Benchmarking Sampling-, Search-, and Optimization-based Approaches for Time-Optimal Kinodynamic Mobile Robot Motion Planning

Wolfgang Hönig, Joaquim Ortiz-Haro, and Marc Toussaint

Abstract— **We consider time-optimal motion planning for dynamical systems that are translation-invariant, a property that holds for many mobile robots, such as differential-drives, cars, airplanes, and multirotors. Previous benchmarks have typically focused on comparing approaches within the same algorithmic class, e.g., sampling-based approaches may be benchmarked using the open motion planning library (OMPL). We provide the first benchmark that compares search-, sampling-, and optimization-based time-optimal motion planning on multiple dynamical systems in different settings.**

I. Introduction

Motion planning for robots with known kinodynamics remains challenging, especially when a time-optimal motion is desired or many obstacles are present. Consider the example of a simple unicycle dynamical model in 2D (3-dimensional state space and 2-dimensional control space). Finding the time-optimal solution is surprisingly challenging for state-of-the-art methods when constraining the control space to model a plane with a malfunctioning rudder, i.e., with a positive minimum speed and asymmetric angular velocity limits.

Current planning approaches are sampling-based, search-based, optimization-based, or hybrid. Each of these methods has their strengths and weaknesses. Sampling-based planners can find initial solutions quickly and have strong guarantees for convergence to an optimal solution. However, in practice the initial solutions are far from the optimum, the convergence rate is low, and the solutions typically require some post-processing. Search-based approaches can remedy those shortcomings by connecting precomputed trajectories, so-called *motion primitives*, using A* or related graph search algorithms. Yet, the seemingly strong theoretical guarantees only hold up to the selected discretization of the state space and the precomputed motions. Moreover, scaling this approach to higher dimensions has proved difficult and requires careful, frequently hand-crafted design of the motion primitives. This curse of dimensionality can be overcome by optimization-based planners, which scale polynomially rather than exponentially with the number of state dimensions. However, these planners are, in the general case, only locally optimal and thus require a good initial guess both for the trajectory and time horizon.

In this paper we contribute the, to our knowledge, first benchmark that compares the three major kinodynamic motion planning techniques on the same problem instances with the objective of computing time-optimal trajectories.

II. Problem Description

We consider states $\mathbf{x} = [\mathbf{x}^t, \mathbf{x}^r] \in \mathcal{X} \subset \mathbb{R}^{d_w} \times \mathbb{R}^{d_x - d_w}$, where the first d_w dimensions indicate the translation in the workspace ($d_w \in \{2,3\}$) of the robot and the remaining $d_x - d_w$ dimensions may contain orientation or derivatives. The robot can be actuated by actions $\mathbf{u} \in \mathcal{U} \subset \mathbb{R}^{d_u}$. We consider dynamics that are *translation invariant*, with

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

where $\mathbf{f}$ only depends on $\mathbf{x}^r$ and not on $\mathbf{x}^t$. In order to employ gradient-based optimization, we assume that we can compute the Jacobian of $\mathbf{f}$ with respect to $\mathbf{x}^r$ and $\mathbf{u}$. The collision-free state space is $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$.

Almost all generic kinodynamic motion planners assume a discrete-time formulation with zero-order hold, i.e., the applied action remains constant during a timestep. We can then frame the dynamics Eq. (1) as

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

using a small timestep Δt so that the Euler approximation holds sufficiently well.

Let $\mathbf{X} = \langle \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T \rangle$ be a sequence of states sampled at times $0, \Delta t, \ldots, T\Delta t$ and $\mathbf{U} = \langle \mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{T-1} \rangle$ be a sequence of actions applied to the system for times $[0, \Delta t), [\Delta t, 2\Delta t), \ldots, [(T-1)\Delta t, T\Delta t)$. Then our goal of moving the robot from its start state to a goal state can be framed as the following optimization problem:

$$\min_{\mathbf{U},\mathbf{X},T} J(\mathbf{U}, \mathbf{X}, T) \tag{3}$$
$$\text{s.t.} \begin{cases} \mathbf{x}_{k+1} = \text{step}(\mathbf{x}_k, \mathbf{u}_k) & \forall k \in \{0, \ldots, T-1\} \\ \mathbf{u}_k \in \mathcal{U} & \forall k \in \{0, \ldots, T-1\} \\ \mathbf{x}_k \in \mathcal{X}_{\text{free}} & \forall k \in \{0, \ldots, T\} \\ \mathbf{x}_0 = \mathbf{x}_s;\ \mathbf{x}_T = \mathbf{x}_f, \end{cases}$$

where $\mathbf{x}_s \in \mathcal{X}$ is the start state and $\mathbf{x}_f \in \mathcal{X}$ is the goal state. The objective function J is application specific; we will focus on time-optimal trajectories, i.e., $J(\mathbf{U}, \mathbf{X}, T) = T\Delta t$.

Example 1: Consider a unicycle robot with state $\mathbf{x} = [x, y, \theta] \in \mathcal{X}$, i.e., x, y are the position and θ is the orientation. The actions are $\mathbf{u} = [v, \omega] \in \mathcal{U}$, i.e., the

The authors are with TU Berlin, Germany. {hoenig, joaquim.ortizdeharo, toussaint}@tu-berlin.de.

This paper is a version of the work accepted at IROS 2022 [1], without details on the proposed hybrid motion planner. Code: https://github.com/IMRCLab/kinodynamic-motion-planning-benchmark

The research was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 448549715 and by the German-Israeli Foundation for Scientific Research (GIF) grant I-1491-407.6/2019. Joaquim Ortiz-Haro thanks the International Max-Planck Research School for Intelligent Systems (IMPRS-IS) for the support.

speed and angular velocity can be controlled directly. The dynamics are translation invariant: $\dot{\mathbf{x}} = [v \cos\theta, v \sin\theta, \omega]$. The choice of $\mathcal{U}$ can make this low-dimensional problem challenging to solve. For example, consider a plane-like case (positive minimum speed, i.e., $0.25 \leq v \leq 0.5\,\mathrm{m/s}$) with a malfunctioning rudder (asymmetric angular speed, i.e., $-0.25 \leq \omega \leq 0.5\,\mathrm{rad/s}$).

III. Kinodynamic Motion Planners Overview

There are several conceptually different algorithmic approaches to solving kinodynamic motion planning problems.

Search-based approaches rely on existing methods for discrete path planning, such as A* and variants. The common approach is to generate short trajectories (*motion primitives*) using a state lattice (i.e., pre-specified discrete state components) [2]. 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 the anytime variant ARA* can be employed without modification.

These methods can solve Eq. (3) if $\mathbf{x}_s$ and $\mathbf{x}_f$ fall within the chosen lattice and retain very strong theoretical guarantees on both optimality and completeness with respect to the chosen primitives. The major challenge is to select and compute good motion primitives, especially for high-dimensional systems [3].

Sampling-based approaches build a tree $\mathcal{T}$ rooted at the start state $\mathbf{x}_s$. During tree expansion, i) a random state $\mathbf{x}_{\mathrm{rand}}$ in the state space is sampled, ii) an existing state $\mathbf{x}_{\mathrm{expand}} \in \mathcal{T}$ is selected, and iii) a new state $\mathbf{x}_{\mathrm{new}}$ is added with a motion that starts at $\mathbf{x}_{\mathrm{expand}}$ and moves towards $\mathbf{x}_{\mathrm{rand}}$. The classic version of this approach, *kinodynamic RRT* [4], is probabilistically complete when using the correct variant. Asymptotic optimality can be achieved when planning in state-cost space (*AO-RRT*) [5] or by computing a sparse tree (*SST**) [6]. These methods rely on a distance function $d : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and often use fast nearest neighbor data structures such as k-d trees for efficiency. The mentioned algorithms work without solving a two-point boundary value problem, which is computationally expensive.

Sampling-based approaches are designed to explore the state space as fast as possible and typically do not use a heuristic function, unlike search-based methods. The exploration/exploitation tradeoff is typically controlled by using goal-biasing instead. These approaches cannot solve Eq. (3) directly, because the probability to reach $\mathbf{x}_f$ by sampling is zero. Instead, the goal constraint is typically reformulated to $\mathbf{x}_T \in \mathcal{X}_f$ using a goal region $\mathcal{X}_f$ rather than a goal state $\mathbf{x}_f$.

Optimization-based approaches locally optimize an initial trajectory using the gradients of J, unlike the previous gradient-free methods. Dynamics, collision avoidance, goal constraints, and control limits are modeled with piece-wise differentiable functions. In *CHOMP*, Hamiltonian Monte Carlo is used to perturb local solutions [7], while *TrajOpt* [8] and *GuSTO* [9] rely on sequential convex programming (SCP). Trajectories can also be computed with optimal control solvers that rely on Differential Dynamic Programming [10] or extend the linear quadratic regulator to nonlinear systems [11].

Both *STOMP* [12] and *KOMO* [13] use only the (geometric) state sequence $\mathbf{X}^t$ as decision variables and support kinodynamic systems via constraints. All optimization-based approaches require an initial guess as a starting trajectory, but this guess does not need to be kinodynamically feasible.

These approaches can solve Eq. (3) directly for a differentiable J and given number of timesteps T. The observed 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 from the curse of dimensionality directly, although higher dimensions might result in more local optima.

Hybrid approaches combine two or more ideas. One can combine search and optimization [14], search and sampling [15], [16], [17], or combine sampling and optimization [18]. For some dynamical systems, using insights from control theory for the motion planning can also be beneficial [19], [20], [21], but requires domain knowledge. Motion planning can also benefit from using machine learning for computational efficiency [22], [23].

Our recent approach, *kMP-db-A** [1], combines ideas from search-, sampling-, and optimization-based methods and uses the translation-invariance of mobile robot dynamics.

IV. Benchmark Results

We compare different motion planners on the same scenarios. For fair comparison, we share code and data structures as much as possible, use the respective state-of-the-art open-source implementations, and focus on settings where the dynamics and not the collision-checking create challenges.

A. Dynamical Systems

Unicycle (1st order) has a 3-dimensional state space $[x, y, \theta] \in SE(2)$ and a 2-dimensional $[v, \omega] \in \mathcal{U} \subset \mathbb{R}^2$ control space with dynamics defined in [24, Eq. (13.18)]. The simplest version (v0) uses bounds $v \in [-0.5, 0.5]$ m/s and $\omega \in [-0.5, 0.5]$ rad/s. More interesting variants are a plane-like version (v1) using a positive minimum speed of $0.25\,\mathrm{m/s}$, and a plane-like version with a rudder damage (v2) ($\omega \in [-0.25, 0.5]$ rad/s).

Unicycle (2nd order) has a 5-dimensional state space $[x, y, \theta, v, \omega] \in \mathcal{X} \subset \mathbb{R}^5$, a 2-dimensional $[\dot{v}, \dot{\omega}] \in \mathcal{U} \subset \mathbb{R}^2$ control space, and dynamics defined in [24, Eq. (13.46)]. Our version (v0) uses $v \in [-0.5, 0.5]$ m/s, $\omega \in [-0.5, 0.5]$ rad/s, $\dot{v} \in [-0.25, 0.25]$ m/s^2, and $\dot{\omega} \in [-0.25, 0.25]$ rad/s^2.

Car with trailer has a 4-dimensional state space $[x, y, \theta_0, \theta_1] \in \mathcal{X} \subset \mathbb{R}^4$, a 2-dimensional $[v, \phi] \in \mathcal{U} \subset \mathbb{R}^2$ control space, and dynamics and visualization given in [24, Eq. (13.19), Fig. 13.6]. We add an additional constraint $|\angle(\theta_0, \theta_1)| < \pi/4$ that avoids that the angle between the car and the trailer exceeds a threshold. Our version (v0) uses $v \in [-0.1, 0.5]$ m/s, $\phi \in [-\pi/3, \pi/3]$, $L = 0.25\,\mathrm{m}$, and $d_1 = 0.5\,\mathrm{m}$, where L and d_1 are defined in [24].

Quadrotor has a 13-dimensional state space (pose and first order derivatives using a Quaternion representation), a 4-dimensional control space (force for each of the four motors),

TABLE I

BENCHMARK RESULTS COMPARING SUCCESS RATE (p), TIME FOR THE FIRST FOUND SOLUTION (t^{st}), THE COST OF THE FIRST FOUND SOLUTION (J^{st}), AND THE COST OF THE SOLUTION AFTER $5\,\mathrm{min}$ (J^f). TIME AND COST ARE THE MEDIAN OVER 10 TRIALS. BEST RESULTS ARE **BOLD**.

#	System	Instance	SST*				SBPL				geom. RRT*+KOMO				kMP-db-A*			
			p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^f[s]$	p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^f[s]$	p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^f[s]$	p	$t^{\mathrm{st}}[s]$	$J^{\mathrm{st}}[s]$	$J^f[s]$
1	unicycle 1st order, v0	park	**1.0**	1.2	5.9	3.5	**1.0**	**0.0**	6.2	6.2	**1.0**	3.1	5.3	3.2	**1.0**	0.9	**3.2**	**3.1**
2		kink	**1.0**	1.2	47.5	17.9	**1.0**	**0.2**	22.6	22.6	**1.0**	9.9	24.8	21.7	**1.0**	5.5	**15.4**	**13.1**
3		bugtrap	**1.0**	**1.1**	63.3	30.0	**1.0**	1.4	36.8	36.6	**1.0**	10.8	40.3	22.2	**1.0**	21.6	**23.8**	**22.1**
4	unicycle 1st order, v1	kink	0.8	38.5	43.2	34.3					0.0	—	—	—	**1.0**	**11.9**	**23.9**	**23.7**
5	unicycle 1st order, v2	wall	0.8	29.9	45.2	37.4					0.0	—	—	—	**1.0**	**7.3**	**20.0**	**18.0**
6	unicycle 2nd order	park	**1.0**	**4.7**	14.4	7.5					**1.0**	5.5	10.6	**6.1**	**1.0**	7.9	**6.8**	**6.1**
7		kink	**1.0**	**2.6**	71.0	59.7					0.5	18.1	24.6	21.1	**1.0**	10.7	**20.4**	**19.8**
8		bugtrap	**1.0**	**5.2**	66.8	51.1					**1.0**	23.2	39.9	27.2	**1.0**	40.6	**33.9**	**25.9**
9	car with trailer	park	0.6	24.6	13.6	13.6					**1.0**	15.6	10.8	6.2	**1.0**	**10.0**	**5.7**	**5.4**
10		kink	**1.0**	**9.1**	70.8	66.5					0.1	217.5	174.4	130.8	**1.0**	94.8	**34.1**	**24.2**
11		bugtrap	**1.0**	**0.7**	47.5	43.8					0.0	—	—	—	**1.0**	8.3	**21.5**	**19.4**
12	quadrotor	empty	0.0	—	—	—					**1.0**	207.2	2.6	2.6	**1.0**	**131.0**	**1.6**	**1.6**

and dynamics defined in [25, Eq. (1)]. We use the parameters of the Crazyflie quadrotor with limits on the motor forces, velocity, and angular velocity. Note that the low thrust-to-weight ratio of 1.4 is very challenging for kinodynamic motion planning and that problem settings with a harsh initial condition prevent the use of specialized methods [26], [27].

We use $\Delta t = 0.1\,\mathrm{s}$ for all dynamical systems except the quadrotor, which uses $\Delta t = 0.01\,\mathrm{s}$.

B. Environments

For most of the dynamical systems, we consider three environments, which are inspired by the common use-cases in the related literature. For the v2 unicycle, we use the *wall* environment. For the quadrotor, we use an *empty* environment without obstacles. The scenario requires the quadrotor to recover from a harsh initial condition with an upside-down initial rotation and nonzero initial first derivatives. All environments only use simple geometric box shapes for efficient collision checking. The environments are bounded, where the bounds only limit the translational part of the state, i.e., parts of the robots are allowed to be outside.

C. Algorithms

For a **search-based** approach, we rely on SBPL (Search-based Planning Library), a commonly used C++ library with integration in the Robot Operating System (ROS). SBPL contains an example for unicycles, although the used dynamics do not match the ones from [24, eq. 13.18]. Thus, we generate our own primitives using optimization. Moreover, we make minor adjustments to the heuristic to enable time-optimal anytime planning using the provided implementation of ARA* in SBPL. Due to limits in SBPL, we limit our evaluation to the v0 first order unicycle.

For a **sampling-based** approach, we rely on OMPL (Open Motion Planning Library), a widely used C++ library with integration in ROS through MoveIt. OMPL implements several kinodynamic planners, including SST* [6], which we use. As part of this work, we contribute minor changes to allow time as an optimization objective. Since sampling-based kinodynamic approaches cannot reach a goal state, we use a goal region instead that we verify to be small enough such that an optimizer can find an exact solution.

For an **optimization-based** approach, we rely on RAI (Robotic AI), a C++ library that implements KOMO and nonlinear optimization algorithms. For each of the dynamical systems, we implement the appropriate constraints and their derivative computation. In case of the trailer and the quadrotor, we add parts of the actions as decision variables (angle ϕ and motor forces, respectively); otherwise the decision variables are the state sequences only. As an initial guess, we use a geometric solution as found by RRT* of OMPL. This combination of *geometric RRT*+KOMO* is anytime like the other approaches we compare to.

For **kMP-db-A***, we use our implementation [1] that uses data structures provided in OMPL to represent states and for nearest neighbor computation.

The benchmark infrastructure is written in Python and all tuning parameters can be found in the open-source repository. Collision checking is done using FCL (Flexible Collision Library) in all cases. All approaches use the Euler integration Eq. (2), although KOMO uses an implicit formulation by design.

D. Benchmark

We execute our benchmark on a desktop computer with AMD Ryzen 9 3900X ($3.8\,\mathrm{GHz}$) and $32\,\mathrm{GB}$ RAM. Our results are summarized in Table I.

We summarize the main results as follows. **SBPL** can compute results very quickly and consistently. The initial solution quality is very high, but due to the limited number of primitives, the solution does not improve much over time (rows 1 – 3, Table I). The approach is not as general as the other ones, and we were unable to use it for all of our dynamical systems. **SST*** can find an initial solution very quickly; however the solution quality is initially poor, especially with higher-dimensional systems (rows 6–11). The convergence is slow – our $5\,\mathrm{min}$ timeout was not sufficient for SST* to fully converge in any of the cases. **Geometric RRT*+KOMO** can find near-optimal initial solutions, but does not work well in instances that require long trajectories

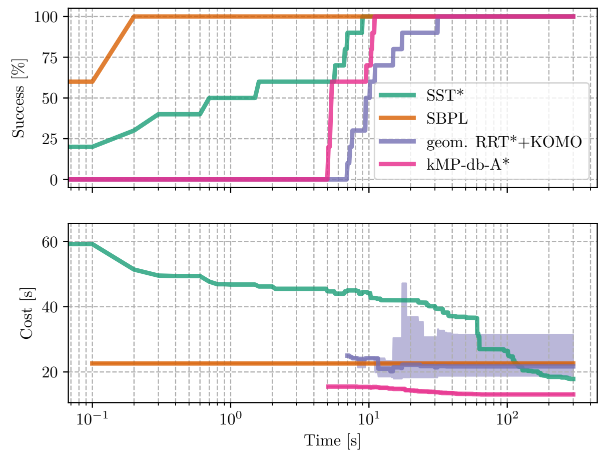

Fig. 1. Success rate and solution cost over runtime (log-scale) for the unicycle (1st order) dynamical system in the kink environment (row 2 in Table I). The line is the median and the shaded region shows the first/third quartile over all trials that found a solution so far (up to 10 in total).

and fails if the geometric initial guess is not close to a dynamically feasible motion. For example, finding an initial solution in the kink and bugtrap examples (rows 7, 8) took significantly longer than parallelpark. Another drawback is that this approach is incomplete, as visible for the v1 and v2 unicycle systems (row 4 and 5) and not globally optimal, e.g., row 10 and 11 show a very poor solution quality after 5 min. **kMP-db-A*** converged to the lowest-cost solution during the time limit in all cases. At the same time, it found the highest-quality first solution in all cases, although it often took more time to compute an initial solution than the other algorithms.

For brevity, Table I does not include any standard deviation. In general, we found that SBPL has almost no variance, SST* has a very high variance, and KOMO and kMP-db-A* are somewhere in between the two extremes. One example that includes the convergence behavior as well as the variance is shown in Fig. 1.

V. Conclusion

We believe that benchmarking approaches across methodologies is important. This requires us to define scenarios and objectives that are independent of a concrete library such as OMPL. For practitioners, our benchmarks can identify suitable algorithms for kinodynamic planning; for researchers, they provide insights and new ideas for hybrid planners.

References

[1] 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.

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

[3] 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.

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

[5] 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.

[6] 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.

[7] M. Zucker, N. D. Ratliff, A. D. Dragan, M. Pivtoraiko, M. Klingensmith, C. M. Dellin, J. A. Bagnell, and S. S. Srinivasa, "CHOMP: Covariant hamiltonian optimization for motion planning," *I. J. Robotics Res.*, vol. 32, no. 9-10, pp. 1164–1193, 2013.

[8] 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.

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

[10] 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.*, pp. 2536–2542.

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

[12] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "STOMP: Stochastic trajectory optimization for motion planning," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2011, pp. 4569–4574.

[13] 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.

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

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

[21] 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.

[22] 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 Trans. Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4298–4305, 2019.

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

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

[25] G. Shi, W. Hönig, Y. Yue, and S.-J. Chung, "Neural-swarm: Decentralized close-proximity multirotor control using learned interactions," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2020, pp. 3241–3247.

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

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