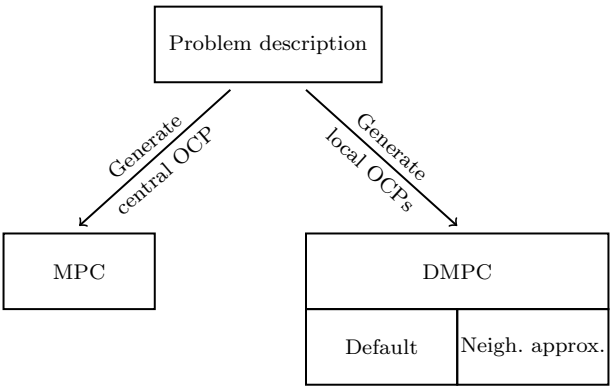

Fig. 3 Based on the same problem description, the global OCP is generated for a centralized controller and the local OCPs in the distributed case. The local OCPs are automatically extended by corresponding terms if neighbor approximation is enabled for the distributed controller

(2015), where the OCP structure and size may change dynamically due to the removal or plug-in of agents. The generation of the OCPs (9) during run-time and the modular structure of the framework allows to integrate plug-and-play functionality in the network. If an agent enters the system in the distributed control setting (left part of Fig. 2), the coordinator informs the direct neighbors of the agents to include the corresponding variables. Hence, only the OCPs (9) of the direct neighbors are updated and the new agent is integrated into the network. If an agent leaves the network, the agents and the coordinator delete the agent from their internal lists of active agents. This results in a smooth transition between two problem formulations.

5 Simulation examples

The modular framework GRAMPC-D is evaluated for different examples. The scalability is shown for a coupled spring-mass system. The plug-and-play functionality and the concept of neighbor approximation are demonstrated for a smart grid and a coupled water tank network, respectively. Finally, a distributed hardware implementation for a system of coupled Van der Pol oscillators is considered by communicating over an actual network using the TCP protocol.

In each example, the terminal and integral cost in (2) are chosen quadratically

$$\begin{aligned} V_i(\vec{x}_i, T) &= \frac{1}{2}\left\|\vec{x}_i(T) - \vec{x}_{i,\mathrm{des}}(T)\right\|_{\vec{P}_i}^2 \\ l_i(\vec{x}_i, \vec{u}_i, t) &= \frac{1}{2}\left\|\vec{x}_i - \vec{x}_{i,\mathrm{des}}\right\|_{\vec{Q}_i}^2 + \frac{1}{2}\left\|\vec{u}_i\right\|_{\vec{R}_i}^2 \end{aligned} \tag{25}$$

with the positive (semi-)definite weighting matrices $\vec{P}_i \in \mathbb{R}^{n_{x,i}\times n_{x,i}}$, $\vec{Q}_i \in \mathbb{R}^{n_{x,i}\times n_{x,i}}$ and $\vec{R}_i \in \mathbb{R}^{n_{u,i}\times n_{u,i}}$. The desired state to be controlled is given by $\vec{x}_{i,\mathrm{des}}$. The computation times are measured on an Intel i5 CPU with 3.4 GHz using Windows 10. The communication effort is neglected if not stated otherwise.

5.1 Scalable system

The scalability of GRAMPC-D is shown for a system consisting of a set of masses that are coupled by springs. Each mass is represented by an agent $i \in \mathcal{V}$ and is described by the differential equations

$$\begin{bmatrix} \ddot{p}_{x,i} \\ \ddot{p}_{y,i} \end{bmatrix} = \begin{bmatrix} u_{x,i} \\ u_{y,i} \end{bmatrix} + \sum_{j\in\mathcal{N}_i} \frac{c}{m}\left(1 - \frac{\delta_0}{\delta_{ij}(p_{x,i}, p_{y,i})}\right)\begin{bmatrix} p_{x,j} - p_{x,i} \\ p_{y,j} - p_{y,i} \end{bmatrix} \tag{26}$$

with the position $(p_{x,i}, p_{y,i})$ of the respective mass in the x- and y-axis and the respective controls $(u_{x,i}, u_{y,i})$. This results in the state and control vectors

$$\vec{x}_i = \begin{bmatrix} p_{x,i} & \dot{p}_{x,i} & p_{y,i} & \dot{p}_{y,i} \end{bmatrix}^{\mathsf{T}} \tag{27a}$$

$$\vec{u}_i = \begin{bmatrix} u_{x,i} & u_{y,i} \end{bmatrix}^\mathsf{T}. \tag{27b}$$

The spring is relaxed at the length $\delta_0 = 1m$. The spring constant is given by $c = 0.5Nm^{-1}$ and each agent has a mass of $m_i = 7.5kg$. The function $\delta_{ij}(p_{x,i}, p_{y,i}) = \sqrt{(p_{x,i} - p_{x,j})^2 + (p_{y,i} - p_{y,j})^2}$ computes the distance between two agents i and j. The dynamics (26) can be split into functions $\vec{f}_i$ and $\vec{f}_{ij}$ corresponding to the neighbor-affine form (1b). The weighting matrices are set to (SI units are omitted for simplicity)

$$\vec{P}_i = \text{diag}[1, 1, 1, 1], \quad \vec{Q}_i = \text{diag}[5, 2, 5, 2], \quad \vec{R}_i = \text{diag}[0.01, 0.01] \tag{28}$$

with the desired state

$$\vec{x}_{i,\text{des}} = \begin{bmatrix} p_{x,i,\text{des}} & 0ms^{-1} & p_{y,i,\text{des}} & 0ms^{-1} \end{bmatrix}^\mathsf{T}. \tag{29}$$

It was shown in Burk et al. (2019) that the computation time per agent is nearly independent of the system size, whereas in the central MPC case the computation time rises drastically. The simulation results for a system with 40×40 agents are given in Fig. 4. It can be seen that the trajectories of the cost are quite similar. While the distributed solution is slightly suboptimal, it would converge to the centralized solution by increasing the number of ADMM iterations. The computation time for each time step, however, is 1072.58 ms for the centralized controller, while the distributed controller requires a maximum of 9.59 ms and an average of 2.23 ms per agent.

5.2 Plug-and-play

The plug-and-play capability of GRAMPC-D is presented using an exemplary setup of a smart grid. The network is described by a set of coupled agents that represent non-controllable power sinks and sources, such as private households, industry or renewable energy as well as controllable power plants. The dynamical behavior of the agents is generalized by describing them as generators with a mechanical phase angle $\theta_i(t) \in \mathbb{R}$ that may differ from the phase of the grid. The corresponding dynamics

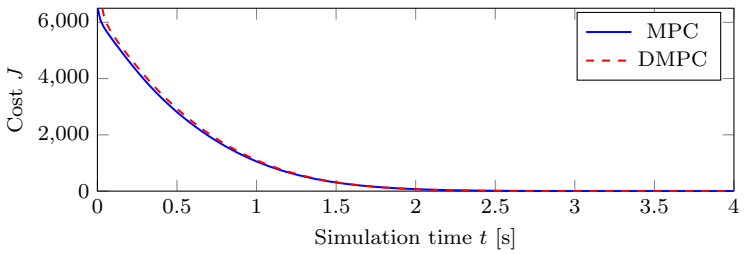

Fig. 4 Global cost trajectory of the scalable spring-mass system with 40×40 agents for the central MPC and DMPC case

$$\ddot{\phi}_i = \frac{1}{I\Omega}\left(u_i + P_{\text{source},i} - \kappa\Omega^2\right) - 2\frac{\kappa}{I}\dot{\phi} - \sum_{j\in\mathcal{N}_i^{\leftarrow}} \frac{P_{\max,ij}}{I\Omega}\sin\left(\phi_j - \phi_i\right) \tag{30}$$

with friction constant $\kappa > 0$ and the moment of inertia I is given in a neighbor-affine form and describes the dynamical behavior of the phase shift $\phi_i(t) \in \mathbb{R}$ given by

$$\phi_i = \theta_i - \Omega\tau \tag{31}$$

between the phase $\Omega\tau$ of the grid with frequency Ω and the mechanical angle θ_i, see Rohden et al. (2012). Hence, the state vector is given by

$$\vec{x}_i = \begin{bmatrix}\phi_i & \dot{\phi}_i\end{bmatrix}^\mathsf{T}. \tag{32}$$

In (30), $P_{\text{source},i}$ describes the generalized non-controllable power, e.g. the demanded power for private households and industry or the generated power by renewable energy. The coupling between two agents consists of the maximum transferable power $P_{\max,ij}$ that depends on the phase shift angle $\phi_j - \phi_i$ between agent $i \in \mathcal{V}$ and its neighbors $j \in \mathcal{N}_i^{\leftarrow}$. Agents that describe power plants have a controllable input u_i that is used to stabilize the grid. A normalized parameterization is used with $\Omega = 1$ Hz and $I = 1$ Js^2, the friction term set to $\kappa = 1 \times 10^{-3}$ Js and the maximum transferable power to $P_{\max,ij} = 0.1$ Js^1. The weighting matrices are set to

$$\vec{P}_i = \operatorname{diag}\left[0s^2, 0.1s^4\right], \quad \vec{Q}_i = \operatorname{diag}\left[0s^2, 1s^4\right], \quad \vec{R}_i = 0.01\ \text{s}^2\ \text{J}^{-2} \tag{33}$$

with the desired state

$$\vec{x}_{i,\text{des}} = \begin{bmatrix}\times & 0s^{-2}\end{bmatrix}^\mathsf{T}. \tag{34}$$

The first element of the desired state $\vec{x}_{i,\text{des}}$ is set arbitrary, as there is no desired value for the phase and the first state is not weighted in the cost functional.

The implemented setup is visualized in Fig. 5. At the start of the simulation, one power plant is given that supplies one non-controllable power-sink such as a private household. During run-time, an additional power sink is coupled to the first one, i.e. the power plant has to supply both using the same connection. The simulation results are shown in Fig. 6, starting with the first power sink that is connected to the power plant. It can be seen that the phase difference between the power plant and the household converges to a stationary value that leads to a transmission of the demanded power. Furthermore, the angular velocity of the

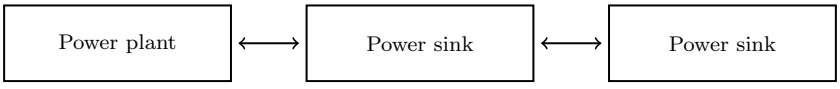

Fig. 5 The plug-and-play capability of GRAMPC-D is presented using the simulation of a smart grid. At the beginning of the simulation, only one power sink is connected to the power plant. During run-time, the second power sink is connected to the first one, i.e. the power plant has to supply both using the same connection

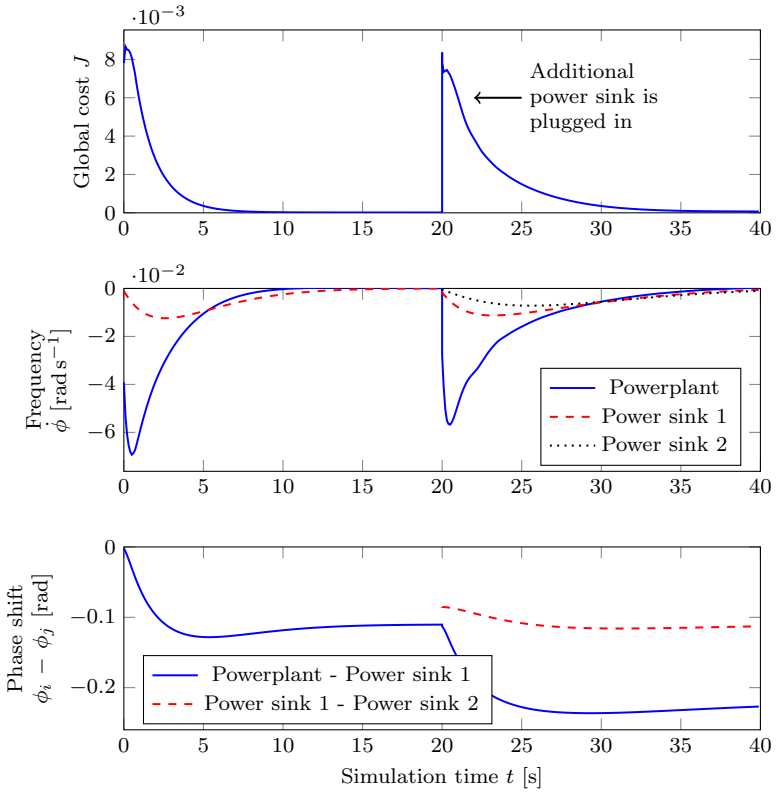

Fig. 6 The Plug-and-play functionality of GRAMPC-D for the smart grid example is shown. The plot at the top shows the trajectory of the global cost, the plot in the middle the frequencies of the single agents and the plot in the bottom the phase shift between the agents. The additional power sink is plugged in at simulation time $t = 20$ s

phase shift converges to zero. At simulation time $t = 20$ s, the second power sink is plugged in, leading to an additional power demand. Consequently, the phase shift between the power plant and the first power sink increases and the additional power is transmitted. The phase shift between the first and second power sink is adapted accordingly. The computation time for the distributed controller is given by a maximum of 84.67 ms and an average of 4.78 ms per agent using a step size of $\Delta_t = 100$ ms. The average time is significantly lower due to the convergence criterion of the ADMM algorithm.

This plug-in and plug-out functionality of agents is supported at any moment during the simulation even if the controllers run on distributed hardware and communicate over a network. GRAMPC-D can handle planned changes in the system such as shown in this simulation example as well as spontaneous disconnections due to a broken network connection.

5.3 Neighbor approximation

The concept of neighbor approximation is evaluated for a system of water tanks that are coupled by pipes, see Fig. 7. Only the first water tank has a controllable input $u_1(t) \in \mathbb{R}$. The last water tank has a constant outflow $d_5 = 0.01\,\mathrm{m^3 s^{-1}}$ and the desired water height $x_{5,\mathrm{des}} = 3\,\mathrm{m}$. In addition, the inequality constraint $h_i \leq 3\,\mathrm{m}$ of a maximum water height has to be satisfied by all tanks. The dynamics of each water tank is given by

$$\dot{h}_i = \frac{1}{A_i}(u_i - d_i) + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \frac{a_{ij}}{A_i} \operatorname{sign}(h_j - h_i) \sqrt{2g\left|h_j - h_i\right|} \tag{35}$$

with the water height $h_i(t) \in \mathbb{R}$ and the state vector $x_i = h_i$. The area of each water tank is given by $A_i = 0.1\,\mathrm{m^2}$ and the diameter of the pipes by $a_{ij} = 0.005\,\mathrm{m^2}$. The weights for the cost functions are set to

$$P_1 = 0\,\mathrm{m^{-2}}, \qquad Q_1 = 0\,\mathrm{m^{-2}}, \qquad R_1 = 0.1\,\mathrm{s^2 m^{-6}} \tag{36a}$$

$$P_i = 0\,\mathrm{m^{-2}}, \qquad Q_i = 0\,\mathrm{m^{-2}}, \qquad R_i = 0\mathrm{s^2 m^{-6}}, \quad i \in \{2,3,4\} \tag{36b}$$

$$P_5 = 1\,\mathrm{m^{-2}}, \qquad Q_5 = 1\,\mathrm{m^{-2}}, \qquad R_5 = 0\mathrm{s^2 m^{-6}}. \tag{36c}$$

The simulation is run with a distributed controller both with and without neighbor approximation using the same set of parameters for the ADMM algorithm and GRAMPC. The convergence of the ADMM algorithm is shown in Fig. 8 for both simulations. The simulation with neighbor approximation converges smoothly to the optimal solution and satisfies the convergence criterion after 7 iterations while 89 ADMM iterations are required without neighbor approximation. Note that the cost is rising instead of falling as the solution is infeasible until the algorithm converges. The improved convergence behavior with neighbor approximation comes with a higher computational complexity per ADMM iteration. However, if a convergence criterion is used, the decreased number of ADMM iterations per time step compensate for the higher computational complexity. The required computation time for the 89 ADMM iterations without neighbor approximation is 955.6 ms and for the 7 iterations using neighbor approximation 203.8 ms. Note that the same configuration for

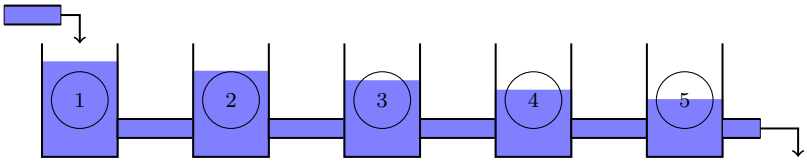

Fig. 7 The concept of neighbor approximation is shown at a simulation example of coupled water tanks. Only the first one has an input and only the last one has a desired water height while being disturbed by a constant outflow

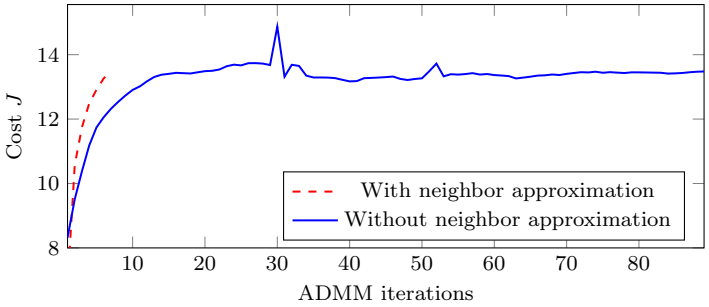

Fig. 8 Convergence behavior of the ADMM algorithm with and without neighbor approximation. The algorithm converges within 89 ADMM iterations without neighbor approximation opposed to 7 iterations if neighbor approximation is used

the ADMM algorithm and GRAMPC is used in this evaluation to provide a comparable result. The standard ADMM algorithm may converge within less iterations if more computation effort is spent per iteration while the algorithm may require even less time if the parameters are tuned for neighbor approximation.

5.4 Distributed hardware implementation

This section shows the capability of GRAMPC-D to solve the ADMM algorithm on distributed embedded hardware. The following simulation is run on four Raspberry Pi 3B+ using Raspberry Pi OS that are connected via Ethernet. Each agent runs on an individual Raspberry Pi while the coordinator and the simulator use the same hardware. The local communication interface from GRAMPC-D based on the TCP protocol is used. The simulation example consists of three coupled (Van der Pol oscillators Barrón and Sen 2009)

$$\ddot{p}_i = \alpha_1\left(1 - p_i^2\right) - p_i + u_i + \sum_{j \in \mathcal{N}_i^{\leftarrow}} \alpha_2\left(p_j - p_i\right) \tag{37}$$

with state $p_i(t) \in \mathbb{R}$, control $u_i(t) \in \mathbb{R}$, the uncoupled oscillator constant $\alpha_1 = 1\,\mathrm{m}^{-1}\mathrm{s}^{-2}$, and the coupling constant $\alpha_2 = 1s^{-2}$. The state vector and desired state are given by

$$\vec{x}_i = \begin{bmatrix} p_i & \dot{p}_i \end{bmatrix}^\mathsf{T} \tag{38a}$$

$$\vec{x}_{i,\mathrm{des}} = \begin{bmatrix} 0\,\mathrm{m} & 0\,\mathrm{m\,s}^{-1} \end{bmatrix}^\mathsf{T} \tag{38b}$$

with the weighting matrices set to

$$\vec{P}_i = \mathrm{diag}\left[1\,\mathrm{m}^{-2} \;\; 1\,\mathrm{m}^{-2}\,\mathrm{s}^2\right], \tag{39a}$$

$$\vec{Q}_i = \mathrm{diag}\left[1\,\mathrm{m}^{-2}\;\; 1\,\mathrm{m}^{-2}\,\mathrm{s}^2\right], \tag{39b}$$

$$R_i = 0.1\,\mathrm{m}^{-1}\,\mathrm{s}^4. \tag{39c}$$

The simulation is run for 10.000 time steps using a fixed number of $q_{\max} = 5$ ADMM iterations.

Figure 9 shows the required time for computation and communication in each time step. The computation time to execute 5 ADMM iterations amounts to 10 ms per agent while the average time required to solve the ADMM algorithm in a distributed manner using the TCP protocol for the communication is 66.51 ms. Hence, the average effort for the communication is 56.51 ms including 72 communication steps. Since all agents have to be synchronized for the ADMM algorithm, either of them has to wait for the slowest agent at each step of the algorithm. All five ADMM iterations can be executed in 48 ms with the worst case time of 114 ms. This results in the minimum time for each communication step of 0.53 ms, an average time of 0.79 ms and a maximum time of 1.44 ms. This time includes preparing the data to be sent, sending and receiving it and recreating the sent data structure from the byte array. These are plausible values as a ping to the loopback address 127.0.0.1 already requires 0.14 ms in average and 0.22 ms at maximum. These results show that the main effort in distributed optimization is the communication effort that requires 82.3% of the overall time in average for this simulation example.

6 Conclusions

The open-source, modular DMPC framework GRAMPC-D is presented in this paper that enables to solve scalable optimal control problems in a convenient way and to stabilize plants using distributed model predictive control. This problem description can be used for both a centralized and a distributed controller. In the

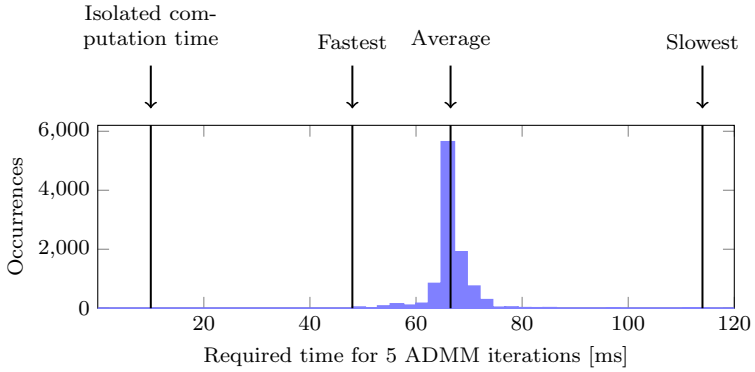

Fig. 9 Communication effort of the ADMM algorithm on distributed hardware with 48 ms (minimum), 66.51 ms (average), and 114 ms (maximum) compared to the computation time of 10 ms

distributed setting, the global optimal control problem is automatically decoupled and solved in a distributed manner using the ADMM algorithm. The convergence behavior of the ADMM algorithm can be improved by sugint he concept of neighbor approximation that allows the agents to anticipate the actions of their neighbors. The presented DMPC framework supports plug-and-play to connect and remove agents at run-time. Besides solving the ADMM algorithm on a single processor, it is possible to solve the local optimal control problems on distributed hardware. The local communication interface enables communication between agents over a network using the TCP protocol. By default, GRAMPC-D uses the MPC toolbox GRAMPC for solving the local optimal control problems on agent level, which is suitable for real-time and embedded implementations.

GRAMPC-D 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-d/grampc-d. Future work will use the modular structure to extend GRAMPC-D. For example, communication protocols besides TCP can be provided or alternative solvers to GRAMPC for the underlying minimization problem implemented to increase the usability and flexibility of the framework.

Acknowledgements This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Project No. GR 3870/4-1.

Funding Open Access funding enabled and organized by Projekt DEAL..

References

Allgöwer F, Zheng A (2012) Nonlinear model predictive control, vol 26. Birkhäuser, Basel

Barrón MA, Sen M (2009) Synchronization of four coupled van der Pol oscillators. Nonlinear Dyn 56(4):357–367

Bertsekas DP (1996) Constrained optimization and Lagrange multiplier methods. Academic Press, Belmont

Bestler A, Graichen K (2019) Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM. Optim Control Appl Methods 40:1–23

Burk D, Völz A, Graichen K (2019) Towards a modular framework for distributed model predictive control of nonlinear neighbor-affine systems. In: Proceedings of 58th IEEE CDC, Nice (France), 2019, pp 5279–5284

Burk D, Völz A, Graichen K (2020) Neighbor approximations for distributed optimal control of nonlinear networked systems. In: Proceedings of ECC, St. Petersburg (Russia), 2020, pp 1238–1243

Boyd S, Parikh N, Chu E, Peleato B, Eckstein J (2011) Distributed optimization and statistical learning via the alternating direction method of multipliers. In: Foundations and trends in machine learning, vol 3, no. 1, 2011. https://doi.org/10.1561/2200000016

Camponogara E, Jia D, Krogh B, Talukdar S (2002) Distributed model predictive control. IEEE Control Syst Mag 22(1):44–52

Engelmann A, Jiang Y, Benner H, Ou R, Houska B, Faulwasser T (2020) ALADIN-α—an open-source MATLAB toolbox for distributed non-convex optimization. arXiv preprint arXiv:2006.01866

Englert T, Völz A, Mesmer F, Rhein S, Graichen K (2019) A software framework for embedded nonlinear model predictive control using a gradient-based augmented Lagrangian approach (GRAMPC). Optim Eng 20(3):769–809

Farina F, Camisa A, Testa A, Notarnicola I, Notarstefano G (2019) DISROPT: a Python Framework for Distributed Optimization. arXiv preprint arXiv:1911.02410

Filatrella G, Nielsen AH, Pedersen NF (2008) Analysis of a power grid using a kuramoto-like model. Eur Phys J B 61(4):485–491

Hentzelt S, Graichen K (2013) An augmented Lagrangian method in distributed dynamic optimization based on approximate neighbor dynamics. In: Proceedings of IEEE SMC, Manchester (United Kingdom), 2013, pp 571–576

Houska B, Ferreau HJ, Diehl M (2011) ACADO toolkit—an open-source framework for automatic control and dynamic optimization. Optim Control Appl Methods 32:298–312

Houska B, Frasch J, Diehl M (2016) An augmented Lagrangian based algorithm for distributed nonconvex optimization. SIAM J Optim 26(2):1101–1127

Houska B, Kouzoupis D, Jiang Y, Diehl M (2018) Convex optimization with ALADIN. Math Program

Jakob W, Rhinelander J, Moldovan D (2017) pybind11—seamless operability between C++11 and Python, 2017. https://github.com/pybind/pybind11/commit/e7d304fbc6f0aa22b6105e2b1a4807c7a294eafa

Kalmari J, Backman J, Visala A (2015) A toolkit for nonlinear model predictive control using gradient projection and code generation. Control Eng Pract 39:56–66

Käpernick B, Graichen K (2014) The gradient based nonlinear model predictive control software grampc. In: Proceedings of ECC, 2014, pp 1170–1175

Gäfvert O (2014) A practical guide to distributed model predictive control, 2014. [Online]. http://github.com/olivergafvert/dmpc. Accessed 9 Mar 2021

Maestre JM, Negenborn RR (2014) Distributed model predictive control made easy. Springer, New York

Mayne D, Rawlings J, Rao C, Scokaert P (2000) Constrained model predictive control: stability and optimality. Automatica 36(6):789–814

Riverso S, Farina M, Ferrari-Trecate G (2014) Plug-and-play model predictive control based on robust control invariant sets. Automatica 50(8):2179–2186

Riverso S, Ferrari-Trecate G (2015) Plug-and-play distributed model predictive control with coupling attenuation. Optim Control Appl Methods 36(3):292–305

Riverso S, Battocchio A, Ferrari-Trecate G (2013) PnPMPC toolbox, 2013. [Online]. http://sisdin.unipv.it/pnpmpc/pnpmpc.php. Accessed 9 Mar 2021

Rohden M, Sorge A, Timme M, Witthaut D (2012) Self-organized synchronization in decentralized power grids. Phys Rev Lett 109(6):064 101-1–064 101-5

Scheu H, Marquardt W (2011) Sensitivity-based coordination in distributed model predictive control. J Process Control 21(5):715–728

Verschueren R, Frison G, Kouzoupis D, van Duijkeren N, Zanelli A, Novoselnik B, Frey J, Albin T, Quirynen R, Diehl M (2019) acados: a modular open-source framework for fast embedded optimal control. arXiv preprint arXiv:1910.13753

Zeilinger M, Pu Y, Riverso S, Ferrari-Trecate G, Jones C (2013) Plug and play distributed model predictive control based on distributed invariance and optimization. In: Proceedings of 52nd IEEE CDC, Florence (Italia), December 2013, pp 5770–5776