CSci 4511W Final Project Report

Baanee Singh & Masooma Sayeda

May 2025

Abstract

This project focuses on developing and comparing two heuristic-based path planning algorithms, A* and D*-Lite, for robot navigation in 2D indoor grid-based environments with static and dynamic obstacles such as walls, furniture, and stairs. Each grid cell is categorized as either free space or an obstacle, and the goal is to determine an optimal path from a start position to a goal location while avoiding collisions. A* is known for its efficiency in static environments, using a cost function based on the Manhattan distance heuristic, while D*- Lite is designed for dynamic environments, allowing real-time path re-planning without restarting the search. We implemented both algorithms in Python using libraries like NumPy for computation and Matplotlib for visualization, and we also used several open-source simulation tools to help develop and test our code. Performance was evaluated based on path optimality, path length, computation time, and obstacle avoidance margin. Based on initial expectations, A* was predicted to perform better in static scenarios, while D*- Lite would show better adaptability in dynamic environments. Our results highlight the strengths and trade-offs of each approach and guide practical robot navigation tasks.

Problem Description

Efficient and reliable path planning is a key challenge in robotics. Autonomous robots and vehicles today operate in environments that are often unpredictable. Whether delivering packages in busy urban areas, assisting medical staff in hospitals, or performing search and rescue missions, robots must navigate safely through spaces that include stationary and moving obstacles. In typical 2-D grid environments, each cell is marked either as free space or as an obstacle, like a wall, piece of furniture, or another object. The robot must find an efficient path from its starting point to a goal location without hitting anything along the way.

Path planning is important not only for efficiency, but also for safety. A robot that can quickly and safely find paths will perform better in real-world applications like self-driving cars, warehouse automation, and home assistance. Finding an efficient path means reaching the destination faster and using less energy, which can also extend battery life for mobile

robots. Safe navigation, on the other hand, prevents accidents and damage to both the robot and its surroundings.

One thing that makes this project particularly interesting is that real-world environments are not static. Obstacles may appear, disappear, or move. For example, a chair could be moved into a hallway, a door could close unexpectedly, or another robot could block a planned path. In these situations, it is not enough for a robot to find a good path once; it needs to adapt its path quickly without starting over from scratch.

One of the most commonly used pathfinding algorithms is A*. A* works well in environments where the map is known ahead of time. It combines the actual cost to reach a node (g-value) and an estimated cost to reach the goal (h-value, given by a heuristic like Manhattan distance) to find the shortest path. However, A* is not designed to deal with unexpected changes. If the environment changes after planning, A* must throw away its work and completely redo the search, which can waste time and resources.

On the other hand, D*-Lite is built specifically for environments that change over time. D*-Lite doesn't throw away its planning work when something changes. Instead, it updates the parts of the path that are affected by the new information. It keeps track of both the current cost (g-value) and a look-ahead estimate (rhs-value) for each node, using a priority queue to manage updates. This makes D*-Lite more flexible and much better suited for real-world use, where things don't stay the same.

Given the importance of both planning efficiency and adaptability, this project focuses on building and comparing A* and D*-Lite in static and dynamic grid environments. We tested how each algorithm behaves under different obstacle layouts, using heuristics like Manhattan, Euclidean, and Chebyshev distances. Both algorithms were implemented in Python, using libraries such as NumPy for grid operations and Matplotlib for visualizations. The evaluation was based on multiple metrics: total cost of the path, number of steps, computation time, and how well the robot avoided obstacles.

By comparing these two approaches in a controlled environment, our project aims to give clear insights into the trade-offs between A* and D*-Lite, helping others choose the right algorithm for their robotic navigation tasks.

Introduction to Robot Path Planning

Robot Path Planning is a crucial domain in robotics and artificial intelligence (AI), tasked with determining optimal routes for autonomous systems to reach a destination without collisions. Its applications span a wide range of fields, from industrial automation to self-driving vehicles, all of which rely heavily on real-time decision-making and spatial reasoning. Effective path planning must account for both static and dynamic environments. In static environments, all obstacles are known beforehand and remain unchanged, whereas dynamic environments involve moving or unpredictable obstacles that require real-time updates and re-planning.

The primary goals of robot path planning include minimizing travel distance and time, ensuring collision-free navigation, and optimizing computational efficiency, particularly in

scenarios demanding real-time performance. Among the various algorithms developed, the A* algorithm is widely regarded as one of the most efficient and optimal heuristic-based search methods.

A* Algorithm in Static Environments

Gigras and Gupta (2012) [1] emphasize the effectiveness of A* in structured, static environments, where complete map information is available and path recalculations are unnecessary. The algorithm is optimal, finding the shortest path under certain conditions and complete, guaranteeing that a solution will be found if one exists, assuming an admissible heuristic. In grid-based indoor environments, commonly used heuristics such as Euclidean or Manhattan distance can significantly influence both time and path quality.

Limitations and Enhancements of A*

Despite its strengths, A* becomes less suitable in dynamic or large-scale environments, where changes require complete re-planning. This makes the algorithm computationally expensive and potentially inefficient for real-time applications. To address these limitations, researchers have explored more adaptive strategies. For instance, Gilbert and Johnson (1985) [2] introduced a method for planning robot movement that helps the robot avoid obstacles while still taking the best possible path. Instead of handling obstacle avoidance separately, they make it part of the main planning process by using distance functions. These functions continuously measure the robot's proximity to obstacles, allowing smoother and more efficient motion. Though originally applied to robotic arms, their method applies to mobile robots navigating indoor spaces. When integrated with algorithms like A* or D*, their approach enhances both path safety and realism by enabling smoother curves and avoiding abrupt movements.

Building on the foundational A* algorithm, Fu et al. (2018) [3] proposed an enhanced A* algorithm tailored for industrial robots, incorporating dynamic weighting in heuristics to improve success rates and reduce path length in cluttered factory layouts. Their method introduces a pre-processing stage that checks for a direct, collision-free path to the goal before searching nearby nodes and a post-processing stage that removes unnecessary points to shorten the path. Their approach demonstrated a higher success rate and shorter paths compared to traditional A*, making it suitable for robotics in constrained industrial settings. However, these improvements come at the cost of increased complexity, leading to a trade-off between accuracy and speed, especially relevant for real-time or embedded systems, where approximate solutions may be preferable.

An additional consideration discussed in "A Guide to Heuristic-based Path Planning" [4] is how various search algorithms can be adapted or extended using domain-specific heuristics or memory-efficient structures. For A*, this includes weighted A* variants and memory-bounded search strategies, which aim to maintain near-optimal paths while significantly reducing runtime and space requirements. These adaptations make A* more viable for larger environments or devices with limited onboard resources.

Introduction to the D* Lite Algorithm

Dynamic environments further complicate path planning by introducing challenges such as moving obstacles, incomplete maps, and unexpected changes. Traditional A* algorithms struggle in such settings due to the need for complete replanning. Addressing this gap, D* Lite was developed as a re-planning algorithm designed to adapt to changes efficiently. Unlike A*, D* Lite retains previous search information and updates only the affected areas when the environment changes. This feature makes it particularly well-suited for real-time navigation in partially known or evolving environments.

Setiawan et al. (2014) [5] compared the performance of A* and D* Lite in both simulation and real-world tests using a mobile robot (AGV). Their findings revealed that while A* could be faster in obstacle-free environments due to its simplicity, D* Lite outperformed it when dealing with dynamic obstacles. Updating paths incrementally allowed for smoother adjustments and improved navigation efficiency in larger or more complex environments.

This is echoed in the experimental work by Pandu Sandi Pratama [6], who evaluated both algorithms on a differential-drive AGV platform. Their results emphasized the flexibility of D* Lite in responding to changes and suggested it as the preferred choice for applications involving frequent re-mapping or unpredictable elements, such as delivery or maintenance robots.

Foundations, Enhancements, and Variations of D* Lite

Further validating the advantages of D* Lite, Bagad et al. (2024) [7] demonstrated that the algorithm significantly reduces the computation time by updating only the nodes affected by the changes rather than recalculating the entire path. Their experiments on grid maps ranging from 100×100 to 500×500 cells showed that D* Lite maintained high success rates and delivered near-optimal paths with faster replanning—an essential quality for real-time applications such as warehouse robotics and indoor mobile navigation.

Koenig and Likhachev (2002) [8] provided a solid theoretical foundation for D* Lite, emphasizing its simplified structure compared to its predecessor, Focused D*. The algorithm's clean design avoids layered conditions and uses consistent rules for tie-breaking, making it easier to implement and expand. They also highlighted that D* Lite can accommodate non-admissible heuristics and customizable rules to enhance performance. Their analysis showed that each node is expanded at most twice after a change, proving D* Lite's computational efficiency and robustness—qualities that make it an excellent choice for future research in real-time robotic planning.

Several enhancements to the standard D* algorithm have also been proposed to improve its practical performance. For instance, Guo et al.(2009) [9] extended the direction resolution in D* from 4 or 8 to 16 possible movement directions. This refinement led to smoother and more efficient paths by reducing unnecessary turns and shortening routes—advantages confirmed through experiments using the WiRobotX80 mobile robot. The results highlighted the importance of fine-grained direction control in improving motion efficiency and path quality in real-world navigation scenarios.

Similarly, Dakulović and Petrović (2011) [10] introduced the two-way D* (TWD*) algorithm, specifically designed for navigating indoor environments such as buildings. By planning from both the start and the goal simultaneously and incorporating straight-line segments, TWD* generated smoother, more natural paths. The inclusion of safety margins around obstacles further improved collision avoidance. Tested in both simulations and real-world settings, TWD* demonstrated its ability to quickly adapt to changes while maintaining high path efficiency, making it a practical choice for service and delivery robots in indoor settings.

An article by Deshpande et al. [11] investigated enhancements to D* Lite by integrating cost prediction models. Their implementation used prior knowledge of terrain types and incorporated weighted cost adjustments, leading to better energy efficiency for wheeled robots operating in industrial areas. Their findings further support the use of D* Lite in energy-sensitive applications and reinforce the value of combining path-planning algorithms with environmental semantics.

Project Relevance

Heuristic algorithms like A* and D* Lite are central to robot path planning. A* is effective in static, predictable settings, whereas D* Lite thrives in dynamic, partially known environments. Variants and improvements continue enhancing efficiency, success rates, and applicability in real-world scenarios. Recent studies have combined simple reactive navigation with Q-learning to help robots move more effectively in unknown areas. This mix helps the robot react quickly while also learning better paths over time. It works in both 2D and 3D spaces, like for ground robots and drones, and has been shown in tests to handle both still and moving obstacles well [12].

Our project leverages both algorithms to develop and evaluate efficient path-planning solutions for indoor robot navigation within a 2D grid. By simulating real-world challenges, we aim to highlight trade-offs in performance, speed, and obstacle avoidance, reinforcing the broader importance of path planning in autonomous systems.

Approach to Solve the Problem

To fairly and systematically compare A* and D*-Lite, we built an indoor robot navigation simulation using Python. The environment was modeled as a 2-D grid where each cell is either a free space (0) or an obstacle (-1). The robot could move horizontally, vertically, and, depending on the heuristic used, diagonally. The start and goal positions were placed at opposite corners of the grid to encourage longer, more complex paths and to test the adaptability of each algorithm over a significant distance.

For A*, we implemented two versions: a static version and a dynamic version. In the static version, the robot knows the entire environment beforehand and plans the full path at the start. Once planned, the robot simply follows the path. If nothing changes in the environment, A* is very efficient because it doesn't need to rethink anything after planning.

In the dynamic version, unexpected obstacles were added while the robot was moving. If the robot encountered a new obstacle that blocked its current path, it would trigger a complete re-planning operation from its current location. This dynamic version highlights A*'s main weakness: every time an unexpected change occurs, the algorithm has to start the entire planning process over, which can be slow, especially in bigger grids.

D*-Lite takes a different approach. Like A*, it starts by planning a path based on what it currently knows. But when something in the environment changes, D*-Lite doesn't start over. Instead, it incrementally updates the parts of the plan that are affected by the change. Each node keeps track of two values: the g-value (the known cost to reach the node) and the rhs-value (a one-step estimate of the cost). When an obstacle appears, D*-Lite updates only the necessary nodes, allowing the robot to quickly recover and continue toward the goal.

To guide the search, we used three types of heuristics:

- **Manhattan Distance** [13]: Sum of the horizontal and vertical distances. Only four directions of movement are allowed.
  Equation: $D_{\text{Manhattan}}(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$
- **Euclidean Distance** [13]: Straight-line distance, allowing diagonal movement.
  Equation: $D_{\text{Euclidean}}(x_1, y_1, x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
- **Chebyshev Distance** [13]: The greater of the horizontal or vertical distance, allowing diagonal moves with the same cost as straight moves.
  Equation: $D_{\text{Chebyshev}}(x_1, y_1, x_2, y_2) = \max(|x_1 - x_2|, |y_1 - y_2|)$

Dynamic obstacles were placed partway along the robot's expected path. In the 10x10 grid, new obstacles appeared near the center. In the 50x50 grid, dynamic obstacles were introduced along major pathways where they could block a straightforward path and force the robot to adapt.

Both static and dynamic environments were tested systematically across different grid sizes:

- **10×10 grid**: To simulate smaller, more compact indoor spaces.
- **50×50 grid**: To represent larger, more complex environments where paths are longer and obstacle avoidance becomes more challenging.

Each experiment was repeated 10 times to reduce variability and ensure consistent measurements. We averaged results across the runs to account for minor variations in system performance.

The algorithms were coded in Python using libraries like NumPy for numerical operations, heapq for handling the priority queues, and Matplotlib for visualizing the grids and paths. The A* algorithm followed ideas from the AIMA Python repository [14] and Atsushi Sakai's PythonRobotics project [15]. D*-Lite was based on Koenig and Likhachev's original paper [8], as well as open-source implementations [16].

All algorithms and heuristics were tested under identical conditions. The grid layouts, dynamic obstacle positions, and start-goal locations were kept the same to ensure a fair comparison focused only on the algorithms' design differences.

Experimental Design

To rigorously evaluate the performance of A* and D*-Lite algorithms in static and dynamic environments, we conducted a series of structured experiments using the simulation framework described previously. Our experiments aimed to measure how quickly and efficiently each algorithm could navigate a robot from a designated start cell to a goal cell in a 2D grid-based environment while adapting to both known and unknown obstacles. In addition to computation time and path length, we were also interested in the algorithms' ability to maintain successful navigation when the environment changed unexpectedly.

Two grid sizes were selected for testing. A small 10×10 grid was used to simulate compact indoor environments, such as small rooms, narrow hallways, or offices. A larger 50×50 grid was used to model more complex and spacious navigation tasks, such as warehouses, open-floor offices, or large indoor arenas. This difference in grid size allowed us to test the scalability of each algorithm and see how performance changed as the navigation problem became harder.

For each grid size, obstacles were manually inserted to create realistic navigation challenges. In the 10×10 grid, obstacles such as walls and blocks were placed along the diagonal and central areas, forcing the robot to find alternate routes. In the 50×50 grid, a long vertical wall spanning rows 10 to 40 was inserted at column 25, creating a significant barrier that split the environment in half and required major detours for the robot to reach the goal.

In addition to static obstacles, dynamic obstacles were introduced partway through the navigation process to simulate real-world scenarios like moving furniture or unexpected blockages. In the 10×10 grid, dynamic obstacles appeared near the middle of the path at cells (5,6), (5,7), and (5,8). In the 50×50 grid, dynamic blocks were placed across columns 26 to 29 at row 30, cutting across a major pathway and forcing mid-course re-planning. Dynamic changes were introduced consistently at pre-specified locations across all experiments to maintain fairness and allow direct comparison between runs.

For each experiment, the robot's start position was set at the top-left corner (0,0), and the goal position was set at the bottom-right corner: (9,9) for the small grid and (49,49) for the larger grid. A run was considered successful if the robot was able to reach the goal without colliding with obstacles, whether static or dynamic.

Three different heuristics were employed to guide the search. Manhattan distance prioritized horizontal and vertical moves, Euclidean distance allowed straight-line diagonal movement, and Chebyshev distance permitted diagonal moves at the same cost as orthogonal moves. These heuristics were chosen to explore how different movement assumptions might affect overall planning efficiency.

Both A* and D*- Lite algorithms were tested under two conditions: static environments, where the obstacle layout remained fixed, and dynamic environments, where new obstacles were introduced midway through navigation. Each configuration (algorithm × grid size × heuristic × static/dynamic) was tested across 10 independent runs to account for random timing fluctuations and ensure the reliability of our observations. Computation time and path length were recorded for every run, and the results were averaged to create stable, meaningful comparisons.

Results

Table 1: 10×10 Grid Results (Averaged over 10 Runs)

Algorithm	**Path Length**	**Time (seconds)**
A* (Static, Manhattan)	19	0.001825
A* (Dynamic, Manhattan)	19	0.000696
D*-Lite (Static, Manhattan)	19	0.002965
D*-Lite (Dynamic, Manhattan)	19	0.004965
A* (Static, Euclidean)	19	0.001622
A* (Dynamic, Euclidean)	19	0.000860
D*-Lite (Static, Euclidean)	19	0.005737
D*-Lite (Dynamic, Euclidean)	19	0.008306
A* (Static, Chebyshev)	19	0.001801
A* (Dynamic, Chebyshev)	19	0.000699
D*-Lite (Static, Chebyshev)	19	0.004581
D*-Lite (Dynamic, Chebyshev)	19	0.006049

Observations on Table 1: 10×10 grid

On small grids, all algorithms found paths of the same length (19 steps), meaning that the obstacle layouts did not give any advantage to either A* or D*- Lite in terms of path quality. A* was consistently faster than D*- Lite in static conditions, which is expected because A* simply performs a direct search without any extra tracking of future changes. Interestingly, in the 10×10 grid, Dynamic A* sometimes showed slightly faster computation times than Static A*. This happened because Dynamic A* could initially plan quicker direct paths and only replan if a new obstacle appeared later, while Static A* computed everything upfront with a full search. D*-Lite was slower even when no dynamic obstacles were present, because it maintains extra internal structures (g-values, rhs-values, and priority queues) to prepare for possible changes. However, in dynamic conditions, D*-Lite adapted immediately without restarting the search, showing its advantage for scalability even in smaller environments.

Observations on Table 2: 50×50 grid

On larger grids, A* still planned faster than D*- Lite when no dynamic changes were present. Its single-shot planning approach made it efficient when the environment was completely known at the start. However, once dynamic obstacles were introduced, A* had to completely restart the planning process. This became more costly as the environment grew larger, making the search much slower in real-time applications. Full replanning became a clear disadvantage at larger scales. On the other hand, D*-Lite handled new obstacles much more smoothly. Instead of rebuilding the entire path, it updated only the parts that were affected

Table 2: 50×50 Grid Results (Averaged over 10 Runs)

Algorithm	**Path Length**	**Time (seconds)**
A* (Static, Manhattan)	99	0.022448
A* (Dynamic, Manhattan)	99	0.020953
D*-Lite (Static, Manhattan)	99	0.157989
D*-Lite (Dynamic, Manhattan)	99	0.085253
A* (Static, Euclidean)	99	0.038832
A* (Dynamic, Euclidean)	99	0.031253
D*-Lite (Static, Euclidean)	99	0.145103
D*-Lite (Dynamic, Euclidean)	99	0.128368
A* (Static, Chebyshev)	99	0.021222
A* (Dynamic, Chebyshev)	99	0.023781
D*-Lite (Static, Chebyshev)	99	0.137029
D*-Lite (Dynamic, Chebyshev)	99	0.105456

by changes, allowing the robot to continue moving efficiently. Even though D*- Lite's total computation time was still a bit higher than A* on small grids, its ability to adapt without starting over became much more important on larger grids. Another interesting observation was that on the 50×50 grid, Dynamic D-Lite* was faster than Static D-Lite*. This shows that in complex environments, adapting to changes locally is easier and faster than building a perfect plan ahead of time. Path lengths stayed the same across all heuristics and algorithms because obstacles were placed away from the shortest direct paths between start and goal. This means that no matter the algorithm or heuristic, all were able to find similar quality solutions.

Additional Observations

In addition to the trends observed between static and dynamic cases, another noticeable pattern was the change in computation time differences as the grid size increased. On the smaller 10×10 grid, the difference between A* and D*- Lite in static conditions was relatively minor, with A* completing searches about 0.001 to 0.003 seconds faster, depending on the heuristic. However, on the 50×50 grid, this gap became much larger. Static A* searches took around 0.02 to 0.03 seconds, while staticD*- Litee searches extended to about 0.13 to 0.16 seconds across different heuristics. This widening gap shows that D*-Lite's baseline overhead becomes more significant as the environment grows larger. Similarly, the difference between static and dynamic computation times in D*-Lite also grew with grid size, reflecting that maintaining update structures and preparing for changes adds more cost on larger maps. These timing patterns reinforce the idea that scalability becomes a much more important factor when moving from small, compact spaces to larger, more realistic environments.

Analysis of the Result

The experiments show clear patterns about when to use A* and when to use D*- Lite.

In static environments where the layout is fully known ahead of time and does not change, A* was faster and more efficient across both small and large grids. It could find the shortest path quickly because it only needed to perform a single search based on a complete map. Its simple cost function (f(n) = g(n) + h(n)) allowed it to focus directly on reaching the goal without extra computation.

However, A* struggled when dynamic changes occurred. Every time a new obstacle appeared, A* had to throw away all its previous work and start the search again from scratch. On small grids like the 10×10, this was not a major problem, but on larger grids like the 50×50, restarting planning took more and more time, reducing the robot's ability to react quickly in real time.

D*-Lite, by contrast, was a bit slower to plan even when there were no changes. This is because D*-Lite always maintains two pieces of information (g-values and rhs-values) for every node and keeps a priority queue ready for updates. This preparation adds overhead, but it allows D*-Lite to adapt quickly later when the environment changes, avoiding full replanning.

When dynamic obstacles appeared, D*-Lite could selectively update only the parts of the path that were affected, rather than rebuilding the entire search tree. This made it much more adaptive and reliable for environments that could change suddenly.

- If the environment is **static**, A* is faster and more efficient.
- If the environment is **dynamic**, D*-Lite is better because it adapts without redoing everything.

The choice of heuristic (Manhattan, Euclidean, Chebyshev) slightly affected computation time, but did not change path quality:

- **Manhattan** was a little faster in purely grid-based settings because it assumes movement in four directions.
- **Euclidean** and **Chebyshev** allowed diagonal movement, which added a tiny amount of extra computation but did not change the final path length in our obstacle layouts.

Another important finding was that path length was almost identical across all algorithms and heuristics. This shows that both A* and D*-Lite are capable of finding good paths as long as the heuristics used are reasonable.

A final important pattern is that the difference between A* and D*- Lite becomes much larger as the environment size increases. While in small grids (10×10) the difference in planning time was minor, in larger grids (50×50) D*-Lite's slower initial planning became much more noticeable. However, D*-Lite's ability to adapt without full replanning made it increasingly valuable as grid complexity grew.

Additionally, it was observed that in larger, dynamic grids, dynamic D * -Lite sometimes outperformed static D * -Lite, showing that adapting to changes locally can be more efficient than building a complete plan in advance. These trends reinforce the importance of considering scalability when choosing between A* and D*- Lite for real-world navigation tasks.

Conclusion

This project systematically compared A* and D*-Lite algorithms for 2D grid-based robot navigation across static and dynamic environments. We measured path optimality, computation time, and adaptability to changes. The results showed that A* is the better choice for fully known, unchanging environments where speed is the top priority. Its simple search strategy makes it fast and effective when no unexpected changes occur.

D*-Lite, while slower initially, showed a clear advantage in environments where new obstacles could appear. Its ability to update paths incrementally without restarting made it much better suited for real-world scenarios like offices, hospitals, or warehouses, where environments can change without warning. Both algorithms consistently found paths of the same quality, but the key differences came down to how they handled environmental changes. A* had to restart completely, causing slowdowns on larger grids, while D*-Lite could quickly repair the affected part of the path and continue moving without significant delays.

The impact of the heuristic used was relatively minor compared to the effects of static versus dynamic environments. All three heuristics—Manhattan, Euclidean, and Chebyshev—worked well, although Manhattan tended to be slightly faster in simple grid-based navigation.

For future work, there are several interesting directions to explore. One possibility is testing these algorithms in three-dimensional environments where the robot can also move vertically. Another is adding more types of robot movement, such as differential drive robots that cannot move sideways easily. Expanding the experiments to environments with moving obstacles that change position over time, rather than just appearing suddenly, would also provide valuable insights. Furthermore, integrating machine learning models to predict environmental changes ahead of time and help dynamically adjust route planning could improve performance even further.

Overall, this project highlights that when designing autonomous navigation systems, the choice between A* and D*- Lite depends heavily on how dynamic the environment is. In static maps, A* is simpler and faster. In dynamic or uncertain settings, the adaptability of D * Lite makes it the smarter and more reliable choice.

Google Colab Notebook

Access the full implementation and results (including graphs and path-planning space) at the following link: Colab Notebook

Notes

- **Masooma Sayeda**: All parts expanded and written, did coding part, found bibliographic references, wrote citations in Overleaf, proofread Overleaf.
- **Baanee Singh**: All parts expanded and written, started ideas and outline, editions, did coding part, found bibliographic references, conversion to Overleaf, proofread Overleaf.
- **Tools Used for Grammar and Style**:
  - Grammarly
  - Overleaf Built-in AI assistant
- **ChatGPT for Latex Formatting**: ChatGPT Link

References

[1] Y. Gigras and K. Gupta, "Artificial intelligence in robot path planning," *International Journal of Soft Computing and Engineering (IJSCE)*, vol. 2, no. 2, pp. 471–474, 2012.

[2] E. Gilbert and D. Johnson, "Distance functions and their application to robot path planning in the presence of obstacles," *IEEE Journal on Robotics and Automation*, vol. 1, no. 1, pp. 21–30, 1985.

[3] B. Fu *et al.*, "An improved a* algorithm for the industrial robot path planning with high success rate and short length," *Robotics and Autonomous Systems*, vol. 106, pp. 26–37, 2018.

[4] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path planning," in *Proceedings of the ICAPS 2005 Workshop on Planning under Uncertainty for Autonomous Systems*, (Monterey, California, USA), 2005.

[5] Y. Setiawan, P. Pratama, S. Jeong, V. Duy, and S. Kim, "Experimental comparison of a* and d* lite path planning algorithms for differential drive automated guided vehicle," in *AETA 2013: Recent Advances in Electrical Engineering and Related Sciences* (I. Zelinka, V. Duy, and J. Cha, eds.), vol. 282 of *Lecture Notes in Electrical Engineering*, pp. 551–558, Springer, 2014.

[6] P. Pratama, *Experimental Comparison of A* and D* Lite Path Planning Algorithms for Differential Drive Automated Guided Vehicle*, pp. 555–564. Springer, 01 2013.

[7] Bagad, Dahatonde, Gagare, Athare, and Ghule, "Optimizing pathfinding in dynamic environments: A comparative study of a* and d-lite algorithms," in *2024 IEEE Pune Section International Conference (PuneCon)*, (Pune, India), pp. 1–11, 2024.

[8] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 476–483, AAAI Press, 2002.

[9] J. Guo, L. Liu, Q. Liu, and Y. Qu, "An improvement of d* algorithm for mobile robot path planning in partial unknown environment," in *2009 Second International Conference on Intelligent Computation Technology and Automation*, (Changsha, China), pp. 394–397, 2009.

[10] M. Dakulović and I. Petrović, "Two-way d* algorithm for path planning and replanning," *Robotics and Autonomous Systems*, vol. 59, no. 5, pp. 329–342, 2011. Special Issue ECMR 2009.

[11] S. Deshpande, A. K. Kashyap, and B. K. Patle, "A review on path planning ai techniques for mobile robots," *Robotic Systems and Applications*, vol. 3, pp. 27–46, jun 2023.

[12] J. Zhang, "Ai based algorithms of path planning, navigation and control for mobile ground robots and uavs," 2021.

[13] GeeksForGeeks, "Chebyshev distance," Dec. 2024.

[14] AIMA, "aima-python/search.py at master · aimacode/aima-python."

[15] A. Saikai, "PythonRobotics/PathPlanning/AStar/a_star.py at master · AtsushiSakai/PythonRobotics."

[16] mDeyo, "d-star-lite/d_star_lite.py at master · mdeyo/d-star-lite."