GRIPPERBOT

BACKGROUND RESEARCH REPORT

PREPARED FOR 5CCS2SEG 12~13

SOFTWARE ENGINEERING GROUP PROJECT

TEAM ENGELBART

[Draganov Konstantin, Fang Zhou, Fernee Mark, Kyei-Baffour Kevin Kofi, Mitchell Finlay Grant, Mosquera Prieto Phillip,
Oh Sewoong, Rakhimgaliyev Sanzhar]

INTRODUCTION

The aim of this report is to address key challenges in implementing a complete solution which meets the requirements of our group project assignment. Below is a reflection of the research carried out and the conclusions drawn by the team with the single main purpose of finding the best solution that can be implemented while taking into account the time and resource constraints that are present. In order to tackle the problem it is necessary to divide the entire project into smaller manageable sub-tasks. The overall goal is to produce control software for autonomous mobile robots. The robots must be able to explore an indoor working environment, map it using sensor data and also navigate it. In addition to these tasks the robots must also be able to compute paths to predetermined locations and follow them without colliding. From the research carried out, the team has arrived at the conclusion that it needs to address a problem referred to as SLAM [1] (Simultaneous Localisation and Mapping). The team has researched and gathered relevant information for the sub-problems of SLAM and the many ways that these problems have been solved. In order to produce the best solution possible, a comparison and evaluation of the advantages and disadvantages of the findings and available solutions gathered has been carried out.

PROBLEM DOMAIN – REQUIREMENTS SPECIFICATION

In order to successfully meet the specified requirements of this year's group project it was necessary to define precisely what the main problems that needed to be solved were. The group project requires the implementation of an autonomous mobile robot control software system. This includes a command line interface as well as a graphical user interface (GUI). The robot should be able to navigate and localise itself while mapping an unknown environment. In addition to this the robot should be able to detect objects defined as 'garbage/rubbish' and move them to a designated collection area.

As previously mentioned the team needs to address SLAM's sub-problems which can be solved in a variety of ways. From the task requirements it can be seen that the robot is required to explore the whole accessible floor area and as the robot needs a way of storing information for places it has visited and the observation it has made for these places. This means that an in-depth knowledge of the mapping sub-problem of SLAM is required. This would be a fundamental part of the solution as all other features would be dependent on that map representation. In order to map the available floor area the robot needs to navigate safely without colliding through the space while also localises itself and store appropriate information for the given location.

Research into different data structures which can be used for storing the map representation has to be carried out. After the exploration of the map is completed the software needs to come up with an efficient way to collect all of the found trash objects and bring them to the specified area. In order for our product to achieve this, research and evaluation of different path planning algorithms has to be done. Research into different path planning algorithms is necessary if the robot is to be able to efficiently collect the trash objects around the map. In order to successfully pick up an object the robot should be precisely guided to a specific position appropriate for taking the rubbish with its gripper.

To meet the full requirements it is necessary to ensure that software can handle multiple robots. This includes ensuring tasks are divided up between the robots and also ensuring that our robots do not collide with each other. These tasks must be executed in the most efficient way (e.g. robots to collect the closest garbage objects to their current location).

SIMULTANEOUS LOCALISATION AND MAPPING

The simultaneous localisation and mapping problem (SLAM) is applicable whenever a mobile robot needs to build a model of the real world while at the same time navigating through the same environment using its own model [2]. SLAM is applicable to a variety of situations, however for the requirements of the given task SLAM for two-dimensional (2D) indoor environments is considered. There are many various readily available solutions, but each of them is concerned with a particular hardware or a constrained environment. There is no universal solution applicable to all situations. SLAM has been successfully solved with a wide range of hardware e.g. sonar sensors, laser sensors, contact (proximity) sensors and even in the last decade using vision cameras. For the assignment of this year's group project the team is concerned with the implementation of SLAM using range sensors (with constrained range and field of view) and one front facing fiducial sensor.

LOCALISATION

Localisation is a core part of SLAM problem – computing the current position regarding the initial unknown position in an unknown environment [2]. In the project the robot will operate in a simulated Player/Stage environment which will provide us with accurate location data. This means there is no need to carry out research and address the localisation sub-problem of SLAM.

MAP REPRESENTATION

Map representation is an essential part of the project as the robot needs a way of knowing which places it has visited in the real world and what observations it has made (e.g. where rubbish objects and obstacles are located). This would also enable us to check when the robot has explored the complete floor area. The map will also be used by the path planning algorithms which will be covered below, so the robot needs an accurate and precise representation of the environment as it will directly affect the efficiency and the accuracy of the path planner. Having carried thorough research on different ways of map representations the team has concentrated its attention to two main ways of modelling the environment, in which the robot will operate in. Sample environments similar to the one that our robot will operate in are showed in figures 1, 2 and 3 below.

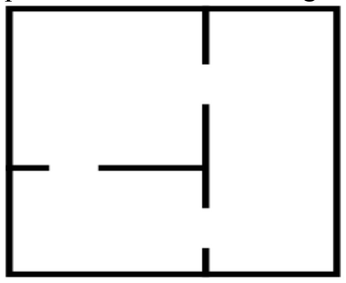


Fig.1

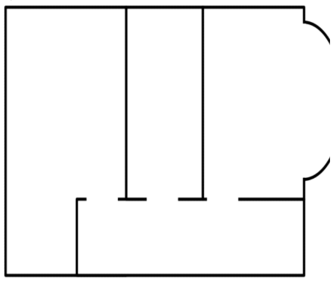


Fig.2

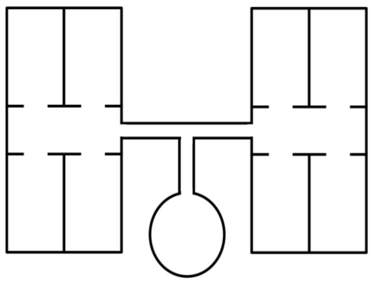


Fig.3

GRID BASED REPRESENTATION

Metric maps represent the real world environment in detail reflecting exactly or approximately the geometry of the represented world. Commonly they are represented using occupancy grid of cells [6]. Most widely used for indoor two-dimensional environments is the use of square cells, other shapes can be used as well. Each square or cell of the grid represents a small part of the world. It stores information whether the cell is unoccupied (it contains no obstacle or rubbish) or it is occupied. Even if it is just partially occupied the cell is marked as such. Using the information from its sensors the robot will update and add information to the map incrementally. In this year's project the robot is equipped with range sensors and fiducial sensor which has shorter range. When a sensor doesn't return its maximum range value this means that an obstacle is sensed. The cell coordinates (position) are computed using basic trigonometry from the robot's current position and its heading both of which are provided by the simulation environment. Then the corresponding cell is marked as occupied. If the maximum range value is returned from the sensor this means that there are no obstacles ahead of the robot and all cells in front of the robot are marked as unoccupied. All unexplored area will contain cells with no information. A robot must be able to move to the centre of a cell for it to be marked as unoccupied [4].

If the robot was operating in a real world environment and was lacking the benefits provided by the simulation environment, a further research would have been necessary to address the problem of noisy sensor readings. In addition to that the robot would have needed a way of computing its current location. If this was the case the team would have to do more research of probabilistic methods which will express the uncertainty in our knowledge of whether the occupied cell observed is really occupied and vice versa. One such method is using Bayes rule [7] which is not covered in further detail here.

When using metric maps for modelling the environment the team should consider what should be the size of the cell. This decision has a direct impact on the performance of the software. Using a smaller cell size means that finer details could be represented (e.g. small passages which otherwise would be marked as occupied), but will also require more memory and computational power.

For metric map representation two main types of data structure can be used and are being considered by the team. Two-dimensional arrays is the first which would be efficient for running path planning algorithm on it, but would be costly in cases where an extension of the current map is needed as the array has become too small. Also there would be loss of space as open space would be represented by multiple grouped free cells. Another data structure that can be used would be a linked list which would deal with the problem of extending the stored map easily, but would take more time for the path planner to compute a path. Using linked list we would store the four neighbouring cells to the robots current position as illustrated in figure 4.

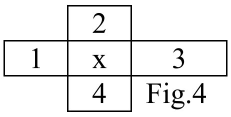


Fig.4

TOPOLOGICAL REPRESENTATION

This form of mapping can be loosely compared to the way in which a human navigates in real life. We humans tend to define routes as a set of landmarks between a set of directions, in order to reach a destination or to work out our current position. As an example: *Go straight for 60 meters, turn right at the church and continue straight for 200 meters until you see the pub.*

From the above sentence the following facts have been established about the environment: There are two distinct landmarks which are the church and the pub, three directions, and specific distance values about the route.

Should a human be given this information about the environment, we would expect to a point of certainty, the person would arrive at the intended goal (the pub). On the other hand if we placed the person in a vast empty space without any distinctive landmarks for example a desert and with the goal of reaching civilization, we would expect to be extremely unlikely that the person would achieve this as they would have no notion to set their direction.

The above logic can be translated to topological mapping, where a robots environment is represented as a graph with landmarks such as doorways, corners etc. depicted as vertices and paths connecting those vertices denoted as edges. The analogy given above also describes distance values which are typically used by a topological/metric map hybrid which stores odometer data representing arcs between vertices. Without any metric data there can be a lack of definition between certain features of the environment i.e. two objects may look the same, causing a conflict in determining the position of the robot. In reference to the human example if a person were to walk only 30 meters and find a church without the information declaring the right turn is at a church, 60 meters from the persons initial position, they will assume this is the correct church and if possible will turn right and will not reach the goal.

Modelling the world using a topological approach utilises sensor data where the observed 'well defined' (landmark - an example can be concave and convex corners [8]) features of the environment will be placed within a data structure where the co-ordinates of each vertex will be stored and a connecting edge will hold the coordinates of each vertex connected. With these coordinates and the use of path finding algorithms such as A*, navigating between vertices is simpler without significant computations. Though, as topological maps lack details, computed paths are unlikely to be optimal. Another advantage when building a topological map of an environment is that it is not affected by the position of the robot or for it to know its own position.

An appropriate data structure that can be used for topological map representation would be list of nodes and edges which will represent as objects. Each node object would store references to the edges it is connected to and all edge objects would store the two connected vertices and the length between them.

CONCLUSION

Mapping of the robot's working environment is one of the most significant challenges the requirements of this year's project the team is presented with. In order to be able to know that the whole floor area has been explored (there is no unexplored place which the robot is able to go to) the robot needs to be able to create an internal model of the environment. Also there is no way of being able to plan how and in what order to collect the garbage objects without having stored information for the observed work area.

Metric (grids-based) map representation approaches are easier to build, represent and maintain. There is no need to worry about differentiating between two places while if a topological approach is to be used there is a need of a way of disambiguating between two separate places which look alike, but in fact they are different [1]. Grid based approaches need accurate localisation of the robot, but this is covered above as something which is solved by the Player/Stage simulation environment the robot operates in. The main advantage of exact representation of the real world is also its biggest disadvantage as it is requires more memory. Also metric approaches are costly in terms of space as there will probably be large portions of the floor which will have no obstacles, but will need to be divided in separate cells for which information has to be stored.

The other well-known approach is topological maps which are more difficult to construct and they can produce suboptimal paths. These suboptimal paths are cause by the lack of detail of topological map representation which also means that they require less memory. Topological map representation can be seen as an abstraction of the grid-base maps, they only store information for particular landmarks which also gives them the advantage of being compact. Key disadvantage of the grid-based approach is it needs the exact robot position in real world, but in the perspective of the group project this is eliminated as already explained above.

Both approaches have pros and cons for the particular assignment considered here. The advantages of the metric map representation are far more than the disadvantages, because of the simplifications which Player/Stage environment supply us with. The best solution would be to use an integrated approach as described in [1] where chunks of the grid map would be split and represented as nodes and will be separated by narrow spaces such as doorways or narrow hallways represented as edges. The team may try to improve the initial metric approach by combining it with a topological one as discussed in [1] depending on the time constraints.

PATH PLANNING

Path planning is an essential requirement for a robot to be able to navigate an environment in order for it to find the shortest path or the optimal path to a rubbish object. When discussing the shortest path we have to consider whether it's actually the easiest and cost effective, as an example: *Is the shortest path to go over the mountain or could there be a slightly longer, but more energy and time efficient path, which just goes around the mountain?*

This is essentially the same question path planning algorithms aim to provide the best possible answer to, and as efficiently as possible. When searching for the optimal path, it requires the algorithm to execute an exhaustive computation of the environment which may have significant time costs while other algorithms aim to find the "as close to optimal solution" without traversing every node on a map.

When assessing different algorithms, we are concerned with the time it takes the robot to navigate to the goal, the efficiency of the algorithm in computing the environment and it's time complexity.

DIJKSTRA'S ALGORITHM

Dijkstra's Algorithm aims to find the shortest path from the robots initial state to an intended goal state. The algorithm uses the robots initial vertex and considers all direct neighbours thereof. Each neighbour is assigned a cost which represents the distance from that point to the initial state in which the algorithm began. Once all neighbours have been considered, the algorithm will move to the next vertex with the lowest cost. The algorithm will expand outwards from the start point and repeat the cost assigning process until the goal state is found. Once reached, the algorithm will terminate, and a path, with the lowest accumulated edge cost, will be generated by moving back from the goal state. The robot will then use this data to traverse the map.

Figure 5 from [14] shows Dijkstra's midway through operation. As it stands, every possible node that could be visited (as defined by the rules of the algorithm at this point of time) has been traversed. The implication of this is that the algorithm is performing many computations that are not going in the right direction, resulting in wasted processing. Figure 6 from [14] represents the final result that the algorithm will produce. As the goal state is in the top right of the map we notice that close to every vertex has been assigned a cost. While the result produced is optimal, the cost in time and resources, that it has taken to achieve this result, is significantly higher than the result that an A* algorithm would produce.

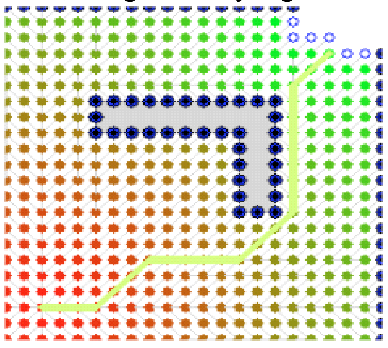


Fig.5

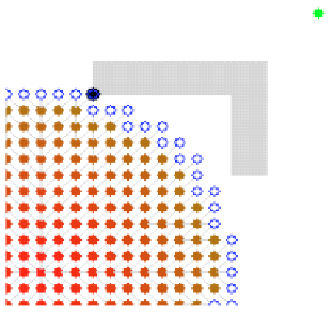


Fig.6

A* SEARCH ALGORITHM

A* search algorithm is considered the best search algorithm used in many areas such as Game Development and Satellite Navigation System. It is essentially the same as Dijkstra's algorithm, however A* is more efficient as it employs heuristic [10]. Given the starting and target node, it is able to produce a series of estimates that calculates the remaining distance from a node to the target node. These estimates thus provide a guide for the direction of the path with each visited node updating the true cost of the path, and this process continues until a particular path is found.

The algorithm considers its next action at each node based on a heuristic score. Let G be the Euclidean distance (shortest distance) between start node and current node, and H be the estimated distance between current node and target node. Then the heuristic score is G + H. A common measure used in calculating distance for path-finding problems is the Manhattan Method which calculates the sum of grids moved horizontally and vertically in order to reach target position from current position in a map, neglecting any diagonal movement and obstacles in the way.

A* maintains two lists, a "Closed" list which stores already scanned nodes, and an "Open" which stores nodes that are yet to be scanned. Initially, the open list adds the start node to it. The algorithm then repeatedly checks the node with the smallest F cost on the open list and switching them to the closed list, adding certain nodes to open list while making the current node the parent of them and comparing nodes that are already on the open list using heuristics. The process repeats until the target node is included to the closed list or there is no path to be found, in which case the process terminates [9].

Because Dijkstra's algorithm does not employ any heuristics and at each step chooses the smallest F cost node, it tends to explore a wider area of the map than Dijkstra's. A* would be less efficient than Dijkstra's when for example, there are several target nodes on the map, but the robot doesn't know which one is closest. Compared to Dijkstra's or the Wavefront algorithms, if a map has few obstacles, A* doesn't search the rest of the map unless it needs to as illustrated in figure 7 and 8 from [15]. A* is the most popular choice for path finding as it utilises the best pieces of information from Dijkstra's and Best First Search, a search algorithm that explores a graph by choosing the most promising node chosen according to a predefined rule[10].

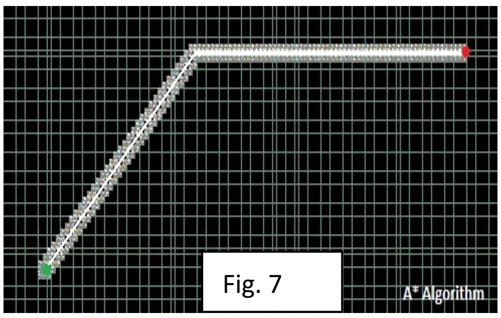


Fig. 7

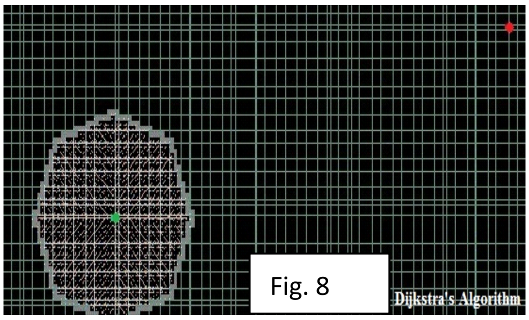


Fig. 8

WAVEFRONT ALGORITHM

The Wavefront algorithm as explained in [13] employs the Breadth First Search principle, a graph search algorithm that starts at the start node and explores all the nodes neighbouring it in order of their distance from the start node [12]. The algorithm splits into layers with one at each depth, nodes are explored in breadth-first order without considering the target node. In A*, the estimated cost of an optimal path is given by the heuristic score F = G + H, where H is an admissible heuristic (estimated distance from current node to target node). Using the G cost as the measure, A* checks the nodes on the open list and decides if there is a better path through that node. Nodes with higher G cost are neglected since such nodes cannot be on an optimal path. Wavefront is rarely used in practice, because the number of nodes it explores is usually bigger than the number of nodes traversed by A*. If all explored nodes are stored in memory, Wavefront uses the same amount of memory or more than A* therefore it has no advantage.

The Wavefront algorithm proceeds as follows: The robot first scans the map and creates a 2D X-Y grid matrix. It represents impassable positions (wall, obstacles, etc.) as 0 as illustrated in Figure 9 from [13]. Then it sets the position of the target and robot's location. The robot's location is represented as R, the target position is represented as G, as illustrated in Figure 10 from [13]. The robot plans the path by searching the grid one column at a time, from top to bottom, left to right. It checks the surrounding cells of each cell visited and marks the current accordingly. If a cell is adjacent to the goal then it is marked with a 3 (as in the example in figure 10). If a cell is adjacent to a numbered cell it is marked with that value incremented by 1. Once the search reaches the cell marked R (for the robots current position) it finds that there is a numbered cell adjacent. This indicates a path exists to the goal and the algorithm is complete. The robot then traverses to its target position by following the shortest route that has been created by the algorithm. In figure 10 from [13] there are two paths of equal length (both indicated by 8). The robot chooses the path with fewest turns to traverse.

The Wavefront algorithm explores a large area before it finds a path. Computationally, it is less efficient than A*. The advantage of Wavefront algorithm is that simple data structures can be used to create the discretized map, for example, a 2D array.

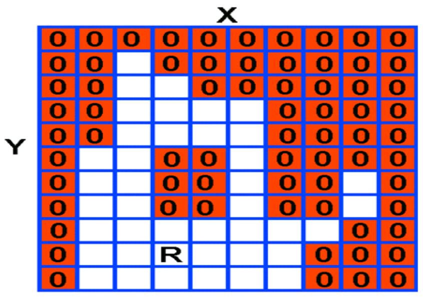


Fig. 9

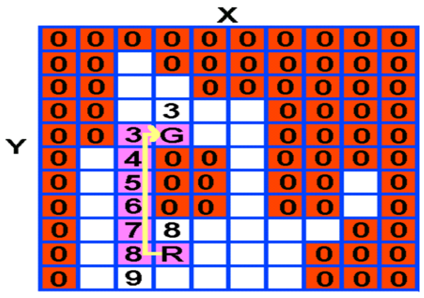


Fig. 10

CONCLUSION

To summarise, Breadth First Search strategies, such as the Wavefront are less informed and efficient compared to Best First Search strategies such as the A* and Dijkstra's. The Wavefront algorithm does not employ any heuristics. It exhaustively searches the entire graph without involving the target node until it finds it. However, it is complete which means if there is a solution, the algorithm will find it regardless of the graph. In order to make a search algorithms more informed and efficient, a heuristic function is needed to represent the knowledge of a specific problem. Best First Search strategies employ heuristics by exploring the graph by choosing the best node possible according to a predefined rule.

At a lower level, it can be concluded that Dijkstra's algorithm explores a large section of the map before finding a path which is the shortest. However, it is not considered efficient, as it traverses too many nodes. A* on the other hand is faster to compute, but will not necessarily produce the shortest path on a given map. It uses Best-First-Search which has a better performance in the average case and is very efficient when compared to Wavefront.

CONTROL ARCHITECTURES

For the robot to successfully complete its task it would need software which would be able to coordinate the movements and actions it will take in order to achieve its goal. This would require some sort of brain like control software and, because our task is not trivial (our robot will need to do several things sometimes this would need to be done in parallel) there is a need for a control architecture which needs to be able to come up with a plan for achieving the long term goal while also dealing with dynamic unpredicted situations (short term/immediate actions) like another robot moving in front of us. This control architecture would help us in modularisation of the code implementation and also with implementing better solution.

REACTIVE CONTROL

Reactive control architecture can be compared to the withdrawal reflex of the humans. For example if you touch something which causes you pain or some other strong unbearable sense you instantaneously try to get from it without even thinking for a moment. This is very much the case with this control architecture. The robot would act differently to various senses as in figure 11 from [11]. One of the difficulties of this architecture is to carefully separate different actions and senses.

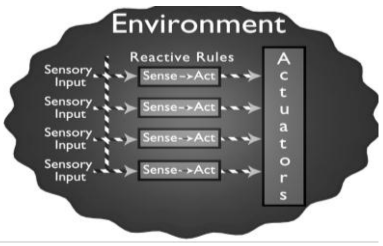
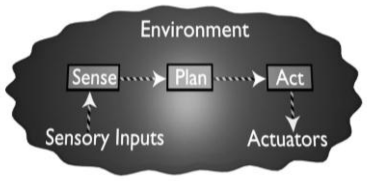


Fig. 11

DELIBARITIVE CONTROL

Deliberative control architecture is based on the principle "think hard-act later" [11]. This means that this architecture would spend a great deal of time of planning its actions and decisions before actually doing something. It assumes that we have an accurate representation of the world we are operating in as all the hard work of decision making is based on it and once we have come up with a plan it cannot be interrupted. Figure 12 from [11] shows a typical deliberative control architecture diagram showing the steps of SPA (sense –plan –act). This control architecture is not very common in robotics nowadays, but still widely used in artificial intelligence (AI).

Fig. 12



HYBRID CONTROL

Hybrid control architecture combines the two above explained control architectures and adds a middle layer which would act as a synchronizer for both of them. From the carried research and considering the requirements of the assignment, the team has chosen to use this control architecture framework, because it is most applicable to the given problem. It tackles the problem of mapping the complete world while safely navigating within it. In addition to this it also has to plan a path to the garbage object location and back to the trash collection area and while moving on this path to deal safely with unexpected obstacles once the entire world has been explored. This means that the path planning would be part of the deliberative subsystem. While the reactive subsystem would be responsible for translating the information from our sensors to the map and also deal with the encountered dynamic and unknown stationary objects and obstacles.

REPORT SUMMARY

In this report the team has discussed different ways of map representation with appropriate data structures and how sensor data can be used to update the map representation. The team has concluded that grid based approach would be fit for purpose in terms of maintenance and ease of implementation in the Player/Stage environment. We have also discussed the topological based approach and found that it would help the path planning algorithm to find an optimal path. During the course of the project, the team will try to combine the two approaches to produce the best solution. For path planning, we have concentrated on three different approaches and decided that the A* algorithm offers the best solution based on our problem domain. In order to successfully combine all of the separate components of the software to act as one we will base our implementation on hybrid control architecture.

REFERENCES

[1] Riisgaard, S. And Rufus Blas, M. (2005) SLAM For Dummies. A Tutorial Approach to Simultaneous Localization And Mapping.

[2] Hugh Durrant-Whyte, Tim Bailey, (2006) Simultaneous Localisation and Mapping (SLAM): Part I The Essential Algorithms, IEEE ROBOTICS AND AUTOMATION MAGAZINE

[3] Elfes, A., "Sonar-based real-world mapping and navigation," *Robotics and Automation, IEEE Journal of* , vol.3, no.3, pp.249-265, June 1987

[4] Learning Maps for Indoor Mobile Robot Navigation, *Sebastian Thrun*

[5] Map-based navigation in mobile robots. II. A review of map-learning and path-planning strategies. *Jean-Arcady MeyerDavid Filliat*

[6] Sebastian Thrun. 2003. Robotic mapping: a survey. In *Exploring artificial intelligence in the new millennium*, Gerhard Lakemeyer and Bernhard Nebel (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA 1-35.

[7] A. Elfes. Using Occupancy Grids for Mobile Robot Perception and Navigation, *In Volume 22, 6th issue of Computer on pages 46-57 sponsored by IEEE Computer Society,* 1989

[8] Wong, S.C.; MacDonald, B.A., "A topological coverage algorithm for mobile robots," *Intelligent Robots and Systems, 2003. (IROS 2003).Proceedings. 2003 IEEE/RSJ International Conference on* , vol.2, no., pp. 1685- 1690 vol.2, 27-31 Oct. 2003

[9] Lester, P. (18/07/2005). A* Pathfinding for Beginners. Retrieved on 19/01/2013, from http://www.policyalmanac.org: http://www.policyalmanac.org

[10] Patel, A. (n.d.). Retrieved 17/01/2013, from Amit's Game Programming Site: http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#the-a-star-algorithm

[11] Maja J. Mataric ,The Robotics Primer, MIT Press, August 2007

[12] "Breadth-Frist Search" retrieved on 27/01/2013 from
http://www.cs.sunysb.edu/~skiena/combinatorica/animations/search.html

[13] Society of Robots - Wavefront Algorithm. Retrieved on 19/01/2013, from
http://www.societyofrobots.com/programming_wavefront.shtml

[14]Animation of Dijkstra's algorithm for robotic path planning & Illustration of A* planning, Subhrajit Bhattachary http://correll.cs.colorado.edu/?p=965 retrieved on 17/01/ 2013

[15] C# - Visualizing Path Finding With Dijkstra, AStar, Bi-directional Dijkstra's, and Bi-directional A* Algorithms, http://www.codeproject.com/Articles/288685/Csharp-Path-Finding-With-Dijkstra-and-AStar-Algorithm retrieved on 13 /01/ 2013