2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 14-18, 2024. Abu Dhabi, UAE

iDb-RRT: Sampling-based Kinodynamic Motion Planning with Motion Primitives and Trajectory Optimization

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

***Abstract*— Rapidly-exploring Random Trees (RRT) and its variations have emerged as a robust and efficient tool for finding collision-free paths in robotic systems. However, adding dynamic constraints makes the motion planning problem significantly harder, as it requires solving two-value boundary problems (computationally expensive) or propagating random control inputs (uninformative). Alternatively, Iterative Discontinuity Bounded A* (iDb-A*), introduced in our previous study, combines search and optimization iteratively. The search step connects short trajectories (motion primitives) while allowing a bounded discontinuity between the motion primitives, which is later repaired in the trajectory optimization step.**

Building upon these foundations, in this paper, we present iDb-RRT, a sampling-based kinodynamic motion planning algorithm that combines motion primitives and trajectory optimization within the RRT framework. iDb-RRT is probabilistically complete and can be implemented in forward or bidirectional mode. We have tested our algorithm across a benchmark suite comprising 30 problems, spanning 8 different systems, and shown that iDb-RRT can find solutions up to 10x faster than previous methods, especially in complex scenarios that require long trajectories or involve navigating through narrow passages.

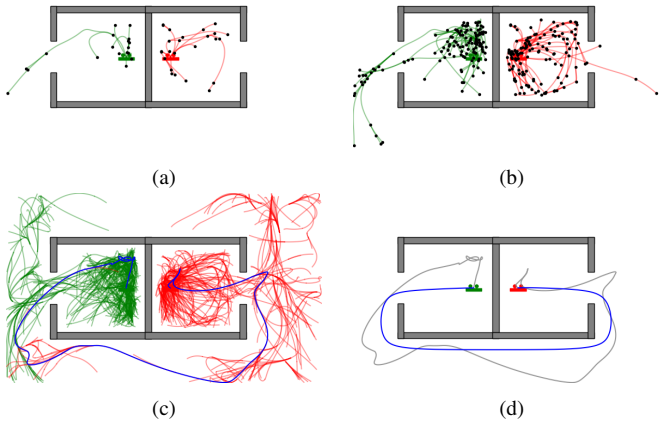


Fig. 1. iDb-RRT combines a forward or bidirectional RRT search with motion primitives (Db-RRT) and trajectory optimization iteratively. (a,b) In the search step, the RRT is expanded by connecting motion primitives with a bounded discontinuity. (c) The output of the RRT is a trajectory with a bounded discontinuity in the dynamics constraints. (d) Using trajectory optimization, we generate a dynamically feasible trajectory. Problem visualization: *Planar Rotor in Double bugtrap*.

I. Introduction

Kinodynamic motion planning is a fundamental problem in robotics where the goal is to find collision-free trajectories in high-dimensional, continuous, and non-convex spaces, while also considering actuation limits and dynamics of the robot. Over the last two decades, a wide variety of sampling-, search-, and optimization-based methods have been proposed to address (kinodynamic) motion planning problems.

A breakthrough was the introduction of Rapidly-exploring Random Trees (RRT) [1], a sampling-based method that incrementally builds a tree of configurations by expanding nodes towards randomly sampled new configurations. RRT-like algorithms (e.g., [2], [3], [4], [5], [6], [7]) are highly efficient for geometric planning, i.e., motion planning settings that involve only joint configurations of the system, since in the geometric setting, two configurations can be connected exactly by using linear interpolation.

Although RRT-like algorithms can be adapted for kinodynamic motion planning (e.g., [8], [9]), their efficiency significantly decreases, as they typically require solving multiple two-point boundary value problems or the propagation of random control inputs. Two-point boundary problems, as they arise for most robotic systems, often do not have an analytic solution, and solving them is computationally expensive, generally requiring the solution of a nonlinear trajectory optimization problem. Propagating random control inputs tends to be uninformative for many systems, as random controls can lead to poor exploration of the state space, particularly in highly nonlinear systems such as quadrotors where random inputs often lead to instability in the system. Further, it is not clear how to perform a bidirectional search, as in RRT-Connect [3], in the kinodynamic setting with the propagation of random inputs.

Alternative approaches for kinodynamic motion planning are optimization-based methods [10], [11], [12], which scale polynomially instead of exponentially but require an initial guess and may fail to converge; and search-based methods [13], [14], which provide strong theoretical guarantees but require a pre-defined discretization of the state or control space. More recently, hybrid methods have been proposed to merge the strengths of the three previous approaches to kinodynamic motion planning [15], [16], [17], [18], [19]. Iterative Discontinuity-Bounded A* (iDb-A*) [20] introduces an approach based on A*-search with *motion primitives*, i.e., short and locally optimal trajectories, that are connected not necessarily exactly, but allowing for a bounded discontinuity (i.e., a nonzero distance) between the last state of a motion

Website: https://quimortiz.github.io/idbrrt/

Code is available at Dynoplan (https://github.com/quimortiz/dynoplan) and Dynobench (https://github.com/quimortiz/dynobench).

[1]Machines in Motion Laboratory, New York University, USA, [2]TU Berlin, Germany, [3]Computational Robotics Lab, ETH Zurich, CH. This work was in part supported by the National Science Foundation grants 1932187, 2026479, 2222815 and 2315396; and in part by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 448549715.

primitive and the first state of the next one. These discontinuities between the motion primitives are later rectified using trajectory optimization (TO). By iteratively combining optimization and search with an increasing number of motion primitives and a reduced discontinuity bound, this method achieves asymptotically optimal motion planning and outperforms state-of-the-art methods across various robotic systems. A primary limitation of iDb-A* is its inefficiency in finding an initial solution, particularly in large environments, where the time required to find the initial solution remains high.

In this paper, we combine the strengths of the exploration of RRT with the concept of discontinuities between motion primitives and trajectory optimization. We present iDb-RRT (iterative Discontinuity-bounded RRT), a new kinodynamic motion planning algorithm that builds on the ideas of allowing discontinuities in an initial motion from iDb-A*, and integrates the RRT exploration strategy with short motion primitives and trajectory optimization. iDb-RRT samples a random configuration, then expands the configuration that is closest using applicable motion primitives with bounded discontinuity. Once a solution is found, we employ trajectory optimization to correct the discontinuities between motion primitives. By incrementally increasing the number of primitives and reducing the allowed discontinuity, our algorithm achieves probabilistic completeness.

We analyze both a forward and a bidirectional version of iDb-RRT. In the open-source benchmark *Dynobench* comprising 30 problems across 8 different systems, iDb-RRT significantly outperforms state-of-the-art methods in initial solution time, especially in complex scenarios requiring long-horizon planning or navigating through narrow passages.

II. Related Work

In this section, we discuss previous work on RRTs for kinodynamic motion planning and methods combining sampling and optimization. A more comprehensive review of methods in kinodynamic motion planning can be found in [20], [21].

Sampling-based methods often grow the search tree towards a randomly sampled configuration by solving two-point boundary value problems [22] to connect two states precisely, or by propagating random control inputs [23]. Previous work has focused on improving the expansion step (also called steering function) for specific systems [8], [24], [25], better exploration by most informative sampling [26], [27], better heuristics [28], better integration of nonlinear solvers as a subroutine in sampling-based planners [6], [29], or using motion primitives [16] in a discretized configuration space. Compared to the previously discussed methods, our approach plans with the full dynamics (with bounded discontinuity), does not require discretization of the workspace, and does not require solving two-point boundary value problems in the RRT expansion step. This is enabled by leveraging precomputed motion primitives and allowing discontinuities in the planning stage, which are later fixed using trajectory optimization (TO).

Leveraging TO is a common approach for both geometric [18], [6] and kinodynamic motion planning, e.g., as a final post-processing step to improve cost and smoothness [30].

In kinodynamic planning, previous work often involves planning using a simplified geometric model [31], [32], [33] and tracking the resulting reference using trajectory optimization or an optimization-based controller. This approach is commonly used in high-dimensional systems, e.g., [34], [35] for UAVs or [36], [37] for legged robots. Unfortunately, initially using a simplified model and accounting for the full dynamics later is limiting and might lead to infeasible optimization problems if the initial guess is not close to a dynamically feasible trajectory [38].

We also use TO for computing the final feasible trajectory, but we plan with the full dynamics (with bounded discontinuity). As this discontinuity can be made arbitrarily low, and optimization and search are combined iteratively, iDb-RRT is probabilistically complete under mild assumptions.

III. Problem Definition

We consider a robot with a continuous state $\mathbf{x} \in \mathcal{X}$ (e.g., $\mathcal{X} \subseteq \mathbb{R}^{d_x}$) and a control vector $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^{d_u}$. 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 control 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), $\mathbf{X} = \langle \mathbf{x}_0, \mathbf{x}_1, \dots, \mathbf{x}_K \rangle$ to denote the sequence of states sampled at times $0, \Delta t, \dots, K\Delta t$ and $\mathbf{U} = \langle \mathbf{u}_0, \mathbf{u}_1, \dots, \mathbf{u}_{K-1} \rangle$ to denote the sequence of controls applied to the system for the time frames $[0, \Delta t), [\Delta t, 2\Delta t), \dots, [(K-1)\Delta t, K\Delta t)$. 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 search problem,

$$\begin{aligned}
&\text{find } \mathbf{U}, \mathbf{X}, K && && \text{(3a)}\\
&\text{s.t. } \mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k) && \forall k \in \{0, \dots, K-1\}\,, && \text{(3b)}\\
&\quad \mathbf{u}_k \in \mathcal{U} && \forall k \in \{0, \dots, K-1\}\,, && \text{(3c)}\\
&\quad \mathbf{x}_k \in \mathcal{X}_{\text{free}} \subseteq \mathcal{X} && \forall k \in \{0, \dots, K\}\,, && \text{(3d)}\\
&\quad \mathbf{x}_0 = \mathbf{x}_s;\ \mathbf{x}_K = \mathbf{x}_g\,. && && \text{(3e)}
\end{aligned}$$

In this paper we focus on finding a valid trajectory quickly (i.e, very little compute time), as opposed to finding the optimal solution. Although there is no explicit minimization of a cost function in our algorithms, we can evaluate the cost of the trajectory a posteriori. We use the cost term $J(\mathbf{U}, \mathbf{X}) = \sum_{k=0}^{K-1} j(\mathbf{u}_k, \mathbf{x}_k)\,\Delta t$, with $j(\mathbf{u}_k, \mathbf{x}_k) = 1$ for minimal time (span) (alternatively, one might use $j(\mathbf{u}_k, \mathbf{x}_k) = \|\mathbf{u}_k\|^2$ for minimal control effort). 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.

IV. iDb-RRT

A. Background

Our approach relies on two concepts, that we now define.

Definition 1 (Discontinuity Bounded Solution): A trajectory $\mathbf{X} = (\mathbf{x}_0, \ldots, \mathbf{x}_K), \mathbf{U} = (\mathbf{u}_0, \ldots, \mathbf{u}_{K-1})$ is a δ-discontinuity bounded solution of the kinodynamic motion planning problem Eq. (3) if: $d(\mathbf{x}_{k+1}, \text{step}(\mathbf{x}_k, \mathbf{u}_k)) \leq \delta$, $d(\mathbf{x}_0, \mathbf{x}_s) \leq \delta$, $d(\mathbf{x}_K, \mathbf{x}_g) \leq \delta$, $\mathbf{x}_k \in \mathcal{X}_{\text{free}}$ and $\mathbf{u}_k \in \mathcal{U}$, where $d(\cdot, \cdot)$ is a distance function, e.g., a weighted Euclidean norm, and $\delta \geq 0$.

The search step of our algorithm, Db-RRT, generates solutions that are discontinuity bounded, while the trajectory optimization step rectifies these solutions to satisfy Eq. (3).

Definition 2 (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.

A large set of motion primitives can be generated offline by sampling random start and goal states, and attempting to connect them using nonlinear trajectory optimization algorithms. This results in a superior distribution of primitives in terms of coverage of the state space, compared to propagating random control inputs, and it guarantees asymptotic coverage of the state space [20]. Importantly, we can later use known properties of the system to adapt primitives on-the-fly to match a state during the search, e.g., by using translation invariance of mobile robots, we can *translate* a primitive to match the position components of the state space [20]. Fig. 2 displays four motion primitives in the system *planar rotor* and how they can be connected with a bounded discontinuity.

B. Overview

Our approach is summarized in Alg. 1. We assume that a large set of motion primitives $\mathcal{M}_{\text{L}}$ has been precomputed and is available before planning. iDb-RRT iteratively runs two steps until the first valid solution is found:

- An RRT search algorithm that connects motion primitives with bounded discontinuity, called Db-RRT. The output is a discontinuity bounded solution, i.e., a collision-free trajectory with bounded violation of dynamic constraints (Definition 1).

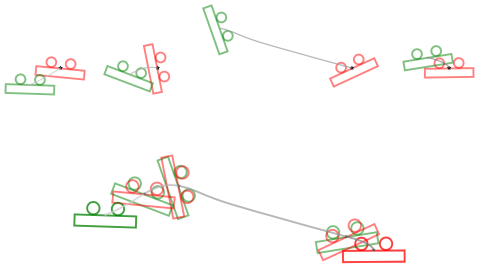


Fig. 2. *Top*: Four motion primitives in the system *Planar rotor*. The initial state (green), final state (red) and duration are randomized. *Bottom*: During the search step (Db-RRT), motion primitives are connected allowing for a bounded discontinuity. In this visualization, we connect these four motion primitives from left to right. The green and red configurations indicate the first and last configurations of each primitive. Note that their rotation component does not match exactly (further, discontinuities in the velocity components are not shown).

Algorithm 1: iDb-RRT – Iterative Discontinuity Bounded RRT

Input: $\mathbf{x}_s, \mathbf{x}_g, \text{step}, \mathcal{X}_{\text{free}}, \mathcal{U}, \mathcal{M}_{\text{L}}$
Result: $\mathbf{X}, \mathbf{U}$
$\delta \leftarrow \delta_0$ ▷ *Choose initial discontinuity bound*
$\mathcal{M} \leftarrow$ `ChoosePrimitives`$(\mathcal{M}_{\text{L}})$ ▷ *Choose initial subset of primitives from* $\mathcal{M}_{\text{L}}$
while not found **do**
  $\mathbf{X}_d, \mathbf{U}_d \leftarrow$ `db-RRT` $(\mathbf{x}_s, \mathbf{x}_g, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta)$
  if $\mathbf{X}_d, \mathbf{U}_d$ successfully computed **then**
    $\mathbf{X}, \mathbf{U} \leftarrow$ `Optimization` $(\mathbf{X}_d, \mathbf{U}_d, \mathbf{x}_s, \mathbf{x}_g, \text{step}, \mathcal{X}_{\text{free}}, \mathcal{U})$
    if $\mathbf{X}, \mathbf{U}$ successfully computed **then**
      `Return` $(\mathbf{X}, \mathbf{U})$ ▷ *New solution found*
    else
      $\delta \leftarrow$ `DecreaseDelta`(δ)
  else
    $\delta \leftarrow$ `DecreaseDelta`(δ)
    $\mathcal{M} \leftarrow$ `IncreasePrimitives`$(\mathcal{M}, \mathcal{M}_{\text{L}})$

- Gradient-based trajectory optimization, which attempts to repair the discontinuities between the motion primitives to produce a dynamically feasible trajectory.

If the search fails to find a solution within a given timeout (TerminateCondition), we increase the number of available motion primitives. If gradient-based optimization fails, we reduce the allowed discontinuity. In practice, we typically require only one or two outer iterations (that is, a call to the search and optimization algorithms) to find a solution. We decrease the allowed discontinuity following a geometric sequence, $d_i = d_{i-1} \cdot d_r$ with a fixed rate $d_r < 1$, and increase the number of primitives also following a geometric sequence $m_i = m_{i-1} \cdot m_r$ with a fixed rate $m_r > 1$.

C. Db-RRT: RRT with Motion Primitives

Db-RRT is an RRT algorithm that connects motion primitives with bounded discontinuity, following the general RRT algorithm to choose the next state to expand. This approach provides a Voronoi bias (i.e., nodes at the frontier of the search tree are more likely to get expanded), thus rapidly exploring the feasible state space. In Alg. 2, we describe our Db-RRT algorithm and highlight our modifications from

Algorithm 2: Db-RRT – Rapidly-Exploring Random Trees with Motion Primitives

Input: $\mathbf{x}_s, \mathbf{x}_g, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta$
Result: $\mathbf{X}_d, \mathbf{U}_d$
1 $\mathcal{T} \leftarrow \texttt{AddNode}(\mathbf{x}_s)$
2 **while** $\neg\texttt{TerminateCondition}()$ **do**
3 **if** rand() $<$ goalBias **then**
4 $\mathbf{x}_{\text{rand}} \leftarrow \mathbf{x}_g$
5 **else**
6 $\mathbf{x}_{\text{rand}} \leftarrow \texttt{Sample}(\mathcal{X}_{\text{free}})$
7 $\mathbf{x}_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, \mathbf{x}_{\text{rand}})$
8 $\mathbf{x}_{\text{new}}, m \leftarrow \texttt{ExpandDb}(\mathbf{x}_{\text{nearest}}, \mathbf{x}_{\text{rand}}, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta)$
9 **if** $\mathbf{x}_{\text{new}} \neq$ NULL **then**
10 **if** d$(\mathbf{x}_{\text{new}}, \mathbf{x}_g) < \delta$ **then**
11 $\mathcal{T} \leftarrow \texttt{AddNode}(\mathcal{T}, \mathbf{x}_{\text{new}})$
12 $\mathbf{X}_d, \mathbf{U}_d \leftarrow \texttt{TracebackTrajectory}(\mathcal{T}, \mathbf{x}_{\text{new}})$
13 $\texttt{Return}\ (\mathbf{X}_d, \mathbf{U}_d)$ ▷ *Discontinuity bounded solution*
14 **else if** $\texttt{NearestDistance}\ (\mathcal{T}, \mathbf{x}_{\text{new}}) > \delta$ **then**
15 $\mathcal{T} \leftarrow \texttt{AddNode}(\mathcal{T}, \mathbf{x}_{\text{new}})$

Algorithm 3: Expand-Db: Focused

Input: $\mathbf{x}_o, \mathbf{x}_t, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta$
Result: $\mathbf{x}, m$
1 $\mathcal{M}_c \leftarrow \texttt{NearestR}(\mathbf{x}_o, \mathcal{M}, \delta)$ ▷ *Get applicable primitives*
2 $m_{\text{b}} = NULL, \quad d_{\text{b}} = \infty$
3 **for** $m \in \mathcal{M}_c$ **do**
4 **if** $m \in \mathcal{X}_{\text{free}}$ and $d(m.\mathbf{x}_f, \mathbf{x}_t) < d_{\text{b}}$ **then**
5 $m_{\text{b}} \leftarrow m, \quad d_{\text{b}} \leftarrow d(m.\mathbf{x}_f, \mathbf{x}_t)$
6 **if** $m_{\text{b}} \neq NULL$ **then**
7 $\texttt{Return}\ m_b.\mathbf{x}_f, m_b$
8 $\texttt{Return}\ NULL, NULL$

Algorithm 4: Expand-Db: Randomized

Input: $\mathbf{x}_o, \mathbf{x}_t, \mathcal{X}_{\text{free}}, \mathcal{M}, \delta$
Result: $\mathbf{x}, m$
1 $\mathcal{M}_c \leftarrow \texttt{NearestR}(\mathbf{x}_o, \mathcal{M}, \delta)$; ▷ *Get applicable primitives*
2 **for** $m \in$ RandomPermutation$(\mathcal{M}_c)$ **do**
3 **if** $m \in \mathcal{X}_{\text{free}}$ **then**
4 $\texttt{Return}\ m.\mathbf{x}_f, m$
5 $\texttt{Return}\ NULL, NULL$

RRT. In Db-RRT, the expansion operation is performed using motion primitives that are applicable with bounded discontinuity. Given the state $\mathbf{x}_{\text{near}}$, we assess which primitives are applicable (e.g., Line 1 in Alg. 3). We then differentiate between focused expansion (Alg. 3), where we select the primitive that is collision-free ($m \in \mathcal{X}_{\text{free}}$) and brings us closest to $\mathbf{x}_{\text{rand}}$ from a finite number of nearby candidates, and uninformed expansion (Alg. 4), where we choose one collision-free primitive at random. With a small probability (the so-called goal bias), we expand towards the goal state instead of a random state.

We stop when we find a state that is within a distance lower than δ of the goal state. Further, the value of δ is also used to avoid creating nodes in the tree that are too close to previously discovered nodes.

Both expansion strategies are guaranteed to find a solution, if one exists, given sufficient compute time. The inherent trade-off is that Alg. 3 requires more compute time, as it involves evaluating collisions for multiple motion primitives, but it provides a more focused and uniform expansion. In our implementation, we utilize Alg. 3 for expansions towards the goal and Alg. 4 for expansions towards random nodes, but any combination of these two approaches is valid. Focused and uninformed expansion are analogous to guided Monte-Carlo and Monte-Carlo propagation in classic RRT literature, but in Db-RRT we use motion primitives instead of randomly sampled controls.

D. Db-RRT-Connect and other Db-RRT variants

The expansion step of Db-RRT (Algs. 3 and 4) can be integrated with many of the variations and enhancements of RRT that have been previously proposed,

a) Backward and Bidirectional Search: Inspired by RRT-Connect [3], we present a bidirectional variant of Db-RRT, where we grow two trees, one from the start (using standard motion primitives) and one from the goal (using reversed motion primitives), and attempt to connect them. The expansion step in a backward search mirrors that of a forward search but requires reversing the order of states and controls in the motion primitives beforehand. The two trees are connected if two of their states are within the discontinuity bound.

b) Asymptotically Optimal Algorithms: Db-RRT can also be applied to RRT variants that require connecting two states precisely, instead of only expanding the state towards random targets. The discontinuity bound δ can be leveraged to consider two states as equivalent—thereby enabling their exact connection in any rewiring step, such as in RRT* [2]. Such rewiring steps, which are essential for the asymptotic optimality of RRT* and its variants, are already implemented in iDb-A* [20].

E. Trajectory Optimization

The output of Db-RRT is a sequence of states and controls that connects the start and goal states with a bounded discontinuity, see Definition 1. In the optimization step of iDb-RRT, we employ nonlinear trajectory optimization to repair the discontinuity between the motion primitives and to obtain a feasible and locally optimal trajectory.

For gradient-based trajectory optimization, we require the gradients of the dynamics and the cost function with respect to the states and controls. These can be easily obtained for most robotics systems using finite differences, analytic expressions, or a differentiable simulator. Instead of the binary collision check in Db-RRT, we now use a signed distance function.

In the trajectory optimization step, the number of time steps and the time duration of the trajectory is fixed to the output of Db-RRT. If desired, we can also optimize the duration of the trajectory by including the length of the time interval in the optimization problem or using other techniques, as explored in [20]. However, because the goal of iDB-RRT is to find a valid trajectory quickly, we choose not to include the time interval as an optimization variable (further, note that because trajectories from RRT-like algorithms tend to be suboptimal, the time duration of the initial guess is often sufficient to reach the goal).

To solve the trajectory optimization problem, we use the Differential Dynamic Programming (DDP) algorithm, which is a second-order method for solving optimal control problems of the form Eq. (4). Collision and goal constraints, and state and control bounds of the original kinodynamic motion planning problem are added to the cost (4a) with a squared penalty method and a max activation function for inequalities. Further, we include small regularization terms on the control effort and the acceleration of the system to improve convergence.

$$\min_{\mathbf{X},\mathbf{U}} \sum_{k=0}^{K-1} c(\mathbf{x}_k, \mathbf{u}_k) + c_K(\mathbf{x}_K)\,, \tag{4a}$$

$$\text{s.t.} \quad \mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k) \quad \forall k \in \{0, \dots, K-1\}\,, \tag{4b}$$

$$\mathbf{x}_0 = \mathbf{x}_s\,. \tag{4c}$$

In particular, we use the optimization algorithm *Feasibility-driven DDP* [39], which can be warm-started with an infeasible sequence of states and controls, providing a good balance between local convergence and globalization.

F. Analysis

The RRT algorithm is probabilistically complete [22], [40], that is, the probability of eventually finding a solution, if one exists, converges to one. The proof assumes that the planning problem is δ_1-robust (informally: the solution should not require traversing a "gap" smaller than δ_1) and that the dynamics are Lipschitz continuous. Formally, it uses an inductive argument over overlapping balls that cover the solution trajectory, demonstrating that the probability of finding an edge between neighboring balls is non-zero.

We first consider Db-RRT (Algorithm 2) with the precondition that we have a sufficiently large set of motion primitives $\mathcal{M}_\text{L}$ and a discontinuity bound $\delta < \delta_1$. Then, the additional if-condition in Line 14 does not prevent finding a solution. Line 8 changes the distribution for the expansion operation but continues to assign a positive probability density to all successors for large sets of randomly generated $\mathcal{M}_\text{L}$. Next, we consider iDb-RRT (Algorithm 1). If Db-RRT fails to find a solution because at least one precondition is violated (a large $\mathcal{M}_\text{L}$ and $\delta < \delta_1$), we adjust both parameters and repeat (Lines 12 to 13), yielding a non-zero probability of executing Db-RRT with parameters that fulfill our assumptions. Finally, we assume that there exists a δ such that if Db-RRT generates a δ-discontinuity bounded solution, the trajectory optimization algorithm will converge with a non-zero probability, which makes our algorithm, iDb-RRT, probabilistically complete.

In practice, we demonstrate that we can use a large discontinuity δ and a small number of primitives to efficiently find solutions to a wide range of problems.

V. Experiments

We evaluate iDb-RRT on 30 problems that include 8 different dynamical systems in various environments. The first 16 problems are inspired by previous work on kinodynamic motion planning [26], [23], [41], [42] (*selected problems* in [20], first 16 rows in Table I). Furthermore, we include 14 additional problems with the same dynamical systems but in larger, more complex environments with more obstacles, which require longer trajectories (last 14 rows in Table I).

All benchmark problems are available in *Dynobench*. It provides a C++ implementation of the dynamical systems (dynamics with analytical Jacobians, state, and bound constraints), collision and signed distance function (based on the Flexible Collision Library, FCL), the environments (in human-friendly YAML files), and visualization tools.

Implementations of iDb-RRT and the other planners are available in *Dynoplan*, including the motion primitives and instructions to replicate the benchmark results. Visualizations of the problems and examples of solution trajectories computed by our algorithm are available on our website.

A. Dynamical Systems

We include a diverse range of dynamical systems and environments, featuring varying state dimensionality (from 3 to 14), the number of underactuated degrees of freedom, and controllability. All systems use explicit Euler integration (2), with $\Delta t = 0.1\,\text{s}$ for all car-like robots and $\Delta t = 0.01\,\text{s}$ for the flying robots and the *Acrobot*.

The 8 systems are (see [20] for a detailed explanation): *Unicycle 1 (1^{st} order)*: 3-dimensional state space and 2-dimensional control space. *Unicycle 2 (2^{nd} order)*: 5-dimensional state space and 2-dimensional acceleration control. *Car with trailer*: 4-dimensional state space and a 2-dimensional control space, *Acrobot*: 4-dimensional state space and 1-dimensional control space. *Quadrotor v0*: 13-dimensional state space and a 4-dimensional control space (force for each of the four motors)[1]. *Quadrotor v1*: The state space is the same as in *Quadrotor v0*, but controls are now the total thrust and torques in the body frame. *Planar rotor*: 6-dimensional state space and 2-dimensional control space, also with 1.3 thrust-to-weight ratio. *Rotor pole*: 2-dimensional control space and 8-dimensional state space.

B. Metrics

Each experiment is run 20 times with different random seeds on a desktop computer[2], single-core. We report:

- $t[s]$: Compute time to get the first solution (median).
- $c[s]$: Cost of the first solution. As a cost, we use the duration of the found trajectory, in seconds (median).

If all the runs of an algorithm fail to find a solution before the timeout of 60 s, we use a dash ('-') in the table. If less than 50% of the runs find a solution, we report the best value but add an asterisk ('*') to indicate a low success rate.

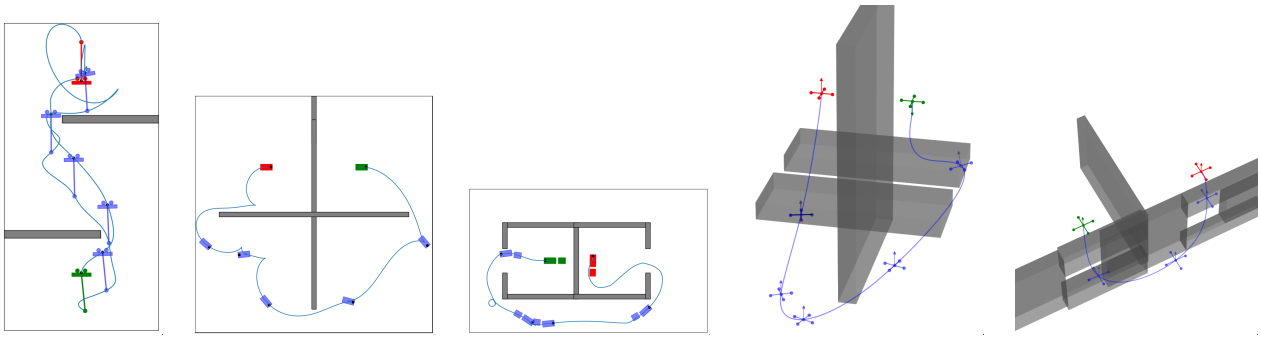
C. Algorithms

We analyze two variants of the iDb-RRT family (Alg. 1):

- `iDb-RRT-F`: using a forward Db-RRT (Alg. 2).
- `iDb-RRT-C`: using a bidirectional Db-RRT inspired by RRT-Connect.

[1]We use the parameters of the Crazyflie 2.1, where the low thrust-to-weight ratio of 1.3 is very challenging for kinodynamic motion planning.

[2]Intel(R) Xeon(R) W-2145 CPU @ 3.70GHz



(a) (b) (c) (d) (e)

Fig. 3. Five kinodynamic motion planning problems in our benchmark *Dynobench*, with a solution found by `iDb-RRT-C`. (a) *Rotor Pole - Up obstacles 2* (b) *Unicycle 2 - Narrow passage* (c) *Car with Trailer - Double bugtrap* (d) *Quadrotor v0 - Recovery obstacles 2* (e) *Quadrotor v1 - Double window*.

We compare our algorithms against state-of-the-art methods that use optimization, search, and sampling, and have available open-source implementations.

• For a sampling-based approach, we use the kinodynamic version of RRT implemented in OMPL [43] (Open Motion Planning Library), which uses the propagation of random control inputs to grow the search tree. Since sampling-based kinodynamic approaches cannot reach a goal state exactly, we use a goal region using the same value of δ used in iDb-A* and iDb-RRT. We denote this algorithm as `Kino-RRT`.

• For optimization-based planning, we choose a standard combination of a geometric motion planner and a trajectory optimizer, which we denote as `Geo-RRT-TO`. Specifically, we use a *geometric* RRT (using the implementation in OMPL) to plan using only the position and orientation of the system, without considering velocity and dynamics. The trajectory optimizer (also based on Feasibility-driven DDP [39], see [20] for details) is warm-started with the geometric guess. If trajectory optimization fails, we run RRT again from scratch and repeat.

• `iDb-A*` is a hybrid method that integrates search with motion primitives and trajectory optimization, but uses incremental A*-searches instead of RRT. Notably, `iDb-A*` has been designed to combine asymptotic optimality with good anytime behavior, as it starts with a small number of motion primitives and incrementally increases the number of available motion primitives during each A*-search. We terminate the algorithm once the first solution is found.

In all algorithms, all hyperparameters are chosen per dynamical system.

D. Results – Comparison with Baselines

Results are summarized in Table I. Due to space constraints, we report only the median of each metric. A graphical representation of these results using boxplots is available on our website. In general, we observe that `iDb-A*` has lower variance than `iDb-RRT`, `Kino-RRT`, and `Geo-RRT-TO`. `iDb-RRT-F` and `iDb-RRT-C` solve all problems with a success rate of 100% (except two problems each, where they achieve 80-90% success rate), outperforming all baseline algorithms in terms of compute time to generate a solution (e.g., `iDb-RRT-C` is the fastest in 19 problems, and `iDb-RRT-F` is the fastest in 6 problems).

- `Kino-RRT`: it finds a first solution in low-dimensional car-like systems in a competitive timeframe (but slower than `iDb-RRT-F` in 13 out of 19 cases) with a higher average cost. However, in agile systems (e.g., flying robots), propagation of random control inputs is very inefficient, and `Kino-RRT` fails to find a solution in 11 problems out of 30.
- `Geo-RRT-TO` often requires multiple runs of RRT to provide a suitable initial guess for trajectory optimization, and sometimes fails completely as the initial guesses never contain information about the dynamics of the system (solving only 18 out of 30 problems with a success rate above 50%). If the initial guess works for the optimizer, it can be very fast (`Geo-RRT-TO` is faster than `iDb-RRT-F` in 7 problems).
- `iDb-A*`: is the strongest baseline, with success rate of 100% in all problems except *Planar rotor/Double bugtrap*. However, `iDb-A*` is always outperformed in the time to find the first solution by `iDb-RRT-C`. The difference between `iDb-A*` and `iDb-RRT-C` increases in the new benchmark (last 14 problems), which require longer plans, with improvements up to 10-20x. On the other hand, the first solution found with `iDb-A*` has a better cost than any other algorithm in 23 cases.

E. Discussion

a) Forward vs Bidirectional Search: Comparing our two variants, we observe that `iDb-RRT-C` is better in 21 out of 30 problems in terms of compute time. These results agree with previous experiments in the RRT literature, where RRT-Connect is generally faster than a forward search (in robotics problems, starting a search from the start and the goal is often beneficial because these configurations are often close to obstacles and narrow passages).

b) Number of primitives and discontinuity bound: Connecting primitives with discontinuities allows our algorithms to plan using a reduced number of primitives. As a reference, for the system *Unicycle 1 v0*, we use an initial set of 200 primitives and an initial discontinuity bound of 0.3. The discontinuity is computed with a weighted Euclidean norm (e.g., weight 1 for position and 0.5 for orientation); thus a δ of 0.3 could represent up to 30 cm of discontinuity in

TABLE I
Median initial solution time (t) and median initial cost (c) for the benchmarked systems and algorithms.

Problem	iDb-RRT-F		iDb-RRT-C		Geo-RRT-TO		iDb-A*		Kino-RRT	
	t [s]	c [s]	t [s]	c [s]	t [s]	c [s]	t [s]	c [s]	t [s]	c [s]
Acrobot/Swing up	0.35	5.39	**0.25**	5.95	0.82	**4.21**	1.49	5.53	0.32	6.83
Acrobot/Swing up obstacles v1	0.36	5.37	**0.18**	**4.86**	0.80	5.06	1.92	5.80	0.38	6.19
Car with trailer/Kink	0.23	53.05	0.24	60.85	0.59	34.45	1.29	**31.10**	**0.20**	68.50
Car with trailer/Park	0.10	10.85	**0.05**	14.00	0.10	**5.05**	0.11	17.90	**0.05**	8.15
Planar rotor/Hole	**0.56**	8.88	1.00	10.93	8.63	5.47	11.77	**3.49**	3.04*	5.99*
Planar rotor/Bugtrap	1.44	9.97	**1.04**	10.48	0.46*	7.84*	12.79	**5.17**	39.23	10.55
Rotor pole/Swing up obstacles	1.91	8.20	**1.14**	8.38	10.70	6.09	2.96	**3.98**	-	-
Rotor pole/Small window	3.84	9.43	**1.14**	9.30	6.21*	2.99*	4.39	**4.54**	-	-
Quadrotor v0/Recovery	0.83	5.61	**0.71**	5.25	1.12	**2.53**	1.32	5.57	-	-
Quadrotor v0/Recovery obstacles	1.29	6.41	1.37	6.20	**0.71**	**3.90**	1.53	5.72	-	-
Quadrotor v1/Obstacle	0.87	6.00	1.36	7.03	**0.25**	**2.72**	2.53	4.54	40.68*	4.90*
Quadrotor v1/Window	**0.61**	5.22	0.88	7.99	9.05	5.53	1.64	**3.71**	9.83*	10.08*
Unicycle 1 v0/Bugtrap	0.13	33.05	**0.11**	30.45	0.40	40.35	0.52	**22.20**	0.14	70.30
Unicycle 1 v2/Wall	0.09	30.70	**0.04**	31.95	0.91	24.30	0.94	**19.60**	0.24	49.45
Unicycle 2/Bugtrap	0.16	59.65	**0.09**	56.35	0.61	43.50	1.65	**25.30**	0.18	69.25
Unicycle 2/Park	0.03	12.20	**0.01**	9.85	0.12	6.15	**0.01**	**5.80**	0.05	13.20
Car with trailer/Double bugtrap	0.70	93.90	**0.65**	101.60	2.44*	53.00*	1.71	**46.80**	3.63	96.65
Car with trailer/Narrow passage	**0.57**	122.55	0.62	132.05	0.82*	74.50*	8.61	**53.90**	2.33	136.25
Planar rotor/Recovery obstacles 2	**0.48**	10.75	0.52	10.95	6.54*	8.94*	20.39	**6.04**	-	-
Planar rotor/Double bugtrap	1.97	14.05	**1.84**	**13.78**	-	-	-	-	19.49*	10.30*
Rotor pole/Up obstacles 2	4.94	10.68	**2.11**	12.15	-	-	14.80	**5.00**	-	-
Rotor pole/Small window 2	3.87	11.59	**1.87**	11.91	0.32*	5.71*	11.84	**6.18**	-	-
Quadrotor v0/Double bugtrap 3D	5.80	11.87	**4.43**	13.19	-	-	23.42	**6.36**	-	-
Quadrotor v0/Recovery obstacles 2	**2.50**	9.74	2.58	10.26	0.33*	3.90*	6.49	**6.41**	-	-
Quadrotor v1/Recovery obstacles 2	2.50	9.71	**2.38**	9.68	0.30*	3.72*	59.71	**6.23**	-	-
Quadrotor v1/Double Window	**2.18**	7.79	2.54	10.86	0.42*	4.62*	25.74	**5.09**	-	-
Unicycle 1 v0/Double bugtrap	0.23	60.10	0.21	60.10	1.70	64.75	0.92	**30.10**	**0.17**	109.30
Unicycle 1 v0/Narrow passage	0.22	81.40	**0.17**	90.35	1.14	83.90	1.88	**37.30**	0.20	133.55
Unicycle 2/Double bugtrap	0.30	94.45	0.28	90.40	2.21	70.85	2.37	**34.60**	**0.25**	103.30
Unicycle 2/Narrow passage	0.31	122.50	**0.24**	118.20	1.11	84.30	7.30	**42.70**	0.34	124.95

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 it can be efficiently repaired in the nonlinear trajectory optimization step of iDb-RRT. For the *Quadcopter v0*, we use 5000 primitives and a discontinuity bound of 0.35, and for the *Rotor pole*, we use 8000 motions and a discontinuity bound of 0.45. The time spent to generate one motion primitive (offline) ranges from 10ms for car-like robots to up to 5s for flying robots (most of the time is spent attempting to solve two-point boundary value problems that do not have a solution).

c) Analysis of compute time in iDb-RRT: In iDb-RRT, the time spent in trajectory optimization dominates the total compute time. For instance, the compute time required to optimize one trajectory in the new benchmark problems with flying robots is between 1s and 3s, while in car-like robots is between 50ms and 200ms. In addition, in the systems *Quadrotor v0* and *Quadrotor v1*, trajectory optimization may fail at the first attempt, and finding a feasible solution requires multiple iterations of iDb-RRT. For car-like systems, we can compute trajectories of duration up to 50s in less than 1s. For flying robots, we require between 0.5s and 4s to generate trajectories of duration up to 14s. A straightforward way to speed up the trajectory optimization step is to reduce the time discretization from 0.01s to 0.05s (with an expected 5x speedup).

d) RRT is easier to tune than incremental A:* The running time of A* with motion primitives in a continuous space is highly sensitive to the number of motion primitives, i.e., the discretization level. With too few primitives, the problem becomes unsolvable; with too many, the state space to be expanded becomes unmanageably large. Conversely, our iDb-RRT algorithms lack an explicit notion of a branching factor. As confirmed by our results, the RRT approach naturally adapts to efficiently solving both simple and complex problems alike, obviating the need for choosing a branching factor while also providing faster exploration.

e) Limitations and future work: The main limitation of iDb-RRT, similar to iDb-A*, lies in its scalability to higher-dimensional systems. As the dimensionality increases, the number of motion primitives required to cover the state space with a small discontinuity grows exponentially. This issue can be partially mitigated by planning with larger discontinuities. In our benchmark, we successfully scaled to 13-DOF for the *Quadrotor* and 8-DOF for *Rotor pole*, thanks to leveraging translation invariance and the second-order linear velocity invariance of the dynamics. To effectively scale to higher dimensions, we see great potential in using function approximation to learn a more informative distance metric and to combine motion primitives with deep generative models or learned policies.

VI. Conclusion

We present iDb-RRT, a novel algorithm for kinodynamic motion planning that combines search and optimization within the framework of Rapidly-Exploring Random Trees (RRT). Our algorithm connects motion primitives with a bounded discontinuity as the expansion step of an RRT, which is later repaired using trajectory optimization. iDb-RRT is probabilistically complete and finds solutions faster

than state-of-the-art kinodynamic motion planning across a diverse set of problems.

Comparatively, iDb-RRT and iDb-A* possess complementary strengths: the former finds solutions significantly faster, while the latter converges to optimal solutions with more compute time. Together, they demonstrate that combining motion primitives, bounded discontinuity, and trajectory optimization, is a promising approach for both sampling-based and search-based motion planning.

REFERENCES

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

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

[3] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, vol. 2, 2000, pp. 995–1001.

[4] R. Bohlin and L. E. Kavraki, "A randomized approach to robot path planning based on lazy evaluation," *Handbook on Randomized Computing*, vol. 9, no. 1, pp. 221–249, 2001.

[5] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2014, pp. 2997–3004.

[6] 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.

[7] 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 *Proc. IEEE Int. Conf. Robot. Autom.*, 2015, pp. 3067–3074.

[8] D. J. Webb and J. van den Berg, "Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 5054–5061.

[9] L. Chen, I. Mantegh, T. He, and W. Xie, "Fuzzy kinodynamic RRT: a dynamic path planning and obstacle avoidance method," in *Int. Conf. on Unmanned Aircraft Systems (ICUAS)*, 2020, pp. 188–195.

[10] 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.

[11] 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.

[12] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Açıkmeşe, "Convex optimization for trajectory generation: A tutorial on generating dynamically feasible trajectories reliably and efficiently," *IEEE Control Systems Magazine*, vol. 42, no. 5, 2022.

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

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

[15] 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.

[16] 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.

[17] 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.

[18] 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 *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2022, pp. 4492–4497.

[19] R. Natarajan, S. Mukherjee, H. Choset, and M. Likhachev, "PINSAT: parallelized interleaving of graph search and trajectory optimization for kinodynamic motion planning," *CoRR*, vol. abs/2401.08948, 2024.

[20] J. O. de Haro, W. Hönig, V. N. Hartmann, and M. Toussaint, "iDb-A*: Iterative search and optimization for optimal kinodynamic motion planning," *CoRR*, vol. abs/2311.03553, 2023.

[21] A. A. Masoud, "Kinodynamic motion planning," *IEEE Robotics & Automation Magazine*, vol. 17, no. 1, pp. 85–99, 2010.

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

[23] 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.

[24] 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.

[25] G. Goretkin, A. Perez, R. Platt, and G. Konidaris, "Optimal sampling-based planning for linear-quadratic kinodynamic systems," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2013, pp. 2429–2436.

[26] 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.

[27] Z. Tang, B. Chen, R. Lan, and S. Li, "Vector field guided RRT* based on motion primitives for quadrotor kinodynamic planning," *Journal of Intelligent & Robotic Systems*, vol. 100, pp. 1325–1339, 2020.

[28] J. Wang, W. Chi, C. Li, and M. Q.-H. Meng, "Efficient robot motion planning using bidirectional-unidirectional rrt extend function," *IEEE Transactions on Automation Science and Engineering*, 2022.

[29] S. Stoneman and R. Lampariello, "Embedding nonlinear optimization in RRT* for optimal kinodynamic planning," in *Proc. IEEE Conf. Decis. Control*, 2014, pp. 3737–3744.

[30] A. Ravankar, A. A. Ravankar, Y. Kobayashi, Y. Hoshino, and C.-C. Peng, "Path smoothing techniques in robot navigation: State-of-the-art, current and future challenges," *Sensors*, vol. 18, no. 9, 2018.

[31] S. A. Bortoff, "Path planning for UAVs," in *Proc. American Control Conf.*, vol. 1, no. 6, 2000, pp. 364–368.

[32] R. E. Allen and M. Pavone, "A real-time framework for kinodynamic planning in dynamic environments with application to quadrotor obstacle avoidance," *Robotics and Autonomous Systems*, 2019.

[33] J. Leu, M. Wang, and M. Tomizuka, "Long-horizon motion planning via sampling and segmented trajectory optimization," in *European Control Conference (ECC)*, 2022, pp. 538–545.

[34] 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.

[35] K. Wahba, J. Ortiz-Haro, M. Toussaint, and W. Hönig, "Kinodynamic motion planning for a team of multirotors transporting a cable-suspended payload in cluttered environments," *CoRR*, vol. abs/2310.03394, 2023.

[36] E. Jelavic, K. Qu, F. Farshidian, and M. Hutter, "LSTP: Long short-term motion planning for legged and legged-wheeled systems," *IEEE Trans. Robot.*, 2023.

[37] C. D. Bellicoso, F. Jenelten, C. Gehring, and M. Hutter, "Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, 2018.

[38] H. Li, R. J. Frei, and P. M. Wensing, "Model hierarchy predictive control of robotic systems," *IEEE Robot. Autom. Lett.*, 2021.

[39] 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.

[40] M. Kleinbort, K. Solovey, Z. Littlefield, K. E. Bekris, and D. Halperin, "Probabilistic completeness of RRT for geometric and kinodynamic planning with forward propagation," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 277–283, 2019.

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

[42] E. Granados, A. Sivaramakrishnan, and K. E. Bekris, "Towards benchmarking sampling-based kinodynamic motion planners with ml4kp," in *Workhsop: Evaluating Motion Planning Performance. IROS*, 2022.

[43] 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.