iDb-A*: Iterative Search and Optimization for Optimal Kinodynamic Motion Planning

Joaquim Ortiz-Haro[1,2], Wolfgang Hönig[2], Valentin N. Hartmann[2,3], Marc Toussaint[2]

Abstract—Motion planning for robotic systems with complex dynamics is a challenging problem. While recent sampling-based algorithms achieve asymptotic optimality by propagating random control inputs, their empirical convergence rate is often poor, especially in high-dimensional systems such as multirotors. An alternative approach is to first plan with a simplified geometric model and then use trajectory optimization to follow the reference path while accounting for the true dynamics. However, this approach may fail to produce a valid trajectory if the initial guess is not close to a dynamically feasible trajectory. In this paper, we present Iterative Discontinuity Bounded A* (iDb-A*), a novel kinodynamic motion planner that combines search and optimization iteratively. The search step utilizes a finite set of short trajectories (motion primitives) that are interconnected while allowing for a bounded discontinuity between them. The optimization step locally repairs the discontinuities with trajectory optimization. By progressively reducing the allowed discontinuity and incorporating more motion primitives, our algorithm achieves asymptotic optimality with excellent any-time performance. We provide a benchmark of 43 problems across eight different dynamical systems, including different versions of unicycles and multirotors. Compared to state-of-the-art methods, iDb-A* consistently solves more problem instances and finds lower-cost solutions more rapidly.

Index Terms—Kinodynamic Motion Planning, Trajectory Optimization

I. INTRODUCTION

KINODYNAMIC motion planning for robots remains a challenging task, particularly when the objective is to compute time-optimal plans. Fig. 1 showcases four interesting problems: obstacle-free recovery motions from unstable configurations with low-power quadcopters (Fig. 1d), swing-up motions with acrobatic pole-copters among obstacles (Fig. 1c), maneuvering an Ackermann steering car with a trailer through a tight corridor (Fig. 1b), and a unicycle with asymmetric angular speed bounds and a positive minimum velocity (Fig. 1a). Together, these examples illuminate the key challenges in kinodynamic motion planning: *i)* the diversity of the intended robotic systems, *ii)* the nonlinearity of the dynamics, and *iii)* nonconvex configuration spaces with narrow passages among obstacles.

Supplementary video: https://youtu.be/GoznOEWbUb8. Website: https://quimortiz.github.io/idbastar/. Code is available at https://github.com/quimortiz/dynoplan.

[1]Machines in Motion Laboratory, New York University, USA, [2]TU Berlin, Germany, [3]Computational Robotics Lab, ETH Zurich, CH.
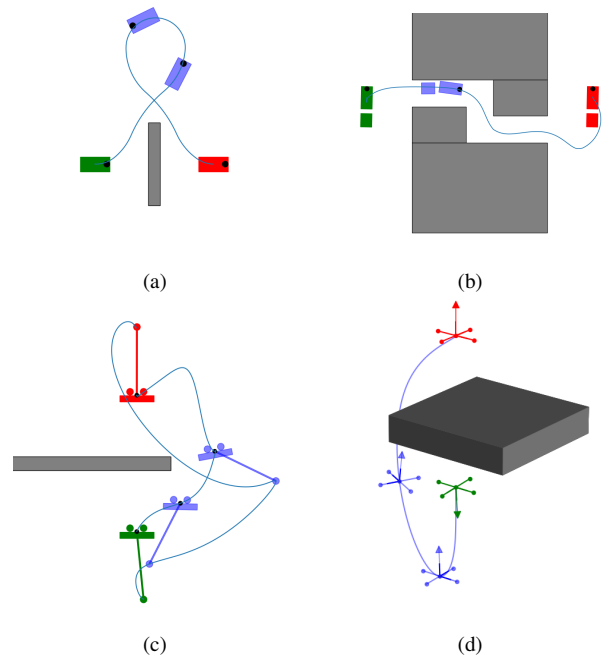
Fig. 1. Examples of kinodynamic motion planning problems. Start and goal configurations are shown in green and red, respectively, while gray boxes represent obstacles. We display some trajectories found by our algorithm, iDb-A*, in blue. (a) A unicycle with asymmetric angular speed bounds and a positive minimum velocity. (b) A car pulling a trailer. (c) Acrobatics with a planar multirotor with an underactuated pendulum. (d) A recovery flight with a quadrotor with a very limited thrust-to-weight ratio.

Current planning approaches are sampling-based, search-based, optimization-based, or hybrid. Each of these methods has its strengths and weaknesses. Sampling-based planners [1], [2], [3] can find initial solutions quickly and have strong guarantees for asymptotic convergence to an optimal solution in theory. In practice, the initial solutions are far from optimal; the convergence rate is low, and the solutions typically require post-processing.

Search-based approaches [4], [5] can remedy some of those shortcomings by connecting precomputed trajectories, so-called *motion primitives*, using A* or related graph search algorithms. Yet, the theoretical guarantees of A* only hold to the selected discretization of the state space and the precomputed motions. Moreover, scaling this approach to higher dimensions or general systems requires careful, frequently hand-crafted design of the motion primitives, which requires domain-specific knowledge and is impractical for many dynamical systems.

Optimization-based planners [6], [7], [8], [9], [10] scale polynomially rather than exponentially with the number of state dimensions, which makes them better suited for high-dimensional planning problems. However, these planners are, in general, only locally optimal and thus require a good initial guess both for the trajectory and the time horizon.

Thus, optimization approaches are typically combined with a sampling-based planner that generates an initial path with a simplified version of the dynamics, e.g., a geometric planner that avoids obstacles. This strategy is not guaranteed to produce valid motion plans and requires in-depth knowledge of the dynamical system to choose an informative, yet simple enough, dynamics model. For instance, a simple linear position model can be used to plan quadcopter motions around the hovering state but fails to plan trajectories that recover from upside-down configurations.

Our main contribution is <u>I</u>terative <u>D</u>iscontinuity <u>B</u>ounded <u>A</u>* (iDb-A*), a novel kinodynamic motion planner that combines a search algorithm, <u>D</u>iscontinuity-<u>B</u>ounded <u>A</u>* (Db-A*), and trajectory optimization in an <u>I</u>terative fashion.

iDb-A* combines key ideas and strengths of the previously introduced methods. We rely on a graph search with short trajectories that are connected with bounded discontinuity because it provides a theoretically grounded exploration-exploitation trade-off. We avoid predefined discretization and instead use a set of randomized motions similar to sampling-based planning.

Introducing discontinuity when connecting primitives (weighted euclidean distance between the end of one primitive and the start of the next one) makes the search tractable: we can reuse the primitives and have a finite number of states to expand. While the output trajectory of the search algorithm is not feasible, it can be used as an initial guess for trajectory optimization that locally repairs the discontinuous trajectory into a valid one. We execute search and optimization iteratively, where the value of the discontinuity bound decreases with each iteration, and the number of primitives increases. For large discontinuity bounds, the search is fast, but the optimizer might fail to find a valid solution. For small bounds, the search requires a longer runtime, but the optimizer has a better initial guess. Thus, the iterative combination results in an efficient anytime planner with probabilistic optimality guarantees.

Our algorithm is implemented in C++ and is publicly available. Our second contribution is an open-source benchmark that compares the three major kinodynamic motion planning techniques on the same problem instances. While we focus in our evaluation on time optimality, our approach supports other cost functions that are additive and non-negative (e.g., energy or squared acceleration).

Statement of Extension: This article is based on our previous conference paper [11], but it provides algorithmic improvements, a faster implementation, and a more extensive evaluation, which includes several problems that require obstacle avoidance and aggressive movements with flying robots.

New Algorithmic Contributions:

- A new strategy to optimize trajectories with free terminal time in the optimization step of iDb-A*.

- The generalization of iDb-A* from translation-invariant systems to systems without invariance (e.g., the acrobot), or with additional linear velocity invariance (e.g., multi-rotors).

Additionally, we provide a refined theoretical analysis and an ablation study of different components, including various optimization strategies, heuristics, and motion primitives.

II. RELATED WORK

Search-based approaches rely on existing methods for discrete path planning, such as A* and its variants. The common approach is to generate short trajectories (*motion primitives*) using a state lattice—a pre-specified discrete set of states [12], [13]. Each primitive starts and ends at a grid cell, and swept cells can be precomputed for efficient collision checking. Once motion primitives are computed, existing algorithms such as A* or an anytime variant (e.g., Anytime Repairing A* [14]) can be employed without modification, providing very strong theoretical guarantees on both optimality and completeness with respect to the chosen primitives. The major challenge is selecting and computing effective motion primitives, especially for high-dimensional systems [15], [16].

Sampling-based approaches build a tree $\mathcal{T}$ rooted at the start state $\mathbf{x}_s$. During tree expansion, i) a random state $\mathbf{x}_{rand}$ in the state space is sampled, ii) an existing state $\mathbf{x}_{expand} \in \mathcal{T}$ is selected, and iii) a new state $\mathbf{x}_{new}$ is added with a motion that starts at $\mathbf{x}_{expand}$ and moves towards $\mathbf{x}_{rand}$. The motions are typically generated by propagating random control inputs, and the classic version of this approach, *kinodynamic RRT* [17], is probabilistically complete [18]. Asymptotic optimality can be achieved when planning in state-cost space (*AO-RRT*) [19], [20], [21] or by computing a sparse tree (*SST**) [1]. These methods rely on a distance function and often require fast nearest neighbor data structures, such as k-d trees, for efficiency.

Sampling-based approaches are designed to explore the state space as rapidly as possible and typically do not explicitly use a heuristic function, unlike search-based methods. Instead, the exploration/exploitation trade-off is controlled by using goal-biasing. In these approaches, the goal constraint is typically reformulated using a goal region rather than a goal state.

Optimization-based approaches locally refine an initial trajectory using the gradients of the cost function, dynamics, and collision constraints, unlike the previous gradient-free methods. The trajectory optimization problem can be formulated as a finite-dimensional nonlinear program (NLP) using either direct collocation or shooting methods [22], and solved with general-purpose nonlinear solvers (e.g., [23], [24]).

For instance, *TrajOpt* [6] and *GuSTO* [7], [25] rely on direct transcription and sequential convex programming (SCP), while *KOMO* [8] combines direct transcription with the Augmented Lagrangian algorithm.

Trajectories can also be computed with optimal control solvers based on Differential Dynamic Programming [26], [27] or the iterative Linear Quadratic Regulator (iLQR) [9].

All optimization-based approaches require an initial guess as a starting trajectory, but this guess does not necessarily

need to be feasible. For nonlinear dynamics and constraints, optimization approaches are incomplete and might fail or converge to a local optimum. In fact, they often converge to infeasible solutions unless the initial guess is close to a feasible solution. When successful, the solution quality is significantly higher (e.g., in terms of smoothness) compared to sampling-based or search-based approaches. Moreover, optimization-based approaches do not suffer directly from the curse of dimensionality, although higher dimensions might result in more local optima.

Hybrid approaches combine search, sampling, and optimization. For instance, one can combine search and optimization [28], search and sampling [29], [2], [30], or combine sampling and optimization [31], [32]. For some dynamical systems, using insights from control theory for motion planning can also be beneficial [33], [34], [35], but it requires domain knowledge. Motion planning can also benefit from using machine learning for computational efficiency [36], [37].

Our algorithm, iDb-A*, combines ideas and tools from the three main approaches to kinodynamic motion planning. The most closely related works are methods that reuse edges within a sampling-based planning framework [30] and search-based methods with duplicate detection [38]. Compared to these works, we include trajectory optimization and reuse locally optimal precomputed motion primitives interconnected with bounded discontinuity for better success and faster convergence.

Apart from the aforementioned approaches, a popular approach to kinodynamic motion planning problems is to first plan with **simplified dynamic models** and to use trajectory optimization or a local controller to follow the reference path while accounting for the true dynamics. The simplest model is a geometric model (holonomic, first-order integrator), which enables geometric motion planning with, e.g., RRT, RRT*, PRM, or PRM* [39], [40], [41]. Second-order systems can be approximated by a double integrator linear model [42].

The trajectories computed with simplified dynamics can then be used as initial guesses for trajectory optimization (that is, optimization-based approaches as previously discussed), model predictive control, or system-specific controllers for quadcopters [43], unicycle-like robots [44], or car-like robots [45]. System-specific motion planners can exploit certain properties of the dynamics, such as differential flatness in quadcopters, which allows faster motion planning as shown for quadcopters [46] and for some specific fixed-wing UAVs [47]. However, differential flatness cannot account for actuation constraints directly—leading to either conservative or infeasible trajectories, especially for small UAVs with a low thrust-to-weight ratio.

Notably, planning with simplified dynamics does not guarantee the generation of valid motion plans and demands an in-depth understanding of the dynamical system. In our algorithm, iDb-A*, connecting motion primitives with bounded discontinuity during the search step can be interpreted as an alternative form of simplified dynamics. However, iDb-A* is complete and asymptotically optimal because it combines search and optimization in an iterative fashion, increasing the number of motion primitives and reducing the allowed

discontinuity in each iteration.

III. PROBLEM DESCRIPTION

We consider a robot with a continuous state $\mathbf{x} \in \mathcal{X}$ (e.g., $\mathcal{X} \subseteq \mathbb{R}^{d_x}$) that is actuated by actions $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^{d_u}$. The dynamics of the robot are deterministic, described by a differential equation,

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

To employ gradient-based optimization, we assume that we can compute the Jacobian of $\mathbf{f}$ with respect to $\mathbf{x}$ and $\mathbf{u}$, typically available in systems studied in kinodynamic motion planning, such as mobile robots or rigid-body articulated systems. We use $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ to denote the collision-free space, i.e., the subset of states that are not in collision with the obstacles in the environment.

We discretize the dynamics (1) with a zero-order hold, i.e., we assume the applied action is constant during a time step of duration Δt. The discretized dynamics can then be written as,

$$\mathbf{x}_{k+1} \approx \text{step}(\mathbf{x}_k, \mathbf{u}_k) \equiv \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\Delta t, \tag{2}$$

using a small Δt to ensure the accuracy of the Euler approximation. We use $K \in \mathbb{N}$ to denote the number of time steps (which is not fixed but subject to optimization), $\mathbf{X} = \langle \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K \rangle$ to denote the sequence of states sampled at times $0, \Delta t, \ldots, K\Delta t$ and $\mathbf{U} = \langle \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{K-1} \rangle$ to denote the sequence of actions applied to the system for the time frames $[0, \Delta t), [\Delta t, 2\Delta t), \ldots, [(K-1)\Delta t, K\Delta t)$. The objective of navigating the robot from its start state $\mathbf{x}_s$ to a goal state $\mathbf{x}_g$ can then be framed as the optimization problem,

$$\min_{\mathbf{U}, \mathbf{X}, K} J(\mathbf{U}, \mathbf{X}), \tag{3a}$$

$$\text{s.t. } \mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k) \quad \forall k \in \{0, \ldots, K-1\}, \tag{3b}$$

$$\mathbf{u}_k \in \mathcal{U} \quad \forall k \in \{0, \ldots, K-1\}, \tag{3c}$$

$$\mathbf{x}_k \in \mathcal{X}_{\text{free}} \subseteq \mathcal{X} \quad \forall k \in \{0, \ldots, K\}, \tag{3d}$$

$$\mathbf{x}_0 = \mathbf{x}_s; \ \mathbf{x}_K = \mathbf{x}_g, \tag{3e}$$

with the cost term $J(\mathbf{U}, \mathbf{X}) = \sum_{k=0}^{K-1} j(\mathbf{u}_k, \mathbf{x}_k)\Delta t$, where $j(\mathbf{u}_k, \mathbf{x}_k) \geq 0$. In this paper, we will focus on time-optimal trajectories, i.e., $j(\mathbf{u}_k, \mathbf{x}_k) = 1$; $J(\mathbf{U}, \mathbf{X}, K) = K\Delta t$, but our framework can be applied to optimize any additive cost function, for example, minimum control effort $j(\mathbf{u}_k, \mathbf{x}_k) = \|\mathbf{u}_k\|^2$.

We assume the dynamics function $\text{step}(\mathbf{x}, \mathbf{u})$, control space $\mathcal{U}$, state space $\mathcal{X}$, and cost function $j(\mathbf{x}, \mathbf{u})$, are known before solving the problem, which allows us to precompute motion primitives.

Example 1. *Consider a unicycle robot with state $\mathbf{x} = [x, y, \theta] \in \mathbb{R}^2 \times SO(2)$, i.e., x, y are the position and θ is the orientation. The actions are $\mathbf{u} = [v, \omega] \in \mathcal{U} \subset \mathbb{R}^2$, i.e., the speed and angular velocity can be controlled directly. The dynamics are $\dot{\mathbf{x}} = [v\cos(\theta), v\sin(\theta), \omega]$. The choice of $\mathcal{U}$ can make this low-dimensional problem challenging to solve. For example, Fig. 1a shows a plane-like case (positive minimum*
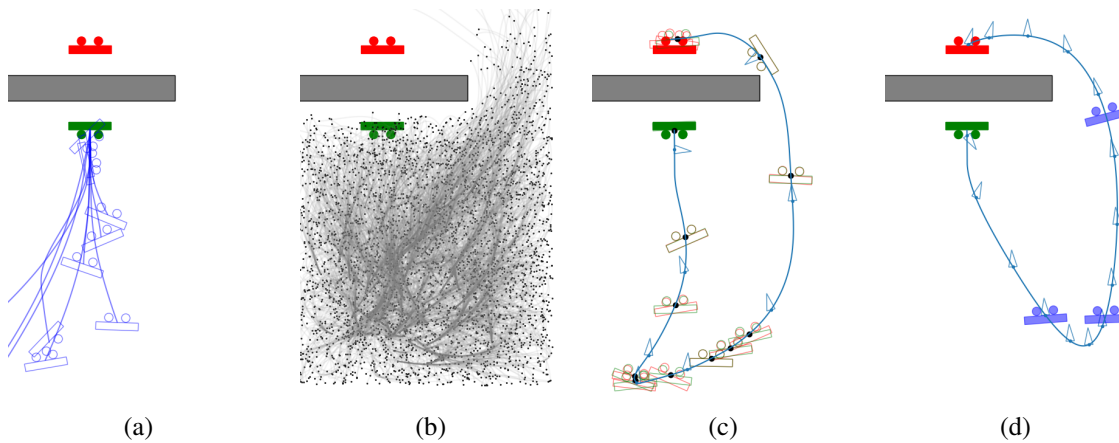
Fig. 2. Visual representation of iDb-A* in the problem *Planar rotor – Recovery obstacles* (i.e., a recovery maneuver with a planar multirotor). Start and goal configurations are shown in solid green (note that the multirotor starts upside down) and red, respectively. (a) In the search step of iDb-A*, called Db-A*, we expand states (in this case, the initial state) using motion primitives that are applicable with bounded discontinuity. (b) Intermediate search tree during the execution of Db-A*. For visualization, the 6D configuration space is projected into a 2D space. (c) Solution found by Db-A*. The transparent, border-only green and red shapes show the start and end of each motion primitive, respectively. They do not match exactly, which highlights the allowed discontinuities when stitching motion primitives (note that the discontinuities in the velocities are not shown in this 2D representation). (d) The output of Db-A* is used to warm-start nonlinear trajectory optimization. The resulting trajectory, shown in blue, fulfills the dynamics constraints and is locally optimal.

speed, i.e., $0.25 \leq v \leq 0.5$ m/s) with a malfunctioning rudder (asymmetric angular speed, i.e., $-0.25 \leq \omega \leq 0.5$ rad/s).

Example 2. *Consider a quadrotor $\mathbf{x} = [\mathbf{p}, \mathbf{v}, \mathbf{q}, \mathbf{w}]$ in $\mathbb{R}^9 \times SO(3)$ where $\mathbf{p}$ represents the position, $\mathbf{v}$ is the velocity, $\mathbf{q}$ represents the orientation using a quaternion, and $\mathbf{w}$ is the angular velocity in the body frame. The control input is the force at each rotor, $\mathbf{u} \in \mathbb{R}^4$. The dynamics are,*

$$\dot{\mathbf{v}} = m^{-1}\mathbf{R}(\mathbf{q})\mathbf{B}_1\mathbf{u} + \mathbf{g}, \tag{4a}$$

$$\dot{\mathbf{w}} = \mathbf{I}^{-1}(\mathbf{B}_0\mathbf{u} - \mathbf{w} \times \mathbf{I}\mathbf{w}), \tag{4b}$$

$$\dot{\mathbf{p}} = \mathbf{v}, \quad \dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \mathbf{w}, \tag{4c}$$

where m is the mass, $\mathbf{I}$ represents the inertia matrix, $\mathbf{g}$ is the gravity vector, $\mathbf{R}(\mathbf{q})$ is the rotation matrix corresponding to the quaternion $\mathbf{q}$, and $\otimes$ denotes the quaternion product. The matrices $\mathbf{B}_0, \mathbf{B}_1 \in \mathbb{R}^{3 \times 4}$ are constant and depend on the quadcopter's geometry. The parameters of the Bitcraze Crazyflie 2.1 robot are used, which, with a very low thrust-to-weight ratio of 1.3 (i.e., $0 \leq u_i \leq 1.3 \times g \times m/4$), pose significant challenges for kinodynamic motion planning.

IV. IDB-A* - OVERVIEW

Our iterative approach, which combines search and optimization, is detailed in Algorithm 1. We require a *large* set of *motion primitives* $\mathcal{M}_L$, which will be used incrementally and can be computed offline. Motion primitives are short trajectories that fulfill our dynamics (see Definition 1 and Section VII for a formal definition and details on primitive generation). In every iteration of iDb-A*, the following steps are performed:

1) We increase the number of available motion primitives for the search (by choosing new primitives from $\mathcal{M}_L$) and decrease the allowed discontinuity bound δ (Lines 4 and 5).

2) The discrete planner, Db-A*, computes a trajectory using the current set of motion primitives. This trajectory may include a bounded violation of the dynamic constraints (`Db-A*` in Line 6, see Section V).

3) The result of Db-A* is used to initialize an optimization-based motion planner that attempts to compute a feasible and locally optimal trajectory (Line 8). See `Optimization` (Section VI).

4) Additional motion primitives are extracted from the output of the trajectory optimization (`Extract Primitives` in Line 12).

iDb-A* executes a sequence of A*-searches using a growing, randomized set of motion primitives, akin to Batch Informed Tree (BIT*) [48], a successful sampling-based planner for geometric motion planning. In each iteration, the computation time and the success of the search and optimization steps depend on the number of motion primitives $n_i = |\mathcal{M}_i|$ and the allowed discontinuity bound δ_i. We can choose `AddPrimitives` (Line 4) and `DecreaseDelta` (Line 5) so that these parameters converge to, respectively, ∞ and 0 to ensure asymptotic optimality (see Section VIII). In our implementation, we choose geometric sequences $n_{i+1} = n_i n_r$ and $\delta_{i+1} = \delta_i \delta_r$, with $n_r > 1$, $\delta_r < 1$, and initial values n_0, δ_0, but different sequences are also possible. In particular, we find this strategy easier to tune than the alternative approach presented in our prior work [11], where we choose only the scheduling for the number of motion primitives and estimate the discontinuity bound based on a desired branching factor.

Motion primitives can also be extracted online in Algorithm 1. The `ExtractPrimitives` procedure utilizes the output of the optimization by dividing the trajectory into small sections. The resulting primitives can be particularly useful for the planning problem at hand as they are computed with full knowledge of the environment.

A visual representation of some key components of iDb-A* is shown in Fig. 2 using the problem *Planar rotor – Recovery*
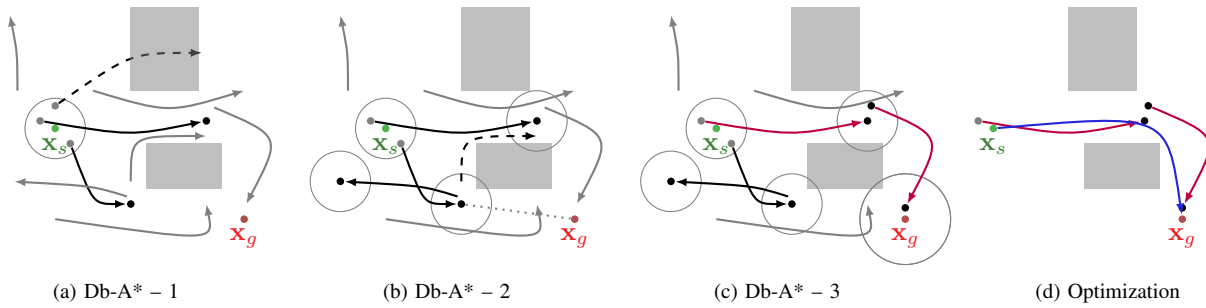
(a) Db-A* – 1   (b) Db-A* – 2   (c) Db-A* – 3   (d) Optimization

Fig. 3. (a, b, c) A graphical description of Db-A*. The gray edges represent motion primitives, and the states $\mathbf{x}_s$ and $\mathbf{x}_g$ are the start and the goal, respectively. (a) Given an initial state $\mathbf{x}_s$, we can only apply primitives that start with a discontinuity lower than $\alpha\delta$ (gray circumference) and are collision-free. The applicable primitives are shown with solid black edges. (b) The search is ordered by a heuristic (e.g., the *Euclidean* heuristic, shown with a dotted line for the best node) and the cost-to-come. When expanding a node, we create new states only if they are not within $(1-\alpha)\delta$ of a previously discovered state (i.e., the dashed edge is not expanded). (c) The search is terminated when a node close to the goal is expanded. (d) The solution of Db-A* will be used to warm-start a trajectory optimization algorithm that repairs the discontinuities and locally optimizes the trajectory. The optimized trajectory is shown in blue.

Algorithm 1: iDb-A* – Iterative Discontinuity Bounded A*

Input: $\mathbf{x}_s, \mathbf{x}_g, \text{step}, \mathcal{X}_{\text{free}}, \mathcal{U}, \mathcal{M}_{\text{L}}$
Result: $\mathbf{X}, \mathbf{U}$

1   $\mathcal{M}_0 \leftarrow \emptyset$ ▷ *Initial Set of motion primitives*
2   $c_{\max} \leftarrow \infty$ ▷ *Solution cost bound*
3   **for** $i = 1, 2, \ldots$ **do**
4     $\mathcal{M}_i \leftarrow \mathcal{M}_{i-1} \cup \texttt{AddPrimitives}(\mathcal{M}_L, i)$
5     $\delta_i \leftarrow \texttt{DecreaseDelta}(i)$
6     $\mathbf{X}_d, \mathbf{U}_d \leftarrow \texttt{Db-A}\star(\mathbf{x}_s, \mathbf{x}_g, \mathcal{X}_{\text{free}}, \mathcal{M}_i, \delta_i, c_{\max})$
7   **if** $\mathbf{X}_d, \mathbf{U}_d$ *successfully computed* **then**
8       $\mathbf{X}, \mathbf{U} \leftarrow \texttt{Optimization}(\mathbf{X}_d, \mathbf{U}_d, \mathbf{x}_s, \mathbf{x}_g, \text{step}, \mathcal{X}_{\text{free}}, \mathcal{U})$
9   **if** $\mathbf{X}, \mathbf{U}$ *successfully computed* **then**
10         $\texttt{Report}(\mathbf{X}, \mathbf{U})$ ▷ *New solution found*
11         $c_{\max} \leftarrow \min(c_{\max}, J(\mathbf{X}, \mathbf{U}))$ ▷ *Cost bound*
12     $\mathcal{M}_i \leftarrow \mathcal{M}_i \cup \texttt{ExtractPrimitives}(\mathbf{X}, \mathbf{U})$

---

obstacles.

V. DISCONTINUITY BOUNDED A* SEARCH

Discontinuity Bounded A* (Db-A*) is a search algorithm that uses a set of motion primitives, which are connected while allowing for a maximum discontinuity.

A motion primitive is a sequence of states and controls that fulfill the dynamics of the system. Formally,

Definition 1 (Motion Primitive). *A motion primitive $m = (\mathbf{X}, \mathbf{U}, \mathbf{x}_s, \mathbf{x}_f, c)$ is a sequence of states $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_N)$, $\mathbf{x}_k \in \mathcal{X}$, and controls $\mathbf{U} = (\mathbf{u}_0, \ldots, \mathbf{u}_{N-1})$, $\mathbf{u}_k \in \mathcal{U}$ that fulfill the dynamics $\mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k)$. It connects the start state $\mathbf{x}_s = \mathbf{x}_0$ and the final state $\mathbf{x}_f = \mathbf{x}_N$, with a corresponding cost $c \in \mathbb{R}^+$. The length of the motion primitive (i.e., the number of states and controls) is randomized.*

In the following, we rely on a user-specified *metric* $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$, which measures the distance between two states (e.g., a weighted Euclidean norm). We assume that $\langle \mathcal{X}, d \rangle$ is a metric space in order to use efficient nearest-neighbor data structures, such as k-d trees.

Definition 2. *The pair of sequences $\mathbf{X} = \langle \mathbf{x}_0, \ldots, \mathbf{x}_K \rangle$, $\mathbf{U} = \langle \mathbf{u}_0, \ldots, \mathbf{u}_{K-1} \rangle$ is a δ-discontinuity bounded solution (with*

$\delta > 0$*) to the kinodynamic motion planning problem* (3) *if and only if the following conditions hold:*

$$d(\mathbf{x}_{k+1}, \text{step}(\mathbf{x}_k, \mathbf{u}_k)) \leq \delta \quad \forall k \in \{0, \ldots, K-1\}, \quad \text{(5a)}$$
$$\mathbf{u}_k \in \mathcal{U} \quad \forall k \in \{0, \ldots, K-1\}, \quad \text{(5b)}$$
$$\mathbf{x}_k \in \mathcal{X}_{\text{free}} \quad \forall k \in \{0, \ldots, K\}, \quad \text{(5c)}$$
$$d(\mathbf{x}_0, \mathbf{x}_s) \leq \delta, \quad \text{(5d)}$$
$$d(\mathbf{x}_K, \mathbf{x}_g) \leq \delta. \quad \text{(5e)}$$

Intuitively, Definition 2 enforces that the sequences connect the start and goal states with a bounded error δ in the dynamics, which corresponds to "stitching" primitives together.

A. Algorithm

Our approach to computing such sequences is Discontinuity Bounded A* (Db-A*).

Db-A*, like A*, is an informed search that relies on a heuristic $h : \mathcal{X} \to \mathbb{R}$ to explore an implicitly defined directed graph efficiently. Nodes in the graph represent states, and an edge between two nodes indicates that there exists a motion that connects the states, allowing up to δ-discontinuity.

The algorithm is shown in Algorithm 2, and Fig. 3 provides a graphical representation. Db-A* keeps track of nodes to explore using a priority queue, which is sorted by the lowest $f(\mathbf{x}) = g(\mathbf{x}) + h(\mathbf{x})$ value, where $g(\mathbf{x})$ is the cost-to-come. The overall structure is the same as in A*: The OPEN priority queue $\mathcal{O}$ is initialized with the start state (Line 1). At each iteration, we remove the first element from $\mathcal{O}$ (Line 3), and that node is expanded using the applicable collision-free motion primitives (Lines 8 to 10). A motion m is applicable in state $\mathbf{x}$ if its start state $m.\mathbf{x}_s$ is within a distance of at most $\alpha\delta$ (Line 7), resulting in a new state $m.\mathbf{x}_f$.

New states are added to $\mathcal{O}$ (Line 17) only if they are not within $(1 - \alpha)\delta$ of previously discovered nodes. If the state is close to a previous node, the previous node is updated if the new cost-to-come is reduced (Line 22). Therefore, unlike A*, we consider two states to be equivalent if they are within $(1 - \alpha)\delta$ of each other.

For computing and updating the cost to come, we consider the cost of the motion primitive $m.c$ and the cost of the discontinuity bound using a lower bound function $l : \mathcal{X} \times \mathcal{X} \to \mathbb{R}^+$

Algorithm 2: Db-A* – Discontinuity Bounded A*

Input: $\mathbf{x}_s, \mathbf{x}_g, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta, c_{\max}$
Result: $\mathbf{X}_d, \mathbf{U}_d$ or Infeasible

1   $\mathcal{O} \leftarrow \{Node(\mathbf{x} : \mathbf{x}_s, g : 0, h : h(\mathbf{x}_s), p : None, a : None)\}$
    ▷ *Initialize open list (priority queue)*
2   $\mathcal{C} \leftarrow \{\}$ ▷ *Initialize list of closed nodes*
3   **while** $|\mathcal{O}| > 0$ **do**
      ▷ *Remove node with lowest f-value*
4     $n \leftarrow \texttt{PriorityQueuePop}(\mathcal{O})$
5   **if** $d(n.\mathbf{x}, \mathbf{x}_g) \leq \delta$ **then**
6   **return** $\mathbf{X}_d, \mathbf{U}_d$ ▷ *Trace back solution*

      ▷ *Find applicable motion primitives with discontinuity up to $\alpha\delta$*
7     $\mathcal{M}' \leftarrow \texttt{NearestNeighborQuery}(n.\mathbf{x}, \mathcal{M}, \alpha\delta)$
8   **foreach** $m \in \mathcal{M}'$ **do**
9   **if** $m \notin \mathcal{X}_{\text{free}}$ **then**
10   **continue**   ▷ *Motion is not collision-free*
11       $\mathbf{x}' \leftarrow m.\mathbf{x}_f$ ▷ *Tentative new state*
12       $g' \leftarrow n.g + m.c + l(n.\mathbf{x}, m.\mathbf{x}_s)$ ▷ *Tentative g score*
13   **if** $g' > c_{\max}$ **then**
14   **continue**

      ▷ *Check if we have previously discovered states within $(1-\alpha)\delta$*
15       $\mathcal{N} = \texttt{NearestNeighborQuery}(\mathbf{x}', \mathcal{O} \cup \mathcal{C}, (1-\alpha)\delta)$
16   **if** $\mathcal{N} = \emptyset$ **then**
17         $\texttt{PriorityQueueInsert}(\mathcal{O}, Node(\mathbf{x}', g', h(\mathbf{x}'), n, m))$
18   **else**
19   **foreach** $s \in \mathcal{N}$ **do**
20           $g'' = g' + l(\mathbf{x}', s.\mathbf{x})$
21   **if** $g'' < s.g$ **then**
22             $s = \texttt{UpdateNode}(\{g : g'', p : n, a : m\})$ ▷ *Update node. If it is in closed list, reinsert in open list.*

23   Insert n in $\mathcal{C}$
24   **return** Infeasible

---

of the true cost. Therefore, given a state $\mathbf{x}$ with cost to come $g(\mathbf{x})$, the cost of a new state $\mathbf{x}' = \mathbf{x} \oplus m = m.\mathbf{x}_f$ is $g(\mathbf{x}') = g(\mathbf{x}) + l(\mathbf{x}, m.\mathbf{x}_s) + m.c$ (Line 12).

The search terminates when we find a node that is within δ distance of the goal state (Line 6).

For efficient search, we employ two k-d trees. The first tree indexes the start states of all provided motion primitives, which can be done once at the beginning. The second k-d tree contains the states of all explored nodes and grows dynamically. It is used to find nearby previously explored states. The discontinuity with a magnitude of up to δ may occur in two cases: first, when we select suitable motion primitives for expansion (Line 7), and second, when we prune a potential new node in favor of already existing states (Line 19). The tradeoff between the two can be adjusted by a user-specified parameter $\alpha \in (0, 1)$.

B. Heuristic Functions

In kinodynamic motion planning, three heuristic functions $h(\mathbf{x})$ are particularly relevant:

a) Euclidean Heuristic: The *Euclidean* heuristic is based on the Euclidean distance to the goal, considering state and control constraints such as maximum velocity or acceleration, while ignoring dynamics and obstacles. It is usually computed by a combination of weighted Euclidean or infinity norms and does not require any precomputation.

b) Roadmap Heuristic: The *Roadmap* heuristic approximates the collision-free space using a geometric roadmap, thus taking collisions and control bounds into account but ignoring the dynamics. It requires a precomputation step to build the geometric roadmap, which can be reused between iterations of iDb-A*, and it is usually more informative in problems where obstacles play a significant role. Given a finite set of state-cost pairs $S = \{(\mathbf{s}_i, c_i) | \ \mathbf{s}_i \in \mathcal{X}, c_i \in \mathbb{R}\}$, the heuristic function is given by:

$$h(\mathbf{x}) = \min_{(\mathbf{s}_i, c_i) \in S: \ d(\mathbf{s}_i, \mathbf{x}) \leq R} \{c_i + l(\mathbf{s}_i, \mathbf{x})\}, \quad (6)$$

where l is a lower bound on the cost for reaching $\mathbf{s}_i$ from $\mathbf{x}$, and R is a user-defined connection radius. To compute S, we construct a roadmap with randomly sampled configurations and annotate each vertex with the geometric cost-to-go (i.e., using the *Euclidean* heuristic for each collision-free edge). Each query requires a nearest-neighbor search (implemented using a k-d tree).

c) Blind Heuristic: Lastly, we also evaluate the *Blind* heuristic, where $h(\mathbf{x}) = 0, \ \forall \mathbf{x}$. This heuristic is motivated by systems where the dynamics play a central role, such that feasible trajectories of the robot strongly differ from straight lines in the state space (e.g., the acrobot), making the *Euclidean* and *Roadmap* heuristics uninformative.

The three heuristic functions are evaluated during the experimental evaluation (Section IX).

Example 3. *Consider the unicycle robot, with state and dynamics as in Example 1. Given a state* $\mathbf{x} = [x, y, \theta]$, *goal* $\mathbf{g} = [g_x, g_y, g_\theta]$, *and the control bounds* $|v| \leq v_{\max}$, $|w| \leq w_{\max}$, *the Euclidean heuristic is:*

$$h(\mathbf{x}) = \max\{ \ v_{\max}^{-1} \|[x, y] - [g_x, g_y]\|,$$
$$w_{\max}^{-1} D_\theta(x_\theta, g_\theta)\}, \quad (7)$$

where $D_\theta(\cdot, \cdot)$ *is the distance metric in SO(2).*

C. Equivalence Between Continuous States

A fundamental issue when applying search algorithms in continuous spaces is that the set of possible reachable states is infinite. The search algorithm will unnecessarily expand similar states, especially when the heuristic is not informative.

To mitigate this issue, in Db-A* we have defined a notion of similarity or equivalence between states, often referred to as duplicate detection in related work [38], [49], [50]. In Algorithm 2, a state is considered not novel if it is close to a previously discovered state, in which case the state is pruned. This makes Db-A* incomplete and suboptimal for fixed values of δ. As Db-A* runs for decreasing δ inside iDb-A*, the size of the equivalence class is iteratively reduced. In combination with subsequent optimization, we found our duplicate detection to be sufficient for both good practical performance and asymptotic optimality (Section VIII).

D. Invariance and Equivariance in the Motion Primitives

To decide which motion primitives are applicable in a state (Line 7), we can exploit invariance and equivariance in the

system dynamics, which allows us to reuse the same primitive in different states with smaller discontinuities.

A prominent example is the translation invariance of the dynamics of mobile robots. Intuitively, a valid motion primitive can be "translated" to match other starting states so that there is no discontinuity in the translation components of the state. This concept is formalized in Section VII, where we provide two examples: translation invariance for a car-like robot and translation and linear velocity invariance for flying robots.

From an implementation perspective, to account for invariances, all primitives are stored in a canonical form (e.g., with 0 translation component) inside a k-d tree. At runtime, we transform the query state into the canonical form to check which primitives are applicable, and the valid primitives are then transformed on-the-fly to expand the query state.

E. Efficient Collision Checks with Collision Shapes

Collision checking is one of the most expensive operations in motion planning. To check collisions between the environment and a motion primitive (Line 9), we use either precomputed collision shapes of the motion primitive (if available) or check collisions at a small temporal resolution. Importantly, precomputed collision shapes can also be transformed online for any translation or rotation of the motion primitive. When available, collision shapes are considerably faster than checking individual configurations at a chosen resolution.

In practice, we observe that the running time of Db-A* is dominated by both nearest neighbor searches to find neighboring states and applicable motion primitives, and by collision detection (see Fig. 7 in Section IX-E).

VI. Trajectory Optimization

For the `Optimization` subroutine (Line 8 in Algorithm 1), we use gradient-based trajectory optimization. We assume that the derivatives of the dynamics, the distance function, and the collision constraints can be computed efficiently, e.g., using analytical expressions, a differentiable simulator, or finite differences. For collisions, we now require a signed distance function instead of a binary collision check.

The objective of the optimization is to solve the original kinodynamic motion planning problem (3), using the output of Db-A* as an initial guess, $\mathbf{X}_d = \{\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_K\}$, $\mathbf{U}_d = \{\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{K-1}\}$.

Even with the initial guess from Db-A*, the optimization problem (3) is challenging for gradient-based optimization, especially when starting with large discontinuities.

The problem is nonconvex even for systems with linear dynamics and constraints, and the infeasible initial guess and underactuation of the systems prevent the use of time-optimal path tracking approaches (e.g., [51], [52]).

For the optimization step of iDb-A*, we choose to perform a joint optimization of trajectory and terminal time using nonlinear optimization.

a) Joint Optimization of Trajectory and Terminal Time (Free-dt): This approach adds the duration of the time interval as an optimization variable for joint nonlinear optimization of time and trajectory:

$$\min_{\mathbf{X}, \mathbf{U}, \Delta t} \sum_k \Delta t \; j(\mathbf{x}_k, \mathbf{u}_k), \tag{8a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)\Delta t, \tag{8b}$$

$$\text{Constraints } (3c), (3d) \text{ and } (3e). \tag{8c}$$

Here, Δt is a variable, initialized to Δt_{ref}, the reference value used for time-discretization in the motion primitives for the given dynamical system. The number of time steps K is fixed. After solving (8), we would like to have the solution trajectory discretized with the original time step duration for consistency. Thus, we recompute the state and control trajectories using the reference time step Δt_{ref}. This requires i) interpolation of the solution of (8) with Δt_{ref}, and ii) a second run of trajectory optimization, now with fixed $\Delta t = \Delta t_{\text{ref}}$ to repair the small errors arising from the Euler integration with different step sizes (note that the second optimization is very efficient because the interpolation of the solution of (8) is already accurate).

b) Algorithm for Trajectory Optimization: Our approach to trajectory optimization requires solving nonlinear optimal control problems. In our previous work [11], we used a direct control method (where both states and controls are variables), namely k-order optimization [8], and the Augmented Lagrangian algorithm. In this revised version, we switch to an indirect control method (where only controls are variables), Differential Dynamic Programming (DDP), which ensures precise dynamics during shooting and therefore more reliable convergence to locally optimal solutions in systems with complex dynamics.

Differential Dynamic Programming is a second-order method for solving optimal control problems that iteratively computes a quadratic approximation of the cost-to-go using a backward pass and updates states and controls using a forward pass. For more details, we refer to [26], [53]. To deal with collisions, goal constraints, and state and control bounds, we use a squared penalty method—adding all constraints in the cost term with a squared penalty.

In particular, we use *feasibility-driven DDP* [26], which can be warm-started with an infeasible sequence of states and actions, providing a good balance between local convergence and globalization. Importantly, we emphasize that users may use other formulations and constrained optimization algorithms to solve (8). In our experiments, we also evaluate a multiple shooting formulation solved with Sequential Quadratic Programming (SQP) [54].

c) Alternative approaches for optimizaion of trajectories with free terminal time: We also evaluate three additional formulations for the optimization step of iDb-A*, based on different approaches in the optimization and control literature: a hierarchical time search, and two sliding window approaches that try to repair the trajectory in a sequential fashion, based on receding horizon model predictive control and contouring control. These methods are described in Section A in the appendix.

VII. Motion Primitives

In our framework, we define a motion primitive as a valid trajectory that fulfills the dynamics, control, and state constraints, disregarding collisions with the environment. A formal definition is provided in Definition 1 in Section V.

A. Generation of Locally Optimal Motion Primitives

In the problem setting outlined in Section III, a key observation is that motion primitives can be precomputed because they are independent of the collision-free space and the start and goal configurations of a particular motion planning problem.

Our algorithms, iDb-A* and Db-A*, are agnostic about how the primitives have been generated. However, the theoretical properties of the algorithms depend on the properties of the set of primitives (Section VIII). In our implementation of iDb-A*, we use locally optimal motion primitives computed with trajectory optimization.

To generate motion primitives, we solve two-point boundary value problems with random start and goal configurations in free space using nonlinear optimization. In contrast to the typical approach in sampling-based motion planning of sampling control sequences at random, our strategy results in a superior primitive distribution, especially for systems with unstable dynamics (e.g., flying robots). Our approach achieves better coverage of the state space and produces smoother and lower-cost motion primitives, which are key factors contributing to the success of the algorithm (see Section IX).

Specifically, we generate motion primitives offline using the following three steps: First, uniform random sampling of start and goal configurations in free space; second, solving (3) with trajectory optimization using the hierarchical time search approach and a trivial initial guess; and third, splitting the resulting motion into multiple pieces of random length.

We observe that our strategy generates a good distribution of motion primitives. However, it requires several hours of offline computation with a standard CPU. In the case of flying robots, most of the time is spent attempting to find trajectories where the goal is not reachable within the given time horizon or where trajectory optimization fails to converge. Our sampling approach might bias the primitives' distribution towards configurations that are easy to connect to other configurations. However, the results in Section IX confirm that the primitives are diverse and can solve a wide range of problems.

B. Invariance and Equivariance in System Dynamics

Several robotic systems of interest exhibit symmetries and invariances in the dynamics that can be exploited to reduce the required number of motion primitives for planning.

Specifically, invariance/equivariance enables the adaptation of primitives on-the-fly to match some components of the state space exactly in our search algorithm, Db-A*. This significantly reduces the number of primitives required to cover the state space, resulting in smaller discontinuities, reduced memory requirements, and faster nearest neighbor searches.

A prominent example is translation invariance, a property that holds for many mobile robots, such as differential-drives,

cars, airplanes, and multirotors. In these systems, we can decompose the state into two components, $\mathbf{x} = [\mathbf{x}^t, \mathbf{x}^r]$, where $\mathbf{x}^t$ represents translations and $\mathbf{x}^r$ contains rotations and possibly velocities. The dynamics $f(\mathbf{x}, \mathbf{u}) = f(\mathbf{x}^r, \mathbf{u})$ only depend on the non-translation part of the state.

We are interested in invariances that preserve optimality. For instance, the translation of a primitive in translation invariant systems retains optimality for a running cost of type $j(\mathbf{x}, \mathbf{u}) = r(\mathbf{u})$, e.g., a minimum time trajectory $r(\mathbf{u}) = 1$, or minimum control effort $r(\mathbf{u}) = \|\mathbf{u}\|^2$.

Example 4 (Translation Invariance in the Unicycle). *Consider the unicycle from Example 1. The dynamics $f(\mathbf{x}, \mathbf{u}) = f(\theta, \mathbf{u}) = [v\cos(\theta), v\sin(\theta), w]$ depend only on the orientation θ, but not on the position $[x, y]$. Using translation invariance, we can translate a motion primitive $m = (\mathbf{X}, \mathbf{U}, \mathbf{x}_s, \mathbf{x}_f, c)$ with $\mathbf{t} \in \mathbb{R}^2$, resulting in $m \oplus \mathbf{t} = m' = (\mathbf{X}', \mathbf{U}' = \mathbf{U}, \mathbf{x}'_s, \mathbf{x}'_f, c' = c)$. The states in $\mathbf{X}'$, $\mathbf{x}'_s$, and $\mathbf{x}'_f$ are transformed as follows:*

$$\mathbf{x}'[x] = \mathbf{x}[x] + \mathbf{t}[x], \tag{9a}$$
$$\mathbf{x}'[y] = \mathbf{x}[y] + \mathbf{t}[y], \tag{9b}$$
$$\mathbf{x}'[\theta] = \mathbf{x}[\theta]. \tag{9c}$$

The operator $[\bullet]$ indicates the $\bullet$-component of a state $\mathbf{x}$ or translation vector $\mathbf{t}$ (e.g., $\mathbf{x}[x] \in \mathbb{R}$ is the "x-component" of the state $\mathbf{x}$).

In some second-order systems, such as the quadrotor, the acceleration depends only on the orientation and the angular velocity but is invariant to both translation and linear velocity. Thus, we can modify primitives to match the starting position and velocity.

Example 5 (Translation and Linear Velocity Invariance in the Quadrotor). *The second-order dynamics of the quadrotor from Example 2 depend only on the rotation and angular velocity $\mathbf{q}, \mathbf{w}$ but not on the position $\mathbf{p}$ or linear velocity $\mathbf{v}$. We can transform a motion primitive $m = (\mathbf{X}, \mathbf{U}, \mathbf{x}_s, \mathbf{x}_f, c)$ with $\mathbf{t} = [\mathbf{t}_p, \mathbf{t}_v]$, $\mathbf{t}_p \in \mathbb{R}^3$, $\mathbf{t}_v \in \mathbb{R}^3$, resulting in $m \oplus \mathbf{t} = m' = (\mathbf{X}', \mathbf{U}' = \mathbf{U}, \mathbf{x}'_s, \mathbf{x}'_f, c' = c)$. The states in $\mathbf{X}'$, $\mathbf{x}'_s$, and $\mathbf{x}'_f$ are transformed as follows:*

$$\mathbf{x}'[v] = \mathbf{x}[v] + \mathbf{t}_v, \tag{10a}$$
$$\mathbf{x}'[w] = \mathbf{x}[w], \tag{10b}$$
$$\mathbf{x}'[q] = \mathbf{x}[q], \tag{10c}$$
$$\mathbf{x}'_k[p] = \mathbf{x}_k[p] + \mathbf{t}_p + k\mathbf{t}_v\Delta t. \tag{10d}$$

While invariance and equivariance are advantageous properties, they require an individual study of each new dynamical system. For simplicity, in our work, we focus only on translation invariance and linear velocity invariance as two classical and ubiquitous properties. Additional properties, such as rotation invariance (for car-like robots), rotation symmetries (for 3D quadcopters), or angular rotational invariance (for planar multirotors), could be exploited to improve performance in particular systems.

VIII. Theoretical Properties

In this section, we analyze the theoretical properties of iDb-A* and argue that it is asymptotically optimal under very mild

assumptions. Intuitively, given enough computational time, iDb-A* (Algorithm 1) will compute the optimal solution, because in each iteration, we add more primitives and reduce the allowed discontinuity δ, eventually producing an initial guess that is close to the optimal solution, which is then locally repaired and optimized with trajectory optimization.

A. Kinodynamic Motion Planning

Assumption 1. *The dynamics function* (1) *has a bounded Lipschitz constant for both states and controls. Moreover, there exists a δ-robust trajectory that solves the kinodynamic motion planning problem.*

Note: A δ-robust trajectory has both obstacle clearance and dynamical clearance of δ. These assumptions are standard in kinodynamic motion planning, and we refer to [21] for a formal definition.

Remark 1. *In a system with Lipschitz dynamics, the discrete-time dynamics converge to the continuous-time dynamics as the time step discretization approaches zero. Thus, we limit our study to the time-discretized system* (2) *used in our framework.*

B. Db-A*

Theorem 1. *Sequences $\mathbf{X}$ and $\mathbf{U}$ returned by Db-A* (Algorithm 2) are δ-discontinuity bounded solutions to the given motion planning problem (Definition 2).*

Proof. When Db-A* terminates, we trace back the solution by following the parent pointers and obtain a sequence of motion primitives $[m_1, \ldots, m_N]$. This sequence defines the sequence of controls $\mathbf{U} = [m_1.\mathbf{U}, \ldots, m_N.\mathbf{U}]$, and states $\mathbf{X} = [m_1.\mathbf{X}_{[:-1]}, \ldots, m_N.\mathbf{X}]$, i.e., we take all but the last state for each primitive except for the last one, which should also include the last state. During Db-A*, we expand a node n with a motion m if the start state $m.\mathbf{x}_s$ is at most $\alpha\delta$ away from the current state $n.\mathbf{x}$. Any node n is reached by a motion $\tilde{m}$ that ends at point $\tilde{m}.\mathbf{x}_f$, which is at most $(1-\alpha)\delta$ from $n.\mathbf{x}$. Using the triangle inequality of the metric space, $d(\tilde{m}.\mathbf{x}_f, m.\mathbf{x}_s) \leq d(\tilde{m}.\mathbf{x}_f, n.\mathbf{x}) + d(n.\mathbf{x}, m.\mathbf{x}_s) \leq \delta$. Thus, (5a) holds for all connections between motion primitives, and (5d) holds for the state in the first motion. We already know that $\mathbf{x}_k \in \mathcal{X}$, $\mathbf{u}_k \in \mathcal{U}$ for all k (5b). Motions are only used as edges if the entire motion is within $\mathcal{X}_{\text{free}}$, thus (5c) holds. Finally, (5e) holds by the termination conditions. $\square$

Definition 3. *Given a start state $\mathbf{x}_s$ and a goal state $\mathbf{x}_g$, a set of motion primitives $\mathcal{M}$, and a discontinuity bound δ, we define the implicit graph $G_{\delta,\mathcal{M}} = (V, E)$ with,*

$$V = \bigcup_{m \in \mathcal{M}} \{m.\mathbf{x}_s, m.\mathbf{x}_f\} \cup \{\mathbf{x}_s, \mathbf{x}_g\}, \tag{11a}$$

$$\begin{aligned} E = &\{(m.\mathbf{x}_s, m.\mathbf{x}_f) \mid m \in \mathcal{M}\} \cup \{(m.\mathbf{x}_f, m'.\mathbf{x}_s) \\ &\mid m, m' \in \mathcal{M} \text{ and } d(m.\mathbf{x}_f, m'.\mathbf{x}_s) \leq \delta\} \\ &\cup \{(\mathbf{x}_s, m.\mathbf{x}_s) \mid m \in \mathcal{M} \text{ and } d(\mathbf{x}_s, m.\mathbf{x}_s) \leq \delta\} \\ &\cup \{(m.\mathbf{x}_f, \mathbf{x}_g) \mid m \in \mathcal{M} \text{ and } d(m.\mathbf{x}_f, \mathbf{x}_g) \leq \delta\}. \end{aligned} \tag{11b}$$

Lemma 1. *For an arbitrary fixed value of $\delta > 0$, Db-A* is incomplete and suboptimal when searching on the implicit graph $G_{\mathcal{M},\delta}$.*

Proof. Consider an example where a robot has to move through a narrow door to navigate to an adjacent room. Even if a δ-discontinuity bounded solution using the available primitives exists (i.e., a path in the graph $G_{\delta,\mathcal{M}}$), Db-A* may not find it because motions are expanded in a random order and only if no previous node is within distance $(1-\alpha)\delta$. This could potentially prune a node that is required in the solution. Since Db-A* is incomplete, it cannot guarantee that no better δ-discontinuity bounded solution exists once it finds one. $\square$

Theorem 2. *For a given set of motion primitives $\mathcal{M}$, there exists a $\delta > 0$ such that Db-A* is complete and optimal when searching in the implicit graph $G_{\mathcal{M},\delta}$.*

Proof. If the choice of a given δ_0 renders Db-A* incomplete on the graph $G_{\mathcal{M},\delta_0}$, we can always reduce δ, resulting in a different graph, where the nodes in the solution are either not pruned during the Db-A* search, or the graph becomes unsolvable. Note that since decreasing δ may render the graph unsolvable, this property is valuable in an asymptotic setting, used to demonstrate asymptotic optimality when incrementally adding more motion primitives. $\square$

C. Optimization

Assumption 2. *(Convergence of the Optimizer) Let $(\mathbf{X}_d, \mathbf{U}_d)$ be a δ-discontinuity bounded solution (Definition 2) to the planning problem. Then, there exists a (small) $\delta > 0$ such that trajectory optimization converges to a locally optimal solution $(\mathbf{X}^*, \mathbf{U}^*)$ of the original kinodynamic motion planning problem* (3).

Discussion: Nonlinear constrained optimization algorithms for trajectory optimization have convergence guarantees toward stationary points: these are either points that satisfy the first-order optimality conditions—thereby converging to a locally optimal feasible solution; or points that locally minimize the constraint violation.

In general, these methods typically require the dynamics, distance function, and collision constraints to have smoothness or bounded derivatives. The exact conditions for convergence can vary slightly among different trajectory optimization algorithms, such as Differential Dynamic Programming [55], Sequential Convex Optimization [56], the Augmented Lagrangian [57], and Interior Points [58].

Specifically, we assume that for a small δ, the initial guess is close to a feasible solution, thereby ensuring that the optimizer converges to a feasible and locally optimal solution. We argue that this is a very reasonable assumption, especially when the optimal solution is a δ-robust trajectory (Assumption 1), as often assumed in the literature. The experimental success rate of our planner (Section IX) also supports this assumption.

D. Motion Primitives

Our optimization-based approach for generating motion primitives, as well as propagation of random control inputs

from random starting points, creates a set of motion primitives that asymptotically covers the state space, which will be required to prove the asymptotic optimality of iDb-A*.

Definition 4. *A set of primitives $\mathcal{M}$ covers the state space with discontinuity $\epsilon > 0$ if and only if, for all pairs of states $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, there exists a ϵ-discontinuity bounded trajectory from $\mathbf{x}$ to $\mathbf{x}'$ (Definition 2) using the primitives $m \in \mathcal{M}$.*

Definition 5. *A method to generate motion primitives asymptotically covers the state if, for every $\tilde{\epsilon} > 0$, there exists a finite set of primitives $\tilde{\mathcal{M}}$, $|\tilde{\mathcal{M}}| < \infty$ such that Definition 4 holds. In other words, asymptotic coverage means that if $|\mathcal{M}| \to \infty$, then $\epsilon \to 0$ in Definition 4.*

Remark 2. *Motion primitives generated with our randomized optimization-based approach, as well as rollouts of random control inputs from random starting points, asymptotically covers the state space.*

E. iDb-A*

So far, we have demonstrated that there exists a small discontinuity bound $\delta > 0$ such that Db-A* finds an optimal discontinuity bounded trajectory (if one exists), and that trajectory optimization can be used to repair a δ-discontinuity bounded solution into a locally optimal solution.

To prove asymptotic optimality for iDb-A*, we use techniques from sampling-based motion planning to show that, as we increase the number of primitives, the solution of Db-A* converges towards an optimal discontinuity bounded solution (and thus, after optimization, we converge towards the optimal solution). In Algorithm 1, we iteratively reduce δ to achieve good anytime behavior. For the proof, it is sufficient to assume that we are using a small fixed discontinuity $\tilde{\delta} > 0$ such that Theorem 2 and Assumption 2 hold.

Theorem 3. *If the set of motion primitives asymptotically optimally covers the state space with discontinuity $\tilde{\delta}$, and assuming convergence of the optimizer, iDb-A* (Algorithm 1) is asymptotically optimal, i.e.,*

$$\lim_{n \to \infty} P(\{c_n - c^* > \epsilon\}) = 0, \ \forall \epsilon > 0, \tag{12}$$

where c_n is the cost in iteration n and c^* is the optimal cost.

Proof. We closely follow the proof strategy from previous work in sampling-based motion planning [19, Th. 3]. Let $S_1, \ldots, S_n$ be random variables denoting the suboptimality $c_n - c^*$. In every iteration of Algorithm 1, we either reduce the cost if we find a new solution or we remain at the same cost, i.e., $c_{n+1} \leq c_n$. We now aim to show that, with sufficient motion primitives, the solution of Db-A* will be close to the true solution, and thus can be correctly optimized by trajectory optimization. Crucially, in each iteration of iDb-A*, there is a positive probability that the new primitives will improve the solution, $E[S_n|S_{n-1}] \leq (1 - \omega)S_{n-1}$, i.e., in expectation, the solution improves by at least a constant amount $\omega > 0$ every iteration. This nonzero probability only holds if the motion primitives asymptotically cover the entire state space with discontinuity $\tilde{\delta}$ (Definition 5). Then, we have

$$E[S_n] = \int E[S_n|S_{n-1}]P(S_{n-1})dS_{n-1} \tag{13}$$

$$\leq (1 - \omega) \int S_{n-1}P(S_{n-1})dS_{n-1}$$

$$= (1 - \omega)E[S_{n-1}] = (1 - \omega)^{n-1}E[S_1].$$

Applying the Markov inequality, we have $P(S_n > \epsilon) \leq E[S_n]/\epsilon = (1 - \omega)^{n-1}E[S_1]/\epsilon$, which approaches 0 as n approaches infinity. $\square$

It remains an open question what the theoretical convergence rate of our proposed algorithm is, a property that is known for some sampling-based planners [20]. Empirically, we have shown that our initial solution is often much closer to the optimum compared to our baselines, and that the region of attraction for trajectory optimization is large enough to plan with a few primitives and large discontinuity bounds.

IX. EXPERIMENTAL EVALUATION

We evaluate iDb-A* on 43 problems that include 8 different dynamical systems in various environments. Most of the problems and systems are selected from previous work in kinodynamic motion planning [1], [30], [59], [60]. Additionally, we include several problems that require dynamic and agile maneuvers with multirotors.

The benchmark problems are available in *DynoBench*, our new benchmark library. It provides a C++ implementation of all the dynamical systems (including dynamics with analytical Jacobians, state, and bound constraints), collision and distance computation with the Flexible Collision Library (FCL), the environments (in human-friendly YAML files), and visualization tools in Python. In the trajectory optimization step, gradients and Jacobians are computed analytically, except for the signed distance function, for which we use finite differences.

Our implementation of iDb-A* and the other planners is available in our repository, along with the motion primitives and instructions to replicate the benchmark results. A visualization of each problem and the corresponding solution trajectories computed by our algorithm is available on our website.

A. Dynamical Systems and Environments

We include a diverse range of dynamical systems and environments, featuring varying state dimensionality (from 3 to 14), number of underactuated degrees of freedom, and controllability.

1) **Unicycle 1 (1st order)** has a 3-dimensional state space $[x, y, \theta] \in \mathcal{X} \subset \mathbb{R}^2 \times SO(2)$ and a 2-dimensional velocity control $[v, \omega] \in \mathcal{U} \subset \mathbb{R}^2$ [61]. The three variants (v0, v1, v2) use different control bounds. See Figs. 1a and 4e.

2) **Unicycle 2 (2nd order)** has a 5-dimensional state space $[x, y, \theta, v, \omega] \in \mathcal{X} \subset \mathbb{R}^4 \times SO(2)$ and a 2-dimensional acceleration control $[\dot{v}, \dot{\omega}] \in \mathcal{U} \subset \mathbb{R}^2$ [61]. See Fig. 5.
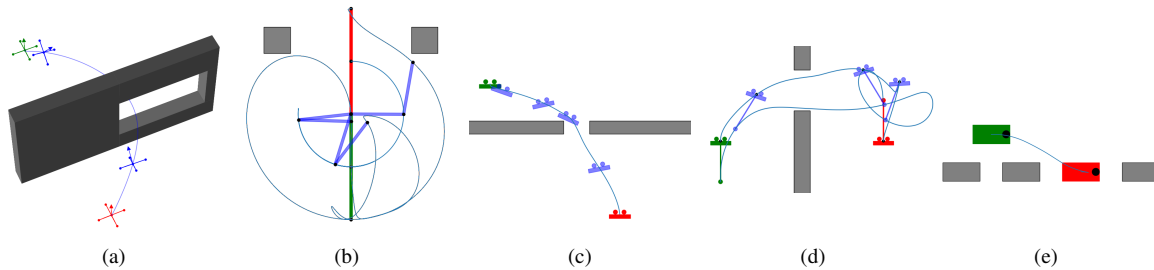
Fig. 4. Kinodynamic motion planning problems: Start and goal positions are represented in green and red, respectively. Obstacles are depicted in gray. Trajectories computed by iDb-A* are illustrated in blue. (a) *Quadrotor v1 – Window*, (b) *Acrobot – Swing up obstacles v1*, (c) *Planar rotor – Hole*, (d) *Rotor pole – Small window*, (e) *Unicycle 1 v0 – Park*.

3) **Car with trailer** has a 4-dimensional state space $[x, y, \theta_0, \theta_1] \in \mathcal{X} \subset \mathbb{R}^2 \times SO(2)^2$ and a 2-dimensional control space $[v, \phi] \in \mathcal{U} \subset \mathbb{R}^2$ (steering angle and velocity) [61]. See Fig. 1b.

4) **Acrobot** is a two-link planar manipulator actuated only at the middle joint [62]. It requires long trajectories to swing up the two links. See Fig. 4b.

5) **Quadrotor v0** has a 13-dimensional state space (position, orientation, and first-order derivatives) and a 4-dimensional control space (force for each of the four motors). Dynamics are defined in Example 2, and we use the parameters of the Crazyflie 2.1. The low thrust-to-weight ratio of 1.3 is very challenging for kinodynamic motion planning, and harsh initial conditions prevent the use of specialized methods [63], [64]. See Fig. 1d.

6) **Quadrotor v1**. The state space is the same as in *Quadrotor v0*. Controls are the total thrust and torques in the body frame, which make sampling-based methods more efficient but approximate real rotor-force limits. We use the system parameters and model from the OMPL APP repository, but increase the control bounds to perform agile maneuvers. See Fig. 4a.

7) **Planar rotor**. The input is the force in each rotor $\mathbf{u} = [f_1, f_2] \in \mathbb{R}^2$ and the state space is 6-dimensional $\mathbf{x} = [x, z, \theta, v_x, v_y, w] \in \mathbb{R}^5 \times SO(2)$. The thrust-to-weight ratio is also limited to 1.3. See Fig. 4c.

8) **Rotor pole** (Planar multirotor with pole) is a planar multirotor with an additional underactuated pendulum. The control space is the same as in *Planar rotor* (but with larger control bounds) and the state space has two additional degrees of freedom $[q, \dot{q}]$ for the pendulum. See Fig. 4d.

All systems use the explicit Euler integration (2), with $\Delta t = 0.1\,\mathrm{s}$ for all car-like robots, and $\Delta t = 0.01\,\mathrm{s}$ for the flying robots and the *Acrobot*, due to the fast rotational dynamics.

For most car-like robots, we consider three environments (*Kink*, *Park*, *Bugtrap*). For the *Acrobot* and *multirotors*, we use environments with and without obstacles to evaluate performance in settings that require both aggressive maneuvers (e.g., recovering from upside-down positions) and navigation around obstacles.

B. Algorithms

Following our previous work [11], the goal of this benchmark is to compare methods for kinodynamic motion planning that use different methodologies: search, optimization, and sampling. We compare our algorithm against state-of-the-art methods that have available open-source implementations.

For a **search-based** approach, we rely on SBPL (Search-based Planning Library), a commonly used C++ library. We generate our own primitives and make minor adjustments to the heuristic to enable time-optimal anytime planning using the provided implementation of ARA* in SBPL. SBPL requires the motion primitives to be connected without discontinuity and to span a lattice. We limit our evaluation to dynamic models that are readily available in SBPL, namely the *Unicycle 1 v0*.

For a **sampling-based** approach, we use SST* [1], which is implemented in OMPL [65] (Open Motion Planning Library). Since sampling-based kinodynamic approaches cannot reach a goal state exactly, we use a goal region instead and run subsequent trajectory optimization with fixed terminal time to generate an exact solution to the goal. The reported time does *not* include the time spent in trajectory optimization, thus providing a favorable lower bound.

For **optimization-based** planning, we choose a classic combination of a geometric motion planner and a trajectory optimizer, which we call `RRT*-TO` in the following. The motion planner generates an obstacle-free initial guess, ignoring the dynamics of the system, i.e., *planning with a simplified model*, and the optimizer uses this trajectory as an initial guess. As a motion planner, we use geometric RRT* (using the implementation in OMPL), which provides anytime behavior by incrementally improving the geometric trajectory. Importantly, the probability of finding a feasible solution with optimization might increase when using the multiple geometric initial guesses provided by RRT*. The initial guess provided by the geometric planner is collision-free but is often not accurate for dynamic constraints. Therefore, for trajectory optimization with free terminal time, we use a hierarchical strategy, where we solve a sequence of optimization problems with fixed terminal time, increasing the terminal time (Eq. (14) in Section A of the appendix). This approach is more robust and provides better success in this planner than the joint optimization of trajectory and time, which is used in `iDb-A*`.

For **iDb-A***, we implement Algorithm 1 and Algorithm 2 in C++. As a heuristic h, we use the Euclidean-based heuristic (e.g., distance divided by the upper bound of the speed, see Section V-B), because it is general, fast to compute, and does not require precomputation. For trajectory optimization, we perform joint optimization of trajectory and time (Section VI), which provides a good balance between local optimality, convergence, and computational time when using a good initial guess, as usually provided by Db-A*.

We provide an ablation study of the choices of heuristic and trajectory optimization strategy in Section IX-E, where we analyze and compare different alternatives.

The rates that control the number of primitives and the discontinuity bound in each iteration of `iDb-A*` (Algorithm 1) are set to $n_r = 1.5$, $\delta_r = 0.9$ in all problems. However, when the search terminates without finding a solution, we keep the discontinuity bound (almost) constant with $\delta_r = 0.999$ and only increment the number of primitives with a rate of $n_r = 1.5$. The parameter α in Algorithm 2 is set to 0.5. The initial number of motion primitives, discontinuity bound, and hyperparameters of the trajectory optimization are chosen per dynamical system. For example, in all problem instances with *Unicycle 1 v0*, we start with 100 primitives and a discontinuity $\delta_0 = 0.3$, and in all instances with *Quadrotor v0*, we start with 2000 primitives and $\delta_0 = 0.7$. The distance function d used to measure the distance between states in `Db-A*` is a weighted Euclidean norm, which uses different weights for the position, orientation, and velocity components of the state (e.g., in the *Unicycle 1* we use weights 1 and 0.5. Thus, a δ of 0.3 could represent up to $30\,\mathrm{cm}$ of discontinuity in position or $0.6\,\mathrm{rad}$ in orientation. Such discontinuities are large enough that the trajectory is not directly applicable to the real robot, but enables a fast search and can be efficiently repaired in the nonlinear trajectory optimization step of `iDb-A*`.

The offline generation of a valid primitive takes a few seconds, from $0.1\,\mathrm{s}$ for the *Unicycle 1 v0* to $6\,\mathrm{s}$ for a *Quadrotor v0* using a single core of a laptop computer with a CPU i7-1165G7@2.80GHz, where most of the time is spent attempting to solve random two-value boundary problems, which cannot be solved with short trajectories.

The benchmarking infrastructure is written in Python, and all tuning parameters for our algorithms and baselines can be found in our open-source repository.

For nearest-neighbors computation in `iDb-A*`, `SST*`, and `RRT*-TO`, we use NIGH [66] in single-core mode, which provides faster lookup and insertion times than the default Geometric Near-neighbor Access Tree (GNAT) implementation in OMPL. All trajectory optimization algorithms are implemented based on the DDP solver in Crocoddyl [26].

C. Metrics

In the following, we report the metrics:

1) Success Rate (p): The ratio of trials where a solution was found within the planning budget of $120\,\mathrm{s}$.
2) Median time required to find a solution (t^{st}).
3) Median cost when 50% of the trials have found a solution (J^{st}).

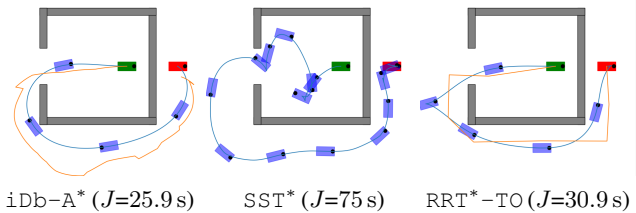

iDb-A* (J=25.9 s)   SST* (J=75 s)   RRT*-TO (J=30.9 s)

Fig. 5. Example of the first trajectory computed by `iDb-A*`, `SST*`, and `RRT*-TO` in *Unicycle 2 – Bugtrap*. The orange line in `iDb-A*` and `RRT*-TO` shows the initial guess trajectory before optimization. The first solution of `iDb-A*` has the lowest cost (J). The average computational time to get the first solution is $1.2\,\mathrm{s}$ in `iDb-A*`, $2.8\,\mathrm{s}$ in `RRT*-TO`, and $0.9\,\mathrm{s}$ in `SST*`.

4) Median cost of the final solution (J^{f}) found within the planning budget of $120\,\mathrm{s}$.

D. Benchmark

We conducted our benchmark on a workstation with a CPU AMD EPYC 7502 32-Core Processor @2.50 GHz. All planners use a single core. Each planner was run 20 times on each problem, each time with a different random seed.

Our results are summarized in Table I, where we provide a selection of 16 problems (two for each dynamical system). We also show convergence plots with median and 95% confidence intervals for three representative problems in Fig. 6. The complete results are available on our project webpage.

We summarize the main results as follows:

• `SBPL` has been excluded from the table because it is only readily applicable to the three problems that use *Unicycle 1 v0*. In this setting, it consistently finds a solution in competitive time: $2.1\,\mathrm{s}$ in *Bugtrap*, $0.2\,\mathrm{s}$ in *Kink*, and $0.1\,\mathrm{s}$ in *Park*.

However, due to the limited number of lattice-based primitives, the initial and final costs are rather constant: 36.6 in *Bugtrap*, 22.6 in *Kink*, and 6.2 in *Park* (and higher than the costs achieved by `iDb-A*`).

• `SST*` can find an initial solution quickly in problems with car-like dynamics; however, the quality of the initial solution is poor, especially in the larger, higher-dimensional systems. The convergence is slow—our $120\,\mathrm{s}$ timeout was not sufficient for `SST*` to fully converge in most cases. Notably, in problems involving multirotors, it was unable to find solutions within the time limit for most problems (for instance, we observed a 0% success rate in the dynamics of *Quadrotor v0* and 10% with *Quadrotor v1*). Because `SST*` relies on propagating random control inputs, it is very inefficient for multirotor systems, where random inputs quickly bring the system into unstable configurations. However, in low-dimensional and stable dynamical systems like cars and unicycles, it finds the first solution faster but is clearly outperformed in terms of the cost of initial and final solutions by both `iDb-A*` and `RRT*-TO`.

• `RRT*-TO` can find near-optimal initial solutions in some problems but fails if the geometric initial guess is not close to dynamically feasible motion, which occurs more often in environments with flying robots. Thus, the main drawback is that this approach is incomplete, with success rates below 70% on several problems and complete failures in others. The cost at convergence is often worse than that of `iDb-A*`. In

TABLE I
BENCHMARK WITH SELECTED PROBLEMS. **BOLD** INDICATES THE BEST RESULT.

| #	System	Instance	iDb-A*				SST*				RRT*+TO			
			p	$t^{\text{st}}[s]$	$J^{\text{st}}[s]$	$J^{\text{f}}[s]$	p	$t^{\text{st}}[s]$	$J^{\text{st}}[s]$	$J^{\text{f}}[s]$	p	$t^{\text{st}}[s]$	$J^{\text{st}}[s]$	$J^{\text{f}}[s]$
0	Acrobot	Swing up	**1.0**	**1.3**	**5.2**	4.9	0.6	16.4	5.3	4.6	0.6	2.0	8.1	**4.0**
1	Acrobot	Swing up obstacles v1	**1.0**	1.7	6.4	5.1	0.4	-	-	-	0.9	**1.7**	**6.1**	**3.9**
2	Car with trailer	Kink	**1.0**	1.6	**26.9**	24.4	**1.0**	**0.6**	75.8	63.1	**1.0**	2.0	44.8	**18.1**
3	Car with trailer	Park	**1.0**	0.4	19.7	**4.6**	0.7	1.6	18.4	9.9	0.8	**0.2**	**5.7**	5.3
4	Planar rotor	Hole	0.7	34.2	**3.8**	3.8	0.9	39.8	13.5	8.0	**1.0**	**1.1**	8.1	**3.4**
5	Planar rotor	Bugtrap	**1.0**	12.3	**5.5**	**5.2**	**1.0**	26.4	13.6	8.6	**1.0**	**1.9**	11.6	8.1
6	Rotor pole	Swing up obstacles	**1.0**	**2.1**	**4.0**	**3.9**	0.0	-	-	-	0.8	19.8	5.6	4.1
7	Rotor pole	Small window	**1.0**	**3.2**	**4.5**	**4.5**	0.0	-	-	-	0.5	-	-	-
8	Quadrotor v0	Recovery	**1.0**	**0.9**	**6.2**	**2.6**	0.0	-	-	-	0.0	-	-	-
9	Quadrotor v0	Recovery obstacles	**1.0**	**1.5**	**5.3**	**3.6**	0.0	-	-	-	0.9	8.8	6.9	3.6
10	Quadrotor v1	Obstacle	**1.0**	1.2	**2.7**	**2.4**	0.2	-	-	-	**1.0**	1.5	4.4	3.5
11	Quadrotor v1	Window	**1.0**	1.7	**2.6**	**2.2**	0.3	-	-	-	0.8	21.1	7.4	3.7
12	Unicycle 1 v0	Bugtrap	**1.0**	0.5	**22.3**	21.0	**1.0**	**0.1**	70.5	23.8	**1.0**	1.2	45.1	23.8
13	Unicycle 1 v2	Wall	**1.0**	0.8	**20.8**	18.4	**1.0**	**0.1**	49.7	19.2	0.1	-	-	-
14	Unicycle 2	Bugtrap	**1.0**	1.2	**25.2**	25.0	**1.0**	**0.9**	98.2	49.4	**1.0**	2.8	44.5	28.9
15	Unicycle 2	Park	**1.0**	**0.1**	**5.8**	**5.8**	**1.0**	0.2	29.2	5.9	0.9	**0.1**	6.0	**5.8**
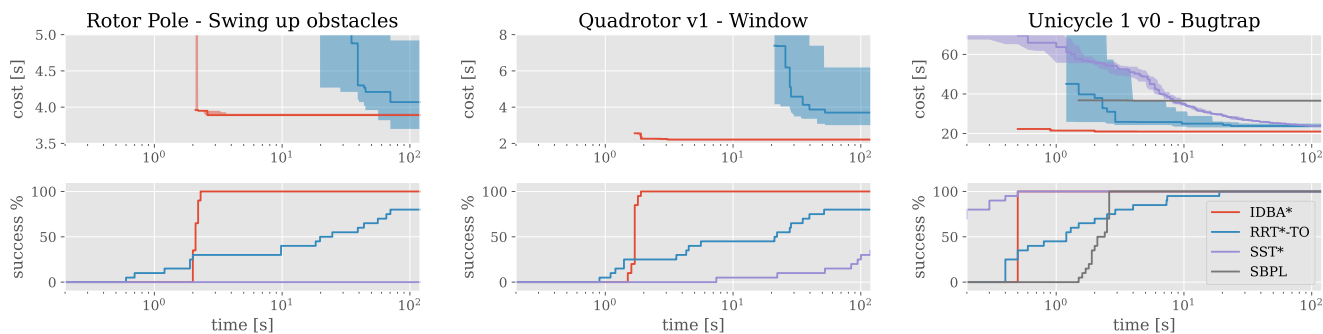
Fig. 6. Success and cost convergence plots for three representative systems. Cost is only plotted when 50% of the runs have found a solution. The shaded region indicates the 95% non-parametric confidence interval for the median.

general, we conclude that RRT*-TO is a good method when the simplified model is informative and when the primary challenge is obstacle avoidance. In these settings, it often matches the time to first solution of iDb-A*.

• iDb-A* finds the highest-quality first solution in 14 out of 16 selected problems (better in the J^{st} column), converged to the lowest-cost solution in 12 out of 16 problems (column J^{f}), and achieved a 100% success rate in 15 out of 16 problems (column p) (and 41 out of 43 total problems). The time to generate the first solution (column t^{st}) is competitive with the other approaches in the problems solved by all methods, while it is the only method that consistently solves all problems with multirotor flying robots.

We can conclude that our method performs well across all systems and environments, from navigation among obstacles with car models to recovery flights with control-limited quadrotors. Note that the performance we report here is considerably better than our previous results [11]. This improvement is due to an improved implementation of the search algorithm, a superior trajectory optimization algorithm and formulation, and a better strategy for choosing the number of primitives and the discontinuity bound.

Fig. 5 shows the different first solutions found by iDb-A*, RRT*-TO, and SST* in the *Unicycle 2 – Bugtrap* problem.

We also display the convergence and success plots for some instructive problems in Fig. 6:

1) *Rotor pole – Swing up obstacles*: This problem involves swinging the pole upwards while avoiding obstacles (Fig. 1c). Only iDb-A* consistently solves the problem within a competitive timeframe. On average, RRT*-TO requires 10x more computational time to find a solution and does not achieve a 100% success rate. The disparate performance across runs of RRT*-TO stems from the uninformative geometric guess, which often leads to failure in the subsequent optimization and thus requires multiple trials with different initial guesses. SST* does not find any solution within the computational budget.

2) *Quadrotor v1 – Window*: In this problem, the quadrotor needs to find a path through a window; see Fig. 4a. SST* achieves a low success rate because propagating random controls is often inefficient, with a low probability of generating useful trajectories. iDb-A* consistently solves the problem in at most two seconds and improves the solution with more compute time. RRT*-TO fails to find a solution in some runs and the median cost is considerably higher.

3) *Unicycle 1 v0 – Bugtrap*: The *Bugtrap* environment

is shown in Fig. 5, but here we use the *Unicycle 1 v0* system instead of *Unicycle 2*. `SST`* finds the first solution the fastest, but convergence to the optimum is slow. `iDb-A`* is also quick and produces a high-quality solution already in the first iteration. `RRT`*`-TO` finds solutions as well but often requires multiple optimization trials to obtain the first valid solution (similar to the previous problems, `RRT`*`-TO` has a high variance in the time required to solve the problem). Regarding `SBPL`, the time to reach the first solution is competitive with the other methods, but due to the limited number of primitives, the solution does not improve.

E. Ablation Studies of iDb-A*

We analyze the main algorithmic components of `iDb-A`* to study the impact on the overall performance and to justify the most important design decisions, namely:

- Scheduling for increasing/decreasing the number of primitives and the discontinuity bound.
- Euclidean heuristic in the search step.
- Joint Optimization of trajectory and free terminal time.
- Optimization-based motion primitives.
- Invariance/equivariance of motion primitives.
- Underlying algorithm for trajectory optimization.

We find a significant interaction between hyperparameters and design decisions, making it challenging to evaluate the impact of each component. For instance, the runtime of the search step with different heuristics is strongly dependent on the number of primitives. Therefore, instead of including more variations of `iDb-A`* in our benchmark, we choose to analyze and discuss each component individually, which we believe provides a clearer understanding. The experiments in the ablation study are conducted on a modern laptop computer with a CPU i7-1165G7@2.80GHz (single core), which has single-core performance similar to that of the workstation used.

1) Time Spent in Each Component: We first evaluate how much computational time is spent in the search or optimization components, and how this varies when we decrease the discontinuity bound and increase the number of primitives.

Fig. 7 shows an analysis of the computation time spent in the search and optimization during the first iteration of `iDb-A`* for three different systems (*Unicycle 1 v0, Unicycle 2*, and *Planar rotor*), using two different values of the discontinuity bound (with a consistent number of motion primitives), in the *Bugtrap* environment (e.g., Fig. 5).

The total time is predominantly occupied by the search component, where both collision checks and nearest-neighbor queries consume a significant fraction of the computational time. When comparing different values of discontinuity bounds (e.g., *u1-0.3* versus *u1-0.2*), we observe that the search is quicker with a larger discontinuity (and a consistently small number of primitives) because it results in a smaller branching factor and fewer states to expand. The computational time of the optimization component remains roughly constant, even when smaller discontinuity bounds provide a better initial guess.
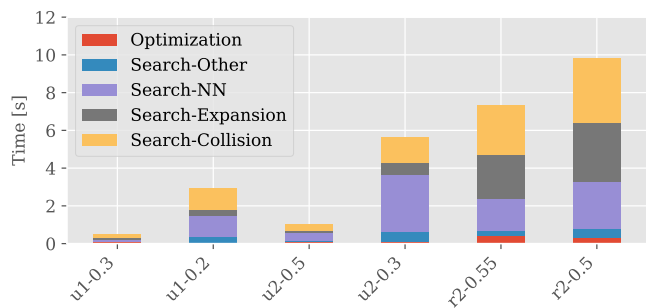

Fig. 7. Time spent in the search and optimization steps during the first iteration of `iDb-A`* in the *Bugtrap* environment, with three different dynamical systems and two different discontinuity bounds. The labels *u1, u2,* and *r2* are short names for the systems *Unicycle 1 v0, Unicycle 2,* and *Planar rotor*. The number after the hyphen indicates the discontinuity bound; for example, *u1-0.3* is *Unicycle 1 v0* with $\delta = 0.3$.

Comparing across systems (e.g., *u1-0.3* versus *r2-0.5*), the time spent in the search increases with the dimensionality of the system's state space, in line with the theoretical exponential complexity. The optimization time also increases, but it remains a minor fraction of the total time.

This analysis offers valuable insights for selecting the initial discontinuity bound and the number of primitives in `iDb-A`*. Ideally, the initial number of primitives and the discontinuity bounds should be chosen to be small (primitives) and large (discontinuity) so that 1) the search finds a discontinuous solution quickly, and 2) the optimizer finds a feasible solution. Thus, our recommendation is to "choose the largest discontinuity bound that the trajectory optimization can handle effectively". To choose the initial number of primitives, given a set of test environments, we recommend starting with a small number of primitives and increasing it until a first valid solution is found in the first iteration of iDb-A* for the easy environments, and after two or three iterations in harder instances—resulting in very competitive anytime behavior.

2) Analysis of Trajectory Optimization with Free Terminal Time: We evaluate our approach for the optimization of trajectories with free terminal time: joint optimization of trajectory and terminal time (*Free-dt*) against three alternative methods described in Section A in the appendix: *(i)* hierarchical time search (*Search-T*), *(ii)* model predictive control (*MPC*), and *(iii)* model predictive contouring control (*MPCC*).

We analyze the computational speed, the success, and the cost value in a set of initial guesses of different discontinuities for some representative problems with different dynamical systems. A summary of the results is shown in Fig. 8 (more extensive results are available on the project webpage).

First, we note that the results highlight a strong variation across the dynamical systems, scenarios, and initial guesses. We can draw the following general conclusions:

1) *Robustness*: *Free-dt* and *Search-T* are more robust and are able to successfully optimize more initial guesses than *MPC* and *MPCC*. On the contrary, *MPC* and *MPCC* are harder to tune and sometimes fail, especially for larger values of the discontinuity bound. The sliding window approaches repair the trajectory locally, step
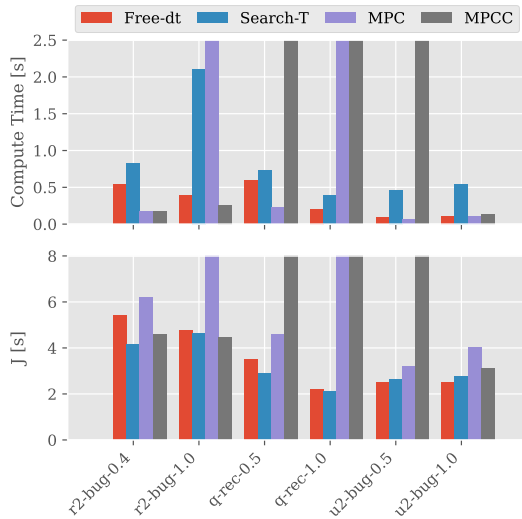
Fig. 8. Analysis of four different strategies for trajectory optimization with free terminal time with six different initial guesses (two per problem). The label *r2-bug* is short for *Planar rotor – Bugtrap*, the label *q-rec* is short for *Quadrotor v0 – Recovery*, and the label *u2-bug* is short for *Unicycle 2 – Bugtrap*. The number after the hyphen indicates the discontinuity bound of each initial guess. A bar reaching the top of the plot indicates the algorithm's failure to find a solution.
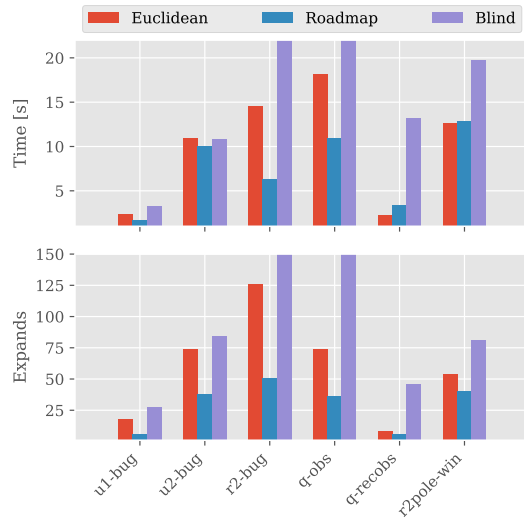

Fig. 9. Analysis of the heuristic function in Db-A* across six different problems: *Unicycle 1 v0 – Bugtrap* (*u1-bug*), *Unicycle 2 – Bugtrap* (*u2-bug*), *Planar rotor – Bugtrap* (*r2-bug*), *Quadrotor v0 – Obstacle* (*q-obs*), *Quadrotor v0 – Recovery obstacles* (*q-recobs*), and *Rotor pole – Window* (*r2pole-win*). The number of expansions is measured in thousands of units; for example, a value of 100 on the graph represents 100,000 expansions. A bar reaching the top of the plot indicates the algorithm's failure to find a solution.

by step, and often cannot reach the final goal if doing so requires jointly improving the initial guess trajectory (where we need to propagate information about the goal across the entire trajectory).

2) *Computation speed*: When *MPC* and *MPCC* manage to find a solution, they are the fastest methods. Comparing *Free-dt* and *Search-T*, *Free-dt* is between 1.5 and 5 times faster than *Search-T*.

3) *Cost convergence*: The joint approaches *Free-dt* and *Search-T* converge to a better cost than the sliding window approaches because they consider the full trajectory at once, but the difference is small.

It is worth noting that compute times for the same problem with different discontinuity bounds are not directly comparable since the initial guesses contain a varying number of steps. Furthermore, these results heavily depend on the underlying optimization algorithm (differential dynamic programming), which effectively addresses the temporal dimension of the trajectory optimization problem (with linear complexity on the number of time steps).

In iDb-A*, the search component consumes more computational time than the optimization process, as illustrated in Fig. 7. In trajectory optimization, we prioritize achieving reliable results and converging to low-cost solutions over computational efficiency. Consequently, *Free-dt* has been selected as the default optimization algorithm for iDb-A*, striking an optimal balance between convergence, robustness, and computational speed.

3) Analysis of Heuristic Functions: We analyze three different heuristic functions to inform the search in Db-A*: the Euclidean heuristic (*Euclidean*), the roadmap heuristic (*Roadmap*), and the blind heuristic (*Blind*) (see Section V).

We report the number of expanded nodes and the computational time for a single search of Db-A* on six different problems involving obstacles in Fig. 9 (with more extensive results on our webpage). We first note that the comparison is highly dependent on the number of motion primitives and the discontinuity bound. For a large discontinuity bound and a small number of primitives, the search's branching factor is small, and the heuristic is relatively irrelevant; even uninformed exploration can find the goal with few expansions. With an increased number of primitives, the potential states to expand grow, necessitating a good heuristic function.

To better illustrate the differences between heuristics, we evaluate Db-A* with smaller discontinuity values (and correspondingly higher numbers of primitives) than those used in the first iteration of iDb-A* in the main benchmark, resulting in many more expansions and computational time.

The *Roadmap* heuristic does not include precomputation time, which requires building a coarse roadmap and computing the optimal cost-to-go for all nodes. This process takes approximately $50\,\mathrm{ms}$ for the unicycles, $70\,\mathrm{ms}$ for the *Planar rotor* (with roadmaps of 1,000 configurations), and $400\,\mathrm{ms}$ for the *Quadrotor v0* (with a roadmap of 3,000 configurations).

We observe that the *Roadmap* heuristic is the most informative (as it considers obstacles) and reduces the number of expanded nodes in all selected problems. However, in some cases, the *Euclidean* heuristic is competitive in terms of compute time (and even faster for some problems), as it is quicker to evaluate because the *Roadmap* requires a k-d tree search with every evaluation. The *Blind* heuristic fails to solve some problems within the time limit of $30\,\mathrm{s}$. Based on these results, in our algorithm, we select the *Euclidean* heuristic because it requires no precomputation or additional hyperparameters,

and it is informative and rapid for all dynamical systems.

4) Motion Primitives - Optimization vs. Sampling: We also evaluated the performance of iDb-A* using motion primitives with random controls (i.e., short trajectories generated by sampling controls uniformly at random within bounds), as opposed to our optimization-based motion primitives.

For car-like robots, the success, convergence, and computation time of iDb-A* are similar with both strategies. The cost of the discontinuity-bounded trajectories produced in the search step using primitives with random controls is often higher, but the optimization step successfully improves these trajectories, achieving a similar cost in the final feasible trajectory. Conversely, motion primitives with random controls often fail to solve problems with flying robots, where such primitives frequently result in unstable final intermediate configurations and erratic trajectories (as exemplified by the performance of SST* with flying robots in the benchmark).

5) Motion Primitives – Invariance and Equivariance: All systems, except for the *Acrobot*, exhibit a form of invariance or equivariance. Using invariance, the same primitive can be transformed on-the-fly (e.g., translated) to be applicable in different states with smaller discontinuity values (Section V-D). For car-like robots, we leverage only translation invariance (Example 4). For flying robots, we utilize translation and linear velocity invariance in the second-order dynamics (Example 5). Since the number of motion primitives required to attain a given coverage resolution grows exponentially with each dimension, exploiting invariance is crucial for planning with a reduced number of primitives. For instance, we use only 2,000 primitives for the *Quadrotor v0* in the first iteration of iDb-A* by utilizing translation and velocity invariance. With only translation invariance, over 50,000 primitives would be required to achieve comparable levels of discontinuity.

6) Choice of the Underlying Algorithm for Trajectory Optimization: Given the central role of the optimization algorithm in our algorithm and baselines, we re-ran the complete benchmark (iDb-A*, RRT*-TO, SST*) with a different underlying trajectory optimization algorithm based on a multiple shooting formulation and sequential quadratic programming [54].

Due to space limitations, the results can be found on our website, with the main conclusions presented here. For iDb-A*, we observe no significant differences between the two optimization methods, especially since the initial guesses are often very informative (collision-free, low dynamics violation, and close to reaching the goal). For the RRT*-TO baseline, multiple shooting performs better in some problems but worse in others, solving 36 with multiple shooting and SQP and 39 problems with differential dynamic programming (with success rate of at least 50%). We conclude that iDb-A* significantly outperforms the baselines in both cases, independent of the optimization algorithm. Using our code, users can switch between algorithms with a simple flag, allowing for easy comparison and experimentation.

F. Limitations and Future Work

From a practical standpoint, the main limitation of iDb-A* is that it necessitates a preliminary offline step to generate motion primitives. Moreover, adding a new dynamical system requires solid theoretical and practical knowledge of two different paradigms in motion planning: search and trajectory optimization, and the interplay between them to choose some important hyperparameters. In this sense, the method is more complex than sample-based motion planners.

The number of required motion primitives grows exponentially with the state dimension. To mitigate this issue, a possible solution is to use a more informative distance metric (instead of the weighted Euclidean metric) when deciding which primitive to apply, which correlates better with the underlying dynamics and the subsequent trajectory optimization. Additionally, more informed sampling strategies for start and goal configurations when generating motion primitives could reduce the number of primitives needed.

To improve the computation time required to find the first solution in some problems (e.g., $1.5\,\mathrm{s}$ in *Quadrotor v0 - Recovery obstacles* or $12.3\,\mathrm{s}$ in *Planar rotor - Bugtrap*) and scale to larger environments with more obstacles, we see great potential in combining our discontinuity-based approach with an RRT-like planner, instead of an incremental A* search. We are also interested in exploring hybrid approaches between our method and the control propagation used in [30].

a) Scalability to Higher-Dimensional Systems: Scalability to higher dimensions is a significant challenge for kinodynamic motion planners. Our benchmark has evaluated a wide range of dynamical systems used in previous general-purpose state-of-the-art kinodynamic motion planners [30], [1].

To scale to higher dimensions, e.g., a quadruped robot or a humanoid robot, we conjecture that we need to tailor some parts of our framework to be system-specific. This can be achieved by using a more informed and user-defined distribution of motion primitives (instead of locally optimal trajectories between states chosen uniformly at random), a more informative distance metric (instead of weighted Euclidean), and learning techniques to adapt motion primitives online to match a given state (instead of only concatenating the primitives together).

X. CONCLUSION

We present iDb-A*, a new kinodynamic motion planner that combines a novel graph-search method with trajectory optimization iteratively. For the graph search, we introduce Db-A*, a generalization of A* that reuses motion primitives to compute trajectories with bounded discontinuity, which are later used as a warm start for trajectory optimization.

iDb-A* amalgamates the ideas and advantages of sampling-based, search-based, and optimization-based kinodynamic motion planners: it converges asymptotically to the optimal solution, rapidly finds a near-optimal solution, and does not require any additional post-processing.

We evaluate iDb-A* on a diverse set of challenging, time-optimal kinodynamic motion planning problems, from obstacle avoidance with car-like robots to highly dynamic maneuvers with quadcopters. iDb-A* consistently outperforms other algorithms on these benchmarks and solves problems that were beyond the capabilities of previous motion planners.

The main limitation is that the number of motion primitives required grows exponentially with the state dimension, which poses a challenge to systems with higher dimensionality.

Finally, we believe that our combination of search, sampling, and optimization lays the foundation for novel kinodynamic planners for robotic manipulation and contact planning.

REFERENCES

[1] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *I. J. Robotics Res.*, vol. 35, no. 5, pp. 528–564, 2016.

[2] Z. Littlefield and K. E. Bekris, "Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 1–9.

[3] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[4] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 2172–2179.

[5] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

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

[7] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "GuSTO: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2019, pp. 6741–6747.

[8] M. Toussaint, "A tutorial on newton methods for constrained trajectory optimization and relations to slam, gaussian process smoothing, optimal control, and probabilistic inference," in *Geometric and Numerical Foundations of Movements*, 2017, vol. 117, pp. 361–392.

[9] W. Li and E. Todorov, "Iterative linear quadratic regulator design for nonlinear biological movement systems," in *ICINCO*, 2004, pp. 222–229.

[10] M. Giftthaler, M. Neunert, M. Stäuble, J. Buchli, and M. Diehl, "A family of iterative gauss-newton shooting methods for nonlinear optimal control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 1–9.

[11] W. Hönig, J. Ortiz-Haro, and M. Toussaint, "db-A*: Discontinuity-bounded search for kinodynamic mobile robot motion planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022.

[12] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 2172–2179.

[13] ——, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*. Munich Germany, 2005, pp. 1–7.

[14] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," *Advances in neural information processing systems*, vol. 16, 2003.

[15] M. Pivtoraiko, "Differentially constrained motion planning with state lattice motion primitives," Ph.D. dissertation, Carnegie Mellon University, 2012.

[16] L. Jarin-Lipschitz, J. Paulos, R. Bjorkman, and V. Kumar, "Dispersion-minimizing motion primitives for search-based motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 12 625–12 631.

[17] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *I. J. Robotics Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[18] T. Kunz and M. Stilman, "Kinodynamic RRTs with fixed time step and best-input extension are not probabilistically complete," in *Int. Worksh. on Alg. Found.*, 2015, pp. 233–244.

[19] K. Hauser and Y. Zhou, "Asymptotically optimal planning by feasible kinodynamic planning in a state-cost space," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1431–1443, 2016.

[20] M. Kleinbort, E. Granados, K. Solovey, R. Bonalli, K. E. Bekris, and D. Halperin, "Refined analysis of asymptotically-optimal kinodynamic planning in the state-cost space," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 6344–6350.

[21] F. Grothe, V. N. Hartmann, A. Orthey, and M. Toussaint, "ST-RRT*: Asymptotically-Optimal Bidirectional Motion Planning through Space-Time," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2022.

[22] J. T. Betts, *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

[23] 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, pp. 25–57, 2006.

[24] P. E. Gill, W. Murray, and M. A. Saunders, "Snopt: An sqp algorithm for large-scale constrained optimization," *SIAM review*, vol. 47, no. 1, pp. 99–131, 2005.

[25] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Acikmese. Convex optimization for trajectory generation.

[26] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An efficient and versatile framework for multi-contact optimal control," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 2536–2542.

[27] 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.

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

[29] B. Sakcak, L. Bascetta, G. Ferretti, and M. Prandini, "Sampling-based optimal kinodynamic planning with motion primitives," *Auton. Robots*, vol. 43, no. 7, pp. 1715–1732, 2019.

[30] R. Shome and L. E. Kavraki, "Asymptotically optimal kinodynamic planning using bundles of edges," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2021, pp. 9988–9994.

[31] S. Choudhury, J. D. Gammell, T. D. Barfoot, S. S. Srinivasa, and S. Scherer, "Regionally accelerated batch informed trees (RABIT*): A framework to integrate local information into optimal path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2016, pp. 4207–4214.

[32] J. Kamat, J. Ortiz-Haro, M. Toussaint, F. T. Pokorny, and A. Orthey, "Bitkomo: Combining sampling and optimization for fast convergence in optimal motion planning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 4492–4497.

[33] A. Perez, R. P. Jr, G. D. Konidaris, L. P. Kaelbling, and T. Lozano-Pérez, "LQR-RRT*: Optimal sampling-based motion planning with automatically derived extension heuristics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2012, pp. 2537–2542.

[34] D. Zheng and P. Tsiotras, "Accelerating kinodynamic RRT* through dimensionality reduction," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 3674–3680.

[35] A. Wu, S. Sadraddini, and R. Tedrake, "R3T: Rapidly-exploring random reachable set tree for optimal kinodynamic planning of nonlinear hybrid systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 4245–4251.

[36] H.-T. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "RL-RRT: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4298–4305, 2019.

[37] B. Ichter and M. Pavone, "Robot motion planning in learned latent spaces," *IEEE Robot. Autom. Lett.*, vol. 4, no. 3, pp. 2407–2414, 2019.

[38] W. Du, S.-K. Kim, O. Salzman, and M. Likhachev, "Escaping local minima in search-based planning using soft duplicate detection," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 2365–2371.

[39] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *I. J. Robotics Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[40] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[41] S. LaValle, "Rapidly-exploring random trees: A new tool for path planning," *Research Report 9811*, 1998.

[42] D. J. Webb and J. v. d. Berg, "Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints," *arXiv preprint arXiv:1205.5088*, 2012.

[43] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor uav on se (3)," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 5420–5425.

[44] B. M. Kim and P. Tsiotras, "Controllers for unicycle-type wheeled robots: Theoretical results and experimental validation," *IEEE Trans. Robot.*, vol. 18, no. 3, pp. 294–307, 2002.

[45] J.-P. Laumond *et al.*, *Robot motion planning and control*. Springer, 1998, vol. 229.

[46] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE, 2011, pp. 2520–2525.

[47] A. Bry, C. Richter, A. Bachrach, and N. Roy, "Aggressive flight of fixed-wing and quadrotor aircraft in dense indoor environments," *I. J. Robotics Res.*, vol. 34, no. 7, pp. 969–1002, 2015.

[48] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 3067–3074.

[49] J. Gonzalez and M. Likhachev, "Search-based planning with provable suboptimality bounds for continuous state spaces," in *Proceedings of the International Symposium on Combinatorial Search*, vol. 2, no. 1, 2011, pp. 60–67.

[50] N. Maray, A. Vemula, and M. Likhachev, "Improved soft duplicate detection in search-based motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE, 2022, pp. 5792–5798.

[51] D. Verscheure, B. Demeulenaere, J. Swevers, J. De Schutter, and M. Diehl, "Time-optimal path tracking for robots: A convex optimization approach," *IEEE Transactions on Automatic Control*, vol. 54, no. 10, pp. 2318–2327, 2009.

[52] H. Pham and Q.-C. Pham, "A new approach to time-optimal path parameterization based on reachability analysis," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 645–659, 2018.

[53] D. H. Jacobson and D. Q. Mayne, *Differential Dynamic Programming*. American Elsevier Publishing Company, Inc, 1970.

[54] A. Jordana, S. Kleff, A. Meduri, J. Carpentier, N. Mansard, and L. Righetti, "Stagewise implementations of sequential quadratic programming for model-predictive control," *Preprint*, 2023.

[55] D. Murray and S. Yakowitz, "Differential dynamic programming and newton's method for discrete optimal control problems," *Journal of Optimization Theory and Applications*, vol. 43, no. 3, pp. 395–414, 1984.

[56] R. Bonalli, A. Cauligi, A. Bylard, and M. Pavone, "Gusto: Guaranteed sequential trajectory optimization via sequential convex programming," in *Proc. IEEE Int. Conf. Robot. Autom.* IEEE, 2019, pp. 6741–6747.

[57] D. Fernández and M. V. Solodov, "Local convergence of exact and inexact augmented lagrangian methods under the second-order sufficient optimality condition," *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 384–407, 2012.

[58] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[59] W. Hönig, J. Ortiz-Haro, and M. Toussaint, "Benchmarking sampling-, search-, and optimization-based approaches for time-optimal kinodynamic mobile robot motion planning," in *Workhsop: Evaluating Motion Planning Performance: Metrics, Tools, Datasets, and Experimental Design Workshop at IROS*, 2022.

[60] E. Granados, A. Sivaramakrishnan, and K. E. Bekris, "Towards benchmarking sampling-based kinodynamic motion planners with ml4kp," in *Workhsop: Evaluating Motion Planning Performance: Metrics, Tools, Datasets, and Experimental Design Workshop at IROS*, 2022.

[61] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[62] R. Tedrake, *Underactuated Robotics*. MIT, 2016. [Online]. Available: https://underactuated.csail.mit.edu

[63] S. Liu, K. Mohta, N. Atanasov, and V. Kumar, "Search-based motion planning for aggressive flight in se(3)," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 2439–2446, 2018.

[64] B. Zhou, F. Gao, L. Wang, C. Liu, and S. Shen, "Robust and efficient quadrotor trajectory generation for fast autonomous flight," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 3529–3536, 2019.

[65] 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, 2012.

[66] J. Ichnowski and R. Alterovitz, "Fast nearest neighbor search in se (3) for sampling-based motion planning," in *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*. Springer, 2015, pp. 197–214.

[67] D. Lam, C. Manzie, and M. Good, "Model predictive contouring control," in *Proc. IEEE Conf. on Decision and Control (CDC)*, 2010, pp. 6137–6142.

Joaquim Ortiz-Haro is a Postdoctoral Fellow at New York University in the Machines in Motion Lab. He received the B.S. degree in Industrial Engineering and the M.S. in Mathematics from the Polytechnic University of Catalonia (Spain), and the Ph.D. degree in Computer Science from Technische Universität Berlin (Germany). His research interests include task motion planning, nonlinear optimization, and deep learning for robotics.

Wolfgang Hönig is an Assistant Professor at TU Berlin, heading the Intelligent Multi-Robot Coordination Lab. Previously, he was an independent junior research group leader at TU Berlin and, before that, a postdoctoral scholar in the Department of Aerospace at the California Institute of Technology. He holds a Ph.D. in Computer Science from the University of Southern California. His research focuses on enabling large teams of physical robots to collaboratively solve real-world tasks using tools from informed search, optimization, and machine learning.

Valentin N. Hartmann is a Postdoctoral Fellow at ETH Zurich in the Computational Robotics Lab. He received the B.S. and M.S. degrees in Mechanical Engineering from ETH Zurich (Switzerland), and the Ph.D. degree in Computer Science from Technische Universität Berlin (Germany). His research interests include task and motion planning for robot teams.

Marc Toussaint is a Professor of Intelligent Systems at TU Berlin. He was previously a Professor of Machine Learning and Robotics at the University of Stuttgart, a Max Planck Fellow at the Max Planck Institute for Intelligent Systems, a visiting scholar at MIT, and for several months, the head of the Core ML Robotics team at Amazon. His research combines planning, optimization, and machine learning to tackle fundamental problems in robotics and physical reasoning.

APPENDIX A
ALTERNATIVE METHODS FOR TRAJECTORY OPTIMIZATION WITH FREE TERMINAL TIME

In this section, we discuss the three alternative methods for the trajectory optimization step of iDb-A*, in addition to the proposed 'joint optimization of trajectory and terminal time'. All methods have been analyzed in the ablation study.

1) Hierarchical Time Search (Search-T): A hierarchical approach that combines a linear search on the terminal time with trajectory optimization with a fixed terminal time. Given the time bounds $\{T_{\min}, T_{\max}\}$ and a time resolution h, we define a set of candidate times $\mathcal{T} = \{T_{\min}, T_{\min}+h, \ldots, T_{\max}\}$, and solve the hierarchical optimization problem:

$$\min_{T_i \in \mathcal{T}} \text{Trajectory\_Optimization}(T_i), \qquad (14)$$

where Trajectory_Optimization(T_i) first rescales temporally the initial guess to have a time duration of T_i (the time step size Δt is kept constant, but the number of time steps of the trajectory varies) and then solves (3) with a fixed number of time steps K_i and fixed Δt. For time-optimal trajectories, we start the search at $T_{\min}$, and stop at the first T_i when Trajectory_Optimization(T_i) is feasible. Based on the duration T_0 of the initial guess, a reasonable choice of the parameters is, e.g., $T_{\min} = 0.5T_0$, $T_{\max} = 2T_0$, and $h = (T_{\max} - T_{\min})/10$.

A. Sliding Window Optimization

Instead of considering the full trajectory at once, the optimization step can try to repair the discontinuities locally. While this approach is more constrained to follow the initial guess, it is also potentially faster. Solving a sequence of smaller subproblems often reduces the computational cost and number of nonlinear iterations, which typically increase for longer trajectories.

The following two approaches are inspired by two optimal control formulations, namely Model Predictive Control (MPC) and Model Predictive Contouring Control (MPCC) [67]. In both approaches, we repair the initial guess trajectory in a sequence of steps, starting from the beginning of the initial guess. At each step, we (i) optimize the sequence of states and controls inside a small optimization window of length W_o (e.g., 50 steps), (ii) fix the first $W_s \leq W_o$ states and controls (e.g., 10 steps), and (iii) move the optimization window by W_s, so that the new start state is the last fixed state. The time step duration Δt is fixed, but the resulting final trajectory might have a different duration than the initial guess from Db-A*.

a) Subgoal Following (MPC): The optimization problem in each step is:

$$\min_{\mathbf{X}_W, \mathbf{U}_W} k_1 d(\mathbf{x}_W, \mathbf{g})^2 + \Delta t \sum_w j(\mathbf{x}_w, \mathbf{u}_w), \qquad (15a)$$

$$\text{s.t.} \quad \text{Constraints (3b), (3c), and (3d).} \qquad (15b)$$

Here, $\mathbf{X}_W, \mathbf{U}_W$ are the sequence of states and controls in the optimization window, $\mathbf{x}_W$ is the last state of the current window, and $\mathbf{g}$ is the subgoal state for this optimization window, chosen from the Db-A* initial guess to encourage making progress in the path. The weight $k_1 > 0$ combines the objective of minimizing the distance to the subgoal $d(\mathbf{x}_W, \mathbf{g})$ with the original control cost function.

b) Path Following (MPCC): The optimization problem in each step is:

$$\min_{\mathbf{X}_W, \mathbf{U}_W, \alpha} - k_1 \alpha + k_2 d(\mathbf{x}_W, \pi(\alpha))^2 +$$
$$\Delta t \sum_w j(\mathbf{x}_w, \mathbf{u}_w), \qquad (16a)$$

$$\text{s.t.} \quad 0 \leq \alpha \leq 1, \qquad (16b)$$

$$\text{Constraints (3b), (3c), and (3d).} \qquad (16c)$$

The function $\pi(\cdot) : [0, 1] \to \mathcal{X}$ is a smooth parameterization of the initial guess (e.g., a spline through the waypoints), and the scalar variable α indicates the progress on the path. The term $k_1 \alpha$, with $k_1 > 0$, tries to maximize the progress along the path. The term $k_2 d(\mathbf{x}_W, \pi(\alpha))^2$, with $k_2 > 0$, minimizes the distance between the last state in the window and the progress on the path. Together, these two terms push the last state $\mathbf{x}_W$ to make progress along the path while following it closely (note that, compared to other MPCC formulations, we only apply the contouring cost to the last state in the window).