Received: 14 December 2018 | Revised: 2 January 2020 | Accepted: 28 February 2020

DOI: 10.1002/oca.2599

RESEARCH ARTICLE

WILEY

Model predictive control of a collaborative manipulator considering dynamic obstacles

Maximilian Krämer | **Christoph Rösmann** | **Frank Hoffmann** | **Torsten Bertram**

Institute of Control Theory and Systems Engineering, TU Dortmund University, Dortmund, Germany

Correspondence
Maximilian Krämer, Institute of Control Theory and Systems Engineering, Otto-Hahn-Str. 8, D-44227 Dortmund, Germany.
Email: maximilian.kraemer@tu-dortmund.de

Summary

Collaborative robots have to adapt its motion plan to a dynamic environment and variation of task constraints. Currently, they detect collisions and interrupt or postpone their motion plan to prevent harm to humans or objects. The more advanced strategy proposed in this article uses online trajectory optimization to anticipate potential collisions, task variations, and to adapt the motion plan accordingly. The online trajectory planner pursues a model predictive control approach to account for dynamic motion objectives and constraints during task execution. The prediction model relates reference joint velocities to actual joint positions as an approximation of built-in robot tracking controllers. The optimal control problem is solved with direct collocation based on a hypergraph structure, which represents the nonlinear program and allows to efficiently adapt to structural changes in the optimization problem caused by moving obstacles. To demonstrate the effectiveness of the approach, the robot imitates pick-and-place tasks while avoiding self-collisions, semistatic, and dynamic obstacles, including a person. The analysis of the approach concerns computation time, constraint violations, and smoothness. It shows that after model identification, order reduction, and validation on the real robot, parallel integrators with compensation for input delays exhibit the best compromise between accuracy and computational complexity. The model predictive controller can successfully approach a moving target configuration without prior knowledge of the reference motion. The results show that pure hard constraints are not sufficient and lead to nonsmooth controls. In combination with soft constraints, which evaluate the proximity of obstacles, smooth and safe trajectories are planned.

KEYWORDS

human-robot collaboration, input delay, model predictive control, online trajectory optimization

1 | INTRODUCTION

Collaborative robots are small and medium-size robotic manipulators, much lighter and less powerful than the conventional industrial robots in large assembly lines. They are easily set up for different tasks in small-scale, nonautomated

production lines sharing the environment with humans as their lightweight structure and collision awareness mediate the risk to cause harm to people. Their range of application is limited by the relatively low payload of typically a few kilograms only. These small-sized and low powered robots perform simple repetitive tasks, such as part handling in human-robot collaboration, to combine the repeatability of robots with the dexterity of humans. Human-Robot collaboration is a viable alternative for semiautomation in production lines for which full-scale process automation is economically nonfeasible for small and medium lot sizes. However, until now, collaborative robots are preferably used in a cooperative sense to work on their tasks in parallel to humans, only exploiting the benefit of quick task reconfigurations. An actual collaboration where the human and robot truly interact as a part of the process is comparatively rarely seen. This article utilizes online trajectory optimization based on model predictive control as a step towards a truly collaborative robotic framework.

1.1 | Online trajectory planning

Trajectory planning provides the link between an abstract task planning layer and the execution of robot reference motions governed by underlying tracking controllers. Depending on the field of application, the requirements for trajectory planning vary. In the following, trajectory planning for collaborative robots is categorized into three classes. The first class comprises approaches that perform offline planning in static environments with fixed target configurations and thus only serve for cooperation tasks.[1,2] Following a fixed trajectory is not practical for collaboration. Imagine, for example, a task where a human coworker assembles multiple pieces into one and then hands it over to a collaborative robot that performs verifications and final packaging. Not only moving obstacles inside of the robot's workspace and the potential risk of collisions require special measures for trajectory planning, but also the uncertainty of task components, as in this example, the hand-over configuration. A step towards collision avoidance for approaches of this class can be achieved by preprocessing joint velocity references inside of the tracking controller.[3] The algorithm searches for a joint velocity that most closely matches the desired one, in the sense of the quadratic Euclidean norm, subject to constraints on the relative speed between approaching bodies. However, since this only modifies the execution of motions, but not the actual planning, the robot reacts to the environment in a way that is not covered by the plan. Furthermore, the approach can reduce joint velocities thus far that the robot entirely stops. This makes it similar to built-in collision detection methods and leads to task resets and process delays. In general, workspace surveillance and online trajectory planning become mandatory for collaborative robots that closely interact with humans in the shared environment.

Approaches of the second class use a preplanned path and modify the time parametrization of the corresponding trajectory online.[4-8] By scaling the time parametrization, the robot responds to a dynamic environment by either slowing down or speeding up its motion, enabling a time-shared collaboration. Additional constraints consider velocity, acceleration, and jerk limitations of feasible motions. However, since the path is fixed, these approaches cannot react to changing waypoints, and the robot might freeze and delays the process. Beckert et al[9] use planned failsafe and recovery maneuvers in these situations to allow the robot to leave its path temporarily.

The third class consists of approaches that modify both the path as well as the time parametrization of a trajectory providing the most flexibility and full collaboration. Online planning might result from the advancement of established offline approaches to achieve real-time capabilities. Wang et al[10] present a grid-based path planning approach that only considers a subset of the search space to reduce computation time. Kohrt et al[11] propose a similar method that restricts the search space to Voronoi regions. The trajectory then emerges from a time parametrization along the path, for example, based on trapezoidal velocity profiles or piecewise polynomial interpolation with imposed velocities and accelerations.[12] Another approach by Yoshida et al[13] performs planning and execution asynchronously. Upon changes in the environment, a subroutine replans a trajectory based on the current initialization and passes it to the execution process. However, this approach places demands on the dynamics of the environment that are not always given.

In contrast to conventional approaches that separate path and trajectory planning, trajectory optimization plans the trajectory as a whole. Trajectory optimization can be seen as shaping an initial path and time parametrization by iteratively improving a solution set with respect to given objectives and constraints. Formulating trajectory planning as an optimal control problem opens an intuitive way to incorporate task objectives and limitations into the planning algorithm by quantitative constraints and cost functions. To cope with the burden of computational complexity, online trajectory planning often considers a limited time horizon that covers the immediate future. This short optimized trajectory is then sent to an execution process preempting the previous one. If the cycle time for planning is less or equal to the discretization of the planned trajectory, only the first element of the trajectory is effective, making it similar to the Model Predictive Control approach. Model predictive control repeatedly solves an optimal control problem on a finite horizon and applies

the first element of the optimal control sequence to the process.[14] Sending only the first element of the planned trajectory combines planning and execution. Model predictive control can explicitly consider the dynamics of low-level joint motion controllers in a prediction model. The closed-loop nature of model predictive control additionally accounts for model uncertainties and disturbances already during planning and not only during execution. Ide et al,[15] Zube,[16] Buizza Avanzini et al[17] propose a model predictive controller to control mobile manipulators, also considering the movement of the robot's base. Because of the predictive abilities, model predictive control for robotic manipulators often establishes a look ahead tracking controller, for which future references are known in advance.[18-21]

Besides optimal control, reinforcement learning also utilizes the dynamic programming theory and is therefore closely related. From a motion planning perspective, reinforcement learning generates a policy for motion commands, which maximize a reward function for collision-free operation, depending on the state of the robot and the environment. For example, Sangiovanni et al[22] perform planning via reinforcement learning by including the robot's and obstacle's states into the search space and learn a policy, which commands joint speeds. Reinforcement learning offers the advantage that no system equations have to be solved online, as the model is mainly data-based. In return, the dimensions of the search space increase through additional environmental states, which can prolong the learning phase. Offline learning in simulators is a good alternative to speed up learning, but this usually requires a model. Optimal control benefits from a precise model, that can be created and simulated with comparatively less effort and avoids preliminary learning steps. This applies, for example, to cascaded structures such as those of a robotic manipulator, in which lower-level joint control loops decouple and linearize the system dynamics.

1.2 | Related work

The literature reports several approaches that similarly address the problem. Wang et al,[23] Diehl et al,[24] Zhao et al,[25] Zube[16] introduce a model predictive controller for online trajectory optimization on manipulators as well, but their analysis is merely based on simulations ignoring model uncertainty and influences of real applications. Wang et al,[26] Buizza Avanzini et al[17] report experiments on a physical robot. However, Wang et al[26] only focuses on a planar workspace, whereas Buizza Avanzini et al[17] pursues a different approach for obstacle avoidance, to be discussed later.

A dynamic model predicts the future arm motion for a designated sequence of control inputs. In the context of model predictive control, there is a tradeoff between model accuracy and computational effort. Model predictive control requires not only to predict but also to optimize the future trajectory within the time period of a control cycle. In many applications of model predictive control, the prediction model comprises the entire nonlinear robot forward dynamics to relate applied joint torques to the evolution of joint position, velocity and acceleration,[20,24,25] leading to a nonlinear multiple-input-multiple-output system. Even though these models are attractive from a theoretical perspective as they capture the arm dynamics, they face some problems in practice. First of all, the computational demand for solving the nonlinear differential equations in real-time prohibits the combination with complex distance and forward kinematic computations. Also, collaborative robots usually only provide an interface to track position or velocity reference trajectories with limited or no access to the underlying low-level torque controllers to not undermine safeguards.

Partitioning the overall control task into smaller subtasks for current, torque, velocity, and position control reduces the complexity of control design. In the cascaded control architecture, the inner control loops can be modeled by linearized and decoupled dynamics.[12] The concept of cascaded control loops is thus rather common in robot motion control and is also applicable to prediction models.[27] Wilson et al[28] propose a Proportional-Integral-Differential controller for the inner control loop and a model predictive controller for the outer cascade to control the motion of a two-link robot arm. The motion model in References 18,19 and 29 is constituted by a series of integrators obtained from feedback linearization, which further simplifies the simulation of the prediction model. However, these approaches also assume low-level interfaces to regulate the joint actuators on either the current or torque level, which are encapsulated from the user on collaborative robots. This work rests upon an identified and validated linear decoupled prediction model that assumes a plausible structure of the underlying inner joint motion control loop of the robot. Zube[16] propose an integrator type model for motion prediction but do not report its accuracy in prediction or simulation.

Anticipation and avoidance of obstacles within the online motion planning are crucial prerequisites for interruption-free and safe human-robot collaborations. Obstacles that move during the robot motion are considered as fully dynamic, whereas semistatic obstacles merely change their pose between task executions. Obstacle avoidance mainly bases either on distance values between two objects calculated, for example, by an Euclidean distance function as by Lumelsky,[30] Gilbert et al,[31] or a binary collision check. The binary collision check is sufficient for discrete path planning

algorithms like Rapidly-Exploring-Random-Trees,[32] Probabilistic Roadmap Planner,[33] or Expansive Search Trees,[34] to determine a collision-free path from start to goal in a static environment. On the other hand, distance functions provide more quantitative information and are thus applicable in a broader context but are computationally more demanding. Cascio et al[35] incorporate an implicit distance function that is formulated as a separate optimization problem inside of the overall optimization problem for trajectory planning. In trajectory planning via an optimal control problem, obstacle avoidance can be achieved by an objective function that maximizes the separation between robot and obstacle bodies.[36] The literature proposes different implementations of objective functions, for example, functions that grow hyperbolically as the separation distance tends towards a minimum distance threshold.[16,36-38] However, the drawback of pure hyperbolic functions is twofold. First, a nonvanishing obstacle cost causes an unnecessary conflict between goal accuracy and separation distance costs, even if the robot has sufficient clearance from the obstacle. Second, a hyperbolic objective function mimics a barrier function that is used to realize hard constraints (HC). While this is useful for a solver that cannot handle HC natively, it misses the purpose of more powerful solvers like interior-point frameworks. To safeguard obstacle avoidance, additional inequality constraints maintain a guaranteed minimal separation. Wang et al[23,26] impose inequality constraints with respect to a distance metric to achieve obstacle avoidance. However, Wang et al[23] focuses on simulation only, whereas Wang et al[26] restricts the approach to a planar workspace. Buizza Avanzini et al,[17] Zube[16] achieve obstacle avoidance through explicit constraints and dedicated terms in the cost function. The approaches assume statics obstacles and do not provide for self-collision avoidance. Furthermore, the implementation of Zube[16] only considers a few isolated points on the robot for obstacle avoidance, rather than the entire body.

The geometric representation of robot and obstacle bodies should be flexible to cover a wide variety of object shapes; on the other hand, complex geometries put an unnecessary burden on computations of pairwise distances between bodies. Landry et al[39] operate with polyhedrons and utilize Farkas lemma in conjunction with back-face culling to reduce computational complexity. However, the algorithm does not run in real-time due to the high complexity of the additional constraints and optimization parameters introduced by Farkas lemma. Several approaches aim to reduce the complexity, either by safety planes,[40] or projection of the workspace onto a plane thus rendering distance computation as a 2D problem.[26] The ordinary Euclidean distance between a pair of points is the least complex metric. To model spherical objects this distance metric is merely augmented with a safety margin.[41] A spheres-only approach comes with the drawback that longitudinally shaped objects either have to be covered by a large number of overlapping spheres or that a single sphere enclosing the entire object contains a large volume of free space. Line swept spheres introduced by Larsen et al[42] nowadays constitute state of the art for obstacle representation.[3,16,35] A swept sphere, also denoted as a capsule, indicates the volume that a sphere covers during its translation along a straight line. Its shape is similar to a cylinder, but the distance metric is more straightforward to compute for the two spherical end sections.

Solving the optimal control problem in real-time constitutes a significant challenge in online trajectory optimization. The complexity of numerical optimization also depends on the representation of the optimal control problem. The CasADi framework by Andersson et al[43] can be used to save computation time as it replaces numerical differentiation by automatic differentiation. The extra computations to set up and prepare the optimization problem in symbolic form are usually negligible for static problems, but become noticeable for dynamic problems.[44]

1.3 | Contribution

This contribution introduces a novel model predictive control scheme based on a hypergraph for online trajectory optimization of collaborative robots. The approach considers motion constraints such as joint state limits in position and velocity, as well as full robot self-collision avoidance and avoidance of moving obstacles in the entire workspace. The proposed cascaded control architecture offers the advantage that the model predictive controller with joint velocity control actions explicitly incorporates constraints on joint velocities in motion planning. The common alternative of planning and commanding also joint configurations comes with the drawback that the joint velocities are changed by the tracking controllers to minimize the joint configuration error and thus are neither optimized nor constrained. The approach achieves obstacle avoidance with minimum separation by inequality constraints, and clearance from obstacles by a cost term in the cost function of the optimal control problem. The cost term grows quadratically for a separation distance below an activation threshold and is zero otherwise.

High priority tasks for safety-related features might delay the execution of motion commands in the low-level software.[45] Input delays affect safe operation and possibly cause collisions as they limit the actual rate and timeliness of control interventions. This contribution explicitly analyzes model accuracies and time delays by utilizing model

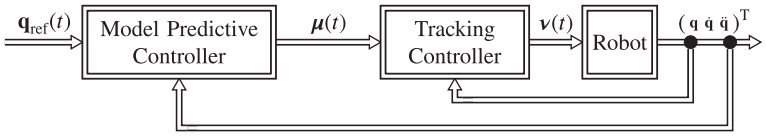


FIGURE 1 The model predictive controller regulates the joint configuration $\mathbf{q}(t)$ to the target configuration $\mathbf{q}_{\mathrm{ref}}(t)$. The reference velocity $\boldsymbol{\mu}(t)$ is the control input to the inner velocity tracking controllers that in turn generate joint actuation torques $\boldsymbol{\nu}(t)$

identification and validation on a real robot. The deduced prediction model within the model predictive controller accounts for input delays caused by dead times and computation times following the approach of Grüne and Pannek.[14] The effectiveness of the model predictive control scheme is validated and analyzed in robotic pick-and-place experiments on Universal Robots[46] collaborative robot UR10 regarding computation time, constraint violations, and trajectory smoothness. The experiments consider variations of task parameters, in particular, a dynamic goal pose as well as different obstacle scenarios including a person. In a previous publication, the authors propose a hypergraph structure to represent the optimal control problem.[44] The computation times equal CasADi for small to mid-sized optimization problems but exhibit a much shorter period of preparation. The introduced approach in this article benefits from efficient preparation as the structure of the optimization problem varies over time, as moving obstacles integrate dynamics into the number of obstacle avoidance constraints.

1.4 | Outline

The next section analyzes the overall control structure and establishes an appropriate robot model for prediction within the model predictive control framework. Section 3 introduces the model predictive controller for online trajectory optimization. It formulates the motion planning task as an underlying optimal control problem with a designated obstacle representation for robot self-collision and obstacle avoidance. Section 3 also provides an insight into the hypergraph structure of the optimal control problem as well as the closed-loop control algorithm with modifications to cope with input delays. Section 4 is concerned with the practical experiments on a Universal Robot UR10 collaborative robot to evaluate the new concept under real-time conditions. This includes the identification and validation of the robot motion model as well as evaluations with a moving target configuration and dynamic obstacles. It analyzes obstacle avoidance strategies concerning computation time, smoothness, and separation distance. Section 5 concludes the pros and cons of the proposed approach and provides an outlook on future research.

2 | SYSTEM OVERVIEW

Figure 1 depicts the cascaded control loop architecture, in which the outer loop contains the model predictive controller, and the inner loop is composed of tracking controllers for velocity references. The model predictive controller plans the sequence of control actions for an optimal point-to-point motion from current joint configuration $\mathbf{q}(t)$ to the target configuration $\mathbf{q}_{\mathrm{ref}}(t) \in \mathbb{R}^N$ for a robot with N degrees of freedom. Control actions $\boldsymbol{\mu}(t) \in \mathbb{R}^N$ constitute the input to the velocity tracking controllers of the inner cascade. The observed joint states $\mathbf{q}(t) \in \mathbb{R}^N$, $\dot{\mathbf{q}}(t)$ and $\ddot{\mathbf{q}}(t)$ constitute the available feedback for the outer loop model predictive controller. The underlying tracking controllers generate the torques $\boldsymbol{\nu}(t) \in \mathbb{R}^N$ commanded to the robot actuators.

The multiple-input-multiple-output robot joint dynamics with interjoint disturbances are decoupled by high gear reductions and centralized tracking controllers of the inner cascade.[12] Usually, these controllers are also able to compensate disturbances caused by collisions or sudden load changes, but in that case the safety software would interrupt further execution. This approach assumes sufficient disturbance rejection on joint level by the inner tracking controllers, which are modeled by N decoupled, linear systems $\mathbf{f}_i : \mathbb{R}^M \times \mathbb{R} \to \mathbb{R}^M$ of order $M \in \mathbb{N}$ with dead times $T_{\mathrm{D},i} \in \mathbb{R}^+$ for $i = 1, 2, \ldots, N$:

$$\begin{aligned} \dot{\mathbf{x}}_i(t) &= \mathbf{f}_i\left(\mathbf{x}_i(t), u_i(t - T_{\mathrm{D},i})\right) \\ \mathbf{x}_i(t) &:= \left(x_{1,i}(t)\ x_{2,i}(t)\ \ldots\ x_{M,i}(t)\right)^{\mathrm{T}}. \end{aligned} \tag{1}$$

Please note the introduction of the reference velocity $u_i \in \mathbb{R}$ as the open-loop variant of μ_i to highlight the difference between the open-loop controls during optimization and the final commanded closed-loop control. Usually, a second-order system to model the velocity tracking controller in conjunction with an integrator are sufficient ($M = 3$), as the former can successfully model rise time as well as possible overshoot, and the latter transfers velocities to joint angles. The poles $p_{i,1} \in \mathbb{C}$ and $p_{i,2} \in \mathbb{C}$ of the second-order system are identified in conjunction with dead times $T_{\mathrm{D},i}$ in Section 4.2. In the subsequent analysis, the third-order model is compared with less complex time-delayed state space models of orders $M = 1$ and $M = 2$. In (1), the first state component $x_{1,i}(t)$ denotes the joint configuration. For a second-order state space model ($M = 2$), the second state component $x_{2,i}(t)$ denotes the actual joint velocity. For a third-order state space model ($M = 3$), the third state component $x_{3,i}(t)$ denotes the joint acceleration. While today's robots usually have accurate and low-noise encoders to measure joint angles, it may be necessary to determine other states such as velocity or acceleration using an observer. The state space model $\mathbf{f} : \mathbb{R}^{NM} \times \mathbb{R}^{N} \times \mathbb{R}^{N} \to \mathbb{R}^{NM}$ for all N subsystems is then given by:

$$
\begin{aligned}
\dot{\mathbf{x}}(t) = \mathbf{f}\left(\mathbf{x}(t), \mathbf{u}(t), \mathbf{T}_{\mathrm{D}}\right) &:= \mathbf{C}\begin{pmatrix} \mathbf{f}_1\left(\mathbf{x}_1(t), u_1(t - T_{\mathrm{D},1})\right) \\ \mathbf{f}_2\left(\mathbf{x}_2(t), u_2(t - T_{\mathrm{D},2})\right) \\ \vdots \\ \mathbf{f}_N\left(\mathbf{x}_N(t), u_N(t - T_{\mathrm{D},N})\right) \end{pmatrix} \\
\mathbf{x}(t) &= \mathbf{C}\begin{pmatrix} \mathbf{x}_1(t) & \mathbf{x}_2(t) & \dots & \mathbf{x}_N(t) \end{pmatrix}^{\mathrm{T}} \\
\mathbf{u}(t) &:= \begin{pmatrix} u_1(t) & u_2(t) & \dots & u_N(t) \end{pmatrix}^{\mathrm{T}} \\
\mathbf{T}_{\mathrm{D}} &:= \begin{pmatrix} T_{\mathrm{D},1} & T_{\mathrm{D},2} & \dots & T_{\mathrm{D},N} \end{pmatrix}^{\mathrm{T}}.
\end{aligned} \tag{2}
$$

Matrix $\mathbf{C} \in \mathbb{N}^{NM \times NM}$ rearranges $\mathbf{x}(t)$ so that it contains joint angles, joint velocities, and joint accelerations in that order.

3 | MODEL PREDICTIVE CONTROL

This section formulates the planning task as a continuous-time optimal control problem including motion objectives and constraints with a particular focus on obstacle avoidance and obstacle representations. The continuous-time optimal control problem is transformed into an approximative nonlinear program through full discretization. The hypergraph formulation exploits the inherently sparse problem structure and facilitates efficient computations of derivatives at runtime. The section concludes with a description of the closed-loop control algorithm and compensation for input delays.

3.1 | Continuous-time optimal control problem

The main objective is the regulation of robot joint motion towards the static target joint state $\mathbf{x}_{\mathrm{ref}} = \begin{pmatrix} \mathbf{q}_{\mathrm{ref}} & 0 & 0 \end{pmatrix}^{\mathrm{T}}$. Due to the moving horizon in model predictive control, the extension to track a target joint state trajectory is straight-forward. For the sake of readability, $\tau \in \left[\tau_0, \tau_f\right]$ denotes the open-loop time index in prediction and optimization whereas $t \in [0, \infty)$ indicates the closed-loop execution time with t_n for $n \in \mathbb{N}$ representing time indices at which planning is triggered. To ensure a smooth and safe robot operation, the planning stage already considers safety and dynamic motion constraints, in particular separation from obstacles, and limits of joint angles and joint velocities.

The general planning task is formulated as a continuous-time optimal control problem:

$$
\min_{\mathbf{u}(\tau)} \left[V_f\left(\mathbf{x}(\tau_f)\right) + \int_{\tau_0}^{\tau_f} \ell\left(\mathbf{x}(\tau), \mathbf{u}(\tau)\right) + \ell_{\mathrm{O}}\left(\mathbf{x}(\tau)\right) + \ell_{\mathrm{L}}\left(\mathbf{x}(\tau)\right) \, \mathrm{d}\tau \right] \tag{3a}
$$

subject to

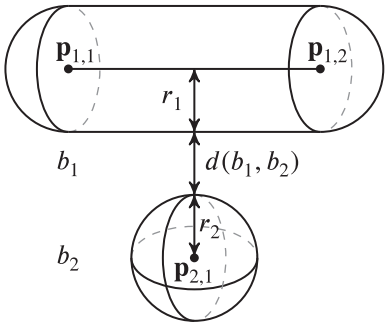
$$
\mathbf{x}(\tau_0) = \begin{pmatrix} \mathbf{q}(t_n) & \dot{\mathbf{q}}(t_n) & \ddot{\mathbf{q}}(t_n) \end{pmatrix}^{\mathrm{T}}, \tag{3b}
$$

$$
\dot{\mathbf{x}}(\tau) = \mathbf{f}\left(\mathbf{x}(\tau), \mathbf{u}(\tau), \mathbf{T}_{\mathrm{D}}\right), \tag{3c}
$$

$$
\mathbf{c}\left(\mathbf{x}(\tau)\right) \leq \mathbf{0}, \tag{3d}
$$

$$
\mathbf{h}\left(\mathbf{x}(\tau), \mathbf{u}(\tau)\right) \leq \mathbf{0}. \tag{3e}
$$

FIGURE 2 Distance between a cylinder $b_1 \in \mathcal{B}$ and a regular sphere $b_2 \in \mathcal{B}$



The problem formulation includes a terminal state cost $V_f : \mathbb{R}^{NM} \to \mathbb{R}$:

$$V_f(\mathbf{x}(\tau_f)) := (\mathbf{x}(\tau_f) - \mathbf{x}_{\text{ref}})^{\text{T}} \, \mathbf{Q}_f \, (\mathbf{x}(\tau_f) - \mathbf{x}_{\text{ref}}). \tag{4}$$

The term $\ell : \mathbb{R}^{NM} \times \mathbb{R}^{N} \to \mathbb{R}$ denotes the common quadratic running cost:

$$\ell(\mathbf{x}(\tau), \mathbf{u}(\tau)) := (\mathbf{x}(\tau) - \mathbf{x}_{\text{ref}})^{\text{T}} \, \mathbf{Q} \, (\mathbf{x}(\tau) - \mathbf{x}_{\text{ref}}) + \mathbf{u}^{\text{T}}(\tau) \, \mathbf{R} \, \mathbf{u}(\tau) + \dot{\mathbf{u}}^{\text{T}}(\tau) \, \mathbf{R}_{\text{d}} \, \dot{\mathbf{u}}(\tau), \tag{5}$$

which penalizes the squared state error, control effort, and control smoothness. The weight matrices $\mathbf{Q} \in \mathbb{R}^{NM \times NM}$, $\mathbf{R} \in \mathbb{R}^{N \times N}$, and $\mathbf{R}_{\text{d}} \in \mathbb{R}^{N \times N}$ are positive definite and determine the compromise between regulation error, control effort, and smoothness. The additional running cost terms $\ell_{\text{O}} : \mathbb{R}^{NM} \to \mathbb{R}$ and $\ell_{\text{L}} : \mathbb{R}^{NM} \to \mathbb{R}$ as well as the inequality constraints (3d) for obstacle avoidance are discussed in the next section. Equation (3b) ensures that the current joint state at time t_n represents the initial state of the predicted trajectory at τ_0. Equation (3c) denotes the dynamic model (2). The term $\mathbf{h} : \mathbb{R}^{NM} \times \mathbb{R}^{N} \to \mathbb{R}^{2N(M+1)}$ in inequality constraint (3e) captures the boundaries on state and control signals:

$$\mathbf{h}(\mathbf{x}(\tau), \mathbf{u}(\tau)) := \begin{pmatrix} \mathbf{x}(\tau) - \mathbf{x}_{\max} \\ \mathbf{x}_{\min} - \mathbf{x}(\tau) \\ \mathbf{u}(\tau) - \mathbf{u}_{\max} \\ \mathbf{u}_{\min} - \mathbf{u}(\tau) \end{pmatrix}. \tag{6}$$

3.2 | Obstacle avoidance

Objects are represented by line swept spheres introduced by Larsen et al.[42] Line swept spheres base on geometric primitives which facilitate distance calculations in three-dimensional space. Geometric primitives are for example points, line segments, or faces. Sweeping a radius r around a point or line segment forms the sphere or line swept sphere depicted in Figure 2. Line swept spheres can enclose elongated objects efficiently as opposed to spheres.

Let $\mathcal{B}$ denote the set of bounding volumes $b \in \mathcal{B}$, which are represented as tuples:

$$b := \left\langle \mathbf{p}_1(t) \in \mathbb{R}^3, \mathbf{p}_2(t) \in \mathbb{R}^3, r \in \mathbb{R}^+ \right\rangle. \tag{7}$$

A line swept sphere is composed of a line segment defined by the two dynamic endpoints $\mathbf{p}_1(t)$ and $\mathbf{p}_2(t)$ and the sweeping radius r. An ordinary sphere constitutes a special case of a line swept sphere with $\mathbf{p}_1(t) = \mathbf{p}_2(t)$. Multiple cylindrical or spherical volumes enclose more complex objects following Martinez-Salvador et al.[41] The separation $d(b_1, b_2) : \mathcal{B}^2 \to \mathbb{R}$ between two objects $b_1 \in \mathcal{B}$ and $b_2 \in \mathcal{B}$ as demonstrated in Figure 2 is given by:

$$d(b_1, b_2) = \tilde{d}(b_1, b_2) - r_1 - r_2, \tag{8}$$

in which $\tilde{d}(b_1, b_2) : \mathcal{B}^2 \to \mathbb{R}$ denotes the Euclidean distance between line segments of b_1 and b_2 computed with the algorithm by Lumelsky.[30]

The set of bounding volumes $\mathcal{B}$ is partitioned into robot links $\mathcal{L} \subseteq \mathcal{B}$ and relevant obstacles $\mathcal{O} \subseteq \mathcal{B}$. An obstacle $o \in \mathcal{B}$ becomes relevant for motion planning once it enters the robots safety volume $s \in \mathcal{B}$, which is defined as a sphere that

encloses the robot workspace plus an additional safety margin:

$$\mathcal{O} := \{o \in \mathcal{B} | d(o, s) < 0\}. \tag{9}$$

Consider the set $\tilde{\mathcal{L}}$ that contains link body pairs $(l_1, l_2) \in \mathcal{L} \times \mathcal{L}$ that potentially cause a self-collision:

$$\tilde{\mathcal{L}} := \{(l_1, l_2) \in \mathcal{L} \times \mathcal{L} | \exists\ \mathbf{x}(t),\ d(l_1, l_2) < 0\}. \tag{10}$$

Obviously, consecutive link bodies are self-collision safe. Obstacle poses are defined in world coordinates of the task space and for robot bodies $l \in \mathcal{L}$, they are obtained from joint states $\mathbf{x}$ by forward kinematics. Their dependency on $\mathbf{x}$ is omitted for better readability.

The function $\mathbf{c} : \mathbb{R}^{NM} \to \mathbb{R}^{|\mathcal{L}\times\mathcal{O}|+|\tilde{\mathcal{L}}|}$ in (3d) includes obstacle avoidance $\mathbf{c}_{\mathrm{O}} : \mathbb{R}^{NM} \to \mathbb{R}^{|\mathcal{L}\times\mathcal{O}|}$ and self-collision avoidance $\mathbf{c}_{\mathrm{L}} : \mathbb{R}^{NM} \to \mathbb{R}^{|\tilde{\mathcal{L}}|}$:

$$\mathbf{c}(\mathbf{x}(\tau)) := \begin{pmatrix} \mathbf{c}_{\mathrm{O}}(\mathbf{x}(\tau)) \\ \mathbf{c}_{\mathrm{L}}(\mathbf{x}(\tau)) \end{pmatrix}. \tag{11}$$

The obstacle avoidance constraint is defined by:

$$\mathbf{c}_{\mathrm{O}}(\mathbf{x}(\tau)) := \begin{pmatrix} \alpha_{j,\mathrm{O}} - d(b_1, b_2) \\ \vdots \end{pmatrix},\ \forall (b_1, b_2) \in \mathcal{L} \times \mathcal{O}, \tag{12}$$

in which, $\alpha_{j,\mathrm{O}} \in \mathbb{R}^+$ for $j = 1, 2, \dots, |\mathcal{L} \times \mathcal{O}|$ denotes the separation distance threshold. The self-collision constraint is defined analogously:

$$\mathbf{c}_{\mathrm{L}}(\mathbf{x}(\tau)) := \begin{pmatrix} \alpha_{j,\mathrm{L}} - d(l_1, l_2) \\ \vdots \end{pmatrix},\ \forall (l_1, l_2) \in \tilde{\mathcal{L}}, \tag{13}$$

with threshold $\alpha_{j,\mathrm{L}} \in \mathbb{R}^+$ for $j = 1, 2, \dots, |\tilde{\mathcal{L}}|$.

Safe operation not only requires mere obstacle avoidance but should also evaluate the general clearance between objects. The separation cost $\rho : \mathcal{B} \times \mathcal{B} \times \mathbb{R} \times \mathbb{R} \to [0, \eta]$ grows quadratically for a separation distance below an activation threshold $\beta \in \mathbb{R}^+$, and is zero otherwise:

$$\rho(b_1, b_2, \eta, \beta) := \begin{cases} \eta\left(\frac{d(b_1,b_2)}{\beta} - 1\right)^2 & d(b_1, b_2) < \beta \\ 0 & d(b_1, b_2) \geq \beta \end{cases} \tag{14}$$

Compared to the original one by Mohri et al,[47] this function can easily be scaled by $\eta \in \mathbb{R}^+$ to adjust the impact of each obstacle on the overall costs without changing the activation threshold. The cost term is bounded as the obstacle avoidance HC (11) already prohibits critical separation distances. The potentials for obstacle avoidance and self-collisions are finally given by:

$$\begin{aligned} \ell_{\mathrm{O}}(\mathbf{x}(\tau)) &:= \sum_{(b_1,b_2)\ \in\mathcal{L}\times\mathcal{O}} \rho(b_1, b_2, \eta_{j,\mathrm{O}}, \beta_{j,\mathrm{O}}),\ j = 1, 2, \dots, |\mathcal{L} \times \mathcal{O}| \\ \ell_{\mathrm{L}}(\mathbf{x}(\tau)) &:= \sum_{(l_1,l_2)\ \in\tilde{\mathcal{L}}} \rho(l_1, l_2, \eta_{j,\mathrm{L}}, \beta_{j,\mathrm{L}}),\ j = 1, 2, \dots, |\tilde{\mathcal{L}}|. \end{aligned} \tag{15}$$

Please note that $\beta_{j,\mathrm{O}} > \alpha_{j,\mathrm{O}}$ and $\beta_{j,\mathrm{L}} > \alpha_{j,\mathrm{L}}$ for (15) to be effective.

3.3 | Direct optimal control—full discretization

Continuous-time optimal control problems like (3) are solved either by indirect or direct methods.[48] Indirect methods apply the calculus of variations to obtain the optimality conditions regarding a boundary value problem which is then

solved numerically. In contrast, direct methods transform the continuous-time problem into a nonlinear program using discretization respectively transcription methods. Model predictive control problems prefer direct methods due to a more robust convergence behavior and availability of efficient and generic implementations of solvers. In the following, the direct method is applied to transform (3) into an approximating nonlinear program. Based on the benchmark results in our previous work,[44] we apply a full discretization method in favor of multiple shooting or Hermite-Simpson collocation since it reveals a more sparse structure that is fully exploited by the hypergraph in Section 3.4.

Full discretization is a particular case of direct collocation or multiple shooting in which both controls and states are uniformly discretized at identical grid points:

$$\begin{aligned} \tau_0 < \tau_1 < \ldots < \tau_k < \ldots < \tau_K = \tau_f \\ \tau_{k+1} - \tau_k = \Delta\tau. \end{aligned} \tag{16}$$

Furthermore, let $\mathbf{x}_{0:K}$ and $\mathbf{u}_{0:K-1}$ form discrete trajectories:

$$\begin{aligned} \mathbf{x}_{0:K} &:= \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k, \ldots, \mathbf{x}_K \\ \mathbf{u}_{0:K-1} &:= \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_k, \ldots, \mathbf{u}_{K-1}, \end{aligned} \tag{17}$$

in which $\mathbf{x}_k := \mathbf{x}(\tau_k)$ and $\mathbf{u}_k := \mathbf{u}(\tau_k)$. The approach assumes fine grid partitions $\Delta\tau$ such that the upper sum rule approximates the integrand of running costs:

$$\int_{\tau_0}^{\tau_f} \ell(\mathbf{x}(\tau), \mathbf{u}(\tau)) + \ell_{\mathrm{O}}(\mathbf{x}(\tau)) + \ell_{\mathrm{L}}(\mathbf{x}(\tau))\, \mathrm{d}\tau \approx \Delta\tau \sum_{k=0}^{K-1} \tilde{\ell}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{u}_{k-1}) + \ell_{\mathrm{O}}(\mathbf{x}_k) + \ell_{\mathrm{L}}(\mathbf{x}_k), \tag{18}$$

in which $\tilde{\ell} : \mathbb{R}^{NM} \times \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ approximates (5):

$$\ell(\mathbf{x}(\tau_k), \mathbf{u}(\tau_k)) \approx \tilde{\ell}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{u}_{k-1}) := (\mathbf{x}_k - \mathbf{x}_{\mathrm{ref}})^{\mathrm{T}}\, \mathbf{Q}\, (\mathbf{x}_k - \mathbf{x}_{\mathrm{ref}}) + \mathbf{u}_k^{\mathrm{T}}\, \mathbf{R}\, \mathbf{u}_k + (\mathbf{u}_k - \mathbf{u}_{k-1})^{\mathrm{T}}\, \mathbf{R}_{\mathrm{d}}\, (\mathbf{u}_k - \mathbf{u}_{k-1})(\Delta\tau)^{-2}. \tag{19}$$

If $k = 0$ then $\mathbf{u}_{-1}$ copies $\mathbf{u}_0$ of the previous optimization or $\mathbf{0}$ in case of the first run. Rösmann et al[44] approximate the system dynamics by forward differences in the full discretization case. In here, integration operates with midpoint differences since they achieve a better approximation without a significant increase in computation time:

$$\dot{\mathbf{x}}(\tau_k) \approx \frac{\mathbf{x}_{k+1} - \mathbf{x}_k}{\Delta\tau} = \mathbf{f}\left(\frac{\mathbf{x}_{k+1} + \mathbf{x}_k}{2}, \mathbf{u}_k, \mathbf{0}\right). \tag{20}$$

The state equality constraint in (3c) without input delay is then mimicked by:

$$\underbrace{(\mathbf{x}_{k+1} - \mathbf{x}_k)\Delta\tau^{-1} - \mathbf{f}\left(\frac{\mathbf{x}_{k+1} + \mathbf{x}_k}{2}, \mathbf{u}_k, \mathbf{0}\right)}_{\boldsymbol{\phi}(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k)} = \mathbf{0}. \tag{21}$$

Since dead time is compensated separately in Section 3.5, the nonlinear program is defined without time delays. The formulation is not restricted to this particular choice of $\boldsymbol{\phi} : \mathbb{R}^{NM} \times \mathbb{R}^{NM} \times \mathbb{R}^N \to \mathbb{R}^{NM}$, other collocation kernels or numerical integrators apply as well. The resulting nonlinear program is given by:

$$\min_{\substack{\mathbf{u}_{0:K-1} \\ \mathbf{x}_{0:K}}} \left[V_f(\mathbf{x}_K) + \Delta\tau \sum_{k=0}^{K-1} \tilde{\ell}(\mathbf{x}_k, \mathbf{u}_k, \mathbf{u}_{k-1}) + \ell_{\mathrm{O}}(\mathbf{x}_k) + \ell_{\mathrm{L}}(\mathbf{x}_k) \right] \tag{22a}$$

subject to

$$\mathbf{x}_0 = \left(\mathbf{q}(t_n)\ \dot{\mathbf{q}}(t_n)\ \ddot{\mathbf{q}}(t_n)\right)^{\mathrm{T}}, \tag{22b}$$

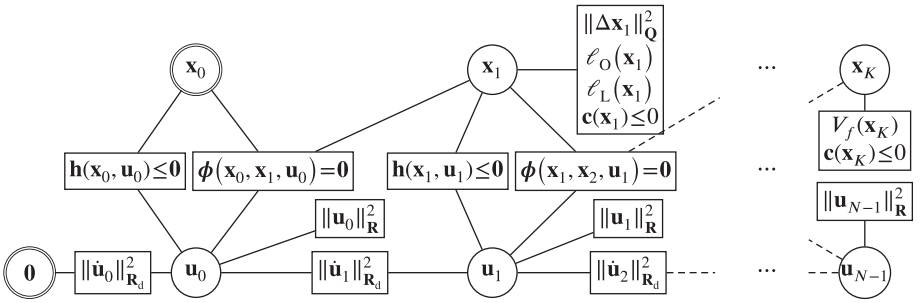


FIGURE 3 Hypergraph for nonlinear program (22). Vertices are depicted as circles and hyperedges as rectangles. Vertices $\mathbf{x}_0$ and $\mathbf{0}$ with a double circle are fixed during optimization

$$\boldsymbol{\phi}(\mathbf{x}_k, \mathbf{x}_{k+1}, \mathbf{u}_k) = \mathbf{0}, \tag{22c}$$

$$\mathbf{c}(\mathbf{x}_k) \leq \mathbf{0}, \tag{22d}$$

$$\mathbf{h}(\mathbf{x}_k, \mathbf{u}_k) \leq \mathbf{0}. \tag{22e}$$

3.4 | Hypergraph formulation

The nonlinear program (22) is solved by standard gradient-based constrained optimization algorithms, for example, interior-point solvers, sequential quadratic programming, or projected gradient methods.[49] Most model predictive control approaches refine the second-order derivatives iteratively by the Broyden-Fletcher-Goldfarb-Shanno algorithm, and first-order derivatives are computed either by automatic differentiation or from finite differences. Central differences constitute a balanced trade-off between accuracy and computational complexity as each constraint and cost term in (22) is evaluated only twice per parameter. The sparse finite difference computation with hypergraphs restricts the calculation of the constraint Jacobian to its structural nonzeros and only evaluates the relevant cost terms to obtain the gradient.

A hypergraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is composed of a set of vertices $\mathcal{V}$ and a set of edges $\mathcal{E}$. A vertex $\mathbf{v} \in \mathcal{V}$ refers to an optimization parameter which is in case of (22) either a state $\mathbf{x}_k$ or control input vector $\mathbf{u}_k$:

$$\mathcal{V} := \{\mathbf{x}_0, \mathbf{u}_0, \mathbf{x}_1, \mathbf{u}_1, \dots, \mathbf{x}_{N-1}, \mathbf{u}_{N-1}, \mathbf{x}_N\}. \tag{23}$$

The set comprises $N+1$ state vertices and N control vertices that correspond to grid points along the prediction horizon. Each vertex caches inherent properties like box constraints as in $\mathbf{h}(\cdot)$ of (22). Furthermore, a vertex is considered as fixed, if its parameters are not subject to optimization, but appear in cost or constraint terms. Tagging vertices as fixed, avoids the validation of trivial equality constraints such as (22b). In contrast to a regular undirected graph, a so-called hyperedge connects an arbitrary number of vertices. It refers to a scalar cost term as well as equality and inequality constraints at time instances k. Figure 3 shows the hypergraph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ of the nonlinear program (22) with the following abbreviations:

$$\begin{aligned}
\Delta\mathbf{x}_k &:= \mathbf{x}_k - \mathbf{x}_{\mathrm{ref}} \\
\|\Delta\mathbf{x}_k\|^2_{\mathbf{Q}} &:= \Delta\mathbf{x}_k^{\mathrm{T}}\ \mathbf{Q}\ \Delta\mathbf{x}_k \\
\|\mathbf{u}_k\|^2_{\mathbf{R}} &:= \mathbf{u}_k^{\mathrm{T}}\ \mathbf{R}\ \mathbf{u}_k \\
\|\dot{\mathbf{u}}_k\|^2_{\mathbf{R}_{\mathrm{d}}} &:= \dot{\mathbf{u}}_k^{\mathrm{T}}\ \mathbf{R}_{\mathrm{d}}\ \dot{\mathbf{u}}_k.
\end{aligned} \tag{24}$$

For the sake of clarity, functions, and constraints that refer to the same set of parameters are combined into a single edge. But also from an implementation point of view, this might be of interest whenever multiple functions share similar precomputations, for instance, distance calculations. Vertex $\mathbf{x}_0$ is fixed during optimization, and hence its unary edges are omitted.

Any solver that requires to compute the cost gradient and constraint Jacobian matrices needs to iterate the set of hyperedges in the graph. For every edge, only the dependent parameters are adjusted to apply finite differences resulting in small dense block Jacobian matrices. In case of the constraints, it is most efficient to directly memory-map the block

Jacobian to the overall sparse constraint Jacobian. Embedding the optimization problem into the hypergraph significantly reduces computation time. The interested reader is referred to Rösmann et al[44] for more details.

3.5 | Closed-loop predictive control

Algorithm 1 details the steps of model predictive control that solves the optimal control problem formulated as nonlinear program (22) at instances t_n with cycle time $T_\mathrm{S} = t_{n+1} - t_n$. The controller observes the current joint state $\left(\mathbf{q}(t_n)\ \dot{\mathbf{q}}(t_n)\ \ddot{\mathbf{q}}(t_n)\right)^\mathrm{T}$, the set of vertices $\mathcal{V}$ and the current obstacle set $\mathcal{O}$ as inputs. The set $\mathcal{V}$ might be empty at the very first invocation, but in subsequent stages, it warm-starts the solver to increase efficiency. Line 2 initializes the optimization parameters by a straight line interpolation of states. The initial controls are set to zero. Subsequent sampling intervals merely shift the previous solution parameters by one time step to the subsequent vertex. The first vertex is initialized with the most recent state vector, and the Nth vertex is extrapolated linearly from vertex $N-1$. Line 3 generates and updates the set of hyperedges, connecting states, and controls with their respective cost functions and constraints. As obstacles enter and leave the safety zone according to (9), the number of edges for obstacle costs and constraints changes dynamically, forcing the hypergraph to update. A numerical solver computes the optimal control and state sequence for (22). Obstacles are considered static during optimization. The algorithm returns the vertex set $\mathcal{V}^\star$ containing the optimal parameter values for warm-starting in the next sampling interval as well as the optimized control sequence $\mathbf{u}^\star_{0:K-1}$. The latter is turned into a piecewise constant, continuous-time trajectory:

$$\mathbf{u}^\star(\tau) := \mathbf{u}^\star_k = \text{const. for } \tau \in [\tau_k, \tau_{k+1}). \tag{25}$$

Algorithm 1. Model predictive controller

1: **procedure** FEEDBACKCONTROL($\mathcal{V}, \mathcal{O}, \mathbf{q}(t_n), \dot{\mathbf{q}}(t_n), \ddot{\mathbf{q}}(t_n)$)
2: $\mathcal{V} \leftarrow$ UPDATEVERTICES($\mathcal{V}, \mathbf{q}(t_n), \dot{\mathbf{q}}(t_n), \ddot{\mathbf{q}}(t_n)$) ▷ Warmstart
3: $\mathcal{E} \leftarrow$ CONSTRUCTHYPEREDGES($\mathcal{V}, \mathcal{O}$) ▷ Structure of (22)
4: $\mathcal{V}^\star \leftarrow$ SOLVE($\mathcal{V}, \mathcal{E}$) ▷ Solve (22)
5: **return** $\{\mathbf{u}^\star_{0:K-1}, \mathcal{V}^\star\}$
6: **end procedure**

Finally, A portion of $\mathbf{u}^\star(\tau)$ is commanded to the robot's velocity tracking controller:

$$\boldsymbol{\mu}(t) := \mathbf{u}^\star(\tau)\big|_{\tau_0 = t_n} \text{ for } t \in [t_n, t_{n+1}). \tag{26}$$

Figure 4 outlines the timing of events during a control iteration for one joint and shows the effects of computation time T_C as well as dead time $T_{\mathrm{D},1}$. The controller observes a measurement $q_1(t_n)$ and starts solving (22). The optimization ends at t_ctrl after $T_\mathrm{C} = t_\mathrm{ctrl} - t_n$ and the controller commands the control intervention $\mu_1(t_\mathrm{ctrl})$. After a dead time of $T_{\mathrm{D},1} = t_\mathrm{exec} - t_\mathrm{ctrl}$ the control intervention becomes effective as $\tilde{\mu}_1(t_\mathrm{exec})$. Usually, the computation time T_C for a single optimization of the nonlinear program can be neglected in comparison to the control cycle duration T_S. However, if the optimization problem becomes more and more complex or the required solver accuracy increases, computation time rises, and causes the controller to react to $q_1(t_n)$ rather than $q_1(t_\mathrm{exec})$ leading to a decreased control performance. Dead time $T_{\mathrm{D},i}$ has a similar effect as the net computation time. Thus both causes are considered as a single coherent delay ΔT. Grüne and Pannek[14] compensate input delays for model predictive control by extrapolating the conventional initial state $\mathbf{x}_0$ for ΔT into the future by forward integration of (2):

$$\begin{aligned}\tilde{\mathbf{x}}_0 &= \mathbf{x}_0 + \begin{pmatrix} \int_0^{\Delta T_1} \mathbf{f}_1(\mathbf{x}_1(\tau), u_1(\tau - T_{\mathrm{D},1}))\mathrm{d}\tau \\ \int_0^{\Delta T_2} \mathbf{f}_2(\mathbf{x}_2(\tau), u_2(\tau - T_{\mathrm{D},2}))\mathrm{d}\tau \\ \vdots \\ \int_0^{\Delta T_N} \mathbf{f}_N(\mathbf{x}_N(\tau), u_N(\tau - T_{\mathrm{D},N}))\mathrm{d}\tau \end{pmatrix} \\ \Delta T_i &= T_{\mathrm{D},i} + T_\mathrm{C},\ i = 1, 2, \dots, N. \end{aligned} \tag{27}$$

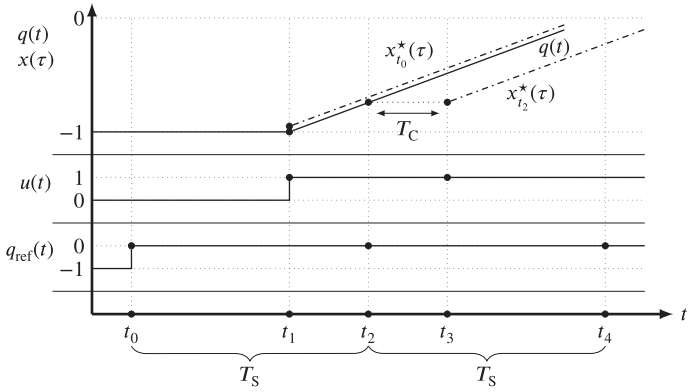


FIGURE 4 The controller observes a new measurement $q_1(t_n)$ and starts solving the optimization problem. The optimized control intervention $\mu_1(t_{\text{ctrl}})$ is sent afterward, leading to a computation time of T_{C}. After dead time $T_{\text{D},1}$ the control intervention excites the system, indicated by $\tilde{\mu}_1(t_{\text{exec}})$

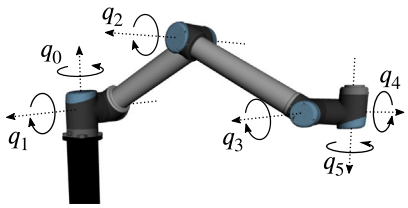


FIGURE 5 Sketch of the Universal Robot UR10 manipulator that is used for practical experiments [Colour figure can be viewed at wileyonlinelibrary.com]

The optimization problem is then initialized with the corrected state $\tilde{\mathbf{x}}_0$. The state evolution and extrapolation rest upon cached control commands. Computation times vary with the complexity and convergence of optimization. Thus a median filter estimates delay T_{C} over past observed delays.

4 | EXPERIMENTAL RESULTS

This section introduces model identification and analyzes alternative model structures. Based on the analysis, the model that constitutes the best compromise between complexity and accuracy is selected and validated on the actual robot. The model predictive controller for online trajectory optimization is evaluated under task variations, in particular, a dynamic target pose and the penetration of the workspace by dynamic obstacles.

4.1 | Robot manipulator

The model is identified and validated on a real Universal Robot UR10 collaborative robot with $N = 6$ joints, which is sketched in Figure 5. The communication between robot and model predictive controller is established through the Robot Operating System (ROS Kinetic). Joint velocities are commanded via ROS messages to the Universal Robot ROS driver, while it publishes current robot states to the controller. The built-in robot firmware does not provide for self-collision avoidance. Therefore the original ROS driver by Andersen[50] is extended by monitoring self-collisions and stopping the robot. The model predictive controller runs on a computer equipped with an Intel i7-6900K CPU at 3.2 GHz using 32 GB RAM at 2.133 GHz under Ubuntu 16.04.

4.2 | Model identification

Figure 6 illustrates the velocity tracking controllers step response for the joint velocity $\dot{q}_1$ of the first joint. At step time $t = 1$ second, the response exhibits a time delay in the order of tens of milliseconds caused by both rise time and dead time. The response also displays an overshoot and a settling time in the order of several hundred milliseconds, which approves the second-order system with dead time introduced in Section 2.

The signals for identification and validation are amplitude-modulated pseudo-random binary joint velocity signals as proposed by Barker et al,[51] with a duration of 30 seconds with random step lengths of multiples of 1 second and random

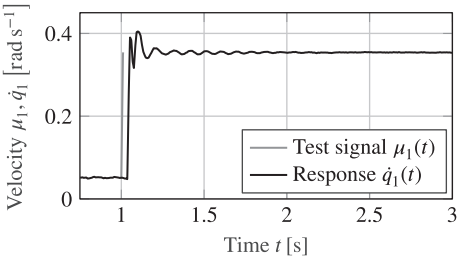


FIGURE 6 Response $\dot{q}_1(t)$ of the first joint to a step change from 0.05 rad second^{-1} to 0.35 rad second^{-1} in desired angular velocity $\mu_1(t)$

TABLE 1 Results of parameter identification for one velocity tracking controller

$m-1$	$p_{i,1}$	$p_{i,2}$	$T_{\mathrm{D},i}$	K_i	F
0	—	—	0 s	0.9849	80.70%
1	−28.4528	—	0 s	1.0035	89.79%
2	−30.4696 −34.4844i	−30.4696 + 34.4844i	0 s	0.9995	91.49%
0	—	—	0.030 s	0.9974	91.45%
1	−102.0675	—	0.023 s	0.9990	91.81%
2	−83.6140 −81.4326i	−83.6140 +81.4326i	0.019 s	0.9985	91.92%

Note: The results are poles $p_{i,j}$, dead times $T_{\mathrm{D},i}$, and goodness of fit F.

joint speed amplitudes from $[-0.3, 0.3]$. The models are identified independently of each other with only one joint active at a time. The identification and validation involve models of zero to second-order with and without explicit dead time. The parameters are estimated by nonlinear least squares of the residuals between the observed and the simulated joint velocities. The accuracy of the identified model is validated on a new amplitude-modulated pseudo-random binary signal sampled with the same bandwidth and amplitude spectrum, using the goodness of fit F:

$$F = 100\,(1 - \mathrm{NRMSE})\ \%, \tag{28}$$

in which NRMSE denotes the normalized root mean square error between measurements $q_i(t)$ and simulations $x_{1,i}(\tau)$.

Table 1 reports the identified poles $p_{i,j}$ with $i = 1, 2, \ldots, N$ and $j = 1, 2$ as well as the goodness of fit of the pure approximated control loop without integrator. For the sake of briefness, the table only reports data for one joint model, the qualities of fit are similar for the remaining joints, and subsequent evaluations also apply. Choosing a prediction model means a compromise between model complexity and accuracy. Lower order prediction models reduce the computational burden in the model predictive control framework and either help to increase the cycle rate or to extend the planning horizon. The state vector dimensions are 6, 12, and 18 for $M = 1$, $M = 2$, and $M = 3$, respectively. Without dead time, the delay is approximated by slower system dynamics as for models with dead time. This raises the model complexity to $M = 3$ to have similar accuracy (91.49 %) as dead time models. Dead times for $M = 1$ are higher than for $M > 1$ because in this case, dead time also approximates slow system dynamics. With modeled dead time, all systems have similar accuracy in the range of 91.73 %, which makes $M = 1$ the natural choice for a prediction model. The remaining model mismatch due to unmodeled dynamics is usually unproblematic given the closed-loop nature of model predictive control.

4.3 | Controller parametrization and stability

In Mayne et al,[52] stability is enforced by imposing proper terminal constraints on the state, whereas Grüne and Pannek[14] prove stability under particular assumptions without introducing terminal constraints or costs. In a well-defined task, start and target configurations exhibit sufficient clearance to obstacles such that neither constraints nor costs for collision avoidance are active at start or target. Under this assumption, the problem reduces to an integrator system with quadratic costs and box constraints for states and controls. As shown by Grüne and Pannek[14] such a system is asymptotically stable for $K \geq 2$, and thus for a free workspace, no additional stabilizing constraints and costs are required. However, as obstacles restrict the workspace in the transition from start to target pose, collision avoidance constraints become active, and the

assumption by Grüne and Pannek[14] no longer applies. Grüne and Pannek[14] also mentions that for additional costs or constraints, the minimum horizon length might increase. In general, the obstacle motion is not known beforehand. As a consequence, the nominal horizon length K and the temporal length, encoded in $\Delta\tau$, are adjusted empirically. A large temporal horizon is required in case of obstacles for a preemptive trajectory refinement and to plan the trajectory beyond the obstacle, where objective costs are less again. The choice of $\Delta\tau$ includes a compromise between the horizon time and the accuracy of system dynamics approximation in the prediction step according to (21). The step size becomes $\Delta\tau = 0.1$ s, since an integrator system has comparatively simple dynamics. A horizon length of $K = 25$ corresponds to a look-ahead time of 2.5 s and constitutes a viable compromise between computational effort and temporal horizon length. In-depth stability properties in the context of model predictive control, in general, are summarized in Grüne and Pannek,[14] Mayne et al.[52]

In summary, the model predictive controller operates with the following parametrization:

- IPOPT[53] with MA27,[54] using up to 50 iterations and a tolerance of 0.001
- Full discretized horizon of length $K = 25$ with $\Delta\tau = 0.1$ s
- State bounds ± 3.1rad for all joints
- Ratio between $\mathbf{Q}$ and $\mathbf{R}$ respectively $\mathbf{R}_\mathrm{d}$ is $\frac{10}{1}$ and $\mathbf{Q}_f = \mathbf{Q}$
- Closed-loop cycle time $T_\mathrm{S} = 0.1$ s
- Window size of three for the moving median filter for extrapolation.

Please note that additional parameters are specified in the experiments. The IPOPT solver belongs to the class of interior point approaches whose large scale capabilities are well suited for fully discretized collocation problems with many optimization variables.

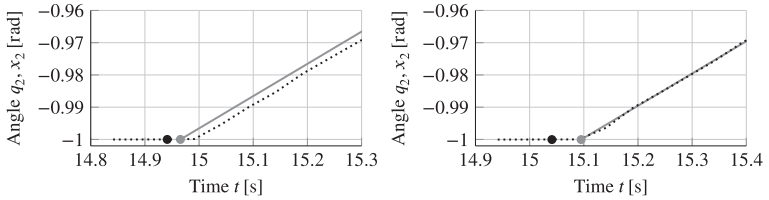
4.4 | Model validation

The validation scenario commands four consecutive joint space configurations that constitute a closed movement in which the first and final waypoint coincide:

$$\begin{aligned}\mathbf{q}_{\mathrm{ref},0} &= \mathbf{0}\\ \mathbf{q}_{\mathrm{ref},1}^{\mathrm{T}} &= \begin{pmatrix}-1.0 & -1.0 & 1.0 & 0.0 & 0.0 & 0.0\end{pmatrix}\\ \mathbf{q}_{\mathrm{ref},2}^{\mathrm{T}} &= \begin{pmatrix}-1.0 & -1.0 & 1.0 & -1.0 & 1.0 & 1.0\end{pmatrix}\\ \mathbf{q}_{\mathrm{ref},3} &= \mathbf{q}_{\mathrm{ref},0}.\end{aligned} \tag{29}$$

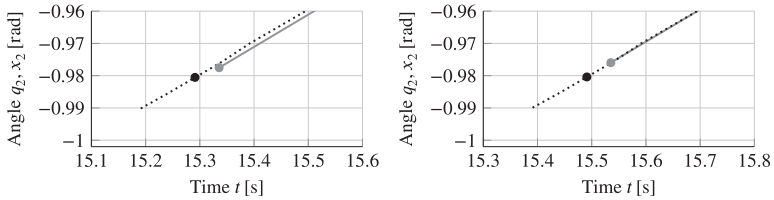
The control bounds are ± 0.1rad second^{-1} for the first three joints and ± 0.3rad second^{-1} for the remaining. The motion between the first two waypoints is restricted to joints 1, 2, and 3. The second part affects the joints 4, 5, and 6, and in the final segment, all six joints move simultaneously. Prediction accuracy is defined as the residual between predicted $\mathbf{x}$ and actual states $\mathbf{q}$ according to (28). The analysis assumes that currently planned and future executed controls coincide over the current planning horizon. This assumption is valid unless the motion is either disturbed or the target configuration emerges on the horizon. To better illustrate the different effects of dead time and computation time, the predicted state sequence in one case starts from a steady state, while in the other case, the robot is in motion.

Figure 7 compares the predicted joint trajectory (gray solid line) and actual joint trajectory (black dotted line) of the second joint without dead time (Figure 7A) and with dead time (Figure 7B). The robot motion starts from rest, and the model predictive controller is commanded to a new target joint state. The black dot indicates the time instance t_n of observation of the current joint state. The predicted trajectory starts at t_ctrl indicated by the gray dot. The discrepancy between executed motion at t_exec and the prediction that ignores dead time $T_{\mathrm{D},2}$ becomes apparent in Figure 7A with a quality of fit of 96.83 %. Figure 7B illustrates the same motion with explicit consideration of dead time in prediction. The extrapolated initial state (gray dot) matches the system state at t_exec improving the overall prediction quality of fit to 99.52 %. Figure 8 presents the same analysis, but in this case, during motion at constant velocity rather than starting from standing still. The additional effects of computation time delay become apparent in Figure 8A, causing a shift between prediction and execution with a goodness of fit of 97.52 %. The computation delay T_C causes the control command to be



(A) Without dead time compensation (B) With dead time compensation

FIGURE 7 Without compensation of input dead time, the prediction (solid gray) expects the system (dotted black) to immediately react on the commanded control, whereas the actual response is delayed (A). Extrapolating the state from cached controls improves the prediction (B). The black dot indicates the measured state and the gray dot shows the first state of the prediction at t_{ctrl}



(A) Only compensating dead time in a non-steady state (B) Additionally compensating computation time delay

FIGURE 8 Without compensating for the computation time delay in a nonsteady state, the prediction (solid gray) is based on a past and no longer valid initial state, indicated by the slight shift in $-x_2$ direction compared to the actual (dotted black) behavior (A). Also compensating computation time by extrapolating the initial state into the future further improves the prediction. The black dot indicates the measured state and the gray dot shows the first state of the prediction at t_{ctrl}

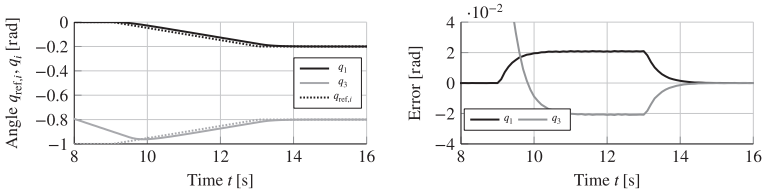
sent at t_{ctrl} when the initial state for prediction no longer matches the actual joint state after optimization. Compensating the computation delay by initializing the optimization with a further extrapolated initial state, better matches prediction and execution (Figure 8B) with an improvement in the quality of fit to 99.55 %.

The mean computation times for solving the nonlinear optimization problem for the simple integrator are 6.69 milliseconds and for the integrator with extrapolation of initial state are 6.73 milliseconds. Considering that the 95 % confidence interval of the estimated mean is [−0.24 millisecond 0.15 millisecond] there is no statistically significant difference between both computation times.

4.5 | Dynamic goal pose

This section investigates the closed-loop behavior for a dynamic reference configuration. This type of task variations are typical for pick and place operations with moving objects, for example, parts transported by a conveyor belt or object handover in human-robot collaborations. In many cases, the motion targets are defined in the workspace and are not necessarily present in the form of joint configurations. This experiment assumes that a moving target pose was previously transferred to the joint space, also taking into account possible redundancies. With modifications to the cost function, the presented approach is also applicable for target poses in the workspace. This, however, leads to an economic model predictive controller that introduces other benefits and drawbacks which are not in the scope of this article.

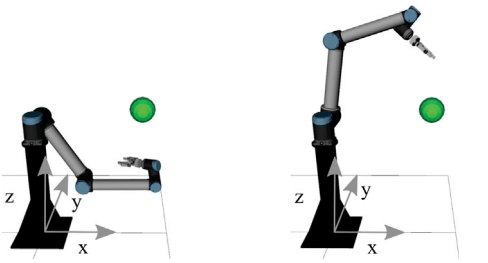
The robot starts from an initial configuration $\mathbf{q} = \mathbf{0}$ and is initially commanded to the static reference configuration $\mathbf{q}_{\text{ref}}^{\text{T}}(t \in [0\ t_1]) = \begin{pmatrix} 0.0 & 1.0 & -1.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$. At time $t_1 = 9$seconds the reference sets in motion, moving linearly for four seconds at a reference joint rate of $\dot{\mathbf{q}}_{\text{ref}}^{\text{T}} = \begin{pmatrix} -0.05 & -0.05 & 0.05 & 0.0 & 0.0 & 0.0 \end{pmatrix}$ rad second^{-1}. At time $t_f = 13$seconds the reference pose motion stops and $\mathbf{q}_{\text{ref}}^{\text{T}}(t_f) = \begin{pmatrix} -0.2 & 0.8 & -0.8 & 0.0 & 0.0 & 0.0 \end{pmatrix}$ remains static. Figure 9A illustrates the joint trajectories of the first and third joint and their corresponding references. At time $t_1 = 9$seconds, the first joint state q_1 already converged to the reference state $q_{\text{ref},1}$, whereas the third joint did not complete its motion yet. Figure 9B illustrates the evolution of the joint state error. During the reference motion, the residual error settles to a nonzero steady state of 0.02



(A) Joint states and joint references.

(B) Residual joint state error.

FIGURE 9 Joint states and joint references (A) and the corresponding residual joint state error (B) for a reference joint motion



(A) Initial joint configuration.

(B) Target joint configuration.

FIGURE 10 Experiment I: Robot and obstacle configurations [Colour figure can be viewed at wileyonlinelibrary.com]

rad attributed to the moving target and the control cycle time T_S. Once the reference joint motion stops, joint state errors converge to zero.

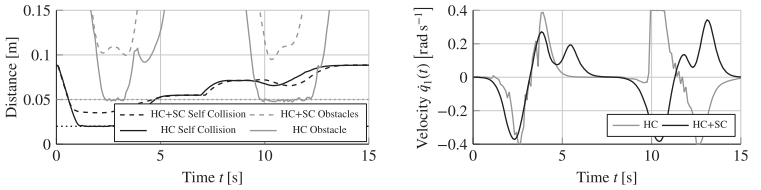
4.6 | Obstacle avoidance

For the following analysis, the parameters for obstacle avoidance are set to:

- Control bounds ±0.4rad second^{-1} for all joints
- Weight $\eta_L = 10$ and margin $\beta_L = 0.05$m for self-collision
- Weights $\eta_O = 4$ and margins $\beta_O = 0.2$m for static and dynamic obstacle avoidance
- Self-collision constraint $\alpha_L = 0.02$m
- Obstacle avoidance constraints $\alpha_O = 0.05$m for static and dynamic obstacles
- Radius of 2 m for the safety volume s.

The first experiment contains a single spherical obstacle that remains static during task execution, but whose location is allowed to change across tasks. The sphere of radius $r_1 = 0.1$m is located at $\mathbf{p}_1^T = \begin{pmatrix}0.8 & 0.16 & 1.0\end{pmatrix}$ relative to the shown robot coordinate system in Figure 10. To improve experimental reproducibility and to eliminate the influence of object detection and tracking on the results, the obstacle is purely virtual rather than physical. Figure 10A,B visualize the initial and final robot configuration for this experiment, respectively. The robot repeatedly moves between $\mathbf{q}_{\mathrm{ref},0}^T = \begin{pmatrix}0.0 & 1.0 & -1.0 & 3.0 & 1.0 & 0.0\end{pmatrix}$ and $\mathbf{q}_{\mathrm{ref},1}^T = \begin{pmatrix}0.0 & -1.4 & 1.1 & 1.0 & 2.0 & 0.0\end{pmatrix}$ with the obstacle blocking the straight line path in joint space. The straight line joint motion towards the reference pose is nonfeasible due to a self-collision at intermediate states. In addition to the presented plots, the attached video shows the robot and the virtual obstacle for better visualization.[55]

In the following analysis HC refer to (11) and soft constraints (SC) refer to (15). Obstacle avoidance with mere HC results in a trajectory with minimal separation from obstacles. However, a limited number of solver iterations and inaccurate delay compensations for oscillating computation times increase discrepancies between predicted and actual joint states. These discrepancies then lead to constraint violations for self-collision ($t = 2$seconds) and obstacle avoidance ($t = 10$seconds) shown in Figure 11A. Even though the residuals of these violations are small, they cannot always be resolved within the forthcoming control cycle and lead to nonfeasible solutions of (22). In consequence, the controls during the motion along the obstacle become nonsmooth, as shown in Figure 11B. They become smooth again once the robot's joint motion is no longer constraint by self-collision and obstacles. Although this variant does not cause collisions, the nonsmooth control commands are not desirable in robot applications.

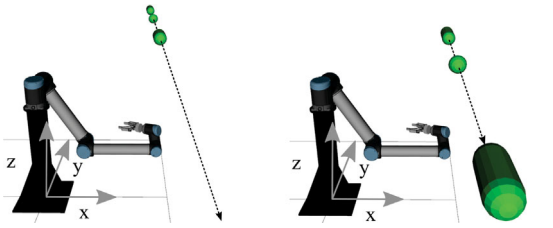


(A) Minium distances regarding self-collision and obstacle avoidance.

(B) Actual closed-loop control sequence of the first joint.

FIGURE 11 Experiment I: Comparing minimum distances (A) for self-collision and obstacle avoidance as well as closed-loop control sequences (B) of the first joint in HC case (HC) as well as hard- and SC case (HC+SC). Nonfeasible solutions of (22) lead to nonsmooth controls in the HC case. Additional SC produce feasible and smooth trajectories

FIGURE 12 Experiment II: Robot and obstacle configurations [Colour figure can be viewed at wileyonlinelibrary.com]



(A) Initial joint configuration.

(B) Moving obstacle traversing workspace.

SC introduce repulsive behavior before the activation of HC and lead to more robust planning alongside obstacles with smooth controls. The obstacle penalty increases with proximity to the obstacle and causes a compromise between trajectory length and clearance from obstacles, as shown in Figure 11A. Figure 11B shows smooth controls of the first joint that allow practical application of online trajectory optimization under real-time constraints. The compromise trajectory solution under SC mainly depends on the relative weights of trajectory and terms in the cost function. Thus, it is not possible to guarantee compliance of constraints in general, and therefore, HC still provide a fall-back safety level.

The second experiment considers three moving obstacles traversing the robot's workspace one after each other. Figure 12 illustrates the obstacle path by the dotted line. The first and third obstacles are cylinders of lengths 0.5 m and 0.3 m, the second obstacle is a sphere. All obstacles share the same radius of 0.1 m and point towards $-y$ direction. The obstacles start at $\mathbf{p}_1^{\mathrm{T}} = \begin{pmatrix}1.1 & 7.0 & 0.8\end{pmatrix}$, $\mathbf{p}_2^{\mathrm{T}} = \begin{pmatrix}1.1 & 5.0 & 0.8\end{pmatrix}$, and $\mathbf{p}_3^{\mathrm{T}} = \begin{pmatrix}1.1 & 3.0 & 0.8\end{pmatrix}$ and move 10.0 m in $-y$ direction for 50 seconds with constant velocity perpendicular to the robots planar $\mathbf{0}$ configuration. They disturb the free space robot motion, which is equal to the last experiment. Please again also refer to the corresponding video for a qualitative impression.[55]

Even though trajectory optimization does not extrapolate the obstacle motion into the future, the minimum distances of 0.034 m and 0.051 m for self-collision and obstacles, respectively, are respected. SC introduce environmental perception, which offers the possibility to already adjust planning for nearby but not yet critical obstacles. This is particularly important in this dynamic scene. Figure 13 collects the control sequences of all joints and demonstrates their smoothness. Some joint motions come to a temporary halt and resume their movement once the obstacle passed. If this occurs for all joints at the same time, planning is trapped in a local minimum where stopping is supposed to be optimal instead of circumventing the obstacle. The next experiment will deal with this case for a more complex obstacle in more detail. Another type of local minima becomes apparent during the second obstacle for $t \geq 26$seconds. The robot plans to pass the obstacle on the left. However, since the obstacle is moving in the same direction, the first joint sheers off instead of avoiding the obstacle on the right. In this case, the local minimum leads to the target configuration but is not globally optimal. Estimating and forward predicting obstacle motions as well as an alternative initialization that considers different topologies could initiate a different behavior.

The third experiment replaces the former virtual dynamic obstacles with a real human obstacle. The person is modeled by a composition of seven bounding volumes using a sphere (head) and six cylinders (arms, thorax, and pelvis), as illustrated in Figure 14A. A marker-based motion capture system tracks the person's movement and reports it to the planning algorithm. Figure 14B, C visualizes the initial and final robot configuration for this experiment. The robot moves

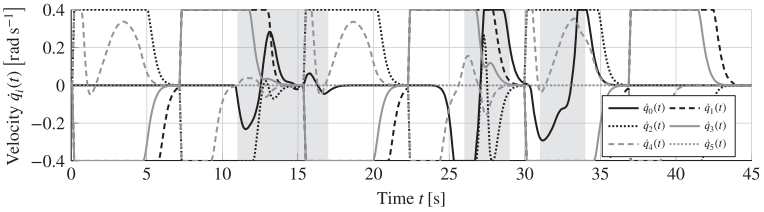
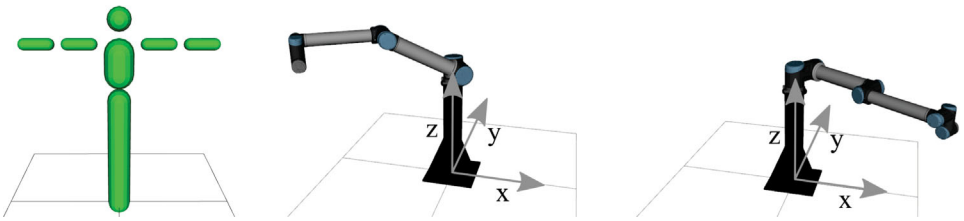


FIGURE 13 Experiment II: Closed-loop control sequences of all joints in hard- and SC case (HC+SC). The marked gray areas indicate intervals of obstacle avoidance



(A) Human model (B) Initial joint configuration. (C) Final joint configuration.

FIGURE 14 Robot and obstacle configurations for experiment III [Colour figure can be viewed at wileyonlinelibrary.com]

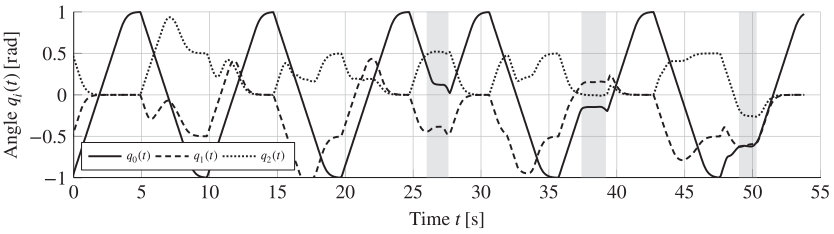


FIGURE 15 Experiment III: Closed-loop states of the first three joints in hard- and SC case (HC+SC). The marked gray areas indicate local minima during planning

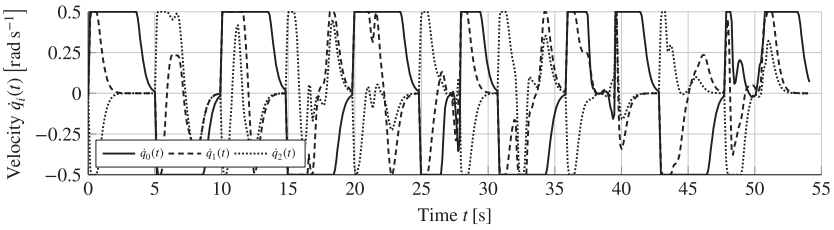


FIGURE 16 Experiment III: Closed-loop control sequences of the first three joints in hard- and SC case (HC+SC)

between $\mathbf{q}_{\text{ref},0}^{\text{T}} = \begin{pmatrix}-1.0 & -0.5 & 0.5 & 0.0 & 0.0 & 0.0\end{pmatrix}$ and $\mathbf{q}_{\text{ref},1}^{\text{T}} = \begin{pmatrix}1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0\end{pmatrix}$ and vice versa while the human periodically interferes. The control bounds are increased to ±0.5rad second^{-1} for all joints. Please again also refer to the corresponding video.[55]

Figure 15 shows the trajectory of the first three joints. The remaining three joints of the wrist play a secondary role and are not shown. The three gray areas mark situations in which the robot stops due to local minima. In these cases, the person blocks the motion of the first joint while the other joints reach their target value. An evasive motion, for example, by folding the entire arm, would cause the converged joints to deviate from their target value and thus increase optimization costs. Also, the limited planning horizon cannot capture reaching the target values behind the obstacle, which would reduce costs. In consequence, the costs are higher for a motion around the obstacle than to stand still. As in the previous experiment, the controls are smooth (Figure 16) and the minimum distances of 0.158 m and 0.067 m for self-collision, and the person are respected, respectively.

4.7 | Computation times

This section summarizes the measured computation times of the previous experiments and gives an impression of the scalability of the approach. Table 2 reports the mean values, standard deviations as well as minimum and maximum values.

TABLE 2 Computation times in different scenarios

	Scenario I			Scenario II	Scenario III
	Reference	HC	HC+SC	HC+SC	HC+SC
Mean	7.39 ms	29.77 ms	29.09 ms	34.27 ms	41.06 ms
SD	2.53 ms	16.07 ms	12.42 ms	11.81 ms	21.40 ms
Min	4.43 ms	5.91 ms	7.31 ms	7.56 ms	8.22 ms
Max	27.21 ms	60.52 ms	63.40 ms	70.18 ms	82.60 ms

Note: The reference is calculated without any obstacles and without any obstacle or collision avoidance strategies.

FIGURE 17 Computation times during the third experiment

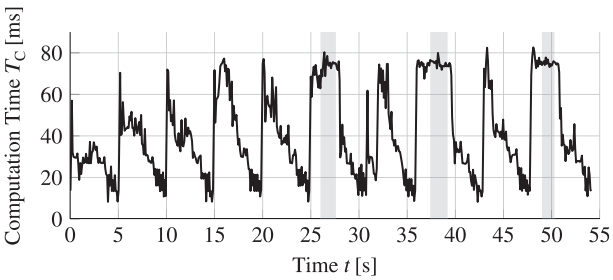
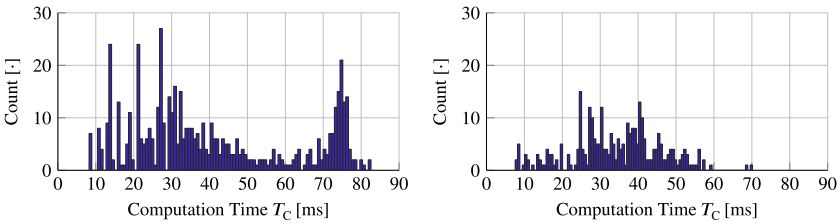


FIGURE 18 Comparison of the computation time histograms of the third (A) and second (B) experiment [Colour figure can be viewed at wileyonlinelibrary.com]



In the second scenario, the measuring range is limited to the interval $[7.3, 36.9]$ seconds, since this is when the avoidance motion occurs. The table also contains reference values that report the computation times of free space trajectory optimization without obstacles and collision avoidance.

The standard deviation (16.07 milliseconds) of the HC variant is greater than that of HCSC (12.42 milliseconds) in the same scenario. This is a consequence of the aforementioned problem in Section 4.6 that violated HC cannot always be resolved. The attempts of the solver to resolve these, results in escalating calculation times. The HCSC variant achieves approximately the same mean value (29.09 milliseconds) as the HC variant (29.77 milliseconds) although it is more complex due to additional SC. SC increase complexity, but also eliminate the aforementioned escalations so that both effects compensate each other. In the second experiment, three moving obstacles are used, and up to two are simultaneously active. On the one hand, the optimization problem is more complicated due to at least two active obstacles and thus requires more computation time on average (34.27 milliseconds) and at the maximum value (70.18 milliseconds) than the HCSC variant in the first experiment. On the other hand, the lack of estimation and prediction of the obstacle's motion prevents warm starts from efficiently exploiting previous solutions. The third experiment emphasizes this effect at the mean (41.06 milliseconds) and maximum values (82.60 milliseconds) by the simultaneous presence of seven obstacles. However, local minima also make a significant contribution to the higher mean value and standard deviation (21.40 milliseconds). Figure 17 shows individual computation times during the third experiment and marked intervals in which local minima occurred. The intervals illustrate the consistently high calculation time at local minima. Since optimization costs are at a constant value above zero, the solver keeps high effort to reduce these costs, which succeeds only if the local minimum is resolved. Figure 18 shows, for comparison, the histograms of the calculation times for this (Figure 18A) and the second (Figure 18B) experiment. The third experiment shows a second cluster starting at 70 milliseconds, which is mainly caused by the local minima (cf. Figure 17). In the second experiment, no such cluster is visible in the upper region.

When obstacles enter or leave the workspace, new edges for collision avoidance are added, or old edges are removed in the hypergraph. For the cost function, only the calculations within the edge change. These relatively simple adjustments

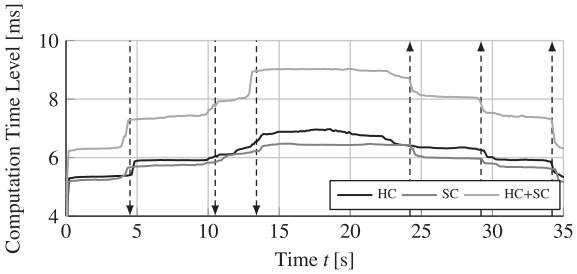


FIGURE 19 Median-filtered computation time levels of three variants for a modified version of the second experiment

are the advantage of hypergraph-based optimization, which, unlike methods that use code generation, eg, does not have to fundamentally redesign the optimization problem in case of structural changes. Figure 19 shows how the computation times of the HC, SC, and HCSC variants scale in a variation of the second scenario. In this scenario, the robot stays in a fixed configuration, and three obstacles pass by without collision. All values are median filtered to improve the visibility of changes in the level of computation times. Please note that these measurements only reflect the pure presence of obstacles and do not consider evading. An arrow pointing downwards indicates the timestamp at which an obstacle enters the robot's workspace. Analogously, an arrow pointing upwards marks a leaving obstacle. At those timestamps, an increase/decrease in the computation time level can be identified. In particular, the HCSC variant stands out from the other variants, which are evenly positioned. The analysis confirms the scalability of trajectory optimization with the hypergraph structure and adaptation of the number of constraints at optimization run-time without significant computational overhead. A detailed comparison of the hypergraph with other methods regarding preparation time can be found in Rösmann et al.[44]

5 | CONCLUSION AND OUTLOOK

The article proposes and analyzes a novel approach for online trajectory optimization of collaborative robots under task variations, self-collision avoidance, and dynamic obstacle constraints. The results of model identification and validation show that explicitly considering input delays like computation time as well as dead time allows for using six parallel integrators with the same accuracy of the prediction during closed-loop control as second or third-order models. The control scheme successfully converges to static target configurations and demonstrates a benign residual tracking error in case of moving targets. The analysis of the collision avoidance behavior during imitated robotic pick-and-place experiments shows that mere HC are not sufficient as they lead to nonsmooth control sequences. Enhancing obstacle avoidance via SC that grow quadratically in the near vicinity of the obstacle enables smooth controls. HC provide a fall-back safety level in case of critical separation distances. Extending the analysis to multiple dynamic obstacles confirms these results. It shows the benefits from the efficient preparation as the hypergraph formulation adapts the number of constraints to the complexity of the scene without significant computational overhead. There is a measurable increase in computation time per obstacle; however, the experiment with a person modeled by seven obstacles has demonstrated the applicability in an authentic scenario. Nevertheless, the performance of recently very successful SQP solver, which can benefit even more from warm starts, will be evaluated in future work. The experiments show two types of local minima that arise from the nature of gradient-based optimization. One which causes the robot to stop in front of large obstacles and another one that leads to long detours when avoiding a dynamic obstacle on the wrong side. In the worst case, the approach, therefore, behaves like known reactive methods that pause motions in front of an obstacle. However, the experiments also demonstrated how the approach successfully avoids the obstacle and reaches the final configuration. Future work will, therefore, transfer the previous topology-based parallel planners for planar mobile robots by Rösmann et al[56] to robotic manipulators to plan topologically distinctive trajectories in 3D space that cover the global minimum. Also, the approach is extended to estimate and extrapolate obstacle movements and consider their future states in the trajectory optimization to further improve the avoidance behavior. While assuming constant velocities is a quick and straightforward improvement, usually movement uncertainties must be considered within the horizon that might lead to the freezing robot problem.

ORCID

Maximilian Krämer https://orcid.org/0000-0003-0179-0684

REFERENCES

1. Kalakrishnan M, Chitta S, Theodorou E, Pastor P, Schaal S. Stomp: Stochastic trajectory optimization for motion planning. Paper presented at: Proceedings of the International Conference on Robotics and Automation; 2011:4569-4574; IEEE.
2. Zucker M, Ratliff N, Dragan AD, et al. Chomp: covariant Hamiltonian optimization for motion planning. *Int J Robot Res*. 2013;32(9-10):1164-1193.
3. Bosscher P, Hedman D. Real–time collision avoidance algorithm for robotic manipulators. *Ind Robot Int J*. 2011;38(2):186-197.
4. Lo Guarino Bianco C, Gerelli O. Online trajectory scaling for manipulators subject to high-order kinematic and dynamic constraints. *IEEE Trans Robot*. 2011;27(6):1144-1152.
5. Xiao Y, Du Z, Dong W. Smooth and near time–optimal trajectory planning of industrial robots for online applications. *Ind Robot Int J*. 2012;39(2):169-177.
6. Uchiyama N, Mori K, Terashima K, Saeki T, Kamigaki T, Kawamura H. Optimal motion trajectory generation and real-time trajectory modification for an industrial robot working in a rectangular space. *J Syst Design Dyn*. 2013;7(3):278-292.
7. Ragaglia M, Zanchettin AM, Rocco P. Safety-aware trajectory scaling for human-robot collaboration with prediction of human occupancy. Paper presented at: Proceedings of the International Conference on Advanced Robotics (ICAR); 2015:85-90; IEEE.
8. Lange F, Albu-Schaffer A. Path-accurate online trajectory generation for jerk-limited industrial robots. *IEEE Robot Automat Lett*. 2016;1(1):82-89.
9. Beckert D, Pereira A, Althoff M. Online verification of multiple safety criteria for a robot trajectory. Paper presented at: Proceedings of the 56th Annual Conference on Decision and Control (CDC); 2017:6454-6461; IEEE.
10. Wang S, Bao J, Fu Y. Real-time motion planning for robot manipulators in unknown environments using infrared sensors. *Robotica*. 2007;25(02):201-211.
11. Kohrt C, Stamp R, Pipe AG, Kiely J, Schiedermeier G. An online robot trajectory planning and programming support system for industrial use. *Robot Comput Integrat Manufact*. 2013;29(1):71-79.
12. Siciliano B. *Robotics: Modelling, Planning and Control*. London, UK: Springer; 2010.
13. Yoshida E, Yokoi K, Gergondet P. Online replanning for reactive robot motion: Practical aspects. Paper presented at: Proceedings of the International Conference on Intelligent Robots and Systems; 2010: 5927-5933; IEEE.
14. Grüne L, Pannek J. Nonlinear model predictive control: theory and algorithms. *Communications and Control Engineering*. 2nd ed. New York, NY: Springer; 2017.
15. Ide S, Takubo T, Ohara K, Mae Y, Arai T. Real-time trajectory planning for mobile manipulator using model predictive control with constraints. Paper presented at: Proceedings of the 2011 8th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI); 2011:244-249; IEEE.
16. Zube A. Cartesian nonlinear model predictive control of redundant manipulators considering obstacles. Paper presented at: Proceedings of the 2015 IEEE International Conference on Industrial Technology (ICIT 2015); 2015:137-142; IEEE.
17. Buizza Avanzini G, Zanchettin AM, Rocco P. Constrained model predictive control for mobile robotic manipulators. *Robotica*. 2018;36(01):19-38.
18. Poignet P, Gautier M. Nonlinear model predictive control of a robot manipulator. Paper presented at: Proceedings of the International Workshop on Advanced Motion; 2000:401-406; IEEE.
19. Vivas A, Mosquera VB. Predictive functional control of a puma robot. Paper presented at: Proceedings of the Association for Computer Studies Educators CICC; 2005.
20. Hedjar R, Boucher P. Nonlinear receding-horizon control of rigid link robot manipulators. *Int J Adv Robot Syst*. 2005;2(1):3.
21. Rybus T, Seweryn K, Sasiadek JZ. Control system for free-floating space manipulator based on nonlinear model predictive control (nmpc). *J Intell Robot Syst*. 2017;85(3-4):491-509.
22. Sangiovanni B, Rendiniello A, Incremona GP, Ferrara A, Piastra M. Deep reinforcement learning for collision avoidance of robotic manipulators. Paper presented at: Proceedings of the European Control Conference; 2018:2063-2068.
23. Wang M, Luo J, Walter U. A non-linear model predictive controller with obstacle avoidance for a space robot. *Adv Space Res*. 2016;57(8):1737-1746.
24. Diehl M, Bock HG, Diedam H, Wieber PB. Fast direct multiple shooting algorithms for optimal robot control. In: Diehl M, ed. *Fast motions in biomechanics and robotics; vol. 340 of Lecture Notes in Control and Information Sciences*. Berlin, Germany: Springer; 2006:65-93.
25. Zhao J, Diehl M, Longman R, Bock HG, Schloeder J. Nonlinear model predictive control of robots using real-time optimization. Paper presented at: Proceedings of the AIAA/AAS Astrodynamics Specialist Conference and Exhibit; 2004; AIAA.
26. Wang Y, Ye X, Yang Y, Zhang W. Collision-free trajectory planning in human-robot interaction through hand movement prediction from vision. Paper presented at: Proceedings of the 2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids); 2017:305-310; IEEE.
27. Incremona GP, Ferrara A, Magni L. MPC for robot manipulators with integral sliding modes generation. *IEEE/ASME Trans Mechatron*. 2017;22(3):1299-1307.
28. Wilson J, Charest M, Dubay R. Non-linear model predictive control schemes with application on a 2 link vertical robot manipulator. *Robot Comput Integrat Manufact*. 2016;41:23-30.
29. Zietkiewicz J, Owczarkowski A. Direct nonlinear model predictive control and predictive control with feedback linearization. a comparison of the approaches. Paper presented at: Proceedings of the International Carpathian Control Conference (ICCC); 2017:451-455; IEEE.
30. Lumelsky VJ. On fast computation of distance between line segments. *Inf Process Lett*. 1985;21(2):55-61.

31. Gilbert EG, Johnson DW, Keerthi SS. A fast procedure for computing the distance between complex objects in three-dimensional space. *IEEE J Robot Automat*. 1988;4(2):193-203.
32. Kuffner JJ, LaValle SM. Rrt-connect: an efficient approach to single-query path planning. *Robotics and Automation*. Piscataway, NJ: IEEE; 2000:995-1001.
33. Kavraki LE, Svestka P, Latombe JC, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans Robot Automat*. 1996;12(4):566-580.
34. Hsu D, Latombe JC, Motwani R. Path planning in expansive configuration spaces. *Int J Comput Geometry Appl*. 1999;09(04n05):495-512.
35. Cascio J, Karpenko M, Gong Q, Sekhavat P, Ross IM. Smooth proximity computation for collision-free optimal control of multiple robotic manipulators. Paper presented at: Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems; 2009:2452-2457.
36. dos Santos RR, Steffen V, Saramago S. Robot path planning in a constrained workspace by using optimal control techniques. *Multibody Syst Dyn*. 2008;19(1-2):159-177.
37. Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. In: Cox IJ, Wilfong GT, eds. *Autonomous Robot Vehicles*. Vol SMC-13. New York, NY: Springer; 1986:396-404.
38. Ataka A, Qi P, Shiva A, Shafti A, Wurdemann H, Liu H, Althoefer K. Real-time pose estimation and obstacle avoidance for multi-segment continuum manipulator in dynamic environments. Paper presented at: Proceedings of the 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2016:2827-2832; IEEE.
39. Landry C, Hömberg D, Henrion R, Gerdts M. Path planning and collision avoidance for robots. *Numer Algebra Control Optimiz*. 2012;2(3):437-463.
40. Jardine PT, Givigi S, Yousefi S. Planar inequality constraints for stable, collision-free model predictive control of a quadcopter. *IFAC-PapersOnLine*. 2017;50(1):9095-9100.
41. Martinez-Salvador B, del Pobil AP, Perez-Francisco M. Very fast collision detection for practical motion planning. i. the spatial representation. Paper presented at: Proceedings of the International Conference on Robotics and Automation; 1998:624-629; IEEE.
42. Larsen E, Gottschalk S, Lin MC, Manocha D. Fast proximity queries with swept sphere volumes. Tech. Rep.; 1999.
43. Andersson JAE, Gillis J, Horn G, Rawlings JB, Diehl M. CasADi – a software framework for nonlinear optimization and optimal control. *Math Prog Comput*. 2019;11(1):1-36.
44. Rösmann C, Krämer M, Makarow A, Hoffmann F, Bertram T. Exploiting sparse structures in nonlinear model predictive control with hypergraphs. Paper presented at: Proceedings of the International Conference on Advanced Intelligent Mechatronics (AIM); 2018:1255-1260; IEEE.
45. Findeisen R. Nonlinear model predictive control: a sampled data feedback perspective (Ph.D. thesis). Universität Stuttgart; 2005.
46. Universal Robots. Universal robot UR10; 2018. https://www.universal-robots.com/products/ur10-robot/.
47. Mohri A, Yang XD, Yamamoto M. Collision free trajectory planning for manipulator using potential function. Paper presented at: Proceedings of the 1995 IEEE International Conference on Robotics and Automation. IEEE; 1995: 3069–3074.
48. Betts JT. Survey of numerical methods for trajectory optimization. *J Guidance Control Dyn*. 1998;21(2):193-207.
49. Nocedal J, Wright SJ. *Numerical Optimization*. 2nd ed. Springer Science+Business Media LLC: New York, NY; 2006.
50. Andersen TT. Optimizing the universal robots ros driver. Tech. Rep.; Technical University of Denmark, Department of Electrical Engineering; 2015.
51. Barker HA, Tucker AJ, Godfrey KR. Comparison of perturbation signals for linear system identification in the frequency domain. *IEE Proc Control Theory Appl*. 1999;146(6):535-548.
52. Mayne DQ, Rawlings JB, Rao CV, Scokaert P. Constrained model predictive control: Stability and optimality. *Automatica*. 2000;36(6):789-814.
53. Wächter A, Biegler LT. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math Prog*. 2006;106(1):25-57.
54. Harwell Subroutine Library. A collection of Fortran codes for large scale scientific computation; 2018. http://www.hsl.rl.ac.uk/.
55. Experimental results on: Model predictive control of a collaborative manipulator considering dynamic obstacles; 2019. https://youtu.be/6e234FqG7K8.
56. Rösmann C, Hoffmann F, Bertram T. Integrated online trajectory planning and optimization in distinctive topologies. *Robot Autonom Syst*. 2017;88:142-153.

SUPPORTING INFORMATION

Additional supporting information may be found online in the Supporting Information section at the end of this article.

How to cite this article: Krämer M, Rösmann C, Hoffmann F, Bertram T. Model predictive control of a collaborative manipulator considering dynamic obstacles. *Optim Control Appl Meth*. 2020;41:1211–1232. https://doi.org/10.1002/oca.2599