Target Rescue

Robot Planning and its application, Master's Degree in Artificial Intelligence Systems

Author: Jonathan Fin
ID and e-mail: 256178, jonathan.fin@studenti.unitn.it

Academic year: 2025-2026

Contents

1. Introduction

Autonomous mobile robots operating in disaster response scenarios must navigate complex environments while optimizing mission objectives under strict time constraints. This report addresses the **Target Rescue** problem: a single robot must locate and rescue victims within a bounded environment, maximizing the total value of rescued targets while reaching an exit gate before a mission timeout expires.

The problem presents several algorithmic challenges. First, the robot's **nonholonomic constraints** (constant velocity $v = 1$ m/s, minimum turning radius $\rho = 1$ m) require smooth, curvature-bounded trajectories that cannot be generated using standard shortest-path algorithms. Second, selecting which victims to rescue and in what order constitutes an **Orienteering Problem** - a variant of the Prize-Collecting Traveling Salesman Problem known to be NP-hard. Third, paths must avoid static obstacles while maintaining safety margins to account for controller tracking errors. Finally, the entire planning pipeline must execute efficiently enough to leave sufficient time for trajectory execution within the timeout window.

This work implements and compares two complementary approaches: a **combinato-**

rial method using Approximate Cell Decomposition (ACD) with A* pathfinding, and a **sampling-based** method using Informed RRT* for collision-free waypoint generation. Both approaches employ Dubins curves for kinematically feasible path generation and dynamic programming for optimal victim selection.
The code is available in this Git Repo, in the `planning_project` folder.

2. Map Representation

The planning system receives environmental data from multiple ROS topics: world boundaries, static obstacles, target locations with associated values, goal pose, and positions of other robots, if present. These are combined into a unified collision map for path planning.
To treat the robot as a point mass, all obstacles are **inflated** by a distance equal to $r_{\text{robot}} + m_{\text{safety}}$, where r_{robot} is the robot's radius and m_{safety} is an additional safety margin, to compensate for possible drifts in the controller. Similarly, the map borders are **shrunk** inward by the same amount. This geometric preprocessing allows all subsequent collision checks to be performed as point-in-polygon tests, significantly simplifying the planning algorithms.
When other robots are present in the environment (such as in the `multiple_robots` configuration), they are modeled as static cylindrical obstacles with radius equal to r_{robot}.
Two distinct map representation approaches were implemented and compared: a **combinatorial** grid-based decomposition and a **sampling-based** approach using Informed RRT*.

2.1. Combinatorial Approach

Combinatorial planning methods discretize the continuous configuration space into a finite graph of feasible states. **Approximate Cell Decomposition** (ACD) was employed, which recursively subdivides the workspace into square cells based on occupancy, using an initial resolution of $\Delta = 0.5m$ and recursively subdividing `MIXED` cells into four equal quadrants (quadtree structure) up to depth $d_{max} = 3$. In the end a connectivity graph is build by linking adjacent `FREE` cells.
This produces a non-uniform spatial discretization with fine resolution near obstacle boundaries and coarse resolution in open space. The resulting graph is used for A* pathfinding during the orienteering and Dubins path planning stages.

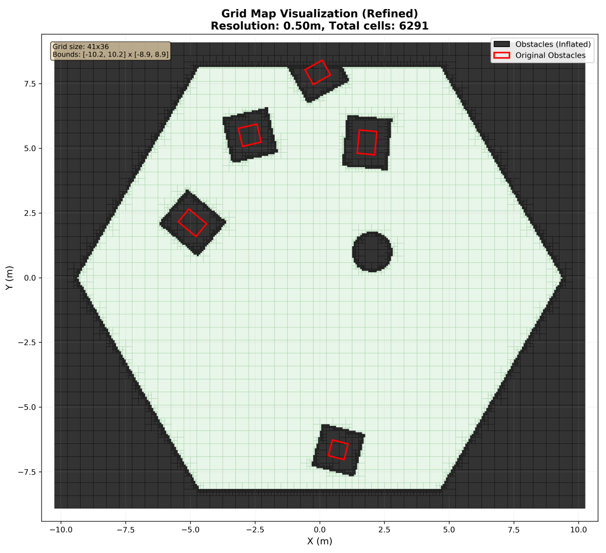

Figure 1: Approximate cell decomposition map. Green cells indicate free space, black cells indicate occupied regions. Finer subdivision is visible near obstacle boundaries (quadtree depth up to 3).

The ACD approach guarantees **resolution completeness**: if a path exists at the chosen grid resolution, it will be found. However, performance depends critically on the grid resolution Δ and refinement depth d_{max}.

2.2. Sampling-Based Approach

Sampling-based planning avoids explicit spatial decomposition by randomly sampling the configuration space and building a connectivity graph incrementally. **Informed RRT*** (Rapidly-exploring Random Tree Star with informed sampling) was used in this project, an asymptotically optimal variant of RRT*.
This implementation uses 3000 maximum iterations per segment and a step size $\delta = 0.4m$, with the following features:

- **Informed sampling**: After finding an initial solution, samples are drawn from an ellipsoidal region containing only points that could improve the current best path (based on Euclidean heuristic).
- **Dynamic rewiring**: Nodes are reconnected to minimize path cost whenever a lower-cost parent is found within a neighborhood radius.

- **Collision checking**: Line segments are validated against the inflated obstacle polygons using geometric intersection tests.

RRT* is applied **segment-by-segment** between consecutive waypoints. If a direct Dubins curve between two waypoints is collision-free, RRT* is skipped for that segment. Otherwise, the raw RRT* path is smoothed using shortcut optimization to reduce unnecessary detours.

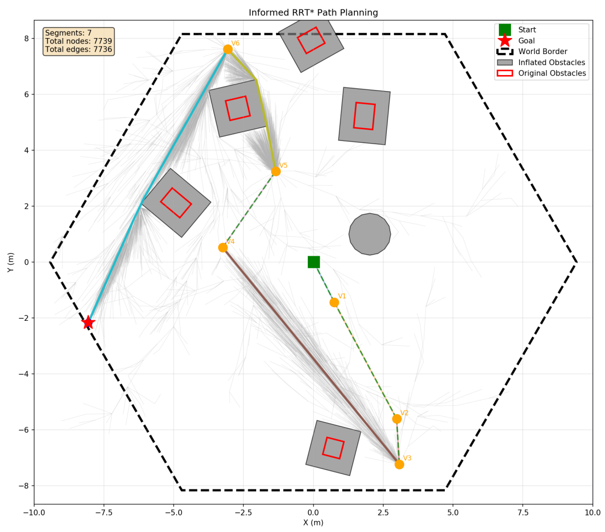

Figure 2: RRT* exploration tree for a collision-obstructed segment.
Light gray edges show the sampling tree, colored lines show the raw and smoothed paths. Direct segments (no obstacles) bypass RRT* entirely.

Compared to ACD, RRT* offers **probabilistic completeness** and avoids the memory overhead of dense grids, but requires more computation for each query. The informed sampling heuristic significantly accelerates convergence compared to standard RRT*.

3. Path Planning

After selecting which victims to visit (solved via the Orienteering Problem, described in Section 4), a feasible trajectory must be generated that respects the robot's **nonholonomic constraints**. The LIMO robot operates at constant velocity $v = 1\ m/s$ and has a minimum turning radius $\rho = 1m$, preventing instantaneous direction changes and requiring smooth, curvature-bounded paths.

3.1. Dubins Curves

Dubins curves represent the shortest paths between two poses (position + orientation) for a vehicle with bounded curvature. Each curve is composed of at most three segments: two circular arcs (Left or Right) and one straight line (Straight), yielding six possible types: `LSL`, `RSR`, `LSR`, `RSL`, `RLR`, and `LRL`.

Given start pose (x_0, y_0, θ_0), goal pose (x_g, y_g, θ_g), and maximum curvature $k_{\max} = 1/\rho$, the optimal Dubins path is computed by evaluating all six types, sorting them by arc length, and selecting the shortest collision-free path. This ensures both kinematic feasibility and obstacle avoidance.

3.2. Multi-Waypoint Dubins Path

The full route consists of multiple segments: start $\rightarrow$ victim$_1$ $\rightarrow$ victim$_2$ $\rightarrow \cdots \rightarrow$ gate. A **multi-point Dubins path** is constructed by chaining individual Dubins curves while optimizing intermediate waypoint orientations.

For each waypoint, the algorithm discretizes possible approach/departure angles (initially in steps of $\pi/4$ radians) and selects the combination that minimizes total path length while maintaining collision-free arcs. This angle optimization is refined iteratively to balance computational cost with solution quality.

The combinatorial approach uses A* on the cell graph to generate intermediate waypoints, while the sampling-based approach uses RRT*-derived smoothed paths as waypoints (as described in Section 2.2). In both cases, waypoints are connected via Dubins curves.

3.3. Collision Checking and Adaptive Safety Margin

Each Dubins arc is validated for collision by sampling points along the curve at fixed intervals (≈ 0.05 m) and testing against inflated obstacle polygons. If any arc collides, the path is rejected.

Rather than immediately failing when the shortest Dubins curve collides with obstacles, the planner evaluates all six Dubins types (LSL, RSR, LSR, RSL, RLR, LRL), sorts them by arc length, and iteratively tests each candidate until finding the shortest collision-free alternative. This significantly improves robustness in cluttered environments where the globally shortest path may be obstructed, but longer alternatives remain feasible.

To account for controller tracking errors and model uncertainties, an **adaptive safety margin retry mechanism** was implemented:

1. **Initial attempt**: Start with maximum safety margin $m_{\text{safety}} = 0.25$ m (100% of robot radius). Obstacles are heavily inflated, providing conservative clearance.
2. **Retry on failure**: If no feasible path exists (either due to orienteering infeasibility or Dubins collision), reduce m_{safety} by 0.05 m and rebuild the map, distance matrix, and path.
3. **Termination**: Retry until $m_{\text{safety}} = 0$ or a valid path is found.

This approach prioritizes safety (large margins), balancing robustness with feasibility.

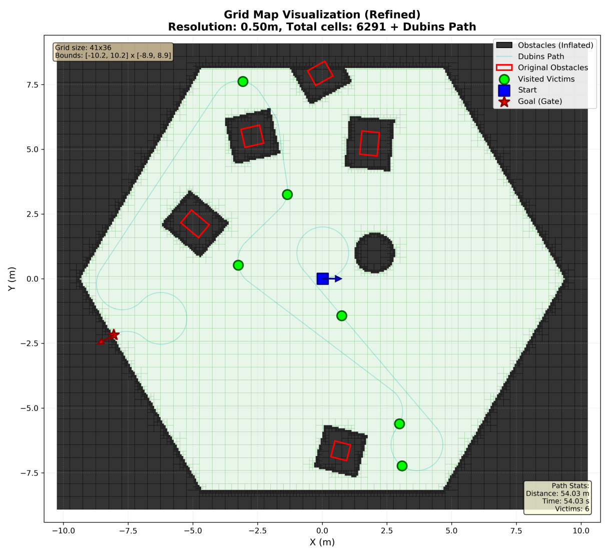

Figure 3: Dubins path
Blue line shows the final trajectory connecting start (blue square), visited victims (green circles), and gate (red star)

4. Orienteering Problem

After constructing the collision map and before generating the final trajectory, the planner must decide **which victims to rescue** and **in what order**. This is formalized as the Orienteering Problem (OP), a variant of the Prize-Collecting Traveling Salesman Problem (PCTSP). The objective is to maximize the total value of rescued victims while ensuring the robot reaches the gate within the given time limit.

4.1. Problem Formulation

The orienteering problem takes as input a set of victims (each with a position and associated value), a distance matrix encoding travel costs between all points (start, victims, and gate), and a distance budget derived from the mission timeout. The goal is to select a subset of victims to visit and determine their visiting order such that the total value is maximized while the total path length remains within budget.

The robot *must* reach the gate. If the gate is unreachable within the budget, the mission fails entirely, regardless of how many victims were rescued. This enforces a strict feasibility requirement on any candidate solution.

The Orienteering Problem is **NP-hard**, as it combines subset selection (exponential in the number of victims) with the Traveling Salesman Problem (finding the optimal visiting order).

4.2. Distance Matrix Computation

The distance matrix encodes the cost of traveling between any two nodes (start, victims, or gate). Two methods were implemented depending on the map representation approach:

Combinatorial approach: A* pathfinding is executed on the cell graph for all pairs of nodes. This produces **exact** shortest path distances that account for obstacles and map topology. The search uses Euclidean distance as the heuristic, ensuring admissibility and optimality. Since victims do not have predefined orientations (only the start and gate have fixed headings), the distance computation conservatively estimates Dubins curve costs by multiplying Euclidean distance by a heuristic factor (typically 1.2–1.4) to account for curvature penalties.

Sampling-based approach: Direct A* is not applicable since no cell graph exists. Instead, distances are approximated using the **Euclidean heuristic** scaled by the same factor. While less accurate, this approach is computationally cheaper and sufficient for orienteering feasibility checks. The actual path planning via RRT* (described in Section 3) refines these estimates during execution.

The distance matrix is precomputed once after the map is built. For typical problem sizes (up to 10 victims), this overhead is negligible.

4.3. Dynamic Programming

Several algorithms exist for solving the Orienteering Problem:

- **Greedy heuristics**: Iteratively select the victim with the best value-to-distance ratio. Fast ($O(N^2)$), but suboptimal and may miss high-value solutions.
- **Branch-and-bound**: Explores the solution space with pruning. Can find optimal solutions, but worst-case complexity remains exponential.
- **Dynamic programming**: Enumerates all victim subsets and solves TSP for each. Guarantees optimality with $O(2^N \cdot N^2)$ complexity.

Given the small problem size (typical scenarios involve 3–5 victims), an **exact dynamic programming approach** was implemented to guarantee optimal solutions. The algorithm proceeds as follows:

1. **Enumerate subsets**: Generate all 2^N subsets of victims.
2. **Solve TSP per subset**: For each subset, compute the shortest path from start through all victims in that subset to the gate.
3. **Filter feasible solutions**: Discard any subset whose minimum path length exceeds the budget.
4. **Select maximum value**: Among feasible solutions, choose the subset with maximum total victim value.

4.4. Time Budget Management

The mission timeout T (in seconds) is provided via the ROS topic `/victims_timeout`. To account for planning overhead and controller tracking delays, a **safety buffer** is reserved. The effective distance budget is computed as:

$$B = (1 - r_{\text{buffer}}) \cdot T \cdot v$$

where $r_{\text{buffer}} = 0.15$ (15% reserve) and $v = 1$ m/s is the constant robot velocity. For example, with $T = 120$ seconds, the effective budget is $B = 0.85 \times 120 \times 1 = 102$ meters.

This conservative budgeting ensures the robot has sufficient time to complete map construction, solve the orienteering problem, generate the Dubins path, and execute the trajectory without exceeding the deadline.

If no timeout is specified, the budget is set to infinity, allowing the planner to visit all victims regardless of distance.

5. Experimental Results

The original environment map used in the tests of this report is shown in Figure 4

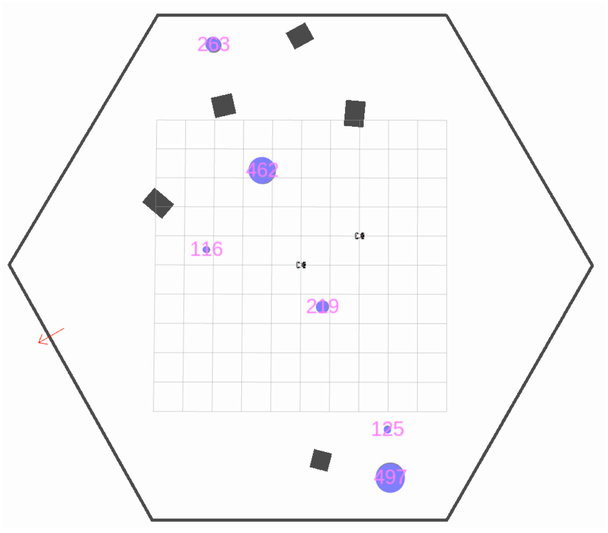

Figure 4: Original Map

The tests were first conducted on an infinite time budget, to ensure correct path generation and tracking. Execution times were also monitored to get a benchmark of the reasoning time.

Combinatorial Approach

The solution path found in this approach is visible in Figure 3, while the time taken to plan is available in Table 1.

Task	**Time taken** (ms)
Map building	96.89
Orienteering	0.09
Path generation	166.38
TOTAL	**297.56**

Table 1: Computational Time of Combinatorial approach

The 15% time budget is probably too strict in this scenario and could be lowered to have a higher total score in rescuing victims, but it will be needed in the sampling-based approach.

The path tracking was also verified using the provided controller and shows good tracking performance also on many curves, as shown in figure 5.

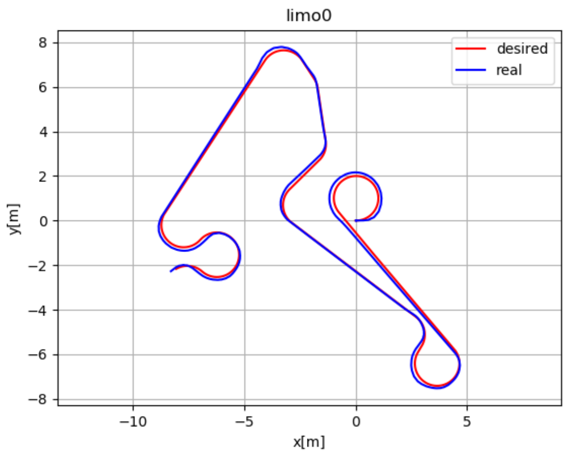

Figure 5: Trajectory tracking

Sampling-based approach

The Dubins path found using this approach can be seen in Figure 6. It produces a smoother and shorter solutions (47.19m against 54.03m of the combinatorial approach), but at the cost of more computation time, as shown in Table 2

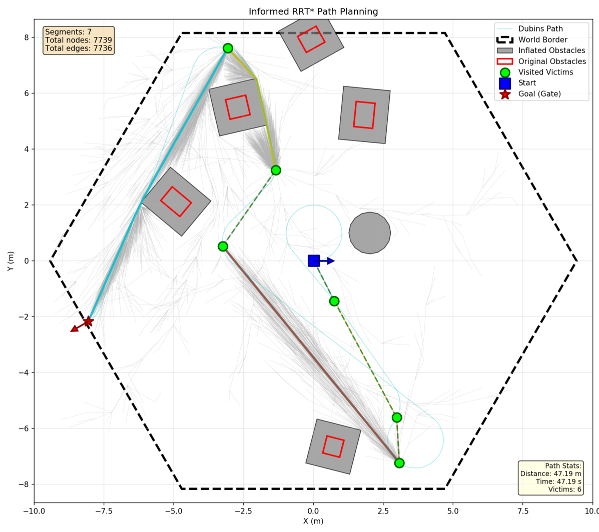

Figure 6: Dubins path using RRT*

The main problem in the sampling-based approach appears especially when no path is found with a large safety margin. If the algorithm needs to recalculate everything with a lower margin, it doubles the time spent reasoning, making real-time replanning challenging.

Task	**Time taken** (ms)
Map building	0.01
Orienteering	0.15
Path generation	2692.76
TOTAL	**2699.97**

Table 2: Computational Time of Sampling-based approach

5.1. Time constraints

Here are also reported the paths chosen for the time constraint cases, using the same scenario with different timeouts: 60 seconds (Figure 7), 45 seconds (Figure 8) and 30 seconds (Figure 9). As the timeout decreases, the orienteering solver correctly prioritizes high-value victims close to the optimal path. For instance, with 30s budget, 3 victims are skipped in favor of the other 3 victims which lie along a more direct route to the gate.
Exact time spent rescuing victims and exiting the map is shown in the lower right corner of each figure, and also reported in Table 3 for simplicity. Reasoning time is $\sim 300ms$ in every scenario, and is included in the table.

Limit	**Time taken**	**Victims Rescued**
60 s	40.77 s	5 (1419 pts)
45 s	36.75 s	4 (1060 pts)
30 s	19.93 s	3 (797 pts)

Table 3: Time budget results

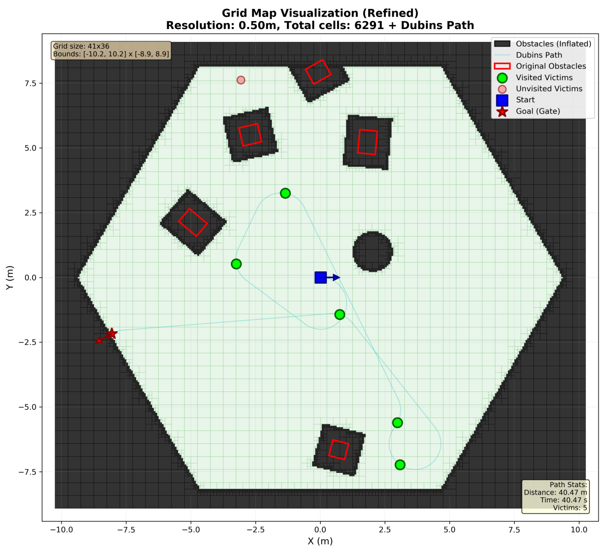

Figure 7: Path with a time budget of $60s$

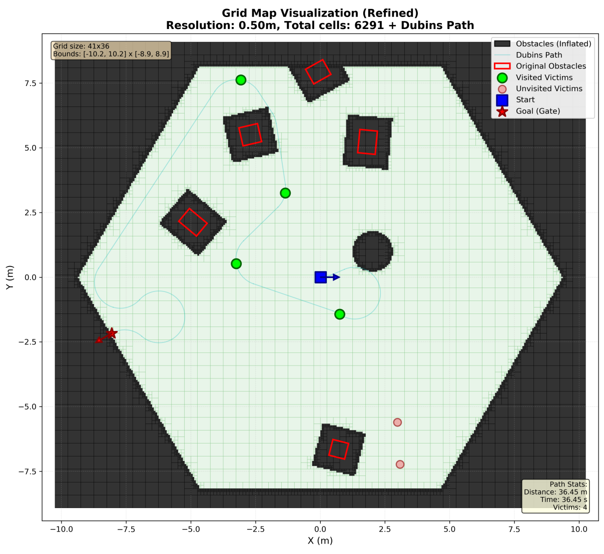

Figure 8: Path with a time budget of $45s$

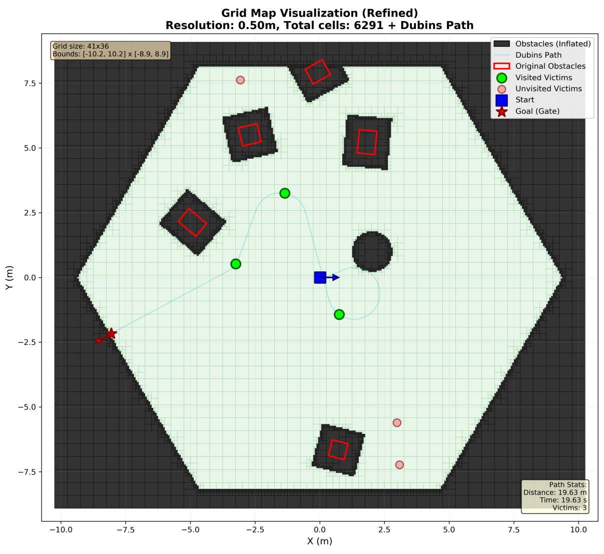

Figure 9: Path with a time budget of $30s$

6. Conclusions

This work successfully implemented and compared two approaches for the Target Rescue problem. Both methods effectively solve the NP-hard Orienteering Problem and generate kinematically feasible Dubins trajectories respecting nonholonomic constraints.

The **combinatorial approach** offers $\sim 9\times$ faster planning than **RRT***, making it preferable for real-time scenarios. RRT* produces smoother paths but suffers severe performance degradation when adaptive margin retries are needed. The implementation of collision-aware Dubins curve selection and the adaptive safety margin mechanism demonstrated a significant improvement in robustness without sacrificing optimality.

For practical deployment, grid-based planning is recommended for time-critical missions, while RRT* suits offline optimization where smoothness justifies computational cost.

Future work could explore hybrid approaches combining grid-based speed with sampling-based refinement, or algorithms that progressively improve solutions until timeout.