2019 IEEE 58th Conference on Decision and Control (CDC)
Palais des Congrès et des Expositions Nice Acropolis
Nice, France, December 11-13, 2019

Towards a Modular Framework for Distributed Model Predictive Control of Nonlinear Neighbor-Affine Systems

Daniel Burk, Andreas Völz and Knut Graichen

Abstract— **The development of real-time capable distributed model predictive controllers (DMPC) for nonlinear systems requires considerable effort both for the numerical solution and for the practical implementation, which is one of the reasons why DMPC so far is rarely found in practice. With the aim of simplifying the development process, this paper presents a modular framework for DMPC of nonlinear neighbor-affine systems. The numerical solution is based on the alternating direction method of multipliers (ADMM). The local optimization problems are solved using the toolbox GRAMPC which implements a gradient-based augmented Lagrangian algorithm that is particularly suited for embedded applications. The performance of the DMPC framework is evaluated for two scalable nonlinear distributed systems, whereby the scalability and the plug-and-play functionality are demonstrated.**

I. INTRODUCTION

The control of large-scale systems such as smart grids or water transport networks offers particular challenges. Centralized control strategies are often not applicable, due to the high computational effort. On the other hand, it is difficult to handle highly coupled systems with purely local controllers. A promising concept for such systems is distributed model predictive control (DMPC) [1], [2] that is based on the distributed solution of a centralized optimization problem. Popular algorithms that can be applied to nonlinear systems are the alternating direction method of multipliers (ADMM) [3], [4], the ALADIN approach [5] or sensitivity-driven distributed model predictive control (S-DMPC) [6].

With regard to the solution of the centralized MPC problem, several software packages, as e.g. ACADO [7], VIATOC [8], or GRAMPC [9], are available that hugely reduce the required effort for implementing nonlinear model predictive controllers. However, the implementation of distributed MPC is more complex as it involves a tight integration of numerical optimization and communication, both of which affect the real-time feasibility of the control system. The PnPMPC toolbox [10] is a MATLAB package for distributed MPC of constrained linear time-invariant systems with a focus on plug-and-play functionality, see e.g. [11] or [12]. Another MATLAB package is the DMPC toolbox [13]. It considers constrained discrete linear time-invariant systems. However, there is a lack of software support for the implementation of real-time capable distributed MPC of nonlinear systems.

In order to close this gap, this paper presents a modular framework for distributed model predictive control of nonlinear continuous-time systems. It is assumed that the dynamics of each agent can be written in a so-called neighbor-affine form, which means that the (in general nonlinear) influences of the neighbors enter additively into the local dynamics. The distributed solution of the centralized optimization problem is based on the alternating direction method of multipliers. Furthermore, the GRAMPC toolbox [9] is employed for the numerical solution of the underlying minimization problems. The focus of the work lies on the modular structure of the framework and a separation between problem formulation, numerical optimization, and communication aspects. A particular goal is to use the same problem formulation both for the centralized and the distributed MPC as well as for simulation and experiments. In contrast to the MATLAB toolboxes mentioned above, the framework is written in C++ with embedded usage in mind. Finally, the application of the framework is demonstrated for a scalable spring-mass-system and a network of coupled water tanks.

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

The authors are with the Chair of Automatic Control, Friedrich-Alexander-University Erlangen-Nuremberg, Germany.

II. PROBLEM FORMULATION

This paper considers a distributed system that is composed of individual agents. The network topology is described by a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with the vertex set $\mathcal{V}$ and the edge set $\mathcal{E}$. Each vertex $i \in \mathcal{V} = \{1, \ldots, N\}$ describes an agent with states $\boldsymbol{x}_i \in \mathbb{R}^{n_i}$ and controls $\boldsymbol{u}_i \in \mathbb{R}^{m_i}$. Each edge in $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ corresponds to a state- and control-dependent coupling between two agents. The set of sending neighbors $\mathcal{N}_{i\leftarrow} = \{j : (j,i) \in \mathcal{E}, j \neq i\}$ describes all agents that influence the agent i, as visualized in Figure 1. Similarly, the set of receiving neighbors $\mathcal{N}_{i\rightarrow} = \{j : (i,j) \in \mathcal{E}, i \neq j\}$ describes all agents that are influenced by the agent i.

The global optimization problem is defined by

$$\min_{\boldsymbol{u}_i,\, i\in\mathcal{V}} \quad \sum_{i\in\mathcal{V}} V_i(\boldsymbol{x}_i(T)) + \int_0^T l_i(\boldsymbol{x}_i(\tau), \boldsymbol{u}_i(\tau))\,\mathrm{d}\tau \tag{1a}$$

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

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

$$\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0} \tag{1d}$$

$$\boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \le \boldsymbol{0} \tag{1e}$$

$$\boldsymbol{g}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j, \boldsymbol{u}_j) = \boldsymbol{0}\,, \quad j \in \mathcal{N}_{i\leftarrow} \tag{1f}$$

$$\boldsymbol{h}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \boldsymbol{x}_j, \boldsymbol{u}_j) \le \boldsymbol{0}\,, \quad j \in \mathcal{N}_{i\leftarrow} \tag{1g}$$

$$\boldsymbol{u}_i \in \mathcal{U}_i\,, \quad i \in \mathcal{V} \tag{1h}$$

with the central cost function (1a) on the time horizon T, the nonlinear dynamics (1b) in the neighbor-affine form, and the

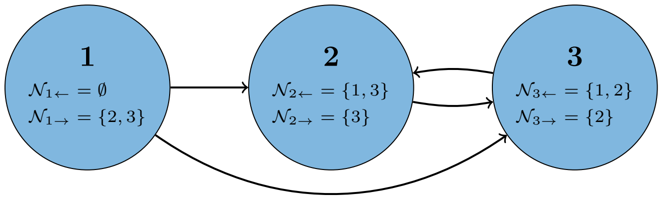
1
$\mathcal{N}_{1\leftarrow} = \emptyset$
$\mathcal{N}_{1\rightarrow} = \{2,3\}$
2
$\mathcal{N}_{2\leftarrow} = \{1,3\}$
$\mathcal{N}_{2\rightarrow} = \{3\}$
3
$\mathcal{N}_{3\leftarrow} = \{1,2\}$
$\mathcal{N}_{3\rightarrow} = \{2\}$

Fig. 1. The set of sending neighors $\mathcal{N}_{i\leftarrow}$ describes all agents that influence the agent i. Similarly, the set of receiving neighbors $\mathcal{N}_{i\rightarrow}$ describes all agents that are influenced by the agent i.

initial condition $\hat{\boldsymbol{x}}_{i,k}$ of agent i in the MPC step k. In addition, equality and inequality constraints are considered. Finally, the controls are constrained by the compact convex sets $\mathcal{U}_i$, $i \in \mathcal{V}$. Note that the internal time argument $\tau \in [0,T]$ is omitted in favor of a compact notation. A centralized model predictive controller solves the optimization problem (1) in each sampling step $t_k = k\Delta t$ with the sampling time Δt and the updated initial condition (1c) and applies the first part of the control trajectory $\boldsymbol{u}_i(\tau)$, $\tau \in [0, \Delta t)$ to each agent $i \in \mathcal{V}$.

The definition of the optimization problem in the form of (1) is the basis of the modular framework as it can be decomposed into simpler subsystems. The cost functions V_i and l_i as well as the functions $\boldsymbol{f}_i$, $\boldsymbol{g}_i$, and $\boldsymbol{h}_i$ are part of the agent model as they only depend on the states and controls of the agent $i \in \mathcal{V}$. In contrast, the functions $\boldsymbol{f}_{ij}$, $\boldsymbol{g}_{ij}$, and $\boldsymbol{h}_{ij}$ are part of the coupling model as they additionally depend on the states and controls of the sending neighbors $j \in \mathcal{N}_{i\leftarrow}$.

III. DISTRIBUTED SOLUTION

The distributed solution of the optimization problem (1) can be carried out using ADMM. Since the states $\boldsymbol{x}_j$ and controls $\boldsymbol{u}_j$ of the sending neighbors $j \in \mathcal{N}_{i\leftarrow}$ are not directly accessible in distributed computation, local copies $\bar{\boldsymbol{x}}_{ji}$ and $\bar{\boldsymbol{u}}_{ji}$ are introduced that serve as additional optimization variables in the agent $i \in \mathcal{V}$. The consistency constraints

$$\boldsymbol{0} = \boldsymbol{z}_i - \begin{bmatrix}\boldsymbol{x}_i^\mathsf{T} & \boldsymbol{u}_i^\mathsf{T}\end{bmatrix}^\mathsf{T}, \quad i \in \mathcal{V} \tag{2a}$$

$$\boldsymbol{0} = \boldsymbol{z}_j - \begin{bmatrix}\bar{\boldsymbol{x}}_{ji}^\mathsf{T} & \bar{\boldsymbol{u}}_{ji}^\mathsf{T}\end{bmatrix}^\mathsf{T}, \quad i \in \mathcal{V},\, j \in \mathcal{N}_{i\leftarrow} \tag{2b}$$

with the coordination variables $\boldsymbol{z}_i \in \mathbb{R}^{n_i+m_i}$ ensure that the local copies and the original variables coincide in the optimal solution. Note that the additional optimization variables $\boldsymbol{z}_i$ are required for the decomposition into simpler problems that can be solved by each agent locally. For the sake of a compact notation, the variables are summarized using the stacked vectors

$$\boldsymbol{w}_i = \begin{bmatrix}\boldsymbol{u}_i \\ \begin{bmatrix}\bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji}\end{bmatrix}_{j\in\mathcal{N}_{i\leftarrow}}\end{bmatrix}, \quad \boldsymbol{w} = \begin{bmatrix}\boldsymbol{w}_1 \\ \vdots \\ \boldsymbol{w}_N\end{bmatrix}, \quad \boldsymbol{z} = \begin{bmatrix}\boldsymbol{z}_1 \\ \vdots \\ \boldsymbol{z}_N\end{bmatrix}. \tag{3}$$

The ADMM algorithm uses an augmented Lagrangian formulation [14] to handle the consistency constraints (2). Thereby, the constraints are adjoined to the cost function (1a) using Lagrangian multipliers

$$\boldsymbol{\mu}_i = \begin{bmatrix}\boldsymbol{\mu}_{ii} \\ \begin{bmatrix}\boldsymbol{\mu}_{ji}\end{bmatrix}_{j\in\mathcal{N}_{i\leftarrow}}\end{bmatrix}, \quad \boldsymbol{\mu} = \begin{bmatrix}\boldsymbol{\mu}_1 & \dots & \boldsymbol{\mu}_N\end{bmatrix}^\mathsf{T} \tag{4}$$

of dimension $\boldsymbol{\mu}_{ii} \in \mathbb{R}^{n_i+m_i}$ and $\boldsymbol{\mu}_{ji} \in \mathbb{R}^{n_j+m_j}$. The augmented Lagrangian function is given by

$$\begin{aligned}J_\rho(\boldsymbol{w},\boldsymbol{z},\boldsymbol{\mu}) &= \sum_{i\in\mathcal{V}} V_i(\boldsymbol{x}_i(T)) + \int_0^T l_i(\boldsymbol{x}_i,\boldsymbol{u}_i) \\ &+ \boldsymbol{\mu}_{ii}^\mathsf{T}\left(\boldsymbol{z}_i - \begin{bmatrix}\boldsymbol{x}_i \\ \boldsymbol{u}_i\end{bmatrix}\right) + \frac{\rho}{2}\left\|\boldsymbol{z}_i - \begin{bmatrix}\boldsymbol{x}_i \\ \boldsymbol{u}_i\end{bmatrix}\right\|^2 \\ &+ \sum_{j\in N_{i\leftarrow}} \boldsymbol{\mu}_{ji}^\mathsf{T}\left(\boldsymbol{z}_j - \begin{bmatrix}\bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji}\end{bmatrix}\right) + \frac{\rho}{2}\left\|\boldsymbol{z}_j - \begin{bmatrix}\bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji}\end{bmatrix}\right\|^2 \mathrm{d}\tau\end{aligned} \tag{5}$$

with the penalty parameter $\rho > 0$. Assuming the existence of an optimal solution $(\boldsymbol{w}^*, \boldsymbol{z}^*, \boldsymbol{\mu}^*)$ as well as strong duality, the augmented Lagrangian function (5) satisfies the saddle point condition

$$J_\rho(\boldsymbol{w}^*,\boldsymbol{z}^*,\boldsymbol{\mu}) \le J_\rho(\boldsymbol{w}^*,\boldsymbol{z}^*,\boldsymbol{\mu}^*) \le J_\rho(\boldsymbol{w},\boldsymbol{z},\boldsymbol{\mu}^*)\,. \tag{6}$$

This allows to reformulate the central optimization problem (1) as max-min-problem

$$\max_{\boldsymbol{\mu}} \min_{\boldsymbol{w},\boldsymbol{z}} J_\rho(\boldsymbol{w},\boldsymbol{z},\boldsymbol{\mu}) \tag{7a}$$

$$\text{s.t.} \quad \dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) + \sum_{j\in\mathcal{N}_{i\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\bar{\boldsymbol{x}}_{ji},\bar{\boldsymbol{u}}_{ji}) \tag{7b}$$

$$\boldsymbol{x}_i(0) = \hat{\boldsymbol{x}}_{i,k} \tag{7c}$$

$$\boldsymbol{g}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) = \boldsymbol{0} \tag{7d}$$

$$\boldsymbol{h}_i(\boldsymbol{x}_i,\boldsymbol{u}_i) \le \boldsymbol{0} \tag{7e}$$

$$\boldsymbol{g}_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\bar{\boldsymbol{x}}_{ji},\bar{\boldsymbol{u}}_{ji}) = \boldsymbol{0}\,, \quad j \in \mathcal{N}_{i\leftarrow} \tag{7f}$$

$$\boldsymbol{h}_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\bar{\boldsymbol{x}}_{ji},\bar{\boldsymbol{u}}_{ji}) \le \boldsymbol{0}\,, \quad j \in \mathcal{N}_{i\leftarrow} \tag{7g}$$

$$\boldsymbol{u}_i \in \mathcal{U}_i\,, \quad i \in \mathcal{V} \tag{7h}$$

where the minimization is performed with respect to the primal variables $(\boldsymbol{w},\boldsymbol{z})$ and the maximization is performed with respect to the dual variables $\boldsymbol{\mu}$. Note that the dynamics (7b) as well as the joint constraints (7f) and (7g) are now decoupled from the non-accessible variables $\boldsymbol{x}_j$ and $\boldsymbol{u}_j$ of the sending neighbors.

ADMM solves the optimization problem (7) in a sequential manner. Given appropriate initial values $\boldsymbol{w}^0, \boldsymbol{z}^0, \boldsymbol{\mu}^0$, the algorithm computes in each iteration $q = 1, \dots, q_{\max}$ the three steps

$$\boldsymbol{w}^q = \arg\min_{\boldsymbol{w}} \; J_\rho(\boldsymbol{w},\boldsymbol{z}^{q-1},\boldsymbol{\mu}^{q-1})\,, \quad \text{s.t. (7b)–(7h)} \tag{8a}$$

$$\boldsymbol{z}^q = \arg\min_{\boldsymbol{z}} \; J_\rho(\boldsymbol{w}^q,\boldsymbol{z},\boldsymbol{\mu}^{q-1}) \tag{8b}$$

$$\boldsymbol{\mu}_i^q = \boldsymbol{\mu}_i^{q-1} + \rho\begin{bmatrix}\boldsymbol{z}_i^q - \begin{bmatrix}\boldsymbol{x}_i^q \\ \boldsymbol{u}_i^q\end{bmatrix} \\ \begin{bmatrix}\boldsymbol{z}_j^q - \begin{bmatrix}\bar{\boldsymbol{x}}_{ji}^q \\ \bar{\boldsymbol{u}}_{ji}^q\end{bmatrix}\end{bmatrix}_{j\in\mathcal{N}_{i\leftarrow}}\end{bmatrix}, \quad i \in \mathcal{V}\,. \tag{8c}$$

The dynamics and constraints are considered in the first minimization problem (8a) that also yields the state trajectories $\boldsymbol{x}_i^q$, $i \in \mathcal{V}$. The minimization with respect to the coordination variables $\boldsymbol{z}$ can be solved analytically. The maximization is performed by the steepest ascent rule (8c) with the penalty parameter ρ as step size, since the maximization problem is unbounded for fixed values of the primal variables.

Algorithm 1 Distributed solution for each agent $i \in \mathcal{V}$.

```
1: Compute $\boldsymbol{w}_i^q$ and $\boldsymbol{x}_i^q$ as solution of
   $\min_{\boldsymbol{w}_i} \; V_i(\boldsymbol{x}_i(T)) + \int_0^T l_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + (\boldsymbol{\mu}_{ii}^{q-1})^{\mathsf{T}} \left( \boldsymbol{z}_i^{q-1} - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} \right) + \frac{\rho}{2} \left\| \boldsymbol{z}_i^{q-1} - \begin{bmatrix} \boldsymbol{x}_i \\ \boldsymbol{u}_i \end{bmatrix} \right\|^2 + \sum_{j \in N_{i\leftarrow}} (\boldsymbol{\mu}_{ji}^{q-1})^{\mathsf{T}} \left( \boldsymbol{z}_j^{q-1} - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix} \right) + \frac{\rho}{2} \left\| \boldsymbol{z}_j^{q-1} - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji} \\ \bar{\boldsymbol{u}}_{ji} \end{bmatrix} \right\|^2 \mathrm{d}\tau$
   s.t. $\dot{\boldsymbol{x}}_i = \boldsymbol{f}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) + \sum_{j \in \mathcal{N}_{i\leftarrow}} \boldsymbol{f}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}) \,, \quad \boldsymbol{x}_i(0) = \hat{\boldsymbol{x}}_{i,k}$
        $\boldsymbol{g}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) = \boldsymbol{0} \,, \quad \boldsymbol{h}_i(\boldsymbol{x}_i, \boldsymbol{u}_i) \leq \boldsymbol{0}$
        $\boldsymbol{g}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}) = \boldsymbol{0} \,, \quad \boldsymbol{h}_{ij}(\boldsymbol{x}_i, \boldsymbol{u}_i, \bar{\boldsymbol{x}}_{ji}, \bar{\boldsymbol{u}}_{ji}) \leq \boldsymbol{0}$
        $\boldsymbol{u}_i \in \mathcal{U}_i \,.$
2: Send $\bar{\boldsymbol{x}}_{ji}^q$ and $\bar{\boldsymbol{u}}_{ji}^q$ to the neighbors $j \in \mathcal{N}_{i\leftarrow}$.
3: Compute $\boldsymbol{z}_i^q$ as
   $\boldsymbol{z}_i^q = \frac{1}{1 + |\mathcal{N}_{i\rightarrow}|} \left( \begin{bmatrix} \boldsymbol{x}_i^q \\ \boldsymbol{u}_i^q \end{bmatrix} - \frac{1}{\rho} \boldsymbol{\mu}_{ii}^{q-1} + \sum_{j \in \mathcal{N}_{i\rightarrow}} \begin{bmatrix} \bar{\boldsymbol{x}}_{ij}^q \\ \bar{\boldsymbol{u}}_{ij}^q \end{bmatrix} - \frac{1}{\rho} \boldsymbol{\mu}_{ij}^{q-1} \right) .$
4: Send $\boldsymbol{z}_i^q$ to the neighbors $j \in \mathcal{N}_{i\rightarrow}$.
5: Compute $\boldsymbol{\mu}_i^q$ as
   $\boldsymbol{\mu}_i^q = \boldsymbol{\mu}_i^{q-1} + \rho \begin{bmatrix} \boldsymbol{z}_i^q - \begin{bmatrix} \boldsymbol{x}_i^q \\ \boldsymbol{u}_i^q \end{bmatrix} \\ \left[ \boldsymbol{z}_j^q - \begin{bmatrix} \bar{\boldsymbol{x}}_{ji}^q \\ \bar{\boldsymbol{u}}_{ji}^q \end{bmatrix} \right]_{j \in \mathcal{N}_{i\leftarrow}} \end{bmatrix} .$
6: Send $\boldsymbol{\mu}_{ji}^q$ to the neighbors $j \in \mathcal{N}_{i\leftarrow}$.
7: Stop if $q = q_{\max}$ or
   $\left\| \begin{matrix} \boldsymbol{z}_i^q - \boldsymbol{z}_i^{q-1} \\ \boldsymbol{\mu}_i^q - \boldsymbol{\mu}_i^{q-1} \end{matrix} \right\|_{L\infty} \leq d \left\| \hat{\boldsymbol{x}}_{i,k} - \boldsymbol{x}_{i,\mathrm{des}} \right\| \,, \quad \forall i \in \mathcal{V} \,.$
   Otherwise, set $q \leftarrow q + 1$ and go to Step 1.
```

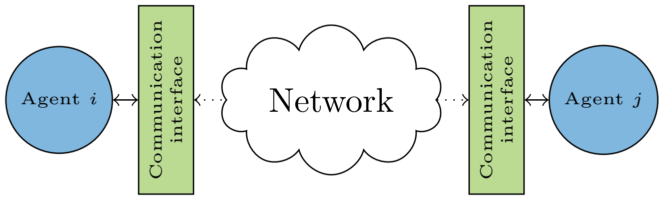

Fig. 2. Each agent uses a communication interface to exchange information with neighboring agents and the central coordinator through the network.

The important point now is that all of the three steps in (8) can be solved locally within each agent $i \in \mathcal{V}$ and only require neighbor-to-neighbor communication to exchange the updated variables. The solution of the central optimization problem (1) is thus computed in a distributed manner. The single steps are outlined in Algorithm 1. Instead of sending the updated multipliers $\boldsymbol{\mu}_{ji}^q$ to the neighbors $j \in \mathcal{N}_{i\leftarrow}$ in Step 6, it is also possible to compute local copies $\boldsymbol{\mu}_{ij}^q$, $j \in \mathcal{N}_{i\rightarrow}$ as the required variables for the steepest ascent are locally available. In Step 7, the stopping criterion from [15] is used. Although other criteria could be used as well, this criterion guarantees exponential decay of the cost function for sufficiently small d under certain assumptions, in particular that the ADMM algorithm is at least R-linearly convergent. Note that the convergence test requires a central coordinator in the network that determines whether the criterion is satisfied for all agents.

Within the distributed optimization algorithm, the solution of the minimization problem in Step 1 constitutes the time-critical part. Especially if the single agents have limited computational power, e.g. due to low-cost hardware, this step requires tailored optimization algorithms. Therefore, the software framework GRAMPC [9] is employed for the numerical solution. It implements an augmented Lagrangian algorithm to handle the nonlinear equality and inequality constraints as well as a projected gradient method that takes the nonlinear dynamics and the input constraints into account. GRAMPC is written in plain C without external libraries, which makes it particularly suited for embedded applications. In order to achieve real-time feasibility of the optimization, only a limited number of iterations are performed in each sampling step and the resulting suboptimal solution is used as warmstart in the next step. Thereby, the suboptimality is reduced in the course of the MPC iterations [16]. Note, however, that the real-time feasibility of the distributed optimization is also affected by the communication effort that is not considered further in this paper.

IV. MODULAR FRAMEWORK

Besides the numerical solution, the practical implementation of distributed model predictive controllers requires considerable effort. Implementing the algorithms for each new system from scratch is highly error-prone. However, reusing existing code requires generic interfaces and a flexible implementation. In the following, the requirements for the modular DMPC framework are described in more detail. The framework should be *generic*, which means that it is independent of the cost functions, dynamics, and constraints that define the optimization problem (1). In addition, it should be *modular* with respect to the number of agents and their couplings, i.e. independent of the structure of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. This also means that the agents only know their direct neighbors and not the complete systems. The modularity additionally allows for plug-and-play functionality. A *flexible* implementation does not prescribe the optimization algorithm or the communication protocol. In particular, it should be possible to use the same problem formulation both for the centralized and the distributed solution. Finally, it should be *extensible* to allow for future modifications, e.g. other distributed optimization algorithms.

The key concept for a generic and modular framework is the definition of agent and coupling models based on the problem formulation (1). The interface agent model provides the agent dynamics $\boldsymbol{f}_i$, the constraints $\boldsymbol{g}_i$ and $\boldsymbol{h}_i$ as well as the cost functions V_i and l_i. Similarly, the interface coupling model contains the coupling dynamics $\boldsymbol{f}_{ij}$ and the joint constraints $\boldsymbol{g}_{ij}$ and $\boldsymbol{h}_{ij}$. Furthermore, the partial derivatives with respect to the states and controls are required for the numerical solution with GRAMPC. The implementations of these interfaces are problem-specific. In practice, however, the agent and coupling models usually have similar structures and merely differ in the parameters, which simplifies the implementation for large-scale systems.

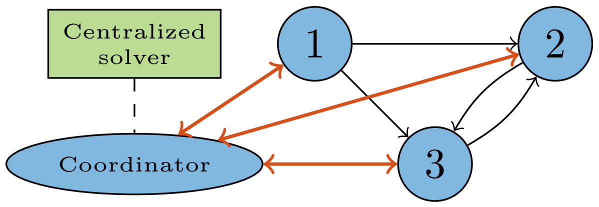

Fig. 3. The central coordinator solves the global optimization problem. All agents exchange information with the coordinator.

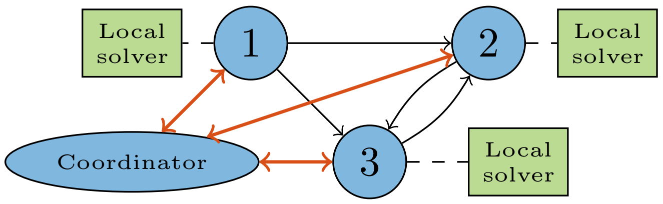

Fig. 4. Each agent solves the local optimization problem and exchanges information with its neighbors. The central coordinator is only needed to trigger the single steps and to determine convergence.

An important aspect of the framework is the independence of the communication protocol. In the early prototyping stage, the centralized and the distributed controller are usually tested running on a single PC without any network communication. The computations are actually distributed in later development stages before, finally, hardware-in-the-loop experiments are performed. Consequently, it should be possible to use the DMPC framework throughout this development process without requiring modifications to the implemented models or optimization algorithms. This goal is achieved by the definition of a communication interface, cf. Figure 2, that provides functions for registering new agents, sending of states, triggering the single ADMM steps, and so forth. Different implementations enable centralized or distributed computations, e.g. based on the UDP protocol. In particular, the control experts can focus on the problem formulation and the optimization, whereas the implementation of the communication can be provided by network experts.

Due to the modularity, the agents are not allowed to access data of the other agents except through the communication interface. In addition, each agent has remote representations of its sending and receiving neighbors that are used to store the local state copies that are required for the ADMM algorithm. A special function is performed by the central coordinator, which is the only node that communicates with all agents. It maintains a list of all agents and their couplings, i.e. a model of the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and can be used to solve the centralized optimization problem, cf. Figure 3. Alternatively, it triggers the single steps of the ADMM algorithm to solve the distributed optimization problem, cf. Figure 4, and determines whether it is converged. Note that triggering the steps from a central node ensures a synchronized execution of the algorithm. Both the centralized and the local solvers construct the optimization problems automatically from the agent and coupling models and use the GRAMPC toolbox for the numerical solution. Thereby, the effort for implementing DMPC for a new system is significantly reduced.

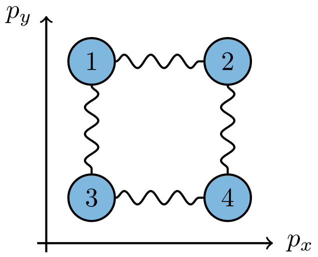

Fig. 5. Each mass in the network is coupled to its neighbors by springs.

V. SIMULATION RESULTS

Two example systems are used to show specific aspects of the framework. The scalability is investigated for a system of concentrated massed that are coupled by springs, whereas the plug-and-play functionality is demonstrated for a system of water tanks that are coupled by pipes. All simulations are performed on a laptop with an i5-8350 CPU with 3.6 GHz.

A. Spring-mass system

At first, a two-dimensional spring-mass system is considered. An example with four agents is visualized in Figure 5. The state and control vectors of each agent $i \in \mathcal{V}$

$$\boldsymbol{x}_i = \begin{bmatrix} \boldsymbol{p}_i \\ \boldsymbol{v}_i \end{bmatrix}, \ \boldsymbol{p}_i = \begin{bmatrix} p_{x,i} \\ p_{y,i} \end{bmatrix}, \ \boldsymbol{v}_i = \begin{bmatrix} v_{x,i} \\ v_{y,i} \end{bmatrix}, \ \boldsymbol{u}_i = \begin{bmatrix} u_{x,i} \\ u_{y,i} \end{bmatrix} \quad (9)$$

correspond to the positions $\boldsymbol{p}_i$ and velocities $\boldsymbol{v}_i$ of the mass i as well as the actuated force $\boldsymbol{u}_i$ in each axis. The dynamics $i \in \mathcal{V}$ are given in the neighbor-affine form as

$$\dot{\boldsymbol{x}}_i = \underbrace{\begin{bmatrix} \boldsymbol{v}_i \\ \frac{1}{m}\boldsymbol{u}_i \end{bmatrix}}_{=:\boldsymbol{f}_i(\boldsymbol{x}_i,\boldsymbol{u}_i)} + \sum_{j \in \mathcal{N}_{i\leftarrow}} \underbrace{\begin{bmatrix} \boldsymbol{0} \\ \frac{c}{m}\left(1 - \frac{\delta_0}{\|\boldsymbol{p}_j - \boldsymbol{p}_i\|}\right)(\boldsymbol{p}_j - \boldsymbol{p}_i) \end{bmatrix}}_{=:\boldsymbol{f}_{ij}(\boldsymbol{x}_i,\boldsymbol{u}_i,\boldsymbol{x}_j,\boldsymbol{u}_j)} \quad (10)$$

with the mass $m = 7.5\,\mathrm{kg}$, the force constant $c = 0.5\,\mathrm{N\,m^{-1}}$, and the distance between the agents $\delta_0 = 0.5\,\mathrm{m}$, for which the springs are unstressed. The cost function (1a) for agent $i \in \mathcal{V}$ is defined by

$$V_i(\boldsymbol{x}_i(T)) = \|\boldsymbol{x}_i(T) - \boldsymbol{x}_{i,\mathrm{d}}\|_{\boldsymbol{P}}^2 \quad (11a)$$

$$l_i(\boldsymbol{x}_i(\tau), \boldsymbol{u}_i(\tau)) = \|\boldsymbol{x}_i(\tau) - \boldsymbol{x}_{i,\mathrm{d}}\|_{\boldsymbol{Q}}^2 + \|\boldsymbol{u}_i(\tau)\|_{\boldsymbol{R}}^2 \quad (11b)$$

with the desired setpoint

$$\boldsymbol{x}_{i,\mathrm{d}} = \begin{bmatrix} p_{x,i,\mathrm{d}} & p_{y,i,\mathrm{d}} & 0 & 0 \end{bmatrix}^\mathsf{T} \quad (12)$$

and the diagonal weighting matrices

$$\boldsymbol{P} = \boldsymbol{Q} = \mathrm{diag}(1, 1, 0.5, 0.5), \ \boldsymbol{R} = \mathrm{diag}(0.01, 0.01). \quad (13)$$

SI units are omitted in (13) for the sake of readability. The optimization uses a horizon $T = 3\,\mathrm{s}$ with $N_\mathrm{hor} = 20$ discretization points and a sampling time of $\Delta t = 0.1\,\mathrm{s}$. The penalty parameter of the ADMM algorithm is set to $\rho = 0.5$. The initial positions of the agents are set to random deviations from their desired positions $p_{x,i,\mathrm{d}}, p_{y,i,\mathrm{d}}$, whereby a uniform distribution between $[-0.35\,\mathrm{m}, 0.35\,\mathrm{m}]$ is employed. GRAMPC is configured for a constant number of five gradient iterations and the ADMM algorithm is limited

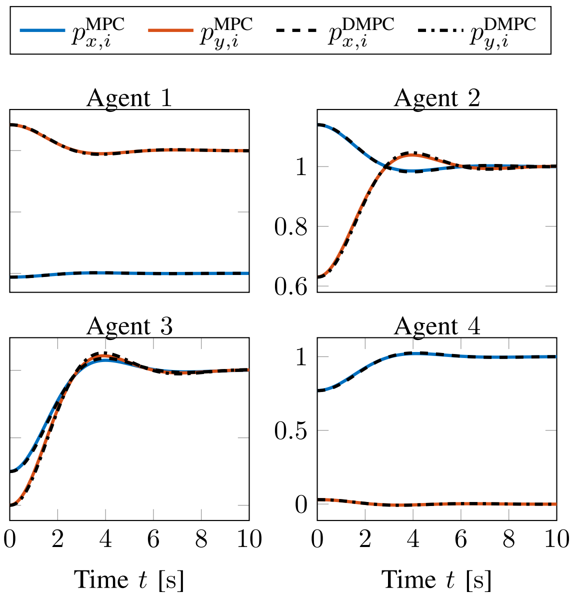

Fig. 6. Position trajectories of the spring-mass-system with four agents computed with the centralized and the distributed MPC.

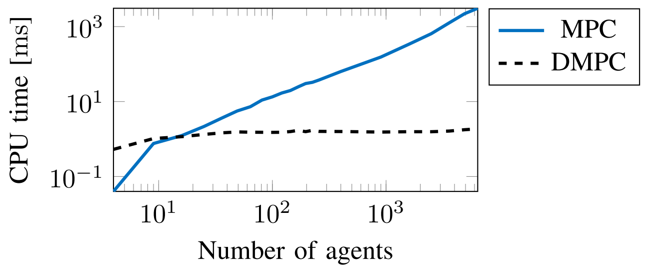

Fig. 7. Comparison of the mean computation times for centralized (total) and distributed MPC (per agent) for the scalable spring-mass-system.

either by the stopping criterion with $d = 0.025$ or by the maximum number of $q_{\max} = 5$ iterations. Figure 6 shows the resulting trajectories for an example system with four agents. As can be seen, all agents return to their desired positions. Furthermore, the solution trajectories of the distributed MPC fit the ones of the centralized MPC very well.

To show the scalability potential of the framework, the number of agents is increased and the total computation time for the centralized MPC is compared to the required time per agent of the distributed MPC. The average computation time over the first 25 time steps is taken as measurement. Figure 7 visualizes the results for agent numbers between 4 and 6400, i.e. a grid structure with 80 × 80 masses. The mean computation time per agent of the DMPC stays approximately constant at $1.5\,\mathrm{ms}$, whereas the computational demand of the centralized solution increases heavily. This shows the advantage of distributed control for large-scale systems. In contrast to centralized control, distributed MPC can handle systems with thousands of agents.

B. Water tank network

A network of coupled water tanks is used to demonstrate the plug-and-play functionality, i.e. the option to add or remove agents and couplings during the operation. The example setup is visualized in Figure 8, where the dashed lines indicate that the couplings to the second tank are added and removed. Only the first agent has a controllable inflow u_1 and only the fourth agent has a desired water level $x_{4,\mathrm{d}} = 2\,\mathrm{m}$. In addition, a constant disturbance $\epsilon_4 = 0.01\,\mathrm{m\,s^{-1}}$ causes the last tank to continuously lose water. The corresponding agent dynamics for $i \in \mathcal{V}$ can be written in the form of

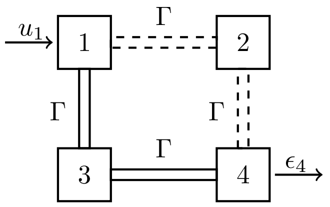

Fig. 8. Configuration of the network of water tanks coupled by pipes. The couplings to tank 2 are added and removed during the simulation.

$$\dot{x}_i = \underbrace{\frac{1}{A} p_i u_i + \epsilon_i}_{=: f_i(x_i, u_i)} + \sum_{j \in \mathcal{N}_{i\leftarrow}} \underbrace{\frac{1}{A} \Gamma(x_i, x_j)}_{=: f_{ij}(x_i, u_i, x_j, u_j)} \tag{14}$$

with the water level x_i, the tank area $A = 0.0144\,\mathrm{m}^2$, and the function

$$\Gamma(x_i, x_j) = \alpha \,\mathrm{sign}\,(x_j - x_i) \sqrt{2g|x_j - x_i|} \tag{15}$$

that describes the flow through a pipe with diameter $\alpha = 0.006\,\mathrm{m}$ and the gravitational acceleration $g = 9.81\,\mathrm{m\,s^{-2}}$, cf. [17] for the physical dimensions. The binary parameter $p_i \in \{0, 1\}$ is introduced to use the same model for all agents and is set to one for the first agent and to zero for all others. Similarly, the disturbance ϵ_i is zero for all agents except the last one. As the function (15) is not differential for $x_j - x_i = 0\,\mathrm{m}$, a polynomial fit is used for $|x_j - x_i| < 0.01\,\mathrm{m}$. The cost function (1a) of agent $i \in \mathcal{V}$ is defined by

$$V_i(x_i(T)) = P_i(x_i(T) - x_{i,\mathrm{d}})^2 \tag{16a}$$
$$l_i(x_i(\tau), u_i(\tau)) = Q_i(x_i(\tau) - x_{i,\mathrm{d}})^2 + R_i u_i(\tau)^2 \tag{16b}$$

with the weighting parameters $P_4 = 1\,\mathrm{m^{-2}}$, $Q_4 = 1\,\mathrm{m^{-2}\,s^{-1}}$ and $R_1 = 1\,\mathrm{s\,m^{-6}}$. The other weights are set to zero. Furthermore, all agents are subject to the state constraint

$$h_i(x_i(\tau), u_i(\tau)) = x_i(\tau) - 3\,\mathrm{m} \leq 0 \tag{17}$$

that limits the maximum water height in the tank. The time horizon of the optimization is set to $T = 1\,\mathrm{s}$ with $N_{\mathrm{hor}} = 20$ sampling points. In each MPC step with the sampling time $\Delta t = 10\,\mathrm{ms}$, the ADMM algorithm computes a fixed number of $q_{\max} = 10$ iterations with the penalty parameter $\rho = 10$. Furthermore, GRAMPC is configured to calculate seven gradient iterations for the approximate solution of the minimization problems. The stopping criterion is not directly applicable for this example, since not each of the water tanks has a desired setpoint.

Figure 9 shows the resulting trajectories for the centralized and the distributed MPC. The simulation time can be split in three parts. During the first 3 s, the second tank is

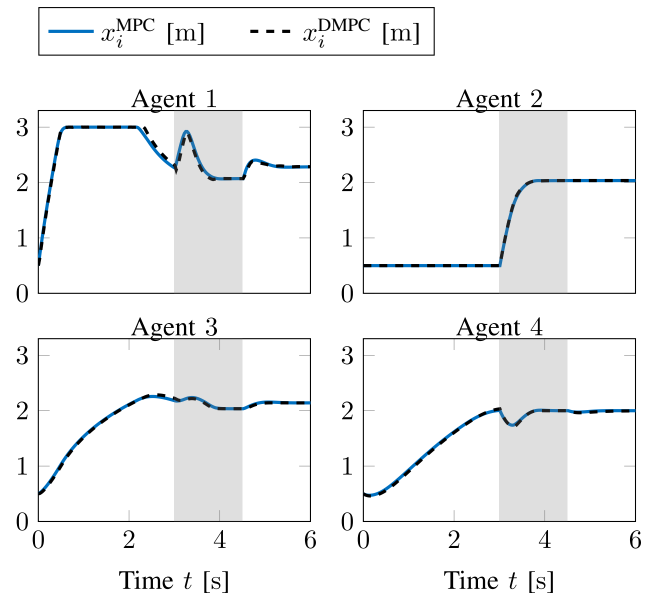

Fig. 9. Trajectories of the water tank network computed with the centralized and the distributed solution. The gray area indicates the plug-and-play functionality, where the second water tank is coupled with the network.

not connected to the other tanks. The first tank uses the controllable inflow u_1 to rise the water height until the constraint (17) is reached. Due to the couplings, this causes the levels of the third and the fourth tank to increase. As soon as the necessary amount of water is in the network, the first tank reduces its inflow and approaches the same water level as the other tanks. In the second part for $t \in [3\,\mathrm{s},\, 4.5\,\mathrm{s}]$, the couplings of the second tank are included. Since the water level x_2 is low, the last tank loses water and the first tank has to increase its inflow again in order to compensate this deviation. The couplings of the second tank are removed again at $t = 4.5\,\mathrm{s}$. Due to the constant disturbance ϵ_4 that acts on the last tank, the other tanks require slightly higher water levels for a stationary compensation.

Throughout the simulation, the average computation time of the DMPC is $1\,\mathrm{ms}$ per agent, which is significantly below the sampling time of $10\,\mathrm{ms}$, i.e. real-time feasibility is given. The plug-and-play functionality is a direct consequence of the modular structure of the framework. In case of the distributed MPC, only the second tank and its two neighbors have to reconfigure their local solvers. The third tank can continue as before. The possibility to modify the graph structure online is an important aspect for the application of DMPC to large-scale systems as this flexibility is difficult to realize with a centralized control strategy.

VI. Conclusions

In this paper, a modular framework for distributed model predictive control of nonlinear neighbor-affine systems is presented. It separates the aspects of problem formulation, numerical optimization, and communication and thereby reduces the effort for implementing DMPC for new systems. The implementation using C++ and the integration of GRAMPC as solver for the local minimization problems are important steps towards real-time feasibility and thus the practical applicability of (nonlinear) DMPC. As shown by the simulation examples, the ADMM-based approach scales well with the number of agents and allows for dynamic reconfigurations of the distributed system in the sake of plug-and-play.

Future work concerns the implementation of communication interfaces for distributed computation, e.g. based on UDP. With regard to the numerical solution, the adaptation of the penalty parameter ρ and the decomposition with approximated neighbor dynamics [18] are considered. Furthermore, it is planned to implement a MATLAB interface for the modular framework and to make it available as open-source software.

References

[1] E. Camponogara, D. Jia, B. Krogh, and S. Talukdar, “Distributed model predictive control,” *Control Systems Magazine*, vol. 22, pp. 44–52, 2002.

[2] J. Maestre and R. Negenborn, *Distributed Model Predictive Control Made Easy*. Springer, 2014.

[3] D. Bertsekas and J. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*. Belmont (USA): Athena Scientific, 1997.

[4] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. Foundations and Trends® in Machine Learning, 2011, vol. 3, no. 1.

[5] B. Houska, J. Frasch, and M. Diehl, “An augmented Lagrangian based algorithm for distributed nonconvex optimization,” *SIAM Journal on Optimization*, vol. 26, no. 2, pp. 1101–1127, 2016.

[6] H. Scheu and W. Marquardt, “Sensitivity-based coordination in distributed model predictive control,” *Journal of Process Control*, vol. 21, pp. 715–728, 2011.

[7] B. Houska, H. J. Ferreau, and M. Diehl, “ACADO toolkit - an open-source framework for automatic control and dynamic optimization,” *Optimal Control Applications and Methods*, vol. 32, pp. 298–312, 2011.

[8] J. Kalmaria, J. Backman, and A. Visala, “A toolkit for nonlinear model predictive control using gradientprojection and code generation,” *Control Engineering Practice*, vol. 39, pp. 56–66, 2015.

[9] 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),” *Optimization and Engineering*, vol. 20, Issue 3, p. 769–809, Jan. 2019, http://sourceforge.net/projects/grampc.

[10] S. Riverso, A. Battocchio, and G. Ferrari-Trecate, “PnPMPC toolbox,” 2013. [Online]. Available: sisdin.unipv.it/pnpmpc/pnpmpc.php

[11] S. Riverso, M. Farina, and G. Ferrari-Trecate, “Plug-and-play model predictive control based on robust control invariant sets,” *Automatica*, vol. 50, pp. 2179–2186, 2014.

[12] S. Riverso and G. Ferrari-Trecate, “Plug-and-play distributed model predictive control with coupling attenuation,” *Optimal Control Applications and Methods*, vol. 36, no. 3, pp. 292–305, 2015.

[13] O. Gäfvert, “A practical guide to distributed model predictive control,” 2014. [Online]. Available: github.com/olivergafvert/dmpc

[14] D. P. Bertsekas, “Multiplier methods: A survey,” *Automatica*, vol. 12, no. 2, pp. 133 – 145, 1976.

[15] A. Bestler and K. Graichen, “Distributed model predictive control for continuous-time nonlinear systems based on suboptimal ADMM,” *Optimal Control Applications and Methods*, vol. 40, pp. 1–23, 2019.

[16] K. Graichen and A. Kugi, “Stability and incremental improvement of suboptimal MPC without terminal constraints,” *IEEE Transactions on Automatic Control*, vol. 55, no. 11, pp. 2576–2580, 2010.

[17] S. Hentzelt and K. Graichen, “Dual decomposition of optimal control problems with coupled nonlinear dynamics,” in *Proc. ECCOMAS*, Vienna, Austria, 2012.

[18] S. Hentzelt and K. Graichen, “An Augmented Lagrangian Method in Distributed Dynamic Optimization Based on Approximate Neighbor Dynamics,” in *Proc. SMC*, 2013, pp. 571–576.