Path planning using search algorithms

André Sousa (202108571), Inês Cardoso (202107268), Paulo Silva (202107359)

[1] FEUP - Faculty of Engineering, University of Porto
[2] FCUP - Faculty of Engineering, University of Porto

Abstract. Abstract. This project explores robotic path planning using different algorithms in the Webots e-puck simulator. We will implement seven algorithms such as A*, Dijkstra, D*, Bidirectional, RRT, RRT*, and APF, additionally, we will also be employing two methods to generate graphs for A, Dijkstra, D, and Bidirectional algorithms, the Probabilistic Roadmap Method (PRM) and the Grid Method. After, we will compare the efficiency of algorithms like A*, Dijkstra, D*, Bidirectional, RRT, RRT*, and APF to optimize navigation. By measuring metrics for instance pathfinding speed, optimality, and adaptability to environments we will have a better understanding of the algorithm's strengths and weaknesses . Our objective is to get a better idea of path planning algorithms consequently their practical applications in the robotics field and contribute to safer and more efficient robotic operations across various domains.

Keywords: robotic path planning, algorithms, graph generation, optimality, metrics, Navigation optimization

[3]

1 Introduction

Robotic path planning is vital for autonomous navigation, facilitating efficient navigation, avoiding obstacles, and effective destination reach. With increasingly complex operational environments, there is a growing demand for advanced path planning algorithms.

The predicament is determining which of the 7 algorithms performs best under different conditions and requirements.

Solving this problem will improve the efficiency and safety of robotic operations across different domains.

This paper strives to answer the following questions: Which path planning algorithm offers the best balance between speed and path optimality in various environments? How do different algorithms adapt to various environmental changes? Our hypothesis is that algorithms such as A* and Dijkstra will exhibit

[3] This project is available at https://github.com/WrekingPanda/Webots_Robotics_Project.

superior performance in static environments, while algorithms like D* and RRT will demonstrate better adaptability.

We will start by implementing and comparing the efficiency of path planning algorithms, including A*, Dijkstra, D*, Bidirectional, RRT, RRT*, and APF, in simulated environments of varying complexity. Two methods for generating graphs will be employed: the Grid method and the Probabilistic Roadmap Method (PRM). We will measure and analyze performance metrics such as pathfinding speed, optimality, and adaptability.

The remainder of this paper is organized as follows: - Section **??**:Related work, in this section we will reference another scientific paper that has a similar problem as ours. - Section 3:Methodological Approach, description of each method employed to solve the problem. - Section 4:Experimental Evaluation we will give an explanation of the experiments and presentation of the performance metrics. - Section 5: Conclusions and Future Work we will present a summary of key findings, discussion of results, and proposals of potential directions for further research and alternative approaches to the problem. - Section 6: In the bibliography section we will provide a list of all the sites we visited in order to help with this project.

2 Related Work

The paper to which we are comparing ours is named Path planning in construction sites: performance evaluation of the Dijkstra, A, and GA search algorithms

Both papers discuss the realm of path planning but within very different contexts and methodologies. Our paper explores robotic path planning utilizing various algorithms in the Webots e-puck simulator. It concentrates on implementing and comparing the efficiency of seven different algorithms: A*, Dijkstra, D*, Bidirectional, RRT, RRT*, and APF. The primary aim is to optimize navigation in robotic operations by evaluating pathfinding speed, optimality, and adaptability.

In contrast, the other paper addresses path planning within construction sites based on multiple objectives. It quantitatively assesses the performance of three optimization algorithms: Dijkstra, A*, and Genetic Algorithms. Specifically, the problem involves finding multi-criteria paths in construction sites, considering both transportation and safety-related costs. The focus is on selecting paths for site operatives and vehicles that are characterized by short distances, low risks, and high visibility.

Even though both share a mutual interest in path planning, they differ in their application areas and methodologies. Our paper tries to emphasize more on simulated environments and robotic operations, while the other paper is grounded in real-world construction site scenarios. Each paper gives valuable insights into the optimization of path planning, although within different contexts.

3 Methodological Approach

3.1 Path Planning Algorithms

As referenced before the following algorithms were implemented :A*, Dijkstra, D*, Bidirectional, RRT (Rapidly-Exploring Random Tree), RRT*, and APF (Artificial Potential Fields), we will now detail how each of these were implemented and how they work:

The A* Algorithm

This is a pathfinding and graph traversal algorithm commonly utilized in computer science and artificial intelligence to find the shortest path between two nodes in a weighted graph. It leverages the strengths of both Dijkstra's algorithm and best-first search to effectively determine optimal paths. Its optimality guarantees the discovery of the shortest path if the heuristic function is reliable. Its efficiency stems from its ability to combine actual values, thereby reducing the number of evaluated nodes compared to other algorithms. Furthermore, its flexibility allows it to adapt to different types of problems.

The implementation uses a mutable priority queue and a heuristic function, in this case of euclidean distance to the target.

The Dijkstra algorithm

Dijkstra's algorithm efficiently determines the shortest path from a single source node to all other nodes in a weighted graph, if the graph does not contain any negative edge weights. It works by iteratively selecting the node with the smallest tentative distance and updating distances to its neighboring nodes. This algorithm might be implemented through various data structures, including arrays, priority queues, or heaps.

It uses a mutable priority queue just like A*, but no heuristic function. It works as if we used A* with a heuristic function that always returned 0.

The D* algorithm

The D* algorithm is an incremental, heuristic-based pathfinding algorithm designed to find the shortest path in a changing, dynamic environment. Unlike traditional pathfinding algorithms such as Dijkstra's algorithm and A*, D* operates in a reverse manner, updating the path from the goal to the current position.

As our maps are not dynamic, it operates just like a Dijsktra but in reverse manner. Starting from the end, and stopping when finding the start node.

The Bidirectional Search algorithm

Bidirectional search is a graph search algorithm that works by simultaneously searching from the starting node and the goal node towards each other, aiming to meet somewhere in the middle. It is normally used in pathfinding algorithms to enhance efficiency, particularly in large search spaces. This algorithm provides some advantages when compared to some traditional unidirectional search algorithms, such as Dijkstra's algorithm or A*, due to the fact that it explores the search space from both ends. This approach will significantly reduce the time complexity of the search.

The implementation uses a Dijkstra's search from both ends looking for each other. The search stops, when the is guarantee that we cannot find a better solution.

The Rapidly-Exploring Random Tree algorithm-RTT

RTT is a probabilistic algorithm utilized in robotics for motion planning and pathfinding in high-dimensional, continuous spaces, it is valuable for generating feasible paths for robots o in complex, dynamic environments. It concentrates the search efforts on unexplored areas making it suitable for motion planning in complex, cluttered environments where traditional methods may encounter difficulties.

Until a solution is found, the algorithm generates a node in a random position and tries to connect it to the closest node to it. This stops when a created node is able to be connected to the tree and has direct access to the destination.

The Rapidly-Exploring Random Tree *

Rapidly-Exploring Random Tree Star is an extension of the RRT algorithm introduced to address its limitations. It amplifies path optimality by introducing a rewiring step and a cost-to-go function, maintaining efficiency. Widely used in robotics, RRT* is suited for motion planning and pathfinding in complex, dynamic environments.

The implementation is very similar to RRT, but includes a technique called 'Rewiring'. What this does is, every time we create a new node, we check if for the nodes around it, it would be closer to arrive to them by the original path, or by going to the new node and then moving to them.

The Artificial Potential Fields algorithm-APF

Artificial Potential Fields is a robotics technique for motion planning and obstacle avoidance. It involves creating virtual force fields around obstacles and goals to direct the robot's movement. APF is valued for its simplicity and effectiveness in navigating robots in dynamic environments. Despite its advantages, it can possibly encounter local minima issues, where the robot becomes stuck in suboptimal configurations due to the potential field structure. To mitigate these challenges, several extensions and modifications to APF have been proposed, enhancing its performance across various robotic applications.

At the start of the algorithm, we calculate the attractive and repulsive forces at every point in the map to generate the force field. When the robot arrives at any position, it moves to the direction of the force of that place.

3.2 Graph Building Algorithms

The Grid Method

The Grid Method establishes a grid structure by positioning nodes at each intersection point of rows and columns. Each node is then connected to adjacent nodes situated above, below, to the left, and to the right.

Probabilistic Roadmap

This planner consists of an offline roadmap construction and an online graph search for generating paths. It creates a roadmap of the configuration space by randomly sampling points and trying to connect each point to its nearby sample points using a local planner. Once the network is constructed, an online graph search is initiated after the initial and goal points are designated.

At the start, N collision free points are generated, then, using a K-D tree, edges are created between every node and the K closest nodes to it.

3.3 Path Pruning Algorithms

Line of sigth path pruning

LOS pruning is a method used to streamline a path by removing unnecessary waypoints, thus making the path more direct and efficient. The process begins with the first node (BNode) and the second node (ENode) of the initially generated path. The algorithm checks for collisions along the direct line between these two nodes. If no collision is detected, the ENode is moved to the next node, continuing this process along the path. If a collision is encountered, the node immediately before the collision is added to the pruned path, and this node is reassigned as the new BNode. The process then resumes from this new BNode, with the ENode set to the next node in the sequence.

This cycle continues until the entire path has been checked and pruned. Finally, the initial and final nodes of the original path are added to the pruned path, ensuring that the start and end points remain the same. The result is a significantly reduced number of nodes, transforming a potentially winding path into a series of straight lines. This simplification not only shortens the path length but also makes it easier for vehicles to follow, as there are fewer sharp turns and changes in direction, thus reducing the strain on vehicle actuators.

Global path pruning

Global path pruning enhances the LOS pruning method by addressing its shortsightedness, aiming for a more globally optimized path. Unlike LOS pruning, which stops pruning when a collision is detected between the BNode and ENode, global pruning considers the entire path. The process starts from the last node of the orginal path and moves backwards, examining whether there is a clear line of sight between the current node and a node further ahead, even if intermediate nodes have collisions. This method allows for skipping over intermediate collisions if the nodes further along are collision-free.

By considering the entire path rather than just local segments, global path pruning can eliminate unnecessary waypoints that LOS pruning might miss. This leads to a smoother, more efficient path with fewer abrupt changes in direction. Although this method slightly increases the processing time, it results in a path that is easier to follow and less likely to contain unnecessary detours, thus providing a better overall solution for path planning in autonomous systems.

4 Experimental evaluation

4.1 Experimental Setup

In order to assess which algorithm is the best for this kind of problem, we created maps that simulate both real life situations, such as a roundabout or mazes, and situations where certain algorithms will probably perform worse, like a tight corridor.

Using these maps, we then measured the time it took for each algorithm to calculate the route and compare the total distance of paths generated by all algorithms for each map, in order to obtain a relative measure optimization. To do this, for each map, the score of each algorithm on this metric is calculated by $\frac{shortest\_distance\_of\_any\_algorithm\_on\_that\_map}{distance\_of\_the\_algorithm\_on\_that\_map}$.

Using these two measures, we can then get the averages for each algorithm across each map, giving us a good way of comparison. It is also important to note that in all maps the robot starts close to the bottom left corner and has to get close to the top right in arbitrary coordinates equal in all maps.

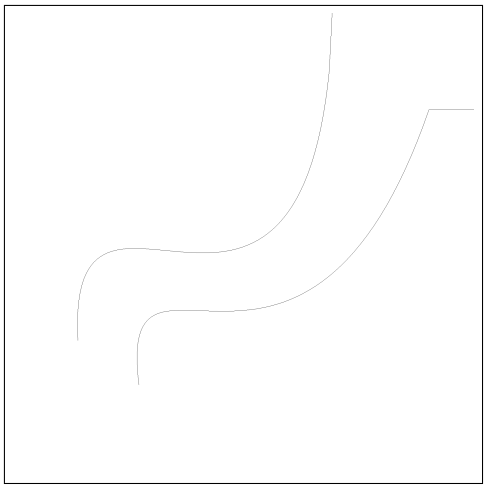


Fig. 1: Example Map

4.2 Results

Algorithm Comparison

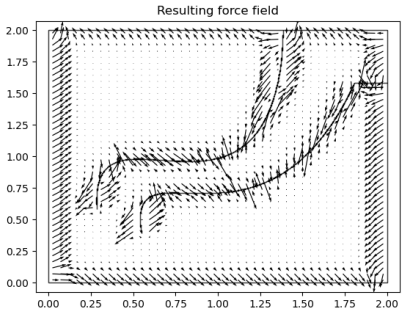
Algorithms	A*	Dijkstra	D*	Bidirect.	RRT	RRT*	APF
Distance Scores	0,72	0,72	0,72	0,72	0,91	0,87	Sol. Not Found
Timings (s)	0,001	0,001	0,001	0,001	188,35	485,021	Sol. Not Found

Here we already see that all the search algorithms have similar scores, although the RRT based algorithms that generate their own paths seem to perform better by themselves. Also, since the APF algorithm didn't find the solution in most maps, we opted to not evaluate it any further due to this lack of data.

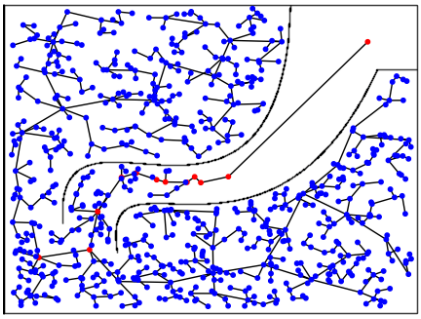
Pruning Methods Comparison

Distances	A*	Dijkstra	D*	Bidirect.	RRT	RRT*
Original	0,72	0,72	0,72	0,72	0,91	0,87
Loss	0,98	0,98	0,98	0,98	0,94	0,97
Global	0,99	0,99	0,99	0,99	0,96	0,95
Loss Gain	0,26	0,26	0,26	0,26	0,03	0,1
Global Gain	0,27	0,27	0,27	0,27	0,05	0,08

On average, the global pruning method has a gain of 0,202 in distance score, whilst the loss method only has a gain of 0,195.



(a) APF Force Field



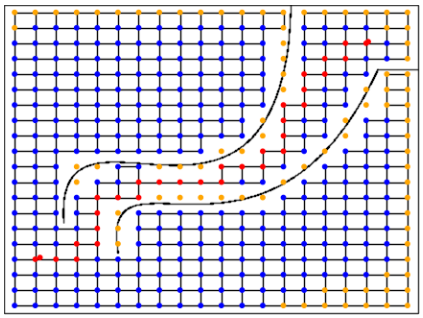
(b) RRT Path

Fig. 2: Example Algorithms

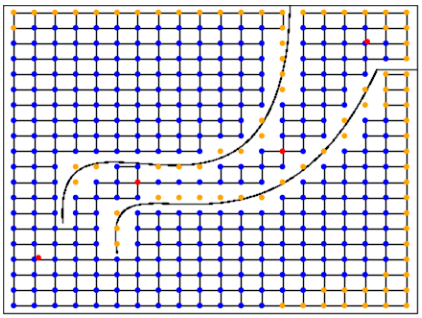
Times (s)	A*	Dijkstra	D*	Bidirect.	RRT	RRT*
Original	0,001	0,001	0,001	0,001	188,35	485,021
Loss	19,063	19,455	27,46	22,5	197,684	491,687
Global	27,68	34,519	33,017	36,351	204,684	499,35
Loss Extra Time	19,062	19,454	27,459	22,499	9,334	6,666
Global Extra Time	27,679	34,518	33,016	36,35	16,334	14,329

The global pruning method takes on average 17,4 seconds more to compute, compared to no pruning done, while the loss method has a bigger delay of 27 seconds, on average.

Bellow, we can see how the global pruning algorithm eliminates several nodes (nodes kept appear in red) from the original path generated by the A* algorithm.



(a) A* without pruning



(b) A* using global pruning

Fig. 3: Pruning Comparison

Graph Building Algorithms Comparison

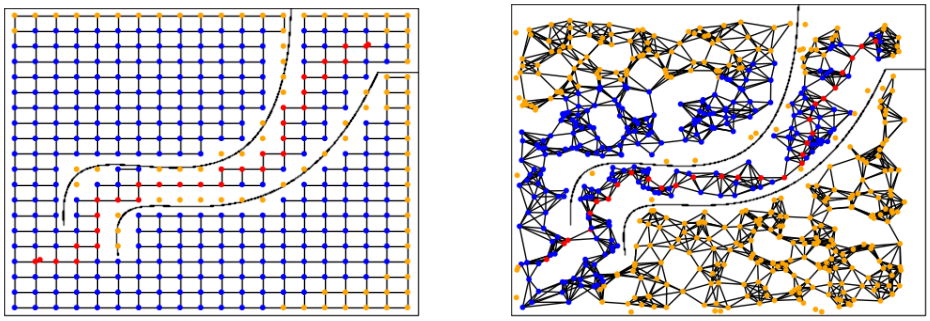
When evaluating the difference in the graph building algorithms used, all algorithms with no pruning all revealed to have an insignificant upgrade in distance scoring (0,06) when using PRM instead of the grid, even though all the timings measured stayed the same.

Distances	Grid	PRM	PRM Difference
A*Loss	0,98	0,94	-0,04
A* Global	0,99	0,97	-0,02
Dijkstra Loss	0,98	0,94	-0,04
Dijkstra Global	0,99	0,97	-0,02
D* Loss	0,98	0,94	-0,04
D* Global	0,99	0,98	-0,01
Bidir. Loss	0,98	0,94	-0,04
Bidir. Global	0,99	0,98	-0,01

Although the PRM based searches didn't affect unpruned routes, as these get bigger the PRM algorithm will most likely generate a graph that doesn't allow such optimal route connections as in the grid one.

Times (s)	Grid	PRM	PRM Difference
A*Loss	19,063	19,552	0,489
A* Global	27,68	30,684	3,004
Dijkstra Loss	19,455	20,319	0,864
Dijkstra Global	34,519	33,361	-1,158
D* Loss	27,46	20,593	-6,867
D* Global	33,017	33,691	0,674
Bidir. Loss	22,5	22,046	-0,454
Bidir. Global	36,351	35,026	-1,325

When it comes to timings, some algorithms benefit from having PRM graphs, although these margins gained can possibly be justified by having a lucky graph for the maps where the algorithm was evaluated. Even though we see a slight upgrade in general while using PRM graphs, the grid graphs offer more consistency with the scaling of the graph sizes.



(a) A* with grid graph (b) A* with PRM graph

Fig. 4: Grid Algorithms Comparison

5 Conclusions and Future Work

In conclusion, our results demonstrate that each of the graph search algorithms, including A*, Dijkstra, D, and Bidirectional, exhibit varying degrees of effectiveness depending on the environment. Significantly, in grid-based scenarios, these algorithms emerge as more reliable. Furthermore, it is evident that the integration of global pruning techniques significantly enhances their performance across different settings. This highlights the importance of considering the inherent characteristics of the environment when selecting the most suitable algorithm. Our findings suggest that with the strategic implementation of global pruning, these algorithms are able to achieve improvements in both efficiency and reliability, thereby affirming their practical applicability in real-world scenarios.

When discussing potential future directions and suggestions, we propose retesting our approach, this time incorporating dynamic environments. By doing so, we would be able to explore the adaptability and complexity of our approach under changing conditions. This re-evaluation would let us evaluate the performance and effectiveness of our solutions in more realistic scenarios, providing valuable insights for further refinement and potential improvements.

6 Bibliography

The following bibliography provides a thorough overview of the various type of information repositories consulted during this research project. The selected links will show some sources that have contributed to a nuanced understanding of the topic at hand.

Robotic Path Planning: RRT and RRT*

Motion Planning Algorithm RRT star(RRT*) Python Code Implementation, with Obstacle

Probabilistic Roadmap (PRM) for Path Planning in Robotics

Motion Planning and Controls

The D*, D* Lite and LPA* Algorithm

Anytime synchronized-biased-greedy rapidly-exploring random tree path planning in two dimensional complex environments

The codes for most of the implemented algorithms, as well as the creation of the maps, were provided by Professor Gonçalo Leão. This professor was an invaluable help throughout the entire project, offering his support and expertise whenever we encountered problems, and ensuring the success of the project with his valuable contributions.