Voronoi-Based Trajectory Optimization for UGV Path Planning

Evgeni Magid, Roman Lavrenov
Intelligent Robotics Dept., Higher Institute for Information Technology and Information Systems (ITIS)
Kazan Federal University
Kazan, Russian Federation
e-mail: {magid, lavrenov}@it.kfu.ru

Ilya Afanasyev
Intelligent Robotics Systems Lab., Robotics Institute
Innopolis University
Innopolis city, Russian Federation
e-mail: i.afanasyev@innopolis.ru

Optimal path planning in dynamic environments for an unmanned vehicle is a complex task of mobile robotics that requires an integrated approach. This paper describes a path planning algorithm, which allows to build a preliminary motion trajectory using global information about environment, and then dynamically adjust the path in real-time by varying objective function weights. We introduce a set of key parameters for path optimization and the algorithm implementation in MATLAB. The developed algorithm is suitable for fast and robust trajectory tuning to a dynamically changing environment and is capable to provide efficient planning for mobile robots.

Keywords-optimization criteria; voronoi diagram; path planning; Unmanned Ground Vehicle (UGV)

I. INTRODUCTION

Path planning in dynamic environment for autonomous robots is one of the most important tasks in mobile robotics. Nowadays, unmanned ground vehicles (UGVs) execute complex mission scenarios such as monitoring, surveillance, data collection in unknown dynamic environments, search and rescue operations, transport and logistics tasks, etc.

The work presented here is a part of a larger project aimed to define localization, mapping and path planning for a crawler-type UGV with a help of a group of unmanned aerial vehicles (UAVs) using active collaborative vision and multi-robot belief space planning. A significant role for successful motion planning belongs to a proper selection of path cost evaluation function. In this paper, we consider a set of key parameters for path search and optimization. According to this approach, initially we construct an UGV path, using global path planning methods. Then, at the local planning stage, the path is dynamically improved in real time by tuning optimization settings of objective function. The algorithm is implemented in MATLAB, and results of UGV path planning for different cost of optimization setting are presented.

II. MOBILE ROBOT PATH PLANNING

Path planning procedure, which is a search of collision-free trajectory in a well-known or uncertain environment, typically applies two methodologies: (1) computing robot trajectory in a previously known global environment with static obstacles; (2) planning motion while utilizing onboard sensor information about dynamical changes in environment. Traditional path planning methods use both mathematically proven search of optimal trajectories and empirical algorithms like A*, D*, probabilistic roadmaps, potential fields, rapidly-exploring random trees (RRT), etc. [1]. Modern algorithms apply also genetic algorithms, neural networks, bio inspired colonies, evolutionary programming, particle swarm optimization and other optimization techniques [2]. Several frequently used methodologies for global and local path planning are briefly considered below.

A. Global Path Planning in a Static Environment

Global path planning assumes a static environment with a priori known map, with a target and obstacles location are being fixed, and concentrates on path planning between two points (known as point-to-point navigation). One of widely used methods applies potential field approach [3], where a target has some positive charge and obstacles have a negative charge. Next, imaginary forces of attraction and repulsion of charges are acting on a mobile robot and thus support route planning process [4], [5]. While being a classical and well-known approach for decades, potential field method still continues to be used and further modified; e.g., a potential field based path planning algorithm in [6] constructs a gravity chain from an initial to a target position.

Another popular approach, called *Roadmap*, constructs a graph around obstacles for a completely known map. Obstacles are usually formed by closed 2D- or 3D-polygons, and a graph is based on the featured points of obstacles (e.g., polygon corners) that serve as graph nodes; next, each two nodes are getting connected with an edge, if the obstacles do not block the straight line between the nodes. Methods of path planning, which use such graphs, are called *roadmap methods*. Typical examples of roadmap methods are Visibility graph [7] and Tangent graph [8] methods. Algorithm A*, which modernizes Dijkstra algorithm, is one of the simplest methods of path finding on graphs.

A roadmap could be constructed with cell decomposition methods. Two basic methods of partitioning, which built cell-based graphs, are trapezoidal decomposition and Morse decomposition. Sampling based methods accelerate route calculation on large maps. These produce sample measurements with a certain sampling rate for their comparison with routes. Examples of such methods include RRT methods [9] and probabilistic roadmaps [10].

978-1-5090-6530-1/17/$31.00 ©2017 IEEE

At present, Voronoi diagram method is widely used for constructing a roadmap. This approach is most suitable for our task as it guarantees to build a safest route by default as graph edges will be located at maximally available distance from all local obstacles [11], [12]. In addition, graph nodes could support spline-based algorithm [3], which calculates a smoothest trajectory for a car-type robot in a known environment.

B. Local Path Planning in a Dynamic Environment

Local planning methods play an important role in real time path planning tasks for mobile robots. In unknown or uncertain environment a robot must be able to detect obstacles and dynamically recalculate its trajectory to avoid them. In these cases, Bug family methods can be useful for local path planning, when a robot moves towards a target, while negotiating obstacles on its way. The main difference within Bug family algorithm is in a way to deal with obstacles. For example, touch sensor based Bug1 is very conservative and always completes a travel around each obstacle that a robot meets on its way. Bug2 also uses touch sensors only but is more adventurous and leaves an obstacle on a first suitable opportunity. A more advanced DistBug algorithm uses a distance sensor [14]. TangentBug algorithm extends DistBug abilities by creating a local tangent graph (LTG) during robot motion toward a target with a local selection of optimal path [15], whereas the CautionsBug method [16] combines LTG and spiral search approaches in order to guarantee competitive path selection.

Modifications of potential field method for local path planning are based on Vector Field Histogram (VFH). This algorithm and its modifications VFH+ and VFH* use 2D Cartesian grid as a world model, computing an obstacle density in each grid cell [17]. For robot motion planning in dynamic environments, the algorithms based on the above-mentioned A* algorithm, such as Animate Repairing A* (ARA), Dynamic A* (D*), Lifelong Planning A* (LPA*), D* Lite [18], Anytime D* (AD*) are often applied.

With unknown and dynamical environments, global path planning methods and their modifications are frequently used as far as robot's onboard computers allow computing and re-computing a roadmap in real time. In these cases, one of the most popular approach is to use Voronoi diagram-based algorithm that divides the space into a grid with cells, which contain an obstacle or empty area [19]. However, Voronoi-based algorithm tests reveal limitations of this approach, since the algorithm highly depends on grid size and builds a redundant Voronoi diagram with many unnecessary lines.

III. Global Path Planning Algorithm Implementation in MATLAB Environment

Our goal is to compute a UGV path in real time, while taking into account possible dynamic changes of path optimization criteria (e.g., path length, curvature, etc.). We selected Voronoi diagram approach for initial path planning, as it provides a set of safe trajectories for obstacle avoidance. Moreover, when dynamic changes of a map occur, the original graph could be partially recalculated (locally) within a limited time. We use Russian UGV "Engineer" (Fig. 1), which is a crawler-type robot with additional flippers capable of overcoming various obstacles [20].

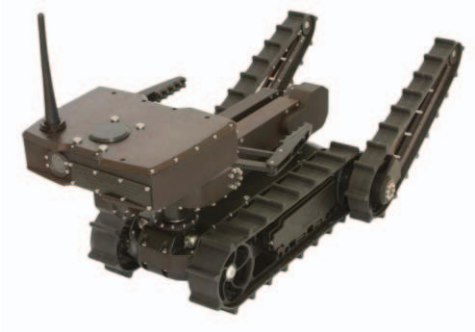


Figure 1. The Servosila "Engineer" crawler-type mobile robot. Courtesy of Servosila company.

We created our software with Voronoi-based path planning algorithm in MATLAB. The algorithm works as follows:

1. Build rays from each obstacle edge with a certain step with the size of half distance between obstacles.
2. Draw points at the rays' intersection.
3. Form segments from these points.
4. Use neighboring segments to build a graph.
5. Use Dijkstra's algorithm with the graph to find a final path.

We modified the algorithm for our task as follows:

- We changed a way of Voronoi diagram construction in order to support all types of obstacles, i.e., we added the ability to use circular and non-convex obstacles with regard to Robot Motion Toolbox package [13];
- We added rays from map borders; this allows graph construction for a map with arbitrary location of obstacles and start and target positions.

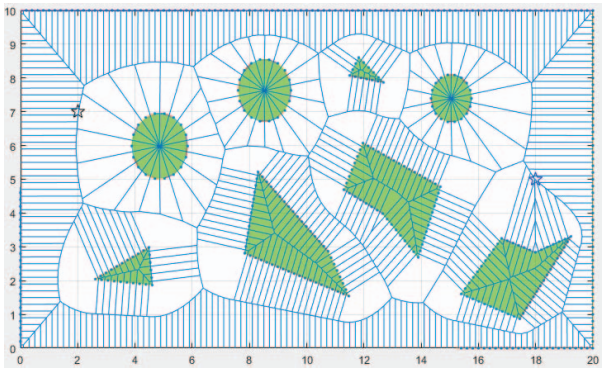


Figure 2. Voronoi diagram-based partition of an obstacle map

An example of trajectory building by our modified algorithm is shown in Fig. 2. To speed up the algorithm we removed ray intersection points from non-convex obstacle edges and segment duplication that allowed to find graph branch nodes (Fig. 3).

Thus, we have changed the Voronoi-based algorithm implementation to allow working with points (e.g.,[19]) and

building a graph for arbitrary positions of obstacles and start/target points. The edges of such graph (i.e. ordered segments between nodes) became *curve lines*. This graph provides an opportunity to calculate path that belong to various homotopy groups[1] within a map.

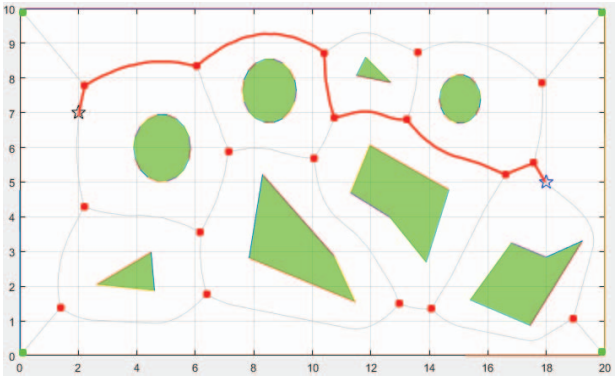


Figure 3. The final Voronoi diagram with the indicated graph branch nodes and optimal point-connected trajectory

IV. Path Optimization Criteria

The computed path must satisfy certain requirements of the objective function and when parameters of the objective function change, the path must be dynamically recalculated each time with regard to these changes. We identified the following key parameters that should be taken in account by the objective function:

- **Path passing time** determines importance of passing a route in a fastest way. The value correlates with trajectory length, smoothness, and a number of other criteria.

- **Path length**, which characterizes the importance of having the shortest path among possible homotopies.

- **Maximum** (or average) **distance to obstacles**. It defines importance of keeping distance to obstacles during motion.

- **Path curvature** points out importance of discontinuous trajectory changes avoidance while following a path [3].

- **Visibility time (line-of-sight time) for start/target points** criteria. For example, if a robot keeps a connection with a monitoring device/router, which helps in path planning, and it is required to be closer to the device as long as possible.

In addition, if there are critical/dangerous points within a map or, on the contrary, attraction points that always should be kept at a certain distance (e.g. for communication routers), then we might set several additional parameters:

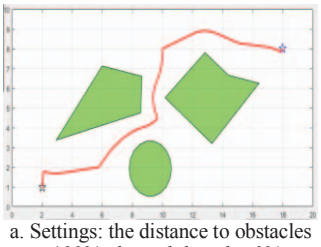
- **Maximum distance to critical points**, which is necessary to be preserved in order to avoid losing contact between a robot and a monitoring device, e.g., while inspecting a map or following a required path.

- **Safe path length rate**, which is defined as a percentage of a sheltered from hazards path length. For example, if there is a radiation source or another source of danger in a critical point on a map, then a mobile robot should follow the safest way (e.g. behind obstacles for protection).

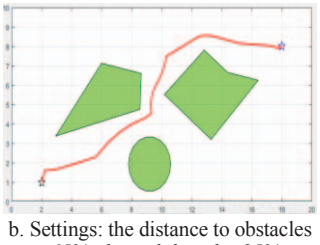
- **Minimum permitted distance to danger points** is the minimal safe distance to a particular dangerous object or location, which a robot should always maintain.

V. Simulation Results

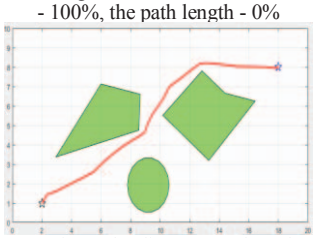
The cost function is responsible for path evaluation and optimization. For MATLAB implementation, a set of cost function parameters includes path length and curvature, distance to obstacles, time to reach a target position and critical points presence. Visibility graph approach [7] calculates the shortest path within the graph in order to verify the results of simulation (for maximal weight of path length parameter) against ground truth. If we increase the maximum distance to obstacles weight parameter, the algorithm selects a path within Voronoi diagram (Fig. 4a-d).



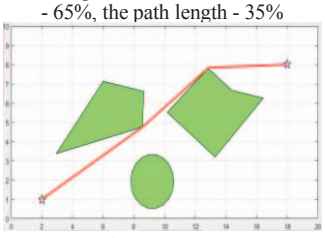
a. Settings: the distance to obstacles - 100%, the path length - 0%



b. Settings: the distance to obstacles - 65%, the path length - 35%



c. Settings: the distance to obstacles - 35%, the path length - 65%



d. Settings: the distance to obstacles - 0%, the path length - 100%

Figure 4. Voronoi diagram-based trajectories for different settings

The trajectories in different homotopies are obtained by replacing the edges in original Voronoi-based path, using settings of visibility time (line-of-sight time) for start/target points. Then, the distances from the start/target points to the first curve of the Voronoi diagram are also compared (Fig. 4a, b). Similarly, the visibility graph vertices substitution allows computing various homotopies within the graph.

Correspondingly, if we apply the spline-based planning method [3] and connect vertices of the Voronoi diagram with a spline (red dots in Fig. 3), the obtained path will satisfy the criterion of minimal total curvature of a trajectory, which is calculated locally and then accumulated within the cost function. Initially, the trajectories are calculated in different homotopies based on a single path-cost criterion (the so-called *major criterion* that has a maximal weight within the cost function), which is temporarily set to 100% weight. Next, only K-best paths with regard to this major criterion (each belonging to a different homotopy) are used for further local optimization within its homotopy; at this second stage we apply a complete cost function with all criteria having a proper weight that is set by a user.

[1] According to the definition of homotopy, two curves belong to the same homotopy class, if it is possible to transform one curve into another by inseparable deformation.

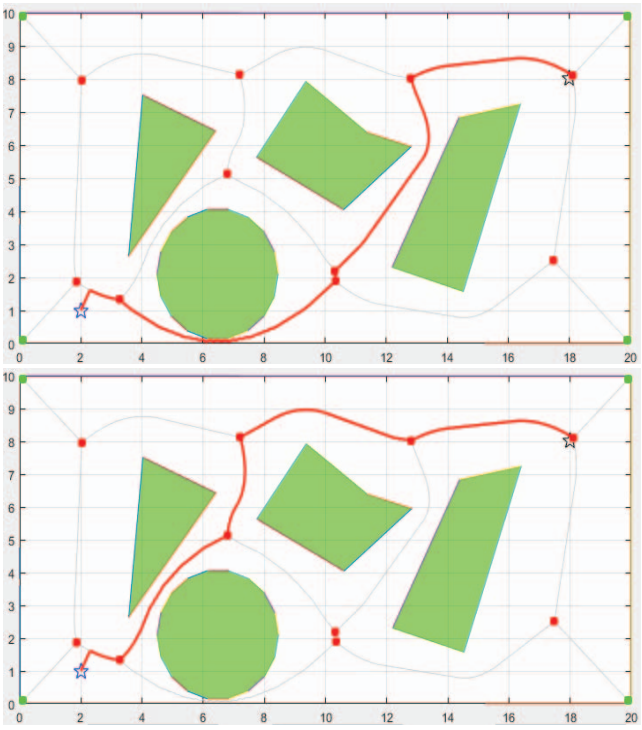


Figure 5. Voronoi diagram-based trajectories within two different homotopies. The path evaluation applies criteria of distance to obstacles (weight 80%) and the line-of-sight time (weight 20%)

Figure 5 demonstrates simulation results of a Voronoi-based path planning within two different homotopies. For this particular example, the cost function V is set to:

$$V = 0{,}8\ Dist + 0{,}2\ LoS \qquad (1)$$

where *Dist* is a criterion of distance to obstacles (with weight 80%) and *LoS* is a criterion of line-of-sight time (weight 20%). Thus, initially the algorithm builds Voronoi graph and selects two paths within the graph that are characterized with the lowest cost relatively to *Dist* criterion only. Next, these two paths are optimized locally with regard to the cost function V of the path (1).

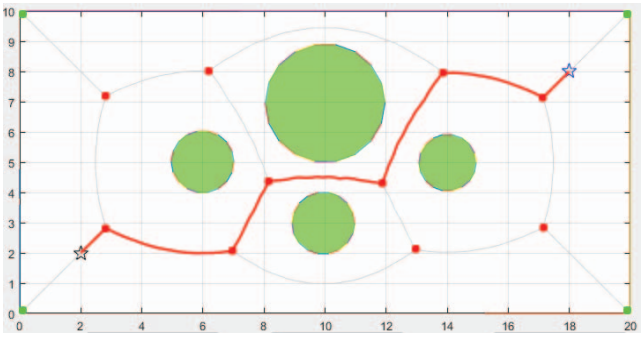


Figure 6. A map without danger points

When danger points (or on the opposite, communication routers) present within a map, the path planning process remains similar, but among the homotopies we select only those, which satisfy the criterion of maximal distance from danger points (or the closest distance to communication routers respectively). Next, for the selected homotopies we apply the same criteria as for the regular path planning that was described above. Figures 6 and 7 demonstrate danger point criterion influence on the initial path selection within the same map. Red crosses at Fig. 7 mark danger points to be avoided by the robot; thus the previously selected in Fig. 6 path gains significant cost and this homotopy loses its attractiveness for the robot.

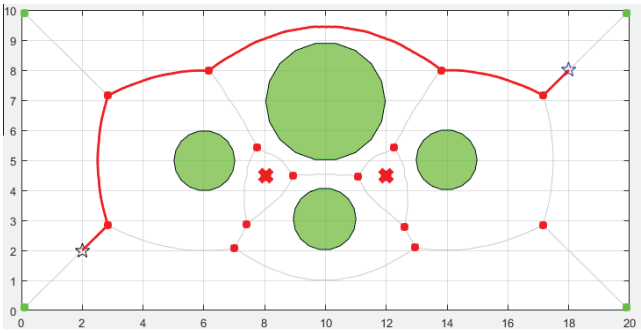


Figure 7. A map with danger points

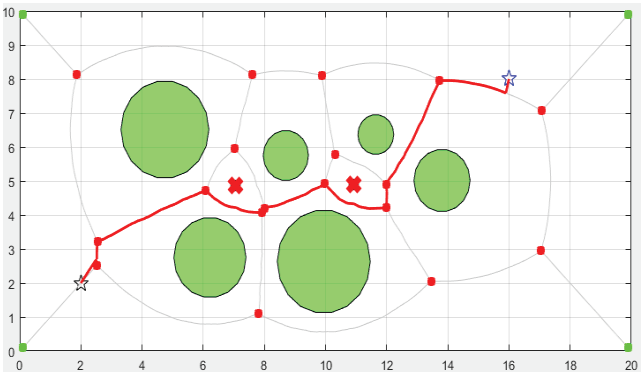


Figure 8. A map with communication routers

When, a map contains communication router points, the algorithm considers them as special obstacles, which must be kept at a certain distance. Figure 8 demonstrates an example where a path is forced to use the homotopy that contains two communication router points.

VI. Conclusions and Future Work

This paper presented a path planning algorithm, which allows to build a preliminary motion trajectory utilizing global information about environment, and then dynamically adjust the path in real-time by varying objective cost function weights of each criterion (parameter). A set of such key criteria for path optimization includes path length and curvature, safety (distance to obstacles), visibility time (line-of-sight time) for start/target points, distance to critical points and safe path length rate.

The proposed approach integrates approaches of global and local path planning by optimizing the computed trajectories for a mobile robot in any homotopy, which corresponds to optimization criteria. At the first stage (global

planning), the algorithm builds a preliminary path. Then at the second stage (local planning), the path is dynamically adjusted in real time by changing the weights of various optimization parameters within a total cost function. The algorithm is suitable for fast and robust trajectory tuning to a dynamically changing environment and is capable to provide efficient planning for mobile robots. The algorithm was implemented in MATLAB and we have verified our solution through a set of simulations.

We are planning to enhance the functionality of our algorithm by adding some parameters to the objective functions, e.g., taking into account surface irregularities with the relevant local path planning algorithms. In addition, we are going to analyze a correlation between path optimization parameters and adjust path planning algorithm for successful work in dynamic obstacle environments. In future, we intend to implement the C++ algorithm version and to integrate it into robotic operating system (ROS) for further experiments with optimal navigation of our crawler-type mobile robot "Engineer".

ACKNOWLEDGMENT

This research was partially supported by the Russian Foundation of Basic Research (RFBR) and Ministry of Science Technology & Space State of Israel (joint project ID 15-57-06010). The part of the work was performed according to the Russian Government Program of Competitive Growth of Kazan Federal University.

REFERENCES

[1] L. Bombini, A. Coati, J.S. Medina, D. Molinari and A. Signifredi, A General Purpose Approach for Global and Local Path Planning Combination, IEEE Int. Conf. on Intelligent Transportation Systems, IEEE Computer Society, USA, 2015, pp. 996-1001.

[2] M.A. Contreras-Cruz, V. Ayala-Ramirez and U. Hernandez-Belmonte, Mobile robot path planning using artificial bee colony and evolutionary programming, Applied Soft Computing, vol. 30, 2015, pp. 319-328. doi:10.1016/j.asoc.2015.01.067

[3] E. Magid, D. Keren, E. Rivlin and I. Yavneh, Spline-Based Robot Navigation," IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2006, pp.2296-2301.

[4] J. R. Andrews and N. Hogan, Inpedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator, Control of Manufacturing Processes and Robotic Systems, 1983.

[5] O. Khatib, Real-Time, Obstacle Avoidance for Manipulators and Mobile Robots, Int. J. of Robotics Research, Springer, New York, 1986, pp. 396-404.

[6] L. Tang, S. Dian and G. Gu and K. Zhou, A novel potential field method for obstacle avoidance and path planning of mobile robot, IEEE Int. Conf. on Computer Science and Information Technology, vol.9, 2010, pp.633-637.

[7] T. Simeon, J.-P. Laumond and C. Nissoux, Visibility based probabilistic roadmaps for motion planning, Advanced Robotics, vol. 14(6), 2000, pp. 477-493.

[8] Y. Liu and S. Arimoto, Path planning using a tangent graph for mobile robots among polygonal and curved obstacles, The Int. J. of Robotics Research, vol. 11(4) 1992, pp.376-382.

[9] S.M. LaValle, Rapidly-exploring random trees: A new tool for path planning, Iowa State University, 1998

[10] L.E. Kavraki, P. Svestka, J.-C. Latombe and M.H. Overmars, Probabilistic roadmaps for path planning in high-dimensional configuration spaces, IEEE Transactions on Robotics and Automation, vol.12(4), 1996, pp. 566-580.

[11] H. Choset and J. Burdick, Sensor Based Motion Planning: The Hierarchical Generalized Voronoi Graph, Advanced Robotics, 1997.

[12] B. Lau, C. Sprunk and W. Burgard, Improved Updating of Euclidean Distance Maps and Voronoi Diagrams, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2010, pp. 281-286.

[13] R. Gonzaleza, C. Mahuleaa and M. Kloetzerb, A Matlab-based Interactive Simulator for Teaching Mobile Robotics, IEEE Int. Conf. on Automation Science and Engineering, 2015, pp. 310-315.

[14] I. Kamon and E. Rivlin, A new range-sensor based globally convergent navigation algorithm for mobile robots, IEEE Int. Conf. on Robotics and Automation, vol.1, 1997, pp. 429-435. C

[15] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, Principles of Robot Motion: Theory, Algorithms, and Implementations, Cam-bridge, MA, 2005.

[16] E. Magid, and E. Rivlin, CautiousBug: a competitive algorithm for sensory-based robot navigation, IEEE/RSJ Int. Conf. on Intelligent Robots and Systems,vol.3, 2004, 2757-2762.

[17] I. Ulrich and J. Borenstein, VFH+: Reliable Obstacle Avoidance for Fast Mobile Robots, IEEE Int. Conf. on Robotics and Automation, vol. 2, 1998, pp.1572-1577.

[18] M. Likhachev and S. Koenig, D* lite, 8th National Conf. on Artificial intelligence, 2002, pp. 1282-1289.

[19] B. Lau, C. Sprunk and W. Burgard, Efficient Grid-Based Spatial Representations for Robot Navigation in Dynamic Environment, Robotics and Autonomous Systems, 61(10), 2013, pp. 1116-1130.

[20] M. Sokolov, R. Lavrenov, A. Gabdullin, I. Afanasyev, E. Magid, 3D modelling and simulation of a crawler robot in ROS/Gazebo, 4th Int. Conf. on Control, Mechatronics and Automation, ACM Publishing, Dec. 2016, pp. 61-65.