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

Fast Motion Planning for Non-Holonomic Mobile Robots via a Rectangular Corridor Representation of Structured Environments

Alejandro Gonzalez-Garcia, Sebastiaan Wyns, Sonia De Santis, Jan Swevers and Wilm Decré

***Abstract*— We present a complete framework for fast motion planning of non-holonomic autonomous mobile robots in highly complex but structured environments. Conventional grid-based planners struggle with scalability, while many kinematically-feasible planners impose a significant computational burden due to their search space complexity. To overcome these limitations, our approach introduces a deterministic free-space decomposition that creates a compact graph of overlapping rectangular corridors. This method enables a significant reduction in the search space, without sacrificing path resolution. The framework then performs online motion planning by finding a sequence of rectangles and generating a near-time-optimal, kinematically-feasible trajectory using an analytical planner. The result is a highly efficient solution for large-scale navigation. We validate our framework through extensive simulations and on a physical robot. The implementation is publicly available as open-source software.**

I. Introduction

Autonomous Mobile Robots (AMRs) are increasingly deployed in industries, such as manufacturing, warehousing, terminals, hospitals, smart farms, and greenhouses [1]–[5]. A central challenge in autonomy is motion planning, where a trajectory or path is computed from one position to another, balancing reliability, computational efficiency, and trajectory quality [6], [7]. In this article, we focus on motion planning for non-holonomic AMRs operating in highly complex but structured environments, such as large factory floors or buildings with long corridors and narrow passages that can be particularly challenging for traditional planners.

Early motion planning methods, such as A* [8] or Dijkstra [9], rely on occupancy grid representations [10]. These methods can reliably provide collision-free paths, but as the map size or resolution increases, the number of grid cells grows rapidly, leading to longer planning times. Additionally, these paths are purely geometric, ignoring the non-holonomic constraints of AMRs. To address this limitation, lattice-based planners introduced precomputed motion primitives that enforce kinematic constraints during the search. These primitives can be generated offline either through closed-form solutions [11], [12] or optimization-based formulations [13], [14]. However, they still suffer from the scalability limitations of grid-based methods. While downsampling the map can mitigate these issues, it may also remove narrow doorways critical for successful navigation.

This work was supported by the Flanders Make SBO project ARENA (Agile & Reliable Navigation).

Authors are with MECO Research Team, Department of Mechanical Engineering, KU Leuven, Belgium and with Flanders Make@KU Leuven, Belgium. {alex.gonzalezgarcia, sonia.desantis, jan.swevers, wilm.decre}@kuleuven.be

Sampling-based planners, such as Probabilistic Roadmaps (PRMs) [15] and Rapidly-Exploring Random Trees (RRTs) [16], avoid exhaustively exploring the map. However, these probabilistic methods provide no deterministic guarantees, their solution quality strongly depends on the sampling density, and, as illustrated in [17], they often struggle in narrow passages, a common characteristic of structured environments with long hallways and door-like scenarios.

An alternative approach is planning through convex covers, by decomposing the environment into safe sets [18]–[22]. [18] demonstrates the value of separating this decomposition into offline preprocessing and online planning phases, using precomputed safe boxes to accelerate motion planning. However, most of these methods focus on local decomposition around a path rather than the entire environment. These methods typically start with a collision-free path from a discrete planner, then inflate geometric shapes around this path to construct convex covers of the free space. This process generally begins by finding inscribed ellipses from points, lines, or polytopes [21]–[23], and converting them into polygonal obstacle-free regions through iterative optimization. Here, the coverage and predictability of the safe sets are dependent on the initial seed. Moreover, trajectories are commonly generated using mixed-integer programming [20], RRT* [22], or piecewise polynomial optimization [7] within the convex covers, or through joint optimization of the trajectory and the convex cover [19]. Nevertheless, most decomposition-based methods target systems with free motion in Cartesian space, such as quadrotors or point-mass models, rather than kinematically constrained systems. In contrast, [24] has shown that near-time-optimal trajectories for non-holonomic unicycle robots can be computed analytically through predefined sequences of rectangular corridors, i.e., rectangular regions that decompose the free space. These trajectories were validated against optimal control problem (OCP) solutions, showing a two-order-of-magnitude reduction in computation time while remaining mostly within $< 1\%$ of the time-optimal solution.

Despite recent advances, existing approaches still exhibit key limitations to achieve real-time motion planning for non-holonomic AMRs in structured but complex 2D environments. Many methods scale badly with map size and resolution, rely on heuristics or sampling, ignore kinematic constraints, or require expensive online computations. To address these challenges, we propose a framework that combines offline free-space decomposition to manage environmental complexity and online analytical planning, entirely avoiding online optimization. In the offline phase, given a map,

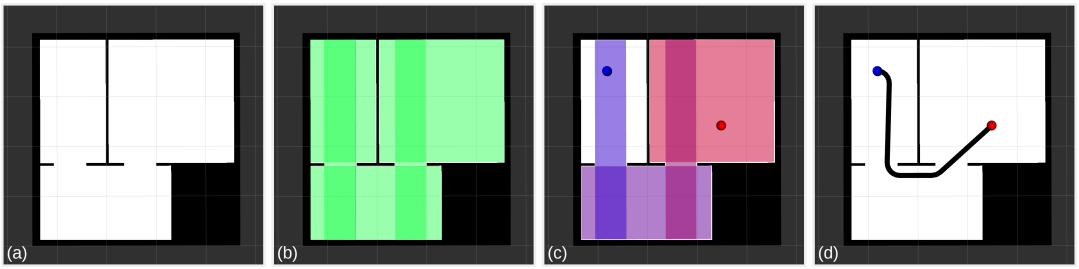

Fig. 1: Example illustrating the corridor-based motion planning framework: (a) input occupancy grid, (b) corridor decomposition, (c) planned corridor sequence (blue to red), and (d) generated analytical trajectory.

the free space is decomposed into overlapping rectangles, hereafter referred to as corridors, a process only repeated if the map changes. This corridor-based representation yields a compact, search-based graph that is inherently collision-free. Unlike grid downsampling approaches, our method provides structural geometric compression, preserving all navigable passages. In the online phase, this precomputed decomposition is used to find a sequence of corridors and leverages analytical methods to generate near-time-optimal, kinematically feasible trajectories in real time. Unlike existing approaches, this framework enables real-time, optimization-free motion planning that scales with structural complexity rather than resolution, as illustrated in Fig. 1.

A. Contributions

This paper introduces a complete framework for fast motion planning of non-holonomic AMRs in highly complex but structured environments. The main contributions of this work include:

- A deterministic, compact representation: We propose a novel algorithm to deterministically decompose the entire free space into a compact, collision-free graph of rectangular corridors. This representation achieves structural compression ratios exceeding 10,000:1, and can be reused across different planning algorithms, not just our specific framework.
- Real-time, kinematically-feasible motion planning framework: Our approach computes a corridor sequence and generates near-time-optimal, kinematically feasible trajectories that include both geometry and timing. Compared to planners that produce only geometric paths, our method achieves planning times up to an order of magnitude faster, making it suitable for real-time operation in complex environments.
- Comprehensive experimental validation: The framework's performance and reliability are demonstrated extensively through simulation on multiple layout maps and on a real robot operating in a laboratory environment.
- Open-source implementation: The framework is implemented in ROS 2 and designed for reproducibility. The source code, and simulation environments are available at: `https://github.com/alexglzg/corridor_navigation`

II. Preliminaries

In this section, we present the problem formulation and an overview of our motion planning framework.

A. Problem Formulation

We consider a non-holonomic AMR, common in industrial settings, modeled as a unicycle with state $\mathbf{x} = [x, y, \theta]^\top \in SE(2)$, where (x, y) denotes the position and θ the orientation. The robot has a circular footprint of radius a. The control vector is $\mathbf{u} = [v, \omega]^\top \in \mathscr{U}$, $\mathscr{U} = [0, v_{\max}] \times [-\omega_{\max}, \omega_{\max}]$, with v the translational velocity and ω the angular velocity. The kinematic model follows:

$$\dot{x} = v\cos\theta, \quad \dot{y} = v\sin\theta, \quad \dot{\theta} = \omega. \tag{1}$$

The environment is represented as an occupancy grid $M \in \{0,1\}^{m \times n}$ with resolution δ meters per pixel, where $M_{ij} = 0$ denotes free space and $M_{ij} = 1$ denotes obstacles. The collision-free configuration space is defined as $\mathscr{C}_{\text{free}} = \{\mathbf{x} \in SE(2) : B(\mathbf{x}, a) \subset \mathscr{F}\}$, where $B(\mathbf{x}, a)$ represents the robot's footprint at pose $\mathbf{x}$ and $\mathscr{F}$ is the free space in the workspace. Given a start pose $\mathbf{x}_s$ and goal pose $\mathbf{x}_g$, the motion planning problem seeks a trajectory $\tau : [0, T] \to SE(2)$ that minimizes the traversal time T subject to: (i) boundary conditions $\tau(0) = \mathbf{x}_s$ and $\tau(T) = \mathbf{x}_g$, (ii) collision avoidance $\tau(t) \in \mathscr{C}_{\text{free}}$ for all $t \in [0, T]$, and (iii) kinematic feasibility under the control constraints $\mathscr{U}$.

Traditional grid-based methods discretize this problem over $O(mn)$ cells, leading to computational complexity that scales with map resolution rather than environment complexity. This motivates our corridor-based decomposition, which reduces the search space to a compact graph whose size depends on the structural complexity of the environment. We define a rectangular corridor as a tuple $r = (\mathbf{c}, \mathbf{d}, \phi)$, where $\mathbf{c} \in \mathbb{R}^2$ is the center, $\mathbf{d} = [w, h]^\top \in \mathbb{R}^2_{>0}$ specifies the width and height, and $\phi \in [0, 2\pi)$ is the orientation. The corridor corresponds to the region

$$\mathscr{A}(r) = \{\mathbf{x} \in \mathbb{R}^2 : |R_\phi^\top (\mathbf{x} - \mathbf{c})| \leq \tfrac{1}{2}\mathbf{d}\}, \tag{2}$$

where R_ϕ is the rotation matrix and the inequality is interpreted element-wise.

B. Architecture Overview

We briefly outline the proposed framework and its key elements, which will be explained in detail in the subsequent

sections. Our framework employs a two-phase approach that decouples environment representation from trajectory generation, as illustrated in Fig. 1. In the offline phase, the occupancy grid undergoes corridor decomposition to produce a set of overlapping rectangles $\mathscr{R} = \{r_1, \ldots, r_{n_r}\}$, where $n_r = |\mathscr{R}|$, and their connectivity graph $G = (\mathscr{R}, E)$, where E encodes adjacency relations through rectangle overlaps. From this representation, we precompute a transition graph $G_T = (V_T, E_T)$, where V_T are feasible points to enter, travel, and exit corridors, and E_T the straight-line connections between them, with $|V_T| \ll mn$ for structured environments.

The online phase processes planning queries through three sequential steps: (i) augmenting G_T with start and goal poses, (ii) finding the shortest path in G_T using Dijkstra's algorithm to obtain a corridor sequence $\mathscr{S}$, and (iii) generating a near-time-optimal trajectory through $\mathscr{S}$ using analytical methods. This separation enables real-time performance with planning complexity $O(E_T + |V_T| \log |V_T|)$, independent of the map resolution.

III. Automatic Corridor Generation

In this section, the Automatic Corridor Generation (ACG) algorithm is described. Fig. 2 illustrates the full corridor extraction pipeline.

A. Design Objectives

The proposed algorithm extracts corridors from 2D occupancy grids to create a compact spatial representation for efficient motion planning in structured indoor environments. Given an occupancy grid $M \in \{0,1\}^{m \times n}$, a set of rectangles $\mathscr{R}$ and their connectivity graph $G = (\mathscr{R}, E)$ are computed, designed with four key objectives:

1) **Safety:** all corridors lie entirely within free space through explicit clearance margins;
2) **Coverage:** maximize free-space coverage while maintaining geometric simplicity;
3) **Compactness:** minimize the number of rectangles to reduce graph complexity;
4) **Efficiency:** achieve polynomial-time complexity scaling with structural features rather than map resolution.

B. Algorithm

1) Stage 1-2, Line Detection and Straightening: We detect wall segments using a line segment detector and cluster them by orientation. Segments are aligned to canonical directions (e.g., $\mathscr{D} = \{0^\circ, 90^\circ\}$) when within tolerance, or to their mean angle otherwise. For each segment, we compute its unit direction and project the original endpoints onto the line through their midpoint, yielding aligned endpoints $\mathbf{P}'$ and $\mathbf{Q}'$. To ensure safety, we determine the inward normal $\mathbf{n}$ by sampling the occupancy grid on both sides and selecting the direction with maximum free-space samples, such that $\mathbf{n}$ always points into navigable space. Each line is shifted inward by a clearance margin expressed in pixels, ρ, along the inward normal, giving a shifted line segment $\ell = \{\mathbf{P}' + \rho\mathbf{n}, \mathbf{Q}' + \rho\mathbf{n}\}$, which provides collision-free geometry.

2) Stage 3-4, Snap Point Extraction and Extension: We cluster nearby endpoints within a threshold d_s (snap distance, i.e., the maximum distance at which endpoints are merged) using union-find in $O(n_\ell \alpha(n_\ell))$ time, where n_ℓ is the number of line segments. Each cluster generates a snap point based on the incident line count and interior angle (see Fig. 2(d)):

- *Full snap points* (convex corners, with angle of amplitude $< 180^\circ$ measured inside free space): placed at line intersection.
- *Half snap points* (concave/hanging corners, with angle of amplitude $\geq 180^\circ$ measured inside free space): represented by two overlapping points linked as *sisters*, so they can later extend in different directions to close gaps.

Half snap points (purple in Fig. 2(d)) cast rays along their wall normal to find connection targets (represented by cast green rays in Fig. 2(e)). Each ray finds the nearest valid wall intersection, after which the half snap either merges with a collinear counterpart or extends to the nearest wall hit, closing gaps in the corridor network.

3) Stage 5-6, Face Identification and Corner Resolution: We trace boundary cycles in the snap graph and classify each as a building interior (face=0) or an obstacle (face$>$0). For each cycle with center $\mathbf{c_c}$, we initialize a score $S = 0$ and process each edge i with length l_i, midpoint $\mathbf{m}_i$, and normal $\mathbf{n}_i$. We compute $d = \mathbf{n}_i \cdot (\mathbf{c_c} - \mathbf{m}_i)$: if $d > 0$, add l_i to S; otherwise subtract l_i. Cycles with $S < 0$ are classified as obstacles. These snap points are grouped by connectivity and removed (see Fig. 2(f)).

After obstacle removal, obtuse full snap points (with angle of amplitude $> 90^\circ$ measured inside free space) are converted to *double snap points*. From each obtuse corner, we cast orthogonal rays along wall normals until they hit opposing walls, then insert extension lines to create *double snap points* that decompose the obtuse angle into two 90° turns. This transformation ensures all corners are either 90° or can be decomposed into 90° turns (acute angles are retained as snap points, but do not contribute to rectangle generation), enabling axis-aligned rectangle generation. Double snap points (shown as yellow circles in Fig. 2(g)) can spawn up to two rectangles from their orthogonal wall pairs.

4) Stage 7-8, Rectangle Generation and Obstacle Carving: We generate maximal axis-aligned rectangles by traversing snap points in priority order (double, full, then half). From each snap point, we follow incident walls to find potential rectangle corners. When four corners form a valid rectangle, it is added to the set. Half snap pairs with opposing normals define corridor rectangles connecting rooms. Successfully created rectangles then remove their corner snaps from working sets, preventing duplicates.

When rectangle r overlaps an obstacle, we split r into up to four axis-aligned fragments that surround the obstacle's bounding box (see Fig. 2(i)). Fragments with a non-positive area are discarded. This ensures safety while maintaining overlap connectivity around obstacles. Finally, if any rectangle does not comply with a minimum width or height to contain the robot footprint, it is discarded.

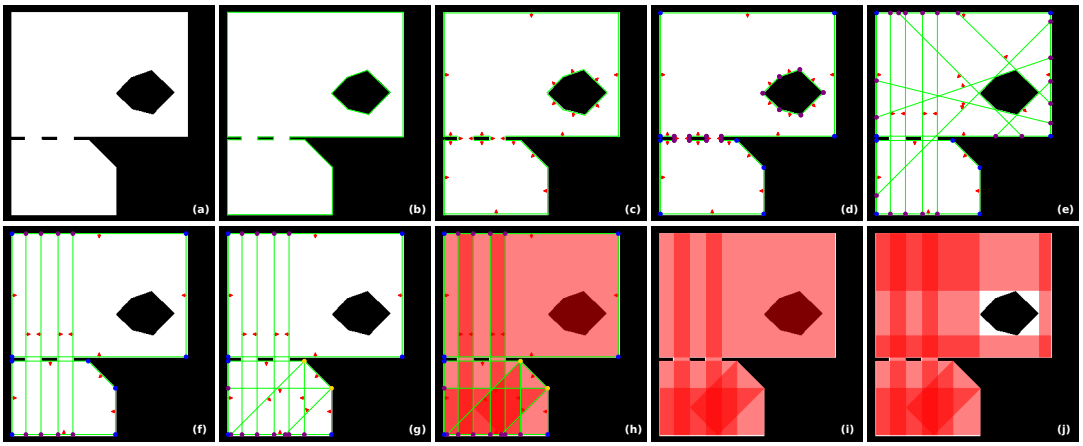

Fig. 2: Automatic corridor generation pipeline example. (a) Floor plan as a binary image. (b) Detected line segments. (c) Straightened and shifted line segments. (d) Snap point extraction, blue dots mark full snap points, purple dots mark half snap points. (e) Closed rooms and continuous hallways after extending half snap points. (f) Pruned snap graph after obstacle removal. (g) Full snap extension, yellow dots mark the new double snap points. (h) Usage of snap points and faces to construct maximal axis-aligned rectangles. (i) Rectangle generation before obstacle carving. (j) Rectangles overlapping obstacles are split into fragments, resulting in the final coverage.

5) Stage 9, Corridor Connectivity Graph Construction: We construct the corridor connectivity graph $G = (\mathscr{R}, E)$ by testing all rectangle pairs for overlap using the Separating Axis Theorem (SAT). An edge $(r_i, r_j) \in E$ exists if rectangles overlap with sufficient area to contain the robot footprint. During overlap testing, we store the intersection polygon $\mathscr{I}_{ij} = \mathscr{A}(r_i) \cap \mathscr{A}(r_j)$ for each valid edge, as these geometries enable transition point extraction for motion planning (Section IV-A).

Remark 1: The corridor graph $G = (\mathscr{R}, E)$ provides a general spatial decomposition that can serve other planning algorithms, such as [18]. While we construct a specific graph G_T for point-to-point navigation, the corridor representation could be adapted for other structured environment tasks to leverage corridor areas or connectivity [25].

6) Complexity Analysis: Dominant costs arise from line sorting $O(n_\ell \log n_\ell)$; snap clustering with union-find $O(n_\ell \alpha(n_\ell))$; snap extension requiring wall intersection tests $O(k_h \cdot n_\ell)$, where k_h is the number of half snap points; rectangle generation with snap traversal $O(k^2)$ where k is the number of snap points; and SAT-based overlap testing $O(n_r^2)$. Since $\alpha(n)$ is effectively constant for all practical values, and typically $k_h \leq k \approx n_\ell$ for structured maps, the corridor decomposition complexity simplifies to:

$$O(n_\ell \log n_\ell + k n_\ell + k^2 + n_r^2) \tag{3}$$

where the rectangle count n_r depends on map complexity rather than area, ensuring scalability.

Remark 2: The rectangle count n_r is bounded by structural features of the environment. Let k_f, k_d, and k_h denote the number of full, double, and half snap points, respectively. Since each double snap generates at most 2 rectangles, each full snap at most 1, and each half snap pair at most 1, the rectangle count before obstacle carving satisfies $n_r \leq 2k_d + k_f + \lceil k_h/2 \rceil$. As snap points arise from wall-corner clusters, n_r scales with the number of corners and junctions in the floor plan, which is invariant to map resolution.

Remark 3: The user can opt to include or ignore obstacles during corridor generation, which bypasses the face identification (Fig. 2(f)) and obstacle carving steps (Fig. 2(j)), thereby reducing algorithmic complexity.

IV. Corridor-Based Motion Planning

In this section, we describe the pipeline for efficient planning based on the proposed rectangular corridor free-space representation.

A. Transition Graph Construction

Given the corridor connectivity graph $G = (\mathscr{R}, E)$ from Section III-B.5, we construct a planning graph $G_T = (V_T, E_T)$ that transforms spatial relationships into a searchable structure. While G captures which corridors connect, G_T specifies where the robot can transition between them. The nodes V_T consist of transition points, including corridor centers $\mathbf{c}_i$, and points extracted from the stored intersection geometries $\mathscr{I}_{ij}$, i.e., intersection centroids and corners. Edges connect points $\mathbf{p}_a, \mathbf{p}_b \in V_T$ if the line segment $\overline{\mathbf{p}_a \mathbf{p}_b}$ lies entirely within at least one corridor, ensuring collision-free paths. This precomputation executes once per map with complexity $O(|E| \cdot |V_T|^2)$.

1) Corridor Sequence Planning: Given start $\mathbf{p}_s$ and goal $\mathbf{p}_g$ positions, we identify their containing corridor and, if a direct path is viable within a single corridor, we use that path. Otherwise, we augment the precomputed G_T with temporary nodes for the start and goal positions, connecting them to reachable transition points within their respective containing corridors. A shortest path is then computed using Dijkstra's algorithm, with edge weights $w(e)$ defined by a combination of Euclidean distance and a penalty for corridor transitions:

$$w(e) = \|\mathbf{p}_i - \mathbf{p}_j\|_2 + \lambda \cdot 1[\text{corridor transition}] \tag{4}$$

where $\mathbf{p}_i, \mathbf{p}_j$ are the positions of the connected nodes, $1[\cdot]$ is the indicator function and $\lambda \geq 0$ penalizes corridor changes. This shortest path yields a sequence of transition points $W = (\mathbf{p}_s, \mathbf{t}_1, \ldots, \mathbf{t}_{n_t}, \mathbf{p}_g)$, where each $\mathbf{t}_i \in V_T$ represents a transition point between corridors. These transition points induce a corridor sequence $S = (s_1, \ldots, s_{n_s})$, $n_s = |S|$, where each $s_j \in \mathscr{R}$, by tracking which corridors contain consecutive points. We remove redundant transitions to further optimize this sequence. Next, a traversal direction is computed for each corridor based on the waypoints, with angles snapped to the nearest axis-aligned direction. When a corridor exceeds a width/height or height/width set ratio, it follows its longest directed axis. The directed sequence $\tilde{S} = (\tilde{s}_1, \ldots, \tilde{s}_{n_s})$ is then passed to a dedicated analytical planner (AP) that generates a smooth, near-time-optimal, and collision-free trajectory respecting the vehicle's dynamic constraints. The online sequence-planning process is dominated by the graph search, with a time complexity of $O(E_T + |V_T| \log |V_T|)$.

Remark 4: The corridor sequence S computation is a general solution independent of system dynamics or planning objectives. Thus, it can be paired with other algorithms for trajectory generation through convex sets, such as [7], [20].

B. Analytical Motion Planning

The AP generates trajectories by concatenating time-optimal-based motion primitives within the free space defined by the directed corridor sequence $\tilde{S}$. The method presented in [24] delivered near time-optimal solutions in two-corridor scenarios, where OCP approaches were still tractable for comparison. Its slight suboptimality stems from heuristic rules used to place the time-optimal primitives in constrained environments. We extend this idea to sequences of two or more corridors, and consider additional heuristic rules to address maps containing long corridors and narrow passages. A brief description of the approach is provided below, with an emphasis on the new rules.

For the unicycle model (1), three time-optimal-based motion primitives are defined: on-the-spot rotations $T^\bullet$ with $v(t) = 0$ and $\omega(t) = \pm\omega_{\max}$, circular arcs $C^\bullet$ with $v(t) = v_{\max}$ and $\omega(t) = \pm\omega_{\max}$ (turning radius $\rho_t = v_{\max}/\omega_{\max}$), and straight line segments S with $v(t) = v_{\max}$ and $\omega(t) = 0$; in $T^\bullet$ and $C^\bullet$, $\bullet \in \{+, -\}$ indicates the sign of $\omega(t)$.

The key principle behind the planner is to decompose the trajectory computation into smaller, decoupled pieces. We obtain this subdivision by placing an *intermediate circle* o_j with radius ρ_t between each two consecutive corridors $(\tilde{s}_j, \tilde{s}_{j+1})$ for $j = 1, \ldots, n_s - 1$, where $n_s = |\tilde{S}| \geq 2$. This circle serves as an intermediate goal, guiding the robot from one corridor to the next while ensuring it remains within the corridor boundaries. We place the center of each o_j either to the right or to the left of $(\tilde{s}_j, \tilde{s}_{j+1})$, relative to their traversal direction, depending on whether a clockwise or counterclockwise rotation is required to align $\tilde{s}_j$ with $\tilde{s}_{j+1}$. Accordingly, we perform the transition from $\tilde{s}_j$ to $\tilde{s}_{j+1}$ by executing a clockwise or counterclockwise circular arc along o_j. In contrast to [24], a first additional rule is introduced for the case where $\tilde{s}_j$ and $\tilde{s}_{j+1}$ share the same traversal direction, which would make the placement of the intermediate circle o_j indeterminate. In this situation, we consider the relative orientation of $(\tilde{s}_j, \tilde{s}_{j+q})$, $q = 2$, and we repeat the procedure necessary by increasing q until a change in direction is detected, or until the final corridor is reached. In the latter case, we determine the last corridor's rotation direction with the angle of the line connecting the center of the penultimate corridor to the final target position.

In general, the solution trajectory is composed of $2N + 3$ motion primitives:

$$T_1^\bullet C_2^\bullet S_3 C_4^\bullet S_5 C_6^\bullet \ldots S_{2N-1} C_{2N}^\bullet S_{2N+1} C_{2N+2}^\bullet T_{2N+3}^\bullet, \qquad (5)$$

where the subscripts indicate the order of appearance of each primitive in the sequence. The overall sequence is obtained by first computing independent trajectory pieces within each corridor, and then connecting them through the circular arcs $C_4^\bullet, C_6^\bullet, \ldots, C_{2N}^\bullet$. The independent trajectory pieces are (i) the initial sequence $T_1^\bullet C_2^\bullet S_3$, connecting the start pose $\mathbf{x}_s$ to o_1; (ii) the segments $S_5, S_7, \ldots, S_{2N-1}$, each of them connecting two consecutive circles o_j, o_{j+1}, $j = 1, \ldots, n_s - 2$; (iii) the final sequence $S_{2N+1} C_{2N+2}^\bullet T_{2N+3}^\bullet$, connecting o_{n_s-1} to the end pose $\mathbf{x}_g$.

We introduce a second additional rule when two or more intermediate circles are closer than a distance ρ_t and share the same direction of rotation. Such a situation often arises in door-like scenarios, where only a short portion of a corridor is traversed. In this case, we merge the circles into a single one, with its position adjusted to avoid collisions with the corridor walls. As a result, the number of motion primitives in the solution sequence (5) is reduced. Finally, each pair of segments associated with an intermediate circle o_j is checked for intersection. If the segments intersect, no arc maneuver is needed to move from $\tilde{s}_j$ to $\tilde{s}_{j+1}$, and the number of motion primitives in (5) is reduced. In particular, the trajectory is updated depending on j: for $j = 1$, the initial portion is recomputed to connect the start pose to o_2; for $j = 2, \ldots, n_s - 2$, the two segments are replaced by a single segment connecting o_{j-1} to o_{j+1}; and for $j = n_s - 1$, the final portion is recomputed to connect o_{n_s-1} to the goal pose. This process is repeated until no intersections remain.

V. Results

In this section, we present simulation and experimental results of our proposed motion planning framework. We begin by detailing the software implementation. Next, we provide a quantitative analysis of the proposed ACG algorithm. Then, we evaluate the performance of our complete motion planning framework against standard open-source planners from the Robot Operating System (ROS) 2 Nav2 Stack [26]. Finally, we present an experimental validation with a physical robot.

A. Software Implementation

The proposed framework was implemented as a modular prototype in `Python` within the ROS 2 [27] ecosystem. Graph operations are handled with `NetworkX` [28]. The architecture is composed of two primary ROS nodes, which

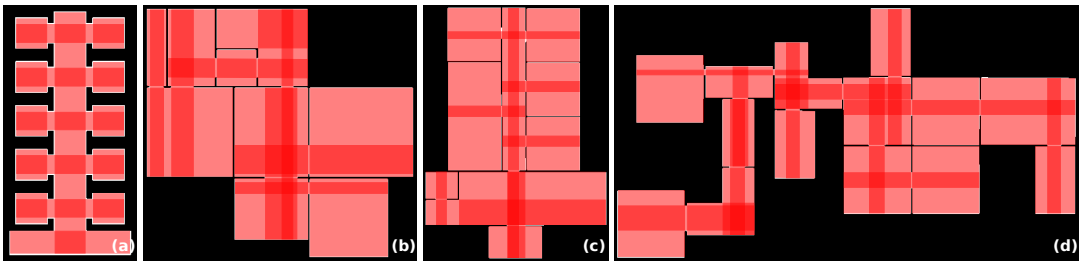

Fig. 3: Representative Corridor Decompositions. (a) Small map (330x630 pixels) with 17 rectangles. (b)-(c) Large maps (1744x1624, 1738x2395 pixels) with 17 rectangles. (d) Large map (3444x1891 pixels) with 27 rectangles.

TABLE I: Corridor Generation Summary

Category	# of Maps	Avg Size (Pixels)	Rectangles $\in \mathscr{R}$ (Avg)	Nodes $\in V_T$ (Avg)	Avg Compression	Avg Time (ms)
Small	12	430K	3-17 (10)	15-117 (59)	12,000:1	31 $\pm$ 15.90
Large	12	3.9M	11-27 (20)	103-234 (166)	24,000:1	169 $\pm$ 66.35
Overall	24	2.2M	3-27 (15)	15-234 (113)	18,000:1	100 $\pm$ 86.51

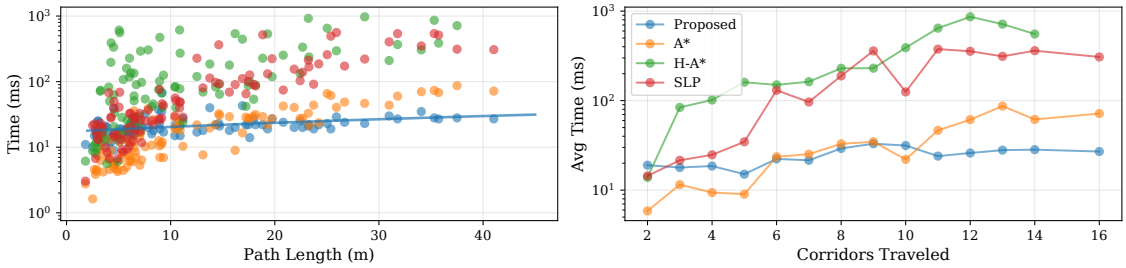

Fig. 4: Motion Planning Benchmark. (Left) Computation Time vs Path Length. (Right) Average Computation Time vs Number of Corridors Traveled. Hybrid-A* failed to solve for some long paths.

TABLE II: Corridor Generation Scalability

Map Pixels	Nodes	Structural Compression	Time (ms)
208K	27	7,700:1	16
1.3M	103	12,500:1	67
4.2M	139	30,000:1	162
7.3M	159	46,000:1	279

separate the offline corridor generation process from the real-time online planning tasks. All simulations were conducted on an Intel Core i7-12800H CPU with 32 GB RAM running Ubuntu 24.04 LTS and ROS 2 Jazzy inside a Docker container.

B. Automatic Corridor Generation

We present an evaluation of the proposed ACG algorithm on 24 synthetic environments, ranging from 330x630 to 2430x3010 pixels, mostly drawn from a large-scale dataset of indoor layouts [29]. While our method can handle environments with sparse obstacles, we evaluate on obstacle-free maps to focus on the algorithm's core strength: capturing global navigational structure. In practice, dense local obstacles are better addressed through hierarchical planning, where our corridor decomposition provides global routes and local planners can handle dynamic obstacles [6]. Table I summarizes the overall performance, demonstrating average compression ratios of 18,000:1 with generation times under 300ms for all tested maps. The algorithm shows consistent behavior across map categories, with the number of nodes scaling primarily with environmental complexity (3-27 rectangles) rather than resolution. Table II further illustrates this scalability through representative examples. This structural compression enables storing a 7.3M-pixel map ($\approx$7MB uncompressed) as a 159-node graph ($<$10KB), achieving over a 10,000$\times$ memory reduction critical for embedded AMR systems. Fig. 3 shows illustrative examples of the corridor generation.

C. Motion Planning Framework

We performed a comparison of the proposed motion planning framework with open-source motion planners, taken from the ROS 2 Nav2 Stack. We compared against their A* implementation, which provides a baseline for computational performance without considering kinematic feasibility, and against their Smac Hybrid-A* (H-A*) and their Smac State Lattice Planner (SLP), which consider kinematic feasibility for differential drive AMRs through Reeds-Shepp curves and motion primitives, respectively [30]. We considered a footprint radius of 0.34m, $v_{max} = 0.5$m/s, $\omega_{max} = 2.0$rad/s. For H-A* and SLP, we used the turning radius $\rho_t = 0.25$m,

TABLE III: Motion Planning Computational Performance Comparison

Query Type	# of Samples	Path Length (m)	# of Corridors	Proposed (ms)		A* (ms)	H-A* (ms)	SLP (ms)
				Framework	AP			
SHORT	56	2-9	2-8	19.2 ± 9.2	1.28 ± 0.38	9.4 ± 6.9	101 ± 138.54	24 ± 15
MEDIUM	32	10-24	3-11	22.2 ± 6.0	1.49 ± 0.32	25.5 ± 8.5	210 ± 212.44	126 ± 101
LONG	12	25-41	8-16	26.7 ± 4.1	1.77 ± 0.38	61.5 ± 14.0	530 ± 276.51	389 ± 137
OVERALL	100	2-41	2-16	21.1 ± 8.1	1.41 ± 0.4	20.8 ± 18.8	176 ± 217.7	100 ± 138

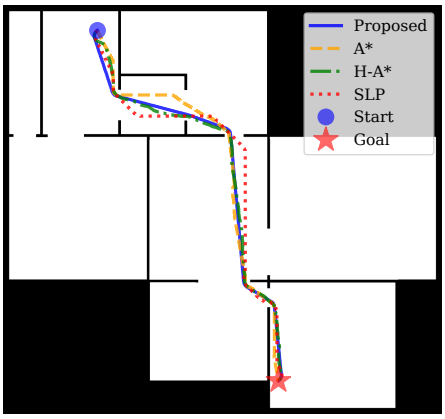

Fig. 5: Illustrative Motion Planning Comparison. This medium query has a path of 18m, traveling through 8 corridors. Compared to A* (31.14ms), Smac Hybrid-A* (467.78ms), and Smac State Lattice Planner (119.28ms), the proposed approach (23ms) achieves the fastest computation time while planning a kinematically feasible trajectory.

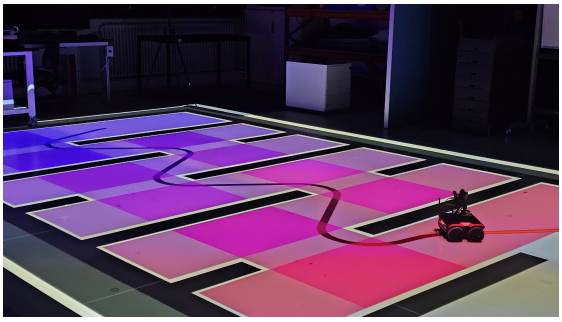

Fig. 6: Experimental validation. Laboratory setup with ROSbot 3 and a projected virtual environment, showing a planned corridor sequence and trajectory pair.

and 16 headings to build the motion primitives. We evaluated 10 maps, testing 10 queries of start-goal pose pairs per map. Through these maps, we tested with increasing path lengths and complexity, measured with the number of corridors traveled, to evaluate the planning time and scalability. A summary of the results is described in Table III. Through different configurations of map size, path length, and corridors traveled, our motion planning framework consistently achieves planning times of 20-40ms. Compared to A*, our proposed framework achieves competitive computational performance on paths shorter than 10m, while achieving a speedup of 2.3× on paths larger than 25m, despite the `Python` interpreted language overhead. This suggests that a high-performance C++ implementation could achieve even faster planning times. In comparison to the kinematically-feasible planners, we achieve computation times an order of magnitude faster on medium and long paths while computing trajectories instead of geometric paths. In addition, most of the overhead in our method comes with the graph augmentation with start/goal points, and the graph search, as the AP remains consistently around 1-2ms, regardless of the path length or corridors traveled. Fig. 4 further illustrates the scalability properties of our method in comparison to A*, H-A*, and SLP. Fig. 5 shows an example query, where our proposed approach achieves the fastest planning time.

D. Experimental Evaluation

The proposed motion planning framework was deployed using a differential drive four-wheeled mobile platform, the Husarion ROSbot 3, with an onboard Raspberry Pi 5 running ROS 2. The robot is localized using an HTC VIVE Tracker 3.0 mounted on the vehicle, along with three HTC VIVE Base Stations. Virtual environmental features are visualized on the laboratory floor using four projectors mounted on the ceiling. With this setup, it is possible to project occupancy grid maps, current corridor sequences, motion plans, etc. In addition to the motion planning ROS nodes, a low-level controller node based on Model Predictive Control (MPC), and a localization node were deployed. The MPC was formulated using `CasADi` [31] and trajectory optimization solver `FATROP` [32]. Sensing and actuation ran onboard the robot, whereas the MPC and motion planning framework nodes ran off-board on an Intel Core i9-9900X CPU @ 3.50GHz with 16 GB RAM. Fig. 6 shows the experimental setup. We validated the framework across 8 different map configurations of 3×6m, with planning times averaging 20±5ms, and corridor generation under 40ms, consistent with simulation results. The accompanying video demonstrates real-time map switching, planning, and replanning.

VI. Conclusion

This paper introduced a framework for motion planning of non-holonomic AMRs based on a deterministic rectangular corridor free-space decomposition. By reducing the search space to a graph whose size depends on map complexity rather than resolution, exceeding structural compression ratios of 10,000:1, the method achieves a 2.3× speedup on long paths compared to conventional grid-based approaches, while directly generating near-time-optimal, kinematically feasible

trajectories. In comparison to kinematically-feasible geometric path planners, our framework shows computing times an order of magnitude faster. The framework's efficiency was validated in simulation across a variety of maps and experimentally on a real robot.

Despite these advantages, the framework has some limitations. The corridor construction method is tailored for structured spaces and faces challenges in maps with irregular geometry or fine features. While our method excels in scenarios where a significant reduction in search space is achieved, its benefits diminish on short paths or in cases with few corridors. Moreover, the AP currently enforces only velocity constraints, leaving acceleration limits to be addressed by local planning/control layers. Finally, unforeseen situations continue to pose a challenge to the AP's heuristic rules.

Future work will focus on extending the corridor representation to more complex and cluttered environments, developing tuning guidelines based on map resolution, extending the motion planning framework to axis-unaligned corridors, and incorporating bicycle-model vehicles.

REFERENCES

[1] G. Fragapane, R. de Koster, F. Sgarbossa, and J. O. Strandhagen, "Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda," *European Journal of Operational Research*, vol. 294, no. 2, pp. 405–426, 2021.

[2] I. R. da Costa Barros and T. P. Nascimento, "Robotic Mobile Fulfillment Systems: A survey on recent developments and research opportunities," *Robotics and Autonomous Systems*, vol. 137, p. 103729, 2021.

[3] J. Pak, J. Kim, Y. Park, and H. I. Son, "Field Evaluation of Path-Planning Algorithms for Autonomous Mobile Robot in Smart Farms," *IEEE Access*, vol. 10, pp. 60 253–60 266, 2022.

[4] S. N. Young, E. Kayacan, and J. M. Peschel, "Design and field evaluation of a ground robot for high-throughput phenotyping of energy sorghum," *Precision Agriculture*, vol. 20, no. 4, pp. 697–722, 2019.

[5] M. H. Ko, B.-S. Ryuh, K. C. Kim, A. Suprem, and N. P. Mahalik, "Autonomous Greenhouse Mobile Robot Driving Strategies From System Integration Perspective: Review and Application," *IEEE/ASME Transactions on Mechatronics*, vol. 20, no. 4, pp. 1705–1716, 2015.

[6] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *Journal of Intelligent & Robotic Systems*, vol. 91, no. 1, pp. 35–58, 2018.

[7] T. Marcucci, M. Halm, W. Yang, D. Lee, and A. D. Marchese, "A Biconvex Method for Minimum-Time Motion Planning Through Sequences of Convex Sets," in *Proceedings of Robotics: Science and Systems*, 2025.

[8] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[9] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs." *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[10] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 1989.

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

[12] M. Likhachev and D. Ferguson, "Planning Long Dynamically Feasible Maneuvers for Autonomous Vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[13] A. Botros and S. L. Smith, "Spatio-Temporal Lattice Planning Using Optimal Motion Primitives," *IEEE Transactions on Intelligent Transportation Systems*, vol. 24, no. 11, pp. 11 950–11 962, 11 2023.

[14] K. Bergman, O. Ljungqvist, and D. Axehill, "Improved Path Planning by Tightly Combining Lattice-Based Path Planning and Optimal Control," *IEEE Transactions on Intelligent Vehicles*, vol. 6, no. 1, pp. 57–66, 3 2021.

[15] L. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[16] J. Kuffner and S. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, 2000, pp. 995–1001 vol.2.

[17] M. Elbanhawi and M. Simic, "Sampling-Based Robot Motion Planning: A Review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[18] 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.

[19] Y. Wu, I. Spasojevic, P. Chaudhari, and V. Kumar, "Towards Optimizing a Convex Cover of Collision-Free Space for Trajectory Generation," *IEEE Robotics and Automation Letters*, vol. 10, no. 5, pp. 4762–4769, 2025.

[20] R. Deits and R. Tedrake, "Efficient mixed-integer planning for UAVs in cluttered environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, 2015, pp. 42–49.

[21] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-D complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 7 2017.

[22] Q. Wang, Z. Wang, M. Wang, J. Ji, Z. Han, T. Wu, R. Jin, Y. Gao, C. Xu, and F. Gao, "Fast Iterative Region Inflation for Computing Large 2-D/3-D Convex Regions of Obstacle-Free Space," *IEEE Transactions on Robotics*, vol. 41, pp. 3223–3243, 2025.

[23] P. Werner, R. Cheng, T. Stewart, R. Tedrake, and D. Rus, "Superfast configuration-space convex set computation on GPUs for online motion planning," in *Proceedings of Robotics: Science and Systems*, 2025.

[24] S. De Santis, A. Astudillo, W. Decré, and J. Swevers, "Towards the Analytical Computation of Time-Optimal Trajectories for Unicycle Robots in Corridor Environments," in *2024 European Control Conference (ECC)*, 2024, pp. 3239–3246.

[25] K. de Vos, E. Torta, H. Bruyninckx, C. A. López Martínez, and M. J. G. van de Molengraft, "Automatic configuration of multi-agent model predictive controllers based on semantic graph world models," in *2024 IEEE International Conference on Robotics and Automation (ICRA)*, 2024, pp. 7194–7200.

[26] S. Macenski, F. Martin, R. White, and J. Ginés Clavero, "The Marathon 2: A Navigation System," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[27] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science robotics*, vol. 7, no. 66, p. eabm6074, 2022.

[28] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in *Proceedings of the 7th Python in Science Conference*, Pasadena, CA USA, 2008, pp. 11 – 15.

[29] T. Li, D. Ho, C. Li, D. Zhu, C. Wang, and M. Q.-H. Meng, "Houseexpo: A large-scale 2d indoor layout dataset for learning-based algorithms on mobile robots," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 5839–5846.

[30] S. Macenski, M. Booker, and J. Wallace, "Open-Source, Cost-Aware Kinematically Feasible Planning for Mobile and Surface Robotics," *Arxiv*, 2024.

[31] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings, and M. Diehl, "CasADi – A software framework for nonlinear optimization and optimal control," *Mathematical Programming Computation*, vol. 11, no. 1, pp. 1–36, 2019.

[32] L. Vanroye, A. Sathya, J. De Schutter, and W. Decré, "Fatrop: A fast constrained optimal control problem solver for robot trajectory optimization and control," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2023.