Two-dimensional Offline Path Planning for a Turtlebot

José Bernal, Paola Ardón, Èric Pairet
Univeristy of Girona

Abstract— **Path planning has been one of the most researched topics in the field of Robotics. Despite several algorithms are development every year, no absolute solution has been reached for this problem since all the approaches are subjected to a wide number of restrictions not only given by the scenario but also by the capabilities of the robots. In this paper, several remarkable algorithms within the state-of-the-art are evaluated using a designed benchmarking to recognise their strengths and weaknesses. The dominating approach is then embedded into a C-FOREST framework to test it in a high-level controller in order to obtain a solution of the proposed simulated and real-life environments. Furthermore, and for the sake of security of the device, a safety box around the robot was considered. The results showed that the RRT* outperforms in the set of measures defined for the assessment in comparison to the other selected algorithms. Also, a better performance of the same algorithm was observed when considering the C-FOREST technique. Finally, it was observed that the proposed approach was able to find an acceptable solution to simulated and real-life environments in real-time.**

I. INTRODUCTION

Through the years, the area of robotics has been providing incredible technological developments. So much so that we can talk nowadays not only about robots performing housework [1], [2], but also critical tasks, such as helping in disasters [3], [4] and assisting surgeons in delicate procedures [5], [6]. Regardless the task the robot has to accomplish, all of them rely on one of the most important topics in the field: motion planning [7]. Planning the motion of a robot consists in finding the best series of movements driving the robot from a starting state onto the given target while taking into account the constraints of the scenario in which it has been placed as well as its own.

The evolution has made humans very skilful in planning the motion since they had to detect, classify and recognise different environmental features under a variety of conditions and distortions to be able to find food and also to avoid dangerous predators. Then, the question rising in Robotics and Computer Science is: how can this knowledge be placed into a computer such that it is able to handle the environment in the same way as we do? In other words, the challenge itself is to build autonomous robots.

Under this perspective, a robot should be able to sense the environment, update its internal map [8] to recognise the features in future moments and, at the same time, to avoid dangerous situations, such as collisions and unsafe conditions due to the economical loss that damages on the device may represent. However, since all these tasks are limited by the perceptual capabilities of the sensors of the robot and the representation it has of the environment, up to date, there is no truly elegant solution for the navigation problem. The best approach, so far, is to combine relative and absolute position measurements to achieve the best localisation. The first kind of measurements include, among others, the odometry and the inertial navigation; while the absolute position measurements include magnetic compasses, sensors and matching models.

Throughout this paper we propose an off-line high-level controller that handles the navigation of the mobile robot Turtlebot [9] in a two-dimensional environment with obstacles. The document is structured as follows: in Section II a review of the state-of-the-art about some path planning algorithms is presented. According to this analysis, the proposed high-level controller is introduced in Section III, while an overview of the implementation is given in Section IV. Then, the evaluation environments are defined in Section V and the implementation is tested in Section VI. Finally, some final remarks are stated in Section VII.

II. STATE-OF-THE-ART

Many path planning algorithms have been developed during the last decades. The most well-known approaches are based on solving low dimensional problems using grid-based techniques [10] which consists in overlaying a grid on top of the configuration space to compute the shape and the connectivity of the free space.

Some researchers have proposed to address the problem as an optimisation of a potential function in which the robot takes into account the distance to the goal and the proximity with respect to the obstacle. But these approaches are computationally expensive when the problem involves several degrees of freedom and constraints. In fact, exact motion planning for high dimensional systems remains computationally intractable [10].

One way to get along with this situation is to reduce the complexity of the problem by smartly choosing few samples from the free configuration space without losing completeness in the process. These methods, called sampling-based algorithms, solve the system quite quickly and avoid the problem of local minima. For these reasons, they are currently considered state-of-the-art for path planning in high-dimensional spaces.

Sampling-based algorithms are used either for online or offline planning in combination with other methods. These last ones only consider the geometric and kinematic constraints of the environment, assuming that every path can be a possible trajectory [11]. Two groups can be identified within the literature regarding the number of queries they

are oriented to solve: single-query or multi-query planners. On one hand, the approach of the single-query planners is to build one or two trees and connect the different nodes as the process goes. Algorithms such as the well-known RRT* [12] –an optimal version of RRT which converges to an optimal path as a function of time–, TRRT [13] –the fusion of RRT and a stochastic optimising mechanism to bias the process towards a low-cost region in the configuration space–, and LBTRRT –a near-optimality version of RRT guaranteeing to converge to a solution within a factor of the optimal solution– fall into this category. On the other hand, multi-query planners are specialised in building a map such that it can be used to perform several queries. Usually, extensions of the PRM algorithm belong to this group, such as LazyPRM* [14] –in which vertices and edges are checked only if they are part of the candidate solution–, PRM* [12] –which considers a gradual increase of connection attempts as the roadmap grows–, SPARS [15] –an asymptotic near-optimality algorithm which selects the best candidate in a search window around the current node instead of taking the closest neighbour and prunes the graph in each iteration–, and SPARS2 –a variant of SPARS using a slightly different approach to compute the shortest path.

Path planning is not an isolated field from the general problems faced in computer science. Because of that, the literature proposes the integration of many generic algorithms to this field, such as heuristics algorithms [16], multi-threading approaches [17] or machine learning techniques [18]. Depending on the requirements to be covered, a combination of these approaches might be considered.

III. PROPOSED APPROACH

A well-known technique exploiting the power of concurrency available in nowadays' computers is presented in the C-FOREST framework. This technique considers feedback between different path planning algorithms which run at the same time each of them in a different CPU. The feedback takes place every time a path from the starting position to the goal is found for one of the methods. The approach consists in sending the solution to the other nodes via message passing so that it can be grafted. This is beneficial since all the trees expand into a valuable area known by at least one tree, and all the trees focus on their search by avoiding regions of the configuration space that cannot produce a globally better solution. Note that this is a greedy strategy, since it does not account for the fact that future tree remodelling may decrease.

The proposed approach for this work is to implement the different algorithms introduced in Section II and build a scientific comparison on their performance. In addition, the most suitable path planning algorithm is then embedded into a C-FOREST framework in order to enhance its performance for the two-dimensional offline problem on a Turtlebot.

IV. IMPLEMENTATION

The approach proposed in Section III has been fully implemented in this section. First, the requirements derived from the proposal are presented in Section IV-A. Then, in Section IV-B, the strategy taken to perform a reliable benchmarking of the different considered path planning algorithms is explained. Finally, the architecture designed to test the proposal in a two-dimensional path planning problem is detailed in Section IV-C.

A. Implementation requirements

For the implementation of the proposed approach several tools have been integrated, such as: Robot Operating System (ROS) [19] –which is an OS specifically designed for the development of robotic applications–, Turtlebot [9] in Gazebo [20] –which allows us to simulate a robot in a given environment–, the Open Motion Planning Library [11] (OMPL) –a package containing many state-of-the-art sampling-based motion planning algorithms in C++– and RViz [21] and Open source Computer Vision [22] (OpenCV) –for visualization purposes. The first two elements are required by the project statement while the remaining ones are considered for improving the robustness and user-friendly aspects of the application.

Additional functionalities have been programmed in C++ and Python not only to integrate all these tools under the same framework, but also to provide the path planning algorithm with a collision checking strategy, which is not initially given by the OMPL.

B. Algorithms benchmarking

Several evaluation processes can be found within the literature, but none of them have been universally adopted in the path planning area. In this particular case, the OMPL benchmarking framework, introduced in Fig. 1, has been considered. This assessment process allows us to easily solve a motion planning problem repeatedly with different approaches, e.g. planners, samplers, or parametrisation of the algorithms. By taking advantage of this toolbox, an extensive comparison of the introduced PRM*, Lazy PRM*, SPARS, SPARS2, RRT*, T-RRT and LBTRRT algorithms is done.

The benchmarking is fourfold. First, an instance of the OMPL benchmarking class is initialised, in which a path planning problem has to be set up. Second, the algorithms, as well as the evaluation measures, are indicated to the framework. At this point, some constraints can be imposed

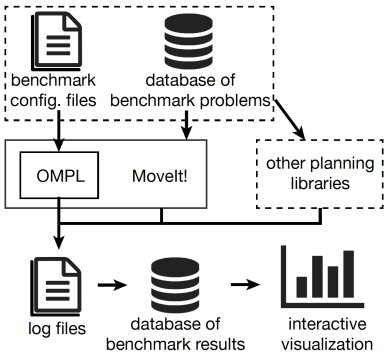

Fig. 1: OMPL benchmarking framework.

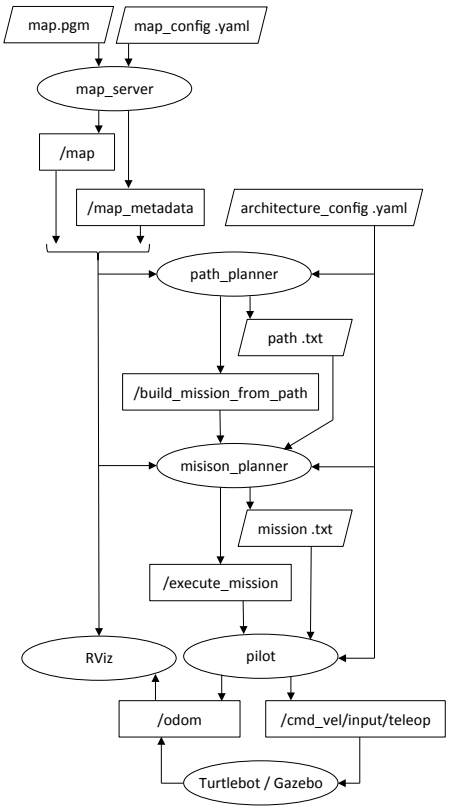

Fig. 2: ROS architecture designed to perform offline path planning in a Turtlebot.

to the path planner algorithm, such as the number of runs, the maximum solving time and the maximum memory usage. Third, the process is carried out and the resulting information, which is contained in a *.log* file, is extracted to a SQLite database through the *ompl_benchmark_statistics.py* Python code provided by the same OMPL. Finally, the parsed data in the database can be plotted using either the same Python code or via the Planner Arena [23], which allows the user to navigate through a wide range of plots.

C. ROS architecture

A ROS architecture has been designed and implemented to fulfil all the previously commented requirements. As it can be observed in Fig. 2, such framework is based on four ROS nodes: the map server, the path planner, the mission planner and the pilot, which are described below. Additionally, it uses the RViz visualiser and the robot, either the real one or the one in the Gazebo simulator.

- The **map server** reads the *.pgm* image which defines the environment. Considering the parametrisation in the *.yaml* configuration file, i.e. origin of the map, its orientation and desired resolution, it builds an occupancy grid map. This sampled map is published to the ROS map server to make it available for the other nodes in the ROS architecture.
- The **path planner** wraps the map in the ROS map server and all the configuration variables specified by the user to set up the path planning problem in the OMPL framework. If the planner finds a solution, the node stores its states in a text file, which at this level of the architecture is defined by the number of cells with respect to the origin frame of the occupancy grid map.
- The **mission planner** transforms the resulting path to the robot frame in metric units. This transformation also considers directing the heading of the robot, at each state, towards the next one. Thus, each intermediate goal is described by a triplet (x, y, θ). The computed mission is displayed and executed only under the user's agreement.
- The **pilot** reads the setpoints proposed by the mission planner node to drive the Turtlebot through the environment. The navigation system of the Turtlebot relies on the odometry while its control system is based on one proportional controller for each of the two Degrees Of Freedom (DOF) of the vehicle. Both controllers are synchronised in a way such that the heading always prevails to the linear motion. Under this perspective, the head of the robot is fixed towards the goal before moving straight to it.

This architecture is executed through a unique launch file according to the parametrisation given in the configuration document. This strategy provides dynamism to the different nodes of the framework. The following parameters can be modified without compiling the full architecture:

- **Simulation:** is a boolean flag indicating whether the architecture will work with the simulated Turtlebot in Gazebo or the real robot.
- **Start and goal positions:** which are given in pixel coordinates with respect to the top-left corner of the occupancy grid map.
- **Time:** is the maximum time allowed to look for a solution of the path planning query.
- **Safety box:** which is the minimum distance that the robot should keep with respect to any obstacle.
- **Linear and angular controllers:** which are all the parameters in charge of the linear and angular movement related to the controller.

The overall architecture has been designed to be consistent; it will only start up if the nodes of the framework and its configuration parameters are properly initialised. Otherwise, an error message will be displayed on the terminal to help the user identify the issue.

V. EVALUATION ELEMENTS

In this section are presented the elements used to evaluate each part of the proposal. The set up of the theoretical benchmarking is detailed in Section V-A while the evaluation of the architecture, both in simulated and real scenarios, is respectively introduced in Section V-B and Section V-C.

A. Algorithms benchmarking

To decide which algorithm was the most suitable to solve the two-dimensional path planning problem, the following aspects were considered into the benchmarking:

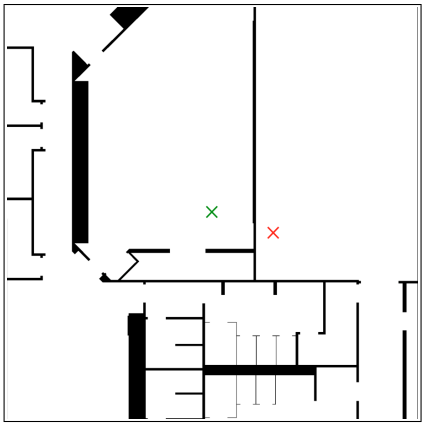

Fig. 3: Path planning problem for benchmarking the planners and simulating trials.

- **Cost:** the cost of a path can be determined through different metrics. Our proposal treats the cost as the length of the path. In this way, the shorter the path, the lower the cost.
- **Memory:** depending on the device in which an algorithm is run, the amount of memory can represent an important constraint.
- **Segments:** which are the edges connecting the different states in the final path. The segments are directly related to the smoothness of the path, i.e. the higher the number of segments, the smoother the path.
- **Smoothness:** since the system relies on odometry measures, smoother paths are preferred to avoid problems when reading the data from the robot actuators.

A path planning problem has to be set up in order to use the OMPL benchmarking framework. In this case, the considered environment is illustrated in Fig. 3 where the start position and the goal are represented with a red and green cross, respectively. The solution for this scenario was computed ten times per algorithm in order to reduce the randomness associated with the sampling-based path planning techniques. Moreover, since any asymptotically optimal and nearly-optimal algorithm would take as much time as allowed, the maximum time to solve the query was fixed to 20 seconds and the maximum usage of memory was limited to 300 MB.

B. ROS architecture in a simulated scenario

Many experiments were performed in simulation to validate the different nodes of the framework. At the end, a concluding trial was designed to validate the functionality of the full architecture. The objective of this test consisted in moving the Turtlebot through the Willow Garage environment of the Gazebo simulator introduced in Fig. 3.

The occupancy grid map of the considered environment was extracted using the collision map creator plugin [24] of Gazebo. This map corresponds to a binary image of the region obtained by defining the area of interest and its resolution. In this particular case, a square with borders of 20 metres length centred at the origin of the Gazebo frame was contemplated. With this parametrisation and using a resolution of ten centimetres, the dimension of the resulting map was of 2000×2000 cells. Due to the narrow corridors, a safety box of 0.6 metres was to be defined.

C. ROS architecture in a real scenario

For testing the ROS architecture under real conditions, a challenging experiment in the P-II building of the Polytechnic school of the University of Girona was designed. The purpose of this scenario consisted in autonomously driving the robot from the red cross to the green one as presented in Fig. 4. The environment was converted to an occupancy grid map using a resolution of ten centimetres resulting in a map of 5430×8725 cells.

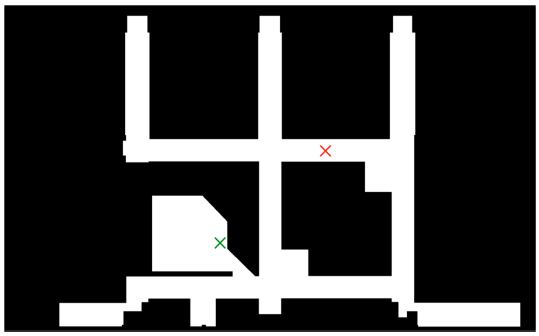

Fig. 4: Path planning problem for the real experiments with the Turtlebot.

To achieve the goal position located in the main hall of the building, the robot had to go through a narrow corridor of approximately two metres. Therefore, when performing the real experiments, a safety box of 1.5 metres was considered. It can be seen that the experiment was difficult since not only the computed path had to be extremely accurate, but also the pre-processed occupancy map.

VI. RESULTS

In this section, the results of the proposed evaluation framework are presented. The outcomes of the benchmarking, the simulated and the real experiments are presented in Section VI-A, VI-B, VI-C, respectively.

A. Algorithms benchmarking

The obtained results regarding length, smoothness, number of segments and memory consumption are shown in Fig. 5. It is important to have in mind that the results of SPARS and SPARS2 are gathered from less number of trials since they were not able to find a solution within 20 seconds in the 60% and 10% of the cases, respectively.

In Fig. 5 (a), the results in terms of length of the final path are evidenced. It can be seen that the shortest path with the smallest variability is found by RRT*.

In Fig. 5 (b), the smoothness of the solutions are shown. It can be observed that multi-query planners are able to find smoother solutions compared to the single-query planners.

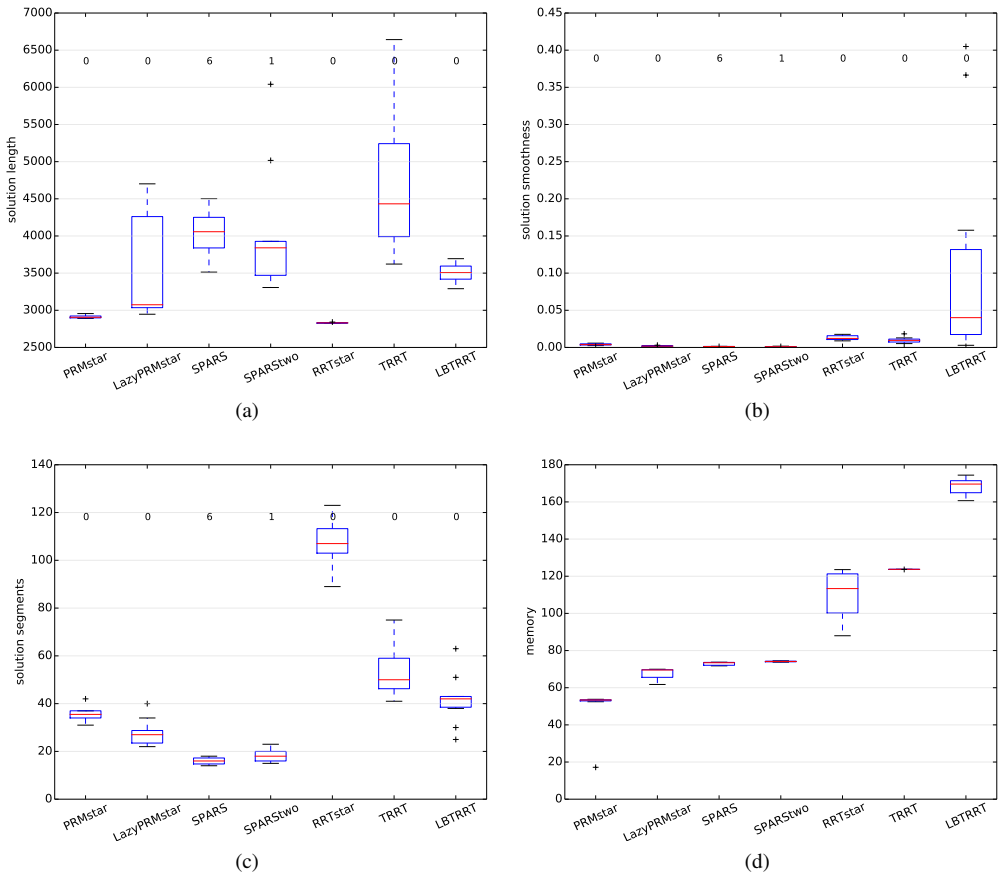

Fig. 5: Benchmarking of optimal and nearly-optimal sampling-based path planning algorithms in the state-of-the-art: (a) length, (b) smoothness, (c) segments, (d) memory.

Unlike in the previous charts, Fig. 5 (c) and (d), the two groups of techniques are distinguishable since the PRM-based approaches achieve better results than the others with the lower variability in terms of Inter-Quartile Range. Furthermore, their memory consumption is around the half of the single-query algorithms.

In general, multi-query algorithms provide better results in terms of the considered evaluation elements. However, the shortest path is found by RRT* since it is able to explore the map more than the other algorithms within the same amount of time.

The RRT* demonstrated to be the most suitable planner since it provides the most optimal solution with a reasonable smoothness and usage of memory. Therefore, we decided to use it as the base of the C-FOREST algorithm to get a planner with a better performance. Tab. I quantitatively summarises the advantages and drawbacks of the proposal.

The C-FOREST RRT*-based not only reduces the cost of the path, but also the number of segments. Moreover, the smoothness of the path is dramatically enhanced, which is completely beneficial to avoid problems with odometry-based navigation systems. Even though, more memory is

	RRT*	**C-FOREST**
Cost	2831	**2825**
Smoothness	0.0122	**0.0043**
Segments	106	**58**
Memory [MB]	**93**	145

TABLE I: Quantitative comparison of the C-FOREST RRT*-based with the conventional RRT* algorithm.

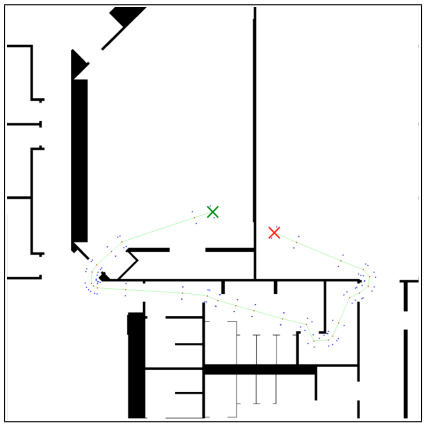

Fig. 6: Proposed path through the simulated environment.

required, such amount can be nowadays afforded by any robot.

B. ROS architecture in a simulated scenario

The path found when dealing with the environment of the Willow Garage of the Gazebo simulator is shown in Fig. 6. As it can be seen, the path preserves the safety distance to all the obstacles. After the mission was accepted under the user's agreement, the Turtlebot was able to reach the goal following the computed path.

C. ROS architecture in a real scenario

The proposed solution of the implemented high-level controller when dealing with the P-II building is presented in Fig. 7. As illustrated in the plot, a successful path was found from the start to the goal position, preserving the configured safety distance. After accepting such mission, the robot performed the whole trajectory in the same way it was planned in approximately five minutes.

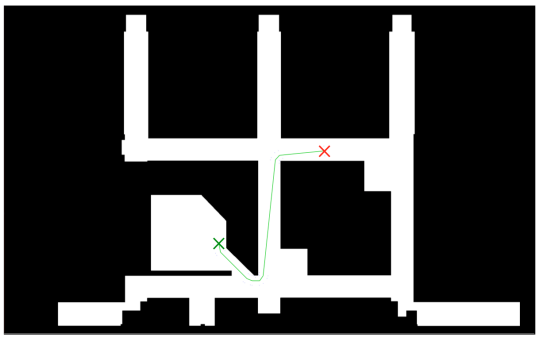

Fig. 7: Proposed path through the real environment

VII. FINAL REMARKS

In this paper, an efficient algorithm for solving a two-dimensional offline path planning problem has been proposed, implemented and integrated using an *ad hoc* ROS architecture. This approach has been tested under simulated and real scenarios.

Among the considered algorithms, RRT* is the most convenient planner to solve the challenges presented in the extensive benchmarking. As a consequence, it was embedded into a C-FOREST framework to enhance its performance. The obtained results not only states its optimality, but also it provides one of the shortest and smoothest paths that can be achieved with any sample-based algorithms in the state-of-the-art.

Some work has been left for further studies. For instance, it could be interesting to investigate if different path planning algorithms can be used in a similar framework as the C-FOREST to take advantage of their corresponding strengths. Moreover, the control system of the implemented framework could be improved by considering a PID controller, with which a smoother path could be achieved.

REFERENCES

[1] D. Pangercic, B. Pitzer, M. Tenorth, and M. Beetz. Semantic object maps for robotic housework - representation, acquisition and use. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4644–4651, Oct 2012.

[2] D. Nyga and M. Beetz. Everything robots always wanted to know about housework (but were afraid to ask). In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 243–250, Oct 2012.

[3] H. Aoyama, A. Himoto, O. Fuchiwaki, D. Misaki, and T. Sumrall. Micro hopping robot with ir sensor for disaster survivor detection. In *IEEE International Safety, Security and Rescue Rototics, Workshop, 2005.*, pages 189–194, June 2005.

[4] H. Sugiyama, T. Tsujioka, and M. Murata. Collaborative movement of rescue robots for reliable and effective networking in disaster area. In *2005 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 7 pp.–, 2005.

[5] K. Yoshimitsu, K. Masamune, H. Iseki, Y. Fukui, D. Hashimoto, and F. Miyawaki. Development of scrub nurse robot systems for endoscopic and laparoscopic surgery. In *Micro-NanoMechatronics and Human Science, 2010 International Symposium on*, pages 83–88, Nov 2010.

[6] M. Wang, L. Sun, Z. Du, and Z. Jia. The design and implementation of virtual system for the robot-assisted setting-bone surgery. In *2006 Bio Micro and Nanosystems Conference*, pages 53–57, Jan 2006.

[7] H.M. Choset. *Principles of Robot Motion: Theory, Algorithms, and Implementation*. A Bradford book. Prentice Hall of India, 2005.

[8] Oliver Henlich. "Where am I going and how do I get there", an Overview of Local/Personal Robot Navigation. `https://www.doc.ic.ac.uk/~nd/surprise_97/journal/vol1/oh/`, 1997. Online accessed 7-May-2016.

[9] TurtleBot. `http://wiki.ros.org/Robots/TurtleBot`. Online accessed 29-April-2016.

[10] Johann Borenstein, Hobart R. Everett, Liqiang Feng, and David K. Wehe. Mobile robot positioning: Sensors and techniques. *J. Field Robotics*, 14(4):231–249, 1997.

[11] The Open Motion Planning Library. `http://ompl.kavrakilab.org/`. Online accessed 7-May-2016.

[12] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011.

[13] Didier Devaurs, Thierry Siméon, and Juan Cortés. Enhancing the Transition-based RRT to deal with complex cost spaces. In *Proc. IEEE ICRA '13*, pages pp. 4105–4110, Karlsruhe, Germany, May 2013.

[14] Robert Bohlin and Lydia E. Kavraki. Path planning using lazy prm. In *ICRA*, pages 521–528. IEEE, 2000.

[15] Andrew Dobson, Athanasios Krontiris, and Kostas E. Bekris. *Algorithmic Foundations of Robotics X: Proceedings of the Tenth Workshop on the Algorithmic Foundations of Robotics*, chapter Sparse Roadmap Spanners, pages 279–296. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[16] Dave Ferguson, Maxim Likhachev, and Anthony Stentz. A guide to heuristic-based path planning. In *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling*, 2005.

[17] Michael W. Otte and Nikolaus Correll. C-FOREST: parallel shortest path planning with superlinear speedup. *IEEE Trans. Robotics*, 29(3):798–806, 2013.

[18] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *J. Intell. Robotics Syst.*, 57(1-4):65–100, January 2010.

[19] Robot Operating System. `http://wiki.ros.org/`. Online accessed 4-May-2016.

[20] Gazebo simulator. `http://gazebosim.org/`. Online accessed 6-May-2016.

[21] RViz visualiser. `http://wiki.ros.org/rviz`. Online accessed 9-May-2016.

[22] Open Source Computer Vision. `http://opencv.org/`. Online accessed 27-April-2016.

[23] Planner Arena. `http://plannerarena.org/`. Online accessed 2-May-2016.

[24] Collision map creator Gazebo plugin. `http://gazebosim.org/tutorials?tut=custom_messages`. Online accessed 20-April-2016.