2026 IEEE International Conference on Robotics and Automation (ICRA 2026)
June 1-5, 2026. Vienna, Austria

Beyond Pairwise Costs: Hyper Graph of Convex Sets for Smoothness-Aware Trajectory Planning

Yunyi Zhang, Meng Guo and Zhongkui Li

***Abstract*— Classical graph of convex sets (GCS) formulations rely on pairwise edge costs, which are insufficient to capture higher-order geometric interactions relevant to trajectory refinement. This paper proposes a hyper graph of convex sets (HGCS), which extends GCS by introducing hyperedges over multiple vertices. Using a 3-uniform construction, a second-order smoothness cost is incorporated to favor path sequences that are more suitable for dynamically feasible trajectory generation. To preserve tractability, the HGCS is converted into an equivalent classical GCS, so the resulting shortest-path problem can still be solved with existing GCS methods. The discrete path is then refined by trajectory optimization within the corresponding safe corridor. Numerical simulations and quadrotor experiments show that the proposed method provides better initialization for downstream optimization, achieves shorter trajectory duration than hierarchical GCS baselines, and is faster than joint spatio-temporal optimization.**

I. Introduction

Trajectory planning in dense obstacle environments must balance collision avoidance, dynamic feasibility, and traversal efficiency. The graph of convex sets (GCS) provides a modeling framework that couples discrete topological decisions with continuous optimization variables over convex free-space regions [1], [2]. Within this framework, geometric path planning and continuous-time trajectory optimization can be handled in a common formulation [3]. When discrete route selection and continuous trajectory variables are modeled jointly, the resulting problem simultaneously involves combinatorial topological decisions and high-dimensional continuous optimization, which can substantially increase the computational burden in cluttered environments. A challenge is to improve computational efficiency without sacrificing the modeling capability of GCS.

A. Related Work

Classical GCS formulates the shortest-path problem (SPP) over convex sets as a mixed-integer nonconvex optimization problem and derives a tight mixed-integer convex program (MICP) reformulation through perspective-based convexification. This framework has been applied to a broad range of planning problems, including motion planning in obstacle environments [3], [4], [5], [6], temporal-logic planning [7], multi-agent planning [8], and non-Euclidean planning [9]. These studies highlight the strong modeling expressiveness of GCS, although its scalability remains limited by the size of the resulting optimization problem.

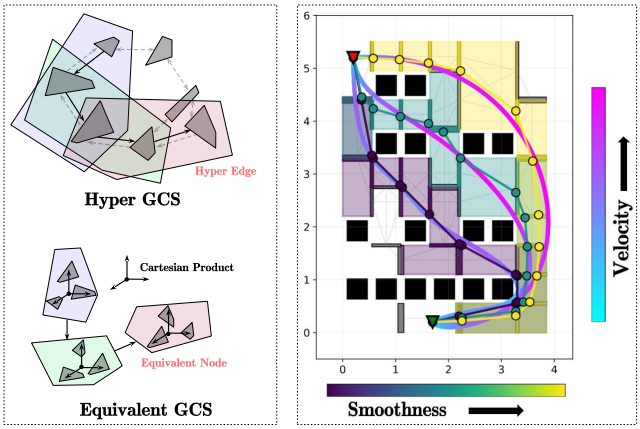

Fig. 1. **Left:** Beyond pairs: hyperedges for richer costs. GCS is extended with hyperedges in blue, green and red that link multiple vertices, enabling complex costs beyond pairwise distances; **Right:** Planning results for three representative values of the smoothness weight α. The colors of the paths and the corresponding convex set sequences indicate the value of α according to the horizontal color bar. In each case, the polyline denotes the recovered path, and the smooth curve denotes the refined trajectory.

Existing efforts to improve efficiency can be broadly grouped into two categories. One line of work accelerates the solution of SPP on GCS through search strategies, lower-bound estimation, and implicit graph constructions [10], [11], [12], [13]. Such methods avoid explicitly storing and updating all optimization variables across the entire GCS, but remain limited in handling time allocation and dynamic feasibility in trajectory optimization problems. Another line of work comes from hierarchical planning architectures, which decouple front-end path generation from back-end trajectory refinement [14], [15], [16]. Similar ideas have been extended to GCS-based planning, where a shortest path on GCS is first computed and the resulting convex set sequence is then refined by continuous optimization [17]. This decomposition is computationally attractive because it avoids solving for discrete and continuous variables within a single coupled optimization problem. A remaining limitation of such hierarchical GCS pipelines is that candidate paths are evaluated mainly through pairwise geometric costs, such as Euclidean distances between adjacent regions or interfaces. While such costs can effectively measure local geometric length, they do not explicitly capture higher-order structure across consecutive path segments. In cluttered environments, this may favor geometrically short routes that contain sharp

The authors are with the School of Advanced Manufacturing and Robotics, Peking University, Beijing, China. This work was supported by the National Natural Science Foundation of China under grants 62425301, U2241214 and T2121002.
Corresponding author: Zhongkui Li, zhongkli@pku.edu.cn.

turns or frequent directional changes, although such routes are not necessarily well suited for downstream trajectory optimization. As a result, the front-end may return an initialization whose path topology is not the most favorable for subsequent continuous-time refinement. These limitations motivate a GCS-based formulation that enables richer path representation while preserving the hierarchical efficiency needed for spatio-temporal trajectory planning.

B. Our Method

This work presents a hierarchical trajectory planning framework based on a hyper graph of convex sets (HGCS). HGCS extends classical GCS by replacing pairwise edges with hyperedges over consecutive regions. In the 3-uniform setting, a second-order smoothness-aware cost is introduced to evaluate path sequences with triplet-based geometric information during front-end planning. The resulting HGCS is then transformed into an equivalent classical GCS, so that the SPP on HGCS can still be solved using existing solvers. The selected discrete path on HGCS is finally used to construct the corresponding safe corridor for downstream continuous-time trajectory refinement. The framework provides a structured way to incorporate sequence-level geometric information into front-end planning while remaining compatible with existing GCS-based solution pipelines. Numerical simulations and hardware experiments support the effectiveness of the proposed method.

The main contribution of this work is *threefold*: (I) classical GCS is generalized to a hypergraph-structured HGCS, extending its ability to represent higher-order geometric interactions; (II) a classical GCS reformulation is developed so that HGCS can be solved using existing GCS-based tools; and (III) in the 3-uniform setting, a smoothness-aware hierarchical planning framework is developed for dense obstacle environments, providing higher-quality initializations for downstream continuous-time trajectory optimization.

II. Preliminaries

A. Graph of Convex Sets

A GCS is a graph $G := (\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ is paired with a convex program, and each edge $e := (u, v) \in \mathcal{E}$ corresponds to convex costs and constraints that couple the programs of vertices u and v [1], [2]. The convex program of vertex v has variables $\boldsymbol{x}_v \in \mathbb{R}^{n_v}$, constraint set $\mathcal{X}_v \subseteq \mathbb{R}^{n_v}$, and objective function $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$. The constraint set and cost function paired with edge $e = (u, v)$ are $\mathcal{X}_e \subseteq \mathbb{R}^{n_u+n_v}$ and $f_e : \mathbb{R}^{n_u+n_v} \to \mathbb{R}$, respectively. The sets $\mathcal{X}_v$ and $\mathcal{X}_e$ are nonempty, closed, and convex. The sets $\mathcal{X}_v$ are also assumed to be bounded. The functions f_v and f_e are convex. Given a source vertex s and a target vertex t, the SPP on GCS minimizes $\sum_{i=0}^{k} f_{v_i}(\boldsymbol{x}_{v_i}) + \sum_{i=0}^{k-1} f_{(v_i,v_{i+1})}(\boldsymbol{x}_{v_i}, \boldsymbol{x}_{v_{i+1}})$ over the discrete path $p := (v_0, \ldots, v_k)$ with $v_0 = s$, $v_k = t$, $(v_i, v_{i+1}) \in \mathcal{E}$ for all $i = 0, \ldots, k-1$, and continuous variables satisfying $\boldsymbol{x}_{v_i} \in \mathcal{X}_{v_i}$ for all $i = 0, \ldots, k$ and $(\boldsymbol{x}_{v_i}, \boldsymbol{x}_{v_{i+1}}) \in \mathcal{X}_{(v_i,v_{i+1})}$ for all $i = 0, \ldots, k-1$.

B. Hypergraph

A hypergraph generalizes a standard graph by allowing each hyperedge to connect an arbitrary number of vertices, thereby enabling the modeling of higher-order interactions among vertices [18]. Formally, a hypergraph is an ordered pair $H := (\mathcal{V}, \mathcal{E})$, where: (I) $\mathcal{V}$ is a finite, non-empty set of vertices; (II) $\mathcal{E}$ is a family of non-empty subsets of $\mathcal{V}$, called hyperedges, with $\mathcal{E} \subseteq \mathcal{P}(\mathcal{V}) \setminus \{\emptyset\}$, where $\mathcal{P}(\mathcal{V})$ denotes the power set of $\mathcal{V}$. To encode directional or sequential dependencies, a directed hypergraph is considered, in which each hyperedge is an ordered tuple of vertices. Consequently, different permutations of the same vertices correspond to distinct directed hyperedges. A hypergraph is said to be k-uniform if every hyperedge contains exactly k vertices. In particular, the case $k = 2$ reduces to a standard graph.

III. Problem Statement

Consider a robot operating in a d-dimensional configuration space $\mathcal{C} \subset \mathbb{R}^d$, where $d \in \{2, 3\}$ represents the spatial dimensions. The robot must navigate from an initial configuration $\boldsymbol{p}_s \in \mathcal{C}$ to a goal configuration $\boldsymbol{p}_t \in \mathcal{C}$ while avoiding collision with obstacles. The configuration space contains a set of obstacles $\mathcal{O} := \{\mathcal{O}_1, \mathcal{O}_2, \ldots, \mathcal{O}_N\}$, where each obstacle $\mathcal{O}_i \subset \mathcal{C}$ is a compact set. The free space is defined as $\mathcal{C}_{\text{free}} := \mathcal{C} \setminus \bigcup_{i=1}^{N} \mathcal{O}_i$. It is assumed that the free space can be decomposed into a finite collection of convex polytopes $\mathcal{X} := \{\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_M\}$ such that $\mathcal{C}_{\text{free}} := \bigcup_{j=1}^{M} \mathcal{X}_j$, where each $\mathcal{X}_j$ is a convex polytope.

Continuous-time trajectories are considered, parameterized as $\boldsymbol{p}(t) : [0, T] \to \mathbb{R}^d$, where $T > 0$ is the trajectory duration. The trajectory is represented by piecewise polynomial functions with continuity up to the $(s-1)$-th derivative, where $s \geq 1$ denotes the control-input order. Let $\boldsymbol{p}^{(k)}(t)$ denote the k-th time derivative of $\boldsymbol{p}(t)$, and define $\boldsymbol{p}^{[k]}(t) := [\boldsymbol{p}^{(0)}(t)^T, \boldsymbol{p}^{(1)}(t)^T, \ldots, \boldsymbol{p}^{(k)}(t)^T]^T \in \mathbb{R}^{d(k+1)}$ as the stacked derivative vector up to order k. The trajectory planning problem is then stated as follows.

Problem 1. Given the free space $\mathcal{C}_{\text{free}}$, initial position $\boldsymbol{p}_s$, and terminal position $\boldsymbol{p}_t$, find a trajectory $\boldsymbol{p}(t)$ and a duration $T > 0$ that solve

$$
\begin{aligned}
\min_{\boldsymbol{p}(t),\, T} \quad & \int_0^T \boldsymbol{p}^{(s)}(t)^T \mathbf{W} \boldsymbol{p}^{(s)}(t)\, dt + \rho T \\
\textbf{s.t.} \quad & \boldsymbol{p}(t) \in \mathcal{C}_{\text{free}}, \quad \forall t \in [0, T], \\
& \mathcal{G}\Big(\boldsymbol{p}(t), \ldots, \boldsymbol{p}^{(s)}(t)\Big) \preceq \mathbf{0}, \quad \forall t \in [0, T], \\
& \boldsymbol{p}(0) = \boldsymbol{p}_s, \qquad \boldsymbol{p}(T) = \boldsymbol{p}_t,
\end{aligned}
\tag{1}
$$

where $\mathbf{W} \in \mathbb{R}^{d \times d}$ is a positive definite weight matrix for the control-effort cost, $\rho > 0$ is a time regularization parameter, and $\mathcal{G} : \mathbb{R}^{d(s+1)} \to \mathbb{R}^{n_g}$ encodes dynamic feasibility constraints such as velocity and acceleration limits. ■

IV. Proposed Solution

This section introduces the proposed pipeline and summarizes its main components. Sec. IV-A introduces the line

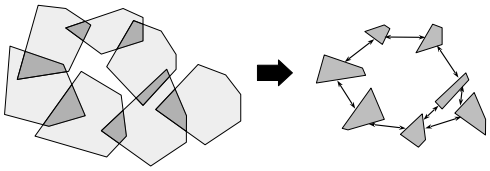

Fig. 2. Construction of the line GCS from the GCS. **Left**: Light-gray polygons denote regions in the GCS, and the dark-gray areas indicate pairwise intersections between adjacent regions. **Right**: Each shaded intersection is lifted to a vertex, while directed edges connect two vertices when the corresponding intersections share a common original convex region.

GCS, which provides a compact representation for the subsequent planning problem. Sec. IV-B formulates the HGCS by extending pairwise edges to hyperedges, thereby accommodating higher-order cost functions beyond pairwise functions. Sec. IV-C then focuses on the 3-uniform HGCS setting for trajectory planning and encodes smoothness via a ternary cost term. Finally, an equivalence-preserving transformation from HGCS to a classical GCS is derived, which allows existing GCS optimization tools to be applied directly.

A. Line Graph of Convex Sets

A line graph construction for path planning through collections of boxes was proposed in [17], where the continuous variables of a GCS are restricted to pairwise intersections of convex sets, thereby inducing a new GCS. As illustrated in Fig. 2, the line GCS treats nonempty intersections as vertices and connects them whenever they can be bridged through a common region. Given a GCS $G := (\mathcal{V}, \mathcal{E})$ with vertex sets $\{\mathcal{X}_v\}$, the line GCS $G^L := (\mathcal{V}^L, \mathcal{E}^L)$ is defined as: (I) $\mathcal{V}^L := \{v_{ij}^L \mid \mathcal{X}_{v_i} \cap \mathcal{X}_{v_j} \neq \emptyset, v_i, v_j \in \mathcal{V}, i < j\}$; (II) $\mathcal{X}_{v_{ij}^L} := \mathcal{X}_{v_i} \cap \mathcal{X}_{v_j}$; (III) $\mathcal{E}^L := \{(v_{ij}^L, v_{kl}^L) \mid \exists \mathcal{X}_{v_s}, \mathcal{X}_{v_{ij}^L} \cap \mathcal{X}_{v_s} \neq \emptyset, \mathcal{X}_{v_{kl}^L} \cap \mathcal{X}_{v_s} \neq \emptyset\}$. By elevating region overlaps to vertices, the line GCS yields a compact representation of feasible transitions in which each discrete step is anchored to an explicit overlap set, thereby preserving safety by construction. This interface-based representation will also be used as the underlying graph for HGCS, where costs can be naturally defined over compositions of transitions.

B. Hyper Graph of Convex Sets

The line GCS captures the connectivity of region overlaps, but incorporating geometric objectives that couple multiple consecutive transitions requires a more expressive structure than pairwise edge costs. Since a classical GCS restricts costs to vertices and pairwise edges, it cannot directly represent higher-order criteria that couple successive segments within a single convex cost. This motivates the HGCS, which generalizes edges to hyperedges and assigns convex cost functions jointly over the variables associated with all incident vertices. The general definition of HGCS is given below, followed by a 3-uniform specialization tailored to trajectory planning.

Definition 1 (Hyper Graph of Convex Sets)**.** Given a GCS $G := (\mathcal{V}, \mathcal{E})$ with $\{\mathcal{X}_v\}_{v\in\mathcal{V}}$, a hyper graph of convex sets $G^H := (\mathcal{V}^H, \mathcal{E}^H)$ is defined as: (I) $\mathcal{V}^H := \mathcal{V}$; (II) $\mathcal{E}^H \subseteq \{(v_1, \ldots, v_k) \mid v_i \in \mathcal{V}^H,\ 1 \le k \le |\mathcal{V}^H|,\ i = 1, \ldots, k\}$; (III) each vertex $v \in \mathcal{V}^H$ is associated with a convex cost function $f_v^H : \mathcal{X}_v \to \mathbb{R}_{\ge 0} \cup \{\infty\}$; (IV) each hyperedge $e = (v_1, \ldots, v_k) \in \mathcal{E}^H$ is associated with a convex cost function $f_e^H : \mathcal{X}_{v_1} \times \cdots \times \mathcal{X}_{v_k} \to \mathbb{R}_{\ge 0} \cup \{\infty\}$. ■

Solving an HGCS amounts to selecting a feasible hyperedge sequence and a consistent assignment of continuous variables on shared vertices, so as to minimize the total vertex and hyperedge cost. To leverage the off-the-shelf shortest-path solver for classical GCS, the HGCS can be reduced to an equivalent classical GCS. Under this reduction, each feasible path corresponds to a hyperedge sequence, and the edge constraints enforce the required consistency conditions. The definition of equivalent GCS is given below.

Definition 2 (Equivalent GCS for HGCS)**.** Given a general HGCS $G^H := (\mathcal{V}^H, \mathcal{E}^H)$ with vertex sets $\{\mathcal{X}_v\}_{v\in\mathcal{V}^H}$, vertex costs $\{f_v^H\}_{v\in\mathcal{V}^H}$, and hyperedge costs $\{f_e^H\}_{e\in\mathcal{E}^H}$, the equivalent GCS $G^E := (\mathcal{V}^E, \mathcal{E}^E)$ is defined as: (I) $\mathcal{V}^E := \mathcal{E}^H$; (II) for $v^E = (v_1, \ldots, v_k) \in \mathcal{V}^E$, $\mathcal{X}_{v^E}^E := \mathcal{X}_{v_1} \times \cdots \times \mathcal{X}_{v_k}$; (III) $\mathcal{E}^E \subseteq \{(v_1^E, v_2^E) \in \mathcal{V}^E \times \mathcal{V}^E \mid v_1^E \neq v_2^E,\ \exists i, j :\ (v_1^E)_i = (v_2^E)_j\}$; (IV) for $v^E = (v_1, \ldots, v_k) \in \mathcal{V}^E$ with $\boldsymbol{x}_{v^E}$, define the equivalent vertex cost by

$$f_{v^E}^E(\boldsymbol{x}_{v^E}) := \sum_{i=1}^{k} f_{v_i}^H(\boldsymbol{x}_{v_i}) + f_{v^E}^H(\boldsymbol{x}_{v^E}),$$

and set $f_{(v_1^E, v_2^E)}^E := 0$ for all $(v_1^E, v_2^E) \in \mathcal{E}^E$; (V) for $(v_1^E, v_2^E) \in \mathcal{E}^E$, $\mathcal{X}_{(v_1^E, v_2^E)}^E := (\mathcal{X}_{v_1^E}^E \times \mathcal{X}_{v_2^E}^E) \cap \mathcal{C}_{(v_1^E, v_2^E)}$, where

$$\mathcal{C}_{(v_1^E, v_2^E)} := \Big\{(\boldsymbol{x}_{v_1^E}, \boldsymbol{x}_{v_2^E}) \;\Big|\; \mathbf{A}_{(v_1^E, v_2^E)} \begin{bmatrix} \boldsymbol{x}_{v_1^E}^\top & \boldsymbol{x}_{v_2^E}^\top \end{bmatrix}^\top = \mathbf{0}\Big\}.$$

Here, let $\mathcal{I}(v_1^E, v_2^E) := \{(i, j) \mid (v_1^E)_i = (v_2^E)_j\}$, and define

$$\mathbf{A}_{(v_1^E, v_2^E)} := \begin{bmatrix} \boldsymbol{e}_i^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_j^\top \otimes \mathbf{I}_d \end{bmatrix}_{(i,j)\in\mathcal{I}(v_1^E, v_2^E)},$$

i.e., $\mathbf{A}_{(v_1^E, v_2^E)}$ is obtained by stacking the block rows $\begin{bmatrix} \boldsymbol{e}_i^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_j^\top \otimes \mathbf{I}_d \end{bmatrix}$ for all $(i, j) \in \mathcal{I}(v_1^E, v_2^E)$. ■

In the equivalent GCS construction, each hyperedge of the HGCS is lifted to a vertex of the GCS, whose associated set is defined as the Cartesian product of the sets incident to that hyperedge. Since Cartesian products preserve convexity, every lifted vertex is associated with a convex set. The edge costs are set to zero so that the objective is entirely captured by vertex costs. Each lifted vertex cost is defined as the hyperedge cost plus the sum of the costs of the original vertices covered by that hyperedge.

Remark 1. The edge set $\mathcal{E}^E$ in Def. 2 is specified as a subset rather than an equality. Sharing a common original vertex is a necessary condition for two equivalent vertices to be connected, but the specific edge set is determined by the application. For k-uniform HGCS, a natural choice is the $(k-1)$-ordered overlap, where the last $k-1$ elements of one hyperedge match the first $k-1$ elements of the next, which ensures unique recovery of the waypoint sequence. ■

For the equivalent GCS $G^E = (\mathcal{V}^E, \mathcal{E}^E)$, the edge costs satisfy $f_{e^E}^E \equiv 0$, so the objective is determined by vertex

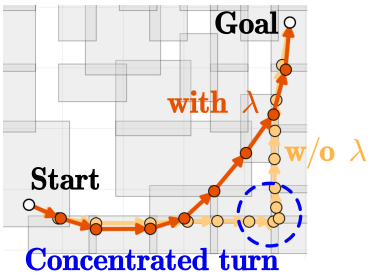

Fig. 3. Illustration of the effect of $\lambda(\cdot)$. The gray boxes are the convex sets, and the colored lines are the paths generated from HGCS. Without normalization (light orange), the turning motion is concentrated in a local region. With normalization (dark orange), the heading change is distributed over multiple consecutive segments, resulting in a smoother path.

costs. Let $s^E, t^E \in \mathcal{V}^E$ be the start and goal vertices, and let $y_{e^E}, y_{v^E} \in \{0,1\}$ denote edge/vertex selection variables. The unit-flow constraints are imposed by

$$y_{v^E} = \sum_{e^E \in \mathcal{E}^{E,\text{in}}_{v^E}} y_{e^E} + \delta_{s^E v^E} = \sum_{e^E \in \mathcal{E}^{E,\text{out}}_{v^E}} y_{e^E} + \delta_{t^E v^E}, \quad (2)$$

for all $v^E \in \mathcal{V}^E$. Introducing the perspective variables $\boldsymbol{z}_{v^E}$ and $\boldsymbol{z}^{e^E}_{v^E}$, the consistency is enforced via

$$\sum_{e^E \in \mathcal{E}^{E,\text{in}}_{v^E}} \boldsymbol{z}^{e^E}_{v^E} = \sum_{e^E \in \mathcal{E}^{E,\text{out}}_{v^E}} \boldsymbol{z}^{e^E}_{v^E}, \quad (3)$$

for all $v^E \in \mathcal{V}^E \setminus \{s^E, t^E\}$. Let $\tilde{\mathcal{X}}_{v^E}$ and $\tilde{\mathcal{X}}^E_{e^E}$ be the vertex and edge perspective sets, and let $\tilde{f}^E_{v^E}$ be the perspective cost function. The MICP formulation for SPP on G^E is given by

$$\begin{aligned} \textbf{min} \quad & \sum_{v^E \in \mathcal{V}^E} \tilde{f}^E_{v^E}(\boldsymbol{z}_{v^E}, y_{v^E}) \\ \textbf{s.t.} \quad & (\boldsymbol{z}_{v^E}, y_{v^E}) \in \tilde{\mathcal{X}}_{v^E}, \quad \forall v^E, \\ & (\boldsymbol{z}^{e^E}_{u^E}, \boldsymbol{z}^{e^E}_{w^E}, y_{e^E}) \in \tilde{\mathcal{X}}^E_{e^E}, \quad \forall e^E = (u^E, w^E), \\ & y_{v^E}, y_{e^E} \in \{0,1\}, \quad \forall v^E, \ \forall e^E, \\ & (2), \ (3), \end{aligned} \quad (4)$$

and in practice, one may relax the integrality constraints to $y_{e^E}, y_{v^E} \in [0,1]$ and recover a path by a rounding procedure.

C. Trajectory Optimization Induced by 3-uniform HGCS

A 3-uniform HGCS is a special case of the general HGCS where all hyperedges have cardinality exactly 3. In this part, the focus is on this particular instantiation of HGCS and its role in trajectory planning is demonstrated below.

Given the line GCS $G^L = (\mathcal{V}^L, \mathcal{E}^L)$, the 3-uniform HGCS $G^H = (\mathcal{V}^H, \mathcal{E}^H)$ is defined by: (I) $\mathcal{V}^H := \mathcal{V}^L$; (II) $\mathcal{E}^H := \{(u^H, v^H, w^H) \mid (u^H, v^H) \in \mathcal{E}^L, \ (v^H, w^H) \in \mathcal{E}^L\}$. Each hyperedge is an ordered triplet, enabling modeling of second-order smoothness constraints through ternary cost functions.

The HGCS is extracted from the line GCS according to Alg. 1 by traversing all connected triplets and adding them as hyperedges. A related construction was used in [10] for the Lower Bound Graph (LBG). Unlike the static graph in [10], the hyperedges here define the feasible domain of the subsequent optimization problem. In addition, this extraction

Algorithm 1: `Build3UHGCS(·)`

Input: Line GCS $G^L = (\mathcal{V}^L, \mathcal{E}^L)$
Output: 3-uniform HGCS $G^H = (\mathcal{V}^H, \mathcal{E}^H)$

$\mathcal{V}^H \leftarrow \mathcal{V}^L$;
$\mathcal{E}^H \leftarrow \emptyset$;
foreach $v \in \mathcal{V}^L$ **do**
  foreach $(u, v) \in \mathcal{E}^L$ **do**
    foreach $(v, w) \in \mathcal{E}^L$ **do**
      $\mathcal{E}^H \leftarrow \mathcal{E}^H \cup \{(u, v, w)\}$;
return $G^H = (\mathcal{V}^H, \mathcal{E}^H)$;

step does not require solving any optimization problem and is therefore computationally efficient.

For the specific case of 3-uniform HGCS, the equivalent GCS $G^E := (\mathcal{V}^E, \mathcal{E}^E)$ is formally defined as: (I) $\mathcal{V}^E := \{(u^L, v^L, w^L) \mid (u^L, v^L) \in \mathcal{E}^L, \ (v^L, w^L) \in \mathcal{E}^L\}$; (II) for $v^E = (v_a, v_b, v_c) \in \mathcal{V}^E$, $\mathcal{X}^E_{v^E} := \mathcal{X}_{v_a} \times \mathcal{X}_{v_b} \times \mathcal{X}_{v_c}$; (III) $\mathcal{E}^E := \{((v_a, v_b, v_c), (w_a, w_b, w_c)) \mid v_b = w_a, \ v_c = w_b\}$.

Let the continuous variable associated with v^E be $\boldsymbol{x}_{v^E} = [\boldsymbol{p}_0^\top, \boldsymbol{p}_1^\top, \boldsymbol{p}_2^\top]^\top \in \mathbb{R}^{3d}$, where $\boldsymbol{p}_i \in \mathbb{R}^d$ for $i \in \{0,1,2\}$. The edge constraint set and consistency set follow Def. 2. For 3-uniform HGCS, the $(k-1)$-ordered overlap yields $(v^E)_2 = (w^E)_1$ and $(v^E)_3 = (w^E)_2$ for each $(v^E, w^E) = ((v_a, v_b, v_c), (w_a, w_b, w_c)) \in \mathcal{E}^E$, so $\mathcal{I}(v^E, w^E) = \{(2,1), (3,2)\}$. Hence $\mathbf{A}_{(v^E, w^E)}$ is the stacking of the two block rows $\begin{bmatrix} \boldsymbol{e}_2^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_1^\top \otimes \mathbf{I}_d \end{bmatrix}$ and $\begin{bmatrix} \boldsymbol{e}_3^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_2^\top \otimes \mathbf{I}_d \end{bmatrix}$, i.e.,

$$\mathbf{A}_{(v^E, w^E)} = \begin{bmatrix} \boldsymbol{e}_2^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_1^\top \otimes \mathbf{I}_d \\ \boldsymbol{e}_3^\top \otimes \mathbf{I}_d & -\boldsymbol{e}_2^\top \otimes \mathbf{I}_d \end{bmatrix},$$

and $\mathcal{C}_{(v^E, w^E)} = \{(\boldsymbol{x}_{v^E}, \boldsymbol{x}_{w^E}) \mid \mathbf{A}_{(v^E, w^E)} \begin{bmatrix} \boldsymbol{x}_{v^E}^\top & \boldsymbol{x}_{w^E}^\top \end{bmatrix}^\top = \mathbf{0}\}$, with $\mathcal{X}^E_{(v^E, w^E)} := (\mathcal{X}^E_{v^E} \times \mathcal{X}^E_{w^E}) \cap \mathcal{C}_{(v^E, w^E)}$. Define difference matrices $\mathbf{D}_1 = [-\mathbf{I}_d, \mathbf{I}_d, \mathbf{0}]$, $\mathbf{D}_2 = [\mathbf{0}, -\mathbf{I}_d, \mathbf{I}_d]$, $\mathbf{B} = [\mathbf{I}_d, -2\mathbf{I}_d, \mathbf{I}_d]$, where $\mathbf{D}_1 \boldsymbol{x}_{v^E} = \boldsymbol{p}_1 - \boldsymbol{p}_0$, $\mathbf{D}_2 \boldsymbol{x}_{v^E} = \boldsymbol{p}_2 - \boldsymbol{p}_1$, $\mathbf{B}\boldsymbol{x}_{v^E} = \boldsymbol{p}_0 - 2\boldsymbol{p}_1 + \boldsymbol{p}_2$.

A node cost combining length and bending penalty with smoothness weight $\alpha \geq 0$ is used:

$$f^E_v(\boldsymbol{x}_{v^E}) := \underbrace{\tfrac{1}{2}\left(\|\mathbf{D}_1 \boldsymbol{x}_{v^E}\|_2 + \|\mathbf{D}_2 \boldsymbol{x}_{v^E}\|_2\right)}_{\text{shortest-path term}} + \underbrace{\frac{\alpha}{\lambda(v^E)} \|\mathbf{B}\boldsymbol{x}_{v^E}\|_2^2}_{\text{smooth regularization term}},$$

where $\boldsymbol{c}_{v^E} := [\boldsymbol{c}_0^\top, \boldsymbol{c}_1^\top, \boldsymbol{c}_2^\top]^\top$ stacks the centers of the convex sets associated with v^E, and $\lambda(v^E) := \max((\|\mathbf{D}_1 \boldsymbol{c}_{v^E}\|_2 + \|\mathbf{D}_2 \boldsymbol{c}_{v^E}\|_2)^2, \epsilon)$ is a scale factor computed from these centers, with $\epsilon > 0$ a given constant. The first term is a pairwise length cost. The second term is a smoothness regularizer based on second-order central differences [19], in the spirit of discrete Laplacian smoothing [20]. The scale factor $\lambda(\cdot)$ normalizes this term across triplets of different geometric scales. As illustrated in Fig. 3, without normalization, the smoothness penalty grows with the local spacing of a triplet. As a result, a comparable turn may be penalized more heavily

Algorithm 2: `AddEndPoints(·)`

Input: $G = (\mathcal{V}, \mathcal{E})$, $G^L = (\mathcal{V}^L, \mathcal{E}^L)$, $G^E = (\mathcal{V}^E, \mathcal{E}^E)$, start point p_s, goal point p_t
Output: Updated equivalent GCS G^E

$\mathcal{V}^E_{\text{old}} \leftarrow \mathcal{V}^E$; add p_s and p_t to $\mathcal{V}^E$;
$\mathcal{A}_s \leftarrow \{ v^L \in \mathcal{V}^L \mid \exists v \in \mathcal{V},\ p_s \in \mathcal{X}_v,\ \mathcal{X}_{v^L} \subseteq \mathcal{X}_v \}$;
$\mathcal{A}_t \leftarrow \{ v^L \in \mathcal{V}^L \mid \exists v \in \mathcal{V},\ p_t \in \mathcal{X}_v,\ \mathcal{X}_{v^L} \subseteq \mathcal{X}_v \}$;
foreach $v^E = ((v^E)_1, (v^E)_2, (v^E)_3) \in \mathcal{V}^E_{old}$ **do**
  if $(v^E)_1 \in \mathcal{A}_s$ **then**
    add node $(p_s, (v^E)_1, (v^E)_2)$ and edges $p_s \to (p_s, (v^E)_1, (v^E)_2) \to v^E$;
  if $(v^E)_3 \in \mathcal{A}_t$ **then**
    add node $((v^E)_2, (v^E)_3, p_t)$ and edges $v^E \to ((v^E)_2, (v^E)_3, p_t) \to p_t$;
return G^E

Algorithm 3: Overall HGCS-based trajectory planning pipeline

Input: Obstacle space $\mathcal{O}$, start $\boldsymbol{p}_s$, goal $\boldsymbol{p}_t$, smoothness weight α
Output: Spatiotemporal trajectory $\boldsymbol{p}^*(t)$

$G \leftarrow$ `BuildGCS`$(\mathcal{O})$;
$G^L \leftarrow$ `BuildLineGCS`(G);
$G^H \leftarrow$ `Build3UHGCS`(G^L);
$G^E \leftarrow$ `BuildEquivGCS`(G^H, α);
$G^E \leftarrow$ `AddEndPoints`$(G, G^L, G^E, \boldsymbol{p}_s, \boldsymbol{p}_t)$;
$(\{\hat{y}_{e^E}, \hat{y}_{v^E}\}, \{\hat{\boldsymbol{x}}_{v^E}\}) \leftarrow$ `SolvePath`(G^E);
$(\mathcal{P}^*, \mathcal{X}^*) \leftarrow$ `RecoverPath`$(\{\hat{y}_{e^E}, \hat{y}_{v^E}\}, \{\hat{\boldsymbol{x}}_{v^E}\})$;
$\boldsymbol{p}^*(t) \leftarrow$ `RefineTraj`$(\mathcal{P}^*, \mathcal{X}^*)$;
return $\boldsymbol{p}^*(t)$;

in a larger-scale region than in a smaller one, which biases the optimizer toward postponing the heading change to a more localized region where the absolute penalty is lower. With normalization, the penalty becomes more comparable across triplets of different scales and therefore encourages the turn to be distributed over multiple consecutive segments.

Remark 2. The $\frac{1}{2}$ factor in interior node costs ensures each path segment is counted exactly once, while boundary costs provide full weighting for start/end segments. This preserves the total path length and smoothness energy. ■

In all the above transformations, the start and end points have not yet been considered. Alg. 2 provides the procedure for adding start and goal nodes to the equivalent GCS. For each endpoint, the algorithm adds a singleton equivalent node $p^E := (p)$ with fixed set $\mathcal{X}^E_{p^E} = \{p\}$, finds line GCS nodes whose convex regions contain p, and creates new equivalent triplets by prepending or appending the endpoint. This ensures path recoverability and cost consistency.

The newly added equivalent GCS nodes are classified according to their relation to the start or end, and the following node cost functions are added. With $\mathbf{B}$ as in the interior node cost, the boundary cost is

$$f^E_v(\boldsymbol{x}_{v^E}) := \begin{cases} \|\mathbf{D}_1 \boldsymbol{x}_{v^E}\|_2 + \frac{1}{2}\|\mathbf{D}_2 \boldsymbol{x}_{v^E}\|_2 \\ \quad + \dfrac{\alpha}{\lambda(v^E)} \|\mathbf{B}\boldsymbol{x}_{v^E}\|_2^2 \quad \text{(start-adjacent)}, \\ \frac{1}{2}\|\mathbf{D}_1 \boldsymbol{x}_{v^E}\|_2 + \|\mathbf{D}_2 \boldsymbol{x}_{v^E}\|_2 \\ \quad + \dfrac{\alpha}{\lambda(v^E)} \|\mathbf{B}\boldsymbol{x}_{v^E}\|_2^2 \quad \text{(goal-adjacent)}. \end{cases}$$

Edge costs are set to zero, i.e., $f^E_{e^E}(\cdot,\cdot) \equiv 0$. The SPP on G^E is cast as the MICP in (4). In practice, the convex relaxation is solved first and a rounding procedure is applied to obtain the selection variables $\{\hat{y}_{e^E}, \hat{y}_{v^E}\}$. The support of selection variables defines an ordered equivalent-node path $(v^E_0, v^E_1, \ldots, v^E_K)$ from s^E to t^E and thus fixes all discrete decisions. With $\{\hat{y}_{e^E}, \hat{y}_{v^E}\}$ fixed, $\{\hat{\boldsymbol{x}}_{v^E}\}$ are obtained by solving a small convex problem.

For the 3-uniform construction, each selected lifted node on a rounded s^E-t^E path is of the form $v^E_k = (v^L_k, v^L_{k+1}, v^L_{k+2})$, with continuous state $\hat{\boldsymbol{x}}_{v^E_k}$ encoding the waypoint triplet $(\hat{\boldsymbol{p}}_k, \hat{\boldsymbol{p}}_{k+1}, \hat{\boldsymbol{p}}_{k+2})$. Because consecutive lifted nodes satisfy the 2-ordered overlap, the waypoint sequence is uniquely recovered by stitching the shared entries: $\mathcal{P}^* = (\hat{\boldsymbol{p}}_0, \hat{\boldsymbol{p}}_1, \ldots, \hat{\boldsymbol{p}}_{K+2})$. The same path induces a line GCS corridor sequence $\mathcal{X}^{L,*} = (\mathcal{X}^L_{v^L_0}, \mathcal{X}^L_{v^L_1}, \ldots, \mathcal{X}^L_{v^L_{K+2}})$. To formalize when the G^L corridor admits a unique realization on the G, we introduce the following mild assumption and the resulting recoverability guarantee.

Assumption 1. Let $(v^L_0, \ldots, v^L_{K+2})$ be the recovered path in G^L. For each $k = 0, \ldots, K+1$, there exists a unique $\rho_k \in \mathcal{V}$ such that $\mathcal{X}^L_{v^L_k} \subseteq \mathcal{X}_{\rho_k}$ and $\mathcal{X}^L_{v^L_{k+1}} \subseteq \mathcal{X}_{\rho_k}$. This is mild in typical sparse convex decompositions and mainly excludes degenerate multi-overlap transitions. ■

Theorem 1. *Under Assumption 1, any rounded s^E-t^E path in G^E uniquely induces $\mathcal{P}^* = (\hat{\boldsymbol{p}}_0, \ldots, \hat{\boldsymbol{p}}_{K+2})$ and $\mathcal{X}^* = (\mathcal{X}_{\rho_0}, \ldots, \mathcal{X}_{\rho_{K+1}})$, with $\hat{\boldsymbol{p}}_k, \hat{\boldsymbol{p}}_{k+1} \in \mathcal{X}_{\rho_k}$ for all $k = 0, \ldots, K+1$. Hence, $(\mathcal{P}^*, \mathcal{X}^*)$ provides a geometry-consistent initialization and a feasible safe corridor for the downstream solver of Problem 1.* ■

Proof. By the 2-ordered overlap in the 3-uniform construction, the shared entries of consecutive lifted nodes are stitched uniquely, which determines $\mathcal{P}^*$. The same rounded path induces $(v^L_0, \ldots, v^L_{K+2})$ in G^L. By Assumption 1, each adjacent pair (v^L_k, v^L_{k+1}) determines a unique $\rho_k \in \mathcal{V}$, and thus a unique corridor sequence $\mathcal{X}^* = (\mathcal{X}_{\rho_0}, \ldots, \mathcal{X}_{\rho_{K+1}})$. Since $\hat{\boldsymbol{p}}_k \in \mathcal{X}^L_{v^L_k}$ and $\hat{\boldsymbol{p}}_{k+1} \in \mathcal{X}^L_{v^L_{k+1}}$, we have $\hat{\boldsymbol{p}}_k, \hat{\boldsymbol{p}}_{k+1} \in \mathcal{X}_{\rho_k}$ for all k. Therefore, $(\mathcal{P}^*, \mathcal{X}^*)$ is a valid front-end output for the downstream solver of Problem 1. □

Therefore, once a path on G^E is obtained, the downstream refinement step can be constructed directly from the recovered waypoint and corridor sequences.

D. Overall Analysis

1) Smoothness-Aware Trajectory Planning Framework: As illustrated in Alg. 3 and Fig. 4, the proposed pipeline

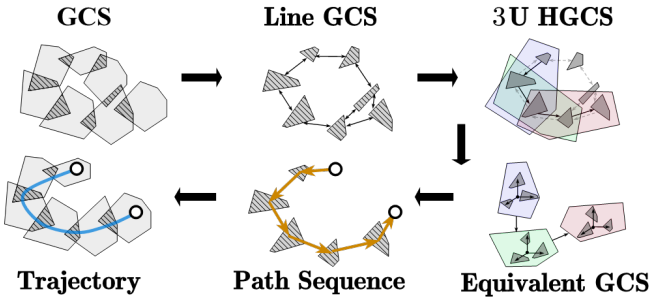

Fig. 4. Overview of the pipeline. The orange polyline denotes the recovered path, and the blue curve denotes the refined trajectory.

consists of a discrete front-end and a continuous back-end. Starting from the obstacle space $\mathcal{O}$, `BuildGCS` and `BuildLineGCS` construct a connectivity-preserving interface graph over convex safe sets. Based on this graph, `Build3UHGCS` and `BuildEquivGCS` encode triplet smoothness and convert the problem into an equivalent GCS, while `AddEndPoints` incorporates the boundary conditions. The resulting SPP is then solved by `SolvePath` using Drake [21], recovered by `RecoverPath` as a waypoint sequence and safe corridor, and finally refined by `RefineTraj` into a trajectory through Gcopter [14].

2) Complexity Analysis: The complexity of the proposed framework is mainly determined by the size of the lifted graphs. Let the original GCS be $G = (\mathcal{V}, \mathcal{E})$ with $n := |\mathcal{V}|$. Define $\Delta := \max_{v\in\mathcal{V}} |\{u \in \mathcal{V} \setminus \{v\} \mid \mathcal{X}_u \cap \mathcal{X}_v \neq \emptyset\}|$, and for the line GCS $G^L = (\mathcal{V}^L, \mathcal{E}^L)$ define $\Lambda := \max_{v\in\mathcal{V}^L} \max\{d_{\text{in}}(v), d_{\text{out}}(v)\}$. Since each vertex in $\mathcal{V}^L$ corresponds to a nonempty pairwise intersection of original convex sets, one has $|\mathcal{V}^L| = O(n\Delta)$, and therefore $|\mathcal{E}^L| = O(n\Delta\Lambda)$. According to Alg. 1, each 3-uniform hyperedge is a connected triplet in G^L, so the number of hyperedges satisfies $|\mathcal{E}^H| = \sum_{v\in\mathcal{V}^L} d_{\text{in}}(v)d_{\text{out}}(v) \le |\mathcal{V}^L|\Lambda^2 = O(n\Delta\Lambda^2)$. For the equivalent GCS $G^E = (\mathcal{V}^E, \mathcal{E}^E)$, each vertex corresponds to one hyperedge in G^H, hence $|\mathcal{V}^E| = |\mathcal{E}^H| = O(n\Delta\Lambda^2)$. Moreover, each edge in G^E connects two consecutive triplets of the form $((a,b,c),(b,c,d))$, so $|\mathcal{E}^E| \le |\mathcal{V}^E|\Lambda = O(n\Delta\Lambda^3)$. Consequently, the graph sizes satisfy $|\mathcal{V}^L| = O(n\Delta)$, $|\mathcal{E}^L| = O(n\Delta\Lambda)$, $|\mathcal{V}^E| = O(n\Delta\Lambda^2)$, and $|\mathcal{E}^E| = O(n\Delta\Lambda^3)$. Thus, `Build3UHGCS` has complexity $O(n\Delta\Lambda^2)$, while `BuildEquivGCS` and `AddEndPoints` are linear in $|\mathcal{V}^E| + |\mathcal{E}^E|$, yielding an overall front-end construction complexity of $O(n\Delta\Lambda^3)$. When Δ and Λ remain bounded, the graph growth is near-linear in n; in dense decompositions, however, the lifted graph may grow rapidly.

V. Simulations and Experiments

The proposed method is evaluated via numerical and hardware experiments to validate its effectiveness. The experiments are conducted on a computer with Intel Core i9-14900HX CPU, 94.1 GB RAM, Ubuntu 20.04.6 LTS. The proposed method is implemented in Python3, and the trajectory refinement is implemented in C++.

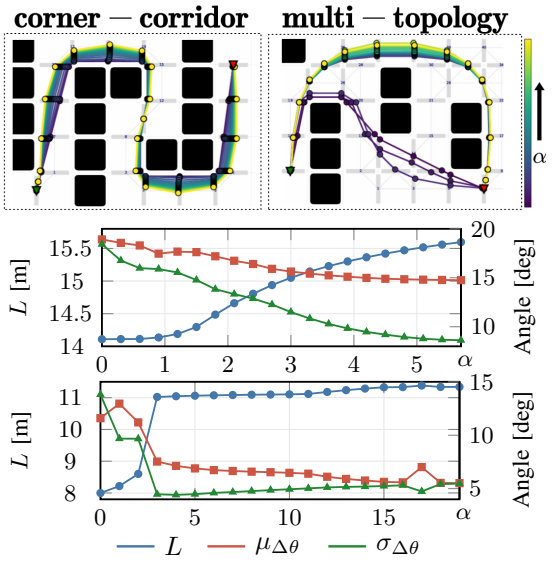

Fig. 5. Effect of the α in two representative cases. **Top:** path visualizations for the corner–corridor case and the multi-topology case. The gray polygons denote the convex sets in the line GCS, and the path color indicates the value of α according to the color bar on the right. **Bottom:** corresponding variations of L, $\mu_{\Delta\theta}$ and $\sigma_{\Delta\theta}$ with respect to α, where the upper and lower plots correspond to the two cases above, respectively.

A. Numerical Simulations

1) Effect of the Smoothness Weight: Given the waypoint sequence $\mathcal{P}^*$, the path length is defined as $L := \sum_{i=0}^{K+1} \|\hat{\boldsymbol{p}}_{i+1} - \hat{\boldsymbol{p}}_i\|_2$. For each interior waypoint $\hat{\boldsymbol{p}}_i$, the turning angle is defined as $\theta_i := \arccos\left(\frac{(\hat{\boldsymbol{p}}_i - \hat{\boldsymbol{p}}_{i-1})^\top (\hat{\boldsymbol{p}}_{i+1} - \hat{\boldsymbol{p}}_i)}{\|\hat{\boldsymbol{p}}_i - \hat{\boldsymbol{p}}_{i-1}\|_2 \, \|\hat{\boldsymbol{p}}_{i+1} - \hat{\boldsymbol{p}}_i\|_2}\right)$ for $i = 1, \ldots, K+1$. The turning-angle variation is defined as $\Delta\theta_i := |\theta_{i+1} - \theta_i|$ for $i = 1, \ldots, K$. The mean and standard deviation of the turning-angle variation are then defined as $\mu_{\Delta\theta} := \frac{1}{K}\sum_{i=1}^{K} \Delta\theta_i$ and $\sigma_{\Delta\theta} := \sqrt{\frac{1}{K}\sum_{i=1}^{K} (\Delta\theta_i - \mu_{\Delta\theta})^2}$. Since $\Delta\theta_i$ measures the change between consecutive turning angles, a smaller $\mu_{\Delta\theta}$ indicates less abrupt directional variation along the path, while a smaller $\sigma_{\Delta\theta}$ implies more uniform turning behavior. Fig. 5 illustrates the influence of the α in two representative cases. As α increases, both cases show a clear trade-off between path length and path smoothness. In the *corner-corridor* case, L increases from 14.11 m to 15.60 m, while $\mu_{\Delta\theta}$ decreases from 18.92° to 14.75°, indicating a gradually smoother path. In the *multi-topology* case, L increases from 8.00 m to 11.34 m, while $\mu_{\Delta\theta}$ decreases from 11.60° to 5.51°. In particular, a topological switch occurs at $\alpha = 3$, where L jumps from 8.60 m to 11.02 m and $\mu_{\Delta\theta}$ drops from 11.20° to 7.53°. This result indicates that HGCS can find globally smoother paths by enabling topology switching as the α varies.

2) Comparison: In this part, three random obstacle maps with increasing scale are constructed. To reduce the number of vertices and edges in the HGCS, adjacent free grid cells are merged into larger rectangular convex sets. To avoid generating excessively irregular rectangles, the aspect ratio of the merged rectangles is constrained to be no greater

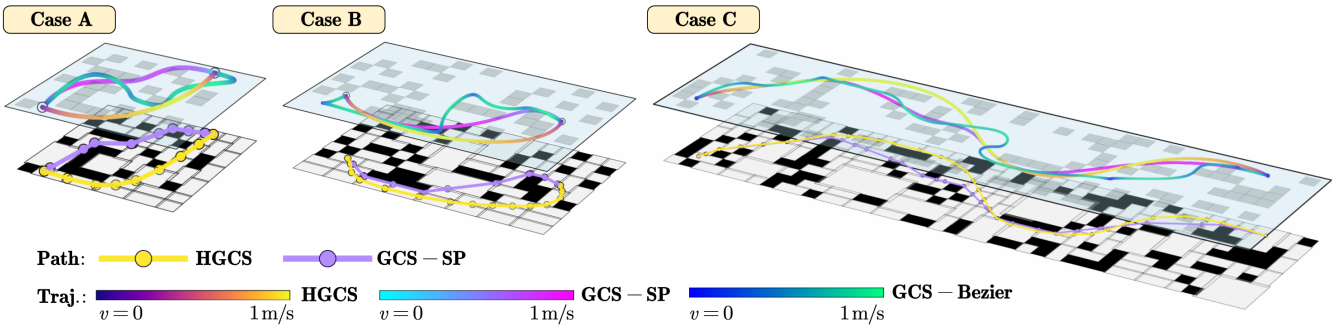

Fig. 6. Comparisons in three representative random cases (Cases A–C). In each case, the lower layer shows the piecewise-linear front-end paths, where the yellow polyline with circular markers denotes the path recovered by HGCS and the purple one denotes the path recovered by GCS-SP. The upper layer shows the corresponding refined continuous trajectories generated by HGCS, GCS-SP, and GCS-Bézier, respectively. Trajectories are color-coded by speed using the method-specific scales in the legend, with cooler colors indicating lower speed and warmer colors indicating higher speed. Note that GCS-Bézier is shown only at the trajectory level, since it directly performs joint spatio-temporal optimization rather than producing a separate front-end discrete path.

than 2:1. The map and start/goal positions for the three scenarios *Case A*, *Case B* and *Case C* are illustrated in Fig. 6. For the Gcopter, the time weight is set to $\rho = 0.01$ and the maximum speed to $v_{\max} = 1.0$ m/s. **HGCS** is benchmarked against two representative baselines: (I) **GCS-SP**, which computes a shortest path on the line GCS [1] and then refines it with Gcopter; and (II) **GCS-Bézier**, which parameterizes the trajectory by Bézier curves and jointly optimizes the spatio-temporal control points within the classical GCS framework [3]. The trajectory energy is defined as $E_{\text{traj}} := \int_0^T \|\boldsymbol{p}^{(3)}(t)\|_2^2 \, dt$. Let t_{cal} denote total computation time, L_{traj} denote trajectory length, T denote flight duration.

In Tab. I, the second column reports the numbers of vertices and edges in G, G^L, and G^E, and therefore reflects the actual discrete problem size encountered by the front end rather than only the geometric scale of the scene. As the environment becomes larger and more cluttered from Case A to Case C, the lifted graph grows substantially, with the equivalent GCS increasing from (944,3964) to (4775,26930). Compared with GCS-SP, HGCS consistently produces only slightly longer refined trajectories, from 12.55 to 12.96 m in Case A, from 15.38 to 16.04 m in Case B, and from 42.14 to 44.39 m in Case C, while yielding shorter flight durations, from 30.66 to 23.00 s, from 30.48 to 26.82 s, and from 58.02 to 55.43 s, respectively. The trajectory energy is also consistently reduced, from 0.061 to 0.046, from 0.051 to 0.043, and from 0.104 to 0.071. These results indicate that a small increase in geometric path length can provide a more refinement-friendly waypoint and corridor initialization, allowing the downstream optimizer to generate trajectories with less aggressive speed adjustment and better dynamic quality. Compared with GCS-Bézier, HGCS is substantially more efficient, reducing the total computation time from 12.87/13.65/56.35 s to 2.06/3.18/24.34 s in Cases A/B/C, while also achieving shorter flight durations in all three cases. This suggests that incorporating triplet-level smoothness information at the front end can better align discrete path selection with continuous-time trajectory optimization than either purely length-driven hierarchical planning or fully coupled joint optimization.

TABLE I

GRAPH SIZES AND PERFORMANCE COMPARISON IN DIFFERENT SCENES.

Case	Graph size $(G/G^L/G^E)$	Method	t_{cal} [s]	L_{traj} [m]	T [s]	E_{traj} [m^2/s^5]
A	(38, 55) (55, 242) (944, 3964)	HGCS (ours)	2.06	12.96	**23.00**	**0.046**
		GCS-SP [1]	**0.76**	**12.55**	30.66	0.061
		GCS-Bézier [3]	12.87	19.08	28.14	5.658
B	(58, 75) (75, 314) (1312, 5078)	HGCS (ours)	3.18	16.04	**26.82**	**0.043**
		GCS-SP [1]	**1.72**	**15.38**	30.48	0.051
		GCS-Bézier [3]	13.65	21.46	30.93	9.349
C	(124, 182) (182, 942) (4775, 26930)	HGCS (ours)	24.34	44.39	**55.43**	**0.071**
		GCS-SP [1]	**13.10**	**42.14**	58.02	0.104
		GCS-Bézier [3]	56.35	47.13	64.66	11.056

B. Hardware Experiments

The hardware experiments further validate the practical impact of the proposed framework on real quadrotor flight. A Bitcraze Crazyflie 2.1 is used as the experimental platform, with state estimation provided by an OptiTrack motion-capture system at 120 Hz. The planned trajectory is fitted by a seventh-order polynomial and tracked by a feedback controller [22], while the maximum flight speed is limited to 1.0 m/s for safety. Under the same start-goal task, HGCS is evaluated with three smoothness weights, $\alpha = 0, 1, 2$.

As shown in Fig. 7, increasing α changes not only the geometric route but also the dynamic quality of the executed trajectory. When $\alpha = 0$, HGCS favors the geometrically shortest route, but the resulting path contains sharper turns, which leads to stronger oscillations in the velocity, acceleration, and jerk profiles after continuous-time refinement. As α increases, the selected topology becomes smoother, reducing aggressive speed adjustment near corners and allowing the quadrotor to maintain high-speed motion for longer intervals.

This trend is consistent with the quantitative results in Tab. II. As α increases from 0 to 2, the discrete path length increases from 7.64 m to 8.98 m, while the refined trajectory length increases from 7.61 m to 8.80 m. Nevertheless,

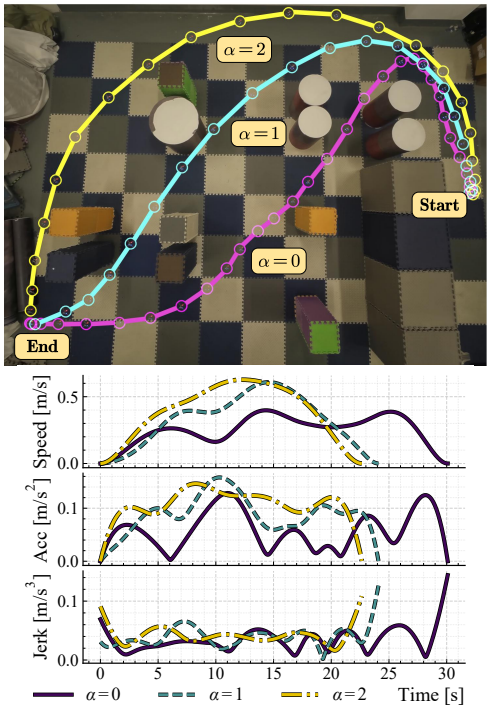

Fig. 7. Hardware experiment results under the same start-goal task with three smoothness weight settings ($\alpha = 0, 1, 2$). **Top:** Corresponding spatial trajectories; for each trajectory, adjacent markers with the same color indicate a temporal interval of 1 s. **Bottom:** Time-parameterized motion profiles (speed, acceleration, and jerk) of the executed trajectories. Increasing α leads to smoother path topology selection, reduced oscillations in speed, acceleration and jerk, and longer sustained high-speed segments.

the traversal time decreases significantly from 30.09 s to 22.65 s, indicating that the geometrically shortest route is not necessarily the most execution-efficient one in cluttered environments. Meanwhile, the planning time remains nearly unchanged, varying only from 0.887 s to 0.826 s. These hardware results show that the smoothness regularization in HGCS improves the compatibility between front-end path selection and downstream trajectory refinement, leading to smoother motion and shorter flight time without noticeable additional online computational cost.

VI. Conclusion

This paper has presented HGCS, a generalization of classical GCS that replaces pairwise edges with hyperedges. Under a 3-uniform formulation, HGCS incorporates a second-order smoothness regularizer to capture higher-order geometric interactions and provide more trajectory-oriented initializations for downstream refinement. The results show that this formulation can improve the geometric quality of the selected path and support the generation of high-quality trajectories. Future work will explore parallel computing and planning under complex temporal logic tasks.

TABLE II

Path and trajectory statistics under different α.

α	t_{cal} [s]	L [m]	L_{traj} [m]	T [s]
0	0.89	7.64	7.61	30.09
1	0.85	8.22	7.82	24.09
2	0.83	8.98	8.80	22.65

References

[1] 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.

[2] T. Marcucci, "A unified and scalable method for optimization over graphs of convex sets," *arXiv preprint arXiv:2510.20184*, 2025.

[3] 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.

[4] D. von Wrangel and R. Tedrake, "Using graphs of convex sets to guide nonconvex trajectory optimization," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 9863–9870, 2024.

[5] C. L. Clark and B. Xie, "Plan optimal collision-free trajectories with non-convex cost functions using graphs of convex sets," *IEEE Transactions on Robotics*, 2025.

[6] Y. Ping, T. Zhang, and T. Liang, "Handover-aware urllc uav trajectory planning: A continuous-time trajectory optimization via graphs of convex sets," *arXiv preprint arXiv:2511.13429*, 2025.

[7] V. Kurtz and H. Lin, "Temporal logic motion planning with convex optimization via graphs of convex sets," *IEEE Transactions on Robotics*, vol. 39, no. 5, pp. 3791–3804, 2023.

[8] S. Zhao, A. G. Philip, S. Rathinam, H. Choset, and Z. Ren, "Cb-gcs: Conflict-based search on the graph of convex sets for multi-agent motion planning," in *2025 IEEE 21st International Conference on Automation Science and Engineering (CASE)*, pp. 2208–2214, 2025.

[9] T. Cohn, M. Petersen, M. Simchowitz, and R. Tedrake, "Non-euclidean motion planning with graphs of geodesically convex sets," *The International Journal of Robotics Research*, vol. 44, no. 10–11, pp. 1840–1862, 2025.

[10] R. Natarajan, C. Liu, H. Choset, and M. Likhachev, "Implicit graph search for planning on graphs of convex sets," *arXiv preprint arXiv:2410.08909*, 2024.

[11] S. Y. C. Chia, R. H. Jiang, B. P. Graesdal, L. P. Kaelbling, and R. Tedrake, "Gcs*: Forward heuristic search on implicit graphs of convex sets," *arXiv preprint arXiv:2407.08848*, 2024.

[12] S. Morozov, T. Marcucci, A. Amice, B. P. Graesdal, R. Bosworth, P. A. Parrilo, and R. Tedrake, "Multi-query shortest-path problem in graphs of convex sets," *arXiv preprint arXiv:2409.19543*, 2024.

[13] K. Sundar and S. Rathinam, "A* for bounding shortest paths in the graphs of convex sets," in *Proceedings of the International Conference on Automated Planning and Scheduling*, vol. 35, pp. 260–268, 2025.

[14] Z. Wang, X. Zhou, C. Xu, and F. Gao, "Geometrically constrained trajectory optimization for multicopters," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3259–3278, 2022.

[15] J. Tordesillas, B. T. Lopez, M. Everett, and J. P. How, "Faster: Fast and safe trajectory planner for navigation in unknown environments," *IEEE Transactions on Robotics*, vol. 38, no. 2, pp. 922–938, 2021.

[16] D. Wang, H. Ye, N. Pan, J. Huang, B. Zhang, Y. Mao, G. Huang, C. Xu, and F. Gao, "Flexible and topological consistent local replanning for multirotors," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5348–5355, 2024.

[17] T. Marcucci, P. Nobel, R. Tedrake, and S. Boyd, "Fast path planning through large collections of safe boxes," *IEEE Transactions on Robotics*, vol. 40, pp. 3795–3811, 2024.

[18] A. Bretto, *Hypergraph Theory: An Introduction*. Mathematical Engineering, Cham: Springer, 2013.

[19] W. Gautschi, *Numerical Analysis*. Springer Science & Business Media, 2011.

[20] G. Taubin, "A signal processing approach to fair surface design," in *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pp. 351–358, 1995.

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

[22] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2520–2525, 2011.