Collision Avoidance using Iterative Dynamic and Nonlinear Programming with Adaptive Grid Refinements

Rebecca Richter   Alberto De Marchi   Matthias Gerdts

Abstract— **Nonlinear optimal control problems for trajectory planning with obstacle avoidance present several challenges. While general-purpose optimizers and dynamic programming methods struggle when adopted separately, their combination enabled by a penalty approach is capable of handling highly nonlinear systems while overcoming the curse of dimensionality. Nevertheless, using dynamic programming with a fixed state space discretization limits the set of reachable solutions, hindering convergence or requiring enormous memory resources for uniformly spaced grids. In this work we solve this issue by incorporating an adaptive refinement of the state space grid, splitting cells where needed to better capture the problem structure while requiring less discretization points overall. Numerical results on a space manipulator demonstrate the improved robustness and efficiency of the combined method with respect to the single components.**

I. INTRODUCTION

Autonomous systems have been in the past decades and still are a significant subject of research in various fields, ranging from industrial applications to autonomous driving, from medical systems to space manipulators. In this paper we consider the problem of designing collision-free trajectories for dynamical systems, possibly subject to kinematic constraints, and moving in cluttered environments. A long-standing problem in robotics, such design is an important task in many applications [1]. Expressed in abstract terms, as a feasibility problem over the infinite-dimensional space of trajectories, the task addressed in this paper is to find states $x\colon [0,T] \to X$, controls $u\colon [0,T] \to U$, and final time $T > 0$ that satisfy

$$
\begin{aligned}
\dot{x}(t) &= f(x(t), u(t)) & \forall t \in\ &[0,T] \\
g(x(t)) &\le 0 & \forall t \in\ &[0,T] \\
u(t) &\in U,\ x(t) \in X & \forall t \in\ &[0,T] \qquad (1)\\
b(x(T)) &= 0,\ x(0) = x_0 & & \\
T &\in [T_{\min}, T_{\max}] & &
\end{aligned}
$$

with state and control bounds $X \subset \mathbb{R}^{n_x}, U \subset \mathbb{R}^{n_u}$, dynamics $f\colon X \times U \to \mathbb{R}^{n_x}$, nonlinear path constraints $g\colon X \to \mathbb{R}^{n_g}$, final conditions $b\colon X \to \mathbb{R}^{n_b}$, and bounds on the final time $T_{\min}, T_{\max} > 0$. An optimal control problem formulation is

often preferred, including a cost function to filter feasible results according to application-specific preferences. However, being obstacles the source of problematic nonconvexities in the trajectory-optimization problem, in this work we focus on reliable methods for seeking *feasible* trajectories.

The popular method of dynamic programming (DP) is able to handle the constraints (1) and provide a solution under mild assumptions, but suffers from the curse of dimensionality [2], [3]. Despite multiple attempts of lowering its computational requirements, including enhanced sampling strategies [4], [5], adaptive state space refinements [6], [7], and massive parallelisation, DP is not yet applicable in practice to high dimensional scenarios. In contrast, sampling-based methods [8] such as Rapidly-exploring Random Trees (RRT) [9] show good performance on larger state space dimensions. Therefore, these methods are widespread, especially in the field of robotics, but kinematics or dynamics are hardly incorporated [1]. Approaches based on solving large-scale nonlinear programs (or systems of equations) are also limited [10]. General-purpose optimizers like Ipopt [11] and WORHP [12] can be highly sensitive to initial guesses and provide, if any, a local solution. Since they additionally tend to struggle with many nonconvex constraints, as those for collision avoidance, several reformulations for specific obstacle structures were presented [13], [14]. Numerical approaches tailored to motion planning tasks are able to improve robustness but remain sensitive to the quality of user-defined initializations [15], [16]. To overcome these

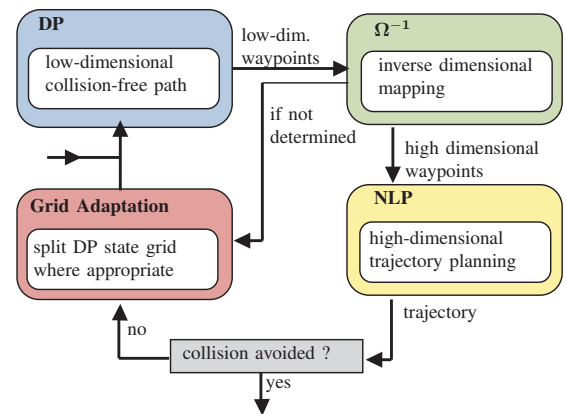Fig. 1. Schematic overview of the proposed iterative scheme for motion planning in constrained environments: low-dimensional collision-free waypoints from dynamic programming (DP) are mapped back to the original dimension and tracked by a trajectory planned with nonlinear programming (NLP). Possible collisions are integrated into the DP via a penalty term and the DP state space grid is adapted, until convergence to a valid solution.

difficulties, the approaches presented in [17], [18] combine low dimensional graph-search and dynamic programming, respectively, with gradient-based optimization in a higher dimensional space, exploiting the strengths of the respective methods. In a similar spirit, "to take the best of the two worlds", [19] brings together network and convex optimization in a mixed-integer programming formulation for a special class of motion planning problems.

Motivation: In this work we focus on the framework proposed in [18], here referred to as Iterative Dynamic and Nonlinear Programming (IDNP), which alternates between DP and nonlinear programming (NLP) to successively improve the design relative to the original problem (1), as follows. Exploiting a problem-specific lower-dimensional representation of (1), the trajectory planning task is decomposed into two coupled components: (i) the generation of a sequence of low-dimensional waypoints and (ii) the dynamically-feasible motion planning through these waypoints, via DP and NLP respectively. Seeking a collision-free trajectory, waypoints leading to infeasible motions are penalized, thus strongly coupling the two components. Waypoints enter into the motion planning task as path constraints on the state [10], whereas the penalty reformulation in DP follows from the theory in [20]. In [18] the state space discretization for DP is precomputed and fixed, requiring a reasonable initial sampling to cover the obstacles. To better capture the environment with the penalty term, this is adapted through the iterations but, because of the fixed state space discretizations, cannot capture narrow passages in between obstacles on coarse discretizations, thus inhibiting the solution process. Moreover, this is especially an issue as high-resolution sampling results in huge computational effort for DP due to the curse of dimensionality.

Contribution: In this paper, with the goal of limiting the memory and runtime requirements on fine grids and enhancing convergence on coarse ones, we incorporate an adaptive state space refinement strategy into IDNP, building upon ideas presented in [6]. However, instead of formulating a refinement criterion as a trigger for cell refinements, we use the collisions detected in the trajectories planned by the NLP block. We demonstrate on a space manipulator numeric example improved robustness and efficiency of the combined method compared to both the original one presented in [18] and full discretization. Figure 1 illustrates the overall structure of the proposed iterative scheme. Notice how each component plays an important role: without DP, NLP struggles with nonconvex constraints; without NLP, DP is not practical for high dimensional systems; without adaptive refinements, DP and NLP combined (as in IDNP) require fine discretizations or may not converge.

Outline: We will detail the components of our approach, sketching those of IDNP and highlighting the proposed changes. Following [18], the low- and high-dimensional problem representations are discussed in Sections II and III, respectively. Then, the state space discretization and adaptive splitting are considered in Section IV. Finally, numerical simulations are reported in Section V.

II. LOW DIMENSIONAL LEVEL

The DP method is capable of handling nonconvex constraints and providing a global solution, but suffers from the curse of dimensionality [2], [3]. In this section we briefly recall the formulation in [18, Section II] to apply DP on a low dimensional representation of (1), with state and control spaces $W \subset \mathbb{R}^{n_w}$ and $V \subset \mathbb{R}^{n_v}$, respectively, associated to a suitable user-defined mapping $\Omega \colon X \to W$. The precise definition of Ω as well as of *high-* and *low-* dimensional representations are problem-dependent. As a prominent example, a robotic manipulator with 12 degrees of freedom may be described with $n_x = 12$, $n_w = 3$, and Ω its forward kinematics. The IDNP algorithm builds upon a Low-Dimensional Problem (LDP) of the form

$$\underset{w,v,\tau_f}{\text{minimize}} \ \mathcal{M}(w(\tau_f)) + \rho \int_0^{\tau_f} \mathcal{P}(w(\tau))\mathrm{d}\tau \qquad \text{(LDP)}$$

$$\text{subject to} \ \ \dot{w}(\tau) = \bar{f}(w(\tau), v(\tau)) \qquad \forall \tau \in [0, \tau_f]$$

$$v(\tau) \in V \qquad \forall \tau \in [0, \tau_f]$$

$$w(0) = \Omega(x_0), \ \tau_f \in [T_{\min}, T_{\max}]$$

with state $w \colon [0, \tau_f] \to W$, control $v \colon [0, \tau_f] \to V$, dynamics $\bar{f} \colon W \times V \to \mathbb{R}^{n_w}$, and a free final time τ_f. It includes the inital condition of (1) using the dimensional mapping Ω, and the terminal conditions are encoded into the Mayer-type term $\mathcal{M} \colon W \to \mathbb{R}_+$, $\mathcal{M}(w) := \min_{x \in X} \{\|b(x)\| \mid \Omega(x) = w\}$. Additionally, the penalty function $\mathcal{P} \colon W \to \mathbb{R}_+$,

$$\mathcal{P}(w) := \min_{x \in X} \{\|\Omega(x) - w\| + \|\max\{0, g(x)\}\|\}, \quad (2)$$

accounts for the collision avoidance constraints, weighted by a penalty factor $\rho > 0$.

A. Dynamic Programming

We discretize (LDP) with respect to time to obtain a finite-dimensional transcription of this optimization problem, which will then be tackled using DP. Let us consider a fixed number of time steps $M \in \mathbb{N}$, and a set of time step sequences $\mathbb{H} := \left\{ h \in \mathbb{R}_+^M \mid \sum_{i=0}^{M-1} h_i \in [T_{\min}, T_{\max}] \right\}$, each defining a certain discrete time grid $\mathbb{G}_\tau$ given by time points $\tau_i = \sum_{j=0}^i h_j$, $i = 0, \ldots, M$. The resulting time-discretized problem can be written as

$$\underset{w_h,v_h,h}{\text{minimize}} \ \mathcal{M}(w_h(\tau_M)) + \rho \sum_{i=0}^{M-1} h_i \mathcal{P}(w_h(\tau_i)) \qquad (3)$$

$$\text{subject to} \ \ w_h(\tau_{i+1}) = \bar{f}_{h_i}(w_h(\tau_i), v_h(\tau_i)) \ \forall \tau_i, \tau_{i+1} \in \mathbb{G}_\tau$$

$$v_h(\tau_i) \in V \qquad \forall \tau_i \in \mathbb{G}_\tau$$

$$w_h(\tau_0) = \Omega(x_0), \ h \in \mathbb{H}$$

with $w_h \colon \mathbb{G}_\tau \to W$ and $v_h \colon \mathbb{G}_\tau \to V$ representing the state and control approximations. The explicit Euler approximation of the dynamics is given by $\bar{f}_h(w, v) := w + h\bar{f}(w, v)$. As described in details in [18, Section 2.B] the method of

approximate DP is applied, based on the recursion

$$\vartheta(\tau_M, w_M) := \mathcal{M}(w_M) \tag{4}$$

$$\vartheta(\tau_\ell, w_\ell) := \min_{v_{h,\ell} \in \mathbb{G}_v, h \in \mathbb{H}} \left\{ \rho h_\ell \mathcal{P}(w_\ell) \right.$$
$$\left. + \sum_{j=1}^{2^{n_w}} \gamma_j \left(\bar{f}_{h_\ell}(w_\ell, v_{h,\ell}) \right) \vartheta \left(\tau_{\ell+1}, \Pi_j^{\ell+1}[\bar{f}_{h_\ell}(w_\ell, v_{h,\ell})] \right) \right\}$$

defined for initial states $z_\ell \in \mathbb{G}_w^\ell$ and times $\tau_\ell \in \mathbb{G}_\tau$, and control grid $\mathbb{G}_v$. To evaluate the value function ϑ on the state grid $\mathbb{G}_w^{\ell+1}$, we employ an arbitrary interpolation, represented by $\Pi_j^{\ell+1} \colon W \to \mathbb{G}_w^{\ell+1}$ and $\gamma_j^{\ell+1} \colon W \to \mathbb{R}$, returning respectively the j-th vertex of the containing cell and its associated weight. In practice, the property $h \in \mathbb{H}$, or $\tau_f \in [T_{\min}, T_{\max}]$, can be guaranteed by setting bounds on each timestep as $T_{\min}/M \le h_i \le T_{\max}/M$.

We stress that (3) stems from a time-discretization of (LDP), whereas [18, Eq. 6] can be interpreted as arising from the penalization of constraint violations on the time grid $\mathbb{G}_\tau$. Though motivated in different ways, both variants serve their purpose for the low-dimensional planning, as reported in Section V.

In contrast to the original IDNP of [18], here we define a state grid for every time point, since they might have different structures due to the adaptive refinements presented in Section IV below. The convergence towards the continuous problem attested in [18, Section 2.B], on infinitely fine time, state, and control discretizations is left untouched. Notice that in general it can be more efficient to have time-dependent, yet adaptive state grids, because they can be much coarser than in the uniformly sampled case. Once the optimal trajectory $\{\tau_j, w_j\}_{j=0}^M$ resulting from (4) has been computed, it is mapped back onto the original space of (1). If feasible, such collision-free sequence enters the NLP block, without explicit formulation of the collision constraints; cf. Figure 1.

III. TRAJECTORY GENERATION

While the adaptive state space refinement process requires some changes to the DP block and interfaces of IDNP, the trajectory planning block remains as in [18, Section III]. The High-Dimensional Problem (HDP)

$$\underset{x,u,T}{\text{minimize}} \quad J(x, u, T) \tag{HDP}$$

$$\text{subject to} \quad \dot{x}(t) = f(x(t), u(t)) \quad \forall t \in [0, T]$$
$$x(t) \in X, \; u(t) \in U \quad \forall t \in [0, T]$$
$$p_j(x(T\tau_j/\tau_M)) = 0 \quad \forall j \in \{0, \dots, M\}$$
$$x(0) = x_0$$
$$T \in [T_{\min}, T_{\max}],$$

accounts for the dynamics, initial conditions, and bounds of the original problem (1). Additionally, state and control sets X and U can encode general constraints, and could also be coupled, as long as simple enough to be handled by NLP solvers (e.g., convex sets described by inequalities). While the objective function $J \colon X \times U \times [T_{\min}, T_{\max}] \to \mathbb{R}$ can reflect some user preferences, functions $p_j \colon X \to \mathbb{R}$,

$j = 0, \dots, M$, represent the waypoint constraints. Their explicit formulation depends on the structure of the dimensional mapping Ω; the prominent example $p_j(x) := \Omega(x) - w_j$ is used in our numerical simulations.

Note that (HDP) is also a free final time problem. However, since the timestamped waypoints $\{\tau_j, w_j\}_{j=0}^M$ from (3) satisfy the time constraints by construction of $\mathbb{H}$, it would be possible to consider it fixed here, with $T = \tau_f$. Despite the issue of dealing with a free final time optimal control problem, we prefer the formulation (HDP) to equip the motion planning task with an additional degree of freedom. Nonetheless, feasibility of (HDP) is still not guaranteed as the waypoints could be outside the reachable set.

In order to apply off-the-shelf NLP solvers to (HDP), the trajectory generation task is transcribed into a finite-dimensional problem by discretization with respect to time. Therefore, we introduce $N \gg M$ discrete time points $\{t_i\}_{i=0}^N$, respective state $\{x_i\}_{i=0}^N$ and control $\{u_i\}_{i=0}^{N-1}$ approximations, and a discrete dynamics defined with respect to an arbitrary approximation scheme. The waypoint timestamps $\{T\tau_j/\tau_M\}_{j=0}^M$ are embedded into the discrete time points $\{t_i\}_{i=0}^N$ to directly encode the waypoint constraints in the time-discretized problem. After solving a time-discretized counterpart of (HDP), the solution trajectory $\{t_i, x_i\}_{i=0}^N$ satisfies the relevant dynamics, state and control bounds, and path constraints. Therefore, if the generated trajectory happens to avoid obstacles, then it is a valid solution to the original problem (1), up to the NLP discretization and tolerance. Finally, notice that (HDP) can be addressed by any techniques able to tackle optimal control problems with state constraints, that is, not necessarily by full discretization; see [10] for other approaches.

A. Space Mapping

In order to connect the low- and high-dimensional problems presented above, the mapping $\Omega \colon X \to W$ is used to transfer points in the original space X to their low dimensional representation in W. While Ω is well-defined, e.g., via forward kinematics models, the inverse operation is often ill-posed, requiring to determine the high dimensional representation of a low dimensional trajectory. Problem

$$\underset{x,\sigma}{\text{minimize}} \quad \varrho_1 \|x - x_{j-1}\|^2 + \varrho_2 \langle 1, \sigma \rangle \tag{5}$$

$$\text{subject to} \quad \Omega(x) = w_j, \quad g(x) \le \sigma \le 0, \quad x \in X$$

is solved in IDNP to determine a state that belongs to $\Omega^{-1}(w_j)$, if possible, is close to a given previous state $x_{j-1} \in X$, and has a large margin before violating the collision avoidance constraints, with weighting factors $\varrho_1, \varrho_2 \ge 0$. As pointed out in [18, Section IV.A], solving this problem recursively starting from $x_0 \in X$ reduces the issue of multiple solutions and leads to more stable high dimensional sequences $\{\tau_j, x_j\}_{j=0}^M$.

Although obstacle avoidance constraints are relaxed and penalized, with the goal of promoting collision-free configurations, problem (5) can be infeasible if the waypoint w_j is not reachable with the forward mapping Ω. For instance,

many robotic manipulators have limited reachable regions because of their mechanical construction. If this is the case, that is, if a waypoint w_j selected by DP is deemed infeasible at this stage, then the corresponding point in the state grid $\mathbb{G}_w^j$ is penalized in (LDP) via $\mathcal{P}$ to discourage its selection at the next DP call. A similar procedure is applied if a trajectory point x_i from (HDP) is not collision-free. In [18, Alg. 1, Step 10], such penalization would be distributed onto neighboring state grid points, without affecting the state grid itself. In contrast, here an infeasible inverse mapping problem triggers a direct call to the grid adaptive refinement, as illustrated in Figure 1 and discussed in the following section.

IV. ADAPTIVE GRID REFINEMENT

Deviating from the IDNP algorithm of [18], we replace the penalty adaptation therein with adaptive refinements of the state space grid. In this way, instead of lumping penalties together, constraints are better captured by the penalty landscape thanks to additional evaluation points, located where needed. In fact, the state space can initially be coarsely sampled and fine obstacle structures will then be recognized, if of interest, steering convergence to feasible trajectories. For reference, we apply the adaptive mesh approach from [6] instead of the penalty approximation in [18, Section IV.B].

As pointed out in Section II, we consider state space grids $\{\mathbb{G}_w^j\}_{j=0}^M$ dividing the admissible space into cells $\{\mathbb{C}_w^j\}_{j=0}^M$, for each discrete time step of the DP algorithm. Allowing arbitrary grid structures, we consider some functions $C_w^j: W \rightarrow \mathbb{C}_w^j$ and $\mathcal{S}_w^j: \mathbb{C}_w^j \rightarrow P(W)$, $j = 0, \dots, M$. The former map a certain point to its corresponding cell, while the latter are splitting operations, with potential set P, returning the grid extended with the points resulting from the splitting of a specific cell. For a detailed discussion we refer to [6, Section III], where the explicit definitions for the case of n-dimensional cube grids are also presented.

As depicted in Figure 1, the grid adaptation process is called in two cases, and with two types of input. On the one hand, the inverse mapping hands over sets $\{\tau_k, w_k\}_{k=0}^K$ of $K \in (0, M]$ waypoints deemed infeasible. On the other hand, the NLP optimized trajectory has $K \in (0, N]$ points $\{t_k, x_k\}_{k=0}^K$ that are not collision-free. In the latter case, the points are transferred to the DP space applying the forward dimensional mapping Ω, at time $\tau_k = \tau_M t_k / T$ and space $w_k = \Omega(x_k)$ respectively, such that a low dimensional sequence $\{\tau_k, w_k\}_{k=0}^K$ is generated.

Let us focus now on the refinement evoked by an infeasible timestamped waypoint $\{\tau_k, w_k\}$. First, we find the unique time index $j \in \mathbb{N}$ from $\mathbb{G}_\tau$ such that $\tau_{j-1} < \tau_k \leq \tau_j$. Then, in order to maintain the Markov property for (LDP), as recommended in [18, Section IV.B], both grids $j-1$ and j will be refined, as in IDNP for the penalty adaptation. Therefore, the set of cells selected for refinement is identified by collecting all cells containing at least an infeasible waypoint, namely

$$\mathbb{K}_w^j := \{c \mid \exists \{\tau, w\} \in \{\tau_k, w_k\}_{k=0}^K,$$
$$c = C_w^j(w), \tau \in [\tau_{j-1}, \tau_{j+1}]\} . \quad (6)$$

Finally, the state grids for the next DP evaluation within IDNP are updated according to $\mathbb{G}_w^j \leftarrow \cup_{c \in \mathbb{K}_w^j} \mathcal{S}_w^j(c)$. While Figure 2 illustrates the adaptive splitting process, Algorithm 1 outlines the overall iterative procedure; compare Figure 1 and [18, Alg. 1].
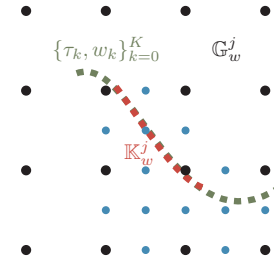

Fig. 2. Splitting process of 2D grid $\mathbb{G}_w^j$ (black) with additional points after update according to [6] (blue), due to the collection of waypoints in collision $\{\tau_k, w_k\}_{k=0}^K$ (green) and those belonging to this grid $\mathbb{K}_w^j$ (red).

Algorithm 1: IDNP scheme with grid refinements

1 Initialize state and control grids $\{\mathbb{G}_w^j\}_{j=0}^M$, $\mathbb{G}_v$
2 **while** *true* **do**
3    $\{w_j, v_j, h\}_{j=0}^M \leftarrow$ (LDP) with grids $\mathbb{G}_w^j$
4   **try** $\{x_j\}_{j=0}^M \leftarrow \{w_j\}_{j=0}^M$ by (5)
5   build $\{p_j\}_{j=0}^M$
6   **catch** $\{w_k\}_{k=0}^K \subset \{w_j\}_{j=0}^M$ with (5) failed at w_k
7   **go to** Step 11
8    $\{t_i, x_i, u_i\}_{i=0}^N \leftarrow$ (HDP) with $\{p_j\}_{j=0}^M$
9   **if** $\forall i \in \{1, \dots, N\}$ $g(x_i) \leq 0$ **then return**
10    $\{w_k\}_{k=0}^K \subset \{\Omega(x_i)\}_{i=0}^N$ with $g(x_k) > 0$
11    $\mathbb{G}_w^j \leftarrow \cup_{c \in \mathbb{K}_w^j} \mathcal{S}_w^j(c)$ by (6)

V. NUMERICAL TESTS

In order to numerically demonstrate the capabilities of our approach, we apply it to the trajectory planning task of a robotic manipulator within a space application and compare the results to the original IDNP [18] as well as to a full discretization approach using Ipopt [11]. The task is challenging due to the many degrees of freedom and the highly nonlinear dynamics. Notice that a comparison with pure DP, including splitting or not, is impractical due to the curse of dimensionality.

A. Implementation

We implemented Algorithm 1 and the original IDNP in C++. Ipopt version 3.14.12 [11] is used, in combination with HSL linear solvers [21] and the Ifopt interface version 2.1.3 [22], to solve the auxiliary problems (HDP), (2), and (5). Further, we adopt the automatic differentiation tool autodiff v1.0.3 [23] to estimate the derivatives required when solving (HDP). To limit the cases where we observe nonzero penalties from local optima at feasible regions, we initially solve a problem without cost function holding solely the collision avoidance and mapping constraint and afterwards use its result as an initial guess on solving (2).
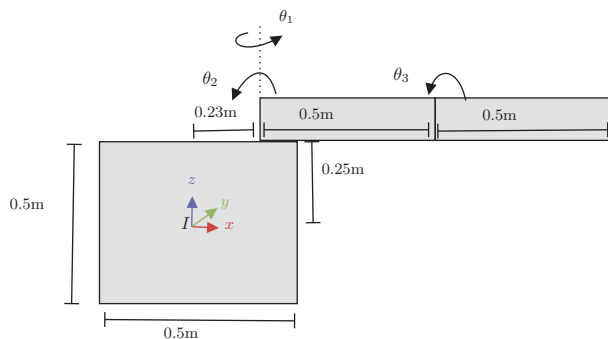
Fig. 3. Illustration of the satellite body with robotic arm. The coordinate axes of the satellites inertial frame I are highlighted.

$\tilde{q}_{\min}$	$\tilde{q}_{\max}$	$\tilde{q}'_{\min}$	$\tilde{q}'_{\max}$	$\tilde{u}_{\min}$	$\tilde{u}_{\max}$	w_{min}	w_{max}	v_{min}	v_{max}
[m]	[m]	[m/s]	[m/s]	[N]	[N]	[m]	[m]	[m/s]	[m/s]
-1.0	15.0	-0.5	0.5	-1.2	1.2	-1.0	15.0	-0.04	0.04
-5.0	5.0	-0.5	0.5	-1.2	1.2	-5.0	5.0	-0.04	0.04
-1.0	15.0	-0.5	0.5	-1.2	1.2	-1.0	15.0	-0.04	0.04
[rad]	[rad]	[rad/s]	[rad/s]	[Nm]	[Nm]				
$-\infty$	∞	$-\pi/6$	$\pi/6$	-1.5	1.5				
-1.47	1.47	$-\pi/6$	$\pi/6$	-1.5	1.5				
$-\infty$	∞	$-\pi/6$	$\pi/6$	-1.5	1.5				
$-\pi/2$	$\pi/2$	$-\pi/10$	$\pi/10$	-1.0	1.0				
$-\pi$	0.0	$-\pi/10$	$\pi/10$	-1.0	1.0				
$-\pi/2$	$\pi/2$	$-\pi/10$	$\pi/10$	-1.0	1.0				

All the results presented were executed on an AMD Ryzen Threadripper 3990X system with 256 GB RAM.

B. Problem

We consider the model of a free-floating satellite with a two link robotic arm attached, see Figure 3. Its 9-dimensional configuration $\tilde{q} = \{q, \theta\}$ comprises the satellite body's pose $q \in \mathbb{R}^6$, with respect to the inertial reference frame I, and the joint angles of the robotic arm $\theta \in \mathbb{R}^3$. The satellite-arm system is acted upon with controls $\tilde{u} \in \mathbb{R}^9$, each one associated to a degree of freedom. Instead of considering a fully free-floating system, we simplify the robotic arm dynamics, approximating them with double-integrators, as $\ddot{\theta}_i = u_{i+6}$ for $i = 1, 2, 3$. Each body's pose with respect to I is determined using the robotic arm's forward kinematics. Subsequently, (translational and rotational) velocities and accelerations are obtained using the first and second time derivatives. For the remaining degrees of freedom, Lagrange mechanics yield the equations of motion [24] in the form

$$M\left(\tilde{q}, \dot{\tilde{q}}\right) \ddot{q} = \tilde{f}\left(\tilde{q}, \dot{\tilde{q}}, \tilde{u}_q\right),$$

with mass matrix $M \in \mathbb{R}^{6\times6}$ and satellite control signals $\tilde{u}_q := \{u_1, \ldots, u_6\}$.

In view of (1), we define the state vector of our second-order satellite model as $x := \{\tilde{q}, \dot{\tilde{q}}\} \in \mathbb{R}^{18}$, with control $u := \tilde{u} \in \mathbb{R}^9$ and include the equations of motion into the multi dimensional dynamics function, defined component wise as

$$
\begin{aligned}
f_i(x(t), u(t)) &= x(t)_{i+9} & i &= 1, \ldots, 9 \\
f_i(x(t), u(t))) &= M^{-1}\tilde{f}(x(t), u(t)) & i &= 10, \ldots, 15 \\
f_i(x(t), u(t)) &= u(t)_{i-9} & i &= 16, 17, 18
\end{aligned}
$$

To complete the geometric information provided in Figure 3 we fix the satellite body's depth to 0.4m, its mass to 50kg and its inertia diagonal matrix values to $\{1.71, 1.71, 2.08\}$kgm^2. Accordingly, the cylindric robotic links have a radius of 0.05m, diagonal inertia entries $\{0.006, 0.107, 0.107\}$kgm^2 and a mass of 5kg each. We limit the state and control according to Table I and the time bounds with $T_{\min} = 0$s and $T_{\max} = 500$s.

In addition, we consider four rectangular static obstacles in our environment, illustrated in Figure 4. We formulate the collision avoidance constraint as in [18, Eq. 13], based on the signed distance between polygons and using a safety bound $\varepsilon = 0.01$m. We formulate the final conditions as

$$
\begin{aligned}
b_0(x) &= \alpha \max\{\|\Omega(x) - w_{\text{ref}}\| - r, 0\} \\
b_i(x) &= x_{i+9} & i &= 1, \ldots, 9
\end{aligned}
$$

with b_0 returning the positive distance to the goal TCP point $w_{\text{ref}} = \{12.0, 0.0, 1.0\}$m, scaled by parameter $\alpha = 10^3$, and zero on a ball with radius $r = 1$m around it; see Figure 4. The other components of b demand zero final velocity.

C. Setup

To solve the above problem using Algorithm 1 we choose function Ω to represent the forward kinematics of the satellite, mapping a 18-dimensional configuration to the corresponding 3-dimensional tool center point (TCP) position. We initially discretize the DP state and control space within the bounds presented in Table I into 10 points along each dimension as well as 10 points on the discretization of the time step size h_i. We set the dynamics function to $\bar{f}(w, v) = v$, the penalty scaling factor $\rho = 40$ and consider $M = 20$ DP time steps. For the inverse space mapping (5), we set parameters $\varrho_1 = \varrho_2 = 1$. We follow the arising waypoints using $p_j(x) = \Omega(x) - w_j$ as well as a minimum effort objective $J(x, u, T) = \int_0^{T_{\max}} \|u(t)\|_2^2 \mathrm{d}t$ in (HDP), discretizing with an equidistant grid with $N = 200$ time intervals, piecewise constant control parametrization, and finite difference approximation of the dynamics with trapezoidal rule. To realize the adaptive grid refinement, we use the cube-based grid structure proposed in [6] along with its respective cell identification and splitting operators. In the experiments, we took the lower part of the initial state space grid e.g. $\{10, 10, 4\}$ points and tried to solve (5) on each of them. On success, we used the resulting configuration $\tilde{q}_0$ to form an initial condition $x_0 := \{\tilde{q}_0, 0\}$ and solved the respective problem using (at most) 15 iterations of Algorithm 1 and IDNP. After adopting the discrete-time version of (LDP) without the stepsize in the objective approximation, as in [18, Eq. 6], we repeated these experiments with (3) for both Algorithm 1 and IDNP.

Finally, to compare against a full discretization approach, we take 6 samples, distributed among the space of initial positions, where Algorithm 1 was able to terminate (on the

first set of experiments). Discretizing (1) as for (HDP), the resulting zero-objective NLP is given to Ipopt, initialized either from a zero guess or with a collision-free path obtained using the RRTConnect implementation from OMPL [25].
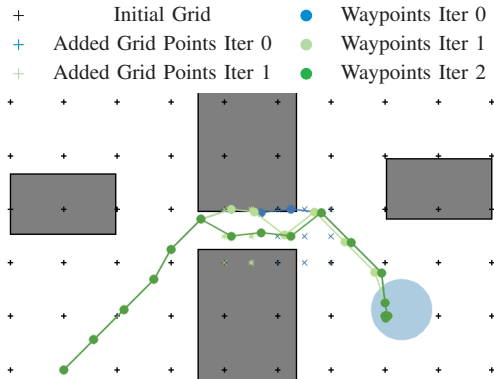

Fig. 4. Illustration of state space grids and waypoints at DP iteration 6 resulting from iterations 1, 2, and 3 of Algorithm 1 using initial condition $\tilde{q}_0 = \{-0.2, -0.14, -0.68, -0.49, 0.34, 0.43, 0.13, -0.02, 0.1\}$. The goal region (light blue) as well as the obstacles (gray) are highlighted.

D. Results and Discussion

The waypoints resulting from the first three iterations of Algorithm 1 and a certain initial condition are illustrated in Figure 4. In the first iteration, since there are no collision-free grid points in the narrow passage, the points cross the obstacles at the region of smallest penalty, namely where there is minimum overlap. As shown in Figure 4 the addition of state space points quickly results in a sampling of the gap between the obstacles, leading to feasible waypoints at iteration 2 and finally to a collision-free trajectory. In contrast, the original IDNP was not able to provide a feasible solution. Evaluating 254 initial conditions in total, IDNP provided solutions in 132, and our algorithm in 163 cases. We did not observe any case where IDNP was able to find a solution but our algorithm was not. Similar (albeit not identical) results were obtained in the experiments with the discrete-time version (3) of (LDP). Finally, Ipopt was able to solve only 4 of the 6 problems when starting from a zero initial guess, while it managed to solve all of them when initialized with collision-free paths. We therefore numerically demonstrated the improved efficiency and robustness of our algorithm, due to its enhanced capability to provide feasible solutions, compared to the original IDNP and a trivially initialized full discretization approach.

VI. CONCLUSIONS

In this work, we proposed a method combining dynamic programming, direct optimal control techniques, and adaptive refinement of state space discretizations to solve a trajectory planning problem with obstacle avoidance. Compared to adopting these techniques separately, enhanced robustness and efficiency of our method stem from the grid refinements, leading to improved convergence and computational scalability. These achievements are numerically demonstrated on a trajectory planning task for a satellite manipulator system.

REFERENCES

[1] M. Hoy, A. S. Matveev, and A. V. Savkin, "Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey," *Robotica*, vol. 33, no. 3, pp. 463–497, 2015.

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

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

[4] 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.

[5] A. Weber, M. Rungger, and G. Reissig, "Optimized state space grids for abstractions," *IEEE Transactions on Automatic Control*, vol. 62, no. 11, pp. 5816–5821, 2017.

[6] R. Richter, A. Britzelmeier, and M. Gerdts, "An adaptive mesh dynamic programming algorithm for robotic manipulator trajectory planning," in *2023 European Control Conference (ECC)*, Bucharest, Romania, 2023, pp. 1–8.

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

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

[9] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Department of Computer Science, Iowa State University, Tech. Rep. Research Report 9811, 1998.

[10] M. Gerdts, *Optimal Control of ODEs and DAEs*. De Gruyter Oldenbourg, 2023, 2nd Edition.

[11] 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, 2006.

[12] C. Büskens and D. Wassel, "The ESA NLP solver WORHP," in *Modeling and Optimization in Space Engineering*, G. Fasano and J. D. Pintér, Eds. Springer New York, 2013, vol. 73, pp. 85–110.

[13] X. Zhang, A. Liniger, and F. Borrelli, "Optimization-based collision avoidance," *IEEE Transactions on Control Systems Technology*, vol. 29, no. 3, pp. 972–983, 2021.

[14] J. Guthrie, "A differentiable signed distance representation for continuous collision avoidance in optimization-based motion planning," in *2022 IEEE 61st Conference on Decision and Control (CDC)*, 2022, pp. 7214–7221.

[15] 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.

[16] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization," in *Proceedings of Robotics: Science and Systems*, Berlin, Germany, 2013.

[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] A. Britzelmeier, A. De Marchi, and R. Richter, "Dynamic and nonlinear programming for trajectory planning," *IEEE Control Systems Letters*, vol. 7, pp. 2569–2574, 2023.

[19] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, p. eadf7843, 2023.

[20] A. Britzelmeier and A. Dreves, "A decomposition algorithm for Nash equilibria in intersection management," *Optimization*, vol. 70, no. 11, pp. 2441–2478, 2021.

[21] HSL, "A collection of fortran codes for large scale scientific computation," 2013. [Online]. Available: http://www.hsl.rl.ac.uk

[22] A. W. Winkler, "Ifopt - a modern, light-weight, Eigen-based C++ interface to nonlinear programming solvers IPOPT and SNOPT." 2018. [Online]. Available: https://github.com/ethz-adrl/ifopt

[23] A. M. M. Leal, "autodiff, a modern, fast and expressive C++ library for automatic differentiation," 2018. [Online]. Available: https://autodiff.github.io

[24] E. Bertolazzi, F. Biral, and M. Da Lio, "Real-time motion planning for multibody systems: Real life application examples," *Multibody System Dynamics*, vol. 17, no. 2–3, pp. 119–139, 2007.

[25] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, https://ompl.kavrakilab.org.