FORMI: A Fast Holonomic Path Planning and Obstacle Representation Method Based on Interval Analysis

FORMI: A Fast Holonomic Path Planning and Obstacle Representation Method Based on Interval Analysis

Pekka Mäenpää, Mohammad M. Aref, and Jouni Mattila
Automation Technologies and Mechanical Engineering Unit, Tampere University
Tampere, Finland 33101.
pekka.maenpaa@tuni.fi, m.aref@ieee.org, jouni.mattila@tuni.fi.

***Abstract*— This paper presents a path planning approach for mobile robots in 2D spaces. The algorithm uses a quadtree decomposition where the discretization precision is improved until a path to the goal is found if one exists. The algorithm uses interval analysis-based methods to categorize the quadtree decomposition to occupied, free and partly occupied cells. The proposed algorithm is compared against other concurrent path planning algorithms, A* on an ordinary quadtree, A* for shortest path on a binary occupancy grid, and a Dijkstra's algorithm for lowest collision probability in a continuous-valued occupancy grid, in five different scenarios.**

Compared to the other methods, the main advantage of our method is achieving a compromise between driving distance, safety, and computation time. The proposed algorithm was found to require significantly fewer collision checks in all scenarios while providing sub-optimum results, based on the obstacle distance and path length criteria. The algorithm is suitable for further extension to include non-euclidean measures and for higher dimensions of configuration spaces. The proposed algorithm will be publicly available on GitHub repository.

I. Introduction

Superior path planning algorithms can improve the safety, energy efficiency and effective speed of mobile robots.

In this context, the basic problem of path planning is to find a collision-free path from the initial configuration to the goal configuration. The planning is further complicated by the need to minimize path cost criteria, which include path length, smoothness and safety among others. This is why many concurrent path planning methods use a two-pronged approach, where the path planner is only in charge of creating a geometric, collision-free path which is then subject to some path-smoothing algorithm [1] [2]. The main flaw of this approach is that the initial path found by the path planner might not be near the global optimum path or even in the same homotopy class. This is a problem for many path smoothing algorithms, which only search for the optimum near the initial value instead of the global optimum.

Concurrent path planning methods usually reduce the world obstacles and restrictions to simple, geometric descriptions, like cell decompositions [3] or Voronoi diagrams, which can be searched by graph search algorithms, or only sample some sparse routes in the configuration space, like probabilistic roadmaps [4] or rapidly-exploring random trees [5]. The search space can also be represented by a potential field, like in [6], and the path can be found by moving along the gradient towards the lowest potential.

These geometric descriptions usually have some maximum resolution, and the others require the use of obstacle envelopes which usually exaggerate the size of the obstacles. The algorithms form a graph from the obstacle- or free space descriptions, and the resulting graphs can be searched by complete graph search algorithms, which either find the path or correctly report that it does not exist. These geometric descriptions can produce very large graphs, which are slow to search, but more importantly, creating high-resolution graphs from the available data can be even more time-consuming. In addition, large parts of these graphs are usually not searched in a single path planning query.

The sampling-based methods cover large parts of the search space quickly and these methods can be biased to prioritize parts of the search space that are likely to be near the final path. As a drawback, they are only probabilistically complete. The probabilistically complete algorithms can search the entire configuration space, but practically, these algorithms must use some iteration or time limit, and in this case they are no longer complete. In addition, if the path can not be found, the algorithm will always run until this iteration or time limit. The iteration or time limit defines the search resolution, but it is much more vague than a resolution defined for the combinatorial algorithms. The sampling-based methods can use uniform sampling to get a clearer search resolution, but this will consume unnecessarily much time in large free spaces.

Both of these types of algorithms can be used to find safe movement corridors instead of only trajectories. For grid-based algorithms, the grid square width can be used as the corridor width, and for all of the algorithms the obstacles can be simply padded by the desired corridor width. However, to find wider corridors without sacrificing planning resolution, many path planners post-process the initial path so that all path segments are inside convex, overlapping safe regions. This has been done for UAVs in [7] and [8].

In the case of mobile robots, the available data is the fused sensor data, which can be integrated to an occupancy grid [9] or to continuous obstacle probabilities with means and covariance matrices using extended Kalman filters, such as in FastSLAM [10]. In mobile robot path planning, the safety of the paths is even more important issue than the length of the paths, and despite the advances in computing performance, the computational cost of the algorithms remains a significant issue [11]. The importance of the planning time is not only

evidenced by the need for algorithms focusing on planning speed, like the QMP [12], but also by the need for faster collision checking for mobile robots by new methods, such as the one by B. C. Heinrich et al.[13]. This issue is most apparent in situations such as planning short-distance routes for UAVs [14] and for two-dimensional cases where the path must be replanned based on new sensor data [15].

The proposed algorithm, *A Fast Holonomic Path Planning and Obstacle Representation Method Based on Interval Analysis* (**FORMI**), improves the scalability of the grid-based algorithms by using a quadtree representation of the configuration space with iteratively improving precision. The algorithm was compared to searching both binary and continuous-value occupancy grids and to an ordinary quadtree representation, and it was found to deliver comparable results in terms of path length and obstacle distance, while requiring much fewer collision checks.

The next section describes the algorithm in more detail, while section three describes the path planning results and compares it to other contemporary methods and section four draws conclusions from these results. The Matlab code of proposed algorithm is available on a private link on Google Drive [1], and will be publicly available on GitHub upon the conference day.

II. The Algorithm

The algorithm is based on the approach presented by Luc Jaulin [16]. In this algorithm, the configuration space C is approximated by a quadtree, and the quadtree resolution is improved until the path can be found in it. Because of this, the final quadtree can have few elements, even if the maximum resolution is high. C is here divided to C_{free}, containing all of the configurations which do not lead to a collision, and C_{obst}, containing the rest of the configuration space.

[1] https://goo.gl/nb6L4B

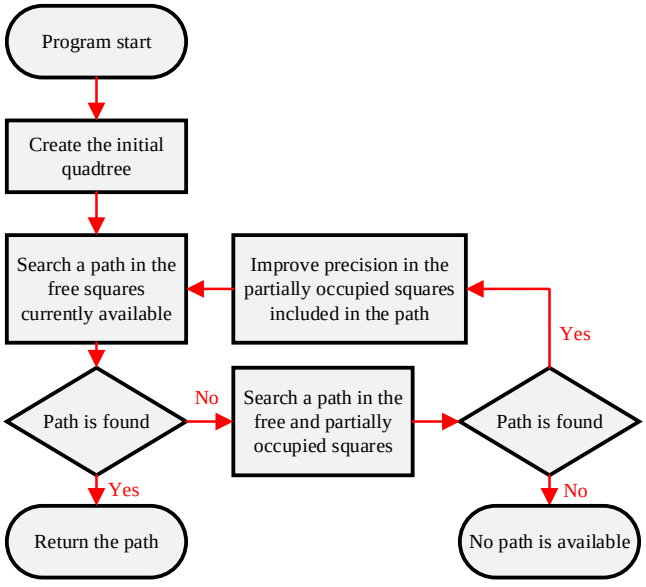

Fig. 1. Decision making flowchart for path planning and paving parsing in a multi-resolution representation of configuration space.

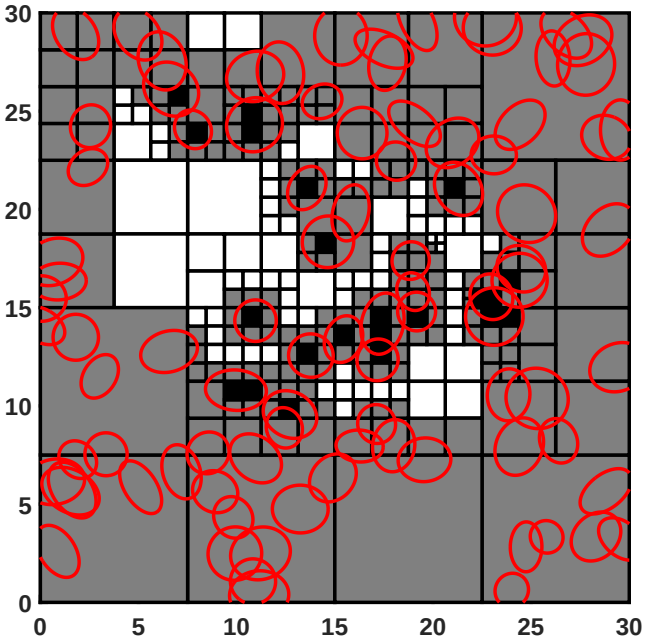

Fig. 2. The quadtree decomposition compared to the true obstacles represented by red ellipses (C_{obs}). Corresponding paving of square cells are shown by their color: white (P_{free}), grey (P_{grey}), and black (P_{obst}). The decomposition is applied to the randomly generated map subject of experiment (d) shown in Fig. 6

To properly describe the algorithm, it is necessary to define a paving. A paving of a rectangle P is a non-overlapping set of squares whose union is equal to P. A subpaving of P is a non-overlapping set of squares which is entirely contained in P. This is why the entire configuration space C can be padded to a same-dimensional cube represented by a paving P_C consisting of three subpavings, one of which is P_{free}, entirely contained within C_{free}, one which is P_{obst}, entirely contained within C_{obst} and one which is P_{grey}, partly contained within both. With fast inclusion tests it is possible to determine correctly the set where a quadtree square belongs to.

Because all these pavings belong to a quadtree, we can define a graph where the vertices correspond to the squares in the quadtree and the edges are between squares facing each other. Based on this, if we find a path in the graph formed from P_{free}, we can find a path in C_{free} simply by following the quadtree square centers.

A. Obstacle definition

The obstacles are described as ellipses, because a SLAM output consisting of obstacle locations and covariance matrices can be easily converted to uncertainty ellipses according to some threshold. In addition, these obstacles can not be split to a limited number of squares, making the comparison between quadtree-based and other methods more fair and representative of real-world configuration spaces. In addition, the inclusion tests between ellipses and rectangles are

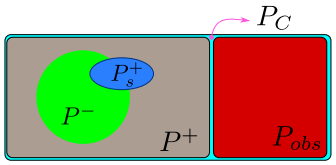

Fig. 3. Venn diagrams for logical relations between different sets including all cell categories.

computationally light even when done exactly [17].

B. Paving generation

The algorithm works by first choosing a very coarse resolution and then iteratively improving the resolution in the key parts of the quadtree. The algorithm first attempts to find a path in the graph formed from P_{free}, but if a path is not found, it is searched from $P_{free} \cup P_{grey}$, the free and partially occupied squares. If the latter path is not found, the search terminates as a path to the goal doesn't exist. If it is found, the partially occupied squares which belong to the latter path are dissected, and the completely free path is searched again. This is repeated until a free path is found or the path is confirmed to not exist. The program flow is described in Fig. 1.

The A* used in the graph accounts both for the Euclidean distance between the vertices and the quadtree square size. The tree depth penalty cost can be used to adjust the planner behaviour, so that lower values lead to straighter paths and more aggressive searches, while higher values lead to more conservative paths with greater distance to obstacles. The tree depth penalty increases when going deeper to the tree, as in [18], but in addition to that the depth penalty has a minimum value, so that the depth penalty of all cells is equal at low enough levels. However, with any values the path search will find a path if one exists and return failure otherwise. Fig. 2 describes the final quadtree, with free cells on white, occupied cells on black and partly occupied cells on grey. The ellipses are the used obstacles, while the dashed line is the planned route, starting from top left and ending to bottom right. The points along the quadtree path are the center points of the crossed square edges.Using the partly occupied cells also allows the use of less accurate collision checks at larger cells of the quadtree, because uncertain results can be simply marked as partly occupied, and re-examined if necessary.

C. Completeness of modified quadtree

We introduce a resolution limit, so that the partly occupied squares at maximum resolution are marked as occupied. This resolution limit is based on the robot size and its perception accuracy. Given this limit, the algorithm is provable to be resolution complete. For the sake of brevity, let the sets $P_{grey} \cup P_{free}$ and P_{free} to be called as P^+ and P^-, respectively. A paving P_{compl} represents the free part of the configuration space C_{free} with the maximum resolution defined by the resolution limit. The paving P_{compl} can be generated by repeatedly taking a cell from P_{grey}, subdividing

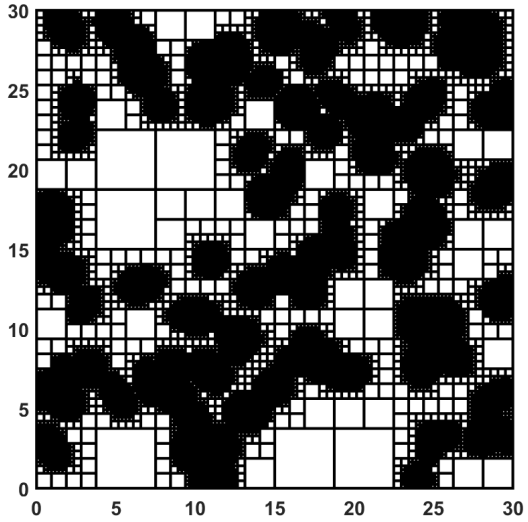

Fig. 4. Traditional quadtree in the same randomized map in Fig. 2 and Fig. 5.

it, and classifying each new cell to P^-, P_{grey}, or P_{obst} until P_{grey} is empty. At this point $P^- = P_{compl} = P^+$.

The goal of the algorithm is to find a path which belongs to P_{compl}, or report that such a path does not exist. By using a complete graph search algorithm, (e.g. A*) , we can search for the path P_s^- in P^-. If this path exists, it belongs to P_{compl}, the search has completed and returns this path. If it doesn't we can search for the path P_s^+ in P^+. If this path does not exist, a path can not exist in the smaller set P_{compl}, and the algorithm correctly returns that the path does not exist. However, if the path is found in P^+ but not in P^-, nothing is proven about the existence of the path, corresponding to the situation in Fig. 3.

In this case, the group $P_s^+ \cup P_{grey}$ must not be empty, because if it was, the path P_s^+ would be entirely contained in P^- and a path could be found in P^-. Now the cells in the group $P_s^+ \cup P_{grey}$ are subdivided and reclassified. After this, we repeat the searches. Because of the resolution limit, the pavings P^- and P^+ can converge to P_{compl} in a finite number of steps, and we can find the path or correctly determine that it does not exist.

III. Results and Performance Comparison

This section contains a brief description of the competitor algorithms and a comparison between these and our approach.

A. The competitor algorithms

FORMI was compared to three path planning methods, two in an ordinary grid and one in a quadtree. The algorithms used A* and Dijkstra's algorithm for the graph search. For the grid-based methods, the number of required collision checks was the number of different explored grid squares. For the quadtrees, the number of collision checks was the number of checked leaf nodes. The first approach used A* on a binary occupancy grid. This approach used the same

TABLE I

COMPARISON OF TEST RESULTS FOR EACH GENERATED MAP

Map	Algorithm	Avg. coll. prob.	Path length	Max. coll. prob.	Number of collision checks
Office Corridor	Dijkstra	2.83×10^{-8} †	25.188 ‡	1.69×10^{-8} †	4513
	Binary A*	4.81×10^{-7}	22.477 †	7.98×10^{-8}	972 ‡
	Ordinary quadtree	6.46×10^{-8} ‡	27.087	2.50×10^{-8} ‡	10065
	FORMI	6.46×10^{-8} ‡	27.087	2.50×10^{-8} ‡	737 †
Maze	Dijkstra	2.15×10^{-17} †	64.510	2.09×10^{-17} †	20177
	Binary A*	5.76×10^{-7}	50.761 †	7.58×10^{-8}	8329
	Ordinary quadtree	1.26×10^{-14} ‡	56.687 ‡	2.72×10^{-15} ‡	3785 ‡
	FORMI	1.26×10^{-14} ‡	56.687 ‡	2.72×10^{-15} ‡	213 †
Fly Trap	Dijkstra	3.95×10^{-12} †	43.533	1.67×10^{-12} †	17364
	Binary A*	1.10×10^{-6}	33.936 †	1.51×10^{-7}	3844
	Ordinary quadtree	2.29×10^{-10} ‡	40.157 ‡	9.74×10^{-11} ‡	3353 ‡
	FORMI	2.29×10^{-10} ‡	40.157 ‡	9.74×10^{-11} ‡	113 †
Randomized Map	Dijkstra	1.28×10^{-11} †	28.068	6.30×10^{-12} †	8697
	Binary A*	5.68×10^{-7}	21.465 †	8.73×10^{-8}	2133 ‡
	Ordinary quadtree	2.99×10^{-10} ‡	26.203 ‡	1.43×10^{-10} ‡	21881
	FORMI	2.99×10^{-10} ‡	26.446	1.43×10^{-10} ‡	1077 †

†The best value achieved in the test. ‡The second-best value achieved in the test.

TABLE II

RESULTS FROM THE INTEL MAP

Algorithm	Avg. coll. prob.	Path length	Max. coll. prob.	Number of collision checks
Dijkstra	0.2883 †	65.2274	0.0189 †	25759
Binary A*	0.9681	50.8024 †	0.0379	12160 ‡
Ordinary quadtree	0.6318 ‡	62.7892 ‡	0.0246 ‡	40033
FORMI	0.6318 ‡	62.7892 ‡	0.0246 ‡	2305 †

†The best value achieved in the test. ‡The second-best value achieved in the test.

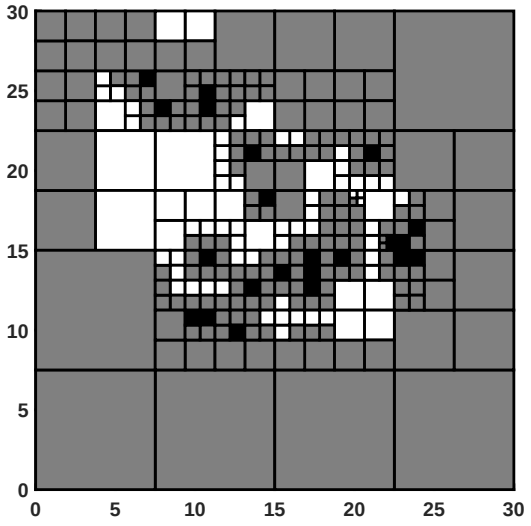

Fig. 5. Modified quadtree in the randomized map, based on the proposed algorithm.

elliptical obstacles as our approach, so that the partly or completely occupied parts of the grid were marked as occupied and others as free. For this method, the A* search was 8-directional, using the octile distance as cost heuristic. The second approach used Dijkstra's algorithm on a continuous-valued occupancy grid. The continuous-valued occupancy grid was generated from bivariate probability distributions generated based on the ellipses. These distributions were then used to calculate a collision probability over each grid square, or in other words, the probability that at least one obstacle is in the given grid square. The third approach used an ordinary quadtree for obstacle decomposition. This approach treated the obstacles as solid ellipses, similar to our approach, and the used path cost evaluation method was also the same as in our approach. In addition, the maximum depth in the quadtree was the same as in our approach. These algorithms were chosen because running a graph search algorithm on an occupancy grid is a natural solution to the path planning problem if the obstacle data can be integrated to an occupancy grid, and the ordinary quadtree approach highlights the differences between it and our approach. In addition, all of the obstacles are evaluated in this approach. Finding the probability values for an occupancy grid is often challenging in real-world scenarios, but often these can be approximated, for example by distances between obstacles and the vehicle center point. These approximations usually increase the number of explored nodes compared to a binary grid. In addition, the collision probabilities also offer a

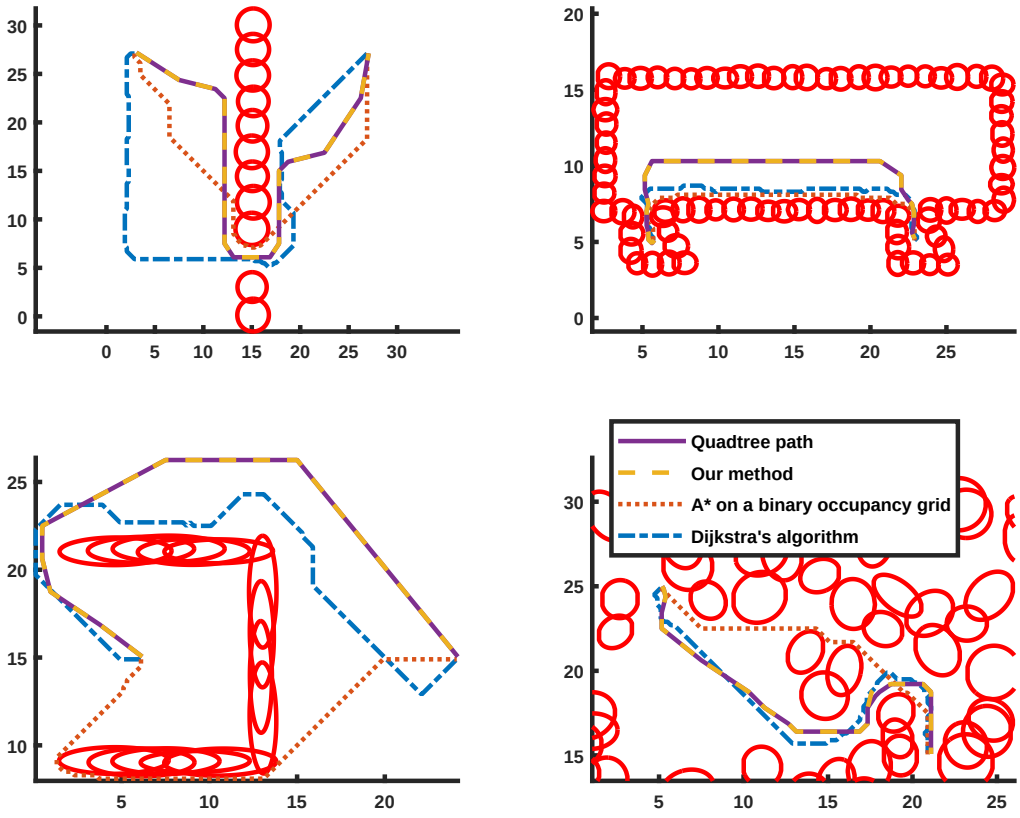

Fig. 6. Test scenarios to highlight differences among 4 methods in their planned path on the same synthetic map: Biharmonic potential field, A* for optimum probability on continuous-valued occupancy grid, A* for shortest path on binary occupancy grid, and the proposed method in this paper. Numerical properties of each path is addressed in Table I. Units for all the axes are meters.

reasonable evaluation method for path quality.

B. Comparison to traditional quadtree

FORMI was compared to a traditional quadtree with the same pathfinding heuristics. Almost the same path, with one different grid square, was found, but the size of the traditional quadtree required significantly more collision checks. In the traditional quadtree, 21881 leaf nodes were evaluated, while in our method only 1077 leaf nodes were evaluated. The quadtrees and the paths by both algorithms are presented in the figures below.

C. Evaluation method

The generated paths were evaluated based on the criteria of maximum and average collision probability, path length and number of collision checks. The collision probabilities were estimated by finding the collision probability under all the grid squares used by the path. For this, the paths

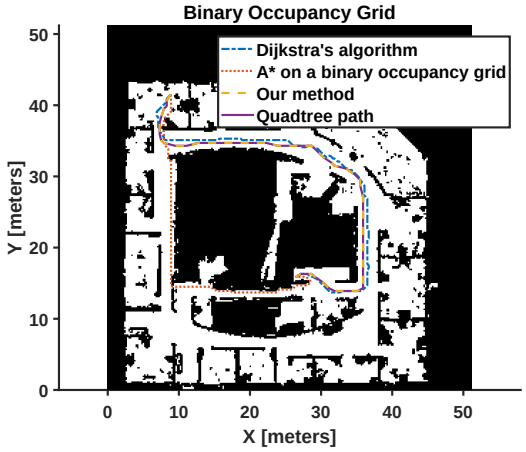

Fig. 7. The map of Intel research lab in Seattle

for the quadtree searches were generated from the box path by finding the center points of the edges between two consecutive boxes, drawing a line through these points and constraining the line to the grid squares. The maximum and average values are used to describe the paths.

D. Performance comparison

The Fig. 6 describes several routes planned by the algorithms, and the Table I describes the evaluation results. In all of these routes, the starting point is on the left and the goal point is on the right.

The A* on a binary occupancy grid found the shortest paths in all of the maps. As a drawback, the path nearly touches the obstacles in all the maps, because the distance is only caused by the map discretization. In the maps with large, empty areas the search explored more vertices than the ordinary quadtree explored leaf nodes, but in more cluttered maps it searched fewer vertices.

The paths found by the Dijkstra's algorithm had the lowest collision probabilities, but the number of searched nodes was also the highest in all of the maps. In addition, far away from obstacles, the path has unnecessary direction changes. This is caused by the extremely low collision probabilities, causing the search to evaluate different nodes equally. It is possible to find a compromise between these two searches, by using both distance and collision probability metrics, but this compromise solution will search more nodes than the binary occupancy grid approach and it will have a higher collision probability than with the Dijkstra's algorithm.

FORMI was not outstanding according to any of the path quality criteria, but it stayed far from obstacles in all the maps, finding paths in the same homotopy class as the Dijkstra's algorithm in all of the maps. As the main benefit, this algorithm required much less collision checking. In the 'Office' map, the quadtree discretization forces the path to be further from the obstacles than necessary.

The algorithms were also run on a map provided by the University of Freiburg. For this map, the collision probability was replaced by map values run through a Gaussian filter. The results were in line with the other tests.

IV. CONCLUSIONS

We have introduced an resolution complete algorithm for finding collision-free paths with minimal collision checking compared to contemporary planning methods. **FORMI** significantly reduces computation costs, without causing a big drawback in other criteria. The algorithm preserves near optimal results in terms of path length, obstacle distance, and smoothness. To highlight the efficacy, we compare the algorithm with three other well-known algorithms based on the path properties and computation time in certain benchmark scenarios.

In path length and collision probability, **FORMI** achieves a reasonable compromise between the optimal grid-constrained length and the optimal collision probability. These are achieved with minimal collision checking due to our modifications in search logics on quadtree. This allows the most significant speedup with relatively complex envelopes. In addition to this, the algorithm generates a modified quadtree representation that includes obstacle-free, occupied and partially occupied cells.

As a future work, we intend to use this representation for path planning of a mobile manipulator that requires mappings between joint space and task space with a much higher resolution for spatial motions. Due to the successful history of extending quadtree (2D) methods to octree (3D), we expect a significant improvement in those areas too.

REFERENCES

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," in *Motion and operation planning of robotic systems*, pp. 3–27, Springer, 2015.

[3] B. Chazelle, "Approximation and decomposition of shapes," *Algorithmic and Geometric Aspects of Robotics*, vol. 1, pp. 145–185, 1985.

[4] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 113–120, IEEE, 1996.

[5] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[6] A. A. Masoud, S. A. Masoud, and M. M. Bayoumi, "Robot navigation using a pressure generated mechanical stress field:" the biharmonic potential approach"," in *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pp. 124–129, IEEE, 1994.

[7] S. Liu, M. Watterson, K. Mohta, K. Sun, S. Bhattacharya, C. J. Taylor, and V. Kumar, "Planning dynamically feasible trajectories for quadrotors using safe flight corridors in 3-d complex environments," *IEEE Robotics and Automation Letters*, vol. 2, no. 3, pp. 1688–1695, 2017.

[8] R. Deits and R. Tedrake, "Efficient mixed-integer planning for uavs in cluttered environments," in *2015 IEEE international conference on robotics and automation (ICRA)*, pp. 42–49, IEEE, 2015.

[9] H. P. Moravec, "Sensor fusion in certainty grids for mobile robots," *AI magazine*, vol. 9, no. 2, p. 61, 1988.

[10] M. Montemerlo, S. Thrun, D. Koller, B. Wegbreit, *et al.*, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Aaai/iaai*, vol. 593598, 2002.

[11] F. Kamil, S. Tang, W. Khaksar, N. Zulkifli, and S. Ahmad, "A review on motion planning and obstacle avoidance approaches in dynamic environments," *Advances in Robotics & Automation*, vol. 4, no. 2, pp. 134–142, 2015.

[12] A. Orthey, A. Escande, and E. Yoshida, "Quotient-space motion planning," *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8089–8096, 2018.

[13] B. C. Heinrich, D. Fassbender, and H.-J. Wuensche, "Faster collision checks for car-like robot motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7533–7538, IEEE, 2018.

[14] J. Tordesillas, B. T. Lopez, and J. P. How, "Fastrap: Fast and safe trajectory planner for flights in unknown environments," *arXiv 1903.03558*, 2019.

[15] M. M. Aref, R. Ghabcheloo, and J. Mattila, "A macro-micro controller for pallet picking by an articulated-frame-steering hydraulic mobile machine," *IEEE Int. Conf. on Robotics and Automation (ICRA), Hong Kong*, pp. 6816–6822, 2014.

[16] L. Jaulin, "Path planning using intervals and graphs," *Reliable computing*, vol. 7, no. 1, pp. 1–15, 2001.

[17] H. Ratschek and J. Rokne, "A two-dimensional ellipse–rectangle intersection test," *Journal of Mathematical Modelling and Algorithms*, vol. 1, no. 4, pp. 243–255, 2002.

[18] A. Abbadi and V. Přenosil, "Safe path planning using cell decomposition approximation," *Distance Learning, Simulation and Communication*, vol. 8, 2015.