Mixed-Integer Convex Programming for Motion Planning in Dual-Arm Manipulation

Karyna Volokhatiuk and Phone Thiha Kyaw
University of Toronto Institute for Aerospace Studies
Email: {karyna.volokhatiuk, phone.thiha}@robotics.utias.utoronto.ca

***Abstract*—Motion planning for high-degree-of-freedom robotic manipulators is inherently challenging due to constraints such as collision avoidance, kinematic limits, and trajectory smoothness. In this project, we propose formulating the motion planning problem as a convex optimization problem. Specifically, we leverage the recently studied Graph of Convex Sets (GCS) framework to model motion planning as a compact mixed-integer program. We also explore an extended formulation that incorporates trajectory duration, length and smoothness using Bézier curve parameterizations. Our method is evaluated through simulations on a 18-degree-of-freedom dual-arm manipulation task, and our findings highlight the key trade-offs between solution quality, computational cost, and the complexity of convex region construction in high-dimensional planning problems. All code and visualizations supporting our results are publicly available at: `github.com/utiasSTARS/ece1505-w25-project`.**

I. Introduction

Motion planning is a fundamental problem in robotics that involves computing a feasible and efficient trajectory for a robot to move from an initial configuration to a goal configuration while satisfying various constraints [1]. The problem becomes particularly challenging in high dimensional systems, which are generally known to be PSPACE-hard [2].

Classical approaches to motion planning rely heavily on sampling-based planners, such as Rapidly-exploring Random Trees (RRT) [3] and Probabilistic Roadmaps (PRM) [4], due to their ability to efficiently explore high-dimensional spaces. These planners have been widely adopted because they can handle complex environments without requiring an explicit representation of the free space. However, when additional constraints such as kinodynamic feasibility, obstacle avoidance, or motion continuity are introduced, these methods struggle to produce high-quality, optimal trajectories.

Recent advances in optimization-based motion planning have tackled these limitations by formulating the problem as a compact mixed-integer convex program [5]. One such approach, the Graph of Convex Sets (GCS) framework [6], enables the encoding of collision-avoidance constraints and system dynamics within a single convex optimization formulation. By leveraging convex relaxations and mixed-integer programming, GCS provides a structured way to computing globally optimal trajectories while ensuring feasibility in constrained environments.

A. Contributions

In this project, we explore the use of GCS framework for kinodynamic motion planning in dual-arm robotic manipulation (Figure 1). Our key contributions are as follows:

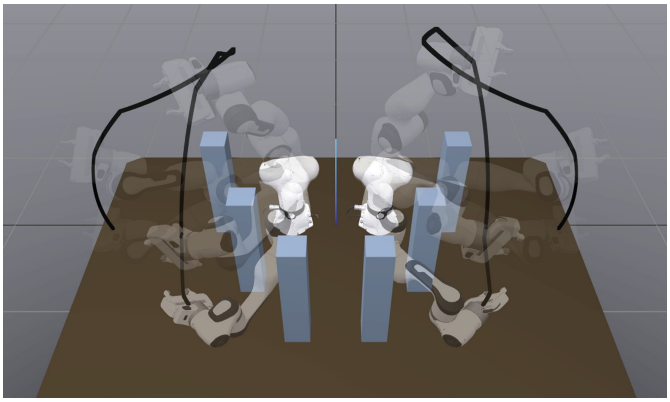


Fig. 1: A composite figure of two Franka Emika Panda robots executing a path found by our proposed convex optimization-based motion planning approach.

- We formulate the problem of generating *collision-free, kinodynamic motion plans* for a dual-arm manipulator as a mixed-integer convex program (MICP), following the approach in [5]. However, unlike prior work, we investigate the scalability of the GCS framework in significantly higher-dimensional settings—specifically, an 18-degree-of-freedom (DOF) dual-arm system operating in a more complex, cluttered environment with narrow passages.
- We conduct a comprehensive experimental study to evaluate the effectiveness of the optimization-based motion planning approach. Our experiments also include comparisons against recent sampling-based algorithms, analyzing critical performance metrics such as solution quality and computation time.

This report is organized as follows: Section II presents the problem formulation of our proposed approach. Section III discusses and analyzes the results from our simulation studies. Finally, Section IV concludes the report.

II. Problem Formulation

A. Problem statement

In this project, we formulate two problems. The simpler one involves using convex optimization to generate a trajectory of minimum Euclidean length between the initial and goal manipulator configurations. The more complex task extends this by also optimizing for trajectory duration and smoothness.

The first solution will be fairly compared against sampling-based methods, which typically do not optimize for trajectory duration or smoothness. The second solution will be evaluated independently to analyze the effect of different weightings for the various optimization components: trajectory length, duration, and smoothness.

B. IRIS

First, it is important to note that the method we explore—Graph of Convex Sets (GCS)—requires as input a set of overlapping, convex, and collision-free regions (safe regions). To generate these regions, we use IRIS (Iterative Regional Inflation by Semidefinite Programming) [7]. While the construction of obstacle-free convex sets is itself an interesting topic, it is not the focus of this project; for more details, please refer to the original IRIS paper.

In brief, the user specifies a collision-free starting configuration for the robot, and the IRIS algorithm generates a convex polytope of valid configurations that includes this starting point. It is the user's responsibility to choose starting configurations that yield overlapping convex sets sufficient for constructing the desired path.

C. Optimization problem: minimum-length trajectory

After constructing the safe regions using IRIS, we build a directed graph $G = (V, E)$. Each vertex $v \in V$ represents a collision-free convex region $\mathcal{X}_v$, and a directed edge $(u, v) \in E$ indicates that the corresponding regions $\mathcal{X}_u$ and $\mathcal{X}_v$ overlap ($u, v \in V$), allowing a transition from one to the other. Note that if $(u, v) \in E$, then $(v, u) \in E$. The length of each edge is variable and depends on two specific robot configurations, $\mathbf{x}_u$ and $\mathbf{x}_v$, chosen from the convex sets $\mathcal{X}_u$ and $\mathcal{X}_v$ that represent vertices U and V associated with the edge. The edge length is defined as the Euclidean distance between these two configurations: $\|\mathbf{x}_v - \mathbf{x}_u\|_2$. To enforce boundary conditions, two auxiliary vertices $s, g \in V$ are introduced, corresponding to the start and goal configurations. For convenience, we set $\mathcal{X}_s = \{\mathbf{x}_s\}$ and $\mathcal{X}_g = \{\mathbf{x}_g\}$.

In this problem, we aim to find the minimum-length trajectory between configurations $\mathbf{x}_s \in \mathcal{X}_s$ and $\mathbf{x}_g \in \mathcal{X}_g$. For this, we can define a graph path p from vertex s to vertex g as a sequence of distinct vertices $(v_0, \ldots, v_K)$, where $v_0 = s$ and $v_K = g$. The corresponding sequence of edges is denoted by $\mathcal{E}_p := \{(v_0, v_1), \ldots, (v_{K-1}, v_K)\}$. We denote the set of all possible paths from s to g as $\mathcal{P}$.

The optimization problem can then be formulated as:

$$
\begin{aligned}
\text{minimize} \quad & \sum_{e=(u,v)\in\mathcal{E}_p} \|\mathbf{x}_v - \mathbf{x}_u\|_2 \\
\text{subject to} \quad & p \in \mathcal{P}, \\
& \mathbf{x}_u \in \mathcal{X}_u, \mathbf{x}_v \in \mathcal{X}_v, && \forall u, v \in p, \\
& (\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_e, && \forall e = (u, v) \in \mathcal{E}_p.
\end{aligned} \tag{1}
$$

Here, the constraint $(\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_e$ is a convex constraint that couples the endpoints of edge $e = (u, v)$. For the minimum Euclidean length trajectory problem, we can specify these constraints in the following way: for all edges $e = (u, v) \in \mathcal{E}$, we define $\mathcal{X}_e$ through the conditions $\mathbf{x}_v \in \mathcal{X}_u \cap \mathcal{X}_v$ for $\mathbf{x}_u \in \mathcal{X}_u$, $\mathbf{x}_v \in \mathcal{X}_v$. This ensures that there is a collision-free path between two vertices.

At first glance, the problem in (1) may seem simple; however, its complexity lies in the need to not only minimize the total length of the trajectory but also determine which convex regions the path should traverse.

D. Mixed-integer convex programming (MICP)

The problem described in Section II-C can be formulated as mixed integer convex programming (MICP), a problem where certain decision variables are restricted to integers (usually binary), while others can be continuous. This combination leads to a non-convex optimization problem. Therefore, we follow the MICP formulation introduced by Marcucci et al. [6], which is based on the network-flow formulation and becomes convex after the relaxation step:

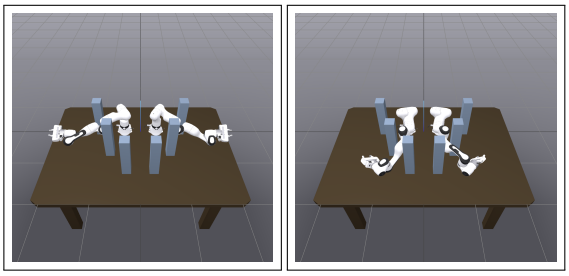
$$
\begin{aligned}
\text{minimize} \quad & \sum_{e\in\mathcal{E}} \widetilde{\ell}_e(\mathbf{z}_e, \mathbf{z}'_e, y_e) && (2.1)\\
\text{subject to} \quad & \sum_{e\in\mathcal{E}_s^{\text{out}}} y_e = 1 \quad \sum_{e\in\mathcal{E}_g^{\text{in}}} y_e = 1 && (2.2)\\
& \sum_{e\in\mathcal{E}_v^{\text{out}}} y_e \le 1 \quad \forall v \in V \setminus \{s, g\} && (2.3)\\
& \sum_{e\in\mathcal{E}_v^{\text{in}}} (\mathbf{z}'_e, y_e) = \sum_{e\in\mathcal{E}_v^{\text{out}}} (\mathbf{z}_e, y_e) \quad \forall v \in V \setminus \{s, g\} && (2.4)\\
& (\mathbf{z}_e, y_e) \in \widetilde{\mathcal{X}}_u, \quad (\mathbf{z}'_e, y_e) \in \widetilde{\mathcal{X}}_v \quad \forall e = (u, v) \in \mathcal{E} && (2.5)\\
& y_e \in \{0, 1\} \quad \forall e \in \mathcal{E} && (2.6)
\end{aligned}
$$

Here, y_e are the flows — decision variables that define if edge e is traversed by the path (if $y_e = 1$, then is traversed). Other decision variables are $\mathbf{z}_e = y_e \mathbf{x}_u$ and $\mathbf{z}'_e = y_e \mathbf{x}_v$, where e is an edge $e = (u, v)$; they represent the flow-weighted configuration of the vertex (either $\mathbf{x}_u \in \mathcal{X}_u$, or $\mathbf{x}_v \in \mathcal{X}_v$; $u, v \in V$) at one end of the edge.

The objective function (2.1) is a sum of perspective functions, which are based on the edge length. If we introduce a function ℓ_e as $\|\mathbf{x}_v - \mathbf{x}_u\|_2$ when $(\mathbf{x}_u, \mathbf{x}_v) \in \mathcal{X}_e$ and $\inf$ otherwise, the perspective function can be defined as:

$$
\tilde{\ell}_e(\mathbf{z}_e, \mathbf{z}'_e, y_e) = \begin{cases} \ell_e(\mathbf{x}_u, \mathbf{x}_v) y_e & \text{if } y_e > 0, \\ 0, & \text{otherwise.} \end{cases}
$$

The sets $\mathcal{E}_v^{\text{in}} := \{(u, v) \in \mathcal{E}\}$ and $\mathcal{E}_v^{\text{out}} := \{(v, u) \in \mathcal{E}\}$ represent the sets of edges incoming to and outgoing from vertex v, respectively. $\tilde{\mathcal{X}}_u$ and $\tilde{\mathcal{X}}_v$ are perspective cones of the convex sets $\mathcal{X}_u$ and $\mathcal{X}_v$, respectively. Constraints in 2.5 ensure that if $y_e = 1$ (edge e is part of the selected path), then $z_e = x_u \in \mathcal{X}_u$ and $z'_e = x_v \in \mathcal{X}_v$; and if $y_e = 0$, then $z_e = z'_e = 0$. The purpose of these cones is to switch on/off vertex participation in the path without breaking convexity. Finally, 2.4 ensures flow conservation.



(a) Start configuration (b) Goal configuration

Fig. 2: A dual-arm manipulation problem for two Franka Emika Panda robots in $\mathbb{R}^{18}$, simulated in Drake. The robots start in an initial configuration (a) positioned between the first two rows of obstacles and must reach the goal configuration (b) between the last two rows of obstacles without collisions.

To achieve convexity, Marcucci et al. [6] relax the binary constraints in (2.6), allowing $y_e \in [0, 1]$. This relaxed variable can be interpreted as the probability of edge e being included in the shortest path [5]. The resulting formulation is a Second-Order Cone Program (SOCP).

Finally, to get an approximate solution, [5] proposes to round probabilities using a randomized depth-first search with backtracking (see Section 4.2 in [5] for details). After this step we can also calculate an (overestimated) optimality gap of the rounded solution: $\delta_{\text{relax}} = (C_{\text{round}} - C_{\text{relax}})/C_{\text{relax}}$.

E. Extension of the optimization problem

Due to space constraints and the increased complexity arising from optimizing not only Euclidean distance but also time and smoothness, we will briefly outline the core ideas behind the formulation proposed in [5].

To enable this richer optimization, each trajectory is parameterized using two Bézier curves. Previously, a configuration variable $\mathbf{x}_v \in \mathcal{X}_v$ was associated with each vertex $v \in V$; now, each such variable corresponds to the set of control points for two Bézier curves. One curve represents the path (spatial component), while the other represents the time-scaling function.

The objective function is expressed as a weighted sum of three terms: the trajectory duration, its length, and the energy of the time derivative of the trajectory. The energy term promotes trajectory smoothness. Additional constraints are introduced to enforce a desired degree of differentiability, as well as to bound the total duration. Despite the increased number of constraints and the more complex objective, the Mixed-Integer Convex Programming (MICP) framework described in Section II-D remains applicable provided we suitably modify the edge cost function ℓ_e and incorporate the additional constraints into the graph edges. For a detailed discussion of the full formulation and solution approach, see [5].

III. Simulation Results

Our convex optimization-based motion planning approach is primarily built using Drake [8], a widely used toolbox for modeling, simulation and optimization of advanced robotic systems. We used Drake's implementations of the GCS framework and IRIS to decompose the robot's free configuration space into convex regions. Note that, to handle the mixed-integer programs generated by the GCS approach, we made use of the MOSEK optimization solver [9], which supports large scale mixed-integer convex programs.

TABLE I: Summary of GCS results for different problem sizes.

Metric	Small	Medium	Large
Relaxation cost	8.7629	8.1071	7.4669
Rounded cost (min)	8.7629	8.1423	8.1423
Rounded cost (max)	8.7629	8.1423	11.0726
Rounded cost (average)	8.7629	8.1423	8.7158
Optimality gap (min)	0.0000	0.0043	0.0904
Optimality gap (max)	0.0000	0.0043	0.4829
Optimality gap (average)	0.0000	0.0043	0.1673

The robot platform used to validate our approach is a 18-degree-of-freedom dual-arm manipulator as shown in Figure 2: each robotic arm comprises 7 joints, and its corresponding gripper includes 2 joints. The environment we designed resembles a cage, where the robot's task is to move from its initial configuration—positioned between the first two rows of obstacles—to a goal configuration located between the last two rows of obstacles. We purposefully constructed this setup to ensure that narrow passages appear in the high-dimensional configuration space of the robot, a known challenge for classical motion planners. All aspects of the robot's geometry, kinematics and collisions were modeled using Drake's built in simulation and collision checking engines.

In addition to our approach, we use sampling-based motion planners from the Open Motion Planning Library (OMPL) [10][1] as benchmarks for comparison. All parameters related to these sampling-based planners were carefully reviewed and adjusted only when necessary to ensure a fair comparison. Due to space constraints, detailed configurations are omitted from this report, and interested readers can refer to our publicly available repository for more information.[2]

A. Analysis of the minimum-length trajectory problem

We tested a version of GCS that minimizes the Euclidean length of the trajectory on graphs of three different sizes: 6, 12, and 18 vertices (see Appendix A-A for more details). Each larger graph contains all the vertices of the smaller ones, along with additional vertices. While it may seem counterintuitive, a larger graph—with more possible configurations—does not necessarily lead to a better (i.e., lower) objective value. For the smallest graph, the feasible objective value after the rounding step is 8.76; for the medium graph, it is 8.14; and for the largest, it is 8.14 (see Table I) .

This discrepancy arises from the relaxation used in the problem formulation and the need to apply a rounding step to convert the relaxed solution into a feasible one. Since we

[1]We used the latest release version 1.7.0 of OMPL for all the experiments.
[2]https://github.com/utiasSTARS/ece1505-w25-project

employed a randomized rounding method (specifically, a randomized depth-first search with backtracking), we conducted 30 experiments for each graph size to evaluate performance. To interpret the table results correctly, it is important to note that the optimality gap is evaluated only with respect to the chosen convex sets. Thus, zero optimality gap only means that the solution is optimal with respect to the given graph. For example, although the optimality gap for the smallest graph appears to be zero, the true optimal cost may actually be lower than the obtained solution, since the IRIS convex sets only approximate the full solution space.

As expected, the relaxation cost is the lowest for the largest graph, this is due to the greater number of possible robot configurations. The relaxation is tight for the smallest graph. In the medium-size graph, the relaxation is not tight, but interestingly, the outcome after rounding remains constant across all 30 experiments. The most variability is observed with the largest graph: although its relaxation cost is the lowest, the rounded cost never outperforms that of the medium graph. Furthermore, the rounded cost in the largest graph varies significantly between experiments, sometimes greatly increasing the initially small relaxation cost.

Additionally, it is important to note that increasing the graph size leads to a significant increase in computation time: GCS takes approximately 0.52 seconds to solve for the smallest graph, 9.89 seconds for the medium graph, and 33.64 seconds for the largest graph (see Figure 3). Thus, the GCS approach has a scalability limitation, at least when using for highly cluttered environments.

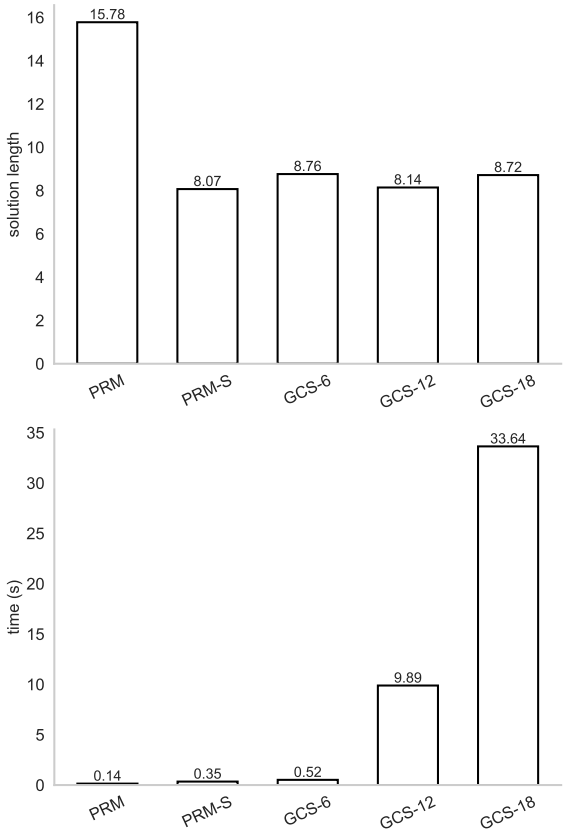


Fig. 3: Average comparison results of the tested algorithms for the dual-arm manipulation problem. The top figure shows the solution quality (measured by path length) across 30 trials for each algorithm. The bottom figure presents the corresponding execution times.

B. Comparison with sampling-based planners

We also conducted experiments to compare our approach with sampling-based methods, which are standard algorithms used in many high-dimensional motion planning applications. The most suitable candidate for benchmarking against ours is the multi-query sampling-based planner, PRM, as it supports precomputation—these planners build a roadmap (graph) prior to planning to accelerate the solution search. Additionally, we applied path shortcutting techniques to the solution paths found by PRM to improve path quality, with some tradeoff in execution time. Specifically, we used the path shortening method proposed in the work of RRT-Rope [11]. Each algorithm was run for 30 trials, and we measured the average solution quality (in terms of path length) and computation time. The results are presented in Figure 3. Here, PRM-S denotes PRM with path shortcutting enabled, while GCS-6, GCS-12, and GCS-18 represent our approach using small, medium, and large convex set sizes of 6, 12, and 18, respectively.

In terms of solution quality, we observed that PRM performs the worst among all tested algorithms. This result is also expected, given the known limitations of random sampling-based methods. In contrast, PRM-S and all GCS variants with different set sizes produce comparable solution qualities. Notably, PRM-S achieves the best solution cost among the tested methods, including ours. These PRM-S results are significantly better than the results presented in [5], and demonstrate that our convex optimization-based approach can still compete with state-of-the-art path shortening techniques in terms of solution quality. We also observed that GCS-12 outperforms the other two GCS variants.

On the other hand, PRM found solutions fastest, with an average computation time of approximately 140 ms, followed by PRM-S at 350 ms and GCS-6 at 520 ms. GCS-12 and GCS-18 took significantly longer, requiring between 9.89 and 33.64 seconds to find a solution. These results highlight the importance of selecting an appropriate number and size of convex sets to effectively trade off between solution quality and execution time.

C. Analysis of the extension of the problem

First, it is important to note that the extended version of GCS runs slower than the version that only optimizes trajectory length. For a graph with six vertices, the extended version is approximately twice as slow. This is due to the use of Bézier curves, which increase the number of configurations optimized at each vertex.

However, this extension provides capabilities not available in traditional geometric sampling-based methods: it allows for control over the smoothness, differentiability, and time optimality of the trajectory. The extended optimization problem

introduces three tunable weights: (a) trajectory duration, (b) trajectory length, and (c) the energy of the time derivative of the trajectory. In addition, it is possible to enforce a constraint on the required degree of trajectory differentiability.

We experimented with different combinations of the weights for (a), (b), and (c), while ensuring the resulting trajectory was continuously differentiable (once):

- Prioritizing (a) results in a high-velocity trajectory with minimal sharp turns.
- Prioritizing (b) yields a shorter trajectory with sharper direction changes, making it more difficult to follow quickly.
- Prioritizing (c) produces a very smooth but slower trajectory.

IV. Conclusion and Discussion

In this project, we explored the use of the Graph of Convex Sets (GCS) framework for motion planning in high-dimensional dual-arm robotic manipulation tasks. By formulating the planning problem as a mixed-integer convex program (MICP), we demonstrated that feasible and near-optimal trajectories can be generated even in complex and cluttered environments.

While GCS requires greater setup effort and computational resources, it offers free certificates of optimality for complex motion planning tasks. This provides valuable insight into solution quality and serves as a theoretical benchmark for evaluating performance. We also show that careful selection of convex, collision-free regions has a significant impact on both the solution quality and runtime of the algorithm. This suggests promising directions for future work in exploring alternative approaches for handling an increasing number of convex sets. Overall, convex optimization-based motion planning proves to be a powerful tool for planning in complex environments—particularly when solution quality and theoretical guarantees are prioritized.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] J. H. Reif, "Complexity of the mover's problem and generalizations," in *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*. IEEE Computer Society, 1979, pp. 421–427.

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

[4] 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.

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

[6] T. Marcucci, J. Umenberger, P. Parrilo, and R. Tedrake, "Shortest paths in graphs of convex sets," *SIAM Journal on Optimization*, vol. 34, no. 1, pp. 507–532, 2024.

[7] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," *Springer Tracts in Advanced Robotics*, vol. 107, pp. 109–124, 04 2015.

[8] R. Tedrake and the Drake Development Team, "Drake: Model-based design and verification for robotics," 2019. [Online]. Available: https://drake.mit.edu

[9] M. ApS, *The MOSEK optimization toolbox for MATLAB manual. Version 9.0.*, 2019. [Online]. Available: http://docs.mosek.com/9.0/toolbox/index.html

[10] 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.

[11] L. Petit and A. L. Desbiens, "Rrt-rope: A deterministic shortening approach for fast near-optimal path planning in large-scale uncluttered 3d environments," in *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2021, pp. 1111–1118.

Appendix A
Implementation Details

A. Construction of convex regions with IRIS

Following the approach in [5], we manually selected a set of robot configurations around which IRIS-generated convex regions were constructed. We used the `IrisInConfigurationSpace` implementation from Drake. Given the complexity of the benchmark task—with 18 degrees of freedom and a densely cluttered environment—this manual process was challenging but necessary. Automatic region generation would have been more time-consuming, as it would attempt to build more convex regions around a larger number of configurations.

The most challenging aspect of the manual process was ensuring that the resulting convex regions formed a connected graph that admitted a collision-free path from the start to the goal configuration. Often, configurations we selected led to disconnected graphs. To address this, we initially generated convex regions around 18 manually chosen configurations, then extracted a minimal, cycle-free subgraph of 6 connected regions linking the start and goal. To evaluate scalability and performance in more complex settings, we also tested larger graphs with 12 and 18 vertices. Since the same graph is intended to support multiple planning tasks, it may be significantly larger and more complex, therefore it is important to ensure that the motion planning algorithm (GCS) remains effective and scalable in such settings.

The time required to generate IRIS convex regions was significant. Generating 6 regions took 109.66 minutes. Expanding this set to 12 regions—including the original 6 and 6 additional ones—took 351.84 minutes. Generating all 18 regions took 511.05 minutes. These measurements reflect sequential execution. While IRIS region generation is inherently parallelizable (one region per thread), the runtime for each region can vary widely. In our experiments, the fastest region was generated in 4.66 minutes, while the slowest took 75.37 minutes.

B. Precomputation of PRM roadmaps

We also precomputed and stored the graph (roadmap) generated by the PRM similar to using IRIS, so that during online planning, the algorithm only needs to perform a graph search. For this experiment, we ran the PRM planner for 10 minutes (600 seconds) for precomputation. We also tested longer durations—20 minutes, 30 minutes, etc.—but found that they did not significantly improve the solution quality. Instead, they increased the planning time because of the larger number of vertices added to the graph. This behavior is expected given the nature of PRM, as it is not an anytime algorithm and does not guarantee better solutions with more computation time.