Comparison of search strategies for path planning of mobile robots

Jonas Brodmann
Valentin Fuchs
Pablo Castellanos
Instituto Superior Técnico, Lisboa, Portugal

February 2022

Abstract

This paper deals with the comparison of search algorithms for path planning of autonomous moving vehicles. It aims at giving an overview on different popular search strategies and shows exemplary in a simulation environment the strength and weaknesses of A*- and Rapidly-exploring random tree algorithms.
Keywords: search strategies, path planning, mobile robots, path search, A* (A-star), RRT

1. Introduction

Path planning is an essential task in the control of mobile robots. Regardless if it is locally, in a fixed space operating vacuum robot or a globally shipped autonomous car: All these robots have to find a route through their environment in order to fulfil their task. The robot therefore must be able to autonomously maintain the information about its environment and plan a path, avoiding obstacles, taking into account its own limitations (kinematic and dynamic) and most of the time respecting a kind of optimization criterion. According to [12] the planning algorithms as well as the representation of information about the environment have a main influence on time complexity, space complexity and path optimality, showing the importance of selecting the right path planning algorithm [4].

Search algorithms provide an efficient way of obtaining a path in compliance with the previously mentioned conditions. This paper will provide an overview of the most common search strategies used for mobile robots, compare them and point out advantages and disadvantages of the presented strategies. Furthermore, the presented strategies are investigated using a python implementation [1] and executing the algorithms on an exemplary maps, showing strengths and weaknesses of the solutions. The work will focus on global path planning, meaning that all the information about the environment is accessible for the robot before starting to execute the path [15].

2. Search strategies

Generally, a path planning algorithms should have the following properties [9]:

1. “The resulting paths should have the lowest possible cost to prevent any indirection”. [9]

2. “It should be fast and correct to not thwart the simulation process, for example it should be robust if there is no collisions occurred during the navigation process”. [9]

3. “The algorithm should be generic with respect to different maps where its means that it should not be fully optimized for a specific map type”. [9]

Search marks only one possibility on the numerous path planning options for mobile robots (for more information see [15]). However, search algorithms have advantages when it comes to understandability how a certain solution is maintained, implementation complexity and optimality. Source [11] gives a current overview on publications on path planning, showing that out of 18 publications 11 publications deal with or at least mention graph search as a solution for global path planning.

In figure 1 an overview of the different path planning methods is given[1].

[1] Please note that Rapidly-Exploring Random Trees (RRT) are missing in this overview

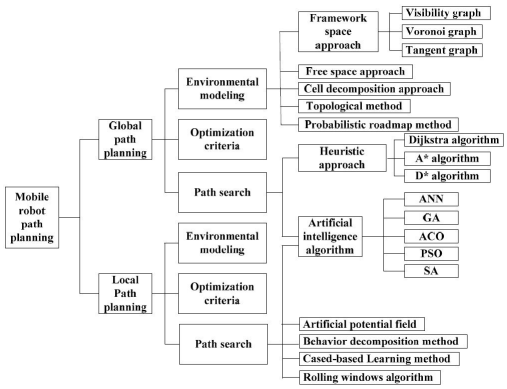


Figure 1: Overview Pathplanning Algorithms [15]

The literature differentiates between informed and uninformed search. For path planning, usually informed search methods are used, taking e.g. the euclidean distance to the goal state as a heuristic function in consideration. Generally, informed search algorithms given a well-defined heuristic function, outperform uninformed search algorithms not only regarding the time complexity but also the space complexity [6].

The following subsections discusses different search algorithms used in path planning. Those search algorithms are often referred to by C-Space-Search-Based path planning algorithms, as they take into consideration all the states (or configurations) the robot can reach, using a discrete state space for the search [6], [11], [15].

2.1. Dijkstra Algorithm

The Dijkstra algorithm is used to find the shortest path between two nodes of a graph (the start and the goal node). It iteratively visits every neighbor node of already visited nodes and collects information about the costs of the pathes. Because the algorithm passes many nodes to obtain the information about the path costs, the efficiency is not high [15], [5].

2.2. A* Search

The A*-graph search[2] algorithm was developed on the basis of the Dijkstra algorithm as a heuristic version of it. It uses a composite evaluation function to determine which node should be expanded. This evaluation function consists of a heuristic function and a cost function which are summed up. The heuristic function determines whether a node is closer or far from the goal node [6], [11], [15].

2.3. D* Search

While the A* search algorithm is mainly used for static environments, the D* (or dynamic A*) algorithm takes in consideration changes in the world the robot is operating in. This algorithm is not using a static environmental representation in the beginning but starts by planning the shortest path making assumptions about unknown parts of the world. The robot is then adjusting this internal world representation if new information about the map is obtained while executing the path. It then calculates a new path during the runtime taking the new information into account [14].

[2]Graph search algorithm in contrary to tree search algorithm have a closed- and open list, so that already visited nodes can be stored and don't have to be checked twice

2.4. Rapidly-Exploring Random Trees

Originally developed for efficiently searching high dimensional space, Rapidly-Exploring Random Trees (here short RRTs, a special case of Rapidly Deterministic Trees) became very popular in path planning in the last years. First introduced in [8] RRTs offers a possibility to iteratively, uniformly expanding a search tree, by explicitly handling nonholomic and kinodynamic constraints. This tree search first selects a random place in the search environment and then expands the closest node in this direction, "pulling" the tree towards the selected point. The node expansion is only rejected if any predefined constraint is violated (e.g. the path would end up in a building instead of the road). The tree expansion stops, if the expanded node is "close enough" to the goal node. Therefore a condition has to be obtained, specifying what "close enough" means. The path then results from applying a backtrack search [11]. There are slightly improved versions like the RRT*, creating shorter path when adding more nodes and therefore providing solutions that are closer to the optimal path [7].

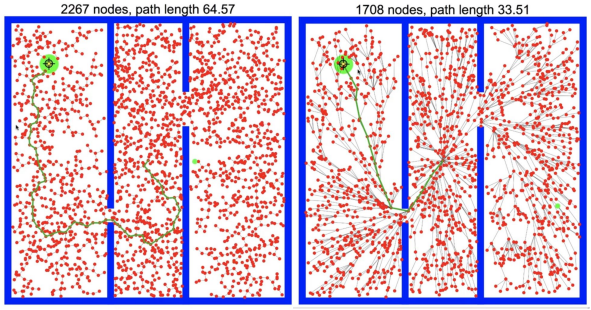


Figure 2: Solution RRT (left) and RRT* search (right) [3]

Table 1 sums up the properties of graph search algorithms (like the A* algorithm) and sampling based algorithms (like the RRT).

3. Comparison of A* and RRT

The paper will now show an exemplary implementation of a RRT and A* algorithm, calculating a path for an autonomous car through an urban area (as a static environment is used, D* won't be dis-

Table 1: Characteristics of C-Space Search (excerpt from [11])

Sub Category	Preliminary Model	Deterministic	Replanning	Optimality	Completeness
Graph Search	Yes	Yes	Only incremental	Global restricted to graph	Yes
Sampling based	No	No	Yes	Asymptotical	Probabilistic

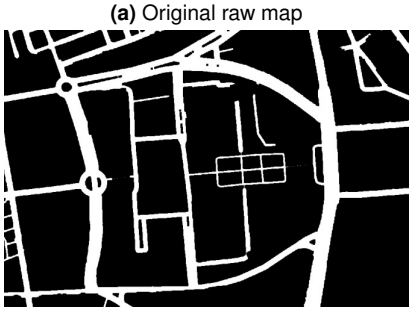
cussed in this paper). Dijkstra algorithm won't be discussed as A* is considered as an improvement of Dijkstra (see [15]). The algorithms performance will be evaluated on a map of the Alameda Campus of the Instituto Superior Técnico in Lisbon.

3.1. Test setup

Map

The maps used for the path planning are greyscale maps obtained by using the Freeware Google Maps wrapper *Snazzymaps*[2]. Those maps then were turned into a binary image by applying a threshold, resulting in an image where roads have the value "1" and obstacles are left with the value "0". The result of this process is shown in the below figure:

(a) Original raw map



(b) Binary map

Figure 3: Computing binary map

Basic A* Search

The A* search algorithm was manually implemented, as there is no python library available containing the A* algorithm. In order to discretize the search, the pixels of the image were chosen as states and the eight moving directions around the pixel were chosen as actions applicable in every state. The evaluation function used for the A* search is given below, where n is the next node on a plausible path, $g(n)$ is the cost of the path up until that node and $h(n)$ is the estimated distance, or *heuristic*, from n to the target-node.

$$f(n) = g(n) + h(n) \quad (1)$$

Improved A* Search

To make sure, that the planned path stayed in the middle of the road and a car wouldn't move too close to the buildings or other obstacles, a gradient map was introduced (see figure 4) and the evaluation function slightly modified.

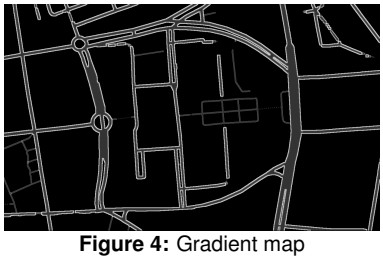


Figure 4: Gradient map

$$f(n) = g(n) + \text{gradient\_value}(n) * h(n) \quad (2)$$

Basic RRT

RRT algorithm has been implemented to perform a random tree along the map, the expansion points are spreaded randomly to avoid getting stuck somewhere on the map likewise, the node expansion step are variable. The main loop will finish when the distance from the last node to the end position is less than certain value.

Algorithm 1 Basic RRT with dynamic step

Require: binary_map, start_pos, end_pos
Ensure: $start\_pos \neq end\_pos$
  $goal = false$
  $max\_distance\_goal = N_{distance}$
  $step = N_{step}$
  $tree = \{\}$ ▷ tree{(node:father_node)}
  $iterations = 0$
  while $goal == false$ **do**
    $rand\_pos = random\_pos(binary\_map\_size)$
    $tree = extend\_tree(binary\_image, rand\_pos, step)$
    $d\_goal = \sqrt{(node\_x - end\_pos\_x)^2 + (node\_y - end\_pos\_y)^2}$
    if $iterations \% N_{iterations}$ & $step \neq 1$ **then**
      $step = step - 1$
    end if
    if $d\_goal <= max\_distance\_goal$ **then**
      $goal = true$
    end if
    $iterations = iterations + 1$
  end while
  $path = get\_path\_tree(tree, start\_pos, end\_pos)$
  return $path$

Improved RRT

Two different ideas were discussed to improve the time and space complexity of a basic RRT algorithm:

- Start the search with 2 or more trees parallelly and merge them, e.g. one expansion starting from the goal and the other from the root node
- Divide the expansion into two parts:
  1) Randomly spread the expansion points (picking the expansion direction randomly) to cover the biggest area in the shortest period of time.
  2) After some iterations pick the points for obtaining the trees expansion direction by applying a Gaussian on the known goal point

As merging two trees is a complex task, the improved RRT algorithm discussed in this paper is an implementation of the second approach. The used Gaussian is mean free and has a standard deviation of 250 pixels.[3] The standard deviation is dynamic and depending on the numbers of iterations:

```
if iter%100 == 0 and sigma != 1 and iter>=200:
sigma-=10
```

3.2. Evaluation criteria

The algorithms were compared in respect to the following criteria:

- computational time
- memory complexity
- characteristics of the obtained path (e.g. smoothness, no too sharp corners, no constraints are violeted ...)

Therefore all four algorithms were performed on the given map. The path used for the evaluation is shown in figure 5.

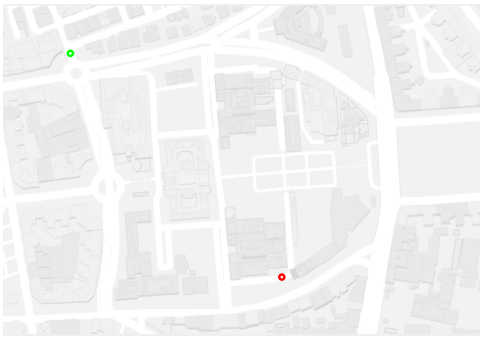


Figure 5: Path used for the evaluation of the searches

[3]Both, the standard deviation and the decreasing factor are optimized for this specific example (so: this JPG-file of the map) and would have to be adjusted for other maps.

4. Results & Discussion

4.1. Basic A* Search

The path obtained by the basic A* search is an optimal path, meaning that the A* search algorithm always showed the shortest path between two given points on the IST-map. Unfortunately the path stayed very close to the obstacles on the map, if the closest existing path would be placed near those obstacles. As a car would not be able to move that close to the buildings, an improvement had to be made, to keep the path in the middle of the streets. Therefore a gradient map was computed and used to improve the A* search. As the path and the computational time (the gradient just goes in as a factor into the evaluation function) does not differ much between the improved A*-search and the basic A* search, results on computational time and space complexity are only discussed for the improved A* algorithm.

4.2. Improved A* Search

The path created by the improved A* search algorithm is shown in figure 6. This path is still optimal (given the newly introduced constraints), but now shifted to the middle of the road.

Figure 6: Obtained path by the improved A* search

The performance of the A*-algorithm is shown in figure 7. The

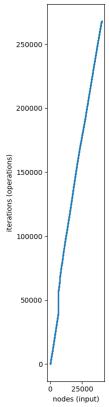


Figure 7: Performance of A* algorithm

The graph above represents the number of nodes (input) given the number of iterations that A* needs to compute all nodes in each expansion. The complexity of the shown algorithm is $O(n \log(n))$. The number of nodes evaluated is about 25.000.

Note that for executing this algorithm, a gradient map has to be computed. This is not included in the plots showing the performance of the improved A* algorithm. Depending on how many pixels are contained in the map, computing this map during the runtime or always before running the algorithm will increase the overall path-computation time significantly.

4.3. Basic RRT Search

The following plot shows the path obtained by the implemented RRT algorithm.

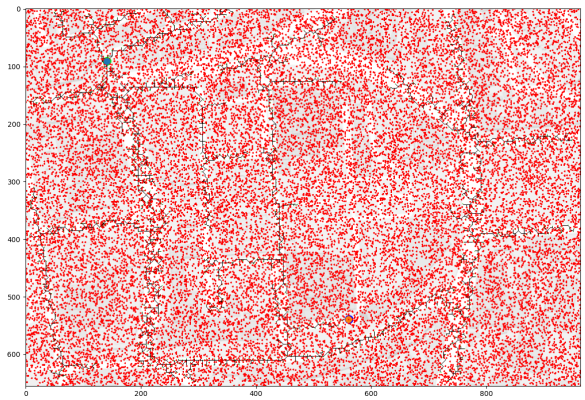


Figure 8: Obtained path by the basic RRT search

The red dots mark the random points the algorithm created to define a new tree expansion direction.

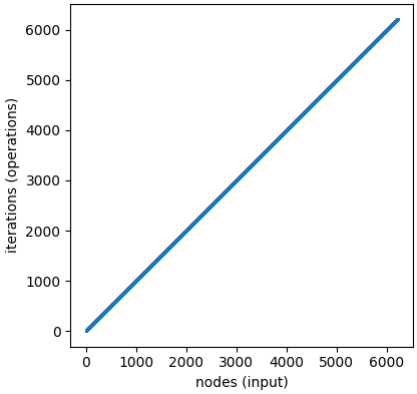


Figure 9: Performance of the basic RRT algorithm

Figure 9 shows that the interdependence between number of iterations and number of nodes expanded is linear, which means that for a longer path the computational time would increase linearly to the number of nodes expanded. That makes sense, as the program has to check the distance of each expanded not to the randomly created point, in order to find the closest one.
It takes the basic RRT over 6000 nodes to expand to find the goal node[4]. A high space and time complexity was one major issue, that was addressed through the improved RRT algorithm developed in this project. Note that the plot only includes the number of iterations necessary to find the goal node. After finding the goal node, the backtracking search in the program developed works with a hash table that contains every parent of an expanded node. As the costs for accessing the hash table is constant, the additional computational time of the backtracking search is directly related to the number of nodes contained in the final path.

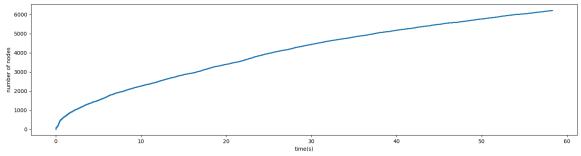


Figure 10: time consumption of basic RRT

4.4. Improved RRT Search

As already described previously in subsection 3.1, the method to improve the space and time complexity of the algorithm, relies on the introduction of the Gaussian. The path obtained by this method is shown in figure 8. The map already gives an idea, that fewer points and iterations were needed to compute the path with the improved algorithm. It also shows how the Gaussian ensures more points close to the goal node and therefore supports a faster convergence. In average, the convergence therefore is faster than for the basic RRT algorithm.

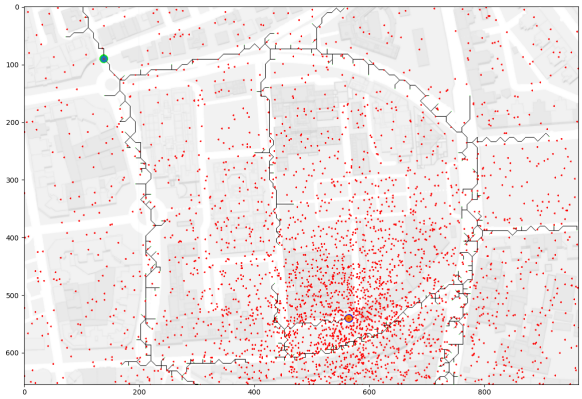


Figure 11: Obtained path by the improved RRT search

The performance measures for the given map are depicted in figure 12.

[4]This number is highly dependent on how the random points are set. In some tests the basic RRT even outperformed the improved RRT search, when the points were randomly set points complimented the tree search algorithm in its expansion. So you could say, it is dependent on how ”lucky” points were set.

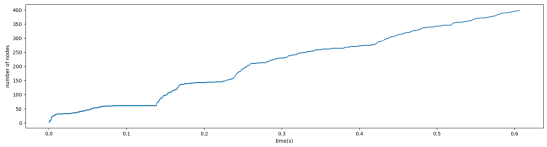


Figure 12: Time consumption of improved RRT

The improved RRT algorithm only needed around 0.6 seconds to successfully find a path (in comparison: the basic RRT needed around 60 seconds), while the relation between expended nodes and number of iterations remains linear. To recap, the complexity of both basic RRT and improved RRT is $O(n)$ One downside that can be noted is, that the path is not optimal and also contains sections with strong changes of the gradient. This would mean, that an actual autonomous car could not or could only follow this path with difficulty, which would make some post-processing (e.g. filtering) necessary to smoothen the path.

5. Conclusions

The experiment showed, that the A*-search finds the optimal path but is computational expensive. The number of nodes expanded and the iterations needed to obtain the solution is higher than for the improved RRT algorithm. Possible improvements to decrease the computation costs for a A* search are e.g. discussed in [13]. Another downside is, that it is not possible to introduce additional constraints like e.g. kinodynamic information about the car into the search, to prevent the search from creating a path that the car is unable to follow.
The RRT is less expensive in respect to computation and therefore offers an interesting alternative to the A*-search, especially for bigger maps. Additionally, it is explicitly possible to introduce kinodynamic constraints into the search algorithm (good results of such an approach shows e.g. [10]). Additionally, the RRT* algorithm (introduced in [7]) already helps to overcome the optimality problem of the RRT algorithm, optimizing the path with each iteration even after a sufficient path was found. In summary, it can be said that it strongly depends on the requirements of the task which search algorithm should be chosen. Engineers should therefore consider the computational capabilities of the hardware available and the demands regarding the planned path.

References

[1] Github repository. https://github.com/P4B5/Path_planning_algorithms.

[2] Raw maps without labels. https://snazzymaps.com/style/55/subtle-greyscale-map. Accessed: 2022-01-28.

[3] A. Becker. Rrt, rrt* & random trees, 2018.

[4] O. Castillo, L. Trujillo, and P. Melin. Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots. *Soft Computing*, 11(3):269–279, 2006.

[5] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[6] W. Hidayat, F. Susanti, and D. R. Wijaya. A comparative study of informed and uninformed search algorithm to solve eight-puzzle problem. *Journal of Computer Science*, 17(11):1147–1156, 2021.

[7] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems VI*. Robotics: Science and Systems Foundation, 2010.

[8] S. M. LaValle. *Rapidly-exploring random trees: A new tool for path planning*. 1998.

[9] C. Niederberger, D. Radovic, and M. Gross. Generic path planning for real-time applications. In *Computer graphics international*, pages 299–306. IEEE, 2004.

[10] R. Pepy, A. Lambert, and H. Mounier. Path planning using a dynamic vehicle model. In *2006 2nd International Conference on Information & Communication Technologies*, pages 781–786. IEEE, 24-28 April 2006.

[11] J. R. Sánchez-Ibáñez, C. J. Pérez-Del-Pulgar, and A. García-Cerezo. Path planning for autonomous mobile robots: A review. *Sensors (Basel, Switzerland)*, 21(23), 2021.

[12] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 Student Conference on Research and Development*, pages 183–188, Piscataway NJ, 2003. IEEE.

[13] N. Sariff and N. Buniyamin. An overview of autonomous mobile robot path planning algorithms. In *2006 4th Student Conference on Research and Development*. IEEE, 2006.

[14] A. Stentz. The d*algorithm for real-time planning of optimal traverses.

[15] H.-y. Zhang, W.-m. Lin, and A.-x. Chen. Path planning for the mobile robot: A review. *Symmetry*, 10(10):450, 2018.