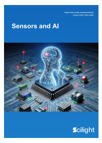
Sensors and AI
https://www.sciltp.com/journals/sai



Article

A Dynamically Updating Graph-Based Navigation Scheme for Autonomous Vehicles

Timothy Sellers [1,2,†], Tingjun Lei [3,†], Chaomin Luo [1,2,*], Zhuming Bi [4] and Gene Eu Jan [5]

[1] Department of Electrical and Computer Engineering, Mississippi State University, Mississippi State, MS 39762, USA
[2] Center for Advanced Vehicular Systems, Mississippi State University, Starkville, MS 39759, USA
[3] School of Electrical Engineering and Computer Science, University of North Dakota, Grand Forks, ND 58202, USA
[4] Department of Civil and Mechanical Engineering, Purdue University Fort Wayne, Fort Wayne, IN 46805, USA
[5] Department of Computer Science and Information Engineering, Asia University, Wufeng, Taichung 41354, Taiwan
* Correspondence: chaomin.luo@ece.msstate.edu
† These authors contributed equally to this work.

How To Cite: Sellers, T.; Lei, T.; Luo, C.; et al. A Dynamically Updating Graph-Based Navigation Scheme for Autonomous Vehicles. *Sensors and AI* **2026**, *2*(1), 1–16.

Received: 9 October 2025
Revised: 10 November 2025
Accepted: 21 November 2025
Published: 8 January 2026

Abstract: The deployment of autonomous vehicles (AVs) in unstructured and dynamic environments such as construction, military, and commercial operations demands robust navigation strategies capable of adapting to continuously changing obstacles. Traditional graph-based path planning methods often fail in these settings, leading to inefficient and suboptimal trajectories. In this paper, we propose a dynamically updating navigation framework that integrates real-time obstacle clustering, a Dynamically-constrained Delaunay Triangulation (D^2T), and an enhanced Ant Colony Optimization (*eACO*) algorithm. Our approach first clusters proximate obstacles using LiDAR data to simplify the environment representation. A local D^2T graph is then incrementally constructed around the robot, facilitating efficient map updates. The eACO algorithm leverages a greedy exploration strategy, enhanced by Lévy flight, to find near optimal paths within the D^2T graph. Finally, a velocity obstacle-based local reactive navigator ensures safe real-time obstacle avoidance. Extensive simulations and comparison studies validate the framework's superior performance in path length, computational speed, and overall adaptability compared to state-of-the-art path planning techniques.

Keywords: delaunay triangulation; autonomous vehicle; graph-based path planning; navigation; obstacle clustering

1. Introduction

Robotics has profoundly transformed industries such as medicine, transportation, and agriculture [1–4]. The growing trend toward full autonomy enables robots and autonomous vehicles to operate autonomously, thereby enhancing operational efficiency and safety [5–8]. A cornerstone of this autonomy is the capability for intelligent navigation and environmental mapping. To this end, graph-based methodologies are widely employed, modeling the environment as mathematical structures of interconnected nodes and edges. These graph-based techniques facilitate spatial understanding for path planning and obstacle avoidance, enable adaptation to real-time environmental changes, and enhance a system's ability to manage uncertainty in complex and dynamic settings [9–13].

1.1. Related Work

Graph-based methods are foundational to solving complex problems in robotics and autonomous systems, particularly in optimization, navigation, and motion planning. Their application spans autonomous vehicle (AV) navigation [14–24], motion planning [25–31], and sensor based perception [32–36]. A primary challenge in this domain is managing computational complexity. To this end, algorithms like Lazy Receding Horizon A* [14] reduce computational burden by balancing edge evaluation and path generation, while others like the L* algorithm [20] achieve linear computational complexity for mobile robot path planning.

Recent advancements have increasingly leveraged learning based methods for dynamic navigation. Deep reinforcement learning (DRL) has been applied to learn complex obstacle avoidance policies directly from sensor data, demonstrating robust performance in unstructured environments [37]. Concurrently, there has been a growing integration of graph structures with learning algorithms. For instance, Graph Neural Networks (GNNs) have been utilized to model the spatiotemporal relationships between agents and obstacles for predictive navigation in dynamic scenes [38], while other works have combined graph-based representations with reinforcement learning to enhance decision making [39]. Furthermore, Transformer based architectures are emerging as a powerful tool for path planning, capable of handling long range dependencies in crowded environments [40]. While these learning based approaches show significant promise, they often require extensive training data and computational resources. In contrast, our work focuses on an efficient geometric graph framework that guarantees performance without dependency on training data, providing a complementary approach to the learning based paradigm.

However, a key limitation persists in many graph-based methods: the generation of dense graphs with excessive vertices and edges for dynamic environments, leading to high computational load and inefficient path planning when obstacles change configuration [41]. This is particularly problematic in real-time settings where computational resources are constrained. While sampling-based planners like RRT* [42] address continuous spaces, their inherent randomness can lead to slow convergence and unpredictable computation times, making them less suitable for scenarios requiring deterministic performance. Unlike prior static graph-based methods, our approach dynamically reconstructs a sparse topological graph with lower computational cost, specifically designed for unstructured and dynamic environments.

A significant body of research has sought to enhance path planning for complex environments. This includes work on unstructured terrain [19], high dimensional spaces using topological constraints [22], and specific domains like multi-agent systems [17] and real-time adaptation [15]. However, a key limitation persists in many graph-based methods: the generation of dense graphs with excessive vertices and edges for dynamic environments, leading to high computational load and inefficient path planning when obstacles change configuration. Our work addresses this gap by introducing a framework that dynamically constructs a sparse, locally relevant graph, enabling efficient replanning and robust navigation.

1.2. Proposed Framework and Contributions

This paper introduces a novel framework for dynamic robot navigation and mapping designed to overcome key limitations in existing graph-based planners. Specifically, many current methods suffer from the computational burden of merging subgraphs and generate excessively dense graphs with redundant edges. These issues lead to slow search times and inefficiency in dynamic environments. To address these challenges, we propose a path planning scheme that uses a dynamically updating graph, emphasizing sparse and efficient local graph construction. The overall architecture of our framework is illustrated in Figure 1.

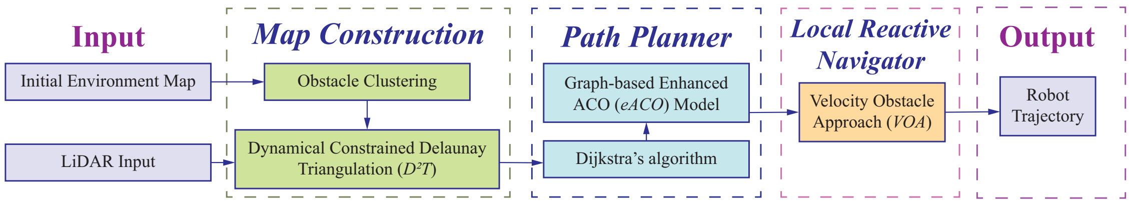


Figure 1. Illustration of the proposed hierarchical navigation framework. The system operates in a closed loop: LiDAR data is processed for Obstacle Clustering, which groups nearby obstacles to simplify the environment. The D^2T module then constructs a sparse, local graph of the free space. The eACO planner finds a globally efficient path on this graph. Finally, the Velocity Obstacle (VO) Local Reactive Navigator handles real-time avoidance of dynamic obstacles, with the robot's new position feeding back to restart the cycle. This tiered structure ensures both long term optimality and immediate safety.

Our main contributions are summarized as follows:

(1) A Hierarchical Navigation Framework: We introduce a tiered structure that systematically decomposes the navigation task. This design integrates high level global routing, based on a locally constructed Delaunay Triangulation (DT), with a low level reactive navigator. This leverages the computational efficiency of DT for precise planning in complex environments.

(2) An Efficient Obstacle Clustering and Mapping Pipeline: We developed a methodology that fuses LiDAR data to cluster proximate obstacles, thereby simplifying the environment representation. This pipeline reduces navigational uncertainty and accelerates the subsequent Delaunay Triangulation process, enabling faster and

more reliable spatial reasoning.

(3) An Enhanced Ant Colony Optimization (eACO) Algorithm: We propose an improved graph-based path planner that incorporates a Lévy flight strategy into a greedy exploration mechanism. This enhancement allows the algorithm to find efficient, near optimal paths within the sparse DT graph, particularly in challenging and irregular terrains.

(4) A Velocity Obstacle based Local Reactive Navigator: We implemented a real-time module to handle sudden environmental changes. This local reactive planner provides immediate adjustments for dynamic obstacles, ensuring safe motion without triggering a computationally expensive global replanning cycle.

The remainder of this paper is organized as follows: Section 2 details the obstacle clustering method. Section 3 presents the Dynamically-constrained Delaunay Triangulation (D^2T). Section 4 describes the enhanced ACO (eACO) algorithm. Section 5 explains the reactive local navigation strategy. Section 6 illustrates the theoretical analysis and complexity of the proposed model. Section 7 provides simulation results and comparative studies. Finally, Section 8 concludes the paper and discusses future work.

2. Obstacle Clustering

This section details our obstacle clustering methodology, which simplifies environmental complexity by merging proximate obstacles into larger convex hulls. This reduction facilitates more efficient graph construction and path planning [43]. We consider a robot operating in a two dimensional space populated by n polygonal obstacles, as illustrated in Figure 2a. The core of our technique is a proximity threshold, Ψ, which is set based on the robot's physical dimensions and desired clearance. Specifically, Ψ is chosen to be slightly larger than the robot's diameter to ensure that gaps which the robot cannot safely traverse are merged into a single navigational obstacle. The distance $D(U_i, U_j)$ between two obstacles is computed as the minimum Euclidean distance between their boundary points. If $D(U_i, U_j) < \Psi$, the obstacles are considered adjacent. This relationship defines a Ψ proximity graph (Figure 2(C-1)) [43], which is subsequently refined into a Ψ planner graph by removing connections impeded by other obstacles.

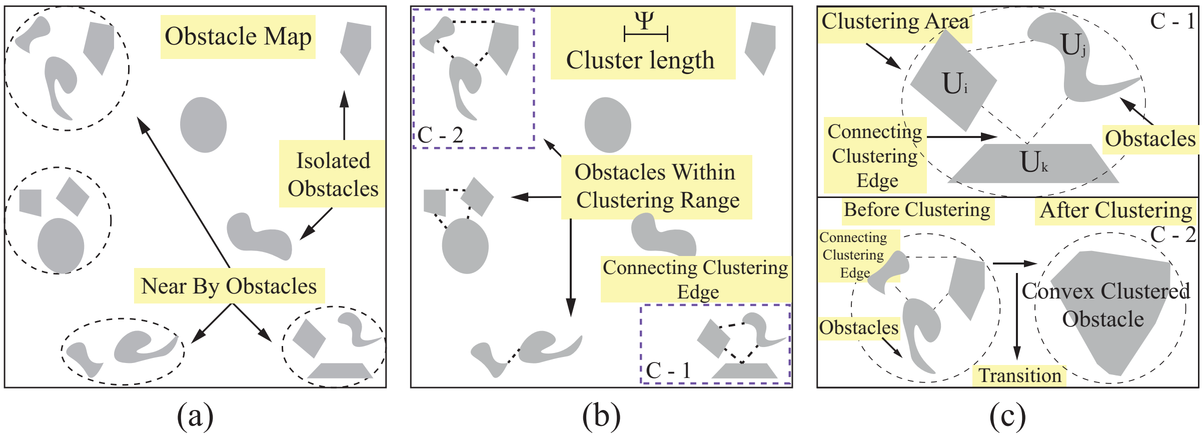


Figure 2. The obstacle clustering pipeline. (**a**) The original environment with numerous discrete polygonal obstacles. Potential clusters are intuitively outlined. (**b**) The application of the proximity threshold Ψ to determine adjacency between obstacles. (**C-1**) The resulting Ψ proximity graph, where nodes represent obstacles and edges connect those within distance Ψ. (**C-2**) The final, simplified environment where connected obstacles in (**C-1**) have been merged into larger, consolidated convex hulls (shown in purple). This process dramatically reduces the number of entities the planner must consider, lowering computational complexity.

Algorithm 1 formalizes this process. It constructs the Ψ proximity graph G by connecting obstacles closer than Ψ that also have an unobstructed line of sight. The connected components of G are then identified, forming the initial clusters. Finally, each cluster is encapsulated by its convex hull, resulting in a simplified set of obstacles for path planning. For convex polygons, the distance check between a pair can be performed efficiently in $O(\log n)$ time after an $O(n)$ preprocessing step. Identifying the connected components using a depth-first or breadth-first search runs in $O(V + E)$ time for a graph with V vertices (obstacles) and E edges (proximity links). This clustering step is crucial for reducing the perceptual and computational load in subsequent planning stages, leading to a more scalable navigation system.

Algorithm 1: Obstacle Clustering

```
Input: Set of obstacles $U = \{U_1, U_2, \ldots, U_n\}$, Proximity threshold $\Psi$
Output: A set of clusters $C = \{C_1, C_2, \ldots, C_k\}$, where each $C_i$ is a connected component in the $\Psi$
        proximity graph.
Initialize an empty adjacency list $G$;
for each unique pair of obstacles $(U_i, U_j)$ do
    $d_{\text{min}} \leftarrow$ minimum Euclidean distance between boundaries of $U_i$ and $U_j$;
    if $d_{min} < \Psi$ and the line of sight between $U_i$ and $U_j$ is unobstructed then
        Add an undirected edge between $U_i$ and $U_j$ in $G$;
    end
end
$C \leftarrow$ FindConnectedComponents$(G)$;
for each cluster $c \in C$ do
    $c \leftarrow$ ComputeConvexHull$(\bigcup_{U_i \in c} U_i)$;
end
return $C$;
```

3. Dynamically Constrained Delaunay Triangulation Formation

Delaunay Triangulation (DT) is a fundamental geometric structure widely used in path planning due to its optimal properties, which promote well shaped triangles and efficient spatial reasoning. Among its most important characteristics are the empty circumcircle property ensuring no vertex lies inside the circumcircle of any triangle and the max-min angle property, which maximizes the minimum angle among all triangles to avoid skinny geometries. These properties yield a graph that provides a natural and efficient representation of the free space [44].

However, a standard DT of all obstacle vertices includes edges that pass through obstacles, rendering it unsuitable for collision free path planning. To overcome this, we introduce the Dynamically-constrained Delaunay Triangulation (D^2T). The D^2T algorithm incrementally constructs a local triangulation around the robot's position while respecting two critical constraints:

- Obstacle Constraints: Edges that intersect obstacles are explicitly forbidden.
- Dynamic Updates: The graph is continuously rebuilt based on the robot's sensory horizon rather than once for the entire environment.

This approach allows the graph to adapt to new or moving obstacles. The result is a sparse, locally relevant graph that accurately represents the traversable space, providing a robust foundation for global path planning within the current field of view.

The core objective of D^2T is to incrementally construct a sparse, locally valid Delaunay Triangulation graph that explicitly avoids obstacle regions, based on the robot's current sensor data. The algorithm takes as input the clustered obstacles from the previous stage and the robot's current position and sensor range.

The D^2T algorithm, formalized in Algorithm 2, proceeds as follows. First, a set of points P_{free} is sampled within the robot's sensor range, excluding areas occupied by obstacles. The extreme vertices of the clustered obstacles are added to this set to ensure the graph captures the environment's structure. An initial Delaunay Triangulation G_{dt} is computed from the combined point set P_{all}.

The key step is enforcing constraints: any edge in G_{dt} that intersects an obstacle polygon is removed. This violates the standard Delaunay property for those edges but is necessary for navigability. Finally, the largest connected component of the resulting graph is extracted to form G_{cdt}, the Constrained Delaunay Graph. This graph provides a sparse representation of the free space, connecting the robot's position to nearby navigable areas and goal points while guaranteeing that all edges are obstacle free. The process is repeated periodically as the robot moves and new sensor data is received, ensuring the graph remains dynamically updated.

Algorithm 2: Dynamically-constrained Delaunay Triangulation (D^2T)

```
Input: Clustered obstacle points $P_{\text{obs}}$, Robot position $P_r$, Sensor range $D$
Output: Constrained Delaunay Graph $G_{\text{cdt}}$
// Step 1:  Sample points in local free space
$P_{\text{free}} \leftarrow \text{SamplePointsInCircle}(P_r, D) \setminus P_{\text{obs}}$;
$P_{\text{all}} \leftarrow P_{\text{free}} \cup \{\text{ExtremePoints}(P_{\text{obs}})\}$;
// Step 2:  Build initial Delaunay Triangulation
$G_{\text{dt}} \leftarrow \text{DelaunayTriangulation}(P_{\text{all}})$;
// Step 3:  Impose obstacle constraints
foreach edge $e_{ij} \in G_{dt}$ do
    if $e_{ij}$ intersects any obstacle polygon $U \in P_{obs}$ then
        $G_{\text{dt}}.remove\_edge(e_{ij})$;
    end
end
// Step 4:  Extract traversable graph component
$G_{\text{cdt}} \leftarrow \text{LargestConnectedComponent}(G_{\text{dt}})$;
return $G_{cdt}$;
```

4. Path Optimization on the Dynamically Updated Delaunay Graph

The constrained Delaunay graph G_{cdt} provides a topologically correct representation of the free space. While a simple graph search (e.g., Dijkstra's algorithm) on G_{cdt} can find a feasible path from start S to goal T, the result is often suboptimal. Such a path is constrained to the graph edges and may take unnecessary detours instead of cutting directly across large triangles, particularly in areas with low obstacle density.

To overcome this limitation, we formulate the path planning task as a continuous optimization problem. Let the initial path from the graph search traverse the sequence of edges $\{L_1, L_2, \ldots, L_n\}$, where each edge L_i connects vertices P_i^1 and P_i^2. We parameterize a more efficient path by allowing it to intersect each edge L_i at any point, rather than being restricted to its vertices. This is defined by a vector of scale factors $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_n)$, where each $\gamma_i \in [0, 1]$. The corresponding point on edge L_i is:

$$P_i(\gamma_i) = P_i^1 + \gamma_i(P_i^2 - P_i^1) \tag{1}$$

The total path length is then the sum of the Euclidean distances between consecutive points:

$$\mathcal{L}(\gamma) = \sum_{i=0}^{n} \|P_{i+1}(\gamma_{i+1}) - P_i(\gamma_i)\| \tag{2}$$

where $P_0 \equiv S$ and $P_{n+1} \equiv T$ are the fixed start and goal points, respectively.

Our objective is to find the optimal scaling vector γ^* that minimizes the total path length:

$$\gamma^* = \arg \min_{\gamma_i \in [0,1]} \mathcal{L}(\gamma) \tag{3}$$

This formulation transforms the discrete graph search into a continuous optimization over an n-dimensional unit cube. The structure of G_{cdt} guarantees that the straight line segments between consecutive points $\overline{P_i(\gamma_i)P_{i+1}(\gamma_{i+1})}$ remain collision free, as the entire path is contained within the triangulation's triangles. Solving Equation (3) thus yields a smooth, near optimal path that is significantly shorter than the original graph-based solution.

4.1. Enhanced Ant Colony Optimization (eACO)

To solve the path optimization problem defined in Equation (3), we propose an eACO algorithm. While standard ACO metaheuristics are effective, they can converge prematurely or struggle with exploration in complex search spaces. Our eACO enhances exploration by integrating a Lévy flight strategy into the state transition rule, helping to escape local minima and navigate the high dimensional unit cube defined by the scaling factors γ.

The Ant Colony System (ACS) forms the basis of our approach. In ACS, an ant k at node i chooses the next node j probabilistically. The core elements are:

- Pheromone Trail (τ_{ij}): Represents the learned desirability of edge (i, j).
- Heuristic Information (η_{ij}): Typically $\eta_{ij} = 1/d_{ij}$, where d_{ij} is the distance, providing greedy guidance.
- State Transition Rule: A pseudo random proportional rule balances exploration and exploitation:

$$j = \begin{cases} \arg\max_{u \in N_i^k} \left\{[\tau_{iu}]^\alpha [\eta_{iu}]^\beta\right\} & \text{if } q \le q_0 \\ J & \text{otherwise} \end{cases} \tag{4}$$

  where q is a random variable uniformly distributed in $[0, 1]$, q_0 is a parameter, and J is a node selected probabilistically according to:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta} \quad \text{if } j \in N_i^k \tag{5}$$

- Pheromone Update: Global updating is performed only on the best so far path:

$$\tau_{ij} \leftarrow (1\rho) \cdot \tau_{ij} + \rho \cdot \Delta\tau_{ij}^{bs} \tag{6}$$

  where $\Delta\tau_{ij}^{bs} = 1/L_{bs}$ if edge (i, j) belongs to the best so far tour of length L_{bs}.

4.2. Lévy Flight Integration

The standard probabilistic selection in Equation (5) can lead to slow exploration in high dimensional search spaces, as it tends to focus on immediate, high pheromone neighbors. To enhance global exploration and help the algorithm escape local optima, we integrate a strategy inspired by Lévy flights into the state transition rule.

A Lévy flight is a type of random walk characterized by a step length distribution with a heavy tail. This property promotes a pattern of many small steps interspersed with occasional long jumps, which is often more efficient for exploring unknown spaces than a standard random walk.

In our eACO, when an ant is in the exploration phase ($q > q_0$ in Equation (4)), we modify the selection probability. With a probability p_l (the Lévy application rate), we use a modified probability distribution $p_{ij}^{k,\text{levy}}$ instead of the standard p_{ij}^k from Equation (5). The modified probability is defined as:

$$p_{ij}^{k,\text{levy}} = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta + \lambda \cdot L(\beta_L)}{\sum_{l \in N_i^k} \left([\tau_{il}]^\alpha [\eta_{il}]^\beta + \lambda \cdot L(\beta_L)\right)} \tag{7}$$

where λ is a scaling factor that controls the influence of the Lévy component, and $L(\beta_L)$ is a step size generated from a Lévy distribution with parameter β_L, calculated using the Mantegna algorithm [45]:

$$L(\beta_L) = \frac{u}{|v|^{1/\beta_L}}, \quad u, v \sim \mathcal{N}(0, \sigma^2) \tag{8}$$

The standard deviation σ is calculated as:

$$\sigma = \left[\frac{\Gamma(1 + \beta_L) \cdot \sin(\pi\beta_L/2)}{\beta_L \cdot \Gamma((1 + \beta_L)/2) \cdot 2^{(\beta_L - 1)/2}}\right]^{1/\beta_L} \tag{9}$$

Mechanism Interpretation: The term $\lambda \cdot L(\beta_L)$ acts as an exploration bonus. When a large step $L(\beta_L)$ is generated, it significantly boosts the selection probability of edges that might otherwise have low pheromone or heuristic values. This encourages ants to explore less traveled paths, effectively widening the search scope and reducing the risk of premature convergence to a sub optimal path.

4.2.1. Application to Path Optimization

In our path optimization context, the "graph" for the eACO is the sequence of DT edges $\{L_1, \ldots, L_n\}$. An ant's solution is a vector $\gamma = (\gamma_1, \ldots, \gamma_n)$. The heuristic desirability η_i for a value γ_i on edge L_i can be related to its contribution to a straight line path towards the goal. The pheromone $\tau_i(\gamma_i)$ is reinforced on values of γ_i that belong to shorter overall paths.

Algorithm 3 outlines the complete eACO procedure for solving our path optimization problem. By combining the exploitation strength of ACS with the enhanced exploration of Lévy flights, eACO efficiently finds high quality solutions for the continuous path optimization problem.

```
Algorithm 3: eACO for Path Optimization
  Input: Sequence of edges ${L_1, ..., L_n}$, Start $S$, Goal $T$, eACO parameters
  Output: Optimal scaling factors $\gamma^*$
  Initialize pheromone trails $\tau$;
  while termination condition not met do
      foreach ant $k$ in colony do
          Construct path by selecting $\gamma_i$ for each edge $L_i$ using eACO rules;
          Evaluate path length $\mathcal{L}(\gamma)$ using Equation (2);
      end
      Update best so far solution $\gamma^{bs}$;
      Update pheromone trails based on $\gamma^{bs}$;
  end
  return $\gamma^{bs}$
```

4.3. Parameter Settings

The parameters (Table 1) for the eACO algorithm were determined through empirical tuning on a set of representative benchmark environments, balancing convergence speed and solution quality. The core ACS parameters were set to classic values found in the literature: the pheromone influence $\alpha = 1.0$, the heuristic influence $\beta = 2.0$ (giving slightly more weight to greedy distance minimization), the exploitation probability $q_0 = 0.7$ to favor convergence, and the pheromone evaporation rate $\rho = 0.1$ to allow a slow decay of trail information.

Table 1. Parameters for the eACO algorithm determining through empirical tuning on a set of representative benchmark environments, balancing convergence speed and solution quality.

Parameter	Value	Justification
α (Pheromone Influence)	1.0	Standard value to balance heuristic influence.
β (Heuristic Influence)	2.0	Gives slightly more weight to greedy distance minimization.
ρ (Evaporation Rate)	0.1	Allows for slow decay of pheromone trails, preventing premature convergence.
q_0 (Exploitation Probability)	0.7	Favors selection of the best edge to promote convergence.
p_l (Lévy Application Rate)	0.2	Moderate probability to inject exploration without disrupting constructive heuristic.
β_L (Lévy Exponent)	1.5	Generates heavy tailed step size distribution for effective exploration.
λ (Lévy Scaling Factor)	0.1	Scales exploration bonus to be comparable to pheromone/heuristic terms.
Colony Size	20	Standard size balancing solution diversity and computational cost.

The novel Lévy flight parameters were tuned to introduce substantial exploration without disrupting the constructive heuristic. The Lévy exponent $\beta_L = 1.5$ generates a heavy tailed distribution with a good mix of small and large steps. The scaling factor $\lambda = 0.1$ was chosen to ensure the exploration bonus $\lambda \cdot L(\beta_L)$ is on a comparable scale with the pheromone and heuristic terms, preventing it from dominating the selection process. The probability of applying the Lévy modification, p_l, was set to 0.2, ensuring it is a frequent but not constant occurrence. The colony size was set to 20 ants. The algorithm terminates after 50 iterations or if no improvement is found for 15 consecutive iterations. A sensitivity analysis confirmed that the algorithm's performance is robust to small variations around these chosen values.

4.4. eACO Integration into the Proposed Model

The complete navigation pipeline integrates all previously described components into a cohesive framework, as illustrated in Figure 3. The process begins with perception and clustering, where LiDAR data is processed to detect obstacle points. The obstacle clustering algorithm (Algorithm 1) then groups proximate obstacles, simplifying the environment representation, as shown in Figure 3a. Subsequently, the graph construction phase is initiated. The D^2T algorithm constructs a sparse, locally relevant graph around the robot's position that explicitly avoids obstacles, resulting in the structure depicted in Figure 3b.

Finally, path optimization is performed using the eACO algorithm (Algorithm 3) to find the optimal path through the D^2T graph. Unlike standard graph searches that are constrained to vertices, eACO treats the sequence of graph edges $\{L_1, \ldots, L_n\}$ traversed between start and goal as a continuous optimization problem. It optimizes the scaling factors γ_i to determine the precise crossing point on each edge, effectively creating shortcuts through large triangles and producing a smoother, shorter path than a vertex constrained route, as visualized in Figure 3c. This integration creates a powerful synergy: the D^2T provides a topologically correct and sparse graph foundation, while eACO performs continuous optimization over this graph structure to generate high quality paths. The framework operates in a receding horizon fashion, where these steps repeat as the robot moves and new sensor data becomes available, enabling robust navigation in dynamic environments.

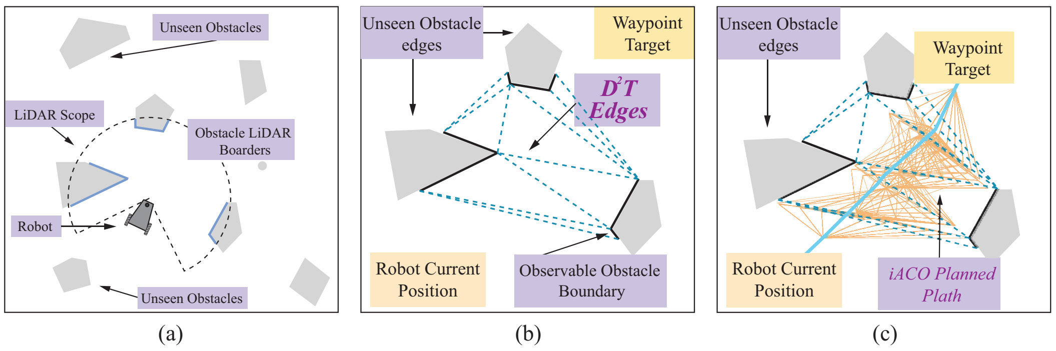


Figure 3. Illustration of the core navigation pipeline. (**a**) The perceived environment after LiDAR sensing and obstacle clustering, where proximate obstacles are merged into convex hulls (blue borders). (**b**) The resulting sparse, locally constructed D^2T graph, which provides a topologically correct representation of the free space. (**c**) The final, optimized path (in red) generated by the eACO algorithm. Note how the path cuts across large triangles, rather than being constrained to graph vertices, resulting in a shorter and smoother trajectory. This sequence demonstrates the effectiveness of the environmental simplification and continuous optimization stages.

5. The Local Reactive Navigator

The global path generated by the eACO algorithm provides a high level plan, but real-world navigation requires reacting to unforeseen dynamic obstacles. To address this, we implement a Velocity Obstacle (VO) based local reactive navigator for real-time obstacle avoidance and local mapping [46]. This module operates concurrently, generating velocity commands that guide the robot toward its immediate subgoal while avoiding collisions. The VO framework provides a mathematically rigorous method for collision checking. Consider an autonomous vehicle A and a dynamic obstacle O. Let $\mathbf{p}_A$, $\mathbf{p}_O$ denote their positions and $\mathbf{v}_A$, $\mathbf{v}_O$ their velocities. The vehicle has a fixed radius R_A, a goal position $\mathbf{p}_{goal}$, and a preferred speed v_{pref}.

To account for the vehicle's physical extent, the obstacle O is enlarged by the vehicle's radius R_A via the Minkowski sum, while the vehicle A is treated as a point. The Minkowski sum and set inversion are defined as:

$$A \oplus B = \{\mathbf{a} + \mathbf{b} \mid \mathbf{a} \in A, \mathbf{b} \in B\}, \quad -A = \{-\mathbf{a} \mid \mathbf{a} \in A\} \tag{10}$$

A ray cast from position $\mathbf{p}$ with velocity $\mathbf{v}$ is given by:

$$\text{Ray}(\mathbf{p}, \mathbf{v}) = \{\mathbf{p} + t\mathbf{v} \mid t \geq 0\} \tag{11}$$

The Velocity Obstacle $VO_{A|O}(\mathbf{v}_O)$ is defined as the set of all relative velocities $\mathbf{v}_A - \mathbf{v}_O$ that would result in a collision between A and O within a time horizon τ:

$$VO_{A|O}(\mathbf{v}_O) = \{\mathbf{v}_A \mid \text{Ray}(\mathbf{p}_A, \mathbf{v}_A - \mathbf{v}_O) \cap (O \oplus -A) \neq \emptyset\} \tag{12}$$

The vehicle is subject to dynamic and kinematic constraints that limit its achievable velocities to an admissible set $\mathcal{V}_A$. Each planning iteration selects an optimal velocity $\mathbf{v}_A^*$ from the collision free velocities $\mathcal{V}_{\text{free}} = \mathcal{V}_A \setminus \bigcup_O VO_{A|O}(\mathbf{v}_O)$, as illustrated in Figure 4b. This velocity command is executed for a short duration before the process repeats. This reactive approach allows the robot to deviate temporarily from the global path to avoid dynamic obstacles, then smoothly rejoin the original trajectory once the obstacle is passed. By leveraging efficient geometric computations, the local reactive navigator maintains real-time performance while ensuring safety in dynamic environments.

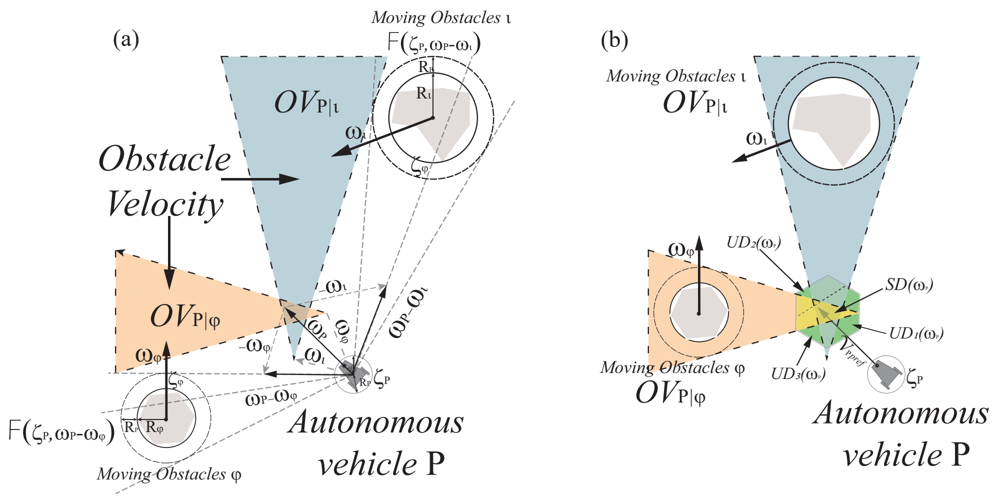


Figure 4. Illustration of velocity obstacle (VO) formulation for dynamic collision avoidance. (**a**) The autonomous vehicle A (reduced to a point) and a dynamic obstacle O (enlarged by the robot's radius via Minkowski sum). Their current velocities are $\mathbf{v}_A$ and $\mathbf{v}_O$. (**b**) The corresponding Velocity Obstacle region $VO_{A|O}(\mathbf{v}_O)$ (red cone) in the velocity space. This cone represents all relative velocities of A that would lead to a collision with O within a time horizon τ. The optimal velocity $\mathbf{v}_A^*$ is selected from the admissible velocities $\mathcal{V}_A$ (blue circle) that lie outside the VO, ensuring safe and collision free motion.

6. Theoretical Analysis of the Computational Complexity

This section provides a theoretical analysis of the computational complexity of the core components of our proposed framework. Understanding these properties is crucial for justifying the framework's suitability for real-time autonomous navigation.

6.1. Experimental Setup

All simulations were conducted in MATLAB R2020a on a Legion Slim 7 IRH8 laptop with an 13th Gen Intel(R) Core(TM) i9-13900H processor, 16 GB of RAM, running the Windows 11 operating system. This consistent hardware and software environment was used for all comparative studies and performance evaluations presented in the following sections.

6.2. Complexity of Obstacle Clustering

The obstacle clustering method (Algorithm 1) involves two main steps: graph construction and connected component analysis. For n obstacles, the naive distance check between all unique pairs has a time complexity of $O(n^2)$. However, this can be significantly optimized to an average of $O(n \log n)$ using spatial data structures like k-d trees for proximity queries. The subsequent step of finding connected components in the Ψ proximity graph via Depth-First Search (DFS) or Breadth-First Search (BFS) has a complexity of $O(V + E)$, where $V = n$ (vertices) and E is the number of edges in the proximity graph, which is typically $O(n)$ for sparse environments. Therefore, the overall complexity of the clustering step is dominated by the distance checks and is $O(n^2)$ in the worst case, but $O(n \log n)$ on average with spatial indexing.

6.3. Complexity of Dynamically-Constrained Delaunay Triangulation (D^2T)

The complexity of the D^2T algorithm (Algorithm 2) is primarily determined by the Delaunay Triangulation (DT) construction. Constructing a DT for m points has a well known worst case time complexity of $O(m \log m)$ [44]. In our framework, m is the number of points sampled within the robot's local sensor range, D, plus the extreme points of the clustered obstacles. This value, m, is intentionally kept small and bounded due to the local sensing horizon, ensuring that graph construction remains efficient. The subsequent steps of edge constraint checking and connected component extraction are linear with respect to the number of edges and vertices in the initial DT, i.e., $O(m)$. Thus, the overall complexity for building the local D^2T graph is $O(m \log m)$, which is efficient for real-time updates given the bounded nature of m.

6.4. Complexity and Convergence of Enhanced ACO (eACO)

The time complexity per iteration of the eACO algorithm (Algorithm 3) is $O(K \cdot N)$, where K is the number of ants in the colony and N is the number of edges in the path sequence being optimized (i.e., the length of the vector γ). This is because each ant constructs a solution by making a decision for each of the N parameters.

Regarding convergence, Ant Colony Optimization (ACO) algorithms have been proven to converge to the optimal solution with a probability that can be made arbitrarily close to 1 given a sufficiently long runtime [47]. Our eACO enhances the standard ACS with a Lévy flight strategy, which does not alter the fundamental global convergence properties but is designed to improve the *speed* of convergence and the ability to escape local optima. While a full theoretical proof of the convergence rate for our specific eACO variant is beyond the scope of this paper, we provide strong empirical evidence of its performance. As shown in Figure 5(B2), the eACO algorithm demonstrates rapid convergence towards a high quality solution within a few dozen iterations, making it highly suitable for practical path planning applications where computational time is limited.

7. Simulation and Comparison Studies

7.1. Path Planning Comparison Studies

To evaluate the performance of the proposed framework, we conducted comparative simulations against several classical and widely used path planning algorithms: Breadth-First Search (BFS), D*, Bi-Directional A*, and A*. These algorithms were selected for their relevance in graph-based planning and their ability to handle obstacle filled environments. All algorithms were tested in two distinct scenarios with varying obstacle configurations, as shown in Figures 5 and 6. Performance was evaluated based on two key metrics: total path length and total execution time, with results averaged over 50 executions for statistical significance.

The quantitative results in Table 2 demonstrate the effectiveness of our approach. The proposed framework achieved the shortest path lengths in both scenarios while maintaining competitive execution times. Specifically, it reduced path length by approximately 9% in Scenario 1 and 4% in Scenario 2 compared to the next best algorithm, while also achieving the fastest computation time.

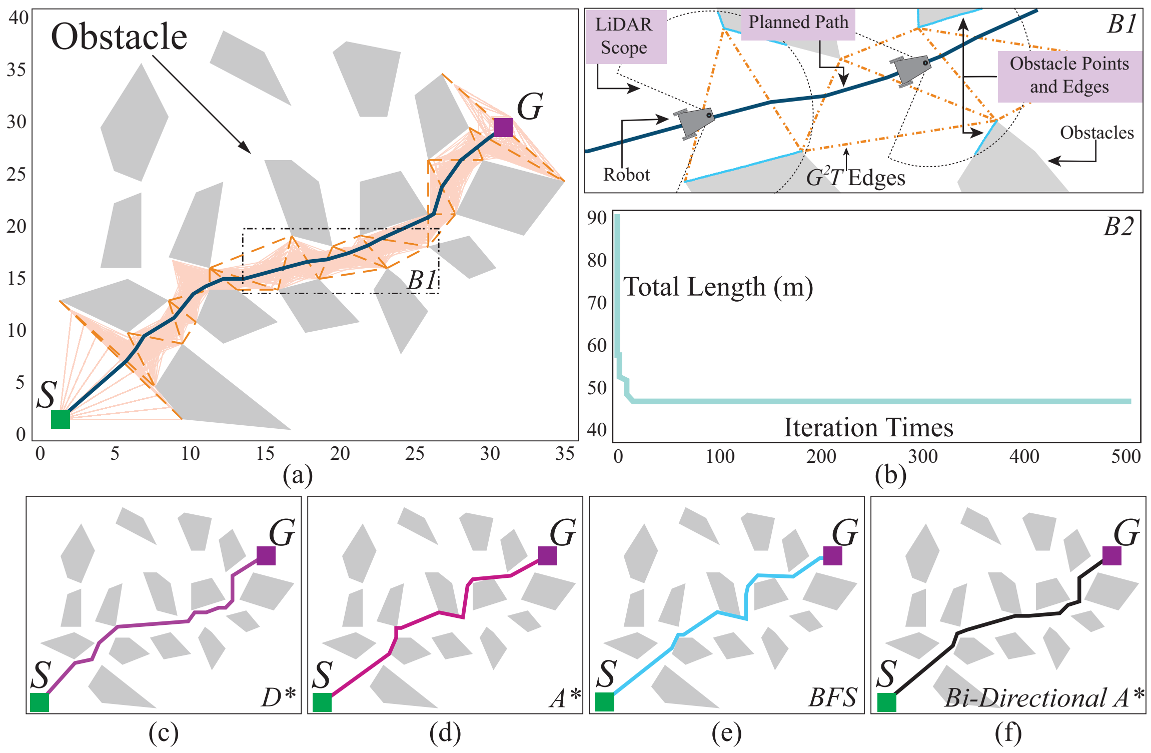


Figure 5. Path planning results for Scenario 1. (**a**) Proposed framework result. (**B1**) Close up view of the proposed path. (**B2**) Convergence behavior of the eACO algorithm. (**c**–**f**) Results from comparison algorithms. Axis units are in meters (m) for all spatial dimensions.

Qualitative analysis of the generated paths provides further insight. As shown in Figure 5c–f, the comparison algorithms tend to produce paths that closely follow obstacle boundaries, resulting in unnecessary detours, particularly around irregular shapes. In contrast, our method (Figure 5a,b) generates smoother trajectories that cut efficiently through large open spaces while maintaining safe clearance from obstacles. This is achieved through the combination of the sparse D^2T graph representation and the eACO's ability to optimize path points continuously along graph edges rather than being constrained to vertices. The convergence behavior of eACO, illustrated in Figure 5(B2), shows the algorithm's efficiency in finding high quality solutions quickly. Similar performance advantages are observed in Scenario 2 (Figure 6), where the proposed method again produces a more direct and efficient path compared to the alternatives.

Table 2. Performance comparison of path planning algorithms. Values represent means from 50 executions. Best results are highlighted in bold.

Scenario	Parameter	BFS	D*	Bi-A*	A*	Proposed
S1	Time (s)	3.37	3.44	7.10	2.59	**2.43**
	Length (m)	49.28	51.52	50.69	50.69	**46.02**
S2	Time (s)	12.32	3.88	5.50	2.65	**2.38**
	Length (m)	48.11	50.35	49.52	49.52	**47.47**

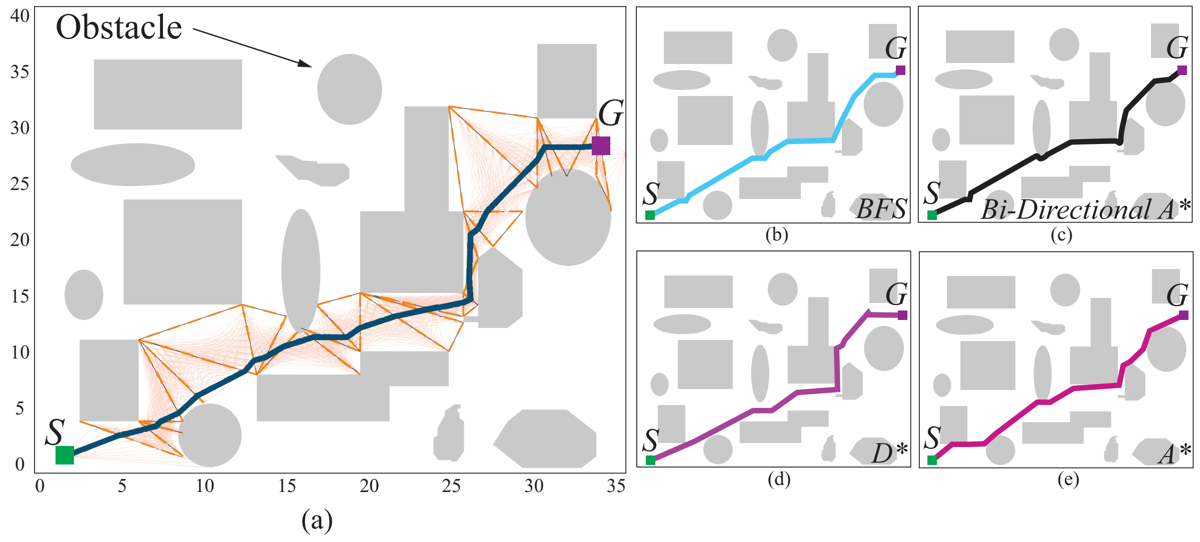


Figure 6. Illustration of the path planning results for Scenario 2. (**a**) Proposed framework result. (**b**–**e**) Comparison algorithm results. Axis units are in meters (m) for all spatial dimensions.

7.2. Computational Complexity Analysis

To quantitatively evaluate the impact of obstacle clustering on computational efficiency, we analyzed the framework's performance in a complex urban environment (Figure 7). Urban settings such as cities, military bases, or college campuses present particular challenges due to their dense and structured obstacle layouts. The clustering process groups proximate obstacles based on spatial proximity, transforming numerous individual obstacles into fewer consolidated convex hulls. This transformation significantly reduces graph complexity in subsequent planning stages. Figure 8 provides a direct comparison between two versions of our framework: one without obstacle clustering and one with the full clustering pipeline. The non clustering version (Figure 8a) must account for every individual obstacle during the D^2T construction, resulting in a dense graph with numerous nodes and edges. In contrast, the clustering version (Figure 8b) generates a substantially sparser graph by treating clustered obstacles as single entities.

The quantitative impact is substantial: the clustering version uses approximately 20 fewer nodes and 35 fewer edges in this scenario, representing a 60% reduction in graph complexity. This reduction directly translates to computational benefits throughout the pipeline: faster D^2T construction, reduced memory requirements, and accelerated eACO optimization due to a smaller search space. These efficiency gains are particularly valuable for real-time applications where computational resources are constrained. In summary, obstacle clustering serves as a critical preprocessing step that enhances the scalability of our framework. By reducing perceptual and computational complexity, it enables efficient navigation in dense urban environments while maintaining the safety and optimality of the generated paths.

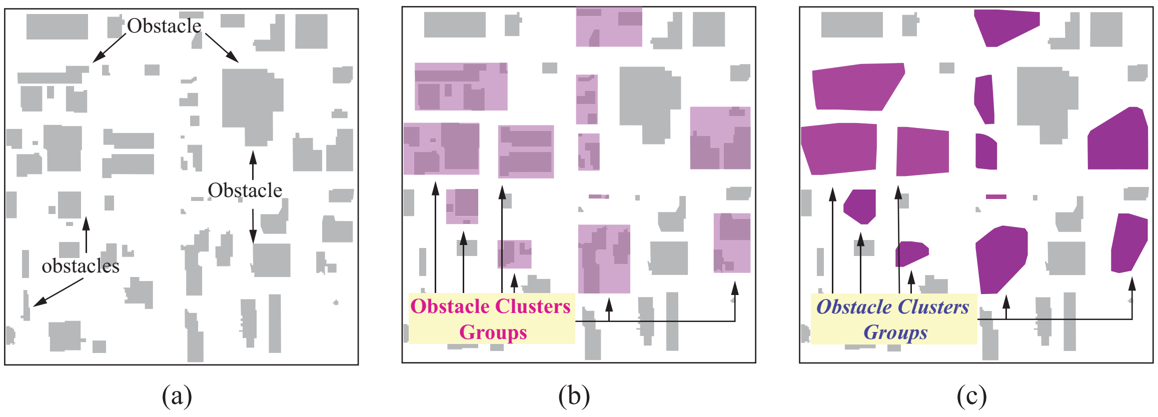


Figure 7. Illustration of the proposed obstacle clustering process in an urban environment. (**a**) Original environment with individual obstacles. (**b**) Intermediate clustering stage. (**c**) Final environment after clustering, where purple polygons represent merged convex obstacles.The map dimensions are 30 m × 30 m, and all spatial measurements are in meters (m).

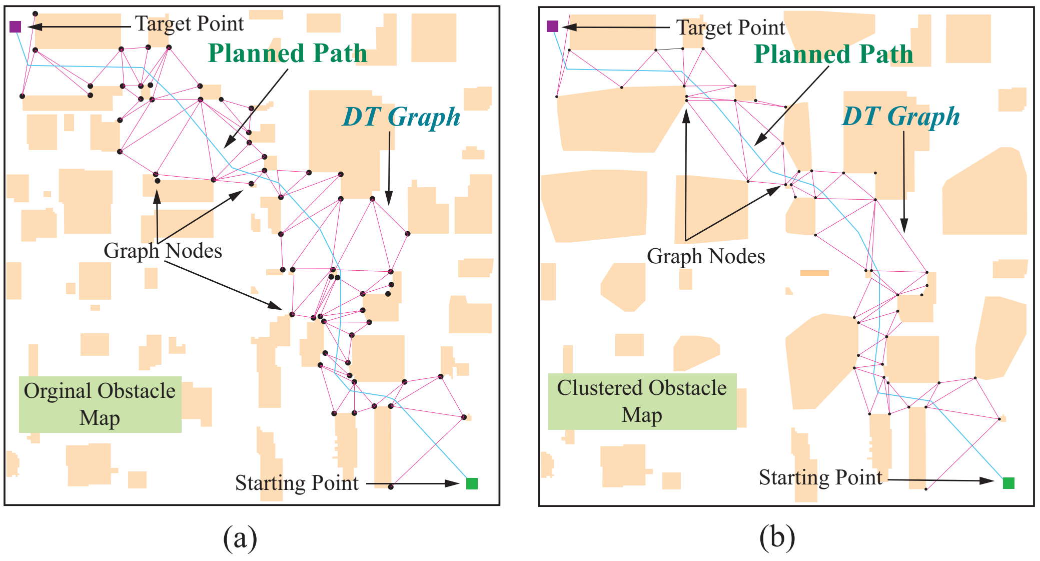


Figure 8. Comparison results of graph complexity with and without obstacle clustering. (**a**) Non clustering version produces a dense graph with high edge count. (**b**) Clustering version generates a sparse graph with significantly fewer edges while maintaining navigability. The map dimensions are 30 m × 30 m, and all spatial measurements are in meters (m).

7.3. Path Planning in Dynamic Real-World Environments

To validate the complete framework in a realistic dynamic setting, we tested our system in a simulated beach environment where an autonomous robot must navigate through a crowd to perform a trash collection task (Figure 9). This scenario presents key challenges: numerous static obstacles (beach equipment, umbrellas) and dynamic obstacles (moving pedestrians) in an unstructured environment.

The environmental processing pipeline begins with obstacle clustering, as shown in Figure 10. The clustering algorithm reduced the number of distinct obstacles from 28 to 9, significantly simplifying the environment representation. The D^2T graph was then constructed over this simplified representation, generating an initial global path (blue line in Figure 10b).

The critical test occurs within the black bordered region in Figure 10b, where two dynamic obstacles (pedestrians) intersect the robot's planned path. Figure 11 demonstrates the local reactive navigator's response to this situation. As the first pedestrian approaches, the robot proactively reduces its velocity and initiates a smooth deviation, maintaining approximately 1.2 m clearance while preserving forward progress. For the second pedestrian, the robot executes a smaller deviation maneuver, leveraging its reduced speed from the first avoidance to make a more efficient adjustment. After clearing both obstacles, the robot smoothly transitions back to the global path without oscillatory behavior or significant delay.

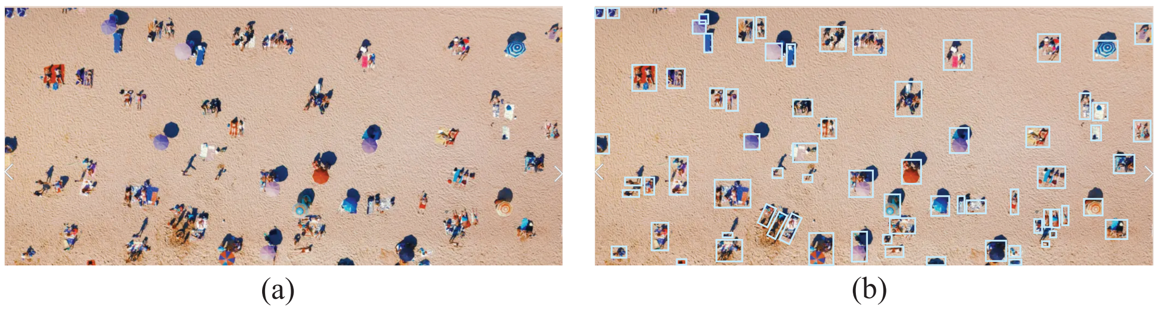


Figure 9. Illustration of the beach navigation scenario. (**a**) Original environment with static and dynamic obstacles. (**b**) Extracted obstacle boundaries for processing. The map dimensions are 50 m × 100 m, and all spatial measurements are in meters (m).

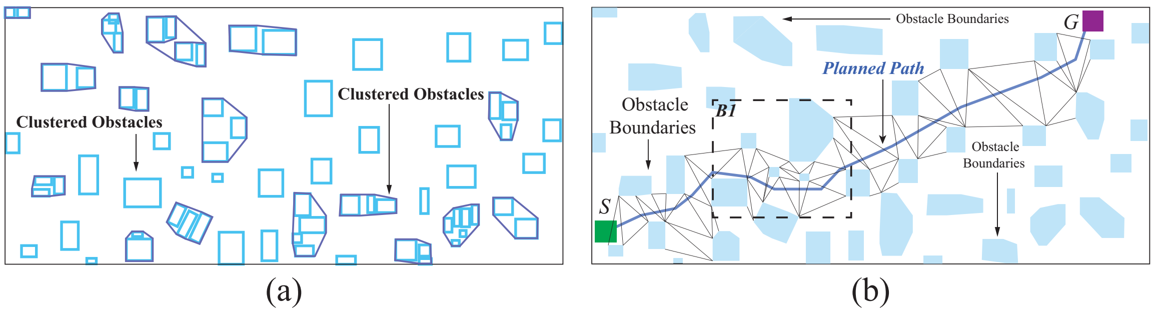


Figure 10. Illustration of the environmental processing and global planning. (**a**) Obstacle clustering results with merged obstacles outlined in blue. (**b**) D^2T graph construction and global path planning (blue line) in the simplified environment.

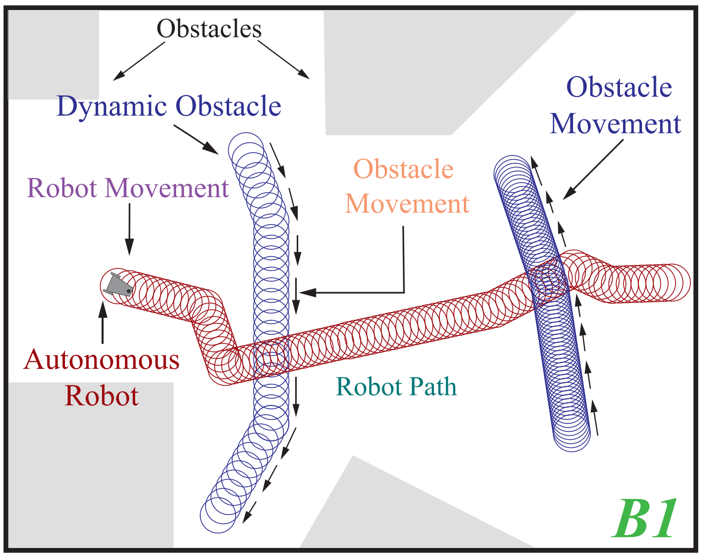


Figure 11. Illustration of the velocity-based local reactive navigation response. The environment is from Figure 10b. The robot successfully avoids two dynamic obstacles while maintaining smooth trajectory and safe clearance distances.

The local reactive navigator's velocity based approach enables several advantages in this scenario: it maintains safety through proactive speed adjustments, ensures passenger comfort through smooth trajectory changes, and preserves efficiency by minimizing deviations from the global plan. The integration of global planning (handling complex static environments) with local reaction (handling dynamic obstacles) demonstrates the framework's robustness for real-world deployment. These results confirm that our hierarchical approach combining environmental simplification through clustering, optimal global planning via eACO on D^2T graphs, and reactive local navigation using velocity obstacles provides a comprehensive solution for autonomous navigation in complex, dynamic environments.

8. Conclusions

This paper has presented a comprehensive framework for autonomous navigation that addresses critical challenges in environment mapping, global path planning, and dynamic obstacle avoidance. Our approach integrates several key innovations: a D^2T for sparse environmental representation, an obstacle clustering method for reducing complexity, an eACO algorithm for continuous path optimization, and a velocity-based local reactive navigator for real-time reaction to dynamic obstacles. Simulation studies across multiple scenarios demonstrate that our framework achieves superior performance compared to conventional graph-based planners. The results show significant improvements in both path quality producing shorter, smoother trajectories and computational efficiency, particularly in complex environments with irregular obstacles. The hierarchical structure of our approach successfully balances global optimality with local reactivity, enabling robust performance in both static and dynamic settings.

Despite these advantages, our study has a key limitation: the comparative analysis has focused on classical graph-based planners. A comparison with modern sampling-based techniques (e.g., RRT*) and learning based methods is essential to fully establish the relative performance and applicability of our approach. Therefore, our immediate future work will include a comprehensive benchmarking study against these state of the art planners across a wider range of dynamic and complex environments. Looking forward, we will also pursue several other research directions. First, we plan to extend the framework to fully three dimensional environments for applications in aerial or underwater robotics. Second, we aim to implement and validate the system on physical robotic platforms to assess its performance under real-world constraints like sensor noise and computational limits. Third, we will explore the integration of learning based methods to adaptively tune the parameters of the eACO algorithm and the local reactive navigator based on environmental characteristics. Finally, we intend to validate the framework in more complex, interactive scenarios such as simulated urban intersections and densely crowded environments, where dynamic obstacles exhibit non linear motion patterns, to provide a more rigorous assessment of the system's capabilities.

Author Contributions

Methodology, T.S., T.L. and C.L.; conceptualization, T.L. and C.L.; software, T.S. and T.L.; validation, T.S. and T.L.; formal analysis, T.L., T.S. and C.L.; investigation, T.L., T.S. and C.L.; resources, C.L.; data curation, T.L. and T.S.; writing—original draft preparation, T.L., T.S. and C.L.,; writing—review and editing, T.S., T.L., C.L., Z.B. and G.E.J.; supervision, C.L., Z.B. and G.E.J.; project administration, C.L. All authors have read and agreed to the published version of the manuscript.

Funding

This research was partially supported by the Mississippi Space Grant Consortium under NASA EPSCoR grant.

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest

Use of AI and AI-Assisted Technologies

No AI tools were utilized for this paper.

References

1. Jan, G.E.; Lei, T.; Sun, C.C.; et al. On the problems of drone formation and light shows. *IEEE Trans. Consum. Electron.* **2024**, *70*, 5259–5268.
2. Lei, T.; Li, G.; Luo, C.; et al. An informative planning-based multi-layer robot navigation system as applied in a poultry barn. *Intell. Robot.* **2022**, *2*, 313–332.
3. Sellers, T.; Lei, T.; Luo, C.; et al. Enhancing human-robot cohesion through hat methods: A crowd-avoidance model for safety aware navigation. In Proceedings of the 2024 IEEE 4th International Conference on Human-Machine Systems (ICHMS), Toronto, ON, Canada, 15–17 May 2024; pp. 1–6.

4. Lei, T.; Chintam, P.; Luo, C.; et al. A convex optimization approach to multi-robot task allocation and path planning. *Sensors* **2023**, *23*, 5103.
5. Hicks, M.; Lei, T.; Luo, C.; et al. A Bio-Inspired Goal-Directed Cognitive Map Approach to Robot Navigation and Mapping. In Proceedings of the 2025 IEEE Congress on Evolutionary Computation (CEC), Hangzhou, China, 8–12 June 2025; pp. 1–8.
6. Riser, E.; Sellers, T.; Lei, T.; et al. Multirobot navigation using improved RRT*-SMART with digital twin technology. In Proceedings of the SPIE Conference: Autonomous Systems: Sensors, Processing, and Security for Ground, Air, Sea, and Space Vehicles and Infrastructure 2024, National Harbor, MD, USA, 21–26 April 2024; Volume 13052, pp. 114–125.
7. Lei, T.; Luo, C.; Sellers, T.; et al. Multitask allocation framework with spatial dislocation collision avoidance for multiple aerial robots. *IEEE Trans. Aerosp. Electron. Syst.* **2022**, *58*, 5129–5140.
8. Sellers, T.; Lei, T.; Carruth, D.; et al. Deep Learning-Based Heterogeneous System for Autonomous Navigation. In Proceedings of the SPIE, San Diego, CA, USA, 20–25 August 2023; Volume 12539, pp. 140–153.
9. Black, B.; Sellers, T.; Lei, T.; et al. Optimal multi-target navigation via graph-based algorithms in complex environments. In Proceedings of the 2024 IEEE 33rd International Symposium on Industrial Electronics (ISIE), Ulsan, Republic of Korea, 18–21 June 2024; pp. 1–6.
10. Sellers, T.; Lei, T.; Luo, C.; et al. A node selection algorithm to graph-based multi-waypoint optimization navigation and mapping. *Intell. Robot.* **2022**, *2*, 333–54.
11. Rogers, J.H., III; Sellers, T.; Lei, T.; et al. Centroid-based cell decomposition robot path planning algorithm integrated with a bio-inspired approach. In Proceedings of the SPIE Conference: Unmanned Systems Technology XXVI. SPIE, National Harbor, MD, USA, 23–25 April 2024; Volume 13055, pp. 43–51.
12. Sellers, T.; Lei, T.; Luo, C.; et al. Human autonomy teaming-based safety-aware navigation through bio-inspired and graph-based algorithms. *Biomim. Intell. Robot.* **2024**, *4*, 100189.
13. Lei, T.; Chintam, P.; Carruth, D.W.; et al. Human-autonomy teaming-based robot informative path planning and mapping algorithms with tree search mechanism. In Proceedings of the 2022 IEEE 3rd International Conference on Human-Machine Systems (ICHMS), Orlando, FL, USA, 17–19 November 2022; pp. 1–6.
14. Mandalika, A.; Salzman, O.; Srinivasa, S. Lazy Receding Horizon A* for Efficient Path Planning in Graphs with Expensive-to-Evaluate Edges. *Proc. Int. Conf. Autom. Plan. Sched.* **2018**, *28*, 476–484.
15. Maurovic, I.; Seder, M.; Lenac, K.; et al. Path Planning for Active SLAM Based on the D* Algorithm With Negative Edge Weights. *IEEE Trans. Syst. Man Cybern. Syst.* **2017**, *48*, 1321–1331.
16. Lei, T.; Luo, C.; Jan, G.E.; et al. Variable Speed Robot Navigation by an ACO Approach. In *International Conference on Swarm Intelligence (ICSI)*; Springer: Cham, Switzerland, 2019; pp. 232–242.
17. Scheffe, P.; Pedrosa, M.V.A.; Flaßkamp, K.; et al. Receding Horizon Control Using Graph Search for Multi-Agent Trajectory Planning. *IEEE Trans. Control Syst. Technol.* **2022**, *31*, 673–688.
18. Lei, T.; Luo, C.; Ball, J.E.; et al. A Hybrid Fireworks Algorithm to Navigation and Mapping. In *Handbook of Research on Fireworks Algorithms and Swarm Intelligence*; IGI Global: Hershey, PA, USA, 2020; pp. 213–232.
19. Graf, U.; Borges, P.; Hernández, E.; et al. Optimization-Based Terrain Analysis and Path Planning in Unstructured Environments. In Proceedings of the 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 5614–5620.
20. Niewola, A.; Podsedkowski, L. L* Algorithm—A Linear Computational Complexity Graph Searching Algorithm for Path Planning. *J. Intell. Robot. Syst.* **2018**, *91*, 425–444.
21. Lei, T.; Chintam, P.; Luo, C.; et al. Multi-Robot directed coverage path planning in row-based environments. In Proceedings of the 2022 IEEE Fifth International Conference on Artificial Intelligence and Knowledge Engineering (AIKE), Laguna Hills, CA, USA, 19–21 September 2022; pp. 114–121.
22. Bhattacharya, S.; Ghrist, R.; Kumar, V. Persistent Homology for Path Planning in Uncertain Environments. *IEEE Trans. Robot.* **2015**, *31*, 578–590.
23. Lei, T.; Luo, C.; Sellers, T.; et al. A Bat-Pigeon Algorithm to Crack Detection-Enabled Autonomous Vehicle Navigation and Mapping. *Intell. Syst. Appl.* **2021**, *12*, 200053.
24. Alarabi, S.; Lei, T.; Santora, M.; et al. Multi-robot path planning using potential field-based simulated annealing approach. In Proceedings of the SPIE Conference: Unmanned Systems Technology XXVI, National Harbor, MD, USA, 21–26 April 2024; Volume 13055, pp. 102–117.
25. Chintam, P.; Lei, T.; Osmanoglu, B.; et al. Informed sampling space driven robot informative path planning. *Robot. Auton. Syst.* **2024**, *175*, 104656.
26. Lei, T.; Sellers, T.; Luo, C.; et al. Digital twin-based multi-objective autonomous vehicle navigation approach as applied in infrastructure construction. *IET Cyber-Syst. Robot.* **2024**, *6*, e12110.
27. Sellers, T.; Lei, T.; Jan, G.E.; et al. Multi-Objective Optimization Robot Navigation Through a Graph-Driven PSO Mechanism. In *International Conference on Sensing and Imaging*; Springer: Cham, Switzerland, 2022; pp. 66–77.
28. Short, D.; Lei, T.; Carruth, D.W.; et al. A bio-inspired algorithm in image-based path planning and localization using visual features and maps. *Intell. Robot.* **2023**, *3*, 222–41.

29. Hicks, M.; Lei, T.; Luo, C.; et al. A bio-inspired goal-directed cognitive map model for robot navigation and exploration. *IEEE Trans. Cogn. Dev. Syst.* **2025**, *17*, 1125–1140.
30. Steen, S.; Lei, T.; Luo, C.; et al. A Moss Growth Optimization Approach to Robot Path Planning. In Proceedings of the S2025 IEEE Congress on Evolutionary Computation (CEC), Hangzhou, China, 8–12 June 2025; pp. 1–4.
31. Lei, T.; Luo, C.; Yang, S.X.; et al. Bio-inspired Intelligence-based Multi-agent Navigation with Safety-aware Considerations. *IEEE Trans. Artif. Intell.* **2024**, *5*, 2946–2961.
32. Rogers, J.H.; Sellers, T.; Lei, T.; et al. Sensor-based multi-waypoint autonomous robot navigation with graph-based models. In Proceedings of the SPIE Conference: Autonomous Systems: Sensors, Processing and Security for Ground, Air, Sea, and Space Vehicles and Infrastructure 2023, Orlando, FL, USA, 30 April–5 May 2023; Volume 12540, pp. 215–224.
33. Sellers, T.; Lei, T.; Rogers, H.; et al. Autonomous Multi-Robot Allocation and Formation Control for Remote Sensing in Environmental Exploration. In Proceedings of the SPIE, Orlando, FL, USA, 30 April–5 May 2023; Volume 12540, pp. 225–241.
34. Short, D.L.; Lei, T.; Liu, L.; et al. A Neural Network Approach to Image-Based Navigation and Localization. In Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN), Rome, Italy, 30 June–5 July 2025; pp. 1–7.
35. Jayaraman, E.; Lei, T.; Rahimi, S.; et al. Immune System Algorithms to Environmental Exploration of Robot Navigation and Mapping. In *Advances in Swarm Intelligence, Proceedings of the 12th International Conference, ICSI 2021, Qingdao, China, 17–21 July 2021*; Springer: Cham, Switzerland, 2021; pp. 73–84.
36. Lei, T.; Sellers, T.; Rahimi, S.; et al. A Nature-Inspired Algorithm to Adaptively Safe Navigation of a COVID-19 Disinfection Robot. In *International Conference on Intelligent Robotics and Application(ICIRA)*; Springer International Publishing: Cham, Switzerland, 2021; pp. 123–134.
37. Chen, X.; Li, S.; Kumar, V. Deep reinforcement learning for dynamic obstacle avoidance in unstructured environments. *Int. J. Robot. Res.* **2023**, *42*, 299–317.
38. Zhang, Y.; Pavone, M.; Fisac, J.F. GNN-based predictive navigation in dynamic scenes using spatiotemporal graphs. In Proceedings of the Robotics: Science and Systems (RSS), Delft, The Netherlands, 15–19 July 2024.
39. Liu, W.; Zhang, H.; Chen, Y.; et al. Graph-based reinforcement learning for autonomous navigation in dynamic environments. *IEEE Robot. Autom. Lett.* **2023**, *8*, 2125–2132.
40. Wang, J.; Singh, A.; Dolan, J.M. Transformer-based path planning for autonomous vehicles in crowded environments. In Proceedings of the 2024 IEEE International Conference on Robotics and Automation (ICRA), Yokohama, Japan, 13–17 May 2024; pp. 1–7.
41. Lei, T.; Sellers, T.; Luo, C.; et al. Graph-based robot optimal path planning with bio-inspired algorithms. *Biomim. Intell. Robot.* **2023**, *3*, 100119.
42. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894.
43. Manandhar, S.K. Efficient Algorithms for Clustering Polygonal Obstacles. Efficient Algorithms for Clustering Polygonal Obstacles. UNLV Theses, Dissertations, Professional Papers, and Capstones. 2016, 2704. http://dx.doi.org/10.34917/9112138
44. Berg, M.; Cheong, O.; Kreveld, M.; et al. *Computational Geometry: Algorithms and Applications*, 3rd ed.; Springer: Berlin, Germany, 2008.
45. Mantegna, R.N. Fast, accurate algorithm for numerical simulation of Levy stable stochastic processes. *Phys. Rev. E* **1994**, *49*, 4677.
46. Lei, T.; Luo, C.; Jan, G.E.; et al. Deep Learning-Based Complete Coverage Path Planning With Re-Joint and Obstacle Fusion Paradigm. *Front. Robot. AI* **2022**, *9*, 843816.
47. Dorigo, M.; Stützle, T. The Ant Colony Optimization Metaheuristic: Algorithms, Applications, and Advances. *Handb. Metaheuristics* **2006**, *57*, 227–263.