Rigid Body Path Planning Using Mixed-Integer Linear Programming

Mingxin Yu and Chuchu Fan, *Member, IEEE*

***Abstract*—Navigating rigid body objects through crowded environments can be challenging, especially when narrow passages are presented. Existing sampling-based planners and optimization-based methods like mixed integer linear programming (MILP) formulations, suffer from limited scalability with respect to either the size of the workspace or the number of obstacles. In order to address the scalability issue, we propose a three-stage algorithm that first generates a graph of convex polytopes in the workspace free of collision, then poses a large set of small MILPs to generate viable paths between polytopes, and finally queries a pair of start and end configurations for a feasible path online. The graph of convex polytopes serves as a decomposition of the free workspace and the number of decision variables in each MILP is limited by restricting the subproblem within two or three free polytopes rather than the entire free region. Our simulation results demonstrate shorter online computation time compared to baseline methods and scales better with the size of the environment and tunnel width than sampling-based planners in both 2D and 3D environments.**

***Index Terms*—Motion and path planning, formal methods in robotics and automation.**

I. Introduction

THE *Piano Mover's Problem* [1] asks whether one can find a sequence of rigid body motions from a given initial position to a desired final position, subject to certain geometric constraints during the motion. The difficulty in planning for point objects comes from the dimension of the objects, which prevents the feasibility of naively tracking the motion of a point, especially when narrow corridors are presented in the environment and an associated higher-dimension configuration space of $SE(2)$ or $SE(3)$ rather than $\mathbb{R}^2$ or $\mathbb{R}^3$. This problem has inspired many motion planning works [2], [3], [4], [5], mainly streamed as sampling-based and optimization-based methods.

Sampling-based methods like probabilistic roadmap (PRM) [2] are widely adopted because of their simplicity. The approaches employ discrete collision detection, which is straightforward and efficient but highly dependent on the chosen resolution. Setting improper parameters can lead to missed obstacles, resulting in invalid paths. On the other side, solving an obstacle avoidance problem can be formulated as optimization problems for optimality and safety guarantee, such as trajectory optimization using Mixed-Integer Linear Programming (MILP) [3], [6] or Graphs of Convex Sets (GCS) [7]. They offer a more robust solution by directly incorporating collision avoidance into the formulations. These methods, unlike sampling-based methods, ensure paths being collision-free by design, thus bypassing the dependency of resolution in collision detection.

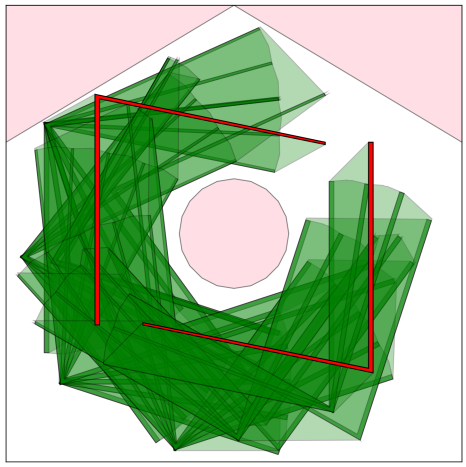


Fig. 1. Figure demonstrates an example solved with our MILP-based method, navigating a non-convex, V-shaped object through a complex environment from the start configuration to the goal configuration, both marked red. In the figure, the obstacles are shown in pink, while the green depicts the regions swept by the object, demonstrating the effectiveness and precision of our approach in challenging environments.

Despite their advantages, current optimization formulations come with their own set of challenges. Solving MILPs, for example, is an NP-hard problem and can be computationally intensive as the environment complexity increases. The time to directly solve a MILP for a feasible path grows exponentially with the number of waypoints, which is proportional to the required maneuvers in an environment. This scalability issue makes brute-force MILP-based methods difficult to apply. On the other hand, the effectiveness of GCS depends on precise initial seeding to capture narrow passages within the configuration space, which brings feasibility challenges to GCS methods. To address these inefficiencies, our approach, as shown in Fig. 1 simplifies the problem by decomposing a single large MILP into manageable, fixed-size smaller MILPs. Additionally, by focusing directly on the workspace rather than the configuration space, our method mitigates the challenges associated with identifying critical narrow passages, thereby enhancing efficiency and scalability in path planning.

Received 17 May 2024; accepted 13 September 2024. Date of publication 25 September 2024; date of current version 4 October 2024. This article was recommended for publication by Associate Editor K. Leahy and Editor L. Pallottino upon evaluation of the reviewers' comments. The work of Mingxin Yu was supported by the Mathworks Fellowship. This work was supported in part by the National Science Foundation (NSF) CAREER Award under Grant CCF-2238030 and in part by MIT-Ford Alliance Program. *(Corresponding author: Mingxin Yu.)*

The authors are with the Department of Aeronautics and Astronautics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: yumx35@mit.edu; chuchu@mit.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2024.3468165, provided by the authors.

Digital Object Identifier 10.1109/LRA.2024.3468165

2377-3766 © 2024 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

In this work, our pipeline is structured into three key stages: *Workspace decomposition* - we first construct a graph of convex set G_c covering the free workspace. This stage is performed offline and the graph is reusable for multiple queries across different objects. We exploit the lower dimensionality of the workspace compared to the configuration space to obtain a higher coverage ratio and fewer narrow tunnels, which are typically challenging to identify and cover. *Path segment validation* - we then construct a graph G_d in configuration space, which is also offline. Here, the nodes represent bottleneck configurations identified by G_c and the edges are viable paths validated through MILPs. This graph allows multiple queries for different start and goal configurations. At the *online* stage, we connect the start and goal configuration into G_d and retrieve the solution path. Our experiment results show that this method significantly reduces online time consumption across various objects and environments, in both 2D and 3D settings.

This work presents several key contributions: (1) By operating directly within the workspace, our method effectively mitigates the challenges associated with covering narrow tunnels in the free space, compared to configuration space-based approaches. (2) We simplify the path planning process by breaking down a large, complex optimization problem into a series of smaller MILPs. Each smaller problem focuses on finding a valid path segment within a region, significantly enhancing the scalability of our approach. (3) We conduct comprehensive experiments in both 2D and 3D environments, comparing against GCS and sampling-based methods. Our results demonstrate scalability with the environment, higher feasibility, and shorter computation time.

II. Related Works

Graph-based methods: Graph-based methods are particularly suitable for scenarios requiring multiple queries within the same environment. Once the graph is constructed, it can be reused to find multiple paths. One category builds the graph of discrete configurations, like sampling-based methods PRM [2] and lattice planners [5]. PRM randomly samples states in C-space as nodes in the graph. Nodes are connected based on the feasibility of direct paths. Some variants of PRM also consider utilizing workspace information to guide sampling in configuration space [8], [9], therefore accelerating the exploration. Lattice planners are commonly used in nonholonomic vehicle parking scenarios and leverage high levels of parallelization [5], [10]. They use a regular grid in the configuration space to define graph nodes [11]. And the edges are computed offline [12], representing motions that adhere to specific constraints. However, one inherent limitation of lattice planners is their scaling with grid resolution. Employing regular lattices across the entire configuration space can be computationally expensive and inefficient. In contrast, our method reduces the size of the graph by only utilizing a regular grid on a set of manifolds within the configuration space, where the metric is strictly zero.

Another category of graph-based planners relies on space decomposition, where the graph explicitly represents the free space. Based on constrained Delaunay triangulation [13], previous works perform well in 2D environments for point objects [3], [14] either through search or by solving MILPs. Some works have addressed the problem for higher DoF robots by incorporating potential field after decomposition for collision avoidance [15] in less-cluttered 3D environments. While these approaches provide a foundation for solving the problem, they may struggle with highly constrained environments or complex object geometries. GCS line of works [7], [16], [17] adopts a different approach by attempting to cover the free space with a set of convex polytopes, thereby retrieving paths through optimization. This method is capable of working in both workspace and configuration space, thus accommodating any shape and scaling to more than 10 dimensions.

Algorithm 1: Construct a Graph of Convex Polytopes G_c.

Require: list of obstacles $\mathcal{O}_{\text{obs}}$, number of samples for visibility graph n_v, number of samples per iteration n_s, coverage threshold α.

$\mathcal{P} \leftarrow \emptyset$
$G_v \leftarrow \text{SampleVisibilityGraph}(\mathcal{O}_{\text{obs}}, n_v)$
while $\text{CheckCoverage}(\mathcal{E}_v, \mathcal{P}) < \alpha$ **do**
  $\mathcal{S} \leftarrow \text{SampleVisibilityEdge}(\mathcal{E}_v, \mathcal{P}, n_s)$
  $\mathcal{P} \leftarrow \mathcal{P} \cup \text{Iris}(\mathcal{S})$
end while
$G_c \leftarrow \text{ConstructGraph}(\mathcal{P})$
return G_c

Exact collision detection Most path-planning algorithms work with discrete collision detection, due to their easy implementation, broad applicability, and fast execution [2], [18]. This method checks a path in C-Space by creating samples along the path and when all these samples are checked to be collision-free, the entire path is assumed to be collision-free. While efficient, its accuracy depends on the sampling resolution, risking missed collisions with coarse samples or delays with overly fine-grained samples. To enhance reliability, one method is free bubble [19], recursively bisecting the samples on the path to guarantee collision-free [20] by calculating the maximum allowable movement without collision. Another method is continuous collision checking [21], [22], which performs well for rigid body motions by directly checking the collisions of the reachable set with the obstacles. But they suffer from the speed. We integrate both methods in the algorithm, aiming for both soundness and efficiency.

III. Pipeline Overview

In this letter, we consider the path planning problem of a rigid body object O. The workspace $\mathcal{W}$, the physical space where the object lies, can be $\mathbb{R}^2$ or $\mathbb{R}^3$ depending on the particular application at hand. And the configuration space for O is $SE(2)$ and $SE(3)$, respectively. A configuration is represented as $q = (p, R)$ with a position vector $p \in \mathbb{R}^2$ or $\mathbb{R}^3$ and a rotation matrix $R \in SO(2)$ or $SO(3)$. The pipeline of our method is shown in Fig. 2. We outline major stages of our method in this section, before going deep into verifying path segments with MILP in Section IV.

A. Workspace Decomposition - Construct Coarse Graph G_c

At this stage, we aim to approximately decompose the free workspace $\mathcal{W}_{\text{free}}$ into a set of convex polytopes $\mathcal{P}$ and build a graph G_c on $\mathcal{P}$ as shown in Algorithm 1. The graph G_c is shared across various objects within the same environment.

Following [23], we first construct a visibility graph $G_v := (\mathcal{V}_v, \mathcal{E}_v)$ where $\mathcal{V}_v$ is uniformly sampled from $\mathcal{W}$ with points inside obstacles excluded, and $\mathcal{E}_v$ is added by checking for

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

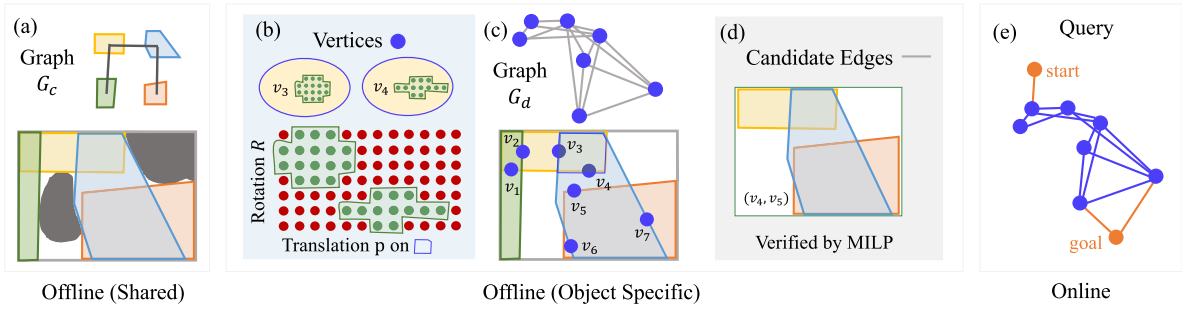


Fig. 2. Overview of our pipeline. (a) *Workspace decomposition:* Decomposition of the free workspace into a graph of convex polytopes G_c. The polytopes serve as graph vertices and are interconnected if they overlap (Section III-A). (b-d) *Path segment validation:* Construction of G_d(Section III-B), where each vertex is a set of interconnected free configurations - illustrated as the set of green points enclosed by the green ring. The motions between vertices (blue edges) are generated by solving MILPs (Section III-B). (e) *Online query:* the start and end configurations are connected to graph G_d and the planned path is retrieved (Section III-C).

collisions along the line segments connecting each pair of points within some distance using Proposition 1. During each iteration, the subroutine SAMPLEVISIBILITYEDGE samples n_s points on the edges in $\mathcal{E}_v$ that are not yet covered by the polytopes in $\mathcal{P}$. The n_s points are then used as initial points for IRIS [16], after which the newly computed polytopes are added into $\mathcal{P}$. The iteration terminates when the coverage ratio of $\mathcal{P}$ over $\mathcal{E}_v$ exceeds threshold $\alpha > 0$. Subsequently, an undirected graph $G_c := (\mathcal{V}_c, \mathcal{E}_c)$ is constructed with vertices $\mathcal{V}_c = \mathcal{P}$, and with an edge $(P_{i_1}, P_{i_2}) \in \mathcal{E}_c$ for every pair of intersected polytopes P_{i_1} and P_{i_2}.

B. Path Segment Validation - Construct Dense Graph G_d

Graph G_c is a GCS, and an optimal solution of the path planning problem for a point object can be found via [17]. However, its direct applicability is limited when considering non-point objects. The limitation arises because an intersection between polytopes - a valid pathway for point objects - does not inherently guarantee feasible traversal for objects with non-negligible dimensions and orientations.

To address this challenge, we introduce another undirected graph $G_d := (\mathcal{V}_d, \mathcal{E}_d)$ that describes the connectivity between adjacent polytopes for a specific object O, detailed in Algorithm 2. Our key insight is that, for an object O to move from P_i to P_j, its center c must traverse the boundary ∂P_{ij} of the intersection $P_{ij} := P_i \cap P_j$.

Vertices $\mathcal{V}_d$: The vertices $\mathcal{V}_d$ in G_d are generated through subroutine SAMPLE&GROUP. The process begins with discretizing ∂P_{ij} into regular grids.

a) Discretization in 2D environments. For 2D, the boundary ∂P_{ij} is a closed ring, which can be parameterized linearly from $\lambda = 0$ to $\lambda = 1$, with the points for $\lambda = 0$ and $\lambda = 1$ being identical. Translations along this boundary are selected with a fixed interval $\delta\lambda > 0$. Rotations are discretized into $n_R > 0$ possible values, with angles $(\theta_1, \ldots, \theta_{n_R})$ being $(1 \cdot 2\pi/n_R, 2 \cdot 2\pi/n_R, \ldots, n_R \cdot 2\pi/n_R)$. These correspond to rotation matrices $(R_1, \ldots, R_{n_R})$. Together, the translation parameter λ and the set of rotations θ create a 2D grid of possible configurations, as shown in Fig. 2.

b) Discretization in 3D environments. In 3D scenarios, the boundary ∂P_{ij} consists of several facets, each treated as a disjoint boundary without considering the connections between them. On each facet, the translations are constructed using a regular rectangle grid, and the rotations are a selected set of rotation matrices $(R_1, \ldots, R_{n_R})$. Together, the translations and rotations create a 3D grid on a facet.

Algorithm 2: Construct a Graph of Convex Polytopes G_d.

Require: list of obstacles $\mathcal{O}_{\text{obs}}$, graph G_c, number of intermediate waypoints N.

$\mathcal{V}_d \leftarrow \emptyset, \mathcal{E}_d \leftarrow \emptyset$
for $(P_i, P_j) \in \mathcal{E}_c$ **do**
  $v_{ij,1}, \ldots, v_{ij,n_{ij}} \leftarrow$ SAMPLE& GROUP$(\partial P_{ij}, \mathcal{O}_{\text{obs}})$
  $\mathcal{V}_d \leftarrow \mathcal{V}_d \cup \{v_{ij,1}, \ldots, v_{ij,n_{ij}}\}$
end for
CandidateEdgeSet $= \{(v_{ij,n_1}, v_{ik,n_2}) | v_{ij,n_1}, v_{ik,n_2} \in \mathcal{V}_d, v_{ij,n_1} \neq v_{ik,n_2}\}$
for $(v_{ij,n_1}, v_{ik,n_2}) \in$ CandidateEdgeSet **do**
  if VERIFYTRAVERSAL$(v_{ij,n_1}, v_{ik,n_2}, G_c, N)$ **then**
    $\mathcal{E}_d \leftarrow \mathcal{E}_d \cup \{(v_{ij,n_1}, v_{ik,n_2})\}$
  end if
end for
return $\mathcal{V}_d, \mathcal{E}_d$

After discretization, we collect the set of free configurations $\mathcal{C}_{ij}$ on all grids, where a free configuration refers to the object being completely contained inside $P_i \cup P_j$. We then try to group free configurations on the same grid, forming of several disjoint subgraphs $\{G_{p,ij,n}\}_{n=1}^{n_{ij}}$, as shown in Fig. 2. Within each subgraph, any two configurations can be interconnected via a collision-free path. This process is detailed in the supplementary materials.[1] Subsequently, the set of vertices $\mathcal{V}_{p,ij,n} \subset \mathcal{C}_{ij}$ of n-th subgraph $G_{p,ij,n}$ on ∂P_{ij} becomes a node $v_{ij,n} = \mathcal{V}_{p,ij,n} \in \mathcal{V}_d$ to serve as potential waypoints in detailed path planning.

Edges $\mathcal{E}_d$: We assess all possible connections among $\mathcal{V}_d$ to establish $\mathcal{E}_d$. For any two vertices v_{ij,n_1} and v_{ik,n_2}, an edge is considered if the proposed traversal between vertices is verified as feasible by VERIFYTRAVERSAL. The edge (v_{ij,n_1}, v_{ik,n_2}) represents the feasibility of moving from polytope P_j to P_k via P_i.

C. Online Query

Graph G_d serves as an offline pre-computed roadmap, enabling rapid online path planning from a start configuration q_{start} to an end configuration q_{end}. We introduce new vertices $v_{\text{start}} = \{q_{\text{start}}\}$ and $v_{\text{end}} = \{q_{\text{end}}\}$ into G_d, connecting them using the

[1] See: https://sites.google.com/view/realm-rigidmilp

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

same VERIFYTRAVERSAL subroutine. The resulting path consists of alternating segments: inter-vertex motions and intra-vertex motions. The path query first tries to extract a sequence of vertices from v_{start} to v_{end} on G_d. Inter-vertex motions are retrieved from the connecting edges. For each vertex $v_{ij,n}$ to be visited, the intra-vertex motions are obtained from the corresponding subgraph $G_{p,ij,n}$, to connect entry and exit configurations. All motion retrieval is performed online using Dijkstra's algorithm.

IV. VERIFYING PATH SEGMENTS

A significant challenge arises in verifying the motion between waypoints due to the complex geometry of the objects. Previously, this is commonly addressed by bloating the obstacles to account for the size of objects, thereby transforming the problem into planning within this bloated environment as with a point object [14]. However, this method can be overly restrictive, especially for objects whose shapes deviate significantly from circular or spherical forms.

In this section, we will tackle the challenge using MILP to verify path segments for polytope objects. We assume the rigid-body object O to be simply connected (no holes) and its geometric center to lie inside the O. We denote the vertices of O are $\{v_{O,i}\}$.

A. MILP Formulation

As discussed in Section III-B, the traversal between vertices (v_{ij,n_1}, v_{ik,n_2}) in G_d is not straightforward. Therefore, we formulate the subroutine VERIFYTRAVERSAL as an MILP, which is to verify whether there exists a piece-wise linear path from a configuration in v_{ij,n_1} to another one in v_{ik,n_2} with N intermediate waypoints.

For the sake of simplicity, we'll denote v_{ij,n_1} as u and v_{ik,n_2} as w in the discussion of MILP, while denoting the set of polytopes $\{P_i, P_j, P_k\} \subset \mathcal{E}_c$ as $\mathcal{P}_c$. The list of free configurations in u or w are denoted as $(q_{u,1}, \ldots, q_{u,n_u})$ and $(q_{w,1}, \ldots, q_{w,n_w})$. Let q_t be the waypoints in the path, including the start and end configurations, indexed by $t \in \{0, \ldots, N+1\}$. Each waypoint q_t consists of a translational part p_t and a rotational part R_t. The translational part in a 2D scenario is given by $p_t = (x_t, y_t)$, and in 3D, it extends to $p_t = (x_t, y_t, z_t)$.

Model: The objective is to minimize the total 1-norm of the translational displacement along the path. The entire MILP model is formulated in (1):

$$\min_{\{p_t\}_t, \{R_t\}_t, \mathcal{B}} \quad \sum_{t=0}^{N} \|p_{t+1} - p_t\|_1, \tag{1a}$$

$$\text{s.t.} \quad (2), (3), (4), (5). \tag{1b}$$

Here $\mathcal{B}$ is the set of all binary variables we'll introduce. The objective in (1a) can be easily transformed to standard linear expressions with additional variables, which we will not detail here.

Basic constraints: The rotational part R_t is encoded as a one-hot vector $\beta_{R,t}$ using binary variables. This is achieved through the constraints

$$R_t = \sum_{i=1}^{n_R} \beta_{R,t,i} R_i, \qquad \forall t = 0, \ldots, N+1 \tag{2a}$$

$$1 = \beta_{R,t}^T \mathbf{1}, \qquad \forall t = 0, \ldots, N+1 \tag{2b}$$

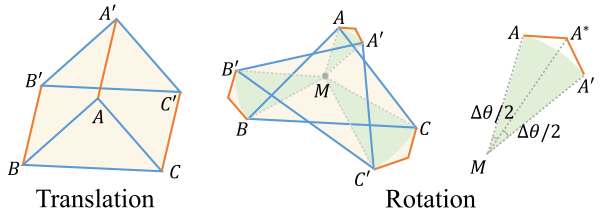


Fig. 3. Figure shows the reachable set of 2D objects for translation and rotation scenarios, respectively. On the right, we illustrate the approximation of the green sectors to yellow polytopes.

Each element in R is a possible rotation matrix, as specified in Section III-B. The index where $b_{R,t} = 1$ indicates the selection of the corresponding rotation at waypoint t.

Between two consecutive waypoints, we enforce the object to either translate or rotate, a decision encoded by binary variable $\beta_{\text{or},t}$ in (3). This constraint addresses the challenges associated with encoding collision-free conditions in MILPs. By restricting the motion to either translation or rotation at each step, this simplified model becomes computationally tractable. Specifically, in 3D scenarios, we further limit the object to translational motion only, equivalent to setting $\beta_{\text{or},t} = 1$.

$$\|p_{t+1} - p_t\|_1 \le M \cdot \beta_{\text{or},t}, \qquad \forall t = 0, \ldots, N \tag{3a}$$

$$\|\beta_{R,t+1} - \beta_{R,t}\|_1 \le M(1 - \beta_{\text{or},t}), \qquad \forall t = 0, \ldots, N \tag{3b}$$

$$\beta_{\text{or},t} = 1, \ \text{(only present in 3D)} \qquad \forall t = 0, \ldots, N \tag{3c}$$

The constraints for start and end configuration are that the first and last waypoints are in u and w correspondingly. We introduce two additional one-hot vectors β_{start} and β_{end}, which are used to select specific configurations from the sets u and w, shown in (4).

$$p_0 = \sum_{i=1}^{n_u} \beta_{\text{start},i} p_{u,i}, \qquad p_{N+1} = \sum_{i=1}^{n_w} \beta_{\text{end},i} p_{w,i}, \tag{4a}$$

$$R_0 = \sum_{i=1}^{n_u} \beta_{\text{start},i} R_{u,i}, \qquad R_{N+1} = \sum_{i=1}^{n_w} \beta_{\text{end},i} R_{w,i}, \tag{4b}$$

$$\sum_{i=1}^{n_u} \beta_{\text{start},i} = 1, \qquad \sum_{i=1}^{n_w} \beta_{\text{end},i} = 1 \tag{4c}$$

B. Collision-Avoidance Constraint

The remaining constraint in the MILP is collision avoidance. While the start (q_0) and end (q_{N+1}) configurations are already checked to be collision-free, ensuring continuous collision avoidance throughout the motion path is critical. Here, the collision avoidance constraint is encoded by enforcing that the each reachable set between two consecutive waypoints (q_t, q_{t+1}) $(t = 0, \ldots, N)$ be completely contained within the union of all relevant collision-free polytopes, $\bigcup \mathcal{P}_c$. Specifically, this is achieved by checking all the boundaries of the reachable sets being inside $\bigcup \mathcal{P}_c$.

1) Compute Boundary of Reachable Set: In 2D, the reachable set is computed as Fig. 3. For translation, the boundary contains the blue edges of O at configurations q_t and q_{t+1}, and orange lines connecting a same vertex at two waypoints. For rotation, the difference is that the lines connecting a same vertex at two waypoints are arcs. While the arcs can not be expressed

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

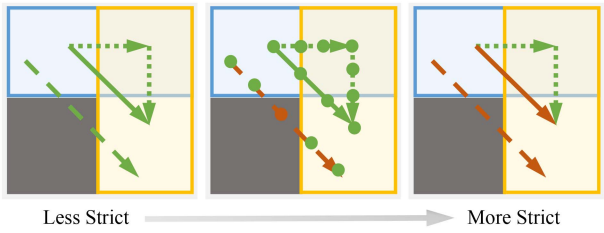


Fig. 4. Models of collision avoidance around obstacles (gray). The free space is decomposed into convex polytopes (blue and yellow). Green lines indicate predicted collision-free paths and red lines indicate detected collisions. *Left:* Less strict encoding that only checks endpoints of subdivided line segments. The dashed red line demonstrates that endpoint-only checking is insufficient as it cuts through an obstacle. *Right* [6]: This overly restrictive model allows only paths like the dotted line where both endpoints lie within the same free region, rejecting the simpler, viable solid line. *Middle (***ours***):* Strict encoding based on Proposition 1. This approach prevents the dashed line from cutting through obstacles by incorporating additional constraints beyond endpoint checking, while including more paths between regions, like the solid line.

in linear expression, we overapproximate the arc with two line segments, with $\|MA^*\|\cos(\Delta\theta_{\max}/2) = \|MA\|$. $\Delta\theta_{\max}$ is the maximum rotation angle allowed in one step, selected as $\pi/3$. In 3D, only translation is allowed inside MILP. The rotation is only allowed inside each V_d vertices, which is not solved by MILP. The reachable set is the union of the polytopes swept by each face of O, which is also a polytope. Therefore, to check the collisions of the faces of the reachable set, we only need to check the superset - the union of the triangle mesh faces of O at q_t and q_{t+1} and the parallelogram swept by each edge of O.

2) Collision Detection Between Surface and Polytope: Compared with existing models of collision avoidance in Fig. 4, we propose a new encoding based on the following corollaries. We just need to check each edge on the boundary of the reachable set to satisfy Proposition 1.

Proposition 1 (Line segment inside two convex polytopes): Given two convex polytopes $P_i = \{x|A_i x \le b_i\}$, $i = 1, 2$, and a line segment l with two endpoints (x_a, x_b), l is contained inside the union of P_1 and P_2 if one of following conditions holds:

1) x_a and x_b are within a same polytope P_i,
2) x_a and x_b are within different polytopes, but there exists a point x satisfying $x \in (P_1 \cap P_2)$.

Proof: For condition (1), the convexity of P_i ensures that all points on l, being a line segment between x_a and x_b, are also contained within P_i. If condition (2) holds, line segments (x_a, x) and (x, x_b) both satisfy condition (1). Hence, the entire line segment l remains within $P_1 \bigcup P_2$. □

We further extend the Proposition 1 from checking line segments to triangles and convex quadrilaterals. By enforcing all edges of these shapes satisfying Proposition 1, we prove that the shapes are contained within the union of two given convex polytopes. The corollaries and proofs are provided in the supplementary materials.[2] So we just need to check the line segments within the sets $\mathcal{S}_{\text{line}}$ based on Proposition 1:

1) the line segments connecting the object's geometric center and vertices at waypoints q_t and q_{t+1}
2) the edges of O at waypoints q_t and q_{t+1}
3) the line segments connecting a same vertex of O between waypoints q_t and q_{t+1}.

[2]See: https://sites.google.com/view/realm-rigidmilp

Accurately verifying condition 2) of Proposition 1 involves a product of two sets of variables - the positions of x_a, x_b and the proportion (x, x_a) occupies. This product cannot be encoded into a mixed integer linear programming. Instead, we divide the line segments into 10 equal parts and only check the 11 endpoints in practice. So the MILP constraints can be written as, $\forall e_s = (p_{s,\text{start}}, p_{s,\text{end}}) \in \mathcal{S}_{\text{line}}$:

$$A_i(\eta p_{s,\text{start}} + (1-\eta)p_{s,\text{end}}) \le b_i + M(1-\beta_{s,i,\eta}),$$
$$\forall P_i \in \mathcal{P}_c, \forall \eta \in \{0, 0.1, \ldots, 1\} \tag{5a}$$

$$\sum_i \beta_{s,i,\eta} \ge 1, \quad \forall \eta \in \{0, 0.1, \ldots, 1\} \tag{5b}$$

$$\beta_{s,\text{cond1},i} \le \beta_{s,i,0}, \quad \beta_{s,\text{cond1},i} \le \beta_{s,i,1} \tag{5c}$$

$$\sum_\eta (\beta_{s,i,\eta} + \beta_{s,j,\eta} - 1) \ge 1 - M(1-\beta_{s,\text{cond2},ij}),$$
$$\forall P_i, P_j \in \mathcal{P}_c,\ P_i \ne P_j \tag{5d}$$

$$\sum_i \beta_{s,\text{cond1},i} + \sum_{(i,j)} \beta_{s,\text{cond2},ij} \ge 1 \tag{5e}$$

where $\beta_{s,i,\eta}$, $\beta_{s,\text{cond1},i}$, $\beta_{s,\text{cond2},ij}$ are binary variables for line segment e_s. $\beta_{s,i,\eta} = 1$ indicates that interpolated points are inside polytope P_i. Similarly, $\beta_{s,\text{cond1},i}$ or $\beta_{s,\text{cond2},ij}$ being 1 means the corresponding condition is satisfied. Equations (5a) and (5b) encodes the preconditions, while (5c) and (5d) encodes the two conditions in Proposition 1, respectively.

C. Discussion

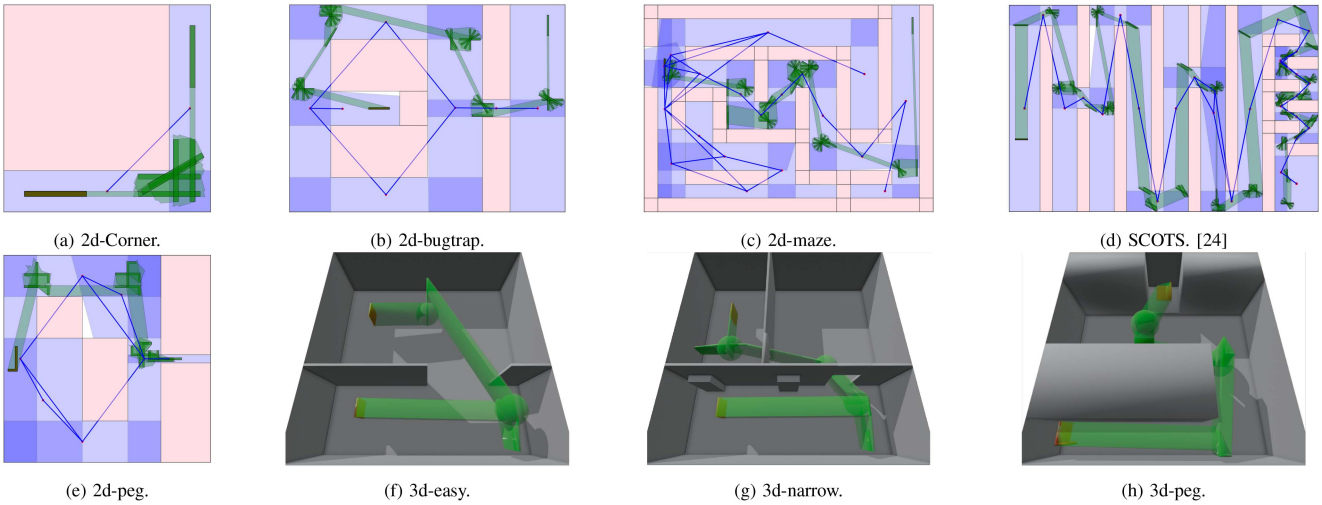
While our approach decomposes the original planning problem and solves each subproblem optimally, it does not guarantee global optimality for the entire path. However, our algorithm is capable of finding multiple solutions or modalities, which can then serve as initial solutions for further refinements. It is important to acknowledge that in the 3D case, our method currently only allows translation within the MILP formulation to simplify the computation of the reachable set. We recognize that this simplification may exclude some potential solutions. To address this limitation, future work will focus on extending the approximation techniques used in the 2D case to the 3D domain.

V. Experiments

We empirically validate our method in this section. We initially set the number of intermediate waypoints, N, to 0. If no solution can be found with $N = 0$, we increase N to 1. Additionally, rotations are discretized into 12 distinct rotations for 2D environments and 24 for 3D environments, respectively. All experiments were launched on a server with 1 AMD Ryzen Threadripper 3990X 64-Core Processor. We adopt Gurobi 10.0.0 [25] as the MILP solver and handle the graphs G_c, G_d with NetworkX [26].

Benchmarks: We collect 8 benchmark environments, visualized in Fig. 5. We consider two types of objects: convex and non-convex. In 2D environments, the convex object is selected as a stick of length 1.2 and width 0.1, while the non-convex object is an L-shape with longer side 1.2, shorter side 0.8 and width 0.1. In 3D environments, the convex object is a pad of size $1.0 \times 0.8 \times 0.1$, and the non-convex object is L-shaped, composed of two pads of size $1.0 \times 0.8 \times 0.1$ and $1.0 \times 0.1 \times 0.4$.

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.



(a) 2d-Corner. (b) 2d-bugtrap. (c) 2d-maze. (d) SCOTS. [24]

(e) 2d-peg. (f) 3d-easy. (g) 3d-narrow. (h) 3d-peg.

Fig. 5. Visualization of planning results in 2D (obstacles: pink) and 3D scenarios. The start and end configurations are plotted with red, while waypoints and their corresponding reachable sets are shown in green. In 2D scenarios, we also illustrate the graph G_c with vertices ($\mathcal{V}_c$) as blue polytopes and edges ($\mathcal{E}_c$) as blue lines connecting red dots, which represent the centers of vertex polytopes. Fig. 5(e) and (h) show the planning results of L-shape non-convex objects, while the rest show the results of convex objects.

Baselines: We compare our method with the following multi-query motion planning algorithms, PRM [2], WCO [8] and GCS [27]. PRM and WCO, sampling-based methods, are evaluated across 5 trials for each problem set with an offline phase of 15 seconds allocated to develop a roadmap before each trial. GCS constructs a graph of convex set in configuration space [23], followed by optimizing for an optimal path with piece-wise linear curves [27]. To ensure a fair comparison in generating an IRIS cover for both GCS and our method, though their application in different spaces, we select the same set of hyperparameters. Specifically, we set $n_v = 512$, $\alpha = 0.95$ and select $n_s = 5$ for our method.

TABLE I
COMPARISON OF OUR METHOD AND BASELINES ON BENCHMARKS

(a) Results for convex objects (A stick in 2D scenarios and a pad for 3D).

	Online Time ↓ (ms)				Waypoint Number ↓			
scenario	ours	GCS	PRM	WCO	ours	GCS	PRM	WCO
2d-corner	**2.8**	607.8	105.3	115.3	8	**6**	20	17
2d-bugtrap	**15.1**	INF	116.4	23.8	**21**	INF	40	23
2d-maze	**57.1**	7.2e5	111.9	211.2	29	**21**	42	24
SCOTS	**55.0**	INF	508.3	360.4	85	INF	97	**76**
2d-peg	**22.1**	INF	122.6	87.5	**13**	INF	31	27
3d-easy	**63.8**	1.2e5	129.3	244.4	10	**9**	10	24
3d-narrow	**212.4**	2.0e5	1045.0	267.0	14	32	**12**	25
3d-peg	**48.1**	INF	287.1	162.1	20	INF	**6**	21

(b) Results for non-convex objects (L-shape for all scenarios).

	Online Time ↓ (ms)				Waypoint Number ↓			
scenario	ours	GCS	PRM	WCO	ours	GCS	PRM	WCO
2d-corner	**2.8**	4.1e4	105.9	258.3	**9**	11	19	21
2d-bugtrap	**11.5**	INF	113.8	25.4	**18**	INF	40	29
2d-maze	**54.8**	2.6e4	110.4	60.1	29	**25**	43	39
SCOTS	**47.8**	INF	3822.2	473.8	97	INF	98	**72**
2d-peg	**12.8**	INF	115.8	106.6	**13**	INF	28	17
3d-easy	**66.7**	2.1e5	269.3	106.3	10	**5**	7	21
3d-narrow	**257.5**	INF	593.7	336.5	16	INF	**11**	23
3d-peg	**52.1**	INF	824.7	224.3	16	INF	**11**	23

A. Planning Results

Metrics: In evaluating the performance of the methods, we assess two key metrics. First, we measure the **online time** required to compute a path when a new start and goal configuration pair is specified. Secondly, we evaluate the **number of waypoints** in the solution path. This metric serves as an indicator of the path's complexity and efficiency in navigating from start to goal. A lower number of waypoints generally suggests a smoother execution in practice.

Result We compare with baselines and show the results in Table I. We also visualized the solution paths in Fig. 5.

a) Online time: Our method consistently achieved the shortest online computation times, typically under 100ms across most test scenarios. This efficiency contrasts with the GCS, which struggles to find feasible solutions in several environments. This is particularly evident in configurations involving narrow tunnels not covered by the C-IRIS strategy, which is explored further in Section V-C.

The online time for our method involves retrieving the path from G_d and connecting the start and end configurations to the graph. In practice, we set the number of intermediate waypoints $N = 0$, allowing us to leverage matrix operations for quick verification instead of solving MILPs for faster online computation. In contrast, the online stage for GCS involves solving an optimization problem. For PRM and WCO, it includes discrete collision checking to connect start and end configurations to a dense graph, much larger than ours.

b) Waypoint comparison: GCS, which optimizes for path optimality, generally produces smoother paths when it can find a solution, leading to fewer waypoints. In 2D environments, our method typically finds paths with fewer waypoints compared to sampling-based methods, indicating a more efficient pathfinding strategy. But the situation differs in 3D. This is because we over-restrict the free configurations on the boundaries in 3D

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

TABLE II
COMPARISON OF OUR METHOD WITH SAMPLING-BASED METHODS FOR AN L-SHAPED OBJECT NAVIGATING A BUGTRAP ENVIRONMENT FIG. 5(B)

Time (ms)	ours	PRM-0.25x	WCO-0.25x	PRM-0.1x	WCO-0.1x	PRM-0.05x	WCO-0.05x
original	**11.6**	113±0.5	25.4±8.1	109±1.4	30.7±15.4	127±22.5	38.5±19.1
large	**13.4**	521±823	39.6 ± 19.3	4.2e3±2.2e3	51.2 ± 25.0	1.5e4±1.2e4	69.3 ± 33.7
narrow	**13.0**	1.4e3±2.5e3	40.8 ± 16.3	5.8e3±4.1e3	47.2 ± 19.8	1.2e4±1.0e4	56.6 ± 24.2
L & N	**17.4**	2.2e4±1.5e4	279.3 ± 208.4	3.7e4±5.6e3	384.5 ± 291.5	3.9e4±4.2e2	487.5 ± 364.6

Online time is evaluated in both large and narrow environments. In large settings, the width mid-right corridor remains constant while the dimensions of the entire environment are doubled. In narrow settings, the corridor width is reduced to 60% of its original size. In L & N settings, the corridor width is reduced and the dimensions of the environment are doubled. The online time limit for PRM is 40s.

TABLE III
METRICS OF OFFLINE STAGES FOR OUR METHOD AND GCS

	Offline Time ↓ (s)			# IRIS regions ↓		# MILP ↓
scenario	G_c	G_d	GCS	ours	GCS	ours
corner	1.1	0.5	23.7	2	24	6
bugtrap	4.1	10.7	21.6	7	42	79
maze	6.1	130.3	251.4	19	131	1624
SCOTS	4.0	45.7	79.3	27	78	355
2d-peg	5.2	18.8	26.5	8	59	170
3d-easy	5.3	15.8	27.8	4	55	42
3d-narrow	7.7	115.5	96.9	8	95	471
3d-peg	16.4	10.1	166.9	4	164	15

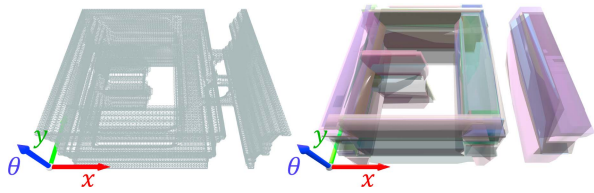


Fig. 6. A 2D motion planning problem consists of a stick getting out of a trap and passing through a narrow gap, where our method is able to achieve almost 100% coverage in workspace Fig. 5(b). The C-space is a subset of $SE(2)$, with two translation axes and a periodic axis corresponding to the rotation. We visualize the collision-free C-space by sampling (left) and the C-IRIS cover acquired from GCS (right).

scenarios, so the solution space is much smaller for our method and our method tends to generate more complex and twisted paths.

c) Impact of object shapes: Our method allows the G_c to be reused for different object shapes. With the dimensions of the object being similar, our method and sampling-based methods are able to maintain a consistent online time for different objects. Conversely, GCS performance is significantly impacted by changes in object shape, altering the geometry of the free configuration space and affecting feasibility and computation times across various environments.

B. Comparison With Sampling-Based Algorithms

Sampling-based methods, such as PRM and WCO, face challenges when scaling to larger environments. One critical hyperparameter affecting their performance is the resolution of collision checking. To assess the adaptability of these methods, we tested their efficiency with varying distances between checked states, specifically at 0.25x, 0.1x, and 0.05x of the object's length, as detailed in Table II .

Our experiments revealed that both PRM and WCO struggle to maintain performance as the environment scales. In contrast, the offline computation time for our method consistently remains around 15 seconds across all test cases, attributable to the minimal changes in the structure of G_c and G_d besides scaling. Similarly, the online execution time remains relatively consistent across scenarios. This is because the MILP approach in our method performs exact continuous collision checking with infinite resolution, enabling it to handle larger environments more effectively. Moreover, we observed that WCO significantly outperforms PRM when scaling, demonstrating the benefit of utilizing workspace connectivity information. By leveraging this information, WCO can more efficiently explore the space and find feasible paths in larger and narrower environments compared to PRM.

C. Comparison With GCS

The major difference between our method and GCS lies in space decomposition: our method is workspace-based while GCS works in C-space. The dimensionality of C-space is usually higher than that of the workspace, accompanied by inherent geometrical and topological differences between these two spaces. In the demonstrated scenario Fig. 6, where the objective is for a stick to navigate out of a trap through a narrow gap. The collision-free C-space consists of two large regions, connected by three thin tunnels. Despite dense sampling revealing the tunnels, GCS, with 42 polytopes, failed to detect any narrow passage, thus incapable of finding a feasible solution for the problem. On the contrary, our method identifies the critical pathways and achieves a near-complete coverage in the workspace. This distinction indicates a challenge in C-space representation, pathways that occupy a small volume in the workspace can correspond to an even smaller fraction of the free space in C-space, considering the increased dimensionality. This observation underscores the necessity of carefully designing a sampling strategy to ensure reliability.

We also conduct a comprehensive comparison of time consumption and problem size for the offline stage, as shown in Table III. The offline stage for our method can be roughly divided into two: the construction of G_c (Section III-A) and G_d (Section III-B). The problem size for G_c is the number of IRIS regions ($|\mathcal{V}_c|$) and its computation only needs to be conducted once across different objects in the same environment. But the graph in GCS needs to be recomputed when either the environment or the object changes. It can be observed that the number of IRIS regions we need to grow is significantly lower than GCS while remaining high feasibility. Moving to G_d, the problem size is determined by the number of MILP required. Despite our method necessitating a substantial number of MILPs, the MILPs are designed to be of small sizes, and batch processing is highly preferred. In the experiment, we assign 2 threads for each MILP. The offline computation time for G_d

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.

scales linearly to the number of MILP. Conversely for GCS, the number of variables is proportional to the number of IRIS regions, which makes the online time extremely long for GCS when the number of IRIS regions is large.

Across the majority of test cases, our method demonstrated reduced offline computation times while maintaining feasibility. However, it's important to note the inherent advantage of GCS in its design for globally optimal solutions. While our method focuses on achieving faster computation and higher feasibility, this comes at the sacrifice of optimality.

VI. Conclusion

In this study, we introduced a path-planning approach for rigid body objects using MILP with better scalability and efficiency. By structuring our method into three distinct stages – workspace decomposition, path segment validation, and online query for rapid path retrieval, we have demonstrated improved planning efficiency in multi-query scenarios via comprehensive experiments against baselines.

Acknowledgment

Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and don't necessarily reflect the views of the sponsors.

References

[1] J. T. Schwartz and M. Sharir, "On the "piano movers" problem I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Commun. Pure Appl. Math.*, vol. 36, no. 3, pp. 345–398, 1983.

[2] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Automat.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[3] M. Vitus, V. Pradeep, G. Hoffmann, S. Waslander, and C. Tomlin, "Tunnel-MILP: Path planning with sequential convex polytopes," in *Proc. AIAA Guid., Navigation Control Conf. Exhibit*, doi: 10.2514/6.2008-7132.

[4] A. Dobson and K. E. Bekris, "Sparse roadmap spanners for asymptotically near-optimal motion planning," *Int. J. Robot. Res.*, vol. 33, no. 1, pp. 18–47, 2014.

[5] D. Nister, J. Soundararajan, Y. Wang, and H. Sane, "Nonholonomic motion planning as efficient as Piano mover's," 2023, *arXiv:2306.0130*.

[6] M. d. S. Arantes, C. F. M. Toledo, B. C. Williams, and M. Ono, "Collision-free encoding for chance-constrained nonconvex path planning," *IEEE Trans. Robot.*, vol. 35, no. 2, pp. 433–448, Apr. 2019.

[7] A. Amice, H. Dai, P. Werner, A. Zhang, and R. Tedrake, "Finding and optimizing certified, collision-free regions in configuration space for robot manipulators," in *Proc. Int. Workshop Algorithmic Found. Robot.*, 2022, pp. 328–348.

[8] H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for prm planning," in *Proc. Algorithmic Found. Robot. VII: Sel. Contributions 7th Int. Workshop Algorithmic Found. Robot.*, 2008, pp. 35–51.

[9] E. Plaku, L. E. Kavraki, and M. Y. Vardi, "Motion planning with dynamics by a synergistic combination of layers of planning," *IEEE Trans. Robot.*, vol. 26, no. 3, pp. 469–482, Jun. 2010.

[10] M. McNaughton, "Parallel algorithms for real-time motion planning," Carnegie Mellon University, 2011.

[11] A. Kelly et al., "Toward reliable off road autonomous vehicles operating in challenging environments," *Int. J. Robot. Res.*, vol. 25, no. 5/6, pp. 449–483, 2006.

[12] M. Pivtoraiko, R. A. Knepper, and A. Kelly, "Differentially constrained mobile robot motion planning in state lattices," *J. Field Robot.*, vol. 26, no. 3, pp. 308–333, 2009.

[13] L. P. Chew, "Constrained delaunay triangulations," in *Proc. 3rd Annu. Symp. Comput. Geometry*, 1987, pp. 215–222.

[14] D. Demyen and M. Buro, "Efficient triangulation-based pathfinding," in *Proc. AAAI Conf. Artif. Intell.*, 2006, vol. 6, pp. 942–947.

[15] O. Brock and L. Kavraki, "Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces," in *Proc. 2001 IEEE Int. Conf. Robot. Automat.*, 2001, vol. 2, pp. 1469–1474.

[16] R. Deits and R. Tedrake, "Computing large convex regions of obstacle-free space through semidefinite programming," in *Proc. Algorithmic Found. Robot. XI: Sel. Contributions 11h Int. Workshop Algorithmic Found. Robot.*, 2015, pp. 109–124.

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

[18] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning," *Int. J. Robot. Res.*, vol. 20, no. 5, pp. 378–400, 2001.

[19] S. Quinlan, *Real-Time Modification of Collision-Free Paths*. Stanford, CA, USA: Stanford University, 1995.

[20] F. Schwarzer, M. Saha, and J.-C. Latombe, "Adaptive dynamic collision checking for single and multiple articulated robots in complex environments," *IEEE Trans. Robot.*, vol. 21, no. 3, pp. 338–353, Jun. 2005.

[21] S. Redon, A. Kheddar, and S. Coquillart, "An algebraic solution to the problem of collision detection for rigid polyhedral objects," in *Proc. 2000 ICRA Millennium Conf.. IEEE Int. Conf. Robot. Automat. Symposia Proc.*, 2000, vol. 4, pp. 3733–3738.

[22] X. Zhang, S. Redon, M. Lee, and Y. J. Kim, "Continuous collision detection for articulated models using taylor models and temporal culling," *ACM Trans. Graph.*, vol. 26, no. 3, pp. 15–es, 2007.

[23] P. Werner, A. Amice, T. Marcucci, D. Rus, and R. Tedrake, "Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2024, pp. 10359–10365.

[24] M. Rungger and M. Zamani, "SCOTS: A tool for the synthesis of symbolic controllers," in *Proc. 19th Int. Conf. Hybrid Syst.: Comput. Control*, 2016, pp. 99–104.

[25] Gurobi Optimization, LLC, "Gurobi optimizer reference manual," 2023. [Online]. Available: https://www.gurobi.com

[26] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using NetworkX," in *Proc. 7th Python Sci. Conf.*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11–15.

[27] T. Marcucci, M. Petersen, D. v. Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Sci. Robot.*, vol. 8, no. 84, 2023, Art. no. eadf7843.

Authorized licensed use limited to: MIT. Downloaded on November 08,2024 at 22:00:02 UTC from IEEE Xplore. Restrictions apply.