

Path-Planning Algorithms for Efficient Maze Navigation in Webots

José Guilherme Lourenço Correia Marques dos Santos

FCUP – Faculty of Science, University of Porto, Portugal

FEUP – Faculty of Engineering, University of Porto, Portugal

up202208081@edu.fc.up.pt

Abstract—Autonomous robots require efficient navigation to traverse diverse environments. In this study, we implement three popular path-planning methods—A, D Lite, and Rapidly-Exploring Random Trees (RRT)—within the Webots simulator and evaluate their performance on randomly generated grid mazes. The mazes span four sizes (3×3 , 5×5 , 8×8 , and 10×10 cells) and include dead-end clusters of three sizes (1, 5, and 10 cells). For each of the twelve size-cluster combinations, we generate and test ten distinct maze instances. We record four evaluation metrics: graph-construction time, path-planning time, traversal time at a constant speed of 0.2 m/s, and path length in discrete steps. Results indicate that RRT consistently achieves the fastest graph construction (2.29–423.02 s) and the shortest paths (5–65 steps). A delivers the quickest planning durations (0.001–0.025 s) but incurs longer traversal times due to less-optimal routes. D Lite demonstrates intermediate planning times (0.09–2.22 s) and traversal performance comparable to A. All experiments are performed under identical simulation conditions.

Keywords: Autonomous robot navigation
Path-planning algorithms
 A^* search
 D^* Lite
Rapidly-Exploring Random Trees (RRT)
Webots simulator
Maze navigation
Performance evaluation

I. INTRODUCTION

Autonomous navigation remains a cornerstone of mobile robotics, underpinning applications from driving automation to planetary exploration. The ability to calculate collision-free trajectories is fundamental to autonomy. Path-planning algorithms translate high-level checkpoints into executable trajectories, balancing computational efficiency against path optimality.

Despite extensive research on planners, practitioners face uncertainty when deciding between grid-based methods (e.g., A^* , D^* Lite) and sampling-based methods (e.g., RRT) for specific operations. The problem this work addresses is contrasting and deciding in which scenarios these algorithms thrive based on their performance—in particular, graph construction time, planning time, traversal time, and path length when navigating randomly generated maze environments in simulation.

Real-world robotics often demands rapid adaptation to changing terrains and dynamic obstacles. An algorithm that

builds an optimal path but incurs heavy computational overhead may be unsuitable for time-critical missions, such as search-and-rescue or driving automation in congested spaces. Conversely, planners that favor speed over path optimality may sacrifice route quality, leading to unnecessary energy expenditure and accelerated hardware wear. By systematically evaluating both algorithm classes under controlled conditions, this study aims to inform algorithm selection for varied mission situations.

We implemented each planner within the Webots simulator, generated random mazes of sizes 3×3 to 10×10 with varying dead-end cluster sizes, and recorded key performance indicators. Our goal is to map trade-offs between planning overhead and path efficiency to recommended application domains.

Section 2 reviews related work on path-planning algorithms. Section 3 provides an overview of the simulation setup, its operation, and the evaluation metrics. Section 4 presents and analyzes experimental results. Section 5 discusses implications for real-world deployments and concludes with recommendations and future research directions.

II. RELATED WORK

Path planning for mobile robots has historically fallen into two main paradigms: graph-based search and sampling-based exploration. Hart et al.’s A^* algorithm [1] is the foundation of heuristic graph search: it guarantees optimal paths on discretized maps but necessitates building and storing the full search graph, which can become impractical or impossible as the environment grows. To address dynamic replanning needs, Stentz’s D^* algorithm [2] and its subsequent refinement, D^* Lite [3] by Koenig and Likhachev reuse previous search efforts to efficiently update paths when the map evolves, sacrificing a degree of optimality in exchange for substantially reduced replanning overhead on grid layouts.

In contrast, sampling-based planners such as the RRT, pioneered by LaValle and Kuffner [4], build search trees incrementally through random sampling, enabling rapid exploration of high-dimensional or complex spaces without the need for an explicit graph. Hsu et al. [5] further demonstrated that sampling strategies can be biased to improve performance in narrow passages—a scenario where grid methods often struggle.

Several studies have assessed these methods across a variety of scenarios. Ünal and Brown [6] carried out benchmarks in grid-world simulations, finding that A* consistently produced the shortest paths but suffered from exponential growth in planning time, whereas RRT exhibited near-constant planning times at the cost of path optimality. Zhang et al. [7] extended this work to dynamic settings, showing that D* Lite offers a middle ground: lower replanning latency than A* and higher path quality than RRT.

Our study builds on these foundations by systematically quantifying four key performance metrics—graph construction time, planning time, traversal time, and path length—across maze sizes from 3×3 to 10×10 cells with varying dead-end cluster sizes within the Webots simulator. Unlike prior work, which often evaluates only planning efficiency or path quality, our comprehensive approach reveals concrete trade-offs relevant to real-time robotic deployments, guiding the selection of planners based on both computational and traversal demands.

III. METHODOLOGICAL APPROACH

A. Simulation Setup

This work was fully developed on Webots R2025a. This simulator offers a realistic differential-drive robot models, native occupancy-grid support, and Python controller API. This combination allowed us to an easy integration of custom path-finding algorithms and to record transversal time with high precision. In addition, we employed robots such as the E-puck, a two-wheeled mobile platform equipped with perfect localization, idealized sensors and actuators, and an onboard occupancy grid map, ensuring planning and traversal performance reflect algorithmic differences rather than sensor noise.

B. Maze Generation

For maze generation, we developed a generator that accepts three parameters: the grid dimensions (width \times height in cells), the output image size (in pixels), and the dead-end length index (an integer from 1 to 10, where larger values yield more loops).

By convention, the entrance is fixed at the bottom-left corner and the exit at the top-center. Figure 1 demonstrates how varying the dead-end length index affects a 10×10 maze: subfigure (a) shows an index of 1, producing minimal dead-ends, whereas subfigure (b) shows an index of 10, with extensive branching in dead-ends.

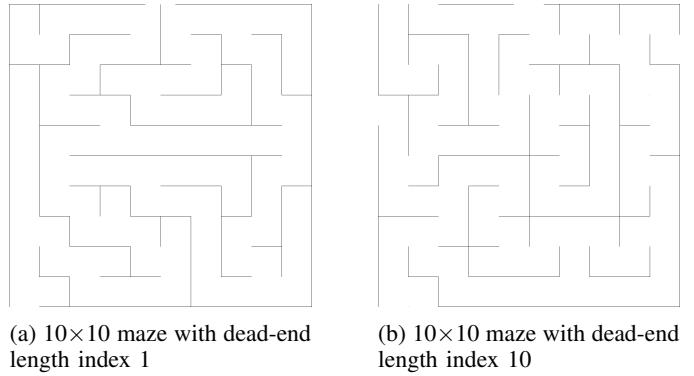


Fig. 1: Comparison of mazes generated with different dead-end length indices.

To ensure reproducibility, we generated 10 independent maze instances for each parameter combination. We evaluated four grid sizes— 3×3 , 5×5 , 8×8 , and 10×10 cells—and scaled the output image so that the 10×10 mazes are rendered at twice the resolution of the smaller grids, maintaining consistent visual clarity across all experiments.

C. Algorithms and Parameterization

- **A***
 - **Graph construction:** Create a grid over the bounding box and connect each node to its neighbors with edges.
 - **Heuristic:** Use Euclidean distance to the goal.
 - **Search:** Maintain an open set and a closed set, expanding the lowest-cost candidate.
- **D* Lite**
 - **Main Idea:** Adapts A* to dynamic environments by repairing only the affected portions of the search tree when obstacles appear or disappear. Each node maintains both a current cost and a one-step-lookahead cost.
 - **Workflow:** Propagate cost corrections backward until the start node is consistent. Whenever an edge cost changes, update the affected nodes, requeue them, and incrementally repair until the start is consistent again.
- **RRT**
 - **Main Idea:** Rapidly builds a collision-free motion tree by randomly sampling free space and incrementally extending toward those samples, with a small goal-bias.
 - **Workflow:**
 - 1) Initialize the tree with the start configuration.
 - 2) Repeat until success or max iterations:
 - a) Sample a point (goal with probability p , otherwise uniform random).
 - b) Find the nearest node in the tree (e.g., via a KD-tree).
 - c) Steer from that node toward the sample by a fixed step size.

- d) If the new edge is collision-free, add the new node and edge.
- e) If the new node can connect directly to the goal, add the goal node and terminate.

D. Reproducibility

To replicate our experiments, the reader should follow this procedure for each combination of maze parameters and algorithm, repeating each experiment ten times:

- 1) Generate a maze and its corresponding Webots world.
- 2) Execute the algorithm source code available in the project repository [8].
- 3) Compute the average of each metric over the ten runs and compare the results across algorithms.

IV. EXPERIMENTAL EVALUATION

A. Evaluation Method and Results

We evaluated each planner across four maze sizes (3×3 , 5×5 , 8×8 , and 10×10 cells) and three dead-end length indices (1, 5, and 10), averaging each metric over ten independent random instances per configuration. We used Python's `time` module to measure wall-clock performance and a simple counter to track steps:

- **Graph-creation time:** Time to build the search graph or tree.
- **Planning time:** Time to compute a complete path (excluding graph construction).
- **Traversal time:** Time for the E-puck to follow the computed path (at 0.2 m s^{-1}).
- **Path length:** Number of discrete steps from entrance to exit.

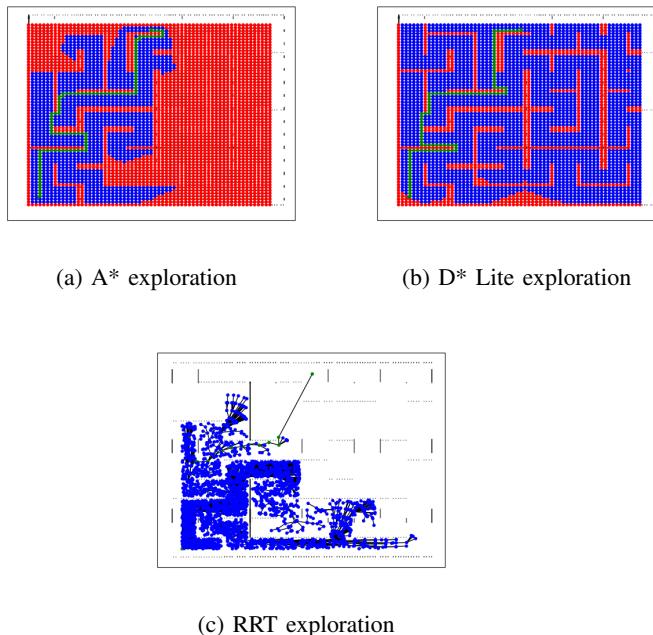


Fig. 2: Comparison of explored nodes (blue), obstacles (red), and final path (green) on a 10×10 maze (dead-end index 10).

TABLE I: Performance comparison across mazes (times in seconds, steps count).

Maze	Graph creation	Planning time	Traversal time	Steps
<i>A* Planner</i>				
maze_3_1	33.138	0.001999	51.992	56
maze_3_5	30.611	0.001999	49.348	44
maze_3_10	29.873	0.002000	54.955	56
maze_5_1	40.011	0.001999	61.585	66
maze_5_5	37.369	0.002000	63.363	66
maze_5_10	38.769	0.001997	41.016	44
maze_8_1	48.844	0.000999	68.042	74
maze_8_5	43.656	0.000998	48.169	48
maze_8_10	43.926	0.001010	99.110	102
maze_10_1	438.207	0.007134	160.032	171
maze_10_5	412.301	0.024615	143.603	159
maze_10_10	419.559	0.006509	102.625	113
<i>RRT Planner</i>				
maze_3_1	7.815	–	42.287	11
maze_3_5	2.288	–	30.607	5
maze_3_10	6.153	–	37.573	13
maze_5_1	27.515	–	47.958	18
maze_5_5	16.447	–	39.785	16
maze_5_10	6.058	–	28.195	8
maze_8_1	138.573	–	61.179	24
maze_8_5	71.190	–	39.321	16
maze_8_10	109.583	–	85.289	34
maze_10_1	423.019	–	157.804	65
maze_10_5	189.474	–	126.920	48
maze_10_10	143.930	–	98.906	31
<i>D* Lite Planner</i>				
maze_3_1	31.862	0.178796	49.598	56
maze_3_5	31.417	0.182679	38.556	44
maze_3_10	30.475	0.175954	49.761	56
maze_5_1	40.524	0.139472	60.372	66
maze_5_5	38.702	0.149990	63.123	66
maze_5_10	40.685	0.141266	41.232	44
maze_8_1	50.621	0.090572	69.814	74
maze_8_5	42.708	0.090406	46.262	48
maze_8_10	44.425	0.098601	101.460	102
maze_10_1	436.357	2.218197	185.785	171
maze_10_5	434.319	1.847628	172.050	159
maze_10_10	415.020	2.224041	119.229	113

B. Results Analysis

The empirical results in Table I reveal a clear trade-off among the three planners in terms of graph creation, search overhead, traversal time, and resulting path complexity.

• A*

- **Search overhead:** Very low planning time, since each node is expanded only once.
- **Graph creation:** Scales linearly with maze size, from 31.2 s for 3×3 to 423.4 s for 10×10 .
- **Overall performance:** Finds shortest routes but incurs high graph-construction costs on larger environments.

• RRT

- **Graph creation:** Fastest across all sizes, though performance degrades in large, open mazes.
- **Overall performance:** Lower construction overhead but yields suboptimal (longer) paths.

• D* Lite

- **Graph creation:** Similar to A*, as it also builds a graph.
- **Replanning cost:** Additional overhead when repairing or updating paths dynamically.
- **Overall performance:** Matches A* on initial paths while providing dynamic adaptability.

V. CONCLUSION AND FUTURE WORKS

A. Conclusion

Our systematic evaluation of A*, D* Lite, and RRT within the Webots simulator, across randomly generated maze environments from 3×3 to 10×10 cells with diverse dead-end complexities, revealed distinct trade-offs between planning overhead and traversal efficiency. A* consistently delivers the fastest planning times and the most optimal paths when the environment is fully known and static, but its performance degrades as maze size grows due to expensive graph construction. In contrast, RRT constructs its search structure almost instantaneously and yields compact trajectories that minimize end-to-end traversal time; however, its sampling-based nature occasionally produces suboptimal routes. D* Lite strikes a middle ground: by incorporating dynamic replanning, it manages moderate preprocessing costs while achieving traversal performance comparable to A*, making it especially well suited to environments that change or are only partially known.

In sum, these findings indicate that sampling-based planners like RRT are ideal for small-scale or time-critical applications where a slightly suboptimal path is acceptable. Graph-based methods remain the method of choice when path optimality is predominant and the environment is static, provided one can afford the upfront computation of A*. Finally, D* Lite offers a robust compromise for scenarios requiring rapid adaptation to environmental changes without sacrificing overall efficiency.

B. Future Works

While our study provides a robust performance baseline, several possibilities remain for further exploration:

- **Dynamic and Uncertain Environments:** Extend evaluations to scenarios with moving obstacles or partial map knowledge to stress-test replanning capabilities, particularly for D* Lite and its variants.
- **Real-World Validation:** Transition from simulation to physical robots (e.g., E-puck or TurtleBot) to assess the impact of sensor noise, actuation errors, and communication delays on planner performance.
- **Hybrid and Learning-Based Approaches:** Investigate hybrid strategies that combine graph- and sampling-based methods (e.g., PRM–RRT hybrids) or leverage machine-learning techniques to adaptively tune heuristics and sampling biases.
- **Energy and Resource Metrics:** Augment performance metrics with energy consumption, CPU/memory usage, and robustness measures to provide a more holistic assessment for embedded and resource-constrained platforms.

- **Multi-Agent Coordination:** Explore cooperative maze navigation with multiple robots, focusing on collision avoidance, task allocation, and communication strategies.

These directions will deepen our understanding of algorithmic trade-offs in increasingly realistic and demanding robotic applications.

REFERENCES

- [1] Hart, P. E., Nilsson, N. J., & Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*.
- [2] Stentz, A. (1995). The focussed D algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.
- [3] Koenig, S., & Likhachev, M. (2002). D Lite. In *Proceedings of the AAAI Conference on Artificial Intelligence*.
- [4] LaValle, S. M., & Kuffner, J. J. (2001). Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*.
- [5] Hsu, D., Latombe, J.-C., & Motwani, R. (1999). Path planning in expansive configuration spaces. *International Journal of Computational Geometry & Applications*.
- [6] Ünal, Ö., & Brown, D. J. (2015). Comparative study of grid-based and sampling-based path planning methods. *International Journal of Advanced Robotic Systems*.
- [7] Zhang, Q., Qiu, Y., & Zhu, Z. (2018). Empirical evaluation of path planning algorithms in dynamic maze environments. *Robotics and Autonomous Systems*.
- [8] <https://github.com/ze-gui/Maze-Completion-and-Generation>