RESEARCH ARTICLE OPEN ACCESS

Survey on Navigation Principles of Autonomous Mobile Robot

Mohammed Hammoud [1], Kinda Aboukassem [2]

[1], [2] M.Sc., Department of Computer and Control Engineering, University of Tishreen - Syria

ABSTRACT

In recent times, the importance of mobile robot has increased rapidly. Accordingly, Navigation for mobile robot are in needed allow the robot to localize and move in free collision path with the least cost depend on application (time, energy). The objective of this paper is to present a state of the art survey of some techniques of localization and path planning for mobile robot. Localization techniques of mobile robot is made with explanation advantages, disadvantages and implementation for each of it. After that different path planning algorithms was explained. These study was developed in order to use some of these algorithms, in the near future, as a second stage after reducing the state space and demonstrate the extent to which of these algorithms are applied in stm32 microcontrollers embedded within the robots.

Keywords: - MR, Autonomous mobile, Automated guided vehicles(AGV),Automated guided vehicles (AGV),Unmanned Aerial Vehicle(UAV), Autonomous Vehicle(AUV),Remote Operate Vehicle (ROV), Path planning, localization.

I. INTRODUCTION

Autonomous mobile robot -These are more autonomous robots, developed by optimizing sensors and providing intelligent robot control. The sensors are capable of perceiving the details of more complex situations, but to cope with these situations, the robot's behavior control must be extremely flexible and adaptive.

II. NAVIGATION IN MOBILE ROBOTS

Mobile robots often operate in an unknown and unstructured environment, and the robot needs to locate itself, plan a path to a goal, build and interpret a map of the environment, and then control its movement in that environment. Thus, the concept of navigation includes several tasks **[2]** as shown in the Fig. 1.

Perception: the robot must interpret its sensors to extract meaningful data. Localization means that robot must determine its position in the environment. Cognition means that robot must decide how to act to achieve its goals. Motion control means that robot must modulate its motor outputs to achieve the desired trajectory**[2]**.

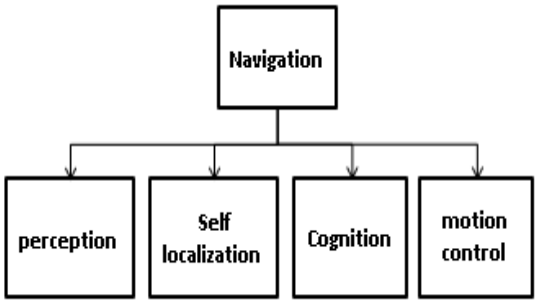

Fig. 1 Disciplines of system systems

A. SELF-LOCALIZATION

Self-localization answers the robot's question, where am I? Relative to the map. The aim of the operation is to determine the location of the robot and its direction, for example, a ground-cleaning robot needs to know that it has covered the entire floor without repeating the cleaning process for the same place or losing its location (lost).

The main difference between the operator and the mobile robot is the estimation of the location, in other words, the operator has a fixed base and by measuring the positions of the robot's joints and knowing its kinematic model can determine the position of the operator while the mobile robot moves as an integrated unit within the environment and there is no direct way to measure its position and direction. Hence, the general solution is to estimate the position and direction of the robot through velocity integration[8].

The map of the environment within which the robot will move may be predefined, and here the path of the robot is planned in advance. We have a relatively stable structure and robust operation is achieved (industrial applications). In the event that the planning of the path is dynamic so that the features of the surrounding environment are sensed, the robot first determines its location and then plans the movement through the areas available for movement, this type is suitable when the work space and tasks are frequently changing[8].

The techniques used**[3]**[6] in localization are divided ,as shown as in Fig. 2.

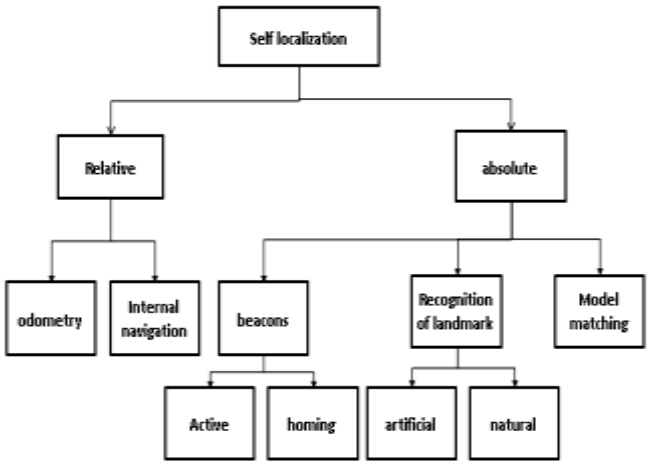

Fig. 2 Self localization techniques

❖ **Relative localization.**

This is performed by odometry or internal navigation robot. Odometer uses encoders to measure wheel/rotation and/or steering angle. Inertial navigation uses gyroscopes / accelerometers to measure the rate of rotation and the angular acceleration, as show in Fig. 3.

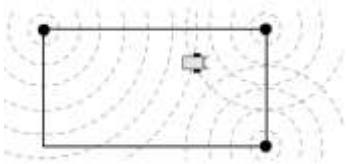

Fig. 3 Relative localization

❖ **Active beacons**

Robot compute absolute position by measuring the direction of incidence of three or more transmitted beacons. The transmitters use light or radio frequencies and are placed at known positions in the environment.

As we have already said, several simultaneous beacons send sonar signals at different frequencies at the same time intervals. If we receive the signal from 2 3, the robot can determine its location by calculating the difference in the arrival time of the signals[6].

Distances are computed by measuring travel time of radio waves from the beacon to the sensor[8].

Using two beacons can narrow down the robot position to two possibilities, the arrival of the signals at the same time means the robot is in the middle, but if the signal arrived from the left before, this means that it is closer to the left side with distance proportional to the time difference.

This method determines the location and not the direction, the direction is determined by a change in position (difference between two successive sites), which is used in GPS.

The use of GPS is possible just in outdoor, but in many cases it is not possible to use it due to limitations in the robot's environment or it is not desirable to prevent the autonomy of the robot, instead in Indoor we use global sensor with sonar, laser, radio beacons[6]. GPS is unacceptable for localizing mobile robots(desk MR, human scale MR, body-navigating Nano robots because first GPS provide accuracy to within several meters, second GPS can't function indoor in obstructed areas and are thus limited in their workspace**[2]**. localization implies more than knowing one's absolute position in the Earth's reference frame.

❖ **Homing beacons.**

Using light emitting homing beacons instead of sonar beacons. With **two** light beacons with different colors, the robot can determine its position and orientation at the intersection of the lines from the beacons at the measured angle. [6] .

In order to **know orientation**, the robot has either to perform a 360° rotation, or to possess an omnidirectional vision system that allows it to determine the angle of a recognized light beacon. Since we do not know the robot's distance from either of the beacons, all we know is the angle difference under which the robot sees the beacons (here: 165°– 45° = 120 as shown as Fig. 4.a.

Knowing only two beacon angles is not sufficient for localization as shown as in Fig. 4-b**.** If the robot in addition knows its global orientation, for example by using an on-board compass, localization is possible as shown as in Fig. 4-c. When using **three** light.

Beacons, localization is also possible without additional orientation knowledge as shown as in Fig. 4-d.

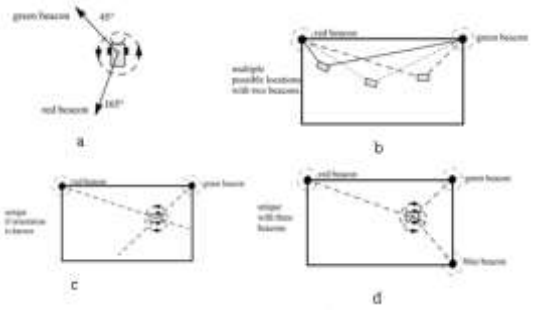

Fig. 4 Homing beacons(a: angle difference , b:use two beacons for localization ,c: using an on-board compass ,d:use 3 beacons for localization)

❖ **Recognition of artificial landmark**

Artificial signs are placed in specific places in the environment and these marks must be highly detectable in appearance even in poor environmental conditions.

❖ **Recognition of natural landmark**

Distinctive features of the environment are identified, and these signs are predetermined. This method is **less reliable** than using industrial landmarks.

❖ **model matching**

The absolute location is estimated by comparing the Sensor information with the robot's environment map.

Floor-based localization techniques are often replaced by laser-based methods. Automated guided vehicles (AGV) in industrial environments use various navigation/guidance technologies: magnetic tape, wire, magnetic spot, laser, and natural.

Laser triangulation methods, in which a spinning laser senses range and azimuth to wall-mounted reflectors, provide accurate localization information and don't need to follow

specific lines on the floor. Laser guidance technology uses multiple, fixed reference points (reflective strips) located within the operating area that can be detected by a laser head mounted on the vehicle as the facility is mapped in advance, paths can be easily changed and expanded as shown as in Fig. 5.

1. Map building and map interpretation

The robot's environment representation can range from a continuous geometric description to a decomposition-based geometric map or even a topological map**[2]**.

Before path planning, the environment needs to be presented in a unified mathematical manner that is suitable for processing in path searching algorithms.**[1]**

B. Traffic control.

There are different ways for map representation for environment[9]**[2]** as shown in Fig. 6.

A map representation should satisfy several basics:**[2]**

- Map accuracy must match the precision the robot needs to achieve its goal.
- The accuracy of the map and the type of features represented must correspond to the accuracy and type of data received from the sensors.
- Map complexity directly affects computational complexity of reasoning about mapping, localization, and navigation.

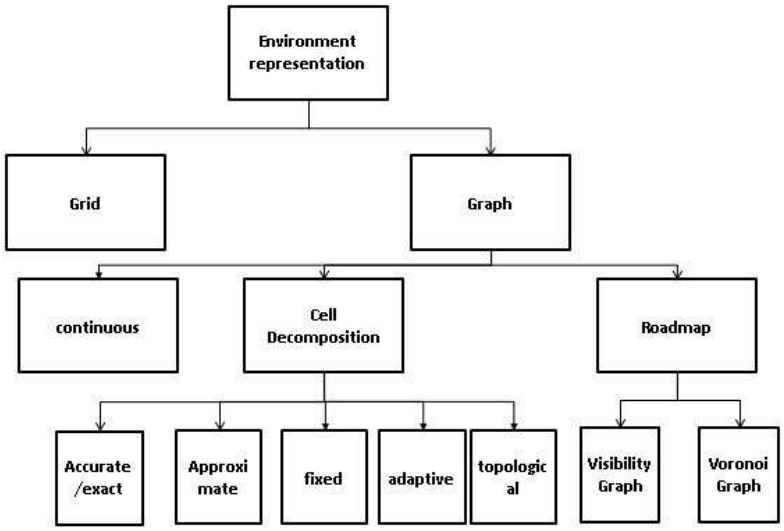

Fig. 5 Methods to build and interpret maps

A. Continuous representation

In a continuous representation of the map the map is represented as a set of lines that converge lines in the real world as shown in Fig. 7**[2]**

It is characterized by the fact that it accurately gives the characteristics of the environment in a connected space in terms of the composition of the environment and the location of the robot within it, this is done by using a filter that removes all the non-linear data.

A disadvantage of this method is that the generated map is computationally expensive. The total storage needed in the map is proportional to the density of objects in the environment, so combination exactness of a continuous representation with the compactness of the closed-world assumption can represent sparse environment by a low-memory map. This means that one assumes that the representation will specify all environmental objects in the map, and that any area in the map that is devoid of objects has no objects in the corresponding portion of the environment. Thus, the total storage needed in the map is proportional to the density of objects in the environment.

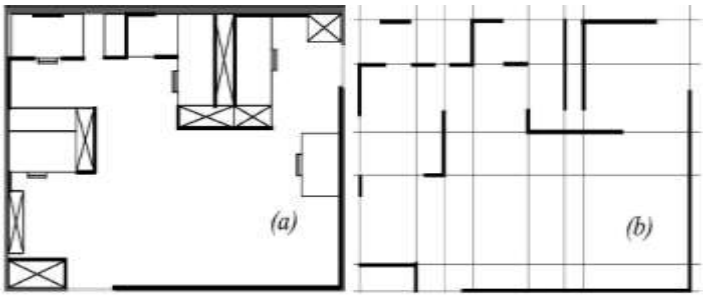

Fig. 6 CONTINUOUS map representation

B. Cell Decomposition

Divide the environment into a number of connected regions. The shape of the region or cell can be vertical strip cells, array of rectangular grid, or unequal size rectangular grid**[4]**. It transforms real environment into discrete by selecting features from the environment and neglecting others,

thus map representation can potentially be minimized, but at the same time it causes a loss of fidelity between the map and the real world **[2]**.

An environment partitioned to cells can be presented with a state transition graph where states are certain points inside the cells and connections among the states are possible only between neighbor cells [1]

Problem in cell decomposition required a lot of memory to analyze the workspace and gave rise to high computational complexities[13].

❖ **Accurate/ Exact cell decomposition**

This method achieves decomposition by selecting boundaries between discrete cells based on geometric criticality**[2]**.

An example of accurate decomposition to cells is vertical decomposition, shown in Fig. 7.

This decomposition can be obtained by using an imaginary vertical line traveling from the left environment border to the right border.

Assumption behind exact decomposition is that the particular position of a robot within each area of free space does not matter. What matters is the robot's ability to traverse from each area of free space to the adjacent areas **[2]**.

Exact decomposition is not always suitable. If the gathering of this information (obstacles of the private environment and free space) is costly or even unknown, then such an approach is not feasible **[2]**.

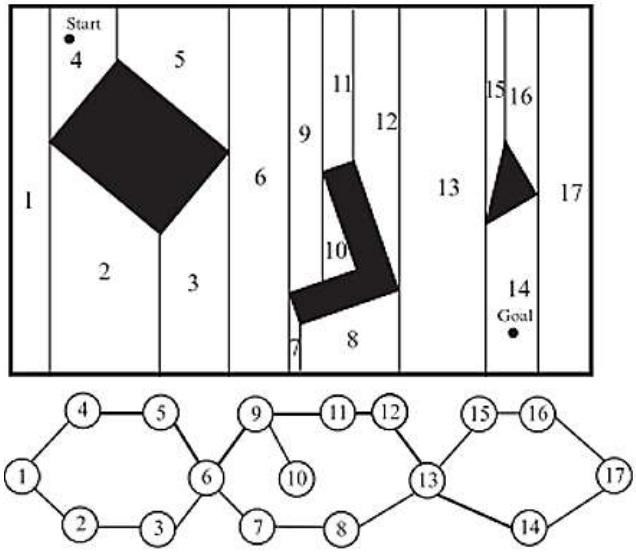

Fig. 7 An example of an exact decomposition

❖ **Fixed cell decomposition**

Fixed decomposition, in which the world is tessellated, transforming the continuous real environment into a discrete approximation for the map.

This approach stems from its inexact nature. It is possible for narrow passageways to be lost during such a transformation, as shown in Fig. 8; this means that fixed decomposition is sound but not complete. Yet another approach is adaptive cell decomposition, as presented in Fig. 8.

Fixed decomposition is extremely popular in mobile robotics; it is perhaps the single most common map representation technique currently utilized. **[2]**

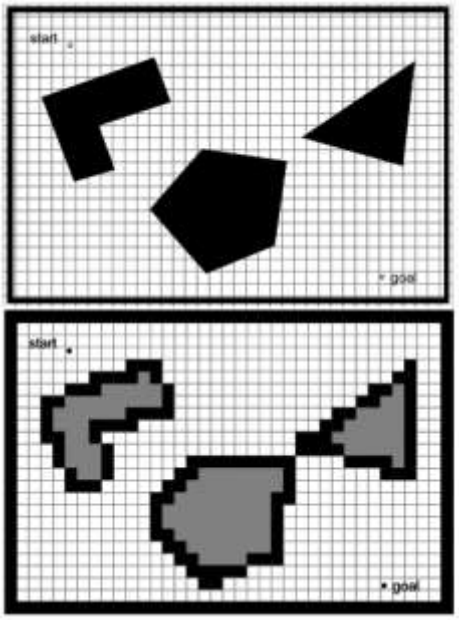

Fig. 8 Fixed localization

❖ **Adaptive / approximate cell decomposition**

In approximate cell decomposition**[3]**, the resulting cells may be free, completely occupied or mixed, or have reached an arbitrary resolution threshold.

One possible way to apply it is to use the occupancy grid method. A possible occupancy grid can be obtained by assigning to each of the cells a value that relates to the probability of this cell's occupation. The decomposition threshold defines the minimum required probability for a cell in the occupancy grid to be deemed occupied.

The idea of "approximate" is to fuse neighboring free cells into larger cells to allow a fast path determination.

In 2D workspaces, approximate cell decomposition operation consists in recursively subdividing each cell that contains a mix of free and obstructed space into cells. Because of that, this method is known as quadtree method.

The recursion is stopped if when each cell is found to contain entirely free or obstructed space or the maximum desired resolution is reached. The height of the decomposition is the maximum allowable level of recursion, and specifies the resolution of the decomposition.

Fig. 109shows an application example of the approximate cell decomposition method. The workspace contains 3 obstacles and the robot has to move from S to G. The result of approximate decomposition is a drastic reduction of the number of cells to be considered. In an example, a high resolution map that contains 250,000 cells, with a crude decomposition of height 4 was reduced to just 109 cells [37]. A problem that has to be faced here is to determine cell

adjacency (i.e., to find which cells share a common border or edge with another one). Actually, many techniques are available to solve the adjacency problem. One of them is to use tesseral (or quad tesseral) addressing which has the ability to map every part of a 2D (or nD) spatial domain into 1D sequence, and when stored with attributes in a database, each address can perform as a single key to data. To generate tesseral addresses, the positive quadrant of 2D-Cartesian space is quartered to give parent tiles with labeling as shown in Figure 11.9A. This process can be continued with new tesseral addresses generated by always appending to the right of the parent addresses, until a desired depth is reached. Another solution to global path planning, via decomposition, is to use local node refinement, path nodes refinement, and curve parametric interpolation. A quad tesseral address can be stored in a quadtree structure. For example, the address of can be stored by the quadtree structure of as shown in Fig. 9

If any big rectangle contains obstacles or boundary, then it is divided into 4 small rectangular, all the larger grid is executed this operation, the operation is repeated until it reaches the solution boundaries. This structure is called quadtree shown[15]

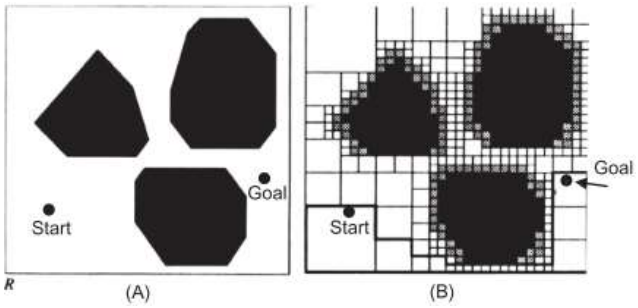

Fig. 9 quadtree structure

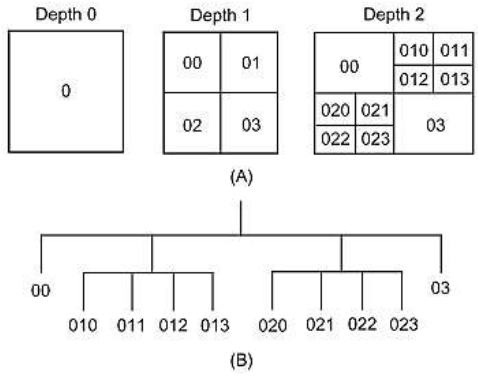

Fig. 10 quadtree method

❖ Topological cell decomposition

The topl method is method of reducing dimensions, and the path plaologicanning problem in high dimensional geometry space is transformed into the discriminant problem of connectivity in low dimension[15]. This method avoids direct measurement of the engineering properties of the environment and focuses on the environmental characteristics most relevant to a robot (may be are not important but useful for localization) **[2]**. The environment is represented in the form of nodes and transitions between them. The nodes represent areas of the environment (they can be of different sizes) based on certain features that help to identify them when entering and exiting the node, and the transitions represent the contiguity between the areas. **[2]**

Compared to the cell decomposition approach, this method only needs less model building time and less storage space, the complexity of the topological method only depends on the number of obstacles, it can achieve fast path planning.

Topology method is suitable for the environment with obvious characteristics and sparse obstacles; otherwise, it is difficult to carry out reliable navigation control. Topology method of environment information is not easy to maintain, when the number of obstacle is increased or decreased, the network is hard to modify, because the process of establishing the topology network itself is quite complex[15].

Fig. 11shows a topological representation of a set in an office in indoor environment. In this case, the robot must have an intersection detector (sonar and vision) to find intersections between halls and between halls and rooms. Nodes capture geometric space and arcs in represent connectivity.

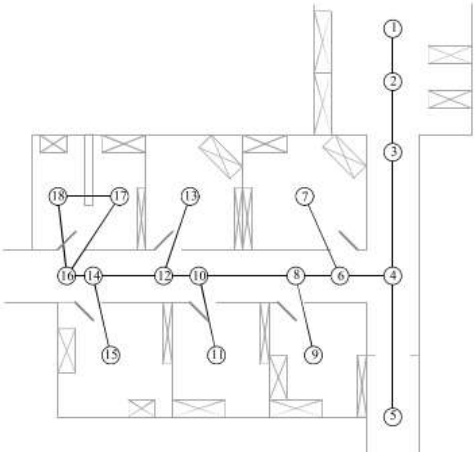

Fig. 11 Topological decomposition of a cell for building a map.

C. Roadmaps

A roadmap is a map that contains roads, consists of lines, curves, and their points of intersection, gives possible connections between points in the free space**[1]**.

Path planning is connecting between the start point and the goal point with an existing road connection in the map to find a connecting sequence of roads. A roadmap depends on environment geometry. The challenge is to find a minimum number of roads that enable a mobile robot to access any free part of the environment.

❖ Visibility Graph.

A visibility graph consists of all possible connections among any two vertices that lie entirely in the free space of the environment.

The start point and the goal point are treated as vertices. Connections are also made between neighboring vertices of the same polygon. An example of a visibility graph is given in Fig. 12Visibility graphs are simple to use but the number of

road connections increases with the number of obstacles, which can result in higher complexity and therefore lower efficiency. A visibility graph can be simplified by removing redundant connections that can be replaced by existing shorter connections.

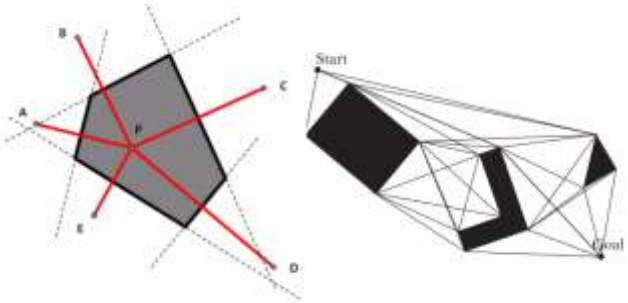

Fig. 12 Visibility graph for map representation..

❖ **Voronoi Graph**

A Voronoi graph consists of road sections that have the largest distance from the obstacles. This also means that the road between any two obstacles has equal distance to both obstacles. [12][11].

It partitions the plane into n regions whose borders define the roadmap. As shown in Fig. 14, the plane environment consisting of obstacles of any shape (square, straight line, etc.). Space is partitioned to regions where each region has exactly one generating obstacle. Any point in a certain region is closer to the generating obstacle than to any other obstacle. Borders between the regions define the roadmap. Driving on such a road minimizes the risk of colliding with obstacles, which can be desired when the robot pose is known with some uncertainty (due to measurement noise or control).

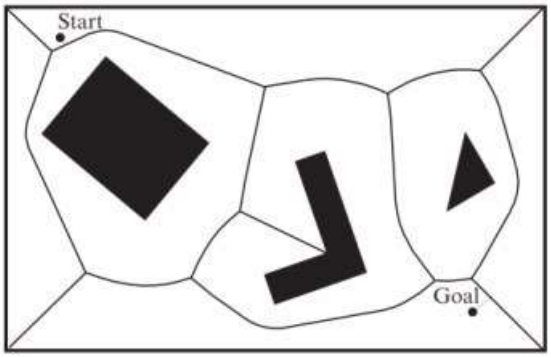

Fig. 13 Voronoi graph for map representation.

This approach maximizes robot distance to the obstacles. However, the obtained path length is far from being the optimal (shortest) one. A robot with distance sensors (such as an ultrasonic or laser range finder) can apply a control to drive equally away from all the surrounding obstacles, which means that it follows roads from the Voronoi graph. Although robots using only touch or vicinity sensors cannot follow Voronoi roads because they may have problems with localization, they can easily follow roads in the visibility graph.

❖ **Triangulation**

This Approach splits the environment into triangular cells. One possible algorithm is the Delaunay triangulation[12], which is a dual presentation of the Voronoi graph. In a Delaunay graph, the center of each triangle (center of the circumscribed circle) coincides with each vertex of the Voronoi polygon. An example of the latter is given in Fig. 14

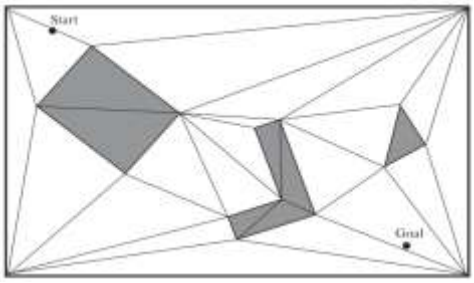

Fig. 14 Triangulation for building a map.

2. PATH PLANNING

Finding visible collision free path that will drive the robot from the start to the goal configuration. Which action is chosen in the current state and which state will be next depending on the used path planning algorithm and used criteria. The algorithm chooses the next most suitable state from the set of all possible states that can be visited from the current state. This decision is made according to some criteria function, usually defined with one of the distance measures, such as the shortest Euclidean distance to the goal state.[1]

Path planning not only save a lot of time but also reduce the wear and capital investment of mobile robot[4]

Between certain start and goal states, there may be one or more paths, or there is no path at all that connects the states. Usually there are several feasible paths (i.e., paths that do not collide with obstacles).

❖ **Critical**

Used in the evaluation of planning algorithms [1] [15] [4] [11]: completeness: is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?

- Cost Optimality: Does it find a solution with the lowest path cost of all solutions**?**
- Time Complexity: How long does it take to find a solution? This can be measured in seconds, or more abstractly by the number of states and actions considered.
- Space Complexity: How much memory is needed to perform the search?
- Path length: Through these criteria, the main goal is to have the shortest path as possible. In order to obtain the total path length, all sub length from the source point to destination point will be total up as the following formula: Path length =

- ❖ $\sum_{i=0}^{n-1} \sqrt{(y_{i+1} - y_i)^2 + (x_{i+1} - x_i)^2}$
  - ❖ n represent the number of nodes from the source point to its target.
- Number of examined cells.
- Symmetry of examined.
- Path's smooth: the path must be smooth without sharp turns, in other words Smoothness objective tries to have a straight path as possible and this will help to reduce energy minimal because turns in a straight way uses a lot less memory than a curvy path. Path smoothness can be calculated through the following equation:
- Pathsmoothness $=\sum_{i=1}^{n}(180 - \theta i)$
- n: number of angles formed from starting position to target destination, θi: value of angle for $1 \leq i \leq n$. the graphical explanation of this objective with n =4, is shown in Fig. 15

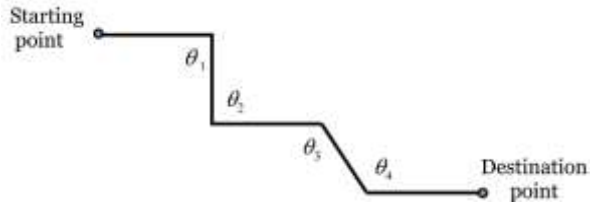

Fig. 15 Path's smoothness explanation

- **Number of turning of robot.**
- **The path should be as far as possible from the obstacles; the path must consider motion constraints (e.g., no holonomic constraints, where at the current time not all driving directions are possible).**

❖ PATH PLANNING ALGORITHMS

A. Global vs local path planning

In global navigation, the mobile robot knows the location of obstacles, type of environment and its target point [4] , in other words Global path planning can be performed only if the environment (obstacles, etc.) is static and perfectly known to the robot(environment is assumed to be a priori known[3]). In this case, the path planning algorithm produces a complete path from the start point to the goal point before the robot starts its motion[3].

Global navigation is based on classical approaches such as cell decomposition, roadmap algorithm and AFP[4].

In local navigation, is more challenging where the location of the obstacles is dynamic. [4]. Local navigational approaches are more intelligence since during movement the robot need to interact with the dynamic environment, taking data from local sensors and execute plan autonomously[4][3].

A. Graph based algorithms (Uninformed Search Strategies)

An uninformed search algorithm is given no clue about how close a state is to the goal(s)[11].

❖ Breadth-first search(BFS)

The root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on. As shown in Fig. 16.a [11]

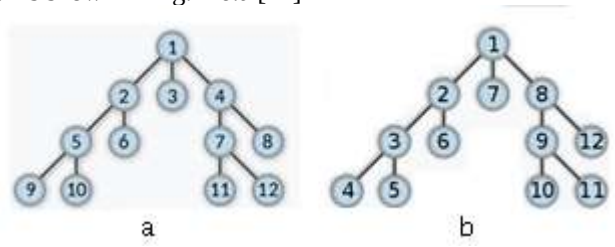

Fig. 16 Depth first search, b: Breadth first search.

❖ Depth-First Search (DFS).

Depth-first search Always expands the deepest node in the frontier first[30], in other words In contrast to breadth-first search, depth-first search expands each node up to the deepest level of the graph (until the current node has no further successors). As those nodes are expanded, their branch is removed from the graph and the search backtracks by expanding the next neighboring node of the start node until its deepest level and so on, as shown in **Error! Reference source not found.**b. An inconvenience of this algorithm is that it may revisit previously visited nodes or enter redundant paths. However, these situations may be easily avoided through an efficient implementation. A significant advantage of depth-first over breadth-first is space complexity. In fact, depth-first needs to store only a single path from the start node to the goal node along with all the remaining unexpanded neighboring nodes for each node on the path. Once each node has been expanded and all its children nodes have been explored, it can be removed from memory.[2]

The depth-first search is not complete. In the case of an infinite graph (with infinite branch that does not end), it can get trapped in one branch of the graph; or in the case of a loop branch (loop in finite graph depth), it can get stuck in cycles. To avoid this problem, the search can be limited to a certain depth only, but then the solution can have a higher depth than the maximum depth limitation.

The algorithm is also not optimal because the found path is not necessarily the shortest one also. This method has low memory usage as it only stores the path from the start node to the current node and intermediate nodes that have not been explored yet. When some nodes and all of their successors are explored, this node no longer needs to be stored in the memory.[1]

❖ Iterative Deepening Depth-First Search (IDDFS).

This algorithm combines advantages of the breadth-first search and depth- first search algorithms. It iteratively increases the search depth limit and explores nodes using the depth-first search algorithm until the solution is found.[1]

Its memory requirements are modest: O(bd) when there is a solution, or O(bm) on finite state spaces with no solution and The time complexity is O(bd) when there is a solution, or O(bm) when there is none.

Like breadth-first search, iterative deepening is optimal for problems where all actions have the same cost, and is complete on finite acyclic state spaces, or on any finite state space when we check nodes for cycles all the way up the path[11].

First, the depth-first search is performed for the nodes that are zero steps away from the starting node. If the solution is not found, the search in the depth is repeated for nodes that are up to one step away from the starting one and so on, as shown in Fig. 17. This algorithm is complete (the solution is found if it exists), and it has small memory usage and is optimal if the cost of all transitions are equal or if transition costs increase with the node depth. If all the nodes have approximately the same rate of branching, then the repeated calculation of nodes is also not a big computational burden because the majority of the nodes are in the bottom of the tree, and those nodes are visited only once or a few times.

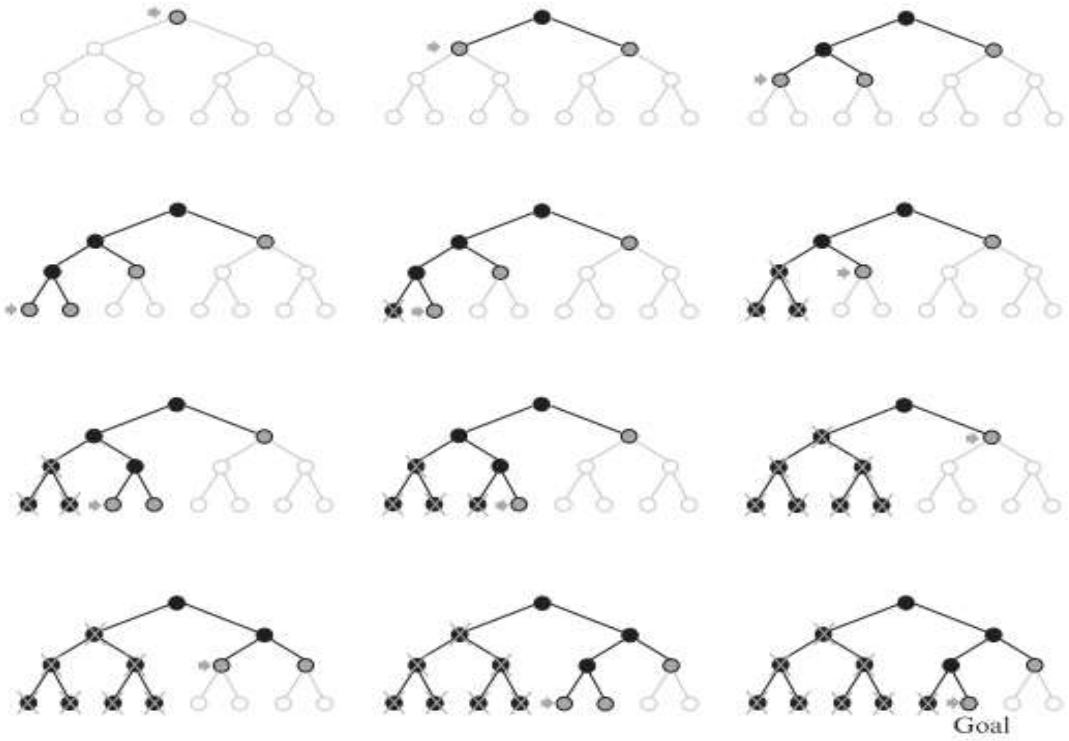

Fig. 17 Iterative Deepening Depth-First Search (IDDFS)

❖ **A*.**

A* is best-first search that uses the evaluation function[11]:

f(n) = g(n) + h(n)

where g(n) is the path cost from the initial state to node and h(n) is the estimated cost of n, the shortest path from to a goal state, so we have f(n) = estimated cost of the best path that continues from n to a goal.

As we said, it includes additional information or heuristic. Heuristic is the cost estimate of the path from the current node to the goal that is for the part of the graph tree that has not been explored yet. This enables the algorithm to distinguish between more or less promising nodes, and consequentially it can find the solution more efficiently[1]

For each node, the algorithm computes the heuristic function that is the cost estimate for the path from this node to the goal, and it is called cost-to-goal. This heuristic function can be Euclidean distance or Manhattan distance (sum of vertical and horizontal moves) from the current node to the goal node. The heuristic can be computed also by some other appropriate function.

The A∗ algorithm is guaranteed to find the optimal path in graph if the heuristic for calculation of the cost-to-goal is admissible (or optimistic), which means that the estimated cost-to-goal is smaller or equal to true cost-to-goal.

The A∗ algorithm is a complete algorithm because it finds the path if it exists, and as already mentioned, it is optimal if the heuristic is admissible (optimistic). Its drawback

is large memory usage. If all costs-to-goal are set to zero, the A∗ operation equals Dijkstra's algorithm. In Fig. a comparison of Dijkstra's and A∗ algorithm performance.

Average time complexity is O(k·logkv) for v nodes with branching factor k, but can be quadratic in worst case.[6]

AS shown in Fig. 18, Node values are lower bound distances to goal b (e.g. linear distances) Arc values are distances between neighbouring nodes[4, p. 210]

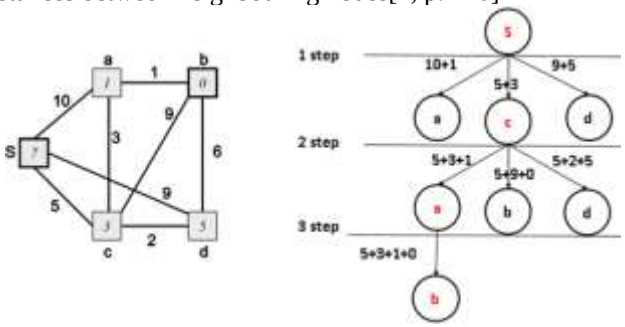

Fig. 18 Algorithm A∗.

The quality of the lower bound goal distance from each node greatly influences the timing complexity of the algorithm. The closer the given lower bound is to the true distance, the shorter the execution time.

❖ **Dijkstra's Algorithm.**

Algorithm for computing all shortest paths from a given starting node to all the other nodes in a fully connected graph. Time complexity for naive implementation is O(e + v2), and can be reduced to O(e + v·log v), for e edges and v nodes. Distances between neighboring nodes are given as edge(n,m).[6]

This algorithm needs Relative distance information between all nodes; distances must not be negative.

Start "ready set" with start node. In loop select node with shortest distance in every step, then compute distances to all of its neighbors and store path predecessors. Add current node to "ready set"; loop finishes when all nodes are included, this algorithm shown in Fig .20**Error! Reference source not found.** [7]

❖ **Greedy best-first.**

This is the informed algorithm. The open list is sorted in the increasing cost-to-goal. Therefore, the search in each iteration is extended to the open node that is closest to the goal (has the smallest cost-to-goal) assuming it will reach the goal quickly. The found path is not guaranteed to be optimal as shown in Fig. 19. The algorithm only considers cost from the current node to the goal and ignores the cost required to come to the current node; therefore, the overall path may become longer than the optimum one. Because the algorithm is not optimal it is also not necessary that the heuristic is admissible as it is important in A∗[1].

Comparison of greedy best-first search (above) and A∗ (below) algorithm.

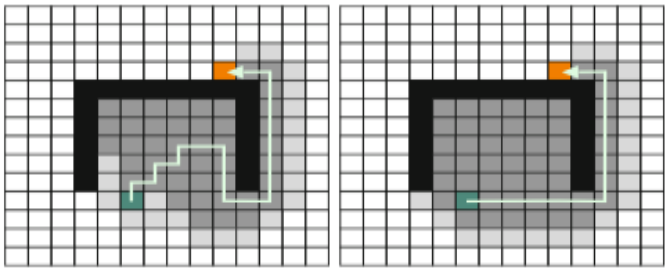

Fig. 19 Greedy best-first.

❖ **.D* algorithm.**

represents an incremental re planning version of A* in other words algorithm reuse previous search effort in subsequent search[2].

At first robot had provided with a crude map of the environment (i.e. obtained from an aerial image). Path planning is done by employing A*. After executing this path for a while, the robot observes some changes in the environment with its onboard sensors. Subsequent to updating the map, a new solution path needs to be computed. Instead of generating a new solution from scratch (as A* would do), only states affected by the added (or removed) obstacle cells are recomputed. Because changes to the map are most often observed locally (due to proprioceptive sensors), the planning problem is usually reversed; node expansion begins from the robot goal state. In this way, large parts of the previous solution remain valid for the new computation. Compared to A* this may decrease search time by a factor of one to two orders of magnitude.

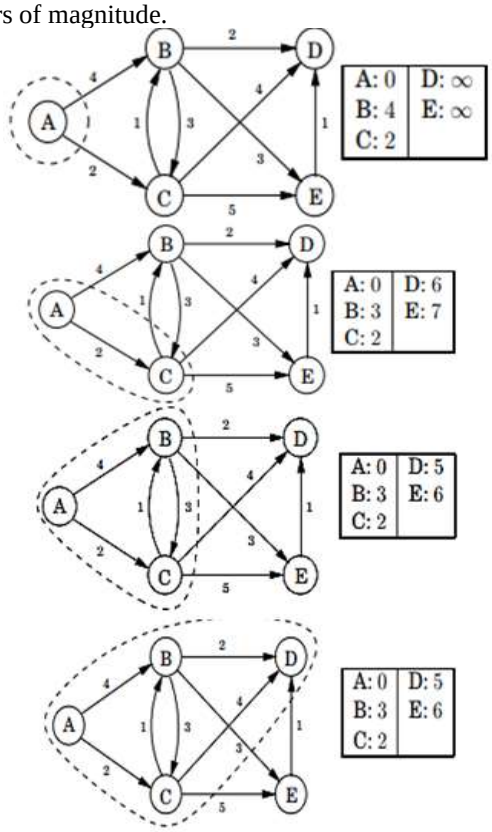

Fig. 20 Dijkstra's Algorithm

❖ **Comparing uninformed search algorithms**

Is the branching factor; is the maximum depth of the search tree; is b m d the depth of the shallowest solution, or is when there is no solution; is the depth limit. [11]

3. Randomized graph search.

When faced with complex problems of planning a path of large dimensions (such as the tasks of manipulating robotic arms or requests for folding and joining molecules to place drugs, etc.), it becomes impossible to solve them comprehensively in a reasonable time. Reverting to heuristic search methods is often impossible due to the lack of a corresponding heuristic function, and the reduction of the problem dimension often fails due to the speed and acceleration constraints imposed on the model, which should not be violated for security reasons. In such situations, randomized search becomes useful because it abandons the optimality of the solution to compute the solution faster [2].

❖ **Rapidly Exploring Random Trees (RRTs).**

RRTs typically grow a graph online during the search process and thus a priori only require an obstacle map but no graph decomposition[15].

A Rapidly-exploring Random Tree (RRT) is designed for efficiently searching nonconvex high-dimensional spaces. RRTs are constructed incrementally in a way that quickly reduces the expected distance of a randomly chosen point to the tree. RRTs are particularly suited for path planning problems that involve obstacles and differential constraints (nonholonomic or kinodynamic)[16].

RRT only require an obstacle map but no graph decomposition.

The algorithm begins with an initial tree (which might be empty) and then successively adds nodes, connected via edges, until a termination condition is triggered. During each step[14], as shown in Fig. 21

- select a random configuration xrand in the free space.
- the tree node that is closest to xrand is computed, denoted as xnear.
- Connect xnear and xrand.
- A new node xnew from xnear is generated with a certain step size ρ.
- If there is no collision with the obstacle during the expansion from xnear to xnew, this new node xnew is added to the random tree to generate a random tree.
- When a child node in a random tree contains a target point xgoal, a path from the initial point xrand to the target point xgoal can be generated in the random tree. Conversely, if a collision occurs, then we discard the expansion.

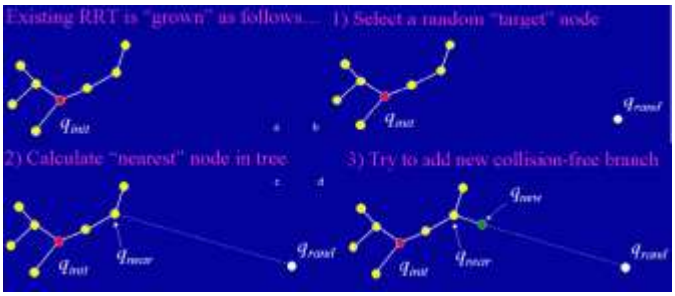

Fig. 21 Rapidly Exploring Random Trees (RRTs).

An RRT alone is insufficient to solve a planning problem. Thus, it can be considered as a component that can be incorporated into the development of a variety of different planning algorithms[16].

❖ **The bidirectional RRT.**

This algorithm as shown in Fig. 22 defines two random trees in the free space, which select the starting point and the ending point as the random root node, respectively, expanding in the opposite direction[15]. The expansion is ended until the two trees meet. That is to say, the search path is found. When the random tree with the starting point as the root node searches for the free space to establish the random tree, the random tree with the ending point as the root node is also established. The two random trees generate a new node by turns and detect whether the Euclidean distance between the new node and the other random tree node is less than the set threshold. When the distance between two nodes is less than the set threshold, the two nodes are connected, that is, the two random trees are merged into one random tree to generate a path.

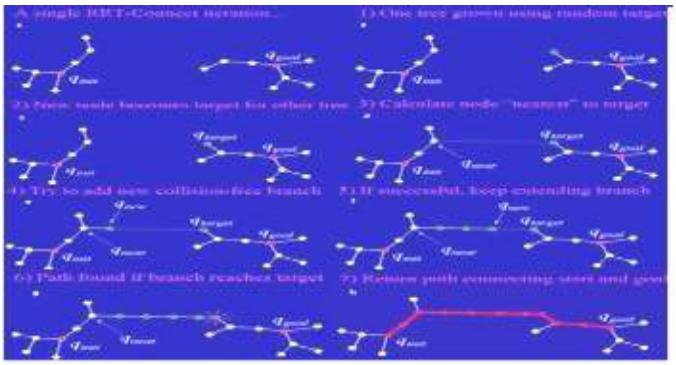

Fig. 22 The bidirectional Rapidly Exploring Random Trees

❖ **Probabilistic roadmap (PRM).**

This method used for path searching between more start points and more goal points.

The algorithm has two steps:

Learning phase where a roadmap or undirected graph of the free space is constructed.

Connecting the current start and goal point to the graph and some graph path searching algorithm is used to find the optimum path.

B. wave algorithm (LI algorithm)

Algorithm for finding the shortest path on a planar graph. Belongs to algorithms based on Breadth First Search methods.

It is mainly used in computer tracing (wiring) of printed circuit boards, connecting conductors on the surface of microcircuits. Another application of the wave algorithm is finding the shortest distance on a map in computer strategy games.

From the initial element, the wave propagates in 4 directions as shown as in Fig. 23. The element to which the wave came forms a wave front. In the figures, numbers indicate the numbers of the wave fronts.

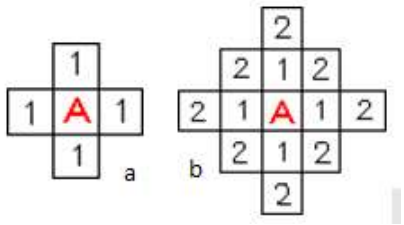

Fig. 23 Lee algorithm working

Each element in the front of the first wave is the source of a secondary wave as shown as in Fig. 24. The elements of the second wave front generate a wave from the third front, etc. The process continues until the end element is reached.

Build the path itself. It is built according to the following rules:

Movement takes place when building the track according to the priorities chosen.

When going from the terminal element to the initial number of the wave front (the trajectory coordinate) it must decrease.

Operation of the algorithm includes three stages: initialization, wave propagation, and path recovery.

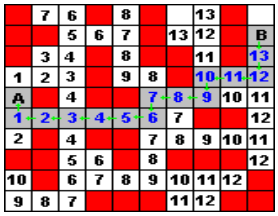

Fig. 24 Lee algorithm working.

The advantages of the wave algorithm are that it can be used to find a track in any maze and with any number of forbidden elements (walls). The only drawback of this algorithm is that the Wave propagates in all directions, so the algorithm is slow and requires a lot of memory.

C. Obstacle avoidance Algorithms

The status of an obstacle may be static (when its position and orientation relative to a known fixed coordinate frame is invariant in time), or dynamic (when either its position), or orientation or both change relative to the fixed coordinate frame change.

The nature of an obstacle is described via its configuration that may be convex shaped, concave shaped, or both.

There are many algorithms used for Obstacle avoidance like **Bug algorithm,** Potential field methods, Vector field histogram, Dynamic window approaches, Nearness diagram, Bubble land technique, Curvature velocity techniques, Schleged, Gradient.

❖ **Bugs algorithm**

Local planning algorithm, In the bug algorithm, the main concept is to track the contour of the obstacles found in the robot's path and circumnavigate it[13].

The Algorithm assume only local knowledge and do not need a map of the environment and they are suitable in situations where an environment map is unknown or it is changing rapidly and also when the mobile platform has very limited computational power**[1]**.

These algorithms operation consists of two simple behaviors: motion in a straight line toward the goal and obstacle boundary following.

The advantages in these algorithms that they require low memory usage but the obtained path is usually far from being optimal.

- **Bug0 Algorithm.**

This algorithm operate two basic behaviors[1]:

- Move toward the goal until an obstacle is detected or the goal is reached.
- If an obstacle is detected, then turn left (or right, but always in the same direction) and follow the contour's obstacle until motion in a straight line toward the goal is again possible.

As shown in Fig. 25.a.

Bug0 algorithm successfully finds a path to the goal in the environment on the left while it is unsuccessful in the environment on the right.

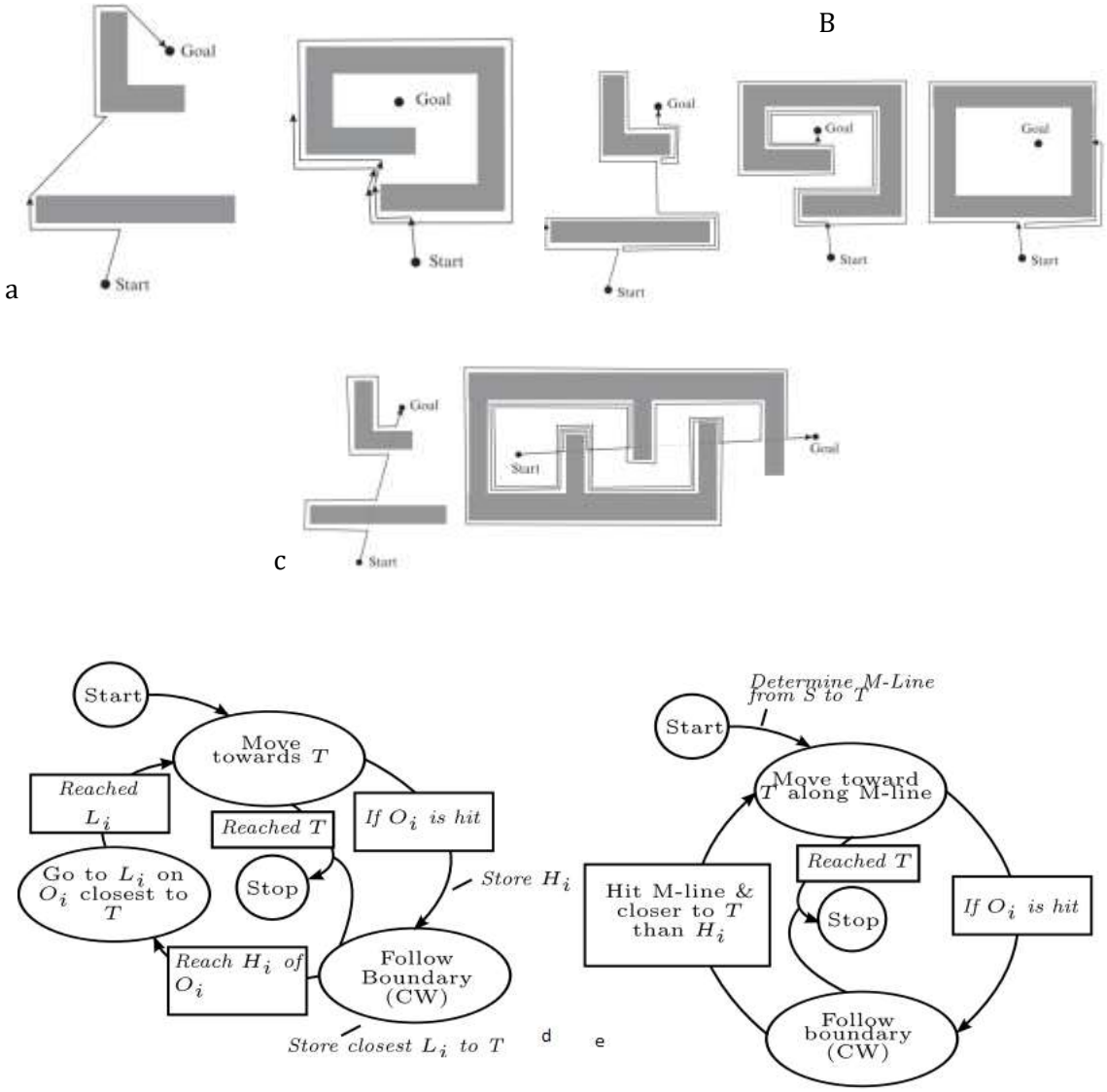

Fig. 25 BUG algorithms (a.bug0 , b,d .bug1, c,e.bug2)

- **Bug1 Algorithm**.

In contrast to Bug0, Bug1 uses more memory and requires several more computations. In each iteration it needs to calculate Euclidean distance to the goal and remember the closest point on the obstacle circumference to the goal **[1]**. Bug1 operations are shown in Fig. 25.d.

- Move on the straight line toward the goal until an obstacle is hit or the goal is reached.
- If an obstacle is detected, then turn left and follow the entire contour of the obstacle and compute the Euclidean distance to the goal. When the point where the obstacle was initially detected is reached again, follow the contour of the obstacle in the direction that is shortest to the point on the contour that is closest to the goal. Then resume moving toward the goal in a straight line. For example Fig. 25.a

The generated path is not optimal and is in the worst case for 3 2 of the length of all obstacle contours longer than the Euclidean distance from the start to the goal configuration.

It detects the obstacle once and therefore never circles among the same obstacles, because of for each obstacle that is detected on its path from the start to the goal; the algorithm finds only one entry point and one leaving point from the obstacle contour.

When the algorithm detects the same obstacle more than once it knows that either the start or the goal point is captured inside the obstacle and the path searching can be terminated Fig. 25.c.

- **Bug2 Algorithm**.

The Bug2 algorithm always tries to move on the main line that is defined as a straight line connecting the start point and the goal point**[1]**. Bug2 operations are shown in Fig. 25.f.

- Comparison Bug1 and Bug2.
  - Bug1 is the more thorough search algorithm, because it evaluates all the possibilities before making a decision.
  - Bug2 is the greedy algorithm, because it selects the first option that looks promising.
  - In most cases, the Bug2 algorithm is more efficient than Bug1 but, the operation of the Bug1 is easier to predict, but Bug1 but it does not guarantee that the robot detects certain obstacles only once, in other words, robot can encircle the same obstacle more than once; therefore, unnecessary circling can occur until it reaches the goal.

❖ Potential Field Method

The idea of imaginary forces acting on a robot has been suggested by Khatib[18]. It is global map generation algorithm with virtual forces, Algorithm needs Start and goal position, positions of all obstacles and walls [6]

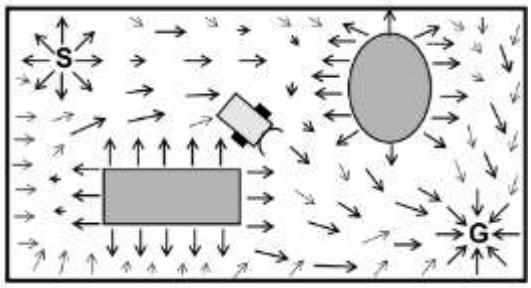

Fig. 26 Potential Field Method

This method is very attractive because of its simplicity and elegance**[3]**. This method assumes an artificial force field consists of attractive and repulsive force in the environment. The goal will produce attractive force that attract the mobile robot to move towards it. While, the obstacles generate a repulsive force and is pointing out from the obstacles. These imaginary forces attract the robot towards the goal by avoiding all obstacles.[13] as shown in Fig. 26. The algorithm enables real-time operations of a mobile robot in a complex environment[8].

The problem in this algorithm that the robot can get stuck in local minima. In this case the robot has reached a spot with zero force (or a level potential), where repelling and attracting forces cancel each other out. So the robot will stop and never reach the goal.[6][13]

❖ Vector field histogram

This is a real-time obstacle avoidance method for mobile robots and permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot toward the target[5].

The algorithm computes obstacle-free steering directions for a robot based on range sensor readings[20].Range sensor readings are used to intermediate data structure about the local obstacle distribution, called polar histogram which is an array of, say, 72 angular sectors to identify obstacle location and proximity. Based on the specified parameters and thresholds, these histograms are converted to binary histograms to indicate valid steering directions for the robot[20].

To take into account the robot changing position and the new sensor readings, the polar histogram is totally updated and rebuilt every, say, 30 ms (sampling period)**[3]**.

The method Fig .27involves two steps)[5]**[3]**:

- Reduce the histogram grid is to one-dimensional polar histogram which is built around the robot's instantaneous location. Each sector in the polar histogram involves a value that represents the polar obstacle density (POD) in this direction.
- The robot moves in the direction, which have low POD.

To achieve these steps, a window (called active window) moves with the robot, overlying a square region of cells (e.g., 33*33) in the histograms.

We must know that active cells are cells that, each time, lie on the moving window. The cell that lies on the sonar axis and corresponds to the measured distance d found by each range reading is incremented and increases the certainty value (CV).

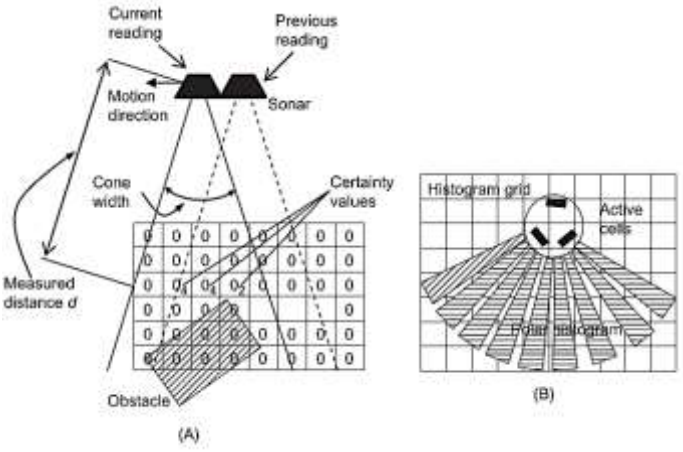

Fig .27 Vector field histogram

The VFH method is a local path planner, i.e., it does not attempt to find an optimal path (an optimal path can only be found if complete environmental information is given). Furthermore, a VFH-controlled robot may get "trapped" in dead-end situations (as is the case with other local path planners). When trapped, mobile robots usually exhibit what has been called "cyclic behavior.

❖ Wandering Standpoint Algorithm

This algorithm is Local path planning algorithm and need Local distance sensor.

Principle working is: Try to reach goal from start in direct line. When encountering an obstacle, measure avoidance angle for turning left and for turning right, turn to smaller angle. Continue with boundary-following around the object, until goal direction is clear again[6].

AS show in Fig. 28, the goal is not directly reachable from the start point. Therefore, the robot switch to boundary-following mode until, at point 1, also the goal is not directly reachable. It repeats boundary-following (point 2, 3, 4, 5) until the goal is directly reachable in a straight line without further obstacles (point 6).

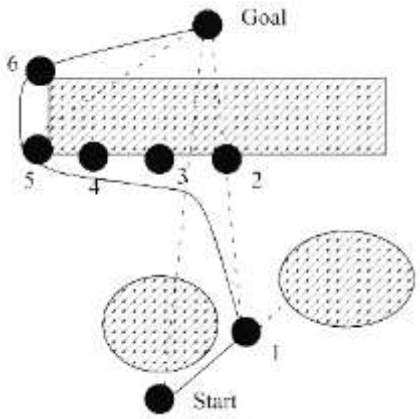

Fig. 28 Wandering Standpoint Algorithm

The disadvantage is that this algorithm can lead to an endless loop for extreme obstacle placements. In this case, the robot keeps driving, but never reaches the goal.

CONCLUSION

This paper presented the basic principles of the mobile robot and several aspects of localization for mobile robot and different path planning algorithms. The advantages and disadvantages of each of them are explained and their potential applications. In the near future is planned to use some of these algorithms to reduce the state space as a first stage for path planning and demonstrate the extent to which these algorithms are applied in stm32 microcontrollers embedded within the robots.

REFERENCES

[1] Oktaviani.J, *Introduction to Autonomous Mobile Robots*, Second., vol. 51, no. 1. 2018.

[2] S. Edition, *Robotics for ouy*, Second. Springer International Publishing AG.

[3] S. G. Tzafestas, *Introduction to Mobile Robot Control*, First. Elsevier, 2013.

[4] T. Bräunl, *Embedded Robotics*. 2003.

[5] I. Š. G. Klanˇcar, A. Zdešar, S. Blaži ˇc, *Wheeled Mobile Robotics*. Elsevier, 2017.

[6] S. Florczyk, *Robot Vision: Video-Based Indoor Exploration with Autonomous and Mobile Robots*. 2005.

[7] N. Adzhar, Y. Yusof, and M. A. Ahmad, "A review on autonomous mobile robot path planning algorithms," *Adv. Sci. Technol. Eng. Syst.*, vol. 5, no. 3, pp. 236–240, 2020.

[8] F. Rubio, F. Valero, and C. Llopis-Albert, "A review of mobile robots: Concepts, methods, theoretical framework, and applications," *Int. J. Adv. Robot. Syst.*, vol. 16, no. 2, pp. 1–22, 2019.

[9] H. Y. Zhang, W. M. Lin, and A. X. Chen, "Path planning for the mobile robot: A review," *Symmetry (Basel).*, vol. 10, no. 10, 2018.

[10] W. Pokojski and P. Pokojska, "Voronoi diagrams – inventor, method, applications," *Polish Cartogr. Rev.*, vol. 50, no. 3, pp. 141–150, 2018.

[11] "Voronoi graph and Delaunay Triangulation Visually Explained | by Dino Cajic | Dev Genius | Medium." [Online]. Available: https://medium.com/dev-genius/voronoi-graph-and-delaunay-triangulation-visually-explained-1df842640c55. [Accessed: 08-Nov-2020].

[12] "Delaunay Triangulation. How to divide a set of scattered points… | by Nabil MADALI | Towards Data Science." [Online]. Available: https://towardsdatascience.com/delaunay-triangulation-228a86d1ddad. [Accessed: 08-Nov-2020].

[13] S. R. and P. Norvig, *Artificial Intelligence A Modern Approach 2nd Ed.*, vol. 53, no. 9. 2019.

[14] S. Dasgupta, C. H. Papadimitriou, and U. . V. Vazirani, "Algorithms, Dynamic programming," pp. 161–162, 2006.

[15] "Research : Rapidly-Exploring Random Trees (RRTs) Algorithm." [Online]. Available: http://www.kuffner.org/james/plan/algorithm.php. [Accessed: 09-Nov-2020].

[16] "RRT Page: About RRTs." [Online]. Available: http://msl.cs.uiuc.edu/rrt/about.html. [Accessed: 09-Nov-2020].

[17] Y. Shi, Q. Li, S. Bu, J. Yang, and L. Zhu, "Research on Intelligent Vehicle Path Planning Based on Rapidly-Exploring Random Tree," *Math. Probl. Eng.*, vol. 2020, 2020.

[18] O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots.," *Int. J. Rob. Res.*, vol. 5, no. 1, pp. 90–98, 1986.

[19] J. Borenstein and Y. Koren, "The Vector Field Histogram—Fast Obstacle Avoidance for Mobile Robots," *IEEE Trans. Robot. Autom.*, vol. 7, no. 3, pp. 278–288, 1991.

[20] "Vector Field Histogram - MATLAB & Simulink." [Online]. Available: https://www.mathworks.com/help/nav/ug/vector-field-histograms.html. [Accessed: 10-Nov-2020].