SE(2) Navigation Mesh

***Abstract*—Global navigation for ground robots in complex multi-level environments requires representations that accurately capture traversable regions while enabling efficient path planning. Current approaches present key limitations: Point clouds and volumetric occupancy maps lack explicit surface structure for traversability estimation, whereas direct pathfinding on dense triangle meshes is computationally prohibitive. Navigation meshes mitigate these challenges through polygonal abstraction of the underlying mesh, but assume yaw-invariant traversability, rendering them unsuitable for non-circular robots in constrained spaces. We propose SE(2) Navigation Mesh (SE(2) NavMesh), a polygonal representation of traversable regions that encodes yaw-dependent traversability. Our method evaluates traversability using footprint masks and constructs a graph over yaw-specific layers with explicit translational and rotational connectivity. Grounded in this representation, we develop an A*-String Pulling-A* (ASA) pathfinding strategy that hierarchically optimizes robot position and heading. We also present an online method that incrementally updates the SE(2) NavMesh from streaming point clouds during concurrent geometry reconstruction. In simulation, the SE(2) NavMesh captures over $50\,\%$ more traversable area than classical NavMeshes, and the SE(2) NavMesh + ASA pipeline consistently outperforms sampling-based baselines in constrained environments. Extensive real-world experiments on a physical robot validate real-time online generation and successful navigation across multiple environments.**

***Index Terms*—legged robots, motion and path planning, computational geometry, global navigation**

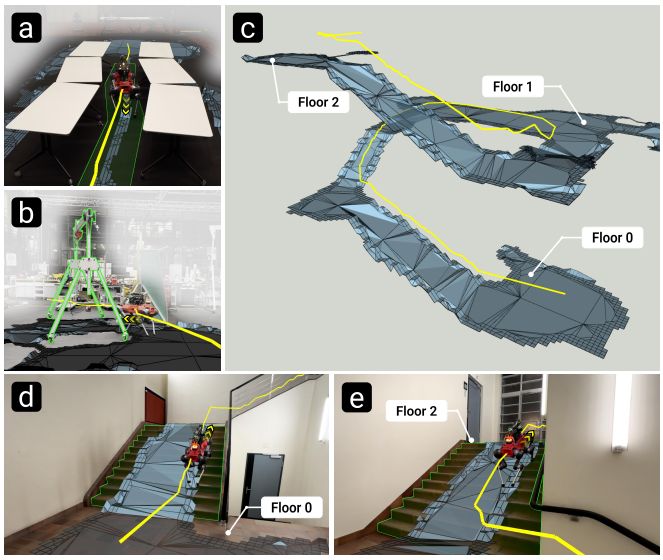


Fig. 1: Real-world robot navigation on the SE(2) NavMesh. Yellow polylines indicate planned paths, and blue polygons represent traversable regions. SE(2) NavMesh captures yaw-dependent traversability in constrained scenes such as (a) narrow passages ($0.8\,\mathrm{m}$ wide in this example, only $0.27\,\mathrm{m}$ wider than the robot) and (b) overhanging obstacles. It also supports multi-level structures (c), enabling successful stair traversal across floors as demonstrated in (d) and (e).

I. Introduction

Ground robots are increasingly deployed for infrastructure and industrial facility inspection [1], [2], owing to their ability to carry heavy payloads and support long-duration missions. Traditionally, these operations have been performed by wheeled robots on relatively flat terrain [3]–[5]. However, many facilities—such as chemical plants, power stations, and oil rigs—contain stairs, elevated platforms, and uneven terrain that are inaccessible to wheeled robots. Recent advances in legged locomotion have enabled legged robots to traverse rough terrain, stairs, and narrow structures [6], [7], making them promising platforms for inspection tasks [8], [9].

Despite these advancing locomotion capabilities, accurately representing traversable regions that capture complex geometry while enabling efficient global navigation remains an open challenge. Point clouds are a common representation of complex environments [10]–[12]. Their lack of structural organization, however, makes point clouds inefficient for traversability estimation, as geometric surfaces and connectivity must be inferred through additional processing [13], [14], leading to substantial computational overhead in large-scale environments [15]. From point cloud observations, volumetric representations such as occupancy maps, truncated signed distance fields (TSDFs), and Euclidean signed distance fields (ESDFs) can be constructed [16], [17]. These representations enable efficient 3D obstacle reasoning and are widely used for aerial robot navigation [18], [19]. Nevertheless, they do not explicitly preserve terrain surface structure, making fine-grained traversability estimation difficult without resorting to point cloud-based geometric analysis or learned models [20], [21]. At a lower dimensionality, 2D grid maps [22] and elevation maps [23], [24] discretize the environment into regular grids. 2D grid maps are effective in simple planar environments [4], [5] but fail to represent uneven terrain or multi-floor scenes. Elevation maps, while widely used for legged robot locomotion and local navigation [23]–[25], are inherently single-layer and cannot represent multi-level structures where surfaces overlap at the same planar location.

Moving to surface-based representations, triangle meshes are commonly used in large-scale 3D scene datasets such as HM3D [26] and HSSD [27]. By preserving detailed surface geometry and adjacency relationships between surfaces, triangle meshes support traversability analysis and graph-based pathfinding [28], [29]. Yet, accurately representing complex environments requires dense triangle meshes, which incur substantial pathfinding overhead. Navigation Meshes (NavMeshes) address this by abstracting traversable regions into connected convex polygons [30] for pathfinding and are widely used in game engines [31], [32] and, more recently, in robot navigation [33], [34]. However, classical NavMesh [35]

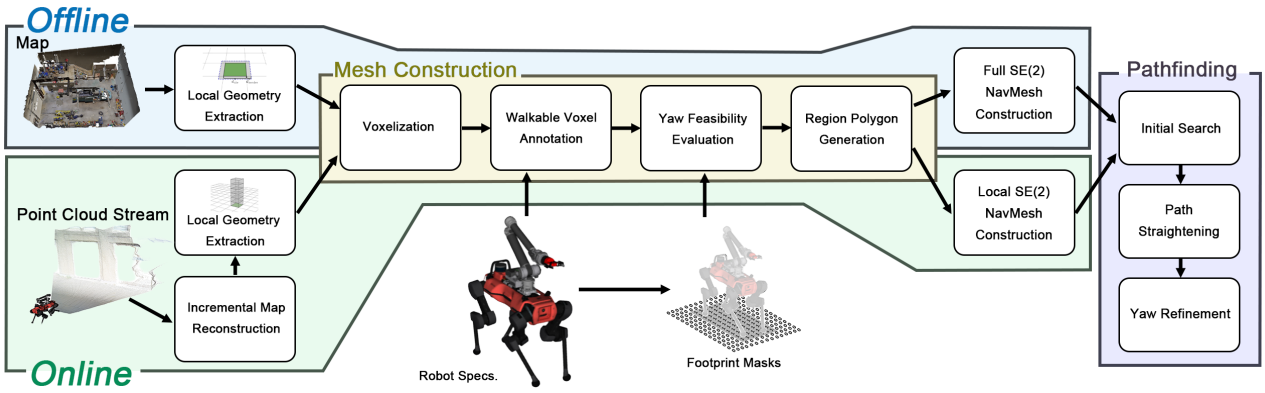


Fig. 2: Overview of the SE(2) NavMesh system. The offline pipeline constructs the mesh from a pre-built map, whereas the online pipeline updates it from a point cloud stream. During mesh construction, map geometry is processed using the robot-specific traversal constraints including traversal capabilities, height, and footprint masks. Path planning is then performed on the generated SE(2) NavMesh using a three-stage pathfinding strategy.

assumes yaw-invariant traversability. While this simplifies mesh construction and planning, it fails to capture yaw-dependent traversability for ground robots with non-circular footprints in narrow environments [34]. This limitation motivates a richer representation that explicitly accounts for robot heading during both navigation mesh construction and path planning.

In this work, we propose an SE(2) Navigation Mesh (SE(2) NavMesh) representation for global navigation of ground robots in complex 3D environments. SE(2) NavMesh extends the classical NavMesh by incorporating the robot's yaw into navigation mesh construction, enabling more accurate traversability estimation and navigation in constrained scenes, as shown in Figure 1. The yaw-encoded structure allows global path planning to explicitly consider both the robot's position and heading. Since the robot's motion is constrained to the surface of the navigation mesh polygons, the resulting navigation space has two translational (x, y) and one rotational (yaw) degree of freedom, motivating the SE(2) naming. Built on this representation, we develop an A*-String Pulling-A* (ASA) pathfinding strategy that leverages the yaw-aware layered structure of the SE(2) NavMesh to hierarchically optimize robot position and yaw. We further introduce an online SE(2) NavMesh generation framework that incrementally constructs a navigation mesh from environment geometry reconstructed in real time from onboard sensor observations. We validate the proposed approaches through experiments in both simulated and real-world environments. The main contributions of this work are summarized as follows:

- We propose SE(2) NavMesh, a novel representation that encodes yaw-dependent feasibility of traversable regions. Using footprint masks for traversability estimation, SE(2) NavMesh represents traversable narrow structures more accurately than the classical NavMesh, which we demonstrate across different scenes.
- We propose a three-stage ASA pathfinding strategy for the SE(2) NavMesh that exploits the layered structure of the SE(2) NavMesh to hierarchically optimize robot position and yaw. The proposed strategy reduces the geometric path length by $6.2\,\%$ and lowers the final path cost to $87\,\%$ of the initial path cost.
- We present an online SE(2) NavMesh generation method that incrementally constructs and updates the navigation mesh from point cloud observations. Combined with concurrent geometry reconstruction, the method runs efficiently on onboard CPUs, enabling real-time updates for real-world deployment.

An overview of the proposed SE(2) NavMesh pipeline is illustrated in Figure 2 and detailed in Section V. A project page with an interactive path planning explorer and videos of real-world experiments is available at https://se2-navmesh.github.io/. Our implementation will be released as open source to support reproducibility and future research.

II. Related Work

Point clouds are a common representation of complex environments, which can be constructed through prior scanning [36] or registered online during robot operation [37], [38] with widely used 3D sensors such as LiDAR or RGB-D cameras. To enable ground robot navigation on point clouds, Liu [13] applied tensor voting [39] to infer local geometry and defined a Riemannian metric-based motion cost that favors planar regions while penalizing rough terrain. The method performs global planning via Dijkstra [40] search on a graph constructed by connecting each point to its k-nearest neighbors (k-NN). A key limitation of this approach is that it neglects the robot's geometry and traversal capabilities, such as maximum climbing height and inclination limits, potentially yielding infeasible paths. Krüsi et al. [14] performed global motion planning directly on point clouds using sampling-based planners, including RRT [41] and RRT* [42]. Local terrain traversability was assessed by estimating surface orientation via principal component analysis (PCA) of neighboring points [43] and evaluating roughness over robot footprint-sized planar patches. While sampling-based planners avoid exhaustive traversability estimation over the full point cloud, statistics-based traversability evaluation may fail to

accurately capture traversability in complex terrain. More broadly, the lack of structural organization in point clouds makes traversability estimation challenging, as geometric surfaces and connectivity must be inferred through additional processing that is error-prone and computationally expensive.

3D occupancy maps discretize the environment into voxels classified as free, occupied, or unknown, represented using uniform voxel grids [19], [44], [45] or memory-efficient octree structures [17], [46]. Frey et al. [21] trained a sparse 3D convolutional neural network (CNN) over diverse simulated terrains to predict voxel-wise traversability risks from volumetric occupancy maps for global navigation. This learned traversability model is coupled to a specific locomotion policy and robot platform, making it difficult to parameterize for different robots. Furthermore, the predicted traversability risks must be combined with volumetric collision checking for obstacle avoidance [47]. Wang et al. [20] extracted traversable ground regions from point clouds according to robot-specific constraints and performed grid-based path planning using a motion cost that combines ESDF-based collision avoidance with terrain roughness estimated from local plane fitting. Since traversability is evaluated using local slope estimates between neighboring points, stairs may be misclassified as non-traversable, restricting navigation for stair-capable robots. Li et al. [48] represented the environment using an octree-based map with normal distributions transform (NDT) Gaussian distributions stored in occupied voxels [49]. Traversability is estimated from terrain metrics, including roughness, slope, and observation sparsity, followed by global path planning on a traversable voxel graph. Although the implicit Gaussian representation improves terrain characterization, the richer map representation increases the computational cost of online map updates.

Compared with 3D occupancy maps, 2D grid maps offer a simpler representation. Multiple layers can encode different obstacle types or constraints, yielding layered 2D grid maps [50]. Kim et al. [4] used an unmanned aerial vehicle to acquire an initial 3D point cloud, which was discretized into a 2D grid-based navigation map for subsequent ground robot navigation, with cell traversability determined by terrain slope and height clearance. Planar grids, however, cannot represent terrain geometry, limiting their applicability in complex environments.

2.5D elevation maps [23] represent terrain using continuous height values and have been widely adopted for legged robot navigation over complex terrain. Wellhausen et al. [51] proposed a reachability-based planner in which foothold feasibility is predicted from elevation maps using a CNN, with a customized LazyPRM* [52] and adaptive sampling to achieve real-time performance. Yang et al. [53] introduced a neural network that predicts energy cost, traversal duration, and failure probability from local elevation maps and motion commands, integrated into a GPU-accelerated framework combining graph search and gradient-based optimization for real-time navigation. Although effective on uneven terrain, the 2.5D nature of elevation maps fundamentally precludes their use in multi-floor environments. Yang et al. [15] addressed this by representing 3D environments as multiple 2.5D tomographic slices, computing traversability costs from ground geometry and ground-to-ceiling clearance, and performing A* search across slices via inter-slice transitions. The approach still assumes a circumscribed circular footprint, which may be overly conservative in narrow passages.

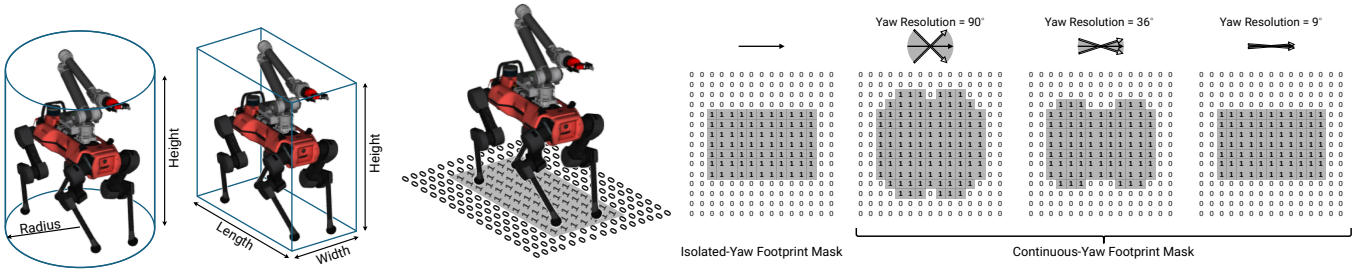
TABLE I: Capability comparison of ground robot navigation methods

	Multi-level Env.	Stair Handling	Yaw-dep. Trav.	Config. Footprint	Config. Trav. Cap.	Online Gen.
Liu [13]	✓	✓	✗	✗	✗	✗
Krüsi et al. [14]	✓	✓	✗	✓	✓	✓
Frey et al. [21]	✓	✓	✗	✗	✗	✓
Wang et al. [20]	✓	✗	✗	✓	✓	✗
Kim et al. [4]	✗	✗	✗	✓	✓	✗
Li et al. [48]	✓	✓	✗	✓	✓	✓
Wellhausen et al. [51]	✗	✓	✗	✗	✗	✓
Yang et al. [53]	✗	✓	✓	✗	✗	✓
Yang et al. [15]	✓	✓	✗	✓	✓	✗
OVPC Mesh [28]	✗	✗	✗	✓	✓	✓
Pütz et al. [29]	✓	✗	✗	✓	✓	✗
GaitMesh [34]	✓	✓	✗	✓	✓	✗
SE(2) NavMesh	✓	✓	✓	✓	✓	✓

Multi-level Env.: Supports navigation across multi-level environments.
Stair Handling: Handles discrete elevation changes such as stairs.
Yaw-dep. Trav.: Evaluates traversability of target terrain across different yaws.
Config. Footprint: Supports configurable robot footprint.
Config. Trav. Cap.: Supports configurable robot traversal capabilities (maximum step height and slope angle).
Online Gen.: Generates maps (and search graphs) incrementally during operation.

Another line of research uses triangle meshes, which encode surface geometry and connectivity and can be reconstructed from point clouds, ESDFs, or TSDFs [16], [54]. OVPC Mesh [28] represents the surrounding environment as a watertight 3D mesh constructed from visible point clouds, classifying traversable space based on surface normals and height variation within each mesh polygon, and uses RRT-connect [55] for local path planning. Pütz et al. [29] computed continuous shortest-path vector fields directly on triangle meshes via wavefront propagation, enabling extraction of global paths by following the field toward the goal. Compared with point clouds, triangle meshes provide a more continuous and interpretable surface representation. Nevertheless, these approaches rely on purely surface-based traversability estimation, struggle with traversable structures featuring vertical surfaces, and incur high planning times when dense meshes are required to represent complex environments.

To overcome the limitations of direct navigation on triangle meshes, NavMeshes abstract traversable surfaces into connected polygons. Recast [35], a widely adopted NavMesh generation framework, combines surface geometry from triangle meshes with voxel-based clearance and connectivity analysis to estimate traversability, producing polygons that represent traversable regions in multi-level environments and form a navigation graph supporting efficient A* [56] search. GaitMesh [34] applies NavMeshes to ground robot navigation in large-scale multi-floor environments by integrating curvature-based terrain analysis and gait controller selection. However, these NavMesh representations assume yaw-independent traversability and approximate the robot



(a) (b) (c)

Fig. 3: Robot geometry approximations and footprint masks. (a) Cylindrical approximation. (b) Cuboid approximation. (c) Binary footprint masks, where 1 denotes cells occupied by the robot footprint and 0 denotes unoccupied cells.

as a cylinder, preventing accurate representation of narrow passages with yaw-dependent traversability. Furthermore, GaitMesh relies on an offline construction pipeline and does not support online generation from incremental observations. In this work, we propose an SE(2) NavMesh that incorporates robot yaw during mesh construction, enabling better traversability capture in narrow passages and supporting planning of both heading and motion direction. An online generation pipeline constructs maps from onboard sensing, after which the resulting navigation mesh supports multiple planning queries. Table I shows that, compared with existing approaches, SE(2) NavMesh uniquely supports all listed navigation capabilities.

III. Problem Formulation

Consider a holonomic ground robot—typical of legged systems—capable of omnidirectional translation and yawing across traversable surfaces. Assume that the robot maintains a relatively fixed planar footprint and that its traversal capability parameters are given. The robot traverses a 3D environment represented as a polygonal surface mesh $\mathcal{M} \subset \mathbb{R}^3$. Its state is defined as

$$\mathbf{x} = (\mathbf{p}, \psi) \ , \tag{1}$$

where $\mathbf{p} \in \mathcal{M}$ denotes the projection of the reference point of the robot base onto the nearest polygonal surface (along the direction of gravity) and $\psi \in S^1$ denotes the yaw. Then, the configuration space of the robot is

$$\mathcal{X} = \mathcal{M} \times S^1 \ . \tag{2}$$

The robot's geometry and traversability characteristics further restrict the feasible configurations. The resulting traversable state space is defined as

$$\mathcal{X}_{\text{free}} = \{(\mathbf{p}, \psi) \in \mathcal{X} \mid \mathbf{p} \in \mathcal{P}_{\text{free}}(\psi)\} \ , \tag{3}$$

where $\mathcal{P}_{\text{free}}(\psi) \subset \mathcal{M}$ denotes the set of feasible positions for a fixed yaw. This formulation, therefore, captures the yaw-dependent traversable surfaces of the environment for the robot.

Our objective is to develop a representation that supports traversability estimation and pathfinding for this robot in a 3D environment. In this context, traversability estimation is the task of deriving the $\mathcal{X}_{\text{free}}$ from $\mathcal{M}$. Pathfinding is then reduced to the problem of computing a continuous path within $\mathcal{X}_{\text{free}}$ between a given start state $\mathbf{x}_{\text{start}} \in \mathcal{X}_{\text{free}}$ and a goal state $\mathbf{x}_{\text{goal}} \in \mathcal{X}_{\text{free}}$. Crucially, this representation should also be capable of running online and being generated directly from sensor input.

IV. Preliminaries

In this section, we review the foundational concepts of the classical NavMesh, following the formulation adopted in Recast [35].

A. Robot Model and Cylindrical Approximation

In the NavMesh, the robot is modeled using four parameters that characterize its geometry and traversal capabilities. As illustrated in Figure 3a, a cylindrical approximation is employed to represent the robot's geometry, which is parameterized by the robot height h_{robot} and the robot circumradius r_{circ}, defined as the radius of the minimum circumscribed circle of its footprint. The robot traversal capabilities are described by the maximum step height h_{step} and the maximum slope angle θ_{climb}. Given that the robot remains upright, the cylinder axis is always aligned with the direction of gravity. Under this modeling paradigm, the robot projects a circular footprint onto the ground.

B. Traversable Regions

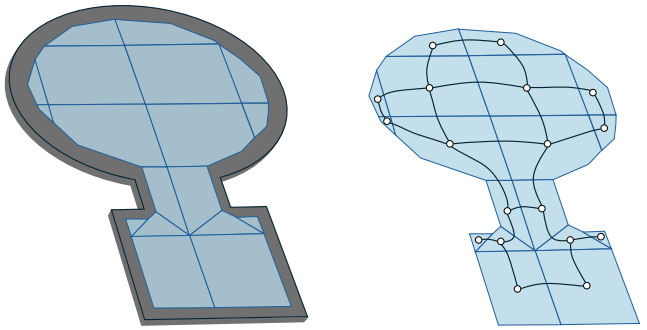
Since the NavMesh does not consider yaw during traversability estimation, it requires that a position be feasible for all possible yaws, corresponding to taking the intersection over all yaw-dependent feasible sets:

$$\bigcap_{\psi \in S^1} \mathcal{P}_{\text{free}}(\psi) \ . \tag{4}$$

This provides a conservative representation of the traversable surfaces in the environment. The NavMesh partitions traversable positions and performs polygonal contour extraction, yielding a simplified representation as a set of traversable regions (see Figure 4a):

$$\mathcal{R}_{\text{trav}} = \{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3, \ldots\}. \tag{5}$$

Each $\mathcal{R}_i \subset \mathcal{M}$ denotes a set of traversable positions represented by a convex polygonal region. Therefore, the set of



(a) Traversable regions. (b) Graph.

Fig. 4: Classical NavMesh representation and graph construction. (a) Traversable regions are represented as non-overlapping convex polygons. (b) The graph represents each polygon as a vertex, with edges encoding connectivity between adjacent polygons.

traversable positions under the NavMesh representation can be written as

$$\mathcal{P}_{\text{trav}} = \bigcup_{\mathcal{R}_i \in \mathcal{R}_{\text{trav}}} \mathcal{R}_i \, . \tag{6}$$

We can further write the estimated traversable state space as

$$\mathcal{X}_{\text{trav}} = \mathcal{P}_{\text{trav}} \times S^1. \tag{7}$$

Due to the conservative estimation of traversable regions in the classical NavMesh, the induced state space satisfies

$$\mathcal{X}_{\text{trav}} \subseteq \mathcal{X}_{\text{free}}. \tag{8}$$

Consequently, feasible configurations that are valid only for specific yaws are not captured by this representation.

C. Connectivity of Traversable Regions

Connectivity is defined as a binary relation over traversable regions: two regions $\mathcal{R}_i$ and $\mathcal{R}_j$ are connected if their boundaries have overlapping segments. This connectivity relation can be written as:

$$\text{Conn}(\mathcal{R}_i, \mathcal{R}_j) = \begin{cases} 1, & \text{if } \mathcal{H}^1(\partial\mathcal{R}_i \cap \partial\mathcal{R}_j) > 0 \\ 0, & \text{otherwise} \end{cases}, \tag{9}$$

where $\partial\mathcal{R}_i$ denotes the boundary of region $\mathcal{R}_i$ and $\mathcal{H}^1$ is the 1-dimensional Hausdorff measure. The connectivity definition implies that the robot can traverse freely between the two regions through the shared polygon edge segment.

D. Graph

Based on the defined connectivity relation, the set of traversable regions can be transformed into a graph representation that enables efficient pathfinding. Graph $G = (\mathcal{V}, \mathcal{E})$ describes how the robot can navigate between traversable regions. Each vertex $v_i \in \mathcal{V}$ corresponds to a traversable region $\mathcal{R}_i \in \mathcal{R}_{\text{trav}}$. An edge between two vertices exists if the corresponding traversable regions are connected, i.e.,

$$(v_i, v_j) \in \mathcal{E} \iff \text{Conn}(\mathcal{R}_i, \mathcal{R}_j) = 1 \tag{10}$$

Figure 4b shows an example of converting the traversable regions and their connectivity into a graph. Pathfinding is achieved via an A* search conducted on the graph. Details regarding A* pathfinding on NavMesh can be found in [57].

V. Method

Figure 2 provides an overview of the proposed SE(2) NavMesh pipeline. This section first defines the components of SE(2) NavMesh and formalizes their relations, clarifying how SE(2) NavMesh extends the classical NavMesh. It then describes the offline generation method, the associated pathfinding procedure, and the online generation method.

A. SE(2) NavMesh Definitions

1) Robot Model and Cuboid Approximation: In this work, we demonstrate the framework using a legged robot with a rectangular footprint. Importantly, the framework itself is not constrained by this choice and applies to ground robots with arbitrary shapes. We describe the geometry of the robot by its length l_{robot}, width w_{robot}, and height h_{robot}, as illustrated in Figure 3b. Compared with the cylindrical approximation, the cuboid approximation provides a more compact representation of the robot. In the same way that Recast adopts a cylindrical approximation, the height direction of the cuboid is aligned with the direction of gravity. The traversal capabilities of the robot are characterized by the maximum step height h_{step} and the maximum slope angle θ_{climb}.

2) Yaw Channel: The continuous yaw interval $[0, 2\pi)$ is uniformly discretized into N_Ψ yaw angles, forming the set

$$\Psi = \{\psi_1, \psi_2, \psi_3, \ldots, \psi_{N_\Psi}\}, \tag{11}$$

where each ψ_i corresponds to a robot heading

$$\psi_i = i \cdot \frac{2\pi}{N_\Psi} \, . \tag{12}$$

Each heading ψ_i is associated with a yaw channel indexed by i. Correspondingly, the set of yaw channels is defined as

$$\mathcal{I}_\Psi = \{1, 2, \ldots, N_\Psi\}. \tag{13}$$

For a potential position $\mathbf{p} \in \mathcal{M}$, the feasibility of the i-th yaw channel is:

$$C_i(\mathbf{p}) = \begin{cases} 1, & \text{if robot can occupy } \mathbf{p} \text{ with yaw } \psi_i \\ 0, & \text{otherwise} \end{cases} . \tag{14}$$

3) Footprint Mask: As illustrated in Figure 3c, the footprint mask is a binary matrix representing the spatial occupancy of the robot footprint on the horizontal plane (e.g., a rectangular footprint for our cuboid-shaped legged robots). Each cell of the mask has a value of 1 if the corresponding position is occupied by the footprint, and 0 otherwise. One cell within the occupied region is designated as the reference cell of the robot base, typically chosen as the center cell.

Depending on the intended use, a footprint mask can be generated either for an isolated yaw angle or for a continuous range of yaw angles. In the former case, the footprint mask, termed the isolated-yaw footprint mask, represents the planar occupancy of the robot associated with a single yaw angle.

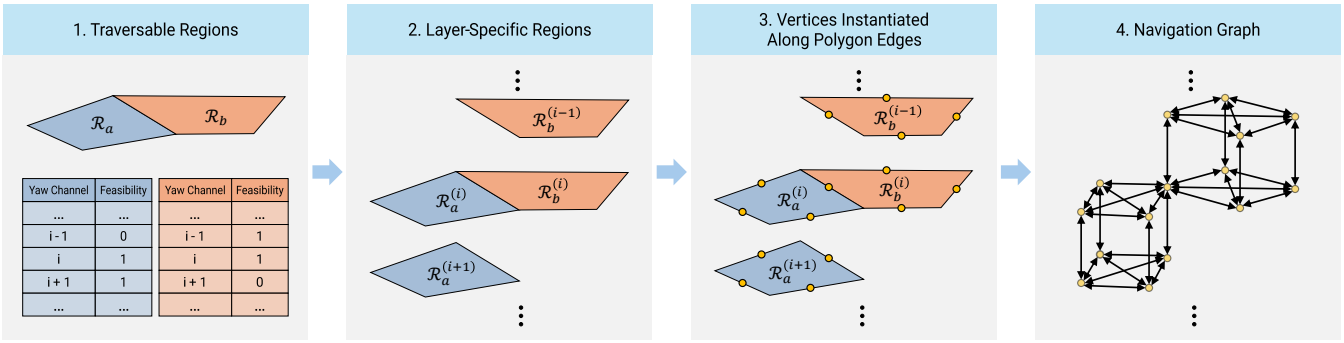


Fig. 5: Generation of layer-specific regions and the navigation graph. Each yaw channel corresponds to a yaw-specific traversability layer. According to the feasible yaw channel sets of traversable regions, layer-specific regions are generated. Graph vertices are instantiated along the polygon edges of the layer-specific regions, and connectivity is established within and across layers to construct the navigation graph.

Such a mask is suitable for collision checking at a fixed robot yaw angle during path planning. In the latter case, the footprint mask, termed the continuous-yaw footprint mask, captures the union of the occupied areas swept by the robot footprint over the yaw interval

$$\left[\psi_i - \frac{\psi_{\text{interval}}}{2},\ \psi_i + \frac{\psi_{\text{interval}}}{2}\right),\ \psi_{\text{interval}} = \frac{2\pi}{N_\Psi}\,. \tag{15}$$

If a position is feasible under the continuous-yaw footprint masks of two adjacent yaw angles, then a safe rotation between these two headings is guaranteed.

4) Safe and Restricted Traversable Regions: For each region $\mathcal{R}_j \in \mathcal{R}_{\text{trav}}$, yaw feasibility is defined to be uniform within the region, such that all positions in $\mathcal{R}_j$ share the same feasible yaw channels. We therefore define the feasible yaw channel set $\mathcal{I}(\mathcal{R}_j) \subseteq \mathcal{I}_\Psi$ as the set of channels that permit traversal within $\mathcal{R}_j$. Formally, for any position $\mathbf{p} \in \mathcal{R}_j$ and channel $i \in \mathcal{I}_\Psi$,

$$C_i(\mathbf{p}) = \begin{cases} 1, & i \in \mathcal{I}(\mathcal{R}_j) \\ 0, & i \notin \mathcal{I}(\mathcal{R}_j) \end{cases}. \tag{16}$$

Depending on the relationship between $\mathcal{I}(\mathcal{R}_j)$ and $\mathcal{I}_\Psi$, we partition the overall traversable space into two distinct subsets: safe ($\mathcal{R}_{\text{safe}}$) and restricted ($\mathcal{R}_{\text{restricted}}$) traversable regions, such that

$$\mathcal{R}_{\text{trav}} = \mathcal{R}_{\text{safe}} \cup \mathcal{R}_{\text{restricted}}\,. \tag{17}$$

A region $\mathcal{R}_j$ is classified as safe if the robot can traverse it under all discretized yaw angles,

$$\mathcal{I}(\mathcal{R}_j) = \mathcal{I}_\Psi, \tag{18}$$

and is classified as restricted if traversal is feasible only under a subset of yaw angles,

$$\emptyset \subsetneq \mathcal{I}(\mathcal{R}_j) \subsetneq \mathcal{I}_\Psi. \tag{19}$$

5) Yaw-Specific Traversability Layer: For each yaw channel $i \in \mathcal{I}_\Psi$, we define a yaw-specific traversability layer $\mathcal{L}_i$. Each layer represents the subset of traversable regions where the robot can traverse under this specific heading:

$$\mathcal{L}_i = \{\mathcal{R}_j \in \mathcal{R}_{\text{trav}} \mid i \in \mathcal{I}(\mathcal{R}_j)\}. \tag{20}$$

The relationship between the yaw-specific layers and the safe/restricted traversable regions can be expressed as

$$\mathcal{R}_{\text{trav}} = \bigcup_{i \in \mathcal{I}_\Psi} \mathcal{L}_i, \tag{21a}$$

$$\mathcal{R}_{\text{safe}} = \bigcap_{i \in \mathcal{I}_\Psi} \mathcal{L}_i, \tag{21b}$$

$$\mathcal{R}_{\text{restricted}} = \bigcup_{i \in \mathcal{I}_\Psi} \mathcal{L}_i \setminus \bigcap_{i \in \mathcal{I}_\Psi} \mathcal{L}_i. \tag{21c}$$

To distinguish the occurrences of a traversable region $\mathcal{R}_j \in \mathcal{R}_{\text{trav}}$ across different yaw-specific traversability layers, we denote by $\mathcal{R}_j^{(i)}$ its layer-specific region in layer $\mathcal{L}_i$, as illustrated in Figure 5. The element $\mathcal{R}_j^{(i)}$ exists if and only if $\mathcal{R}_j$ is feasible at yaw angle ψ_i.

6) Translational Connectivity and Rotational Connectivity: When a robot moves between a restricted traversable region and another traversable region, it must carefully choose its yaw, as the yaw feasibility of the two regions may not coincide, and it might not be able to maintain a fixed heading. Consequently, connectivity in SE(2) NavMesh extends the standard NavMesh definition by incorporating the yaw angle, categorizing it into two types: translational and rotational. Translational connectivity is defined between two layer-specific regions that lie in the same yaw-specific traversability layer, corresponding to the robot's motion while keeping the fixed yaw. Specifically, two layer-specific regions $\mathcal{R}_a^{(i)}$ and $\mathcal{R}_b^{(i)}$ from layer $\mathcal{L}_i$ are translationally connected if $\mathcal{R}_a$ and $\mathcal{R}_b$ have overlapping polygon edges:

$$\text{Conn}_{\text{trans}}(\mathcal{R}_a^{(i)}, \mathcal{R}_b^{(i)}) = \begin{cases} 1, & \text{if } \mathcal{H}^1(\partial\mathcal{R}_a \cap \partial\mathcal{R}_b) > 0 \\ 0, & \text{otherwise} \end{cases}. \tag{22}$$

Rotational connectivity is defined between different layer-specific regions of the same traversable region. It represents the robot's ability to rotate in-place within a region $\mathcal{R}$ while remaining collision-free. Specifically, for a traversable region $\mathcal{R}_k$, two of its layer-specific regions $\mathcal{R}_k^{(i)}$ and $\mathcal{R}_k^{(j)}$ are considered rotationally connected if their corresponding yaw

angles are adjacent:

$$\mathrm{Conn}_{\mathrm{rot}}(\mathcal{R}_k^{(i)}, \mathcal{R}_k^{(j)}) = \begin{cases} 1, & \text{if } j \equiv i \pm 1 \pmod{N_\Psi} \\ 0, & \text{otherwise} \end{cases} \tag{23}$$

Due to the periodicity of yaw, adjacency is interpreted cyclically, hence the use of the modulo operation.

7) Navigation Graph: For pathfinding purposes, we construct a navigation graph $G = (\mathcal{V}, \mathcal{E})$ based on layer-specific regions of traversable regions, as illustrated in Figure 5. For each polygon edge of every traversable region, a representative geometric point, such as its midpoint, is chosen and used to instantiate a vertex in each corresponding layer-specific region. Vertices within the same layer share the same yaw attribute and are connected by translational edges $\mathcal{E}_{\mathrm{trans}}$. Vertices at the same spatial location in adjacent yaw-specific layers are connected by rotational edges $\mathcal{E}_{\mathrm{rot}}$, representing changes in the robot's heading. We define the edge cost as the time required for the robot to execute the corresponding motion.

B. Offline Generation of SE(2) NavMesh

The offline pipeline generates an SE(2) NavMesh from the input triangle mesh. Two sets of parameters are needed for the generation process:

- *Robot Parameters*: length (l_{robot}), width (w_{robot}), height (h_{robot}), maximum step height (h_{step}), and maximum slope angle (θ_{climb}).
- *Map Configuration Settings*: the map voxelization resolution for length, width, and height ($s_{\mathrm{voxel}} \times s_{\mathrm{voxel}} \times h_{\mathrm{voxel}}$), and the number of yaw layers (N_Ψ).

The generation process first constructs footprint masks for discretized yaw angles. The environment is partitioned into tiles and converted into voxels that represent the local terrain. Yaw feasibility is then evaluated on these terrain voxels to determine traversable states under different headings. Finally, traversable voxels are aggregated to generate convex polygonal regions, which are merged across tiles to form the global SE(2) NavMesh together with its navigation graph.

1) Footprint Masks Generation: We use continuous-yaw footprint masks to ensure safe rotation. Each footprint mask, as shown in Figure 3c, is defined on a grid with a cell size of $s_{\mathrm{voxel}} \times s_{\mathrm{voxel}}$, matching the resolution used for map voxelization.

2) Tile Partitioning: The TileMesh approach from Recast [35] is adopted to generate SE(2) NavMesh. As illustrated in Figure 6, the map is partitioned into square tiles on the horizontal plane, with each tile spanning $d_{\mathrm{tile}} \times d_{\mathrm{tile}}$ voxels along the horizontal dimensions and being responsible for generating the SE(2) NavMesh for all height levels within its horizontal extent. This partitioning enables parallel processing, as each tile can be generated independently using only a local subset of the map geometry as input. This subset covers a square area of $(d_{\mathrm{tile}} + 2d_{\mathrm{border}})^2$ voxels, where d_{border} equals the circumradius of the robot footprint in voxels plus an additional safety margin of several voxels. This margin is required to correctly evaluate traversability near the tile border. All subsequent steps of the offline SE(2) NavMesh generation pipeline are carried out on a per-tile basis.

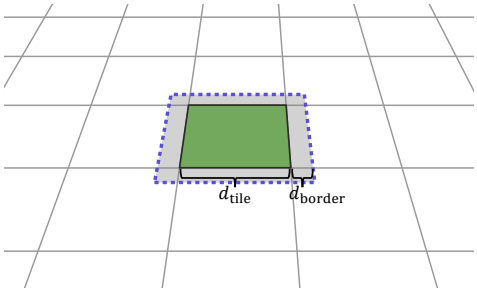


Fig. 6: Tile-based SE(2) NavMesh generation. Generating the SE(2) NavMesh for a target tile (green) requires geometry from the tile itself and its neighboring area, shown as the gray region enclosed by the purple dashed line.

3) Map Voxelization and Walkable Voxel Annotation: The input triangle mesh is voxelized. Vertically connected voxels at the same planar position are merged, and we refer to them as *solid spans*. The empty intervals between solid spans form the *free space spans*. Based on surface geometry, robot height, and step constraints, each free space span is classified as walkable or non-walkable. The bottommost voxels of free space spans represent the terrain surfaces and collectively form a *tile terrain voxel map*, which serves as the basis for the subsequent steps.

4) Yaw Feasibility Evaluation: On the tile terrain voxel map, a distance map is computed to represent the distance from each voxel to the nearest locally invalid voxel. A voxel is treated as locally invalid if it is non-walkable, adjacent to non-walkable voxels or unobserved regions, or if the height difference to any of its four neighboring voxels exceeds the traversability constraints. Locally invalid voxels are assigned a distance value of 0. Based on the distance map, voxels are further classified into three categories analogously to the traversable regions: safe, restricted, and inaccessible. A safe voxel can accommodate the robot footprint for any yaw, while a restricted voxel only supports the robot under a subset of yaw angles. An inaccessible voxel is too close to locally invalid voxels for the robot to reach.

The complete voxel classification procedure is summarized in Algorithm 1. Under the rectangular footprint approximation, the footprint inradius r_{in} and circumradius r_{circ} are given by:

$$r_{\mathrm{in}} = \frac{\min(l_{\mathrm{robot}}, w_{\mathrm{robot}})}{2}, \tag{24}$$

$$r_{\mathrm{circ}} = \frac{\sqrt{l_{\mathrm{robot}}^2 + w_{\mathrm{robot}}^2}}{2}. \tag{25}$$

Based on the distance map, voxels whose distance to locally invalid voxels is smaller than the footprint inradius are directly classified as inaccessible, while voxels whose distance exceeds or equals the footprint circumradius are classified as safe. However, distance information alone is insufficient for voxels in the intermediate range between the inradius and circumradius. These voxels therefore require additional yaw feasibility evaluation using the precomputed continuous-yaw footprint masks. For a given yaw channel, feasibility at the voxel level is determined by whether all occupied cells in the

Algorithm 1 Voxel Classification

Input: Tile terrain voxel map $\mathcal{W}_{\text{tile}}$, distance map D, yaw channel indices set $\mathcal{I}_\Psi$, footprint masks $\{M_i\}_{i\in\mathcal{I}_\Psi}$, footprint inradius r_{in}, footprint circumradius r_{circ}

Output: Feasible yaw channel set $\mathcal{I}(v)$ and voxel classification $\text{Class}(v)$ for all $v \in \mathcal{W}_{\text{tile}}$

for each voxel $v \in \mathcal{W}_{\text{tile}}$ **do**
  $\mathcal{I}(v) \leftarrow \emptyset$ ▷ Initialize feasible yaw set
  $d \leftarrow D(v)$
  if $d < r_{\text{in}}$ **then**
    $\text{Class}(v) \leftarrow$ INACCESSIBLE
  else if $d \geq r_{\text{circ}}$ **then**
    $\text{Class}(v) \leftarrow$ SAFE
    $\mathcal{I}(v) \leftarrow \mathcal{I}_\Psi$
  else
    for each yaw index $i \in \mathcal{I}_\Psi$ **do**
      ALIGNREFERENCECELL(M_i, v)
      if MASKFEASIBLE$(M_i, v, \mathcal{W}_{\text{tile}})$ **then**
        $\mathcal{I}(v) \leftarrow \mathcal{I}(v) \cup \{i\}$
      end
    end for
    if $\mathcal{I}(v) = \emptyset$ **then**
      $\text{Class}(v) \leftarrow$ INACCESSIBLE
    else
      $\text{Class}(v) \leftarrow$ RESTRICTED
    end
  end
end for

footprint mask correspond to walkable voxels. To perform this check, the footprint mask is first aligned with the candidate voxel by translating the mask so that the designated reference cell coincides with the voxel. Starting from this reference cell, a breadth-first search is performed over all occupied cells in the mask. The feasibility check succeeds if and only if all visited cells correspond to walkable voxels in the terrain voxel map.

5) Region Polygon Generation: Based on the distance field, a watershed algorithm [58] is applied to partition safe voxels and restricted voxels into non-overlapping regions. During the partitioning process, only voxels with identical feasible yaw channel sets are allowed to be grouped into the same region. The resulting regions are generally irregular in shape and are triangulated via the ear clipping algorithm [59] and subsequently merged into convex polygonal regions following the standard pipeline in [35]. This ensures straight-line reachability within each region, facilitating efficient path planning and path smoothing. These convex polygonal regions are categorized into safe and restricted traversable regions according to their feasible yaw channel sets, as defined in Section V-A4. The regions constructed within each tile are merged into a global structure, where translational and rotational connectivity are established according to section V-A6.

C. Pathfinding

We propose a three-stage pathfinding strategy for the SE(2) NavMesh, termed ASA (A*-String Pulling-A*), as illustrated in Figure 7. The strategy is structured as follows: 1) Initial Search: An A* algorithm computes an initial path and its corresponding layer-specific region sequence. 2) Path Straightening: A string-pulling method geometrically refines the path to produce straightened waypoints. 3) Yaw Refinement: A second A* search optimizes the robot's heading over the refined positions, constrained by the compact region sequence. The specific formulations for each stage are detailed below.

1) Initial Search: Given a start state $\mathbf{x}_{\text{start}} \in \mathcal{X}_{\text{free}}$ and a goal state $\mathbf{x}_{\text{goal}} \in \mathcal{X}_{\text{free}}$, we first identify the traversable regions $\mathcal{R}_{\text{start}}$ and $\mathcal{R}_{\text{goal}}$ that contain the spatial positions $\mathbf{p}_{\text{start}}$ and $\mathbf{p}_{\text{goal}}$. The yaw ψ_{start} and ψ_{goal} are then used to determine the corresponding yaw-specific layer indices l_{start} and l_{goal}, locating the layer-specific regions $\mathcal{R}_{\text{start}}^{(l_{\text{start}})}$ and $\mathcal{R}_{\text{goal}}^{(l_{\text{goal}})}$. The navigation graph is constructed using vertices predefined at the midpoints of polygon edges for each layer-specific region. At the start and goal positions, temporary query vertices are instantiated at the same spatial location for all corresponding layer-specific regions. Vertices are connected according to the graph construction defined in Section V-A7.

During the A* search, the cost function is defined according to the type of edge connecting two states $\mathbf{x}_a = (\mathbf{p}_a, \psi_a)$ and $\mathbf{x}_b = (\mathbf{p}_b, \psi_b)$. We define the cost as the estimated traversal time between two states: The translational cost is determined by the robot's longitudinal and lateral velocities, while the rotational cost accounts for the robot's yaw rate. A translational edge at the i-th layer corresponds to a movement from $(\mathbf{p}_a, \psi_i)$ to $(\mathbf{p}_b, \psi_i)$ while maintaining the current yaw ψ_i. Let $\Delta\mathbf{p}_{xy}$ denote the horizontal projection of the displacement between the two states, obtained from the (x, y) components of $\mathbf{p}_b - \mathbf{p}_a$. The robot's local longitudinal and lateral directions are represented by the unit vectors:

$$\mathbf{n}_{\text{long}} = \begin{bmatrix}\cos\psi_i \\ \sin\psi_i\end{bmatrix}, \quad \mathbf{n}_{\text{lat}} = \begin{bmatrix}-\sin\psi_i \\ \cos\psi_i\end{bmatrix}. \tag{26}$$

By projecting the displacement vector onto these local axes, the translational cost is defined as:

$$\text{Cost}(\mathbf{x}_a, \mathbf{x}_b) = \frac{|\Delta\mathbf{p}_{xy} \cdot \mathbf{n}_{\text{long}}|}{v_{\text{long}}} + \frac{|\Delta\mathbf{p}_{xy} \cdot \mathbf{n}_{\text{lat}}|}{v_{\text{lat}}}, \tag{27}$$

where v_{long} and v_{lat} denote the robot's longitudinal and lateral velocities, respectively. If a rotational edge exists between $\mathbf{x}_a$ and $\mathbf{x}_b$, the motion corresponds to an in-place rotation ($\mathbf{p}_a = \mathbf{p}_b$) of angle ψ_{interval} (defined in Equation (15)) between two adjacent layers. The corresponding cost is defined as:

$$\text{Cost}(\mathbf{x}_a, \mathbf{x}_b) = \frac{\psi_{\text{interval}}}{\omega} = \frac{2\pi}{N_\Psi \omega}, \tag{28}$$

where ω denotes the robot's yaw rate.

The heuristic function used in the A* search is defined as:

$$\text{Heur}(\mathbf{x}_j) = \frac{\|\mathbf{p}_{\text{goal}} - \mathbf{p}_j\|_2}{\max\{v_{\text{long}}, v_{\text{lat}}\}} + \frac{\min\{|\psi_{\text{goal}} - \psi_j|, 2\pi - |\psi_{\text{goal}} - \psi_j|\}}{\omega}, \tag{29}$$

which estimates the remaining time to reach the goal by combining a lower bound on the translational travel time and a lower bound on the rotational adjustment time.

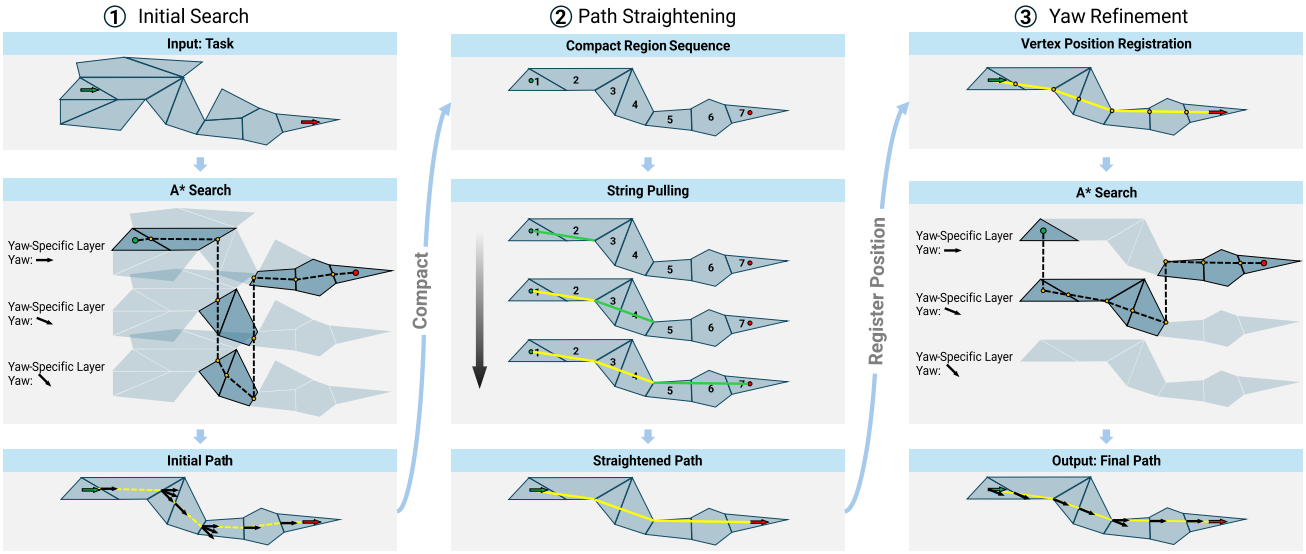


Fig. 7: Overview of the ASA pathfinding strategy. The method consists of three stages: (1) an initial A* search that obtains a feasible path and layer-specific region sequence, (2) string pulling that straightens the path, and (3) yaw refinement through a second A* search that optimizes robot headings while respecting the straightened path constraints.

The A* search returns a sequence of states

$$\tau_{\text{init}} = (\mathbf{x}_{\text{start}}, \mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{\text{goal}}), \quad \mathbf{x}_k = (\mathbf{p}_k, \psi_k^\tau), \quad (30)$$

where each state explicitly encodes both the spatial position and the yaw of the robot. This state sequence spans multiple yaw-specific layers and provides a feasible connection between the start and goal states.

Each state $\mathbf{x}_i$ is associated with a layer-specific region $\mathcal{R}_{k_i}^{(l_i)}$. Mapping the states in τ to their corresponding layer-specific regions yields an ordered sequence

$$\Pi_{\text{init}} = \left(\mathcal{R}_{\text{start}}^{(l_{\text{start}})}, \mathcal{R}_{k_1}^{(l_1)}, \mathcal{R}_{k_2}^{(l_2)}, \ldots, \mathcal{R}_{\text{goal}}^{(l_{\text{goal}})}\right), \quad (31)$$

which describes the layer-specific regions traversed.

2) Path Straightening: The sequence of states τ_{init} is generally not length-optimal in the Euclidean sense, since it is constrained to pass through the midpoints of the polygon edges. To reduce the resulting zigzag behavior, we perform geometric path refinement using string pulling. The sequence Π_{init} is first projected onto the original traversable regions by discarding yaw-specific layer information. Consecutive occurrences of the same regions are then merged to obtain a compact region sequence. For example, the sequence

$$\left(\mathcal{R}_a^{(2)}, \mathcal{R}_b^{(2)}, \mathcal{R}_b^{(3)}, \mathcal{R}_b^{(4)}, \mathcal{R}_c^{(4)}\right) \quad (32)$$

is compacted to

$$\Pi_{\text{compact}} = (\mathcal{R}_a, \mathcal{R}_b, \mathcal{R}_c). \quad (33)$$

The compact region sequence Π_{compact} defines a continuous polygonal corridor. Within this corridor, string pulling is applied to compute a shortest path in the Euclidean sense, connecting the start and goal. We denote the resulting straightened path as

$$\gamma = (\mathbf{p}_{\text{start}}, \mathbf{p}_{\gamma,1}, \mathbf{p}_{\gamma,2}, \ldots, \mathbf{p}_{\text{goal}}), \quad (34)$$

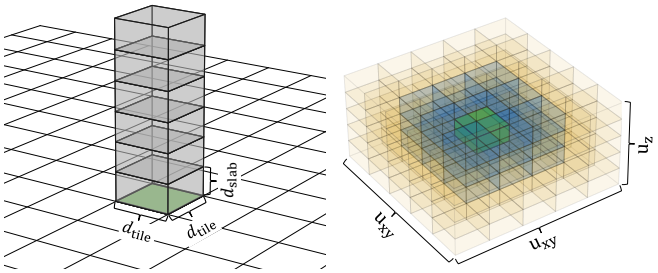
where $\mathbf{p}_{\text{start}}$ and $\mathbf{p}_{\text{goal}}$ denote the start and goal positions, respectively, and the intermediate positions are given by the intersection of the path with the polygon edges of the traversable regions forming the corridor. Although string pulling is performed purely in $\mathbb{R}^3$ without explicit yaw information, the resulting path remains feasible since it stays within the corridor Π_{compact}. As the corridor is obtained from an A* search over yaw-specific layers, feasible yaw transitions between consecutive traversed regions are already guaranteed. For details of the string pulling procedure, we refer to [60].

3) Yaw Refinement: The path γ alters the direction of motion along each path segment, which necessitates replanning the robot's yaw along the path. To this end, we perform yaw refinement constrained to γ. For each position along γ, we identify the traversable regions containing it and retrieve the corresponding feasible yaw channel set. Vertices are then registered for all feasible yaw angles at these positions, and a new navigation graph G is constructed following the procedure described in Section V-A7. An A* search on this graph finds an optimal state sequence that is consistent with both the refined straightened path and the yaw-dependent traversability constraints. The cost and heuristic functions used in this stage are identical to those defined in Section V-C1.

D. Online Generation of SE(2) NavMesh

The online method incrementally generates and updates the SE(2) NavMesh using point cloud measurements of the surrounding environment acquired by onboard sensors. The following description focuses on the components specific to the online method, since most processing steps reuse the procedures introduced in Section V-B.

1) Map Update: We employ Voxblox [16] to process the input point clouds and incrementally construct a triangle mesh representation of the environment. Voxblox maintains spatial information in a block-based structure, where space



(a) Slabs. (b) Update window.

Fig. 8: Slab-based online SE(2) NavMesh update. (a) A tile is divided along the vertical axis into equal-height slabs, shown as gray cuboids. (b) During an update, the central green slab contains the changed map geometry. The green and blue slabs require local SE(2) NavMesh updates, while geometry from the green, blue, and yellow slabs is used for the updates.

is discretized into cubic Voxblox voxels with an edge length of s_{map}. To avoid ambiguity, we use the term *Voxblox voxel* for the Voxblox map representation, while *voxel* refers to the discretization used in the SE(2) NavMesh unless otherwise specified. Voxblox voxels are grouped into cubic blocks of size $d_{\text{block}} \times d_{\text{block}} \times d_{\text{block}}$, where d_{block} denotes the number of Voxblox voxels along each dimension. The triangle mesh in these blocks is generated from the TSDF values using the marching cubes algorithm [61].

2) SE(2) NavMesh Update: Whenever the map is updated, the SE(2) NavMesh must be updated accordingly. Reconstructing the SE(2) NavMesh over the entire map after every update is inefficient, as the map is expanding and map updates are typically spatially localized. We therefore perform local SE(2) NavMesh updates restricted to affected regions of the map. To support efficient local updates, each tile introduced in Section V-B2 is further divided along the vertical axis into multiple slabs, as illustrated in Figure 8a. Each slab spans $d_{\text{tile}} \times d_{\text{tile}} \times d_{\text{slab}}$ voxels, where d_{slab} denotes the slab height in voxels. Slab boundaries are aligned with the Voxblox block grid, and each slab dimension is chosen as an integer multiple of the corresponding block dimension to enable efficient indexing and geometry retrieval. When the map geometry within a slab changes, all slabs within a surrounding 3D update window centered at the modified slab are affected and must be updated. The update window spans $u_{\text{xy}} \times u_{\text{xy}} \times u_{\text{z}}$ slabs, where u_{xy} and u_{z} denote the number of covered slabs in the planar and vertical direction, respectively:

$$u_{\text{xy}} = \left(1 + 2\left\lceil \frac{d_{\text{border}}}{d_{\text{tile}}} \right\rceil\right), \tag{35}$$

$$u_{\text{z}} = \left(1 + 2\left\lceil \frac{d_{\text{zborder}}}{d_{\text{slab}}} \right\rceil\right). \tag{36}$$

Here, d_{border} is the planar margin in voxels defined in Section V-B2, while d_{zborder} is the vertical margin. The latter is chosen as the robot height in voxels plus an additional safety margin of several voxels. For each affected slab, surrounding geometry is retrieved from neighboring slabs within the same spatial window. The local SE(2) NavMesh for each affected slab is then reconstructed by reusing the corresponding steps described in the offline method (Sections V-B3 to V-B5). The updated local structures are subsequently merged into the global SE(2) NavMesh, where connectivity is re-established with vertically adjacent slabs in the same tile and horizontally adjacent slabs at the same height level in neighboring tiles. Figure 8b illustrates the spatial relationship among the slab containing the updated map geometry, the affected slabs requiring local SE(2) NavMesh updates, and the surrounding geometry access region. The illustration assumes that the required planar and vertical margins do not exceed the dimensions of a single slab.

TABLE II: ANYmal Parameters

Category	Parameters	Values
Dimensions	$l_{\text{robot}}, w_{\text{robot}}, h_{\text{robot}}$	$0.93\,\text{m}, 0.53\,\text{m}, 0.89\,\text{m}$
Traversal Cap.	$h_{\text{step}}, \theta_{\text{climb}}$	$0.25\,\text{m}, 30°$
Speed	$v_{\text{long}}, v_{\text{lat}}, \omega$	$0.5\,\text{m/s}, 0.1\,\text{m/s}, 0.5\,\text{rad/s}$

VI. Experiments

We conduct extensive experiments to assess the proposed SE(2) NavMesh in terms of representation quality, planning performance, and real-world deployability. First, across multiple simulated environments, we compare SE(2) NavMesh with the classical NavMesh in traversable region coverage. Second, using the generated SE(2) NavMesh, we evaluate the proposed ASA pathfinding pipeline, quantify the contribution of its individual stages, and benchmark it against representative sampling-based planners using success rate, path cost, and planning time. Finally, we deploy the online SE(2) NavMesh system on a real robot to verify real-time online updates and demonstrate autonomous navigation in real-world environments.

A. Comparison of NavMesh and SE(2) NavMesh

We compare the generation of the NavMesh and the proposed offline SE(2) NavMesh in terms of the traversable area and the generation time. Simulation experiments are conducted in six diverse indoor environments selected from HM3D [26], as shown in Figure 9. For each scene, both meshes are constructed using identical robot parameters and mesh generation parameters. The simulated robot is modeled after the ANYmal [62], a quadruped robot capable of locomotion over stairs and sloped terrain. The robot parameters used in the simulation are listed in Table II. Dimensions and traversal capabilities follow [63], while the locomotion speed is conservatively chosen for safe navigation. The parameters for mesh generation are provided in Table III. For the NavMesh, the robot circumradius r_{circ} is derived from the robot length l_{robot} and width w_{robot}. All simulation experiments are conducted on an Intel i7-11800H @ 2.30 GHz CPU.

The selected scenes cover diverse layouts and navigation challenges, including multi-level structures, narrow passages, and doorways.

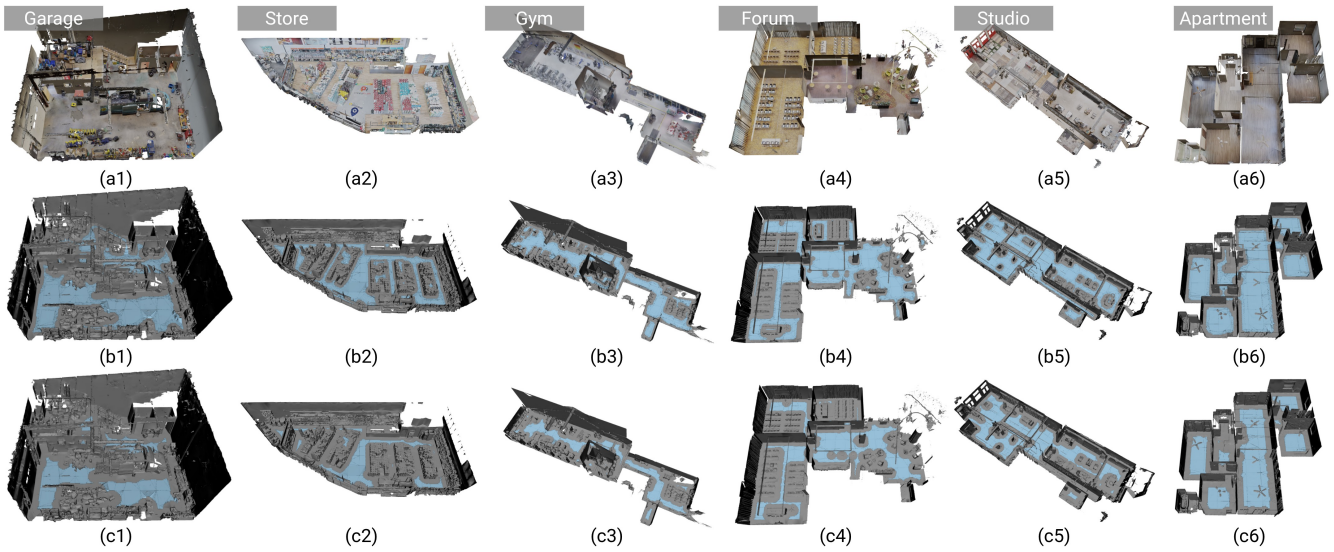


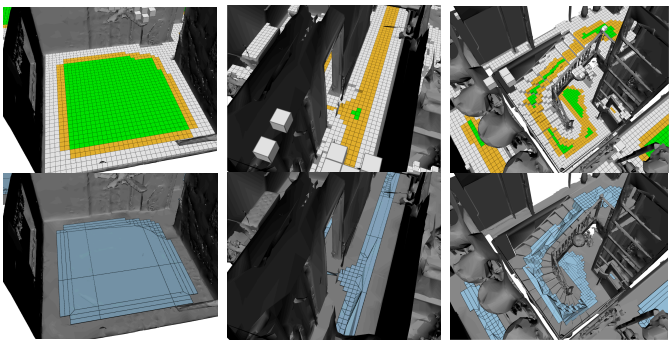
Fig. 9: Offline mesh generation results across six simulation scenes. The first row shows the 3D scene models: (a1) *Garage*, (a2) *Store*, (a3) *Gym*, (a4) *Forum*, (a5) *Studio*, and (a6) *Apartment*. The second row (b1-b6) shows traversable regions generated by SE(2) NavMesh, and the third row (c1-c6) shows those generated by the NavMesh. Traversable regions are colored blue.

TABLE III: Mesh Generation Parameters

Category	Parameters	Values
Voxel	$s_{\text{voxel}} \times s_{\text{voxel}} \times h_{\text{voxel}}$	$0.1\,\text{m} \times 0.1\,\text{m} \times 0.1\,\text{m}$
Tile	$d_{\text{tile}} \times d_{\text{tile}}$	16×16
Yaw	N_Ψ	40

- *Garage*: A two-level environment with a loft. The ground floor contains many objects and is connected to the loft by a narrow staircase. Its dimensions are $15.7\,\text{m} \times 12.5\,\text{m} \times 8.2\,\text{m}$.
- *Store*: A single-level environment containing many shelves. Narrow aisles are formed between the shelves. The overall scene dimensions are $30.3\,\text{m} \times 13.0\,\text{m} \times 5.1\,\text{m}$.
- *Gym*: A two-level environment in which the upper level contains fitness equipment. The two levels are connected via a staircase. Its dimensions are $40.4\,\text{m} \times 11.7\,\text{m} \times 11.3\,\text{m}$.
- *Forum*: A single-level environment consisting of several meeting rooms. Each meeting room is connected to the main hall through a door. Its dimensions are $31.4\,\text{m} \times 30.0\,\text{m} \times 4.7\,\text{m}$.
- *Studio*: A single-level environment with sparse furnishings. The main area and a secondary area are connected, with a narrow corridor leading to a restroom. Its dimensions are $39.4\,\text{m} \times 12.2\,\text{m} \times 4.2\,\text{m}$.
- *Apartment*: A single-level environment representing an empty apartment composed of multiple rooms. Its dimensions are $16.6\,\text{m} \times 15.1\,\text{m} \times 3.0\,\text{m}$.

Figure 9 compares the traversable regions generated by SE(2) NavMesh and the NavMesh. By accounting for yaw-dependent traversability, the proposed method preserves narrow passages and near-boundary regions that are discarded by the NavMesh. Figure 10 provides close-up views of voxel classification and the resulting traversable region polygon



(a) Empty room. (b) Narrow corridor. (c) Staircase.

Fig. 10: Close-up views of voxel classification (top row) and generated SE(2) NavMesh polygons (bottom row) of an empty room in *Apartment*, a narrow corridor in *Studio*, and a staircase in *Gym*. Safe voxels are shown in green, and restricted voxels are shown in yellow.

generated by SE(2) NavMesh. In open areas (Figure 10a), safe voxels can be merged into large convex polygons (safe traversable regions). In contrast, in constrained environments such as corridors (Figure 10b) and staircases (Figure 10c), the feasible yaw channel set varies frequently across neighboring voxels, preventing merging and resulting in a greater number of smaller polygons that encode local yaw-dependent traversability. Together, this property naturally yields an adaptive spatial discretization, allowing efficient traversal through safe regions while maintaining fine planning resolution in restricted regions.

A comparison of the traversable regions generated by the NavMesh and the proposed SE(2) NavMesh is shown in Figure 11. For each scene, we evaluate both the total area of the traversable regions and the area of the largest connected traversable component. Owing to the inclusion of restricted

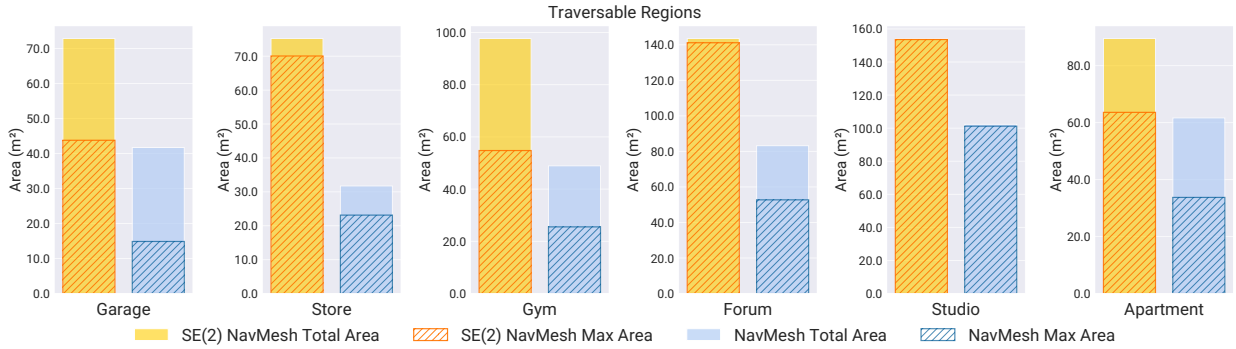


Fig. 11: Comparison of traversable regions generated by the NavMesh and SE(2) NavMesh across simulation scenes.

TABLE IV: Build Time and Region-Polygon Count for the NavMesh and SE(2) NavMesh across simulation scenes

Method	Mask Gen. (ms)	Mesh Const. (ms)	Total (ms)	# Region Polygons
Garage				
NavMesh	-	37.80	37.80	278
SE(2) NavMesh	2.44	86.25	88.69	2859
Store				
NavMesh	-	48.15	48.15	374
SE(2) NavMesh	2.48	91.96	94.44	4007
Gym				
NavMesh	-	43.53	43.53	478
SE(2) NavMesh	2.37	89.15	91.52	4362
Forum				
NavMesh	-	42.02	42.02	628
SE(2) NavMesh	2.16	107.25	109.41	5055
Studio				
NavMesh	-	42.87	42.87	603
SE(2) NavMesh	2.37	102.17	104.54	4521
Apartment				
NavMesh	-	25.83	25.83	193
SE(2) NavMesh	2.28	60.47	62.75	1729

traversable regions and a more accurate evaluation of yaw-dependent traversability, SE(2) NavMesh consistently outperforms the NavMesh across all scenes, capturing over 50 % more traversable area in every evaluated scene. Furthermore, restricted traversable regions bridge previously disconnected safe traversable regions, leading to a substantial increase in the size of the largest connected traversable component.

The time consumed by different stages of the two generation pipelines across all simulation scenes is summarized in Table IV. For SE(2) NavMesh, the total build time consists of continuous-yaw footprint mask generation and mesh construction. The mask generation stage introduces only a small overhead, averaging approximately 2.35 ms. Despite producing substantially more region polygons than the NavMesh (approximately 8 to 10 times more), SE(2) NavMesh requires only 2 to 3 times longer total build time. This indicates that the proposed method greatly enriches the representation of traversable regions at a modest additional computational cost.

B. Pathfinding Benchmark

We evaluate pathfinding performance on the SE(2) NavMesh using the proposed ASA strategy and compare it against sampling-based planners, including RRT [41], RRT* [42], and PRM [64]. All sampling-based planners are implemented using OMPL [65]. The robot parameters and mesh generation parameters are identical to those used in the Section VI-A. We first describe the configuration of the sampling-based planners, including the state space and the state validity check methods. We then consider two benchmark settings: time-dependent planning performance and randomly sampled start-goal pairs. Finally, we analyze path quality comparisons across its individual stages.

Algorithm 2 Voxel-Based State Validity Checker

Input: State $\mathbf{s}$, global terrain voxel map $\mathcal{W}_{\text{global}}$, height tolerance ε_z
Output: Valid $\in \{\text{true}, \text{false}\}$

$v \leftarrow \text{FINDNEARESTVOXELINCOLUMN}(\mathcal{W}_{\text{global}}, \mathbf{s})$
if $v = \text{null}$ **then**
  return false
end
$z \leftarrow \text{GETVOXELHEIGHT}(v)$
if $|z_s - z| > \varepsilon_z$ **then**
  return false
end
$M \leftarrow \text{GENERATEFOOTPRINTMASKFROMYAW}(\psi_s)$
$\text{ALIGNREFERENCECELL}(M, v)$
if not $\text{MASKFEASIBLE}(M, v, \mathcal{W}_{\text{global}})$ **then**
  return false
end
return true

1) Sampling-Based Planner Configuration: The sampling-based planners sample states $\mathbf{s} = (\mathbf{p}_s, \psi_s) \in \mathbb{R}^3 \times S^1$, where $\mathbf{p}_s = (x_s, y_s, z_s)$. Each sampled state is evaluated by a state validity checker, which verifies both individual states and interpolated states along candidate edges. We implement two state validity checkers. The first checker uses a global terrain voxel generated following the procedure described in Section V-B3, where each voxel is annotated as walkable or non-walkable. The sampled position is projected onto the nearest terrain voxel along the corresponding vertical column

Algorithm 3 SE(2) NavMesh-Based State Validity Checker

Input: State $\mathbf{s}$, SE(2) NavMesh $\mathcal{M}_{\text{nav}}$, height tolerance ε_z
Output: Valid $\in \{\text{true}, \text{false}\}$

$\mathcal{R} \leftarrow \text{FINDNEARESTREGION}(\mathcal{M}_{\text{nav}}, \mathbf{s})$
if $\mathcal{R} = \text{null}$ **then**
  return false
end
$z \leftarrow \text{GETREFERENCEHEIGHT}(\mathcal{R}, \mathbf{s})$
if $|z_s - z| > \varepsilon_z$ **then**
  return false
end
if not $\text{ISYAWFEASIBLE}(\mathcal{R}, \psi_s)$ **then**
  return false
end
return true

and is accepted only if the vertical deviation is below a predefined threshold. If the voxel is walkable, an isolated-yaw footprint mask corresponding to ψ_s is further evaluated, and the state is valid when all covered voxels are walkable. The second checker uses the generated SE(2) NavMesh: the sampled position must lie within a traversable region and satisfy the same vertical threshold, and the state is valid only if its yaw ψ_s lies within that region's feasible yaw range. The workflows of the two checkers are presented in Algorithms 2 and 3. Both checkers are integrated with RRT, RRT*, and PRM. Planners using the voxel-based checker retain their original names, while those using the SE(2) NavMesh-based checker are referred to as RRT-SE2NM, RRT*-SE2NM, and PRM-SE2NM, respectively.

Interpolated states partition the motion between two connected states into multiple segments. Within each segment, the robot is assumed to traverse the path with the yaw of the starting state and rotate only at the segment end. The total motion cost is obtained by summing the segment costs computed using the formulation in Section V-C.

2) Time-Dependent Planning Performance: We design five pathfinding tasks to evaluate the time-dependent performance of ASA and the sampling-based planners. For each task, the start and goal states are selected within the same connected traversable component to ensure that a feasible path exists. This allows the comparison to focus on planning efficiency and solution quality rather than reachability. For RRT and RRT-SE2NM, the maximum planning time is fixed at $100\,\text{s}$. RRT*, PRM, RRT*-SE2NM, and PRM-SE2NM are asymptotically optimal planners whose solution quality improves as planning time increases. Therefore, these planners are evaluated under planning time limits of $1\,\text{s}$, $10\,\text{s}$, and $100\,\text{s}$ to analyze their performance over time. For each task, every method is executed for 100 trials.

We evaluate pathfinding performance using two metrics: the success rate (SR) and the success weighted by inverse path cost (SPC). We define SPC as a success-weighted normalized efficiency metric, inspired by the success weighted by inverse path length (SPL) [66] and the success weighted by completion time (SCT) [67], given by

$$\text{SPC} = \frac{1}{N} \sum_{i=1}^{N} S_i \frac{c_i^*}{\max(c_i, c_i^*)} \,. \tag{37}$$

Here, N denotes the total number of pathfinding problems. S_i is a binary indicator of success for the evaluated method on the i-th problem. c_i denotes the path cost produced by that method, and c_i^* is the minimum path cost achieved for the problem, which in this experiment is taken as the minimum path cost obtained by all methods across all trials. This metric jointly accounts for both SR and solution quality in terms of path cost.

Figure 12 depicts Tasks 1 to 4, in which the start and goal are separated by challenging structures such as stairs (Task 1 and Task 3), narrow passages (Task 2), and narrow doorways (Task 3 and Task 4). These features make the tasks representative of geometrically constrained pathfinding scenarios. The pathfinding performance of all methods in Tasks 1 to 4 is presented in Figure 13. In addition, Task 5 (shown in Figure 14a) represents a contrast case in which the environment between the start and goal is relatively open and contains no narrow regions.

Across Tasks 1 to 4, ASA consistently requires the shortest planning time while achieving the highest SPC. This demonstrates that the proposed strategy, together with the generated graph, enables rapid search in constrained environments and obtains low-cost paths. Among the sampling-based planners, PRM-SE2NM generally achieves the highest SPC. However, it requires substantially more planning time to construct a sufficiently dense roadmap and optimize the path before reaching comparable solution quality. In Task 5, where the environment is relatively open, all methods achieve a SR of $100\,\%$ within the given time limits, as shown in Figure 14b. In this setting, RRT-SE2NM achieves slightly shorter planning times than ASA. This is because the sampling-based exploration of RRT-SE2NM can rapidly connect the start and goal in open space without the need to search over the mesh structure. However, ASA produces lower-cost paths, as its graph-based search enables more cost-efficient pathfinding.

Across the sampling-based planner experiments, the SE(2) NavMesh-based state validity checker achieves an average per-state checking time on the order of $0.1\,\mu\text{s}$, which is significantly faster than the voxel-based state validity checker, whose average per-state checking time is on the order of $10\,\mu\text{s}$. This computational advantage allows PRM-SE2NM to sample and validate substantially more states, resulting in noticeably higher SR and SPC compared with PRM. However, faster state validity checking does not necessarily translate into higher SR for all planners. The continuous-yaw footprint mask used in SE(2) NavMesh covers a slightly larger area than the isolated-yaw footprint mask. Consequently, this more conservative design filters out some states that lie too close to obstacles for safety. In particularly narrow regions, such as the staircase entrance in Task 1, this effect makes it more difficult for RRT-SE2NM and RRT*-SE2NM to sample valid states and expand the search tree, leading to slower planning performance than their voxel-based counterparts.

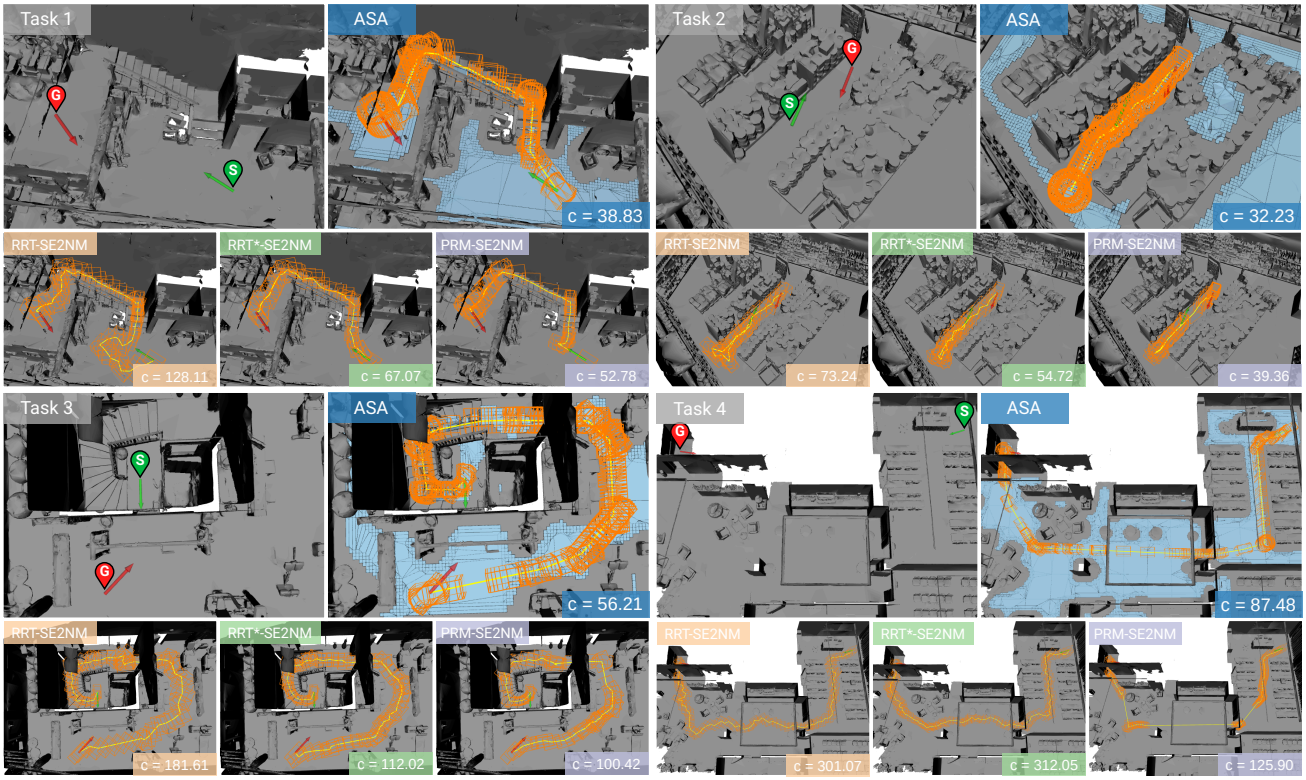


Fig. 12: Pathfinding tasks and example solutions in constrained environments. For each task, the left image in the first row shows the environment and task setup. Green and red arrows denote the start and goal states. The planned path is visualized as a yellow polyline, with orange wireframe boxes indicating the robot states. The path cost c is shown in the bottom right corner of each example solution figure. Compared with sampling-based methods, ASA produces straighter paths with smoother rotation.

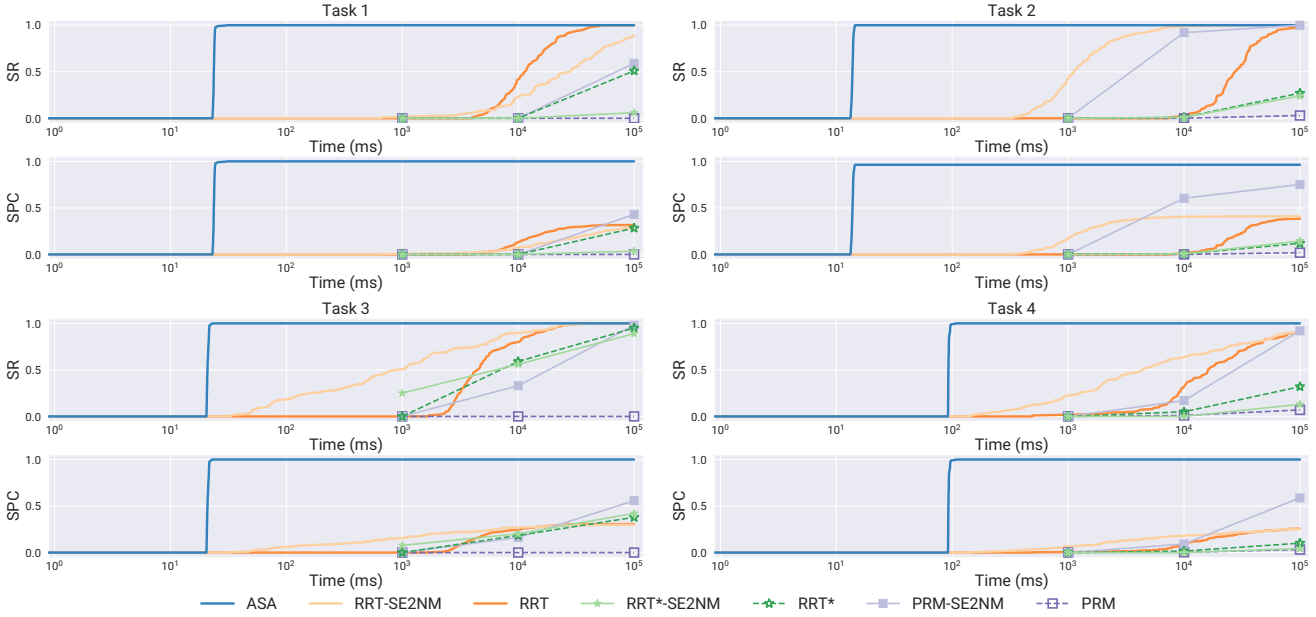
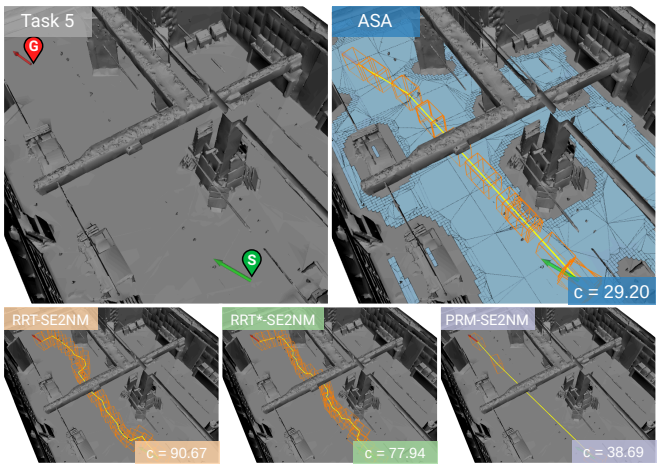
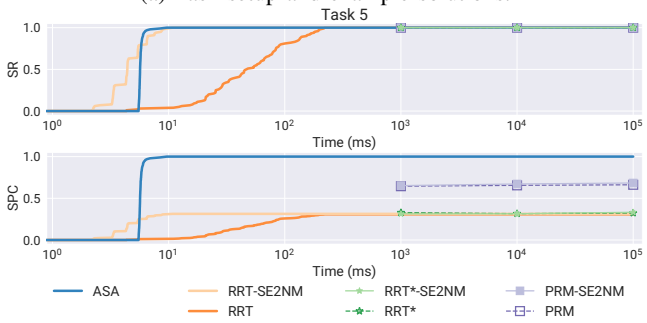


Fig. 13: Trends of SR and SPC over planning time for different planners in constrained environments. ASA achieves the best overall performance, while PRM-SE2NM generally achieves the highest SPC among sampling-based planners.

3) Randomly Sampled Start-Goal Evaluation: Among the multi-level scenes, *Gym* has the largest SE(2) NavMesh coverage and contains two major traversable components, with the largest traversable components spanning two floors connected by a staircase. Among the single-level scenes, *Studio* provides the largest coverage and is largely connected. These two



(a) Task setup and example solutions.



(b) Trends of SR and SPC over planning time for different planners.

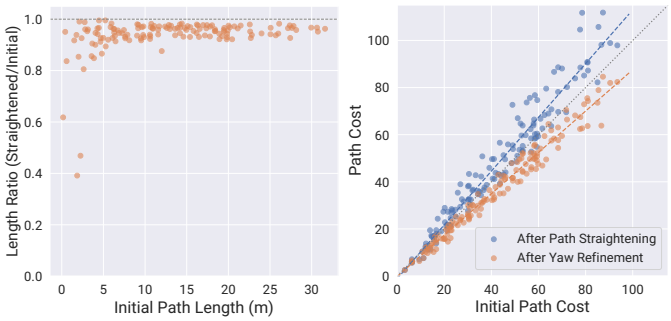
Fig. 14: Pathfinding performance in Task 5, an open-space scene. RRT-SE2NM requires slightly less planning time in open space, while ASA consistently produces lower-cost paths.

TABLE V: SR and SPC on Randomly Sampled Tasks

Planner	Gym		Studio	
	SR ↑	SPC ↑	SR ↑	SPC ↑
ASA	**41 %**	**0.41**	**98 %**	**0.98**
RRT-SE2NM	40 %	0.13	97 %	0.29
RRT*-SE2NM	32 %	0.18	82 %	0.26
PRM-SE2NM	**41 %**	0.27	96 %	0.63
RRT	**41 %**	0.13	97 %	0.29
RRT*	36 %	0.16	80 %	0.25
PRM	36 %	0.15	84 %	0.42

representative scenes are selected for the randomly sampled start-goal evaluation. In each scene, we randomly sample 100 start-goal pairs. Both the start and goal correspond to valid states within their respective traversable regions. Note that due to the random nature of the sampling, the start and goal do not necessarily belong to the same connected traversable component. For all sampling-based planners, the planning time limit is set to 100 s. The c_i^* used in SPC is the lowest cost achieved among all methods within the 100 s planning time limit.

Table V reports the SR and the SPC achieved by ASA and the sampling-based planners in both scenes. The connectivity of traversable regions strongly affects SR: the fragmented traversability structure in *Gym* leads to lower SR across all methods, with an average of 38 %, while the largely connected



(a) Path length reduction. (b) Path cost comparison.

Fig. 15: Path quality analysis of the ASA pipeline. Path straightening reduces path length but may increase path cost due to mismatched yaw assignments. The yaw refinement stage restores yaw consistency and further reduces the overall path cost.

Studio scene yields substantially higher SR, with an average of 91 %. Across both scenes, ASA consistently performs better than all sampling-based methods in overall performance, achieving the highest SR and SPC. For every successful start-goal problem, ASA achieves an average SPC close to 1, indicating that its successful solutions are consistently among the lowest-cost paths found by all methods. While RRT-SE2NM and RRT achieve comparable SR to ASA, their substantially lower SPC values suggest that they often produce higher-cost paths. Under the current planning time limit, whether state validity checking is performed on the generated SE(2) NavMesh has little effect on RRT and RRT*. In contrast, PRM-SE2NM substantially outperforms PRM, improving the average SR by 8.5 percentage points and increasing the average SPC from 0.29 to 0.45. When operating on the SE(2) NavMesh, state validity can be determined directly by querying the feasible yaw channel set of the corresponding traversable region without generating footprint masks. This allows PRM-SE2NM to perform significantly more sampling within the same time budget, greatly increasing the probability of finding feasible paths. The denser roadmap constructed from these additional samples also improves path quality, resulting in the highest SPC among all sampling-based planners.

4) Effectiveness of ASA Pathfinding: To assess the effectiveness of the ASA pathfinding strategy, we compare path costs and path lengths across its three stages using the same randomly sampled start-goal pairs (Section VI-B3). As shown in Figure 15a, path straightening consistently shortens the paths produced by the initial search, reducing the path length by 6.2 % on average. However, the refined intermediate positions become mismatched with the yaw assignments obtained in the initial stage, which increases the path cost by approximately 10 %, as indicated by the blue markers in Figure 15b. The final yaw-refinement stage effectively reduces the path cost by re-optimizing yaw at these positions, as shown by the orange markers in Figure 15b, and in most cases achieves a lower cost than the initial paths. Overall, the final cost averages 87 % of the initial path cost.

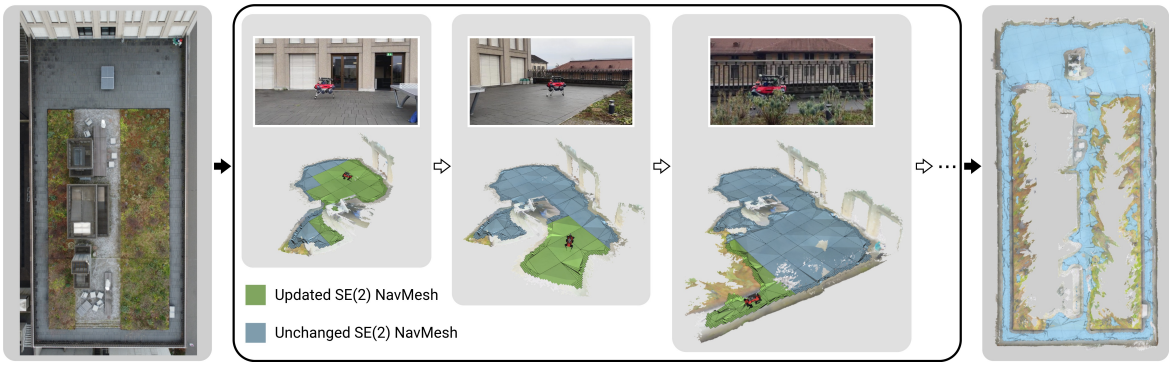


Fig. 16: Online SE(2) NavMesh generation in a garden. The left panel shows the real-world scene. The middle panel shows incremental geometry reconstruction and local SE(2) NavMesh updates. Only slabs affected by geometry changes have their local SE(2) NavMesh updated (green), while unaffected slabs retain their existing SE(2) NavMesh (blue). The right panel shows the final reconstructed geometry and the generated SE(2) NavMesh.

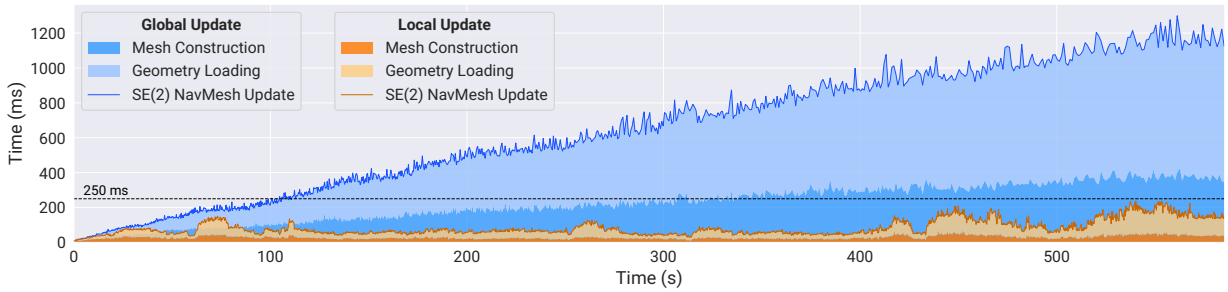


Fig. 17: Comparison of local and global SE(2) NavMesh update times during online generation. Local updates maintain the target real-time update rate, whereas global updates incur increasing latency as the reconstructed map grows.

TABLE VI: Voxblox and Slab Parameters Used for Online SE(2) NavMesh Generation

Category	Parameters	Values
Voxblox Voxel	$s_{\text{map}} \times s_{\text{map}} \times s_{\text{map}}$	$0.05\,\text{m} \times 0.05\,\text{m} \times 0.05\,\text{m}$
Voxblox Block	$d_{\text{block}} \times d_{\text{block}} \times d_{\text{block}}$	$16 \times 16 \times 16$
Slab	$d_{\text{tile}} \times d_{\text{tile}} \times d_{\text{slab}}$	$16 \times 16 \times 8$

C. Real-World Experiments

In the real-world experiments, we deploy the proposed online SE(2) NavMesh approach on an ANYmal platform equipped with a 6-DoF manipulator. A ZED X Mini camera is mounted on the end-effector of the manipulator to provide forward-facing perception. RGB-D images captured by the ZED X Mini are converted into colorized point clouds and downsampled to reduce computational load. The resulting point clouds are used as input to the proposed approach, with both image processing and SE(2) NavMesh generation executed onboard by an NVIDIA Jetson Orin. Due to the additional height introduced by the manipulator, the robot height h_{robot} is set to $1.00\,\text{m}$. The maximum slope angle θ_{climb} is set to $40°$, as we empirically observed that the robot can traverse steeper slopes than its nominal specifications. Voxblox parameters and slab parameters are presented in Table VI, while all remaining parameters of SE(2) NavMesh follow Table III. Under this configuration, each slab aligns with four Voxblox blocks in the horizontal plane.

We validate the online generation capability of the proposed SE(2) NavMesh through real-world outdoor and indoor experiments. In each experiment, the robot is first teleoperated to walk in the environment, during which the SE(2) NavMesh is incrementally generated from input point clouds. The resulting SE(2) NavMesh is then used for pathfinding to execute navigation tasks in the reconstructed environment. Instead of reconstructing it globally after each update, only the local SE(2) NavMesh for slabs affected by the latest updated geometry is generated. Figure 16 illustrates the process of the online SE(2) NavMesh generation in a garden. To evaluate the computational efficiency of this local update strategy, we use the point cloud data recorded during the walk in the garden and compare the update time (defined as the total execution time from the start to the completion of each update) of local SE(2) NavMesh updates against global SE(2) NavMesh updates. Figure 17 shows the update time of both methods over the duration of point cloud acquisition, with the target update frequency set to $4\,\text{Hz}$. The SE(2) NavMesh update time includes geometry loading, which comprises extraction of the required triangle mesh from Voxblox and construction of the bounding volume hierarchy, as well as NavMesh construction. For both local and global methods, geometry loading dominates the update time at each update step. For global updates, the update time increases approximately linearly as time evolves, leading to a continuous growth in update latency. Once the explored map becomes sufficiently large (after approximately

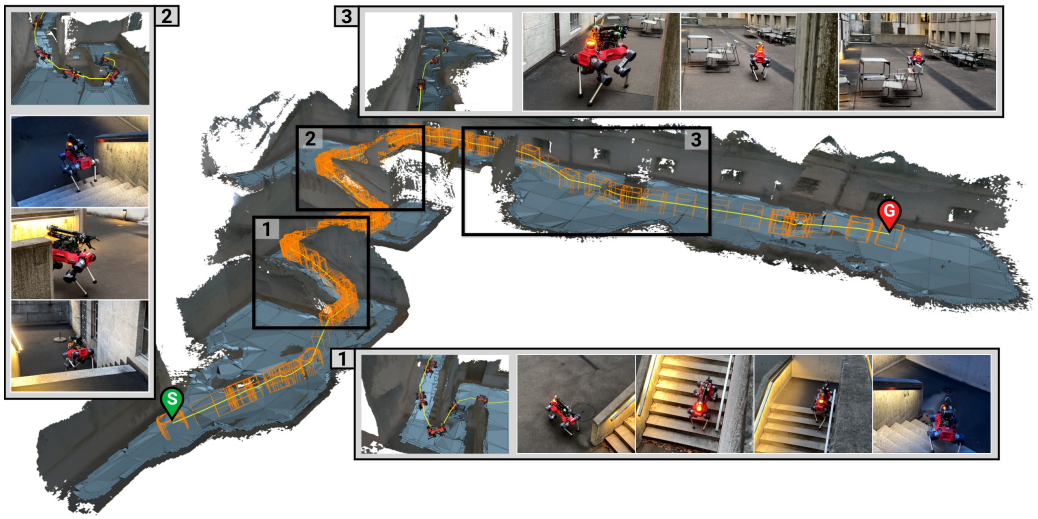


Fig. 18: Real-world navigation in an outdoor multi-level environment. Starting from the lower level, the robot follows a path planned on the SE(2) NavMesh to reach the goal on the upper platform. Along the route, it ascends multiple stair segments (1 and 2) and avoids tables and chairs on the platform (3).

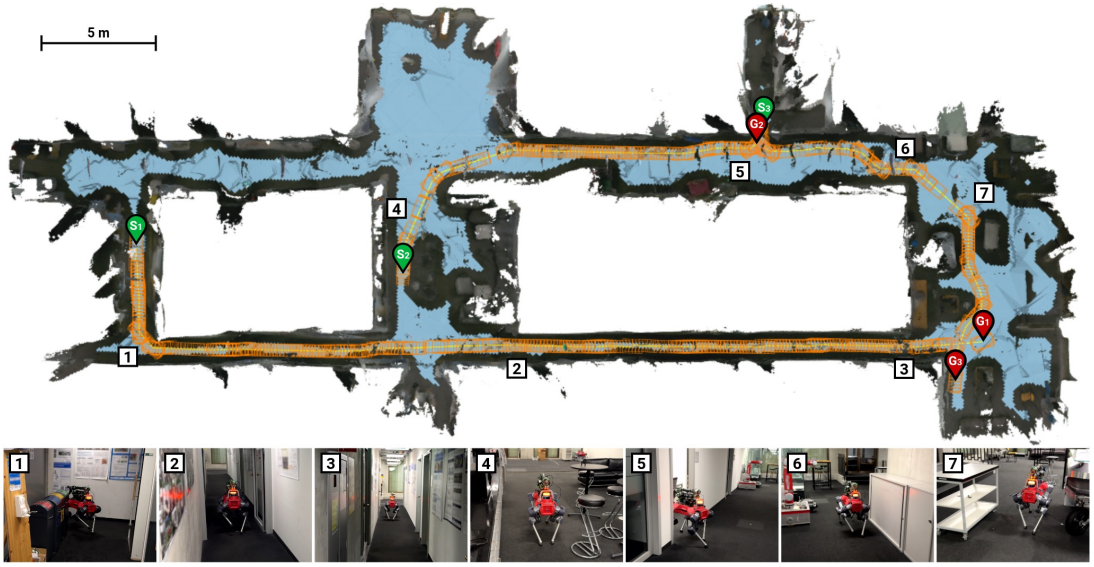


Fig. 19: Real-world navigation in an indoor single-level environment. The reconstructed scene and generated SE(2) NavMesh are shown with three planned paths. The robot traverses narrow corridors (1, 3), doorways (2 and 5), and cluttered regions (4, 6, and 7).

100 s in this experiment), the system can no longer sustain the target update frequency of 4 Hz. In contrast, the local update method achieves a peak update time of 237 ms and an average update time of 83 ms, maintaining the 4 Hz update frequency throughout the entire online SE(2) NavMesh generation. The computational behavior of local updates correlates with the spatial extent of geometry changes. When the robot traverses narrow corridors, the camera observes only nearby surfaces, resulting in limited geometry updates and short local update times. In open areas, a larger portion of the environment is reconstructed at each step, increasing the number of affected slabs and the corresponding update time. These results demonstrate that the proposed online SE(2) NavMesh pipeline achieves real-time generation by restricting updates to local regions.

Figure 18 shows the generated SE(2) NavMesh in an outdoor multi-level environment. The robot is first controlled to walk from the upper level to the lower level while incrementally generating the SE(2) NavMesh online. Using the generated SE(2) NavMesh, a navigation task is then defined with the start state located on the lower level and the goal state on the upper platform. The planned state sequence is tracked as navigation waypoints, enabling the robot to successfully ascend multiple stair segments and return to the upper platform. Figure 19 presents a single-level environment containing narrow corridors, doorways, and cluttered objects. Using the generated SE(2) NavMesh, the robot successfully completes a 41 m navigation task through a long narrow corridor following

the planned path. In addition, the robot consecutively executes two navigation tasks that require entering and exiting the designated room through narrow doorways, with path lengths of $19\,\mathrm{m}$ and $18\,\mathrm{m}$, respectively, without failure. We further evaluate the proposed approach in two challenging scenarios shown in Figure 1a and Figure 1b. The first scenario features a passage with a width of only $0.8\,\mathrm{m}$, requiring accurate yaw-aware navigation through tight lateral clearance. The second scenario requires the robot to navigate beneath an overhanging crane, demonstrating height-aware traversability estimation in the presence of limited overhead clearance. In both scenarios, the proposed approach accurately captures the traversable regions and generates feasible paths for the robot.

These experiments demonstrate that the proposed online SE(2) NavMesh pipeline supports real-time generation using only onboard sensing and computation. The approach is validated across diverse real-world scenes, including multi-level environments, cluttered spaces, overhangs, and narrow passages. By encoding yaw-dependent traversability, the generated SE(2) NavMesh enables reliable path planning in regions where feasible traversal strongly depends on robot heading.

VII. Limitations and Future Work

While our approach demonstrates strong performance in both simulation and real-world experiments, there are still some limitations. Some ground robots can adjust their body height, whereas our method considers only a fixed agent height. As a result, regions that could be traversed by lowering the agent height are filtered out during traversability estimation, leading to an underestimation of the traversable area in the environment. The current SE(2) NavMesh generation process relies solely on geometric information about the environment. Therefore, low-height objects placed on the ground may be confused with the ground surface and incorrectly classified as traversable. Incorporating semantic segmentation would help distinguish terrain from objects and differentiate terrain types. The proposed framework supports online SE(2) NavMesh generation from streaming sensor observations. Future work could integrate autonomous exploration strategies with the online generation process to enable fully autonomous mesh construction and navigation in previously unknown environments.

VIII. Conclusions

In this work, we presented SE(2) Navigation Mesh, a representation for global navigation of ground robots in complex 3D environments. By incorporating the robot's yaw into navigation mesh construction, SE(2) NavMesh captures yaw-dependent traversability that classical NavMeshes cannot represent. We introduced a yaw-specific layered representation that describes traversable regions under different robot headings and defines their inter-layer and intra-layer connectivity. We developed an ASA pathfinding strategy for SE(2) NavMesh to produce geometrically efficient paths while respecting yaw feasibility constraints. In addition, we proposed an online generation pipeline that constructs SE(2) NavMesh efficiently from point cloud observations. Experiments in multiple simulation scenes demonstrate that SE(2) NavMesh significantly improves traversable area coverage and connectivity compared with classical NavMeshes, while introducing only modest computational overhead. Real-world experiments further validate that SE(2) NavMesh can be generated online and can support reliable navigation of legged robots in complex environments.

References

[1] D. Lattanzi and G. Miller, "Review of robotic infrastructure inspection systems," *Journal of Infrastructure Systems*, vol. 23, no. 3, p. 04017004, 2017.

[2] S. Halder and K. Afsari, "Robots in inspection and monitoring of buildings and infrastructure: A systematic review," *Applied Sciences*, vol. 13, no. 4, p. 2304, 2023.

[3] H. Miura, A. Watanabe, M. Okugawa, T. Miura, and T. Koganeya, "Plant inspection by using a ground vehicle and an aerial robot: lessons learned from plant disaster prevention challenge in world robot summit 2018," *Advanced Robotics*, vol. 34, no. 2, pp. 104–118, 2020.

[4] P. Kim, J. Park, Y. K. Cho, and J. Kang, "Uav-assisted autonomous mobile robot navigation for as-is 3d data collection and registration in cluttered environments," *Automation in Construction*, vol. 106, p. 102918, 2019.

[5] G. K. Fischer, M. Bergau, D. A. Gómez-Rosal, A. Wachaja, J. Graeter, M. Odenweller, U. Piechottka, F. Höflinger, N. Gosala, N. Wetzel *et al.*, "Evaluation of a smart mobile robotic system for industrial plant inspection and supervision," *IEEE Sensors Journal*, vol. 24, no. 12, pp. 19 684–19 697, 2024.

[6] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, "Learning robust perceptive locomotion for quadrupedal robots in the wild," *Science Robotics*, vol. 7, no. 62, p. eabk2822, 2022.

[7] A. Agarwal, A. Kumar, J. Malik, and D. Pathak, "Legged locomotion in challenging terrains using egocentric vision," in *Conference on robot learning*. PMLR, 2023, pp. 403–415.

[8] H. Kolvenbach, D. Wisth, R. Buchanan, G. Valsecchi, R. Grandia, M. Fallon, and M. Hutter, "Towards autonomous inspection of concrete deterioration in sewers with legged robots," *Journal of field robotics*, vol. 37, no. 8, pp. 1314–1327, 2020.

[9] K. Afsari, S. Halder, M. Ensafi, S. DeVito, and J. Serdakowski, "Fundamentals and prospects of four-legged robot application in construction progress monitoring," *EPiC series in built environment*, vol. 2, pp. 274–283, 2021.

[10] N. Pérez-Higueras, A. Jardón, Á. Rodríguez, and C. Balaguer, "3d exploration and navigation with optimal-rrt planners for ground robots in indoor incidents," *Sensors*, vol. 20, no. 1, p. 220, 2019.

[11] F. Yang, C. Cao, H. Zhu, J. Oh, and J. Zhang, "Far planner: Fast, attemptable route planner using dynamic visibility update," in *2022 ieee/rsj international conference on intelligent robots and systems (iros)*. IEEE, 2022, pp. 9–16.

[12] F. Kong, X. Liu, B. Tang, J. Lin, Y. Ren, Y. Cai, F. Zhu, N. Chen, and F. Zhang, "Marsim: A light-weight point-realistic simulator for lidar-based uavs," *IEEE Robotics and Automation Letters*, vol. 8, no. 5, pp. 2954–2961, 2023.

[13] M. Liu, "Robotic online path planning on point cloud," *IEEE transactions on cybernetics*, vol. 46, no. 5, pp. 1217–1228, 2015.

[14] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments," *Journal of Field Robotics*, vol. 34, no. 5, pp. 940–984, 2017.

[15] B. Yang, J. Cheng, B. Xue, J. Jiao, and M. Liu, "Efficient global navigational planning in 3-d structures based on point cloud tomography," *IEEE/ASME Transactions on Mechatronics*, vol. 30, no. 1, pp. 321–332, 2025.

[16] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1366–1373.

[17] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous robots*, vol. 34, no. 3, pp. 189–206, 2013.

[18] X. Zhou, Z. Wang, H. Ye, C. Xu, and F. Gao, "Ego-planner: An esdf-free gradient-based local planner for quadrotors," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 478–485, 2020.

[19] Y. Ren, Y. Cai, F. Zhu, S. Liang, and F. Zhang, "Rog-map: An efficient robocentric occupancy grid map for large-scene and high-resolution lidar-based motion planning," in *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2024, pp. 8119–8125.

[20] J. Wang, L. Xu, H. Fu, Z. Meng, C. Xu, Y. Cao, X. Lyu, and F. Gao, "Towards efficient trajectory generation for ground robots beyond 2d environment," in *2023 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2023, pp. 7858–7864.

[21] J. Frey, D. Hoeller, S. Khattak, and M. Hutter, "Locomotion policy guided traversability learning using volumetric representations of complex environments," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 5722–5729.

[22] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," *Computer*, vol. 22, no. 6, pp. 46–57, 2002.

[23] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, "Robot-centric elevation mapping with uncertainty estimates," in *Mobile Service Robotics*. World Scientific, 2014, pp. 433–440.

[24] P. Fankhauser, M. Bloesch, and M. Hutter, "Probabilistic terrain mapping for mobile robots with uncertain localization," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3019–3026, 2018.

[25] T. Miki, L. Wellhausen, R. Grandia, F. Jenelten, T. Homberger, and M. Hutter, "Elevation mapping for locomotion and navigation using gpu," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 2273–2280.

[26] S. K. Ramakrishnan, A. Gokaslan, E. Wijmans, O. Maksymets, A. Clegg, J. Turner, E. Undersander, W. Galuba, A. Westbury, A. X. Chang *et al.*, "Habitat-matterport 3d dataset (hm3d): 1000 large-scale 3d environments for embodied ai," *arXiv preprint arXiv:2109.08238*, 2021.

[27] M. Khanna*, Y. Mao*, H. Jiang, S. Haresh, B. Shacklett, D. Batra, A. Clegg, E. Undersander, A. X. Chang, and M. Savva, "Habitat Synthetic Scenes Dataset (HSSD-200): An Analysis of 3D Scene Scale and Realism Tradeoffs for ObjectGoal Navigation," *arXiv preprint*, 2023.

[28] F. Ruetz, E. Hernández, M. Pfeiffer, H. Oleynikova, M. Cox, T. Lowe, and P. Borges, "Ovpc mesh: 3d free-space representation for local ground vehicle navigation," in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 8648–8654.

[29] S. Pütz, T. Wiemann, M. K. Piening, and J. Hertzberg, "Continuous shortest path vector field navigation on 3d triangular meshes for mobile robots," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 2256–2263.

[30] W. Van Toll, R. Triesscheijn, M. Kallmann, R. Oliva, N. Pelechano, J. Pettré, and R. Geraerts, "A comparative study of navigation meshes," in *Proceedings of the 9th International Conference on Motion in Games*, 2016, pp. 91–100.

[31] Unity Technologies, "Navigation overview," https://docs.unity3d.com/Packages/com.unity.ai.navigation@2.0/manual/NavigationOverview.html, 2026.

[32] Epic Games, "Navigation system," https://dev.epicgames.com/documentation/en-us/unreal-engine/navigation-system-in-unreal-engine, 2024.

[33] K. Yadav, J. Krantz, R. Ramrakhya, S. K. Ramakrishnan, J. Yang, A. Wang, J. Turner, A. Gokaslan, V.-P. Berges, R. Mootaghi, O. Maksymets, A. X. Chang, M. Savva, A. Clegg, D. S. Chaplot, and D. Batra, "Habitat challenge 2023," https://aihabitat.org/challenge/2023/, 2023.

[34] M. Brandao, O. B. Aladag, and I. Havoutis, "GaitMesh: Controller-aware navigation meshes for long-range legged locomotion planning in multi-layered environments," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3596–3603, 2020.

[35] M. Mononen, "Recast Navigation," https://github.com/recastnavigation/recastnavigation, 2014.

[36] M. Wand, A. Berner, M. Bokeloh, P. Jenke, A. Fleck, M. Hoffmann, B. Maier, D. Staneker, A. Schilling, and H.-P. Seidel, "Processing and interactive editing of huge point clouds from 3d scanners," *Computers & Graphics*, vol. 32, no. 2, pp. 204–220, 2008.

[37] T. Tuna, J. Nubert, Y. Nava, S. Khattak, and M. Hutter, "X-icp: Localizability-aware lidar registration for robust localization in extreme environments," *IEEE Transactions on Robotics*, vol. 40, pp. 452–471, 2023.

[38] E. Jelavic, J. Nubert, and M. Hutter, "Open3d slam: Point cloud based mapping and localization for education," in *Robotic Perception and Mapping: Emerging Techniques, ICRA 2022 Workshop*. ETH Zurich, Robotic Systems Lab, 2022, p. 24.

[39] G. Medioni, C.-K. Tang, and M.-S. Lee, "Tensor voting: Theory and applications," in *Proceedings of RFIA*, vol. 2000. Hermes, Lavoisier Paris, France, 2000.

[40] E. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[41] S. M. LaValle, "Rapidly-exploring random trees: Progress and prospects," *Algorithmic and Computational Robotics: New Directions 2000 WAFR*, p. 293, 2001.

[42] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[43] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle, "Surface reconstruction from unorganized points," in *Proceedings of the 19th annual conference on computer graphics and interactive techniques*, 1992, pp. 71–78.

[44] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3d reconstruction at scale using voxel hashing," *ACM Transactions on Graphics (ToG)*, vol. 32, no. 6, pp. 1–11, 2013.

[45] B. Tang, Y. Ren, Y. Cai, F. Kong, W. Liu, F. Zhu, L. Yin, L. Shi, and F. Zhang, "Memory-efficient boundary map for large-scale occupancy grid mapping," *The International Journal of Robotics Research*, p. 02783649261425266, 2026.

[46] D. Duberg and P. Jensfelt, "Ufomap: An efficient probabilistic 3d mapping framework that embraces the unknown," *IEEE Robotics and Automation Letters*, vol. 5, no. 4, pp. 6411–6418, 2020.

[47] C. Chen, J. Frey, P. Arm, and M. Hutter, "Smug planner: A safe multi-goal planner for mobile robots in challenging environments," *IEEE Robotics and Automation Letters*, vol. 8, no. 11, pp. 7170–7177, 2023.

[48] Y. Li, K. Chen, Y. Wang, W. Zhang, J. Wang, H. Chen, and Y. Liu, "Real-time multi-level terrain-aware path planning for ground mobile robots in large-scale rough terrains," *IEEE Transactions on Robotics*, 2025.

[49] J. Ahtiainen, T. Stoyanov, and J. Saarinen, "Normal distributions transform traversability maps: lidar-only approach for traversability mapping in outdoor environments," *Journal of Field Robotics*, vol. 34, no. 3, pp. 600–621, 2017.

[50] D. V. Lu, D. Hershberger, and W. D. Smart, "Layered costmaps for context-sensitive navigation," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2014, pp. 709–715.

[51] L. Wellhausen and M. Hutter, "Rough terrain navigation for legged robots using reachability planning and template learning," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 6914–6921.

[52] K. Hauser, "Lazy collision checking in asymptotically-optimal motion planning," in *2015 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2015, pp. 2951–2957.

[53] B. Yang, L. Wellhausen, T. Miki, M. Liu, and M. Hutter, "Real-time optimal navigation planning using learned motion costs," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2021, pp. 9283–9289.

[54] T. Wiemann, I. Mitschke, A. Mock, and J. Hertzberg, "Surface reconstruction from arbitrarily large point clouds," in *2018 Second IEEE International Conference on Robotic Computing (IRC)*. IEEE, 2018, pp. 278–281.

[55] J. J. Kuffner and S. M. 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. IEEE, 2000, pp. 995–1001.

[56] 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.

[57] X. Cui and H. Shi, "A*-based pathfinding in modern computer games," *International Journal of Computer Science and Network Security*, vol. 11, no. 1, pp. 125–130, 2011.

[58] S. Beucher, "Use of watersheds in contour detection," in *Proc. Int. Workshop on Image Processing, Sept. 1979*, 1979, pp. 17–21.

[59] D. Eberly, "Triangulation by ear clipping," *Geometric Tools*, pp. 2002–2005, 2008.

[60] M. Mononen, "Simple Stupid Funnel Algorithm," https://digestingduck.blogspot.com/2010/03/simple-stupid-funnel-algorithm.html, 2010.

[61] W. E. Lorensen and H. E. Cline, "Marching cubes: A high resolution 3d surface construction algorithm," *ACM SIGGRAPH Computer Graphics*, vol. 21, no. 4, pp. 163–169, 1987.

[62] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch *et al.*, "Anymal-a

highly mobile and dynamic quadrupedal robot," in *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2016, pp. 38–44.

[63] ANYbotics, "Anymal technical specifications," https://www.anybotics.com/anymal-technical-specifications.pdf, 2022.

[64] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 2002.

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

[66] P. Anderson, A. Chang, D. S. Chaplot, A. Dosovitskiy, S. Gupta, V. Koltun, J. Kosecka, J. Malik, R. Mottaghi, M. Savva *et al.*, "On evaluation of embodied navigation agents," *arXiv preprint arXiv:1807.06757*, 2018.

[67] N. Yokoyama, S. Ha, and D. Batra, "Success weighted by completion time: A dynamics-aware evaluation criteria for embodied navigation," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2021, pp. 1562–1569.