Mobile Robot Navigation on Partially Known Maps using a Fast A* Algorithm Version

Paul Muntean
Technical University of Munich, Germany
paul.muntean@in.tum.de

Abstract—Mobile robot navigation in total or partially unknown environments is still an open problem. The path planning algorithms lack completeness and/or performance. Thus, there is the need for complete (i.e., the algorithm determines in finite time either a solution or correctly reports that there is none) and performance (i.e., with low computational complexity) oriented algorithms which need to perform efficiently in real scenarios.

In this paper, we evaluate the efficiency of two versions of the A* algorithm for mobile robot navigation inside indoor environments with the help of two software applications and the Pioneer 2DX robot. We demonstrate that an improved version of the A* algorithm which we call the \textit{fast} A* algorithm can be successfully used for indoor mobile robot navigation. We evaluated the A* algorithm first, by implementing the algorithms in source code and by testing them on a simulator and second, by comparing two operation modes of the fast A* algorithm w.r.t. path planning efficiency (i.e., completeness) and performance (i.e., time need to complete the path traversing) for indoor navigation with the Pioneer 2DX robot. The results obtained with the \textit{fast} A* algorithm are promising and we think that this results can be further improved by tweaking the algorithm and by using an advanced sensor fusion approach (i.e., combine the inputs of multiple robot sensors) for better dealing with partially known environments.

I. INTRODUCTION

Motion planning—also known as the navigation problem or the piano mover’s problem—is a term used in robotics for the process of breaking down a desired movement task into discrete motions that satisfy movement constraints and possibly optimize some aspect of the movement. Motion planning has several robotics applications, such as: (i) robot navigation, (ii) automation, (iii) the driver-less car, (iv) robotic surgery, (v) digital character animation, (vi) protein folding, (vii) safety and accessibility in computer-aided architectural design, (viii) UCAV Path Planning [19], etc.

A basic motion planning problem is to produce a continuous motion that connects a start configuration S and a goal configuration G, while avoiding collision with known obstacles. The robot and obstacle geometry is described in a 2D or 3D workspace, while the motion is represented as a path in (possibly higher-dimensional) configuration space (describes the pose of the robot, and the configuration space C is the set of all possible configurations).

Problem statement: Recent studies show that every day activity of people in cities and countries living in the modern society is rapidly increasing [18] in such a way that efficient navigation of people movement is needed. Researchers have tried to come with new and better navigation approaches in the past as for example Jones [8]. These approaches lack efficiency or applicability to mobile robot navigation in real path planning environments.

Available solutions: Path planning w.r.t. low-dimensional problems can be addressed using: (i) grid-based approaches which overlay a grid on a configuration space and assume that each configuration is identified by a grid point. At each grid point, the robot is allowed to move to adjacent grid points as long as the line between them is completely contained within \( C_{\text{free}} \) (the set of configurations that avoids collision with obstacles is called the free space \( C_{\text{free}} \) (this is tested with collision detection), (ii) interval-based search which is similar to grid-based search approaches except that they generate a paving covering entirely the configuration space instead of a grid [1], (iii) geometric algorithms which are used to point robots among polygonal obstacles based on a visibility graph, cell decomposition and translating objects among obstacles using the Minkowski sum [13], (iv) potential fields which are used to treat the robot’s configuration as a point in a potential field that combines attraction to the goal and repulsion from obstacles. The resulting trajectory represents the new path which is computed fast. However, they can become trapped in local minima of the potential field, and fail to find a path, (v) sampling-based algorithms which represent the configuration space with a road-map of sampled configurations. A basic algorithm samples \( N \) configurations in \( C \), and retains those in \( C_{\text{free}} \) to use as milestones. A road-map is then constructed that connects two milestones P and Q if the line segment PQ is completely in \( C_{\text{free}} \). Most notable algorithms are the A* and D* algorithms which can rapidly explore random trees and probabilistic road-maps.

A motion planning algorithm is said to be complete if the planner determines in finite time either a solution or correctly reports that there is none. Most complete algorithms are geometry-based. Resolution completeness is the property that the planner is guaranteed to find a path if the resolution of an underlying grid is fine enough. Most resolution complete planners are grid-based or interval-based. Probabilistic completeness states that, as more “work is performed, the probability that the planner fails to find a path (if one exists) asymptotically approaches zero. The performance of a probabilistically complete planner is measured by the rate of convergence. Incomplete planners do not always produce a feasible path when one exists. The performance of a complete planner is assessed by its computational complexity computed using the big \( \mathcal{O} \) notation.

Deficiencies of available solutions: In summary, existing path planning algorithms lack in determining a path when one
exists or they need to much time to compute one. Thus, the main limitations of these algorithms are related to completeness and/or performance. Thus, in this work we seek for a suited robot path planning algorithm which is complete and performant.

**Our idea:** Our insight is that an improved A* algorithm (we call this the fast A* algorithm) can be efficiently used for path planning of real robots in a partially known environment. We evaluated two versions of the A* algorithm and presented the results obtained with the Pioneer 2DX robot [13]. The communication (closed loop) between our PC and the real robot was achieved by sending real-time navigation commands via a wireless connection based on the Lantronix WiBox [11]. Note that during the experiments the Pioneer 2DX robot used only the ultrasonic sensors in order to partially reconstruct a map of the partially known (containing unknown obstacles) environment—not mapped on the initial on-line mode navigation map.

In this paper, we address the problem of efficient and complete motion planning of a three wheeled mobile robot by implementing two algorithms (the A* algorithm and the fast A* algorithm) and comparing the efficiency (with focus on completeness and performance) of this two approaches on a path planning algorithm simulator and afterwards with the real Pioneer 2DX robot.

**Our contributions:** In summary, the main contributions are:

- We develop an improved version of the A* algorithm which proves to be faster in offline testing (with a software simulator) and efficient in real environments when tested with the real Pioneer 2DX robot.

- We implement two applications: first, a simulator used for path planning simulation in offline mode (not with the real robot) and assessed the performance and completeness of the A* algorithm and of the fast A* algorithm and second, a path planning application used in online-mode (with the Pioneer 2DX mobile) to navigate him through a partially known map using only the fast A* algorithm in two different operation modes.

- We demonstrate that the fast A* algorithm is effective when tested with the Pioneer 2DX mobile robot inside a partially known indoor environment.

The remainder of this paper is organized as follows. Section II highlights background work. Section III presents the A* algorithm. Section IV highlights implementation details. Section V depicts experiments results. Finally, in Section VI we conclude and present future work.

**II. BACKGROUND**

**A. Brief Routing History**

In the 1970 scientists started research on routing algorithms for moving chess pieces on a chess-board and on how to efficiently move fragments on a puzzle map [5]. As a consequence the research on routing algorithms started. The main reason for starting the research in the area of routing algorithms was that these problems can be easily abstracted and further on the results can be applied to more complex fields of study such as robot navigation. Thus, with the development of path finding, several new classical routing algorithms have emerged at that time with the goal to generate better routing results.

The Dijkstra algorithm is the most famous algorithm. The algorithm evaluates the moving cost from one node to any other node and sets the shortest moving cost as the connecting cost of two nodes [5]. Around the same period the Best First Search (BFS) algorithm was introduced. The BFS is different from the Dijkstra algorithm, since the BFS estimates the distance from the current position to goal position and it chooses the next step that is more closer to the goal position [1].

As the complexity of the path finding scenarios was growing the path finding algorithms had to be improved in order to meet new requirements as for example 3D maps.

**B. The A* Algorithm and Extensions**

As response to the new path planning requirements the A* algorithm appeared. The goal of the new A* algorithm is path planning efficiency. The A* algorithm is a BFS algorithm which uses huge amounts of memory in order to keep track of the data related to the current proceeding nodes [14]. The A* algorithm tries to combine the advantages offered by the Dijkstra algorithm and the BFS algorithm. The A* algorithm tries during each new movement to take the shortest step and tries to determine if the step lies on the direction from source to target [8]. The disadvantage of the A* algorithm is that it uses large amounts of memory in order to store the path planning environment.

The A* algorithm proved to have its limitations and in response new methods of using the A* algorithm appeared. The bidirectional A* algorithm [14] is used in order to reduce the time cost of the A* algorithm. The most important difference of the bidirectional A* algorithm w.r.t. the classical A* algorithm (which is searching from the source to the target location) is that it can search from source to target and vice-versa. The path search stops immediately when the two directional searching processes meet each other.

The Iterative Deepening A* (IDA*) [9] is a space-efficient version of the A* algorithm, which suffers from cycles in the search space (it uses no storage), repeated visits to states (the overhead of iterative deepening), and a simplistic traversal of the search tree. Since it is a depth-first search algorithm, its memory usage is lower than in A*, but unlike ordinary iterative deepening search, it concentrates on exploring the most promising nodes and thus does not go to the same depth everywhere in the search tree. Unlike A*, IDA* does not utilize dynamic programming and therefore often ends up exploring the same nodes many times [6].

Routing in three dimensions (3D) is much more complex than under two space dimensions, thus the traditional A* algorithm should be improved in order to meet the additional routing requirements. The three dimensional A* algorithm has emerged as a response for better dealing with 3D environments. The three dimensional A* algorithm was obtained by adding several modifications to the A* algorithm in order to

---

1 Demo movie available: https://goo.gl/OYXMDy
be used for computing navigation paths in 3D maps (e.g., the path planning of a cart in a mine system which has multiple levels).

Furthermore, a frequently used approach for solving simple three-dimensional path planning problems is to map the three-dimensional map into a two-dimensional expression. In this way the traditional A* algorithm can be used for solving path planning [12] in 3D environments. Note that this technique of mapping 3D maps to 2D maps is working for path planning in simple 3D scenarios—reduced set of constraints. In complex scenarios this mapping method can not be used and thus more complex approaches are needed.

III. THE A* ALGORITHM

In this section, we briefly describe the main parts of the A* algorithm. The A* algorithm [3] uses the BFS algorithm in order to find the least-cost path from a given initial node to one goal node (the last position could be a single or multiple nodes). It uses a distance-plus-cost heuristic function (usually denoted by $h(x)$) to determine the order in which the search visits nodes in the node tree. The distance-plus-cost heuristic $f(x)$ is expressed as sum of two functions: (a) the path-cost function, represents the cost from the starting node to the current node (usually denoted $g(x)$) and (b) an admissible “heuristic estimate” used to model an estimated from the current position/node to the the goal position/node (usually denoted with $h(x)$). The distance-plus-cost heuristic function can be framed as follows.

$$f(x) = g(x) + h(x)$$  \hspace{1cm} (1)

An important constraint is that the $h(x)$ component of $f(x)$ must be an admissible heuristic—briefly this means that it is important to not overestimate the distance from current node to the goal node. The $g(x)$ component of $f(x)$ represents the total cost from the start node and not only the cost from the previously expanded (visited) node. In case of determining the shortest distance between two locations (nodes) it is known that the straight line is the shortest distance. In case of routing, $h(x)$ could be represented as a straight-line distance from current position to the goal position. Next we impose the following constraint on $h(x)$.

$$h(x) \leq d(x, y) + h(y)$$  \hspace{1cm} (2)

The mathematical expression (2) imposes that every edge represented by $x$ and $y$ belonging to a graph where $d(x, y)$ represents the length of the given edge results in an $hx$ which is consistent or monotone. Furthermore, (2) guarantees that one node is processed only once and in this case the implementation of the $A^*$ is more efficient. In this case running the $A^*$ algorithm is similar to running the Dijkstra’s algorithm having the cost reduced. Next we impose the following constraint on the length of a graph edge.

$$d'(x, y) = d(x, y) - h(x) + h(y)$$  \hspace{1cm} (3)

The $A^*$ algorithm is an informed search algorithm. A particularity of informed search algorithms is to search for the routes (paths) that appear to be most likely to lead to the goal position. Note that the $A^*$ algorithm differs from the greedy best-first search algorithm because it takes into consideration the already travelled distance. The process of finding the path from a starting position to a target position by using the $A^*$ algorithm is repetitive and ends when the current visited node is equal to the target node or when the target position is reached. During graph nodes traversing the $A^*$ algorithm follows a path from the lowest known path based on keeping a priority queue of all alternate path segments along the path. When an edge of the path is traversed which has a higher cost than another previously encountered path segment then it immediately abandons the current path segment (having higher cost) and continues with the lower-cost path segment.

IV. THE FAST A* ALGORITHM IMPLEMENTATION

In this section, we first, present implementation details of our fast A* algorithm and second, we present implementation details of our two tools.

A. The Fast A* algorithm Implementation

Listing 1: The $A^*$ algorithm—informal description

0. initialize the open list
1. initialize the closed list
2. initialize goal node // this is the target node
3. initialize start node // add the node to the open
4. while open list is not empty {
5. get node $n$ from the open list with the lowest $f(n)$
6. add $n$ to the closed list
7. if $n$ is equals the goal node then stop;
8. return solution;
9. generate each successor node $n'$ of $n$;
10. for each successor node $n'$ of $n$ {
11. set the parent of $n'$ to $n$;
12. // heuristic estimate distance to goal node
13. set $h(n')$
14. set $g(n') = g(n) + cost$ from $n$ to get to $n'$
15. set $f(n') = g(n') + h(n')$
16. if $n'$ contained in open and the existing node
17. is as good or better then discard $n'$
18. and continue;
19. if $n'$ is contained in closed and the existing
20. node is as good or better then discard
21. $n'$ and continue;
22. remove all occurrences of $n'$ from open and
The algorithm depicted in Listing 1 has as input the open list containing all nodes which can be visited. The open list is implemented as a balanced binary tree sorted based on f values, with tie-breaking in favor of higher g values. The tie-breaking mechanism results in the goal state being found on average earlier in the last f value pass. In addition to the standard open and closed lists, marker arrays are used for finding in constant time whether a state (node) is in the open list. Note that the closed list can be omitted (yielding a tree search algorithm) if a solution is found. Each path search is assigned a unique increasing ID that is then used to label array entries relevant for the ID of each search. The marker arrays are used in online-mode with the Pioneer 2DX robot.

The experiments were conducted in this manner in order to find out which is the best configuration for the set of parameters used inside the two path planning algorithms. We selected while final destination was selected around the center of the map.

A. The A* Algorithm vs. Fast A* Algorithm in Offline Mode

Figure 1 represents the map used for calculating the times for each of the run-times during offline simulation of the A* algorithm and fast A* algorithm. The rectangles filled or having orange border depicted in Figure 1 represent obstacles (not passable map areas). The path depicted in Figure 1 with interconnected blue tiles from the top left corner towards the middle of the map represents a valid robot navigation path. The valid path avoids obstacles depicted in Figure 1 with rectangles having an orange border and additionally several borders depicted with different levels of grey color. Note that darker the grey color tone is (in the map tiles) as forbidden the area is for the path planning algorithm. Thus, the algorithm tries to avoid these areas as much as possible.

Note that we conducted each run for a set of parameters by increasing the heuristic number (Heuristic #) see Table I and Table II from 0 to n as long as the run-time calculated in seconds was decreasing. The first time we noticed that the runtime was increasing we stopped the test run and we selected another formula and repeated the experiment by starting with the heuristic number 0. The experiments were conducted in this manner in order to find out which is the best configuration for the set of parameters used inside the two path planning algorithms. Note that in a real scenario (the environment can constantly change) path planning computations need to be performed with a higher rate (e.g., in our opinion less than 100 milliseconds).

| Run | Heuristic # | Formula | Time [sec] |
|-----|-------------|---------|------------|
| 1   | 0           | M(x,y)  | 1.2         |
| 2   | 1           | M(x,y)  | 1.3         |
| 3   | 2           | M(x,y)  | 1.4         |
| 4   | 3           | M(x,y)  | 1.5         |
| 5   | 4           | M(x,y)  | 1.6         |
| 6   | 5           | M(x,y)  | 1.7         |
| 7   | 6           | M(x,y)  | 1.8         |
| 8   | 7           | M(x,y)  | 1.9         |
| 9   | 8           | M(x,y)  | 2.0         |
| 10  | 9           | M(x,y)  | 2.1         |

Table I and Table II depict: (#) the number of the run, (Heuristic #) the heuristic number which can vary between 0 and n, (Diagonals) if diagonals on the path were allowed or not, (Formula) different formulas used for the distance metric (e.g., m-manhattan, $M(x,y) = \max(Dx,Dy)$, D.S.-diagonal shortcut, $E$-Euclidean and SQR-Euclidean without square) and (Time [sec]) in seconds for each run.
TABLE I: Test results of the A* algorithm

| #  | Heuristic | Diagonals | Formula | Time [sec] |
|----|-----------|-----------|---------|------------|
| 1  | 0         | ✓         | m       | ≥30        |
| 2  | 1         | ✓         | m       | 1.34       |
| 3  | 2         | ✓         | m       | 0.01       |
| 4  | 3         | ✓         | m       | 0.03       |
| 5  | 0         | ✓         | M(x,y)  | 10.68      |
| 6  | 1         | ✓         | M(x,y)  | 2.74       |
| 7  | 2         | ✓         | M(x,y)  | ≥30        |
| 8  | 0         | ✓         | D.S.    | ≥30        |
| 9  | 1         | ✓         | D.S.    | 1.31       |
| 10 | 2         | ✓         | D.S.    | 0.03       |
| 11 | 3         | ✓         | D.S.    | 0.34       |
| 12 | 0         | ✓         | E       | 25.30      |
| 13 | 1         | ✓         | E       | 2.20       |
| 14 | 2         | ✓         | E       | ≥30        |
| 15 | 0         | ✓         | SQR     | 25.96      |
| 16 | 1         | ✓         | SQR     | ≥30        |
| **Total** | | | | ≥219.94 |

TABLE II: Test results of the fast A* algorithm

| #  | Heuristic | Diagonals | Formula | Time [sec] |
|----|-----------|-----------|---------|------------|
| 1  | 0         | ✓         | m       | 0.09       |
| 2  | 1         | ✓         | m       | 0.03       |
| 3  | 2         | ✓         | m       | 0.01       |
| 4  | 3         | ✓         | m       | 0.003      |
| 5  | 4         | ✓         | m       | 0.003      |
| 6  | 0         | ✓         | M(x,y)  | 0.10       |
| 7  | 1         | ✓         | M(x,y)  | 0.04       |
| 8  | 2         | ✓         | M(x,y)  | 0.02       |
| 9  | 3         | ✓         | M(x,y)  | 0.01       |
| 10 | 4         | ✓         | M(x,y)  | 0.01       |
| 11 | 5         | ✓         | M(x,y)  | 0.01       |
| 12 | 6         | ✓         | M(x,y)  | 0.008      |
| 13 | 7         | ✓         | M(x,y)  | 0.007      |
| 14 | 8         | ✓         | M(x,y)  | 0.007      |
| 15 | 0         | ✓         | D.S.    | 0.10       |
| 16 | 1         | ✓         | D.S.    | 0.02       |
| 17 | 2         | ✓         | D.S.    | 0.01       |
| 18 | 3         | ✓         | D.S.    | 0.0034     |
| 19 | 4         | ✓         | D.S.    | 0.0037     |
| 20 | 0         | ✓         | E       | 0.11       |
| 21 | 1         | ✓         | E       | 0.04       |
| 22 | 2         | ✓         | E       | 0.02       |
| 23 | 3         | ✓         | E       | 0.02       |
| 24 | 4         | ✓         | E       | 0.01       |
| 25 | 5         | ✓         | E       | 0.01       |
| 26 | 0         | ✓         | SQR     | 0.12       |
| 27 | 1         | ✓         | SQR     | 0.01       |
| 28 | 2         | ✓         | SQR     | 0.0012     |
| 29 | 3         | ✓         | SQR     | 0.0011     |
| 30 | 4         | ✓         | SQR     | 0.0015     |
| **Total** | | | | 0.83       |

The goal of this experiment is to measure the run-times obtained for different runs and to find out how the robot manages to follow a given path by avoids previously unknown path obstacles. In this experiment we used the WiFi based application which planned and steered the Pioneer 2DX robot in a partially known environment by using two running modes.

B. Path Planning with the Fast A* Algorithm in On-line Mode

Figure 2 depicts a partially known environment. Note that the rectangles with diagonals lines inside (O1 and O2) depicted in Figure 2 represent unknown obstacles which were previously not modeled inside the path planning application map (Java back-end application) depicted in Figure 3. The fast A* algorithm was tested on this two maps (with only O1 and then with both O1 and O2) with the real Pioneer 2DX robot simulator [17] with the goal to find out if the robot can deal with partially known environments. The experiment was performed in a room having six by eight meters and by remodeling it in the steering application depicted in Figure 3. We used for the online experiments the 14 configuration from Table II (i.e., Heuristic # 8, Diagonals on ✓ and Formula M(x,y)). We decided to use this configuration because it was the longest run from our experiments where the Heuristic # number could be increased (8 times) until the search time
First, an unknown obstacle (i.e., depicted in Figure 2 with O2) was added to the test environment (room) and the path planner application (online mode) was run. Second, another unknown obstacle (i.e., depicted in Figure 2 with O2) was placed in the same test environment as before. As result the test environment contained two unknown obstacles (i.e., O1 and O2). Finally, for both of this scenarios the runtimes of the Pioneer 2DX robot were measured by navigating from the initial location (depicted in Figure 2 with letter R) to the final location (depicted in Figure 2 with letter D). The results of these experiments are depicted in Table III.

Note that the obstacles depicted in the Pioneer simulator map (Figure 2) are not present in the path planner application—Figure 3. Thus the robot had to deal with these obstacles in order to reach its target destination which was previously given (i.e., denoted with letter D in Figure 3).

Table III depicts the run-times for the two running modes on two real environments. In column four of Table III we observe that for the first environment (Figure 2 with one obstacle) the best run-time (47 seconds) is obtained with M1 selected and that for the second environment (Figure 2 with two obstacles) the best run-time (40 seconds) is obtained with M2 turned on. Note that in Figure 2 we had two unknown obstacles (i.e., O1 and O2) which were added one after each other for each of our experiments. When running the fast A∗ algorithm on the map depicted in Figure 2 (containing only O1) with M1 the run-time increases w.r.t. M2 because the range of the ultrasonic sensors was set tp 50 millimeters. Note that the sensors distance parameter for M1 was set to 50 millimeters whereas for M2 this value was set to 225 millimeters. Thus, the robot can make decisions earlier or later along the path. As result the obstacle depicted in Figure 2 (i.e., O1) is detected later as compared to the detection of both obstacles depicted in Figure 2 when the range of the sensors was increased to 225 millimeters. Thus, the result is the addition of several recovery actions needed in order to recuperate the robot and point him to the target destination.

However, when performing path planning with the map depicted in Figure 2 with two unknown obstacles using M2 the run-time decreases because the range of the ultrasonic sensors was set to 225 millimeters and the result is that the obstacles are detected earlier. This removes additional recovery actions needed by the robot in order to find a new obstacle avoiding path, thus time is not wasted. We infer (with caution) from these results that the second mode is best suited for environments with more unknown obstacles whereas the first mode is better suited for environments with less unknown obstacles.

VI. CONCLUSION AND FUTURE WORK

In this paper, we evaluated the A∗ algorithm and the fast A∗ algorithm w.r.t. completeness and we shown that the fast A∗ algorithm can be successfully used for indoor mobile robot navigation by using only data collected from ultrasonic sensors. We built two software tools (for offline and online algorithm testing) which helped to tweak the used algorithms and to take further decisions based on this results. The results obtained from comparing the A∗ algorithm and the fast A∗ algorithm (Section V-A) indicate a speed-up of two orders of magnitude w.r.t. the fast A∗ algorithm inside our offline simulator. The second mode used with the fast A∗ algorithm is best suited for environments with less unknown obstacles whereas the first mode is better suited for environments with more than one unknown obstacles (in our experiments). We are aware that further experiments are need in order to fully claim the above stated. Additionally, we showed in our experiments that the fast A∗ algorithm is complete (finds a path in due

![Fig. 3: Environment map available in the path planner. Initial robot position (R); Final robot destination (D).](image-url)

Fig. 3: Environment map available in the path planner. Initial robot position (R); Final robot destination (D).
time). We leave the computation of its performance as a future exercise.

In future we want to further tweak the fast $A^*$ algorithm and use other algorithms with more complex unknown environments. We want to use more advanced path planning algorithms and we want to combine input from multiple sensors (i.e., perform fusion of data from several sources) which will give a more precise description of the environment. Furthermore, we want to compute the performance of the used algorithms and compared them with each other.

ACKNOWLEDGEMENTS

We want to express our gratitude to the anonymous reviewers for their constructive criticism.

REFERENCES

[1] Amit’s Game Programming Information, http://www.csstudents.stanford.edu/amitp/gameprog.html, accessed on the 28 Jan. 2010.
[2] Aria, ActivMedia Robotics, Inc. Aria Reference Manual 1.1.10, 2002.
[3] $A^*$ algorithm, Available [Online], http://en.wikipedia.org/wiki/A*_search_algorithm.
[4] Counter-strike video game, Available [Online], https://en.wikipedia.org/wiki/Counter-Strike, 1999.
[5] P. W. Eklund, S. Kirkby, and S. Pollitt, A Dynamic Multi-source Dijkstra Algorithm for Vehicle Routing, In Proceedings of the Conference on Intelligent Information Systems, 1996.
[6] Iterative deepening $A^*$ Available [Online], https://en.wikipedia.org/wiki/Iterative_deepening_A*
[7] L. Jaulin, Path planning using intervals and graphs. Reliable Computing 7 (1), 2001.
[8] J. H. Jones, $A^*$ Tutorial, http://cs.nyu.edu/courses/fall10/G22.2965-001/astartravsales) accessed on the 28 Jan. 2010.
[9] R. Korf, Depth-first Iterative-Deepening: An Optimal Admissible Tree Search, In Artificial Intelligence 27: 97109, 1985.
[10] K. Konolige, K. Myers, E. Ruspini, and A. Saffiotti, The SAPHIRA architecture: a design for autonomy, In Artificial Intelligence Based Mobile Robots: Case Studies of Successful Robot Systems, MIT Press, 1998.
[11] Lantronix, WiBox 2100 Quick Start Guide, Available [Online], https://www.lantronix.com/wp-content/uploads/pdf/WiBox_QS.pdf, Lantronix, 2004.
[12] K. Makanac, and M. Takaki, $A^*$ Tutorial, Development of the 3-Dimensional Urban Spatial Data Model and Application to the Pedestrian Navigation System, ITS, 2004.
[13] Minkowski addition, Available [Online], https://en.wikipedia.org/wiki/Minkowski_addition.
[14] P. C. Nelson, and A. A. Toptsis, Unidirectional and Bidirectional Search Algorithms, 1992.
[15] Pioneer2DX Robot, Available [Online], http://wiki.ros.org/Robots/AMR_Pioneer_Compatible.
[16] J. Rosel, and P. Iñiguez, Path planning using Harmonic Functions and Probabilistic Cell Decomposition, In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2005.
[17] Saphira, Saphira Operations and Programming Manual Version 6.2, Mobile Robots ROBOTICS, 1999.
[18] R. Siegwart, and I. Nourbakhsh, Introduction to Autonomous Mobile Robots, MIT Press, ISBN: 9780262015356, 2004.
[19] Y. Zhang, L. Wu, and S. Wang, UCAV path planning based on FSCABC, In Information-An International Interdisciplinary Journal, pp. 687692, 2011.