# BITKOMO: Combining Sampling and Optimization for Fast Convergence in Optimal Motion Planning

Jay Kamat[1,2], Joaquim Ortiz-Haro[1], Marc Toussaint[1], Florian T. Pokorny[3], Andreas Orthey[1]

*Abstract*— Optimal sampling based motion planning and trajectory optimization are two competing frameworks to generate optimal motion plans. Both frameworks have complementary properties: Sampling based planners are typically slow to converge, but provide optimality guarantees. Trajectory optimizers, however, are typically fast to converge, but do not provide global optimality guarantees in nonconvex problems, e.g. scenarios with obstacles. To achieve the best of both worlds, we introduce a new planner, BITKOMO, which integrates the asymptotically optimal Batch Informed Trees (BIT*) planner with the K-Order Markov Optimization (KOMO) trajectory optimization framework. Our planner is anytime and maintains the same asymptotic optimality guarantees provided by BIT*, while also exploiting the fast convergence of the KOMO trajectory optimizer. We experimentally evaluate our planner on manipulation scenarios that involve high dimensional configuration spaces, with up to two 7-DoF manipulators, obstacles and narrow passages. BITKOMO performs better than KOMO by succeeding even when KOMO fails, and it outperforms BIT* in terms of convergence to the optimal solution.

## I. INTRODUCTION

Generating optimal motions plans is crucial for almost any robotic tasks ranging from typical manipulation tasks such as bin-picking to autonomous navigation of mobile robots. To solve such tasks, the robotics community relies on two powerful motion planning frameworks: Sampling-based planners and trajectory optimization.

Sampling-based planners like RRT* [1], BIT* [2] or FMT* [3] converge asymptotically to optimal solutions and almost surely provide a solution if one exists [4]. However, these planners are slow at converging to the optimal trajectory, because improvements to the current best solution only arise when we sample a state nearby [5], and often provide non-smooth trajectories that may require post-processing [6].

Trajectory optimization methods like KOMO [7], CHOMP [8], STOMP [9] and TrajOpt [10] use optimization methods and can exploit gradient and second order information to converge to a local optimal solution. These optimization-based methods are typically fast at converging to the local optimum, however, due to the non-convexity of the problem, the optimizer might converge to a locally optimal or even an infeasible trajectory. These methods therefore do not have convergence guarantees, i.e. they may not converge to a solution even if one exists, and the feasibility of the solution often depends heavily on the
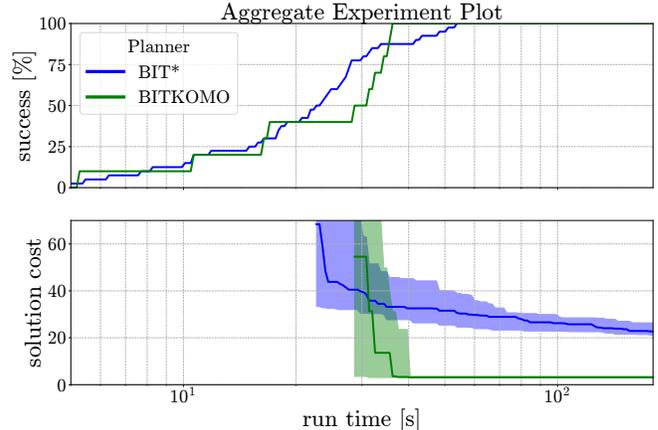
**Fig. 1:** Aggregate plot over all our experiments, showing how BITKOMO converges faster than BIT*, while keeping a similar success rate. KOMO is not included, because it fails to find solutions on more than half of our experiments.

initial trajectory [11], [12], [13]. Hence, these methods usually work well in environments with few obstacles or when provided with good initial guesses [14], e.g. for post-processing paths produced by sampling-based planners.

In order to combine the benefits of both frameworks, we propose to integrate the asymptotically optimal Batch Informed Trees (BIT*) [2] planner with K-Order Markov Optimization (KOMO) [7], [15]. Combining sampling and optimization helps us play on the strengths of each framework and mitigate their weaknesses. Our novel algorithm, BITKOMO, uses BIT* to iteratively generate non-optimal initial paths that are then optimized by KOMO, which results in quick convergence to local minima. The cost of the optimized path is then used by BIT* to carry a more informed search. With this method, we maintain the asymptotic global optimality guarantees by BIT* while also benefiting from the fast convergence of KOMO (Fig. 1).

At the core of BITKOMO lies a new relaxed edge collision checking method. Relaxed edge collision is an intermediate approach between full and lazy [16] collision checking, where we allow partially-valid edges to remain, because the trajectory optimizer KOMO can often push invalid paths out of collision [15] to converge to feasible solutions. Even though this modification allows for invalid edges, we do not sacrifice any of the asymptotic guarantees provided by BIT*. To summarize, we make two major contributions:

1) *Relaxed edge collision checking:* A method for BIT* that allows edges partially in collision to be included in the motion tree.

2) *BITKOMO: A planner integrating BIT\* and KOMO.* We integrate sampling and optimization to obtain fast convergence to the global optimum while maintaining the guarantees provided by the sampler.

## II. RELATED WORK

Combining both asymptotically-optimal planners [17], [5], [18], [19] and trajectory optimization [7], [8], [9], [10], [20] into one concise algorithm is an important long-term goal of the robotics community [21]. There exist three main approaches. The *two-step approach*, where we run a sampling-based planner like the rapidly-exploring random tree (RRT) [4], or the probabilistic roadmap (PRM) [22] until a valid solution path is found. In a post-processing step [23], [6], we call a trajectory optimizer which optimizes the path to find a (local) optimal solution. This method is the most common approach, but does not hold guarantees on optimality, and does not allow the sampling-based planner to find improved solutions by leveraging information acquired by the optimizer. Our BITKOMO approach improves upon this methodology by tightly integrating sampling and optimization in an iterative fashion. Moreover, we use relaxed collision checking to quickly find partially-valid paths. Optimizers like KOMO [7] can often repair those paths to quickly find a solution.

The second approach is the *optimizer-as-steering* method, where the integration of sampling and optimization is done using the steering function from the planner. The regionally-accelerated BIT\* (RABIT\*) [24] is an extension of BIT\* that uses an optimizer to push infeasible edges out of collision. However, while the obtained solution cost is often better than BIT\*, calls to an optimizer are often expensive and slow down the algorithm. It is also possible to integrate such an approach into a roadmap planner [25], to interleave edge creation and optimization steps. We differ by postponing optimization to the point where a full partially-valid path has been found, which reduces the number of unnecessary calls to the nonlinear optimizer.

Finally, the *path-proposal* method, where the planner proposes solution paths, which are then sent to the optimizer [26], [27]. An important aspect of this method is to propose many diverse solution paths, so that the optimizer has a lesser chance to converge to similar solutions. This can be accomplished by leveraging sparse roadmaps [28], to generate diverse initial trajectories which we can then send to the trajectory optimizer [29], [30], [31]. The closest work to ours is [21], where RRT\* is integrated with a nonlinear optimizer in an iterative fashion.

Our method is complementary, in that we also use the path-proposal method. However, we differ from previous works by our choice of the underlying components (BIT\* and KOMO) and/or the interface. Further, the novel collision checking improves the performance in problems involving narrow passages, when the sampling-based planner fails to find a path in the first place.
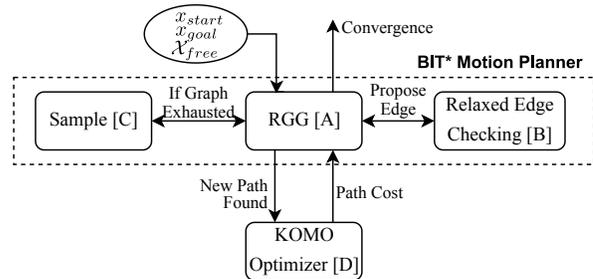


**Fig. 2:** Overview of BITKOMO, which combines the BIT\* architecture with our custom relaxed edge checking for path planning, and the KOMO optimizer for path optimization.

## III. PROBLEM DEFINITION

We consider a motion planning problem in a configuration space $\mathcal{X} \subset \mathbb{R}^n$ of the form $(\mathcal{X}_{\text{free}}, x_{\text{start}}, x_{\text{goal}}, c)$ where $\mathcal{X}_{\text{free}} \subseteq \mathcal{X}$ is the free configuration space, $x_{\text{start}} \in \mathcal{X}_{\text{free}}$ is the start configuration, $x_{\text{goal}} \in \mathcal{X}_{\text{free}}$ is the goal configuration and $c : \mathcal{P} \to \mathbb{R}$ is the cost functional mapping a trajectory $p \in \mathcal{P}$ in the free configuration space to a real number. Our goal is to find a trajectory $p : [0, 1] \to \mathcal{X}_{\text{free}}$ from $p(0) = x_{\text{start}}$, to $p(1) = x_{\text{goal}}$ that is optimal, i.e. the value of $c(p)$ is the lowest among all possible paths.

$$\min_{p(t)} \quad c(p(t)) \tag{1a}$$

$$\text{s.t.} \quad p(0) = x_{\text{start}}, \quad p(1) = x_{\text{goal}} \tag{1b}$$

$$\forall t \in [0, 1], \quad p(t) \in \mathcal{X}_{\text{free}} . \tag{1c}$$

In this paper, we focus on path-length optimization, i.e. $c(p) = \int_0^1 ||\dot{p}(t)|| \, dt$. Typical examples of alternative cost functionals are the sum of the velocities squared $\int_0^1 ||\dot{p}(t)||^2 \, dt$, and smoothness $\int_0^1 ||\ddot{p}(t)||^2 \, dt$.

## IV. BITKOMO

BITKOMO integrates two state-of-the-art motion planners: BIT\* and KOMO. BIT\* is an anytime, asymptotically-optimal planner that samples collision-free configurations in batches and generates paths using the A\* graph search algorithm. KOMO is a non-linear trajectory optimizer that locally optimizes an initial trajectory (possibly in collision) to minimize a cost and fulfil collision avoidance and goal constraints. Our planner maintains the asymptotic optimality guarantees of BIT\* while converging faster to the global minimum by leveraging trajectory optimization. Since KOMO uses the Augmented Lagrangian algorithm for constrained optimization, it can sometimes push partially invalid paths out of collision. To exploit this feature, we introduce relaxed edge checking, which allows BIT\* to produce partially infeasible paths for subsequent optimization with KOMO when necessary.

### A. The BITKOMO Algorithm: An overview

Our planner (Fig. 2) requires a valid start state ($x_{\text{start}}$), a goal state ($x_{\text{goal}}$) and the full information about the environment ($\mathcal{X}_{\text{free}}$). We also need to provide the Planner Termination condition ($PTC$) and the edge relaxation number ($\delta$). To

begin, the BIT* planner samples a batch of configurations $x \in \mathcal{X}_{\text{free}}$ and builds an *edge-implicit* Random Geometric Graph (RGG) [32] [Block A in Fig. 2]. The best edge that can possibly improve the cost to goal (as in A*) is then chosen and passed to the *Relaxed Edge Checker* [B] to carry out the collision checking. The Relaxed Edge Checker performs a validity check and returns the collision penalty ($\mathcal{CP}$), an integer that provides a proxy measure for the fraction of the edge that is in collision. The planner uses this integer to decide regarding the addition of the edge to the tree. If an improved path to the goal is found, it is passed to the KOMO optimizer [D] which locally optimizes the path and, if valid, returns the new cost to the BIT* planner. BIT* uses this path cost to prune the unnecessary vertices and edges and carry out a more focused search. When no new edges can be expanded, the Sample function [C] is called, which adds another batch of samples to the RGG.

---

**Algorithm 1** BITKOMO

**Input** $\mathcal{X}_{\text{free}}, x_{\text{start}}, x_{\text{goal}}, PTC, \delta$

1: $\mathcal{V} \leftarrow \{x_{\text{start}}\}; \mathcal{E} \leftarrow \emptyset; \mathcal{T} \leftarrow (\mathcal{V}, \mathcal{E});$
2: $\mathcal{Q}_V \leftarrow V; \mathcal{Q}_E \leftarrow \emptyset;$
3: $X_{\text{uc}} \leftarrow x_{\text{goal}};$
4: $c_i \leftarrow \infty; c_{\text{best}} \leftarrow \infty;$
5: $c_{\text{max}} \leftarrow \text{GETDIAGONALLENGTH}(\mathcal{X}) \times 3;$
6: **while** $\neg PTC$ **do**
7:     **if** $\mathcal{Q}_E \equiv \emptyset$ and $\mathcal{Q}_V \equiv \emptyset$ **then**
8:         $X_{\text{uc}} \overset{+}{\leftarrow} \text{PRUNE\&SAMPLE}(\mathcal{T}, X_{\text{uc}}, m, c_i);$
9:         $\mathcal{Q}_V \leftarrow V;$
10:     **end if**
11:     **while** $\text{BESTVALUE}(\mathcal{Q}_V) \leq \text{BESTVALUE}(\mathcal{Q}_E)$ **do**
12:         $\text{EXPANDNEXTVERTEX}(\mathcal{Q}_V, \mathcal{Q}_E, c_i);$
13:     **end while**
14:     $E = \{v_{\text{min}}, x_{\text{min}}\} \leftarrow \text{POPBESTINQUEUE}(\mathcal{Q}_E);$
15:     **if** $\text{EDGEADDITIONHELPS}(E, c_i)$ **then**
16:         $\mathcal{CP} \leftarrow \text{CHECKEDGERELAXED}(E);$
17:         **if** $\mathcal{CP} \leq \delta$ **then**     ▷ If true, Edge is used
18:             $c_{\text{edge}} \leftarrow \hat{c}(E) + \mathcal{CP} \times c_{\text{max}};$
19:             **if** $\text{EDGEIMPROVESCOST}(E, c_i, c_{\text{edge}})$ **then**
20:                 $\text{ADDEDGETOTREE}(E);$
21:             $c_i \leftarrow \text{GETCOSTTOGO}(v_{\text{goal}});$
22:             **if** $c_i < c_{\text{max}}$ **then**   ▷ If path is feasible
23:                 $c_{\text{best}} \leftarrow c_i;$
24:             **end if**
25:             $path \leftarrow \text{GETBESTPATH}(v_{\text{goal}});$
26:             $optiPath \leftarrow \text{OPTIMIZE}(path);$
27:             **if** $\text{ISVALID}(optiPath)$ **then**
28:                 $c_i \leftarrow \text{GETCOST}(optiPath);$
29:                 $c_{\text{best}} \leftarrow c_i;$
30:             **end if**
31:             **end if**
32:         **end if**
33:     **end if**
34: **end while**
35: **return** Solution;

*Labels A–E bracket the algorithm lines as indicated.*
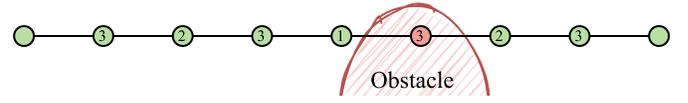


**Fig. 3:** *Levels in Edge Checking* The edge is first subdivided into $n_d$ points and these points are then checked for collision level by level. The number on the nodes represent the level of the node. This example has 3 levels.

Algorithm 1 describes in more detail the different parts of the planner. The highlighted lines are our addition to the BIT* planner. Blue — the *Relaxed edge collision checking*, Orange — the interface between BIT* and KOMO.

*1) Initialize (A):* Vertex set ($\mathcal{V}$), Edge set ($\mathcal{E}$), Tree ($\mathcal{T}$), Vertex queue ($\mathcal{Q}_V$), Edge queue ($\mathcal{Q}_E$), Set of unconnected vertices ($X_{\text{uc}}$). Also initialize three important cost parameters: 1) $c_i$ — cost used by the BIT* tree, it includes infeasible paths; 2) $c_{\text{best}}$ — the cost of the best feasible path; 3) $c_{\text{max}}$ — a penalty cost higher than any feasible path BIT* could converge to.

*2) Batch Addition (B):* When we run out of the batch samples (line 7), we prune our graph using the ellipsoid method and add a new batch of samples [33].

*3) Edge Selection (C):* (The A* search) If expanding a vertex can help improving the cost of our solution, it is expanded, i.e., relevant vertices and edges are added to their respective queues (line 12)

*4) Edge processing (D):* Decides whether to add new edge to the tree. If the new edge can improve the overall cost to goal (line 15), and the edge is collision free / partially in collision (line 16, 17) such that it still is a good edge to add (line 19), it is added to the tree (line 20). ADDEDGETOTREE(.) rewires the tree if necessary.

*5) KOMO Optimization (E):* If the addition of the new edge provides us with a better path to goal, this solution path is optimized using KOMO (line 26). The resulting path ($optiPath$) is then checked for validity and the costs ($c_i$ and $c_{\text{best}}$) are updated accordingly. For completeness, we also check if the initial guess is valid by checking if the path cost is less than $c_{\text{max}}$ (line 22) and update $c_{\text{best}}$ if valid. The working of KOMO is explained in IV-C.

### B. Relaxed Edge Checking

High dimensional spaces containing narrow passages are challenging for sampling based planners. This is because it is difficult to sample collision free edges through narrow passages. Since KOMO can push paths out of obstacles, we could allow paths partially in collision into the BIT* tree. However, these edges need to be added with sufficient collision penalty to ensure that BIT* does not mistake a path in collision to be of a lower cost than the true minima. We also want our collision checker to quickly guess the extent of collision so as to be quick in finding a solution for BIT*. We solve this problem by introducing Relaxed Edge Checking which returns a number instead of a Boolean which is used to assign a collision penalty (line 18). It returns 0-if edge is collision free, 1-if it fails at the last level, 2-if it fails on the

second to last level, and so on. Adding the collision penalty this way also helps our planner to prefer collision free initial paths for optimization as the likelihood of finding a feasible trajectory from a collision-free path is higher.

Suppose for a given resolution, we need to check $n_d$ equally spaced points to confirm the edge to be collision-free. The Relaxed Edge Checker conducts a level wise collision checking (see Fig. 3) whereby the resolution of checking is increased until the required resolution is reached or a collision is detected. We first check the mid point (level 1), then the quarter points (level 2) and so on by slowly doubling the resolution of checking. If a point fails in the validity check, an integer, collision penalty ($\mathcal{CP} = \mathcal{L} - \mathcal{L}_c + 1$) is returned. Where $\mathcal{L} = \lceil \log_2 n_d \rceil$ is the total number of levels, and $\mathcal{L}_c$ is the level of the failed point. This number provides a proxy measure for the fraction of the edge that is in collision.

### C. KOMO

K-Order Markov Optimization (KOMO) is a trajectory optimization framework that represents a path with a discrete sequence of waypoints $\langle x_0 \ldots x_T \rangle$. Cost and constraints are evaluated on, up to $k+1$ consecutive waypoints (Markov assumption)

$$\min_{x_{0:T}} \quad \sum_{t=0}^{T} f_t(x_{t-k:t})^\top f_t(x_{t-k:t}) \tag{2a}$$

$$\text{s.t.} \quad \forall t : g_t(x_{t-k:t}) \leq 0, \quad h_t(x_{t-k:t}) = 0 , \tag{2b}$$

where $x_{t-k:t}$ is a $k+1$ tuple of consecutive states. In our setting, where the goal is to minimize the path length, $k = 1$, and we use, as cost, the sum of squared distances $\sum ||x_t - x_{t-1}||^2$, which corresponds to $f_t(x_{t-1}, x_t) = x_t - x_{t-1}$.

Inequality constraints correspond to collision avoidance and joint limits and equality constraints model the terminal goal condition $x_T = x_{\text{goal}}$. The optimization problem (2) is solved with the Augmented Lagrangian algorithm for constrained optimization. The Markov structure, together with second order information, enables very efficient solving, with complexity linear on the number of waypoints and polynomial on the dimension of the configuration space [7].

### D. Convergence and Optimality Guarantees

BITKOMO maintains the convergence and optimality guarantees of BIT* [2]. The additional trajectory optimization can only improve the solution proposed by BIT* (lines 26-30 in Alg. 1). The *relaxed edge checking* assigns cost $c > c_{\text{max}}$ (line 18 in Alg. 1) to any edge in collision (recall that $c_{\text{max}}$ is an upper bound on the optimal solution cost, that can be chosen arbitrarily large). Even if the subsequent optimization fails, the edge cost does not prevent BIT* and hence BITKOMO from finding a solution with cost $c < c_{\text{max}}$.

## V. EVALUATION

### A. Scenarios

We evaluate our algorithm on 6 different robotic scenarios[1]. In all scenarios, the robot moves from the initial configuration (solid color) to the goal configuration (translucent



**(a)** Disc Robot    **(b)** Kuka from shelf    **(c)** Kuka into box

**(d)** Fixed Pandas    **(e)** Two Mobile Pandas    **(f)** One Mobile Panda
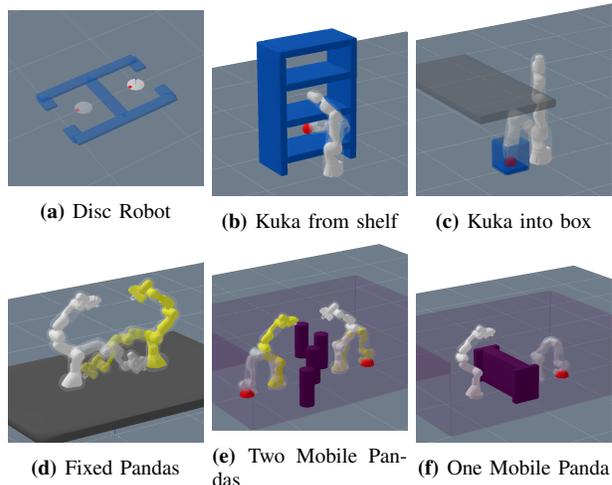
**Fig. 4:** Scenarios used in our experimental evaluation. See the supplementary video for the solution trajectories.

color) (Fig. 4). The trajectories computed by BITKOMO and the baseline algorithms are shown in the supplementary video[2]. We emphasize the challenges of each problem with the keywords: *narrow passage*, *not informative heuristic* and *high-dimensional*.

1) *Disc Robot in Rooms:* A Disc Robot needs to move from the center of one room to another (Fig. 4a). The difficulty is that, to go to the other room, the robot first needs to come out of the starting room and then move to the target room. Challenges: narrow passages and not informative heuristic.

2) *Kuka to reach onto the shelf:* The Kuka robot needs to reach the red object at level 1 from it's current position where the end-effector is at level 2 (Fig.4b). Challenges: High-dimensional.

3) *Kuka to reach into a box:* The Kuka Robot needs to reach to the object inside the box that is located under a table while avoiding collision with the table or the box (Fig. 4c). Challenges: Narrow Passage, High Dimensional.

4) *Two Fixed Pandas:* The robotic manipulators (Pandas) need to get to the base of the opposite robot while avoiding hitting each other (Fig. 4d). Challenges: High Dimensional.

5) *Two Mobile Pandas in cluttered environment:* Two mobile panda robots need to move to the other side of the room while avoiding obstacles and also avoiding each other (Fig. 4e). Challenges: High Dimensional, Narrow Passages.

6) *One Mobile Panda to avoid large obstacle:* The mobile panda needs to move to catch an object on the other side of the scene, with a large obstacle blocking it's way (Fig. 4f). Challenges: High Dimensional, Narrow Passages.
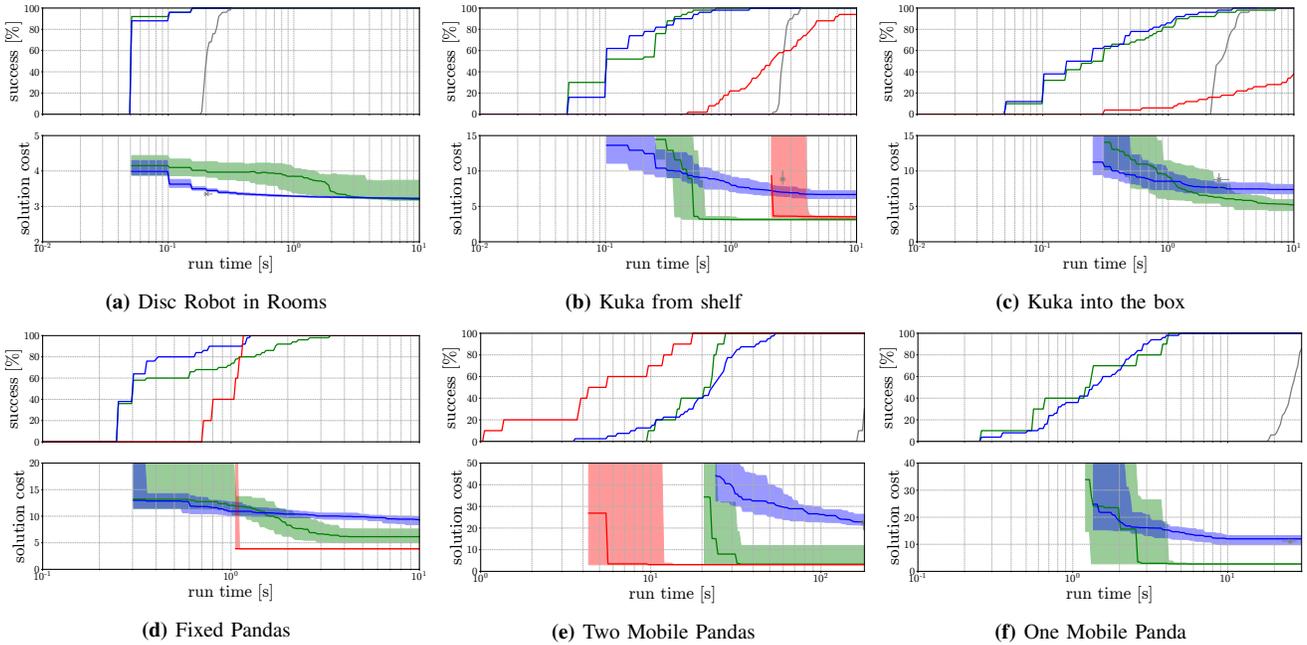
**Fig. 5:** Results: Success rates and best cost plots for ■ BITKOMO, ■ BIT*, ■ FMT* and ■ KOMO on the 6 different example environments.

*B. Baselines*

We compare our BITKOMO planner with BIT* [2], KOMO [7] and FMT* [3]. Path length minimization is used as the optimization objective for all the experiments. We used the Open Motion Planning Library (OMPL) [34] framework for the implementations of the sampling based planners and for carrying out the benchmarks.

For the KOMO planner we use the sum of squares of the distances between waypoints as optimization objective, and an initialization with random noise around $p(t_i) = x_{start} \; \forall t_i$. The trajectory is represented with a constant number of waypoints (20 points). Random initialization and optimization are executed iteratively until timeout, updating the path cost when a better path is found.

*C. Metrics*

We evaluate the planners on 2 different metrics:

1) *Success rate (%):* The % of runs that have found a feasible solution at time $t$. This metric gives information about how fast the planner finds the first feasible path.
2) *Cost:* The average best cost of the planner at time $t$. This metric gives us an understanding about how the best cost solution of a planner evolves over time and the practical convergence speed before the timeout.

*D. Experimental Results*

For getting unbiased results, all experiments were conducted on the same machine[3]. Every planner was executed 50 times on all the six example scenarios. The maximum execution time however was different for different scenarios depending on the difficulty. The edge relaxation number $\delta$

[3]Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz having 16GB RAM

was set to 1 for all examples. The results of the benchmarks are shown in Fig. 5.

*1) Success rate:* The success rate of BIT* and BIKOMO were higher compared with other planners in all examples except the *Two Mobile Pandas* example. This scenario has narrow passages which makes it hard for sampling based planners, however, the optimal solution is very similar to a straight line path in the configurations space, making it very easy for KOMO to find a solution here. The relaxed edge checking helps BITKOMO in having a slightly better success rate than BIT* here. The anomaly in BIKOMO success rate in Fig. 5d is because of a failed optimization. This failure is because of the thin obstacles in the C-Space arising due to collision between robots. This, however, is not a large time difference. Choosing a smaller edge relaxation number, $\delta$, will fix it. Overall, the success rates of BIT* and BITKOMO were found to be very similar because they generate initial paths using the same base algorithm.

*2) Cost:* BITKOMO decreases the cost significantly faster than BIT*, with better convergence before the timeout. This is because the combination of sampling and optimization converges to the local minima quickly and consistently. We however see an abnormality in Fig.5a. This is because — 1) KOMO is not much faster than sampling for low dimensions, and 2) The waypoints maintain a certain minimum distance from the obstacles to avoid edge collisions.

Overall, we conclude that our planner is mostly as good as BIT* in finding the first feasible solution and slightly faster in high dimensional narrow passage problems, but much faster at converging to the global optimal solution.

## VI. DISCUSSION AND CONCLUSION

Our planner, BITKOMO, combines BIT* and KOMO to achieve fast convergence to the optimal solution while being

anytime and asymptotically converging to the global minimum. Our experiments indicate that BITKOMO converges to the global optima, faster than BIT*. It also provides convergence guarantees which KOMO does not. Using Relaxed Edge Checking, our planner exploits the ability of KOMO to move trajectories away from the obstacles that are in collision by allowing partially infeasible paths as initial guesses to the optimizer. This helps BITKOMO find motions through narrow passages faster than BIT*.

Even though we observe faster convergence than BIT* to optimal paths, our planner does not have a better success rate. A dedicated planner could be developed to generate improved initial guesses to the optimizer, resulting in an increased success rate. The optimization and sampling modules could also easily be parallelized, providing a higher speedup. Calling the KOMO optimizer ahead of the BIT* planner could increase the speed further.

Our experiments clearly demonstrate that BITKOMO can robustly achieve fast convergence to optimal motion plans. This is an important step towards making optimal motion planners converge as quickly as trajectory optimizers — all while keeping asymptotic optimality guarantees.

## REFERENCES

[1] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE conference on decision and control (CDC)*. IEEE, 2010, pp. 7681–7687.

[2] 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.

[3] L. Janson, E. Schmerling, A. Clark, and M. Pavone, "Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions," *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

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

[5] O. Salzman and D. Halperin, "Asymptotically near-optimal rrt for fast, high-quality motion planning," *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, 2016.

[6] R. Geraerts and M. H. Overmars, "Creating high-quality paths for motion planning," *The international journal of robotics research*, vol. 26, no. 8, pp. 845–863, 2007.

[7] M. Toussaint, "Newton methods for k-order markov constrained motion problems," *arXiv preprint arXiv:1407.0414*, 2014.

[8] N. Ratliff, M. Zucker, J. A. Bagnell, and S. Srinivasa, "Chomp: Gradient optimization techniques for efficient motion planning," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 489–494.

[9] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal, "Stomp: Stochastic trajectory optimization for motion planning," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 4569–4574.

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

[11] W. Merkt, V. Ivan, and S. Vijayakumar, "Leveraging precomputation with problem encoding for warm-starting trajectory optimization in complex environments," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2018, pp. 5877–5884.

[12] T. S. Lembono, A. Paolillo, E. Pignat, and S. Calinon, "Memory of motion for warm-starting trajectory optimization," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2594–2601, 2020.

[13] J. Ichnowski, Y. Avigal, V. Satish, and K. Goldberg, "Deep learning can accelerate grasp-optimized motion planning," *Science Robotics*, vol. 5, no. 48, p. eabd7710, 2020.

[14] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[15] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence*, 2015.

[16] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *IEEE International Conference on Robotics and Automation*. IEEE, 2015, pp. 2951–2957.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[18] J. D. Gammell and M. P. Strub, "Asymptotically optimal sampling-based motion planning methods," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, pp. 295–318, 2021.

[19] M. P. Strub and J. D. Gammell, "AIT* and EIT*: Asymmetric bidirectional sampling-based path planning," *arXiv preprint arXiv:2111.01877*, 2021.

[20] M. Mukadam, J. Dong, X. Yan, F. Dellaert, and B. Boots, "Continuous-time gaussian process motion planning via probabilistic inference," *International Journal of Robotics Research*, vol. 37, no. 11, pp. 1319–1340, 2018.

[21] V. N. Hartmann, O. S. Oguz, and M. Toussaint, "Planning planning: The path planner as a finite state machine," 2020.

[22] 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.

[23] J. Kim, R. A. Pearce, and N. M. Amato, "Extracting optimal paths from roadmaps for motion planning," in *IEEE International Conference on Robotics and Automation*, vol. 2. IEEE, 2003, pp. 2424–2429.

[24] 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 *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2016, pp. 4207–4214.

[25] K. V. Alwala and M. Mukadam, "Joint sampling and trajectory optimization over graphs for online motion planning," in *IEEE International Conference on Intelligent Robots and Systems*. IEEE, 2020, pp. 4700–4707.

[26] A. Kuntz, C. Bowen, and R. Alterovitz, "Fast anytime motion planning in point clouds by interleaving sampling and interior point optimization," in *Robotics Research*. Springer, 2020, pp. 929–945.

[27] M. Xanthidis, N. Karapetyan, H. Damron, S. Rahman, J. Johnson, A. O'Connell, J. M. O'Kane, and I. Rekleitis, "Navigation in the presence of obstacles for an agile autonomous underwater vehicle," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 892–899.

[28] A. Orthey, B. Frész, and M. Toussaint, "Motion planning explorer: Visualizing local minima using a local-minima tree," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 346–353, April 2020.

[29] S. Dai, M. Orton, S. Schaffert, A. Hofmann, and B. Williams, "Improving trajectory optimization using a roadmap framework," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 8674–8681.

[30] C. Park, F. Rabe, S. Sharma, C. Scheurer, U. E. Zimmermann, and D. Manocha, "Parallel cartesian planning in dynamic environments using constrained trajectory planning," in *2015 IEEE-RAS 15th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2015, pp. 983–990.

[31] A. Orthey and M. Toussaint, "Visualizing local minima in multi-robot motion planning using multilevel morse theory," in *Workshop on the Algorithmic Foundations of Robotics*, 2020.

[32] M. Penrose, *Random geometric graphs*. OUP Oxford, 2003, vol. 5.

[33] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Batch informed trees (bit*): Informed asymptotically optimal anytime search," *The International Journal of Robotics Research*, vol. 39, no. 5, pp. 543–567, 2020.

[34] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.