IEEE Control Systems Letters paper presented at
2023 62nd IEEE Conference on Decision and Control (CDC)
December 13-15, 2023. Marina Bay Sands, Singapore

Dynamic and Nonlinear Programming for Trajectory Planning

Andreas Britzelmeier, Alberto De Marchi, and Rebecca Richter

***Abstract*— Direct optimal control techniques, relying on numerical methods for constrained optimization, are typically used in trajectory planning tasks in high-dimensional spaces. However, general-purpose solvers often fail to find a feasible solution when facing cluttered environments. Sampling- or graph-based methods, instead, can explore complex configuration spaces but struggle with dynamic constraints. Here, we propose to combine dynamic programming (DP) and derivative-based methods to reliably solve trajectory planning problems. Specifically, we exploit DP to generate a sequence of waypoints in a low-dimensional space, which are then encoded as pointwise path constraints for a high-dimensional trajectory, whose constraint violations are then represented as a penalty within the Bellman equation to recompute the waypoints. This iterative approach, alternating path and trajectory optimization, avoids both the curse of dimensionality for DP and problematic nonconvexities (such as obstacles) for motion planning. We demonstrate our strategy using numerical experiments on a six-degree-of-freedom robotic manipulator moving in a confined space.**

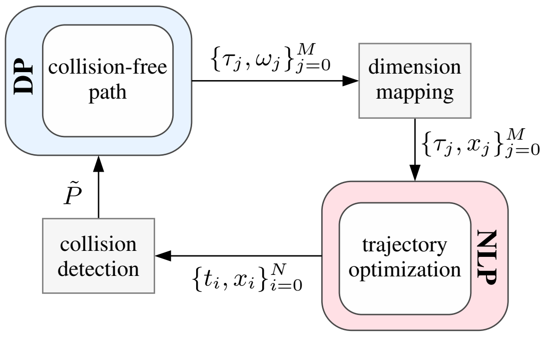

Fig. 1. Illustration of the proposed iterative scheme for motion planning, alternating between nonlinear programming (NLP) for trajectory optimization in the original space and dynamic programming (DP) in a lower-dimensional space.

I. Introduction

The task of trajectory planning in presence of obstacles is a key challenge in several fields, ranging from automated driving to robotic manipulation, from domestic chores to space applications. A trajectory planning problem subject to obstacle avoidance seeks a final time T, a state trajectory $x\colon [0,T] \to X \subseteq \mathbb{R}^n$, and controls $u\colon [0,T] \to U \subseteq \mathbb{R}^m$, satisfying dynamic constraints, collision avoidance, boundary conditions, and hard bounds, which can be formulated as

$$\begin{aligned} &\dot{x}(t) = f(x(t), u(t)) && \forall\, t \in [0,T], \qquad \text{(P)}\\ &g(x(t)) \le 0,\ x(t) \in X,\ u(t) \in U && \forall\, t \in [0,T],\\ &x(0) = x_0, \quad b(x(T)) = 0, &&\\ &T \in [T_{\min}, T_{\max}], && \end{aligned}$$

where functions $f\colon X \times U \to X$, $b\colon X \to \mathbb{R}^{n_b}$, and $g\colon X \to \mathbb{R}^{n_g}$ model dynamics, terminal conditions, and obstacles, respectively. Objectives and costs are often included as well, encoding preferences among feasible trajectories.

General-purpose nonlinear optimization solvers like Ipopt [1] are popular for tackling (P) but often struggle to find feasible trajectories due to the nonconvex constraints. While combined or custom numerical methods improve robustness, solvers can get stuck at infeasible points and often require careful hand-tuning and initialization [2], [3], [4]. Another strategy to enhance convergence is to exploit the time structure of the problem, e.g., via differential dynamic programming [5], [6], [7], [8], or sequential convexification [9]. Nevertheless, these approaches are sensitive to the initialization point and converge, if at all, to local minimizers or stationary points. In contrast, dynamic programming (DP) is able to identify global optima also in the presence of nonconvex constraints, but suffers from the curse of dimensionality [10], [11]. Despite multiple attempts to lower its computational footprint [12], with enhanced sampling strategies [13] or dynamic grid adaptation [14], [15], among others, DP remains prohibitive in high-dimensional scenarios. Sampling-based strategies established themselves as popular alternatives for dealing with high-dimensional state spaces [16], but appear inadequate to treat system dynamics. Focusing on quadrotors flight, the work in [17] is tailored to interleave a low-dimensional graph search with high-dimensional gradient-based methods, exploiting their respective strengths. Similarly, a reinforcement learning strategy has been adopted in [18] to guide trajectory optimization toward better solutions.

Approach: In this paper, the idea of reconciling two problem representations and solvers is generalized to address a wide range of applications. In particular, we propose an iterative approach to solve problem (P) by combining dynamic programming and derivative-based methods, particularly, direct optimal control and nonlinear programming (NLP) techniques. Taking advantage of both perspectives and strategies, the task is split into two parts: one accounting for obstacle avoidance, faced by DP, and another one incorporating the results into the possibly high-dimensional dynamics constraints, solved by gradient-based methods. To

This research work is funded by dtec.bw – Digitalization and Technology Research Center of the Bundeswehr as part of SeRANIS – Seamless Radio Access Networks for Internet of Space; dtec.bw is funded by the European Union – NextGenerationEU.

The authors are with the University of the Bundeswehr Munich, Department of Aerospace Engineering, Institute of Applied Mathematics and Scientific Computing, 85577 Neubiberg/Munich, Germany (e-mail: {andreas.britzelmeier, alberto.demarchi, rebecca.richter}@unibw.de).

enable interactions between these two layers, we design the low-dimensional problem to generate waypoints to be tracked during trajectory generation, whose output is gathered into an adaptive penalty approximation for the successive DP iteration, as illustrated in Figure 1. Bearing this scheme in mind, a reasonable formulation of the problem solved by DP should account for the intricacies arising from the mapping of higher dimensions, as well as a valid representation of the original collision constraints. Clearly, the actual formulation varies depending on the specific application. Nevertheless, in general we require a mapping $\Omega\colon X \to W$ from the original space X to a low-dimensional space W, where *low* and *high* are relative to the dimensional restrictions imposed by DP. Then, because of this dimensional gap, the inverse transformation Ω^{-1} is often ill-posed. This is the case in robotic applications, where the mappings Ω and Ω^{-1} are associated to forward and inverse kinematics, respectively, and establish a link between joint and task spaces. Another example is the planning of vehicle manoeuvers, where the high- and low-dimensional representations may refer to a detailed model and a simple point-mass approximation thereof.

Outline: Patterning Figure 1, the DP block for path planning is discussed in Section II, the NLP block for trajectory optimization is described in Section III, and their interactions are detailed in Section IV. Numerical tests and results can be found in Section V.

Notation: We denote the set of natural, real, and nonnegative real numbers by $\mathbb{N}$, $\mathbb{R}$, and $\mathbb{R}_+$, respectively. The set of n dimensional vectors over $\mathbb{R}$ is denoted by $\mathbb{R}^n$. We use 0 and 1 to denote a vector or matrix of zeros and ones, respectively, of the appropriate dimensions.

II. Dynamic Programming and Penalty Reformulation

In order to deal with the nonconvexity introduced by, e.g., collision constraints, we intend to use dynamic programming (DP) to find a numerical solution. Moreover, to better cope with the curse of dimensionality, we seek such a feasible trajectory using a low-dimensional representation of (P) induced by some user-defined mapping Ω. Then, collision avoidance constraints are handled via a penalty approach, with subsequent discretization of time, states, and controls. Convergence, feasibility, and optimality guarantees of the DP block follow directly from [11], [12].

A. Penalty Reformulation

Denoting $\tau_0 := 0$ the initial time, we let the controls $v\colon [\tau_0, \tau_f] \to \mathbb{R}^{n_v}$ be measurable functions and the states absolutely continuous functions $\omega\colon [\tau_0, \tau_f] \to \mathbb{R}^{n_\omega}$, and consider

$$\begin{aligned} &\underset{(\omega,v,\tau_f)}{\text{minimize}} && \mathcal{M}(\omega(\tau_f)) && \text{(DP-P)}\\ &\text{subject to} && \omega'(\tau) = \bar{f}(\omega, v), && \text{a.e. } \tau \in [\tau_0, \tau_f]\\ &&& \omega(\tau_0) = \Omega(x_0), &&\\ &&& \omega(\tau) \in \Lambda,\ v(\tau) \in \mathbb{V} && \tau \in [\tau_0, \tau_f] \end{aligned}$$

where the low-dimensional dynamics $\bar{f}\colon \mathbb{R}^{n_\omega} \times \mathbb{R}^{n_v} \to \mathbb{R}^{n_\omega}$ can be defined, e.g., by $\bar{f}(\omega, v) := v$. The sets of admissible controls and states are respectively denoted by $\mathbb{V}$ and $\mathbb{W}$. The objective function $\mathcal{M}\colon \mathbb{R}^{n_\omega} \to \mathbb{R}$ is defined by

$$\mathcal{M}(\omega) := \min_{x\in X} \{\|b(x)\| \mid \Omega(x) = \omega\} \tag{1}$$

for embedding the targeted terminal conditions of (P). This definition is well-posed, in the sense that $\mathcal{M}$ attains a finite value somewhere, if the final condition in (P) is indeed kinematically reachable. The nonconvex collision constraints are encompassed by the set

$$\Lambda := \{\omega \in \mathbb{W} \mid \exists x \in X\colon\ \Omega(x) = \omega,\ g(x) \leq 0\} \tag{2}$$

and must be satisfied for all $\tau \in [\tau_0, \tau_f]$. Now, to resolve these nonconvex constraints, a penalty reformulation is adopted. Hence, let $\mathcal{P}\colon \mathbb{W} \to \mathbb{R}_+$ with

$$\mathcal{P}(\omega) := \min_{x\in X} \{\|\Omega(x) - \omega\| + \|\max\{0, g(x)\}\|\} \tag{3}$$

denote the penalty function for $\omega \in \Lambda$, where the max is applied componentwise. Note that $\mathcal{P}$ is real-valued everywhere if inequality constraints $g(x) \leq 0$ and bounds $x \in X$ in (P) are consistent.

The following discretization scheme is adopted to solve the continuous-time problem (DP-P). Let the number of time intervals $M \in \mathbb{N}$ and the time stepsize bounds $h_{\min}, h_{\max}$ be given, such that $0 \leq h_{\min} \leq h_{\max} \leq T_{\max}$. Then, let $h := (h_0, \ldots, h_{M-1})$ denote the entire sequence of time stepsizes and $\mathbb{H} := \{h \in \mathbb{R}_+^M \mid \sum_{i=0}^{M-1} h_i \in [T_{\min}, T_{\max}]\}$ be the associated admissible set. Hence, with $h \in \mathbb{H}$ we let the time grid be represented by $\mathbb{G}_\tau := \{\tau_i \mid i = 0, \ldots, M\}$ with $\tau_{i+1} := \tau_0 + \sum_{j=0}^{i} h_j$ for $i = 0, \ldots, M-1$. Next, we consider a piecewise constant approximation $v_h\colon [\tau_0, \tau_f] \to \mathbb{R}^{n_v}$ of the control v and a piecewise affine approximation $\omega_h\colon [\tau_0, \tau_f] \to \mathbb{R}^{n_\omega}$ of the state ω defined by an explicit Euler scheme with a variable stepsize, which yields

$$\omega_h(\tau_{i+1}) = \omega_h(\tau_i) + h_i \bar{f}(\omega_h(\tau_i), v_h(\tau_i)). \tag{4}$$

Ensuing from the discretization scheme and by aggregating the dynamics and control constraints, the feasible set of the discrete-time problem is defined by

$$\begin{aligned} \Xi_h := \big\{ (\omega_h, v_h, h) \mid\quad & h \in \mathbb{H}, \\ & \omega_h(\tau_0) = \Omega(x_0), \text{ and } \forall i = 0, \ldots, M-1: \\ & \text{(4) holds and } v_h(\tau_i) \in \mathbb{V} \big\}. \end{aligned} \tag{5}$$

Then, we denote the contribution in the i-th time step by $P_i(\omega_h) := \mathcal{P}(\omega_h(\tau_i))$, for $i = 0, \ldots, M$. Hence, using (3) we can formulate the discrete penalized optimal control problem

$$\underset{(\omega_h, v_h, h)\in\Xi_h}{\text{minimize}} \quad \mathcal{M}(\omega_h(\tau_M)) + \rho \sum_{i=0}^{M} P_i(\omega_h), \tag{6}$$

with penalty parameter $\rho > 0$. Then, a discrete-time feasible and optimal trajectory can be obtained by solving (6) with ρ sufficiently large, yet bounded, relative to the discretization stepsizes, as shown in [12, Chapter 4].

B. Dynamic Programming

Dynamic Programming is based on Bellmann's principle of optimality [10], where the value function $\vartheta(\tau, z)$ reflects the optimal cost of (DP-P) for a trajectory starting at an arbitrary initial time $\tau \in [\tau_0, \tau_f)$ and initial state $z \in \mathbb{W}$. A global solution of the discrete problem (6) can be computed using approximate dynamic programming [11]. Therefore, we build a discretization in the state and control variables, that is, we consider the respective state and control grids $\mathbb{G}_\omega$ and $\mathbb{G}_v$ as lattices over $\mathbb{W}$ and $\mathbb{V}$. Refining time, state, and control discretizations, the corresponding DP solutions converge to those of the continuous problem (DP-P), as attested [12, Chapter 4]. Then, we let $\tau_\ell \in \mathbb{G}_\tau$ denote an initial time, $z_\ell = \omega_h(\tau_\ell)$ the corresponding state, and, for one time step, we let

$$\begin{aligned}\Xi_{\mathbb{G}}(\tau_\ell, z_\ell) := \big\{(\omega_h, v_h, h) \mid \quad & h \in \mathbb{H}, \\ & \omega_h(\tau_\ell) = z_\ell, \quad v_h(\tau_\ell) \in \mathbb{G}_v, \\ & \omega_h(\tau_{\ell+1}) = \omega_h(\tau_\ell) + h_\ell \bar{f}(\omega_h(\tau_\ell), v_h(\tau_\ell))\big\}\end{aligned} \tag{7}$$

define the discrete feasible set, in analogy with (5). Then, the approximate value function is recursively defined by

$$\begin{aligned}\vartheta_h(\tau_M, \omega_M) &:= \mathcal{M}(\omega_M), \\ \vartheta_h(\tau_\ell, z_\ell) &:= \min_{(\omega_h, v_h, h) \in \Xi_{\mathbb{G}}(\tau_\ell, z_\ell)} \Big\{\rho P_\ell(\omega_h) \\ &\qquad + \vartheta_h(\tau_{\ell+1}, \omega_h(\tau_{\ell+1}))\Big\}\end{aligned} \tag{8}$$

for all $\ell = M-1, \ldots, 0$. Since the evolution of the dynamical system does not necessarily coincide with a point on the state grid $\mathbb{G}_\omega$, the propagated value function is approximated using an arbitrary interpolation scheme; see [11], [12]. Given any point ω in the state space, then the value function is approximated by combining its value at the neighboring grid points denoted by $\Pi_j[\omega]$, $j = 1, \ldots, 2^{n_\omega}$, with weighting function $\gamma\colon \mathbb{G}_\tau \times \mathbb{W} \to \mathbb{R}_+$ such that $\sum_{j=1}^{2^{n_\omega}} \gamma(\omega, \Pi_j(\omega)) = 1$ for all $\omega \in \mathbb{W}$. For convenience, we write $\gamma_j(\cdot) := \gamma(\cdot, \Pi_j(\cdot))$. Hence, considering an arbitrary initial time τ_ℓ and state $z_\ell \in \mathbb{W}$, the approximate value function $\tilde{\vartheta}$ is recursively defined by

$$\begin{aligned}\tilde{\vartheta}(\tau_M, \omega_M) &:= \mathcal{M}(\omega_M) \\ \tilde{\vartheta}(\tau_\ell, z_\ell) &:= \min_{(\omega_{h,\ell}, v_{h,\ell}, h) \in \Xi_{\mathbb{G}}(\tau_\ell, z_\ell)} \Big\{\rho P_\ell(\omega_h) \\ &\qquad + \sum_{j=1}^{2^{n_\omega}} \gamma_j(\omega_{h,\ell+1}) \tilde{\vartheta}(\Pi_j[\omega_{h,\ell+1}])\Big\}.\end{aligned} \tag{9}$$

Here, instead of evaluating the penalty defined in (3), one may consider surrogate values obtained by a consistent proxy of $\mathcal{P}$; as discussed in Section IV-B below. The optimal trajectory of the approximated value function is then given by $\{\omega_j\}_{j=0}^M$, which is subsequently proceeded as an input to the high-dimensional trajectory generation.

III. Trajectory Generation

Once a low-dimensional trajectory $\{\tau_j, \omega_j\}_{j=0}^M$ is generated by solving (DP-P), we face the task of generating a suitable high-dimensional trajectory. In view of (P), we seek a final time T and controls u that optimally steer the system's state x through those low-dimensional waypoints. Thus, the trajectory generation problem may be formulated as

$$\begin{aligned} \underset{(x,u,T)}{\text{minimize}} \quad & J(x, u, T) && \text{(NLP-P)} \\ \text{subject to} \quad & \dot{x}(t) = f(x(t), u(t)) && \forall\, t \in [0, T] \\ & x(0) = x_0 \\ & p_j(x(T\tau_j/\tau_M)) = 0 && \forall\, j \in \{0, \ldots, M\} \\ & x(t) \in X,\ u(t) \in U && \forall\, t \in [0, T] \\ & T \in [T_{\min}, T_{\max}], \end{aligned}$$

where dynamics, initial condition, and constraints on state, control, and final time are inherited from (P). The cost functional J can be user-defined; with the aim of generating smoother or more efficient trajectories, J may collect tracking, control, and time costs. State constraints $g(x(t)) \leq 0$ and termination conditions $b(x(T)) = 0$ of (P) are replaced with equality path constraints at $M+1$ time points, encoded by p_j, $j = 0, \ldots, M$, in (NLP-P). Here, functions p_j are adopted to lift and exploit the waypoints $\{\omega_j\}_{j=0}^M$, and their formulation depends on the specific problem and relationships between low and high dimensional representations. A prominent example is $p_j(x) := \Omega(x) - \omega_j$, which requires x to traverse the low-dimensional state ω_j.

Several approaches exist for tackling optimal control problems such as (NLP-P), leading to direct (multiple) shooting and collocation techniques or exploiting Pontryagin's maximum principle; for an overview see [19], [20].

Following the discretize-then-optimize approach, we introduce a time grid $\{t_i\}_{i=0}^N$ partitioning the time interval $[0, T]$ into $N \gg M$ time intervals, and we consider state and control approximations there, denoted by $\{x_i\}_{i=0}^N$ and $\{u_i\}_{i=0}^N$, such that $x_i \approx x(t_i)$ for each $i = 0, \ldots, N$. Notice that the time points $\{T\tau_j/\tau_M\}_{j=0}^M$ are included as a subset of $\{t_i\}_{i=0}^N$, and we indicate by $\mathcal{I}(j)$ the index $i \in \mathbb{N}$ such that $t_i = T\tau_j/\tau_M$. Therefore, state and control bounds can be enforced at each point on the time grid and, in particular, path constraints can be readily included as $p_j(x_{\mathcal{I}(j)}) = 0$, $j = 0, \ldots, M$. Discrete-time dynamics are obtained by using finite differences or arbitrary integration schemes, and appear as equality constraints [19]. Overall, the transcripted problem is a finite-dimensional, constrained nonlinear program (NLP) that can be built and solved using off-the-shelf tools [1], [19]. Note that, if needed or for improved efficiency, one can construct an initial guess for $\{x_i, u_i\}_{i=0}^N$ based on the DP trajectory $\{\omega_j\}_{j=0}^M$ via lifting and interpolation.

A notable case for (NLP-P) is that of linear dynamics f and constraints p_j, and polyhedral sets X and U. Paired with a convex quadratic cost J, the associated fixed final time problem is that of a linear-quadratic regulator. Then, full discretization yields a convex quadratic program (QP), with a plethora of numerical solvers available; see [21], [22] and references therein.

IV. Interfaces: Space Mapping and Penalty

Let us recall the iterative scheme sketched in Figure 1. Therein, the DP block receives a penalty function approximation $\tilde{P}$, based on information gathered from the high-dimensional trajectory generated by NLP, in order to generate a collision-free solution to (P), eventually. Conversely, lifting the low-dimensional DP solution yields suitable path constraints for the high-dimensional trajectory generation by NLP. Building upon the idea of splitting the problem into simpler parts, the overall procedure, delineated in Algorithm 1, requires a careful orchestration of the different inputs and outputs of DP and NLP blocks, invoked respectively at Steps 4 and 7. Thus, the interfaces between these blocks are critical components of the iterative process.

Algorithm 1: Iterative scheme with DP and NLP

```
1  Construct state and control grids $\mathbb{G}_\omega$, $\mathbb{G}_v$
2  Initialize penalty $\tilde{P}_0$ using (3) over state grid points
3  for $k \leftarrow 1, 2, \dots$ do
4      $\{\omega_j^k, v_j^k, h^k\}_{j=0}^M \leftarrow$ (6) with penalty grid $\tilde{P}_k$
5      try $\{x_j^k\}_{j=0}^M \leftarrow \{\omega_j^k\}_{j=0}^M$ by (10), build $\{p_j^k\}_{j=0}^M$
6      catch $\Theta^k \leftarrow \{\omega_j^k\}_{j=0}^M$ and go to Step 10
7      $\{t_i^k, x_i^k, u_i^k\}_{i=0}^N \leftarrow$ (NLP-P) with $\{p_j^k\}_{j=0}^M$
8      if $\forall i \in \{1, \dots, N\}$ $g(x_i^k) \leq 0$ then return
9      $\Theta^k \leftarrow \{\Omega(x_i^k)\}_{i=0}^N$
10     $\widehat{P}(\cdot; \Theta^k), \tilde{P}_{k+1} \leftarrow \Theta^k, \tilde{P}_k$ by (11) and (12)
```

Finally, we shall note that trajectories returned by Algorithm 1 are guaranteed to be, up to the time discretization for (NLP-P), kinematically feasible and collision-free, by Steps 5 and 8, respectively. Conversely, if the original problem (P) is indeed feasible, the discretizations for DP are sufficiently fine, and the penalty parameter ρ large enough, then Algorithm 1 converges to an approximate solution for (P) and terminates, based on [12, Chapter 4].

A. From DP to NLP: Space mapping and path constraints

Given a low-dimensional state $\omega_j \in \mathbb{W}$, we are interested in devising a high-dimensional counterpart $x_j \in X$, possibly not unique, such that $\Omega(x_j) = \omega_j$. Then, path constraints in (NLP-P) can be encoded by functions p_j based on x_j, for $j = 0, \dots, M$. Executed at Step 5 of Algorithm 1, this procedure aims at lifting the waypoints from DP to path constraints for NLP. Recalling the goal of finding a feasible trajectory, we embed the relevant constraints of (P) into the regularized inverse problem

$$\begin{aligned} \underset{(x,\sigma)}{\text{minimize}} \quad & \varrho_1 \|x - x_{j-1}\|^2 + \varrho_2 \langle 1, \sigma \rangle \\ \text{subject to} \quad & \Omega(x) = \omega_j, \quad g(x) \leq \sigma \leq 0, \quad x \in X. \end{aligned} \tag{10}$$

Although the ill-posedness of inverse problems cannot in general be avoided, numerical experience suggests that regularization terms mitigate the issue of multiple solutions, and induce more stable sequences $\{x_j\}_{j=0}^M$. Given some $\varrho_1, \varrho_2 > 0$, high-dimensional candidates $x \in X$ are selected based on their proximity to the previous point x_{j-1} (if $j > 0$, or to x_0 if $j = 0$) and their margin to violating constraints $g(x) \leq 0$, monitored by the auxiliary variable σ. In practice, one can recursively compute $\{x_j\}_{j=1}^M$ starting from the initial state $x_0 \in X$.

Retrieving a high-dimensional waypoint $x_j \in X$, corresponding to $\omega_j \in \mathbb{W}$, one can then define a path constraint function p_j, based on the identity $p_j(x_j) = 0$ and application-specific knowledge. Then, included in (NLP-P), this constraint forces the generated trajectory to traverse, in some sense, the low-dimensional waypoint ω_j. Instead, if no feasible solution to (10) is found, because of kinematic infeasibility or unavoidable collision, the waypoint ω_j is deemed infeasible. In this case, covered by Step 6 in Algorithm 1, a penalty adaptation is required.

B. From NLP to DP: Penalty adaptation and approximation

As the penalty function $\mathcal{P}$ in (3) involves the mapping Ω and constraints g, its behaviour can be highly nonlinear, and hence its value can greatly vary within a single cell of the DP discretization. Therefore, we intend to make the DP execution rely on a penalty function $\tilde{P}$ that is not merely the pointwise evaluation of $\mathcal{P}$ over the grid $\mathbb{G}_\omega$, but also accounts for penalty values observed inside nearby cells. In particular, we exploit the generated high-dimensional trajectories to better capture the penalty landscape and consequently improve the path planned employing DP.

Given a collection of points $\Theta := \{\omega_j\}_{j \in \mathcal{J}} \subseteq \mathbb{W}$ corresponding to time points $\{\tau_j\}_{j \in \mathcal{J}}$, $\mathcal{J} \subset \mathbb{N}$, we can construct a penalty function approximation over the grid, denoted $\widehat{P} \colon \mathbb{G}_\omega \to \mathbb{R}_+$, as the weighted average

$$\widehat{P}(\omega_i; \Theta) := \frac{\sum_{j \in \mathcal{J}_i} \alpha(\omega_j, \omega_i) \mathcal{P}(\omega_j)}{\sum_{j \in \mathcal{J}_i} \alpha(\omega_j, \omega_i)}. \tag{11}$$

Here, for the given grid point $\omega_i \in \mathbb{G}_\omega$, the set $\mathcal{J}_i \subset \mathbb{N}$ collects all indices $j \in \mathcal{J}$ such that $\tau_j \in (\tau_{i-1}, \tau_{i+1})$, and the weighting function $\alpha \colon \mathbb{W} \times \mathbb{G}_\omega \to \mathbb{R}_+$ assigns importance to samples based on their position relative to the point of interest ω_i. If $\mathcal{J}_i$ is empty, then no information is available to estimate the penalty; we set its value to zero. Notice that the definition of set $\mathcal{J}_i$ imposes a time restriction on the samples, in order to maintain the separability of penalty terms with respect to time, i.e., the Markov property.

Now, at the k-th iteration of Algorithm 1, DP adopts a penalty $\tilde{P}_k$, which we dynamically adapt based on the latest available approximation obtained via $\widehat{P}$. Let $\tilde{P}_k \colon \mathbb{G}_\omega \to \mathbb{R}_+$, $k \in \mathbb{N}$, denote the approximated penalty over the state grid $\mathbb{G}_\omega$. Then, given a collection of points $\Theta^k \subseteq \mathbb{W}$ generated at the k-th iteration, we define recursively

$$\tilde{P}_{k+1} := \tilde{P}_k + \widehat{P}(\cdot; \Theta^k), \tag{12}$$

where the initialization $\tilde{P}_0$ can be obtained, e.g., by evaluating $\mathcal{P}$ on the grid. Once an approximation $\tilde{P}_k$ of $\mathcal{P}$ is available on the grid, interpolation procedures can generate values for arbitrary points $\omega \in \mathbb{W}$. Acting as a surrogate of $\mathcal{P}$, this can be adopted in (9) to avoid its explicit evaluation.

Finally, we shall comment on the constraint and penalty evaluation at Steps 8 and 10 of Algorithm 1. As some given high-dimensional state $x_i \in X$ and time $t_i \in [0,T]$ are readily mapped into some $\omega_i := \Omega(x_i) \in \mathbb{W}$ and $\tau_i := \tau_M t_i/T$, for the feasibility of $\omega_i \in \Lambda$ it remains only to check whether x_i satisfies $g(x_i) \le 0$ or not. Suppose this holds, then ω_i is feasible and necessarily $\mathcal{P}(\omega_i) = 0$. Conversely, ω_i could be feasible even if $g(x_i) \le 0$ is violated. However, this requires the explicit minimization of (3) to evaluate $\mathcal{P}(\omega_i)$.

V. Numerical Tests

We implement the algorithms and a testing environment in MATLAB (R2022a), on a system with Intel i7-11700K (8 cores, 3.60 GHz) and 32GB RAM. The proposed algorithmic framework is evaluated by means of a robotic manipulator application and compared to CHOMP [2]. To this end, let us define the task model in terms of (P). We have the high-dimensional state $x := (q, \dot{q})$ aggregating the joint positions and velocities, as well as the joint accelerations as control u. Consequently, the robot arm dynamics are modeled as $f((q,\dot{q}),u) := (\dot{q}, u)$ and we choose a minimal effort objective $J(x,u,T) := \int_0^T \|u(t)\|^2 \mathrm{d}t$ for the trajectory optimization. The boundary condition is set to $b(x) = \max\{\|\Omega(x) - \omega_{\text{ref}}\| - \omega_{\text{tol}}, 0\}$, with target position ω_{ref}. Additionally, the state constraint $g(x) \le 0$ is characterized by the signed distance

$$g(x) := \varepsilon - \min_{c \in C \cup R(q) \;\; c \neq r \in R(q)} \operatorname{dist}(c, r) \tag{13}$$

between the set of robot collision objects at a certain configuration $R(q)$ and the set of environment obstacles C, with a constant safety bound $\varepsilon \ge 0$. Herein, $\operatorname{dist}(c,r) \in \mathbb{R}$ denotes the signed distance between two convex polyhedrons c and r [23] and is evaluated by means of an extension for the negative part of the separating axis theorem [24] in conjunction with the Gilbert-Johnson-Keerthi (GJK) algorithm [25]. Note that this formulation not only accounts for obstacle-robot interactions, but also for self-collisions. Moreover, our approach does not rely on the specific collision geometry adopted here, which strikes a balance between accurate obstacle description and fast distance computation. Alternative representations include mesh [3] and ball approximations [2], among others. With respect to (DP-P), the lower-dimensional state vector $\omega := (x, y, z)$ is defined by the Cartesian coordinates of the robot's tool center point (TCP), with controls $v := \dot{\omega}$ the respective linear Cartesian velocities and time step h. Consequently, the mapping Ω is characterised by the direct kinematics of the robot, returning the TCP Cartesian coordinates given the robot's joint configuration. Considering the penalty approximation, we select $\alpha(\cdot, \omega_i)$ in (11) as the triangular function with stencil $[\tau_{i-1}, \tau_{i+1}]$.

A. Solver setting

The penalty (3) and inverse problem (10) are evaluated using `fmincon` with its default `interior-point` algorithm. Considering the trajectory generation phase, an optimal final time is computed employing `fminbnd`, concurrently with an optimal trajectory for a fixed time problem via full discretization [26]. We employ finite differences and trapezoidal rule for discretizing dynamics and running cost. Then, linear dynamics and quadratic cost lead to a convex QP, which is solved using `quadprog`.

For the comparison with CHOMP [2] we use the implementation in the MoveIt Motion Planning Framework shipped with ROS (Robot Operating System) Noetic Ninjemys. We use this planner with default settings and with enabled failure recovery, which allows multiple retries on failure with dynamically adjusted parameters.

B. Numerical experiments

We considered a pick-and-place scenario for a 6 DoF Kinova MICO$^{\text{TM}}$ robotic arm with static rectangular obstacles, as in Figure 2. Measuring time in seconds, joint angles in degrees, and TCP coordinates in meters, we set the TCP target $\omega_{\text{ref}} = (0.59, 0.0, 0.45)$ with $\omega_{\text{tol}} = 0.06$. Bounds on states and controls are set to $q_{\min} = (-180, 50, 35, -360, -360)$ and $q_{\max} = (-180, 310, 325, 360, 360)$, as well as $(\dot{q}^i_{\min}, \dot{q}^i_{\max}, u^i_{\min}, u^i_{\max})_{i=1}^6 = (-20, 20, -180, 180)$.

Discretization and boundary parameters for DP are set to $h \in [0,1]$, $[T_{\min}, T_{\max}] := [0, 300]$, $M_\omega = (19, 19, 19)$, $M_v = (3,3,3,3)$, $M = 20$, $\omega_{\min} = (-0.7, -0.7, 0)$, $\omega_{\max} = (0.7, 0.7, 1)$ and $(v^j_{\min}, v^j_{\max})_{j=1}^3 = (-0.11, 0.11)$, with scaling parameter $\rho = 10^4$ in (6) and safety distance $\varepsilon = 0.01$ in (13). In (10) we set $\varrho_1 = 1$, $\varrho_2 = 10$, and we choose $N = 500$ for the time discretization of (NLP-P). Finally, we set a limit on the number of iterations, that is, Algorithm 1 terminates as soon as $k > 20$.

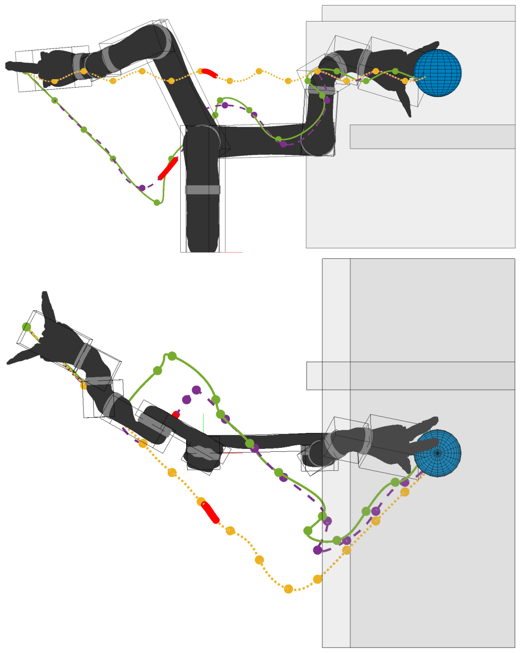
Fig. 2. Resulting trajectories of the first (orange dotted), second (purple dashed) and third (green solid) iteration, with colliding positions (red markers), obstacles (gray), target region (blue sphere), and optimal initial and final configurations of the robotic arm with collision geometry (boxes).

Figure 2 illustrates the resulting trajectories produced by the proposed algorithmic framework using the initial configuration $q_0 = (29, 209, 98, 12, -19, 0)$. It can be observed, that after each trajectory proposal in a collision, the penalty adaptation leads to an amendment of the waypoints, consequently resulting in a different trajectory. The algorithm terminates in 3 iterations and, as intended, avoids the infeasible regions previously detected. Let us mention that the resulting state trajectories are smooth and the TCP traverses the waypoints generated by DP, placed at non-uniform timepoints.

In order to validate our approach for multiple initial configurations, we sample the space $\mathbb{W} \subset \mathbb{R}^3$ with an uniform grid containing 10 points in each dimension. For each feasible point, according to (10), we execute Algorithm 1 with the resulting joint configuration and zero velocity as initial condition. Upon success, we invoke CHOMP to plan a path between the same initial and final joint-space coordinates.

Out of 526 runs, Algorithm 1 found a solution in 512 cases (97%), requiring on average between 2 and 3 iterations. The 14 failures do not show any apparent pattern in task-space, and we consider them to be caused by the rough penalty approximation resulting from coarse discretization. Considering the CHOMP planner, it was able to compute a valid trajectory in 108 of these instances (21%) using the default parameter settings, whereas 302 cases (59%) were solved by enabling the failure recovery strategy. For each instance, CHOMP's runtime was in the order of seconds, whereas our proof-of-concept implementation of Algorithm 1 required a few minutes. However, there exist various techniques, such as adaptive discretization and parallel computing, to improve the performance of our tool, which is currently designed for functionality and flexibility. Overall, this illustrates the capabilities of our combined approach for planning collision-free trajectories.

VI. Conclusions

We proposed an algorithmic framework that fruitfully integrates dynamic programming (DP) and nonlinear programming (NLP) methods for planning trajectories in cluttered environments. Exploiting a lower-dimensional system's representation, the nonconvexity due to obstacles is handled by DP, while the system dynamics are accounted for by direct optimal control techhniques, overcoming the curse of dimensionality for DP. Numerical tests on robotic manipulation tasks showed that the proposed combination of methods perform successfully.

Future research may focus on adaptively refining the state space discretization, possibly based on the penalty updates, and considering a common objective to the low- and high-dimensional subproblems, overcoming the difficult coupling due to potentially different time scales.

References

[1] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical Programming*, vol. 106, no. 1, pp. 25–57, 3 2006.

[2] M. Zucker, N. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant Hamiltonian optimization for motion planning," *The International Journal of Robotics Research*, vol. 32, no. 9–10, pp. 1164–1193, 2013.

[3] J. Schulman, Y. Duan, J. Ho, A. Lee, I. Awwal, H. Bradlow, J. Pan, S. Patil, K. Goldberg, and P. Abbeel, "Motion planning with sequential convex optimization and convex collision checking," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1251–1270, 2014.

[4] T. A. Howell, B. E. Jackson, and Z. Manchester, "ALTRO: A fast solver for constrained trajectory optimization," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 7674–7679.

[5] D. Mayne, "A second-order gradient method for determining optimal trajectories of non-linear discrete-time systems," *International Journal of Control*, vol. 3, no. 1, pp. 85–95, 1966.

[6] Y. Tassa, N. Mansard, and E. Todorov, "Control-limited differential dynamic programming," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1168–1175.

[7] F. Farshidian, M. Neunert, A. W. Winkler, G. Rey, and J. Buchli, "An efficient optimal planning and control framework for quadrupedal locomotion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 93–100.

[8] S. Singh, J.-J. Slotine, and V. Sindhwani, "Optimizing trajectories with closed-loop dynamic SQP," in *2022 International Conference on Robotics and Automation (ICRA)*, 2022, pp. 5249–5254.

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

[10] R. Bellman, *Dynamic Programming*. Princeton, New Jersey: Princeton University Press, 1957.

[11] D. P. Bertsekas, *Dynamic Programming & Optimal Control*. Belmont, Massachusetts: Athena Scientific, 2005, 3rd edition.

[12] A. Britzelmeier, "Decomposition and Hamilton-Jacobi-Bellman methods for nonconvex generalized Nash equilibrium problems," Ph.D. dissertation, University of the Bundeswehr Munich, 2021.

[13] C. Cervellera, M. Gaggero, D. Macciò, and R. Marcialis, "Quasi-random sampling for approximate dynamic programming," in *The 2013 International Joint Conference on Neural Networks (IJCNN)*, 2013, pp. 1–8.

[14] R. Munos and A. Moore, "Variable resolution discretization in optimal control," *Machine Learning*, vol. 49, no. 2, pp. 291–323, 2002.

[15] R. Richter, A. Britzelmeier, and M. Gerdts, "An adaptive mesh dynamic programming algorithm for robotic manipulator trajectory planning," in *2023 European Control Conference (ECC)*, 2023.

[16] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE access*, vol. 2, pp. 56–77, 2014.

[17] R. Natarajan, H. Choset, and M. Likhachev, "Interleaving graph search and trajectory optimization for aggressive quadrotor flight," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5357–5364, 2021.

[18] G. Grandesso, E. Alboni, G. P. R. Papini, P. M. Wensing, and A. D. Prete, "CACTO: Continuous actor-critic with trajectory optimization—towards global optimality," *IEEE Robotics and Automation Letters*, vol. 8, no. 6, pp. 3318–3325, 2023.

[19] M. Gerdts, *Optimal Control of ODEs and DAEs*. De Gruyter, 2011.

[20] F. Biral, E. Bertolazzi, and P. Bosetti, "Notes on numerical methods for solving optimal control problems," *IEEJ Journal of Industry Applications*, vol. 5, no. 2, pp. 154–166, 2016.

[21] A. De Marchi, "On a primal-dual Newton proximal method for convex quadratic programs," *Computational Optimization and Applications*, vol. 81, no. 2, pp. 369–395, 2022.

[22] B. Hermans, A. Themelis, and P. Patrinos, "QPALM: a proximal augmented lagrangian method for nonconvex quadratic programs," *Mathematical Programming Computation*, vol. 14, no. 3, pp. 497–541, 2022.

[23] S. Cameron and R. Culley, "Determining the minimum translational distance between two convex polyhedra," in *1986 IEEE International Conference on Robotics and Automation Proceedings*, vol. 3, 1986, pp. 591–596.

[24] P. J. Schneider and D. H. Eberly, *Geometric Tools for Computer Graphics*. Morgan Kaufmann, 2003.

[25] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, "A fast procedure for computing the distance between complex objects in three-dimensional space," *IEEE Journal on Robotics and Automation*, vol. 4, no. 2, pp. 193–203, 1988.

[26] A. De Marchi and M. Gerdts, "Free finite horizon LQR: a bilevel perspective and its application to model predictive control," *Automatica*, vol. 100, pp. 299–311, 2019.