Online Motion Planning based on Nonlinear Model Predictive Control with Non-Euclidean Rotation Groups

Christoph Rösmann, Artemi Makarow and Torsten Bertram

***Abstract*— This paper proposes a novel online motion planning approach to robot navigation based on nonlinear model predictive control. Common approaches rely on pure Euclidean optimization parameters. In robot navigation, however, state spaces often include rotational components which span over non-Euclidean rotation groups. The proposed approach applies nonlinear increment and difference operators in the entire optimization scheme to explicitly consider these groups. Realizations include but are not limited to quadratic form and time-optimal objectives. A complex parking scenario for the kinematic bicycle model demonstrates the effectiveness and practical relevance of the approach. In case of simpler robots (e.g. differential drive), a comparative analysis in a hierarchical planning setting reveals comparable computation times and performance. The approach is available in a modular and highly configurable open-source C++ software framework.**

I. INTRODUCTION

In the context of robotics and autonomous driving, online trajectory planning, usually as part of a hierarchical planning architecture, is still an essential part of research to meet the requirements of navigation in increasingly complex and highly dynamic environments. In contrast to classical path planning approaches [1], [2], trajectory planning takes the temporal profile into account and enables improved performance as well as explicit compliance with kinodynamic and dynamic constraints. An established method is the extension of the well-known elastic band (EB) path planning approach [3] to trajectory deformation in [4]. Lau et al. present a method that represents trajectories by Bzier splines and adheres to kinodynamic constraints of non-holonomic robots [5]. Not only for mobile robots, but especially for integrator dynamics, online planning algorithms based on covariant or stochastic gradient descent and obstacle potentials are provided in [6], [7]. Delsart et al. extend the EB approach to the deformation of trajectories rather than paths [4]. Lau et al. [5] optimize trajectories represented by splines according to kinodynamic constraints of the robot.

Many optimization based approaches can be derived from a generic optimal control formulation [8], [9] and interpreted with state feedback as a variant of predictive control. Predictive controllers repeatedly solve an optimal control problem (OCP) during runtime while commanding the first action of the optimal control trajectory to the system in each closed-loop step. Fig. 1 shows two examples for such derivations. The left side shows what is commonly known as model predictive control (MPC) of continuous-time systems. Direct methods discretize the OCP to obtain a nonlinear program (NLP) for which many efficient solving techniques exist [10]. State feedback then completes the approach to a model predictive controller. Note that the controller type depends on the OCP formulation, e.g., receding horizon MPC and variable horizon MPC. Since the computational burden is large, many approaches restrict the solution space or approximate the original problem. For example the dynamic window approach (DWA) [11] can be interpreted as a special receding horizon MPC scheme [12]. The search space is restricted to a (collision-free) constant control input for the complete horizon. The required computation times are low but due to the less degrees of freedom in control, the approach is suboptimal and cannot predict motion reversals such that control of car-like robots is rather limited. Another approach is the Timed-Elastic-Band (TEB) approach, which was originally derived as a scalarized multi-objective least squares optimization [13]. It can also be derived from a time-optimal control problem as shown in the right of Fig. 1. Hereby, the robot kinematics are expressed geometrically similar to a flat system model in the pose space and a nonlinear program is obtained by applying direct transcription, in particular finite-difference collocation. The nonlinear program is then transformed to an unconstrained least-squares problem by applying quadratic penalty functions [14]. In [15], the TEB is extended to car-like robots, however, the flat system model limits the definition of arbitrary constraints, e.g. limits on the steering rate are not included yet. Recently, an extension and reformulation in terms of Lie groups is proposed in [16].

Since computational resources have increased and algorithms have become more efficient, the interest in full MPC-

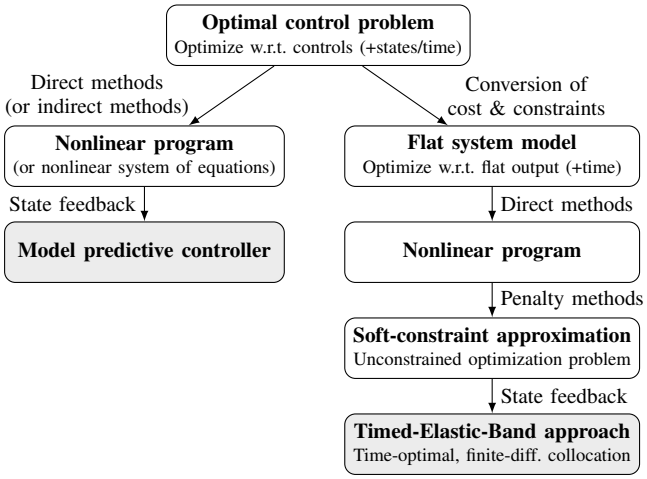

Fig. 1: From optimal control to online motion planning

The authors are with the Institute of Control Theory and Systems Engineering, TU Dortmund University, 44227 Dortmund, Germany {forename.surname}@tu-dortmund.de

based approaches has been growing. Path following control with recursive feasibility and stability properties is provided in [17]. Zhang et al. generate collision-free trajectories and considers general obstacles and that can be represented as the union of convex sets [18]. A recent approach based on convex inner approximations provides feasible and collision-free solutions in a few solver iterations [19].

Motion planning for robot navigation usually includes non-Euclidean rotational components, i.e. the robot's heading. The contribution of this paper is as follows: We propose an MPC-based motion planning scheme that differs from others by the explicit consideration of orientation groups during optimization. To our best knowledge, available MPC-based approaches are tailored for Euclidean state spaces. Our MPC formulation is versatile and includes many common realizations including receding horizon quadratic form and shrinking horizon time-optimal objectives. We further provide a generic, highly customizable and easily extendable C++ software framework with Robot Operating System (ROS) integration (*mpc_local_planner*).

The paper is organized as follows: Section II provides the formal problem description of the approach and Section III the numerical realization. Section IV first evaluates the proposed approach with a kinematic bicycle model and then conducts a comparative analysis for a common navigation scenario. Finally, section V concludes the work.

II. MOTION PLANNING PROBLEM DESCRIPTION

This paper addresses robotic systems described by nonlinear and time-invariant differential equations with time $t \in \mathbb{R}$, state trajectory $\mathbf{x} : \mathbb{R} \mapsto \mathcal{X}$ and control trajectory $\mathbf{u} : \mathbb{R} \mapsto \mathcal{U}$:

$$\dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big). \tag{1}$$

The mapping $\mathbf{f} : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}^p$ with $p = \dim(\mathcal{X})$ is continuous and Lipschitz in its first argument. The state and control spaces, $\mathcal{X}$ and $\mathcal{U}$ respectively, do not have to be exclusively Euclidean spaces, as discussed later. System (1) is also subject to state and input constraint sets originating from internal robot constraints, but also from the dynamic environment in which the robot operates. Therefore, state constraints $\mathbf{x}(t) \in \mathbb{X}(t) \subseteq \mathcal{X}$ and input constraints $\mathbf{u}(t) \in \mathbb{U}\big(\mathbf{x}(t), t\big) \subseteq \mathcal{U}$ are time-variant.

A. Local Planning via Optimal Control

The planning task is to guide the robotic system (1) from $\mathbf{x}_\mathrm{s} \in \mathcal{X}$ at time t_s to an intermediate or ultimate goal set $\mathbb{X}_\mathrm{f}(t_\mathrm{s}) \subset \mathcal{X}$ within time $t_\mathrm{f} \in I$ while minimizing an objective function and adhering to constraints. Hereby, $I \subseteq \mathbb{R}$ denotes a predefined time interval containing t_s. With running cost $\ell : \mathcal{X} \times \mathcal{U} \mapsto \mathbb{R}_0^+$ and terminal cost $J_\mathrm{f} : \mathcal{X} \mapsto \mathbb{R}_0^+$, the OCP is given as follows:

$$\min_{\mathbf{u}(t),\, \mathbf{x}(t),\, t_\mathrm{f}} \quad J_\mathrm{f}\big(\mathbf{x}(t_\mathrm{f})\big) + \int_{t=t_\mathrm{s}}^{t_\mathrm{f}} \ell\big(\mathbf{x}(t), \mathbf{u}(t)\big)\, \mathrm{d}t \tag{2}$$

subject to

$$\mathbf{x}(t_\mathrm{s}) = \mathbf{x}_\mathrm{s}, \quad \dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big), \quad \mathbf{x}(t_\mathrm{f}) \in \mathbb{X}_\mathrm{f}(t_\mathrm{s}),$$
$$\mathbf{x}(t) \in \mathbb{X}(t),\ \mathbf{u}(t) \in \mathbb{U}\big(\mathbf{x}(t), t\big),\ \dot{\mathbf{u}}(t) \in \mathbb{U}_\mathrm{d},\ t_\mathrm{f} \in I.$$

State and control constraints are included as described before. In addition, the control derivative $\dot{\mathbf{u}}(t)$ is restricted to a given set $\mathbb{U}_\mathrm{d}$. Note, the same result can be achieved by augmenting the state space with integrator dynamics. However, in robot applications the system is often described by a kinematic model with velocities as input, and therefore such a constraint (i.e. acceleration limits) eliminates the need to increase the number of optimization parameters. OCP (2) is generic at this point and includes the most common objectives like minimizing time or minimizing control error and effort (quadratic form). These are detailed in Section III. Note that potential fields for increasing the distances to obstacles or attraction terms for approaching waypoints may also be included in $\ell(\cdot)$.

B. Feedback Control

An MPC-based local planner provides control actions directly to the robot or to a cascaded low-level controller. In practice, (2) can only be solved at discrete time instances and hence the control law is defined according to the grid $t_0 < t_1 < \ldots < t_n < \ldots < \infty$ with $n \in \mathbb{N}_0$ and $t_n \in \mathbb{R}_0^+$. The control law $\mu : \mathcal{X} \mapsto \mathcal{U}$ for $t \in [t_n, t_{n+1})$ is given by:

$$\mu\big(\mathbf{x}(t)\big) := \mathbf{u}^*(t)\big|_{\mathbf{x}_\mathrm{s}=\mathbf{x}(t_n), t_s=t_n}. \tag{3}$$

Hereby, $\mathbf{u}^*(t)$ denotes the resulting optimal control trajectory from OCP (2) with substitutions $\mathbf{x}_\mathrm{s} = \mathbf{x}(t_n)$ and $t_s = t_n$. The current state $\mathbf{x}(t_n)$ is either directly measurable or estimated by a state observer.

III. NUMERICAL REALIZATION

This section transforms the generic motion planning problem into a numerically tractable realization.

A. Alternative State Spaces

Common MPC formulations usually assume that the state space is Euclidean (resp. the real *n*-space). But especially in robotics, state spaces often include rotational components which span over the non-Euclidean rotation groups *SO(2)* or *SO(3)*. A possible approach is to provide an over-parameterized formulation with constraints. However, local derivative-based optimization schemes apply local increments $\mathbf{z}^* = \mathbf{z} + \Delta\mathbf{z}$ with parameter $\mathbf{z} \in \mathbb{R}^M$ and increment $\Delta\mathbf{z} \in \mathbb{R}^M$ over an M-dimensional Euclidean parameter space in each solver iteration [20] without maintaining any of these constraints. To overcome these difficulties, we adapt and extend the idea of alternative parameterizations from graph-based SLAM [21], which we have already successfully applied in simplified form in the TEB implementation. The idea follows the observation that $\Delta\mathbf{z}$ is usually a small perturbation around $\mathbf{z}$ and hence far from singularities. Therefore, $\Delta\mathbf{z}$ represents a minimal representation computed in the local Euclidean surroundings of $\mathbf{z}$. A nonlinear increment operator then converts and applies $\Delta\mathbf{z}$ to the correct space. Due to the limited scope of this paper, we restrict ourselves to 2D rotation groups *SO(2)* which applies to most robot navigation scenarios. In this case, even $\mathbf{z}$ does not need to

be over-parameterized but the nonlinear increment operator still maintains proper 2D rotations.

Consider a state $\mathbf{x} \in \mathcal{X}$. An increment $\Delta\mathbf{x} \in \mathbb{R}^p$ is then applied by the nonlinear increment operator $\boxplus : \mathcal{X} \times \mathbb{R}^p \mapsto \mathcal{X}$:

$$\mathbf{x}^* = \mathbf{x} \boxplus \Delta\mathbf{x}. \tag{4}$$

Hereby, $\mathbb{R}^p$ is a local Euclidean space. To properly evaluate the system dynamics in the alternative state space, the difference $\mathbf{x}_2 - \mathbf{x}_1$ for $\mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}$ must be embedded into $\mathcal{X}$. This is achieved by defining a nonlinear difference operator $\boxminus : \mathcal{X} \times \mathcal{X} \mapsto \mathcal{X}$ such that $\delta\mathbf{x} = \mathbf{x}_2 \boxminus \mathbf{x}_1$. Note that no singularities occur for rotational components in *SO(2)* and the operator defines the shortest angular distance between two states without discontinuities. In this work we apply these operators only to states, but the extension to alternative control input spaces is done in exactly the same way.

Example 1: As an example consider the special Euclidean group 2, i.e. $\mathcal{X} = \textit{SE(2)} = \mathbb{R}^2 \times \textit{SO(2)}$ that is defined in terms of a 2D translation in $\mathbb{R}^2$ and a rotational component. The state vector is defined as $\mathbf{x} = (x, y, \theta)^\intercal \in \mathcal{X}$ with $\theta \in [-\pi, \pi)$. Let $\mathrm{normAngle}(\varphi)$ define a function that normalizes an angle $\varphi \in \mathbb{R}$ to the interval $[-\pi, \pi)$. Then operator $\boxplus$ is specified as:

$$\mathbf{x} \boxplus \Delta\mathbf{x} := \big(x + \Delta x, y + \Delta y, \mathrm{normAngle}(\theta + \Delta\theta)\big)^\intercal. \tag{5}$$

Accordingly, the difference operator is: $\mathbf{x}_2 \boxminus \mathbf{x}_1 := \mathbf{x}_2 \boxplus -\mathbf{x}_1$.

B. Direct Transcription

This section applies direct transcription [10] in combination with the previously defined increment and difference operators to convert OCP (2) to an NLP. The time interval $[t_\mathrm{s}, t_\mathrm{f}]$ of the planning horizon is now discretized according to the following grid: $t_\mathrm{s} = t_0 \leq t_1 \leq \ldots \leq t_k \leq \ldots \leq t_N = t_\mathrm{f}$ with $t_k \in I$, $k = 0, 1, \ldots, N$ and $N \in \mathbb{N}$. Note that index k indicates that the context belongs to prediction rather than closed-loop control. Furthermore, $\Delta t_k = t_{k+1} - t_k$ denotes the time interval for an individual grid partition. In this paper, direct transcription relies on a piecewise constant control trajectory with respect to the temporal grid, i.e. $\mathbf{u}(t) := \mathbf{u}_k = \text{constant}$ for $t \in [t_k, t_k + \Delta t_k)$ for $k = 0, 1, \ldots, N-1$. The states at grid points t_k are denoted as $\mathbf{x}(t_k) := \mathbf{x}_k$ for $k = 0, 1, \ldots, N$.

Several methods exist to discretize the boundary value problem in OCP (2) induced by system (1). Established methods are multiple shooting and collocation [10]. Whereas multiple shooting usually applies explicit integration schemes, collocation mainly refers to implicit schemes. In the following, we utilize low order collocation via finite-differences as they provide a reasonable trade-off between computational resources and accuracy and they are directly suitable for the nonlinear difference operator $\boxminus$.

The system dynamics error on the k^{th} grid partition is approximated by a basis function $\boldsymbol{\phi}(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k, \Delta t_k) = (\mathbf{x}_{k+1} \boxminus \mathbf{x}_k)\Delta t_k^{-1} - \boldsymbol{\xi}(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k)$ with a suitable finite difference kernel $\boldsymbol{\xi}(\cdot)$, i.e. either forward differences (explicit, first order, similar to multiple shooting with forward Euler and full discretization):

$$\boldsymbol{\xi}(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) := \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k), \tag{6}$$

or Crank-Nicolson differences (implicit, second order):

$$\boldsymbol{\xi}(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k) := 0.5\big(\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \mathbf{f}(\mathbf{x}_{k+1}, \mathbf{u}_k)\big). \tag{7}$$

By further approximating the integral cost in (2) by the right Riemann sum and $\dot{\mathbf{u}}(t)$ by forward differences, the resulting NLP is given as follows:

$$\min_{\substack{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}, \\ \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_N, \\ \Delta t_0, \Delta t_1, \ldots, \Delta t_{N-1}}} J_\mathrm{f}\big(\mathbf{x}_N\big) + \sum_{k=0}^{N-1} \ell(\mathbf{x}_k, \mathbf{u}_k)\Delta t_k \tag{8}$$

subject to

$$\mathbf{x}_0 = \mathbf{x}_\mathrm{s},\ \mathbf{x}_N \in \mathbb{X}_\mathrm{f}(t_\mathrm{s}),\ \mathbf{x}_k \in \mathbb{X}(k\Delta t_k),\ \mathbf{u}_k \in \mathbb{U}\big(\mathbf{x}_k, k\Delta t_k\big),$$
$$(\mathbf{u}_{k+1} - \mathbf{u}_k)\Delta t_k^{-1} \in \mathbb{U}_\mathrm{d},\ (\mathbf{u}_0 - \mathbf{u}_\mathrm{p})\Delta t_\mathrm{p}^{-1} \in \mathbb{U}_\mathrm{d},$$
$$\Delta t_{\text{min}} \leq \Delta t_0 \leq \Delta t_{\text{max}}, \quad \Delta t_k = \Delta t_{k+1},$$
$$\boldsymbol{\phi}(\mathbf{x}_{k+1}, \mathbf{x}_k, \mathbf{u}_k, \Delta t_k) = \mathbf{0}, \quad k = 0, 1, \ldots, N-1.$$

The limitation of the first control w.r.t. its derivative at t_0 requires the specification of the previous control input $\mathbf{u}_\mathrm{p} \in \mathcal{U}$ and the time elapsed since $\mathbf{u}_\mathrm{p}$, i.e. $\Delta t_\mathrm{p} = t_n - t_{n-1}$. In the very first closed-loop step (3), $\mathbf{u}_\mathrm{p}$ is set to zero or some control for which $\mathbf{f}(\mathbf{x}_\mathrm{s}, \mathbf{u}_\mathrm{p}) = \mathbf{0}$ holds. The same holds for the last control $\mathbf{u}_N$ which is not subject to optimization but provides the final boundary value for the control derivative. This way, a robot with velocity input always plans to stop at the end of the horizon for safety reasons. Note that this is not conventional in receding-horizon MPC as this inherently leads to differing open-loop and closed-loop solutions and might affect convergence to $\mathbb{X}_\mathrm{f}(t_\mathrm{s})$. However, with terminal equality conditions as in the time-optimal variant shown later, this does not lead to changes in terms of convergence due to the optimality principle. Condition $t_\mathrm{f} \in I$ in (2) has been replaced by $\Delta t_{\text{min}} \leq \Delta t_0 \leq \Delta t_{\text{max}}$ with bounds $\Delta t_{\text{min}}, \Delta t_{\text{max}} \in \mathbb{R}_0^+$. Furthermore, condition $\Delta t_k = \Delta t_{k+1}$ ensures uniformity between all time intervals. Notice that for the set evaluations $t_k = k\Delta t_0 = k\Delta t_k$ holds due to the uniform grid and choosing $k\Delta t_k$ preserves the local structure of the optimization problem.

Remark 1: NLP (8) is derived w.r.t. individual optimization parameters for each time interval but with constraints enforcing uniformity (local uniform grid approach). Another formulation, the global uniform grid, is obtained by replacing all Δt_k with a single time parameter Δt and omitting constraints $\Delta t_k = \Delta t_{k+1}$. The optimal solution is identical, but the structure of the optimization problem differs slightly. For more details refer to [22].

C. Receding-Horizon Quadratic-Form MPC

The most common MPC realization is defined in terms of a quadratic-form objective to minimize the quadratic control error and effort. To account for the alternative state space representations defined in Section III-A, the control error metric must be adjusted. Common metrics for rotation groups

are provided in [1], but in this case we include the nonlinear difference operator in the quadratic form. The terminal and running costs are given as follows:

$$J_{\mathrm{f}}(\mathbf{x}_N) = (\mathbf{x}_N \boxminus \mathbf{x}_{\mathrm{f}})^{\intercal} \mathbf{Q}_{\mathrm{f}} (\mathbf{x}_N \boxminus \mathbf{x}_{\mathrm{f}}), \tag{9}$$

$$\ell(\mathbf{x}_k, \mathbf{u}_k) = (\mathbf{x}_k \boxminus \mathbf{x}_{\mathrm{f}})^{\intercal} \mathbf{Q} (\mathbf{x}_k \boxminus \mathbf{x}_{\mathrm{f}}) + \mathbf{u}_k^{\intercal} \mathbf{R} \mathbf{u}_k. \tag{10}$$

Hereby, $\mathbf{Q}, \mathbf{Q}_{\mathrm{f}} \in \mathbb{R}^{p \times p}$ and $\mathbf{R} \in \mathbb{R}^{q \times q}$ denote weighting matrices with state dimension $p = \dim(\mathcal{X})$ and input dimension $q = \dim(\mathcal{U})$, respectively. Furthermore, $\mathbf{x}_{\mathrm{f}} \in \mathbb{X}_{\mathrm{f}}(t_n)$ represents the current intermediate goal state. In case the full goal state is not available, it is possible to either choose additional states to complete a steady state or to set related components in $\mathbf{Q}$ and $\mathbf{Q}_{\mathrm{f}}$ to zero. Note that $\mathbf{x}_{\mathrm{f}}$ could also be replaced by a time-dependent reference $\mathbf{x}_{\mathrm{f}}(t)$ during optimization for tracking control. In this MPC realization, the horizon length resp. final time t_{f} is fixed to $I = \{N \Delta t_{\mathrm{s}}\}$ which implies $\Delta t_{\mathrm{min}} = \Delta t_{\mathrm{max}} = \Delta t_k = \Delta t_{\mathrm{s}}$ with a predefined grid resolution $\Delta t_{\mathrm{s}} \in \mathbb{R}^{+}$. Therefore the optimization w.r.t. time becomes obsolete and thus the optimization parameters Δt_k with $k = 0, 1, \ldots, N-1$ as well as temporal constraints are excluded to reduce the dimensions of the optimization problem.

Stability and recursive feasibility conditions for MPC usually rely on proper terminal conditions $J_{\mathrm{f}}(\cdot)$ resp. $\mathbf{Q}_{\mathrm{f}}$ and $\mathbb{X}_{\mathrm{f}}(t_n)$ to, e.g., approximate an infinite horizon. The interested reader is referred to [23] and notice that a sampled-data discrete-time representation follows by setting $\Delta t_{\mathrm{s}} = 1$. For (output) path-following MPC, [17] provides terminal conditions and conditions on the reference path under which stability is guaranteed. However, as a suitable terminal region is difficult to determine, most practical approaches usually relay on sufficiently long horizon lengths to ensure stability and feasibility and choose $\mathbb{X}_{\mathrm{f}}(t_n) := \mathcal{X}$. For static environments and a class of robotic systems, i.e. differential drive robots, [24] guarantees stability for shorter receding horizons.

D. Time-Optimal MPC

Another interesting realization of NLP (8) is a pure time-optimal MPC as addressed for real spaces in [22]. By setting $J_{\mathrm{f}}(\mathbf{x}_N) := 0$ and $\ell(\mathbf{x}_k, \mathbf{u}_k) := 1$ in (8), the resulting state trajectory reaches $\mathbb{X}_{\mathrm{f}}(t_n)$ in minimum time. Δt_{max} is set to a worst case accuracy for the system dynamics error or is even omitted ($\Delta t_{\mathrm{max}} = \infty$) as the grid adaptation scheme described below better controls the accuracy in changing environments. For example, approaching obstacles extend the trajectory (longer traveling times for detours) while obstacles moving away contract the trajectory due to time-optimality. Similar as for the TEB approach [14], the temporal resolution Δt_k is adapted during closed-loop control w.r.t. a given reference $\Delta t_{\mathrm{s}} \in \mathbb{R}^{+}$ and hysteresis $\Delta t_{\epsilon} \in \mathbb{R}^{+}$:

$$N_{n+1} = \begin{cases} N_n + 1 & \Delta t_n^* > \Delta t_{\mathrm{s}} + \Delta t_{\epsilon} \\ \max(N_n - 1, N_{\mathrm{min}}) & \Delta t_n^* < \Delta t_{\mathrm{s}} - \Delta t_{\epsilon} \end{cases}.$$

The grid size at closed-loop time t_n is denoted by N_n with initial size $N_0 = N \geq N_{\mathrm{min}}$ and safe guard $N_{\mathrm{min}} \in \mathbb{N}$. Δt_n^* denotes the time interval obtained from the solution of NLP (8) at closed-loop time t_n. Note that this linear search lowers the changes between subsequent solver calls (numerical robustness), but requires some iterations to react on highly dynamic environments. In practice, changing the grid size one by one pointed out to be sufficient as the control rate is much faster than the changes in the environment. If this is not the case, grid adaptation may change N_n further or, alternatively, an inner loop terminates NLP (8) prior to convergence and an outer loop adapts the grid size.

For time-optimal control, the terminal condition is often set to the intermediate goal state $\mathbf{x}_{\mathrm{f}}$, i.e. $\mathbb{X}_{\mathrm{f}} = \{\mathbf{x}_{\mathrm{f}}\}$. Closed-loop convergence and recursive feasibility results are provided in [22].

E. Obstacle Avoidance Constraints

In this work, obstacles are defined as simply connected regions $\mathcal{O}_l(t), l = 1, \ldots, L$ with L as the number of obstacles. The time-dependency explicitly accounts for dynamic obstacles. Furthermore, let $\mathcal{R}(\mathbf{x})$ denote a geometric robot collision model dependent on the state $\mathbf{x} \in \mathcal{X}$, e.g. position and orientation. Usually these sets are embedded in *SE(2)* or *SE(3)*. The minimum distance $d_l : \mathcal{X} \times \mathbb{R} \mapsto \mathbb{R}_0^+$ between the robot and obstacle l is then determined by:

$$d_l(\mathbf{x}, t) = \min_{p_1 \in \mathcal{R}(\mathbf{x}), p_2 \in \mathcal{O}_l(t)} \|p_2 - p_1\|_2. \tag{11}$$

By specifying a minimum separation to obstacles $d_{\mathrm{min}} \in \mathbb{R}_0^+$, the collision-free state space $\mathbb{X}_{\mathrm{o}}(t)$ is given as follows:

$$\mathbb{X}_{\mathrm{o}}(t) = \{\mathbf{x} \in \mathcal{X} \mid d_l(\mathbf{x}, t) \geq d_{\mathrm{min}}, l = 1, 2, \ldots, L\}. \tag{12}$$

Furthermore, let the robot's internal state constraints are described by set $\mathbb{X}_{\mathrm{i}}(t) \in \mathcal{X}$, then $\mathbb{X}(t)$ in (2) and (8) is given by $\mathbb{X}(t) = \mathbb{X}_{\mathrm{o}}(t) \cap \mathbb{X}_{\mathrm{i}}(t)$. Note that there exist techniques in the literature for computationally efficient optimization-based obstacle avoidance, e.g. [18].

F. Solution of the Nonlinear Program

For NLP (8) general necessary and sufficient optimality conditions apply [20]. Any practical implementation replaces constraint sets $\mathbb{X}, \mathbb{U}, \mathbb{U}_{\mathrm{d}}$ and $\mathbb{X}_{\mathrm{f}}$ by algebraic equality and inequality functions. The NLP is solved by standard local constrained optimization techniques, such as interior point methods or sequential quadratic programming. For the time-optimal formulation even sequential linear programming can be applied. Warm-starting the NLPs in subsequent closed-loop steps with the previous solution significantly increases performance. To comply with the alternative parameterizations (Section III-A), the solvers need to be modified in a few places. Let $\mathbf{z} = (\mathbf{u}_0, \mathbf{x}_0, \Delta t_0, \ldots, \mathbf{u}_{N-1}, \mathbf{x}_{N-1}, \Delta t_{N-1}, \mathbf{x}_N)$ define the optimization parameter vector of (8). Local solutions $\mathbf{z}^* = \mathbf{z} \boxplus \Delta\mathbf{z}$ after each solver step follow by incrementing components as follows: $\mathbf{u}_k + \Delta\mathbf{u}_k$, $\Delta t_k + \Delta\Delta t_k$ and $\mathbf{x}_k \boxplus \Delta\mathbf{x}_k$. Also the calculation of Jacobians and gradients w.r.t. $\mathbf{x}$ requires to apply proper increments. E.g. the Jacobian of some vector valued state-dependent function $\mathbf{g}(\cdot)$ evaluated at $\bar{\mathbf{x}}$ is given by:

$$\mathrm{D}_{\mathbf{x}}\, \mathbf{g}(\bar{\mathbf{x}}) = \left. \frac{\partial \mathbf{g}(\bar{\mathbf{x}} \boxplus \Delta\mathbf{x})}{\partial \Delta\mathbf{x}} \right|_{\Delta\mathbf{x} = \mathbf{0}} \tag{13}$$

Since second-order derivatives can be calculated numerically using quasi-Newton methods or finite differences from first-order information, no dedicated increment is required.

Remark 2: Note that the nonlinear increment in (13) can be omitted for *SO(2)* components in case $\mathbf{g}$ inherently normalizes the components. For example, consider the collocation constraint $\mathbf{g} := \boldsymbol{\phi}(\cdot)$: Here $\mathbf{x}_{k+1} \boxminus \mathbf{x}_k$ also applies to unnormalized components and the robot system dynamics $\mathbf{f}(\cdot)$ often considers rotational components only within trigonometric functions. The same applies to (9), (10) and standard constraints including obstacle avoidance.

The structure of NLP (8) is sparse and utilizing sparse algebra for efficient optimization is crucial. While established NLP solvers like IPOPT [25] already support sparse algebra, it is possible to further speed up optimization by computing derivative information only for structured non-zeros. Similar to the TEB, NLP (8) is represented as a hypergraph with cost functions and constraints as hyperedges and states, controls and time invervals as vertices. This facilitates the computation of sparse finite differences by iterating the edge set. A major advantage of the hypergraph is its simple algorithmic reconfiguration at negligible overhead. This is crucial for real-time control as the NLP dimensions change during runtime either due to changing environments or grid adaptation. Refer to [26] for general performance results and a detailed description on how to formulate NLPs in MPC as hypergraph.

G. Local Planning in Distinctive Topologies

Solving NLP (8) as described before leads to locally optimal solutions. Many local minima are caused by the presence of obstacles that imply non-convexity. Identifying these local minima in advance coincides with exploring and analyzing distinctive topologies between start and goal poses. Due to the limited scope of this paper, we cannot provide a detailed description, but would like to point out that distinctive topologies (homotopy classes) can be managed during runtime analogous to [14] for $\mathbb{X}_\mathrm{f} = \{\mathbf{x}_\mathrm{f}\}$. Several NLPs (8) with state trajectories initialized for each active class are solved in parallel (multi-threaded). For dynamic obstacles and thus (x-y-t) spaces, the equivalence relation that identifies those classes is changed to a 3D variant according to [27]. And for evaluating this relation, sampled roadmaps in [14] are augmented with temporal profiles (maximum velocities).

IV. EVALUATION

The proposed MPC-based planning approach copes with arbitrary system dynamics (1) for states embedded in *SE(2)* or any larger space containing *SE(2)/SO(2)* subsets. Note that this includes common kinematic and dynamic models of mobile robots, vehicles and ships with velocity, acceleration, jerk, force and torque input spaces. The following sections examine two common applications in more detail.

A. Kinematic Bicycle Model

For motion planning and control in autonomous driving, the kinematic bicycle model is often preferred over dynamic single or double track models due their sufficiently high accuracy over a large range of operation and less computational burden [28]. Let $x \in \mathbb{R}$ and $y \in \mathbb{R}$ define the coordinates of the center of mass and $\theta \in$ *SO(2)* the inertial heading of the vehicle. Furthermore, $l_\mathrm{f} \in \mathbb{R}_0^+$ and $l_\mathrm{r} \in \mathbb{R}_0^+$ are the distances from the center of mass to the front and rear axles, respectively. Given a front steering angle $\delta \in (-\pi/2, \pi/2)$, the angle β between the current velocity vector at the center of mass and the direction of the longitudinal axis of the vehicle is given by:

$$\beta(t) = \tan^{-1}\left(\frac{l_\mathrm{r}}{l_\mathrm{l} + l_\mathrm{r}} \tan\big(\delta(t)\big)\right). \tag{14}$$

With state vector $\mathbf{x}(t) = \big(x(t), y(t), \theta(t)\big)^\intercal$, vehicle speed $v(t) \in \mathbb{R}$ and control input $\mathbf{u}(t) = \big(v(t), \delta(t)\big)^\intercal$, the kinematic bicycle model is defined by [28]:

$$\dot{\mathbf{x}}(t) = \mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) = \begin{pmatrix} v(t)\cos\big(\theta(t) + \beta(t)\big) \\ v(t)\sin\big(\theta(t) + \beta(t)\big) \\ \frac{v(t)}{l_\mathrm{r}}\sin\big(\beta(t)\big) \end{pmatrix}. \tag{15}$$

Note, for this evaluation we choose speed v as input rather than the acceleration $\dot{v}$ in contrast to [28] and hence omit the additional integrator dynamics resp. state. Limits on accelerations are still included by control deviation bounds.

The planning task constitutes an automated parking scenario, since the advantages of proper planning in *SE(2)/SO(2)* are particularly apparent here. When planning overtaking and (non-u-shape) turning maneuvers, θ usually does not reach $\pm\pi$ w.r.t. the local planning coordinate system. Vehicle parameters l_f =1.1 m and l_r =1.7 are borrowed from [28] for a Hyundai Azera.

Control inputs are restricted by $\mathbb{U}$, i.e. $|v(t)| \leq 4\,\mathrm{m/s}$ and $|\delta(t)| \leq 0.65\,\mathrm{rad/s}$. Set $\mathbb{U}_\mathrm{d}$ represents control deviation bounds ensuring $-3\,\mathrm{m/s^2} \leq \dot{v}(t) \leq 1.5\,\mathrm{m/s^2}$ and $|\dot{\delta}(t)| \leq 0.31\,\mathrm{rad/s^2}$. Internal states are not restricted, i.e. $\mathbb{X}_\mathrm{i}(t) := \mathcal{X}$. The vehicle's start state is $\mathbf{x}_\mathrm{s} = (1\,\mathrm{m}, 1.75\,\mathrm{m}, -3.1\,\mathrm{rad})^\intercal$ with $\mathbf{u}_\mathrm{p} = \mathbf{0}$ and its parking destination is $\mathbf{x}_\mathrm{f} = (-4\,\mathrm{m}, -6\,\mathrm{m}, 1.57\,\mathrm{rad})^\intercal$ with $\mathbf{u}_\mathrm{N} = \mathbf{0}$. Road boundaries and the parking lot are modeled by six line segments according to Fig. 2 and define obstacle sets $\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_6$. The road width is 3 m and the depth of the parking lot is 4.75 m. A pill resp. stadium-shape footprint with $w = l_\mathrm{r} + l_\mathrm{f}$ and $r = 0.9\,\mathrm{m}$ serves as collision model:

$$\mathcal{R}(\mathbf{x}) = \left\{\mathbf{p} \in \mathbb{R}^2 \,\middle|\, \left\| \begin{pmatrix} x + (\mu w - l_\mathrm{r})\cos\theta \\ y + (\mu w - l_\mathrm{r})\sin\theta \end{pmatrix} - \mathbf{p} \right\|_2 \leq r \,\forall \mu \in [0,1] \right\}.$$

To demonstrate that the approach is also suitable for dynamic obstacles, although the time during optimization can be variable, the following obstacle with constant speed $v_\mathrm{o} = 1\,\mathrm{m/s}$ is added. The start position is on the other lane at $x_\mathrm{o} = -13\,\mathrm{m}$ and $y_\mathrm{o} = -1.25\,\mathrm{m}$. Its time-dependent collision model with $r_\mathrm{o} = 0.9\,\mathrm{m}$ and $l_\mathrm{o} = 2.5\,\mathrm{m}$ is given as follows:

$$\mathcal{O}_7(t) = \left\{\mathbf{p} \in \mathbb{R}^2 \,\middle|\, \left\| \begin{pmatrix} x_\mathrm{o} + \mu l_\mathrm{o} + t v_\mathrm{o} \\ y_\mathrm{o} \end{pmatrix} - \mathbf{p} \right\|_2 \leq r_\mathrm{o} \,\forall \mu \in [0,1] \right\}.$$

The minimum separation is set to $d_\mathrm{min} = 0.2\,\mathrm{m}$. We choose a variable time realization according to Section III-D. We merely extend the running costs by a small weighted term for

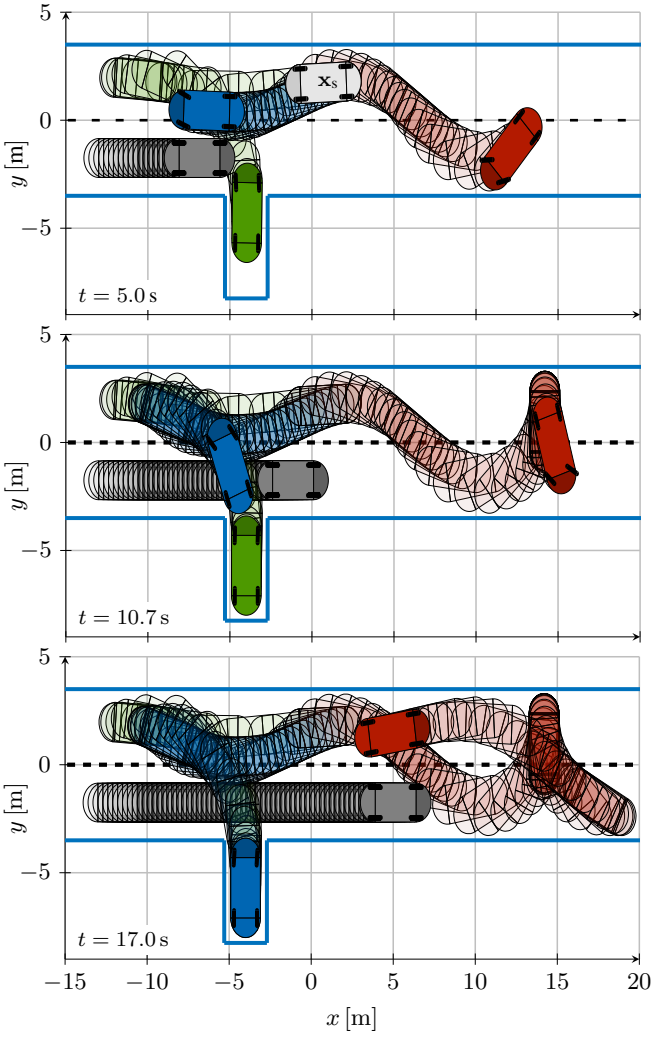

Fig. 2: Quasi-time-optimal parking maneuver with dynamic obstacle (gray). Topologically distinctive solutions in (x-y-t) are shown in blue and green. The red solution is obtained with standard Euclidean optimization. The initial pose $\mathbf{x}_s$ is shown in the top plot and the temporal resolution of traces is 0.3 s. The front part of each vehicle is filled with slightly darker color.

the control effort, making it a hybrid objective: $\ell(\mathbf{x}_k, \mathbf{u}_k) = 1 + \mathbf{u}_k^\mathsf{T}\mathbf{R}\mathbf{u}_k$ with $\mathbf{R} = \mathrm{diag}(0.01, 0)$. The low weight still leads to quasi-time-optimal solutions, but the vehicle tends to prefer braking to swerving out, especially if it is waiting for the dynamic obstacle to pass. Local planning is based on Crank-Nicolson differences (7) and includes the final goal state as terminal condition, i.e. $\mathbb{X}_\mathrm{f} = \{\mathbf{x}_\mathrm{f}\}$. As the state space is the *SE(2)*, operator overloading follows from Example 1. Further parameters are $N = 50$, $\Delta t_\mathrm{s} = \Delta t_\mathrm{p} = 0.1\,\mathrm{s}$, $\Delta t_\mathrm{min} = 0.001\,\mathrm{s}$, $\Delta t_\mathrm{max} = \infty$, $N_\mathrm{min} = 2$, $\Delta t_\epsilon = 0.1\Delta t_\mathrm{s}$.

Fig. 2 shows the open-loop solutions obtained after convergence for two topologically distinctive initializations resp. homotopy classes (HC) in (x-y-t) space. The related control input and orientation profiles are depicted in Fig. 3. While in the first HC the ego vehicle slows down and waits for the obstacle to pass (blue), in the second HC it reaches the parking lot before the obstacle passes (green). Note, the variable time horizon is very advantageous here to find feasible trajectories that ultimately reach $\mathbf{x}_\mathrm{f}$. We would like to point out that the chosen obstacle distances are deliberately kept small in order to demonstrate the abilities and limit mathematical exposition. Adding further potential fields to the running cost and preferring control effort to time optimality increases safety and comfort in practice. The third solution (red) follows from solving (8) without proper nonlinear operators for *SO(2)* components. The orientation $\theta(t)$ continuously rotates from -3.1 rad to 1.57 rad. In contrast, $\theta(t)$ in HC1 and HC2 jumps without affecting the local continuity of the optimization problem itself. Note that it would also be possible to get the desired behavior in this scenario by adjusting the start and goal orientations by multiples of 2π. However, this additional logic is error-prone and not as flexible and complete as the explicit consideration in the optimization.

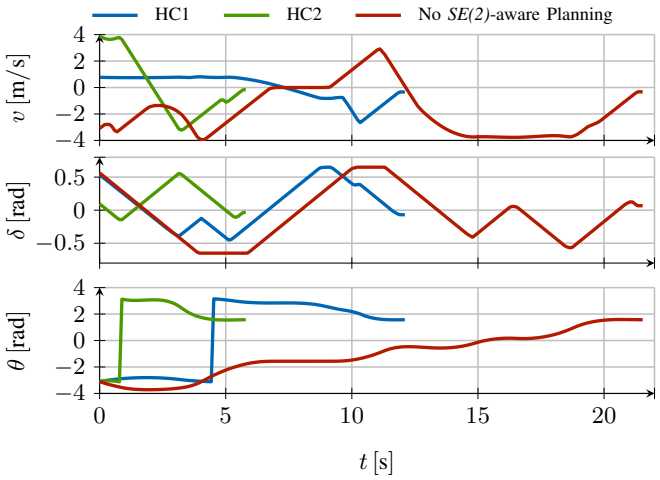

Fig. 3: Control input and orientation profiles for the parking scenario in Fig. 2

B. Comparative Analysis for a Differential Drive Robot

The parking scenario with the bicycle model shows how versatile the approach is. The TEB approach also supports car-like robots, but only with a simple model and without steering rate limits, so the previous example would not have been feasible. This makes our approach suitable for especially more complex models and scenarios. In this section, we conduct a comparative analysis for differential drive robots in a common hierarchical planning framework. Simulative experiments are carried out with the navigation stack in ROS, for which several local planners for differential drive robots exist, e.g. the classic EB, DWA resp. trajectory rollout, TEB, and our new MPC-based planner[1]. The simulation environment is *stage* in ROS which runs in a separate process and thus also simulates communication and calculation delays.

[1] http://wiki.ros.org/{eband,dwa,teb,mpc}_local_planner

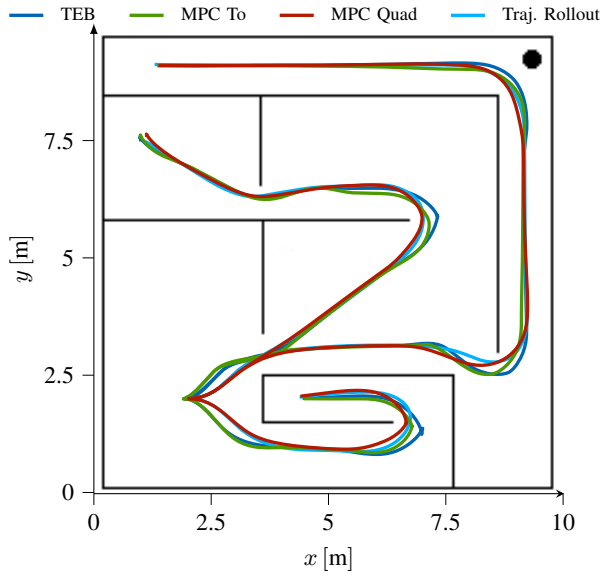

Fig. 4: Navigation of a differential drive robot for three different goals and several methods

The circular robot with collision model

$$\mathcal{R}(\mathbf{x}) = \left\{\mathbf{p} \in \mathbb{R}^2 \mid \|(x,y)^\intercal - \mathbf{p}\|_2 \leq 0.17\,\mathrm{m}\right\} \tag{16}$$

is initially located at $(2\,\mathrm{m}, 2\,\mathrm{m}, 0\,\mathrm{rad})^\intercal$. The navigation task is to drive to three goals located at $(1\,\mathrm{m}, 7.5\,\mathrm{m}, \pi/2\,\mathrm{rad})^\intercal$, $(1.2\,\mathrm{m}, 9.1\,\mathrm{m}, 3.14\,\mathrm{rad})^\intercal$ and $(4.3\,\mathrm{m}, 2.0\,\mathrm{m}, -3.14\,\mathrm{rad})^\intercal$ in a $10\,\mathrm{m} \cdot 10\,\mathrm{m}$ map as shown in Fig. 4. The kinematic model in *SE(2)* and linear and angular velocities as input $\mathbf{u} = (v, \omega)^\intercal$ is given by:

$$\mathbf{f}\big(\mathbf{x}(t), \mathbf{u}(t)\big) = \big(v(t)\cos\big(\theta(t)\big), v(t)\sin\big(\theta(t)\big), \omega(t)\big)^\intercal .$$

Considered limits are $-0.2\,\mathrm{m/s} \leq v(t) \leq 0.4\,\mathrm{m/s}$, $|\omega(t)| \leq 0.4\,\mathrm{rad/s}$, $|\dot{v}(t)| \leq 0.25\,\mathrm{m/s^2}$ and $|\dot{\omega}(t)| \leq 0.25\,\mathrm{m/s^2}$. Hierarchical planning is realized by the standard Dijkstra-based global planner in ROS which refreshes its path every 2 s. The local planners operate at 10 Hz in a local costmap of size $5\,\mathrm{m} \cdot 5\,\mathrm{m}$ and a resolution of $0.1\,\mathrm{m/cell}$ centered at the current robot location. Intermediate goals for local planning, i.e. $\mathbf{x}_\mathrm{f}$, are centered on the global path within the local costmap and a lookahead distance of $1.5\,\mathrm{m}$. The planner parameters are almost set to the default setting in addition to those that meet the above mentioned kinodynamic constraints correctly.

Note that unlike the other planners, the classic EB is a pure path planning method, so the package also implements a PID-based path following controller. Modified parameters are the minimum bubble overlap ($0.2\,\mathrm{m}$), costmap weight (20) and the velocity multiplier (8).

The DWA planner is configured in the computationally more expensive trajectory rollout mode, because the dynamic window does not work well with small acceleration limits like here and does not always converge. Besides default parameters, the horizon length is set to 2 s, the granularity to 0.1 s and the number of v- and ω-samples to 10 and 20, respectively. Note that this setting is not tuned for computational efficiency, but for comparable grid sizes.

TABLE I: Benchmark results for different planners

	CPU Time [ms]	Travel Time [s]	Path Length [m]	Control Effort
Elastic Band	10.2 [1.7, 18.3]	192.5	48.1	43.3
Traj. Rollout	19.1 [14.8, 22.1]	122.1	42.1	20.9
TEB	5.9 [3.0, 9.4]	126.9	44.2	21.9
MPC To	12.5 [5.4, 21.8]	124.6	43.1	22.3
MPC Quad	15.8 [7.5, 25.3]	116.0	41.6	20.4
MPC Hybrid	14.2 [7.6, 22.4]	140.1	41.9	17.5

For TEB and MPC, each occupied cell is treated as a single point obstacle. The number of occupied cells in all three navigation runs is 103 ± 27. Prior to each optimization step, further filtering associates each discrete state only with nearby obstacles on the left and right side. Three different configurations are considered for MPC:

i) A quadratic form realization according to Section III-C with $\mathbf{Q} = \mathbf{Q}_\mathrm{f} = \mathrm{diag}(1, 1, 0.25)$, $\mathbf{R} = \mathrm{diag}(2, 2)$, $\mathbb{X}_\mathrm{f} = \mathbb{X}$, $\Delta t_\mathrm{s} = 0.3\,\mathrm{s}$ and $N = 30$ (*MPC Quad*),
ii) A minimum-time realization according to Section III-D with $\mathbb{X}_\mathrm{f} = \{\mathbf{x}_\mathrm{f}\}$ (*MPC To*),
iii) A hybrid realization based on ii) with $\ell(\cdot) = 1 + \mathbf{u}_k^\intercal \mathbf{R} \mathbf{u}_k$ and $\mathbf{R} = \mathrm{diag}(2, 2)$ (*MPC Hybrid*).

The NLPs with forward differences (6) are warm-started and solved with the established C++ interior point solver IPOPT [25] and sparsity exploitation according to Section III-F.

Table I shows the benchmark results obtained with a PC running Ubuntu 18.04 (Intel Core i7-4770 CPU at 3.4 GHz, 8 GB RAM). Travel time, path length and control effort $\left(\int_0^\infty \mathbf{u}(t)^\intercal \mathbf{u}(t)\,\mathrm{d}t\right)$ are the accumulated absolute values of all three movements from start to the three goals. These values are comparable despite the classic EB. It is important to note that each approach can be further configured according to individual needs. All approaches successfully reach their goals. Trajectory rollout and quadratic form MPC reveal slightly faster travel times, but this is due to the non existing terminal conditions and therefore prefer cutting corners (cf. Fig. 4). More significant is the comparison of the CPU times, which are represented by their median and [0.05, 0.95]-quantiles. The TEB approach requires the least computational resources but also the MPC solutions are obtained in a relatively short time, enabling their application in real-world scenarios at common controller rates.

V. CONCLUSIONS

MPC is a powerful and highly customizable approach to robot motion planning. While approaches in the literature are usually based on purely Euclidean optimization spaces, the proposed nonlinear operator technique for the explicit treatment of rotational components in *SO(2)* during optimization points out to be crucial for holistic, generic and cross-scenario motion planning. Even though the CPU times for simple robotic applications (e.g. differential drive) are within the usual range, there are no significant advantages from a performance point of view, so that efficient planners such as the TEB approach need not be replaced. But as soon as

more complex kinematic or dynamic models or environments become necessary, the previous planners (at least in ROS) reach their limits. Our provided open-source C++ MPC framework with ROS integration is highly modular, versatile and computationally efficient.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. New York, USA: Cambridge University Press, 2006.

[2] F. Lamiraux, D. Bonnafous, and O. Lefebvre, "Reactive path deformation for nonholonomic mobile robots," *IEEE Transactions on Robotics*, vol. 20, no. 6, pp. 967–977, 2004.

[3] S. Quinlan, *Real-time modification of collision-free paths*. Stanford, CA, USA: Stanford University, 1995.

[4] V. Delsart and T. Fraichard, "Reactive Trajectory Deformation to Navigate Dynamic Environments," in *EUROPEAN ROBOTICS SYMPOSIUM*, 2008.

[5] B. Lau, C. Sprunk, and W. Burgard, "Kinodynamic motion planning for mobile robots using splines," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2009, pp. 2427–2433.

[6] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *IEEE Int. Conference on Robotics and Automation*, 2009.

[7] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *IEEE International Conference on Robotics and Automation*, 2011.

[8] S. Gulati, "A framework for characterization and planning of safe, comfortable, and customizable motion of assistive mobile robots," PhD thesis, 2011.

[9] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *IEEE International Conference on Robotics and Automation*, 2019, pp. 6741–6747.

[10] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*, 2nd ed., ser. Advances in Design and Control. Society for Industrial and Applied Mathematics, 2010.

[11] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[12] P. Ogren and N. E. Leonard, "A convergent dynamic window approach to obstacle avoidance," *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 188–195, 2005.

[13] C. Rösmann, W. Feiten, T. Wösch, F. Hoffmann, and T. Bertram, "Trajectory modification considering dynamic constraints of autonomous robots," in *German Conference on Robotics (ROBOTIK)*, 2012, pp. 74–79.

[14] C. Rösmann, F. Hoffmann, and T. Bertram, "Integrated online trajectory planning and optimization in distinctive topologies," *Robotics and Autonomous Systems*, vol. 88, pp. 142–153, 2017.

[15] C. Rösmann, F. Hoffmann, and T. Bertram, "Kinodynamic trajectory optimization and control for car-like robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2017, pp. 5681–5686.

[16] J. Deray, B. Magyar, J. Sol, and J. Andrade-Cetto, "Timed-elastic smooth curve optimization for mobile-base motion planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2019, pp. 3143–3149.

[17] T. Faulwasser and R. Findeisen, "Nonlinear model predictive control for constrained output path following," *IEEE Transactions on Automatic Control*, vol. 61, no. 4, pp. 1026–1039, 2016.

[18] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," 2017. arXiv: 1711.03449 [math.OC].

[19] T. Schoels, L. Palmieri, K. O. Arras, and M. Diehl, "An nmpc approach using convex inner approximations for online motion planning with guaranteed collision avoidance," 2019. arXiv: 1909.08267 [cs.RO].

[20] J. Nocedal and S. J. Wright, *Numerical optimization*, 2nd ed., ser. Springer series in operations research. New York: Springer, 2006.

[21] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *IEEE International Conference on Robotics and Automation*, 2011, pp. 3607–3613.

[22] C. Rösmann, A. Makarow, and T. Bertram, "Stabilizing quasi-time-optimal nonlinear model predictive control with variable discretization," 2020. arXiv: 2004.09561 [eess.SY].

[23] L. Grüne and J. Pannek, *Nonlinear model predictive control: Theory and algorithms*, 2nd ed., ser. Communications and Control Engineering. Springer, 2017.

[24] M. W. Mehrez, K. Worthmann, G. K. Mann, R. G. Gosine, and T. Faulwasser, "Predictive path following of mobile robots without terminal stabilizing constraints," in *IFAC World Congress*, 2017, pp. 9862–9857.

[25] A. Wächter and L. T. Biegler, "On the implementation of a primal-dual interior point filter line search algorithm for large-scale nonlinear programming," *Math. Programming*, vol. 106, no. 1, pp. 25–57, 2006.

[26] C. Rösmann, M. Krämer, A. Makarow, F. Hoffmann, and T. Bertram, "Exploiting sparse structures in nonlinear model predictive control with hypergraphs," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, 2018, pp. 1332–1337.

[27] S. Bhattacharya, M. Likhachev, and V. Kumar, "Identification and representation of homotopy classes of trajectories for search-based path planning in 3d," in *Robotics: Science and Systems*, 2011.

[28] J. Kong, M. Pfeiffer, G. Schildbach, and F. Borrelli, "Kinematic and dynamic vehicle models for autonomous driving control design," in *IEEE Intelligent Vehicles Symposium*, 2015, pp. 1094–1099.