Incremental Adaptive Probabilistic Roadmaps for Motion Planning under Uncertainty

Weria Khaksar, Md. Zia Uddin, *Senior Member, IEEE*, and Jim Torresen, *Senior Member, IEEE*

Abstract— **As the application domains of sampling-based motion planning grow, more complicated planning problems arise that challenge the functionality of these planners. One of the main challenges in the implementation of a sampling-based planner is their weak performance when reacting to uncertainty in robot motion, obstacles motion, and sensing noise.**

In this paper, a multi-query sampling-based planner is presented based on the optimal probabilistic roadmaps algorithm that employs a hybrid sample classification and graph adjustment strategy to handle diverse types of planning uncertainty such as sensing noise, unknown static and dynamic obstacles and inaccurate environment map in a discrete-time system. The proposed method starts by storing the collision-free generated samples in a matrix-grid structure. Using the resulted grid structure makes it computationally cheap to search and find samples in a specific region. As soon as the robot senses an obstacle during the execution of the initial plan, the occupied grid cells are detected, relevant samples are selected, and in-collision vertices are removed within the vision range of the robot. Furthermore, a second layer of nodes connected to the current direct neighbors are checked against collision which gives the planner more time to react to uncertainty before getting too close to an obstacle. The simulation results in problems with various sources of uncertainty show significant improvement comparing to similar algorithms in terms of failure rate, processing time and minimum distance from obstacles. The planner was also successfully implemented on a TurtleBot in two different scenarios with uncertainty.

I. Introduction

In the field of motion planning, sampling-based planners have been successfully applied to solve difficult problems in high-dimensional spaces. Theses algorithms are unique in the fact that planning occurs by sampling the configuration space. Original sampling-based planners such as Probabilistic Roadmaps (PRM) [1], Rapidly-Exploring Random Trees (RRT) [2], and Expansive Space Trees (EST) [3], are proved to be probabilistically complete as the probability of finding a solution in these planners is one when the input size goes to infinity. These algorithms have been improved further to achieve some form of optimality in the generated solutions. Optimal sampling-based planners such as PRM* and RRT* [4] are asymptotically optimal as the solutions found by these algorithms converge asymptotically to the optimum, if one exists, with the probability one as the input size goes to infinity.

The failure of a robot to navigate in uncertainty is becoming an important challenge as the robots are finding their way to operate in our homes, offices and outdoor environments and participate in complex tasks such as health monitoring and elderly care. Because of the uncertainty associated with a robot's motion and its sensory readings, the real robot state is often not available. Therefore, any path planner must be able to account for these uncertainties to provide safe and collision-free navigation plans. Uncertainty in path planning is often caused by three main sources including motion error, sensing error, and imperfect environment map [5]. Despite the proven advantages of sampling-based algorithms in path planning and even in other fields such as computer games and drug design [6], they fail to deal with planning under uncertainty. The main necessity for a typical sampling-based path planner is to have a map of the environment or the knowledge to decide whether any given configuration is in collision with obstacles or not. These algorithms generate random or semi-random samples in the free configuration space and therefore, they should be able to detect collisions beforehand. This restrictive assumption strongly limits the applicability of sampling-based planners to robots operating in uncertain environments. In addition, as a part of most of randomized algorithms, a local planner should be available to detect possible collision-free connection between two given configurations. Moreover, dealing with dynamic obstacles poses additional complexity to the uncertain path planning problem. Not knowing the position of a dynamic obstacle or equivalently the collision status of a configuration over time, leads a typical sampling-based planner to failure. Recently, conventional sampling-based planners have been upgraded to deal with some levels of uncertainty including sensing error, uncertain environment map and dynamic obstacles. These methods will be discussed in the next section however, an overall evaluation on the performance shows that they are computationally demanding as compared to their counterparts that do not consider uncertainty. Furthermore, focusing on one aspect of

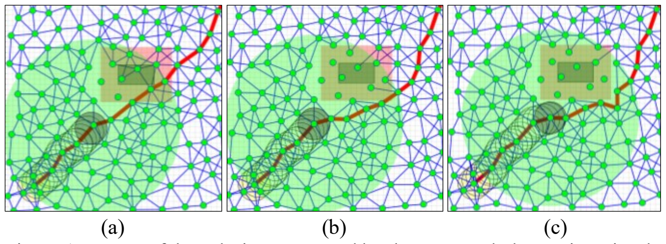

Figure 1. A part of the solution computed by the proposed planner in a simple 2D environment with a static unknown obstacle. (a) The robot is following the pre-planned path. (b) As the robot senses the new obstacle, the graph is adjusted and the path is repaired accordingly. (c) As the robot keeps moving, the graph keeps being adjusted and the generated path is improved during the planning. The obstacle and its expanded version are shown by gray and red squares respectively. The robot is the grey circle with the green circle around it as the vision range and the generated path is the thick red line.

* This work is supported by the Research Council of Norway as a part of the Multimodal Elderly Care Systems (MECS) project, under grant agreement 247697.

Authors are with the Robotics and Intelligent Systems Group (ROBIN), Department of Informatics, University of Oslo, Blindern, 0316 OSLO, Norway. E-mail: {weriak*, mdzu, jimtoer} @ ifi.uio.no.

uncertainty normally requires deterministic knowledge on other aspects. For instance, having an efficient path planner to deal with dynamic obstacles require a very accurate sensory system. In other words, the cumulative effect of all sources of uncertainty can be difficult to model and account for in the planning phase before task execution.

In this paper, an extension of the optimal Probabilistic Roadmaps (PRM*) [4] is proposed which is able to handle different types of planning uncertainty in a single package without a considerable increase in the computational cost. First, a sampling radius $R_s(n)$ is applied to the sampling process to have a more sparse and monotone graph and avoid oversampling. Second, it stores the generated samples in a grid-based matrix GR, based on the corresponding cartesian coordinates $(x_i, y_i)^T$ to make it computationally cheap when performing regional adaptation on the graph. Next, an uncertainty matrix, COL, will be updated with a predefined frequency, Δt, which directly updates itself based on the information provided by the robot's sensor(s). The most important part of the proposed algorithm is a graph-adjustment component which adjusts the resulted graph in real-time by refining not only in-collision nodes but also the corresponding connected neighbors. The proposed incremental graph adjustment process enables the planner to deal with any new obstacle in the same way without knowing whether the obstacle is static or dynamic. Other types of uncertainty such as noisy sensors or inaccurate maps are treated the same way since regardless of the source of uncertainty, it results in encountering an obstacle when it wasn't accounted for. Figure 1 shows the performance of the algorithm in dealing with a static unknown obstacle. The performance of the proposed planner is tested in several simulation scenarios with uncertainty. Furthermore, to test the results on a real robot, an implementation procedure is introduced that converts the output of the planner to a control vector for a non-holonomic mobile robot moving in a 2D indoor environment. This implementation is tested on a TurtleBot in two different path planning scenarios with uncertainty.

The remainder of this paper is organized as follows: Section II is a summary of the related literature. Section III contains formal definitions and notations about the problem while section IV describes the proposed algorithm. The simulation and experimental results are provided in section V and finally the work is concluded and potential future work is discussed in section VI.

II. Background

A common restrictive assumption in sampling-based algorithms is that the environment is well defined such that the relative location of the robot to obstacles is completely known. This assumption is valid in static environments where industrial manipulators are used or in CAD applications in which the environment is user-defined. For autonomous robots operate in uncertain environments that cannot be modeled or estimated, the assumption of a well-defined static environment does not hold true. There is an uncertainty that arises because of sensing errors and noise and the imprecision of actuators and other uncontrollable factors such as unknown static or dynamic obstacles [7]. In the past years, sampling-based algorithms have been updated to deal with various sources of uncertainty. Based on single or multi-query nature of the base planner, different improvements have been proposed in the presence of uncertainty. Even though for a single-query planner, regenerating a search tree may be a valid approach, it requires appropriate parameters tuning and various heuristics in different instances. There are several extensions of RRT algorithm to deal with uncertainty [8-11] which mostly deal with dynamic obstacles and show poor performances when facing other forms of uncertainty such as imperfect sensing or noisy environment maps.

In the field of multi-query algorithms, several extensions of PRM planner have been introduced to deal with uncertainty. A PRM was proposed for dynamic motion planning based on regenerating a roadmap while assuming an obstacle-free space [12] while the data structure of PRM was improved to accommodate changes in the environment and consequently, in the roadmap. However, this algorithm only handles dynamic environment. A similar approach attempts to use a tree-based planner to connect the roadmap nodes in dynamic environments and encodes obstacle positions in local connections [13]. A generalized PRM was introduced in surroundings where obstacle movements are restricted to local sectors [14]. PDR maintains a roadmap whose paths can be deformed, thus numerous paths can be obtained between two configurations [15]. A sampling-based motion planner was proposed to deal with sensing uncertainty through a utility guided process that incorporates uncertainty directly into the planning procedure [16]. Guided Cluster Sampling (GCS) is a global motion planner which was introduced to handle problems with uncertainty. GCS uses the point-based Partially Observable Markov Decision Process (POMDP) approach [5]. GCS uses domain specific properties to construct a more suitable sampling strategy. A real-time path planner was proposed that guarantees probabilistic feasibility for autonomous robots with uncertain dynamics operating amidst dynamic obstacles with uncertain motion patters [17]. This method builds a learned motion pattern model by combining the flexibility of Gaussian process with the efficiency of RRT planner. BU-RRT* [18] is a novel optimizing sampling-based motion planner that guarantees feasibility of linear systems subject to a bounded uncertainty. FIRM [19] is a feedback-based information roadmap for planning under uncertainty which is a belief-space variant of the PRM planner. In this method, the costs associated with the edges are independent of each other and it preserves the optimal substructure property. FIRM also relies on feedback from local planners to reduce the uncertainty propagation between states. The problem of motion planning for a linear system subject to Gaussian motion noise was considered and the CC-RRT*-D planner [20] was developed to deal with risk-aware path planning under uncertainty. This planner employs the chance-constraint approximation and leverages the asymptotically optimal property of RRT* framework to compute risk-aware and asymptotically optimal trajectories under motion uncertainty. A Sampling-based real-time motion planning algorithm has been proposed [21] for planning under state uncertainty which is an extension of the closed-loop rapid belief tree. A RRT-based planner (HFR) was reported that is able to perform high-frequency re-planning under uncertainty using parallel sampling-based planners [22]. RRT^X [23] is a tree-based asymptotically optimal planner which is capable of solving dynamic motion planning problems by refining and repairing the same graph over the entire navigation. Whenever obstacles change or the robot moves, a graph rewiring cascade quickly remodels the existing search-graph and repairs its shortest path. Recently, a localization-aware sampling-based planner has been introduced [24] for incremental motion planning under uncertainty using a measure of localization ability of the samples. This planner puts more samples in regions where sensor data is able to achieve higher uncertainty reduction while maintaining adequate samples in regions where uncertainty reduction is poor.

Most of the abovementioned planners focus on one source of uncertainty and at some level require accuracy on other aspects which is not the case in complex planning problems under different forms of uncertainty. Furthermore, total processing time of the planner when dealing with uncertainty is a crucial factor which usually is neglected. Having a motion planning algorithm with a computationally expensive process, is not practical when dealing with a real robot. To understand the effect of high process runtime in the implementation of a planner consider a mobile robot that stops for a minute each time it senses a new obstacle.

III. PROBLEM FORMULATION

In this section, the basic definitions and descriptions of the proposed algorithm is provided. A mobile robot is moving in a d-dimensional state space. Initially, there is a map of the environment that at least specifies the boundaries of the space. The only requirement of the planner is to have or to be able to generate an initial solution before the navigation starts.

Let $Q \subseteq \mathbb{R}^d$ be the state space of the navigation problem which includes two main subset such as $Q_{obs} \subset Q$ where the state is in collision with obstacles, and $Q_{free} = Q \backslash Q_{obs}$ where the robot is free to move. Let $(x_{init}, y_{init})^T$ and $(x_f, y_f)^T$ be the desired initial and final configurations of the robot respectively.

Definition 1 (Samples): Let $(x_i, y_i)^T \in Q$ be a randomly selected configuration in Q and $S = \{(x_i, y_i)^T, i = 1, \dots, n\}$ be the set of n randomly generated samples in Q. We consider S as a valid set of samples if:

$$(x_i, y_i)^T \in Q_{free}, \qquad \forall\, (x_i, y_i)^T \in S \text{ and}$$

$$||(x_i, y_i)^T, (x_j, y_j)^T|| \le R_S(n), \qquad \forall\, (x_j, y_j)^T \in S$$

$$S = \begin{bmatrix} x_1 & \dots & x_n \\ y_1 & \dots & y_n \end{bmatrix}$$

where $||A, B||$ denotes the Euclidean distance between two points A and B and $R_S(n) = [L(Q_{free})(n - \lambda)/(\pi n^2)]^{1/2}$ is a sampling radius based on the number of samples, n. L is the Lebesgue measure (i.e. volume) and λ is a positive scaling constant. More details about the sampling radius can be found in [25].

Definition 2 (Collision-free path): A sequence of states, $\sigma: [0,1] \to \mathbb{R}^d$ is called a collision-free path between $(x_i, y_i)^T$ and $(x_j, y_j)^T$ if:

- $\sigma(\tau) \in Q_{free}, \ \forall\, \tau \in [0,1]$
- $\sigma(0) = (x_i, y_i)^T \text{ and } \sigma(1) = (x_j, y_j)^T$

If $\sigma(0) = (x_i, y_i)^T, \sigma(1) = (x_j, y_j)^T$ and for all $\tau \in (0,1)$, $\sigma(\tau) = 0$, then there is a direct collision-free path, σ_D, between $(x_i, y_i)^T$ and $(x_j, y_j)^T$

Definition 3 (Graph): Let $CON_{n\times n} = [con_{i,j}]$ be a matrix showing the connection between all $(x_i, y_i)^T, (x_j, y_j)^T \in S$. Any two pair of samples in S are connected if there exist a collision-free direct path between them and the length of this path is less than a given connection radius $R_c(n)$.

$$con_{i,j} = \begin{cases} ||\sigma_D(i,j)|| & \text{if:} \\ \sigma_D(i,j) \neq \emptyset \ \text{ and } & ||\sigma_D(i,j)|| \le R_c(n) \\ 0 & otherwise \end{cases}$$

$$R_c(n) = \gamma[\log(n)/n]^{1/d}$$

$$\gamma > \gamma^* = 2(1 + 1/d)^{1/d}[L(Q_{free})/\zeta_d]^{1/d}$$

$$CON = \begin{bmatrix} con_{1,1} & \cdots & con_{1,n} \\ \vdots & con_{i,j} & \vdots \\ con_{n,1} & \cdots & con_{n,n} \end{bmatrix}$$

where d is the dimension of the configuration space and ζ_d is the volume of the unit ball in the d-dimensional Euclidean space. The concept of connection radius was taken from the PRM* [4] algorithm to guarantee asymptotically optimal solutions.

Definition 4 (Optimal path planning): Let Σ be the set of all feasible paths between $(x_{init}, y_{init})^T$ and $(x_f, y_f)^T$. The optimal path planning problem between $(x_{init}, y_{init})^T$ and $(x_f, y_f)^T$ can be defined as finding the path σ^*, that minimizes a given cost function, $s: \Sigma \to \mathbb{R}_{\ge 0}$, while connecting $(x_{init}, y_{init})^T$ to $(x_f, y_f)^T$ through Q_{free}.

Definition 5 (Uncertainty): Let $COL_{n\times n}(t)$ be a matrix that represents the uncertainty in the planning problem as a function of time.

$$COL_{n\times n}(t) = [\mathrm{col}_{i,j}(t)], \qquad i,j = 1, \dots, n$$

$$\mathrm{col}_{i,j}(t) = \begin{cases} 0 & \text{if in time } t\text{:} \quad \sigma_D(i,j) \neq \emptyset \\ 1 & otherwise \end{cases}$$

$$COL(t) = \begin{bmatrix} col_{1,1}(t) & \cdots & col_{1,n}(t) \\ \vdots & col_{i,j}(t) & \vdots \\ col_{n,1}(t) & \cdots & col_{n,n}(t) \end{bmatrix}$$

The matrix of uncertainty shows whether, in a specific time t, a given configuration $(x_i, y_i)^T$ is in collision with obstacles or not by the value of $\mathrm{col}_{i,i}(t)$. It also shows if there is a direct path $\sigma_D(i,j)$ between any two configurations $(x_i, y_i)^T, (x_j, y_j)^T$, by the value of $\mathrm{col}_{i,j}(t)$. This matrix will be updated continuously during the navigation by analyzing the readings of the robot's sensory system.

IV. ALGORITHM

In this section, the proposed algorithm is presented in detail which includes the graph construction and graph adjustment. Like any multi-query planner, the graph construction phase starts by learning the configuration space through sampling. Initially, it requires to have an approximation on the boundaries of the space. Having any additional information is optional and does not affect the performance of the planner.

According to the initial available map, the sampling takes place and the set of all samples $S = \{(x_i, y_i)^T\}$ is created and filled with randomly selected collision-free configurations. At the same time, another matrix structure $GR = \{(x_i, y_i)^T\}$ is created which stores the elements of S in a grid structure with a predefined resolution $\Delta_{GR} < R_S(n)$. The main difference between S and GR is the order of storing the coordinates. While S saves the coordinates on a first-come first-served base, GR stores the coordinates in a 2D structure based on their corresponding grid cell. Considering a sample $(x_i, y_i)^T$ in S, the corresponding position of $(x_i, y_i)^T$ in GR, (α, β) can be calculated as follows:

$$\alpha = \left\lceil {}^{x_i}/_{\Delta_{GR}} \right\rceil, \qquad \beta = \left\lceil {}^{y_i}/_{\Delta_{GR}} \right\rceil$$

where $\lceil \alpha \rceil$ shows the smallest positive integer, which is greater than or equal to α. Having a grid resolution smaller than the sampling radius $R_S(n)$ guarantees that any given cell in the grid matrix at most, includes one sample. Using this simple structure make it computationally cheap to search the visible area around the robot and find the neighbor nodes without searching the whole graph. At the current position of the robot, the surrounding grid cells are considered as visible if the center of the cell is within the sensing range. This strategy provides enough number of visible grid cells without being pessimistic or optimistic as presented in Figure 2.

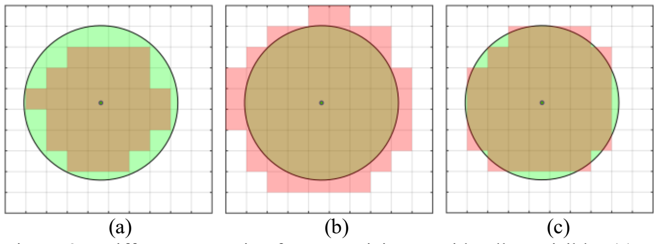

Figure 2. Different strategies for recognizing a grid cell as visible. (a) A pessimistic strategy that accepts a cell if the entire cell is within the sesnsing range, (b) an optimistic strategy that accepts a cell if it is partially visible, and (c) the proposed strategy that recognizes a cell if the center of the cell is visible. Unrecognized cells are shown by white color.

After generating the samples and storing them in GR, the graph will be constructed based on the values in CON matrix and an initial solution will be generated using a graph search algorithm such as A*. Now the robot is ready to move towards the final position. Algorithm. 1 presents the Graph_Construct phase. As the robot starts to move, the surrounding area is scanned and visible grid cells, as shown in Figure 2(c), are marked as free or occupied based on the readings of the sensor(s), $\rho(\theta, \Delta t)$. By knowing the occupied grid cells within the vision range, it is possible to update the uncertainty matrix COL. For every grid cell within the range, the value of the corresponding nodes in the uncertainty matrix will be updated. If a node was defined as free before and now, the corresponding grid cell to that node is not reachable, i.e. occupied, the status of that node will be updated to occupied, $\text{col}_{i,i}(t) = 1$. On the other hand, if a node was marked as occupied before and now it is reachable, it's corresponding uncertainty value is updated to 0. The next step is to adjust the graph based on the new values in the matrix of uncertainty as presented in Algorithm 2. An instance of the graph adjustment is shown in Figure 3, where the graph adjustment is shown in two different positions and some vertices are removed or added back to the roadmap.

ALGORITHM 1: GRAPH_CONSTRUCT

```
$S \leftarrow \{(x_i, y_i)^T\}, GR \leftarrow (\alpha_i, \beta_i)^T, CON \leftarrow \{con_{i,j}\}, i, j = 1, \dots, n$
while $reach = false$ and $fail = false$
|   $time = time + \Delta t$;
|   Scan: $\rho(\theta, time)$
|   Update $COL_{n\times n}$
|   Graph-Adjust
|   GraphShortestPath[$sparse(CON), (x_c, y_c)^T, (x_f, y_f)^T = [dist, path]$
|   if $\|(x_c, y_c)^T, (x_f, y_f)^T\| \le \varepsilon$   then
|   |   $reach \leftarrow true$
|   If $path = \emptyset$   then
|   |   $fail \leftarrow true$, Return
|   $Move \leftarrow d = V \times \Delta t$
```

ALGORITHM 2: GRAPH_ADJUST

```
$COL_{n\times n}(t) = [\text{col}_{i,j}(t)], \quad i, j = 1, \dots, n$
for all $(x_i, y_i)^T \in visible\ range$
|   if $col_{i,c} = 1$ and $con_{i,c} \neq 0$ then
|   |   $CON(i,:) = 0, \quad CON(:,i) = 0;$
|   if $col_{i,c} = 0$ and $con_{i,c} \neq 0$ then
|   |   for all $(x_j, y_j)^T \in visible\ range$
|   |   |   if $\text{col}_{i,j}(t)=1$ then
|   |   |   |   $CON(i,j) = 0, CON(j,i) = 0;$
|   if $col_{i,c} = 0$ and $con_{i,c} = 0$ then
|   |   $CON(i,c) = \|(x_i, y_i)^T, (x_c, y_c)^T\|, CON(c,i) = CON(i,c);$
|   |   for all $(x_j, y_j)^T \in visible\ range$
|   |   |   if $\text{col}_{i,j}(t)=0$ and $con_{i,j} \neq 0$ then
|   |   |   |   $CON(i,j) = \|(x_i, y_i)^T, (x_j, y_j)^T\|, CON(j,i) = CON(i,j);$
```

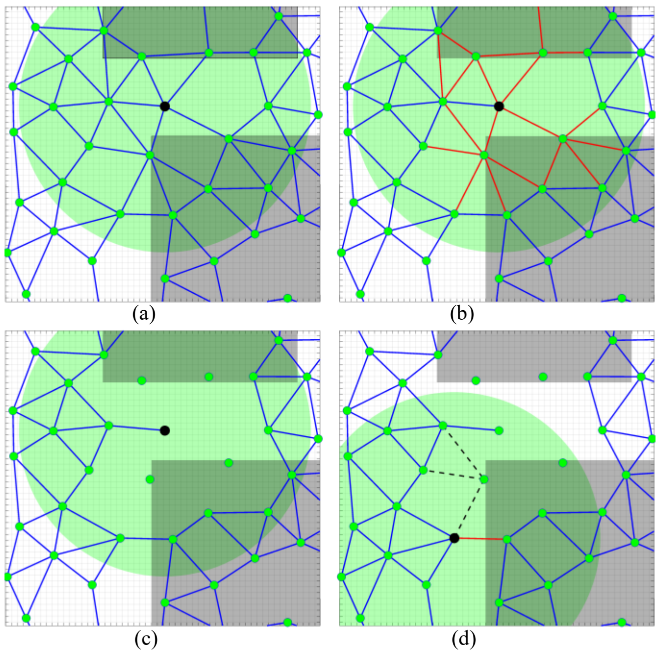

Figure 3. The Graph-Adjust procedure. (a) The original graph before sensing the obstacle, (b) the obstacles are detected and in-collision edges are determined as shown by red color, (c) the adjusted graph after removing the in-collision edges and (d) after the robot moves to another position with a different sensing outcome, all of the previously removed edges are added back to the graph if the corresponding uncertainty values are not zero. The black point is the robot's current position and the green circle represents the robot's vision range.

Now, the graph connection matrix, CON will be updated based on the changes in COL. First, for all nodes within the vision range, if there is a direct path $\sigma_D(c, i)$ between robot's current position $(x_c, y_c)^T$ and that node $(x_i, y_i)^T$ in the current CON, i.e. $con_{i,c} \neq 0$, and the corresponding value of $\text{col}_{i,i}(t)$ has been updated to 1, then all of the connections of that node will be removed in the graph connection matrix, $CON(i,:) = 0$, and $CON(:,i) = 0$. The opposite procedure applies on the nodes that have been disconnected before and now are have a collision-free connection to the current node. Next, for all other nodes within the sensing range and connected to the neighbors of the current node, in-collision connections are removed and collision-free connections are added to adapt the graph to the uncertainty of the space. The graph adjustment to two immediate layers of neighbors enables the robot to detect collision without getting close to the obstacles. This process can be extended for more than two layers; however, it worsens the computational cost of the process since more nodes need to be checked for collision. Applying the Graph_Adjust procedure has another benefit that improves the planning efficiency. According to lines 9-13 in Algorithm. 2, if there are some nodes that have been removed from the graph in previous iterations of the algorithm and now the planner can conclude that they are not in collision, they will be added back to the graph. This situation happens when an obstacle is blocking a collision-free node or there is a dynamic object in the environment. This is more effective than adding back the edges to the graph as soon as they are out of the vision range. Adding back the removed edges as soon as they are not visible anymore may cause a local minimum in which the robot keeps moving between two positions forever. Figure 4 shows an example of the local minima situation.

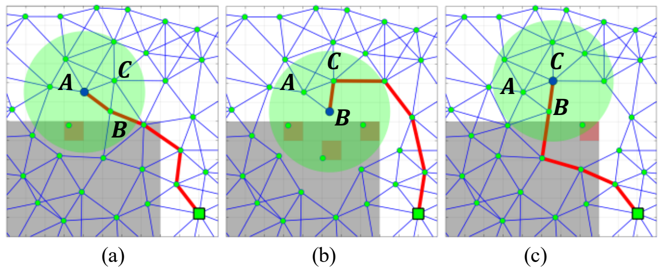

Figure 4. A local minimum trap where the robot moves between two local optima forever. (a) The robot is at $\boldsymbol{A}$ and the current shortest path goes through point $\boldsymbol{B}$. (b) When the robot reaches point $\boldsymbol{B}$, the graph is adjusted and now the path goeas through point $\boldsymbol{C}$. (c) As the robot reaches point $\boldsymbol{C}$, some parts of the nodes that are out of the vision range are added back to the graph which forces the robot to move back to point $\boldsymbol{B}$. This loop continious for ever.

Limiting the graph adaptation to the visible region avoids local minima. Now that the graph was adjusted, the shortest path from the robot's current position to the final configuration will be calculated and the robot continues moving but in the latest generated path. This procedure repeats with a constant frequency Δt (sec.) until the robot reaches the obstacle or concludes that no solution exists. As presented in Figure 3, the proposed planner is capable of disconnecting the in-collision nodes and reconnecting free nodes.

V. RESULTS AND DISCUSSION

To evaluate the performance of the algorithm and compare it with similar planners, the algorithm was simulated and further implemented on a real mobile robot. The results are described in the following sections.

A. Simulation Studies

The planner was simulated in MatLab R2017a to perform in four different planning scenarios as presented in Figure 4. All simulations were run on a desktop with a 3.40-GHz Intel Core i7 processor with 32 GB of memory. In the first case, a mobile robot is moving in a 2D bounded environment without initially having any obstacles. As soon as the robot finishes the pre-planning, two polygonal obstacles are added to the environment and the robot starts to navigate without having any knowledge about them. In the second scenario, the robot is supposed to move in a known maze, but after reaching the middle of the maze, a door is closed which blocks the current path of the robot and the current solution becomes infeasible. Next, the robot is supposed to move in a plain 2D bounded environment which later contains two dynamic obstacles moving in different directions. Finally, the robot is given a noisy map while the real map of the environment is quite different. The planner is required to guide the robot through the actual map only by initially having the noisy map. Since similar situations are created in the experimental studies on a TurtleBot, the size of the environments and the robot was set to be exactly the same as the experiments which will be discussed later. The performance of the planner is compared to six similar algorithms as presented in Table 1 for dealing with different types of planning uncertainty as described in section II including GCS [5], RR-GP [17], BU-RRT* [18], FIRM [19], CC-RRT* [20] and RRBT-LAS [24]. The results are described based on path length (PL) in meters, which is the total travelled distance by the robot, processing time (RT) in seconds, which is the total planning time minus the navigation time, failure rate (FL), which is the percentage of failure, and the minimum shortest distance to the obstacles (DM) in meters, which is calculated using the following equation:

TABLE 1. PERFORMANCE COMPARISON BASED ON THE SIMULATION RESULTS.

$n = 200$		GCS [5]	RR-GP [17]	BU-RRT* [18]	FIRM [19]	CC-RRT*-D [20]	RRBT-LAS [24]	Proposed Planner
Scene 1	PL(m) / Std	13.38 / 1.63	14.87 / 2.66	13.85 / 1.55	12.44 / 3.55	13.80 / 2.59	14.53 / 2.45	**11.71 / 0.57**
	RT(sec) / Std	18.57 / 2.55	12.36 / 4.00	11.90 / 2.57	15.87 / 4.08	12.39 / 1.89	11.99 / 5.66	**4.66 / 0.08**
	Fail(%)	1	2	2	1	0	2	**0**
	DM(m) / Std	0.20 / 0.02	0.22 / 0 05	0.19 / 0 05	0.19 / 0.06	0.20 / 0 08	0.20 / 0.07	**0.21 / 0.03**
Scene 2	PL(m) / Std	10.22 / 2.87	12.90 / 1.73	12.38 / 1.85	9.63 / 1.22	13.13 / 2.71	11.64 / 2.99	**9.18 / 0.62**
	RT(sec) / Std	5.80 / 1.32	5.19 / 2.66	4.87 / 1.96	6.16 / 1.24	4.21 / 1.99	6.01 / 2.59	**2.19 / 0.11**
	Fail(%)	6	5	5	4	2	3	**0**
	DM(m) / Std	0.29 / 0.09	0.31 / 0.08	0.32 / 0.11	0.28 / 0.09	0.33 / 0.05	0.34 / 0.08	**0.34 / 0.05**
Scene 3	PL(m) / Std	18.20 / 3.18	21.04 / 3.80	19.19 / 4.00	18.34 / 3.33	17.55 / 4.50	16.87 / 2.97	**15.96 / 2.87**
	RT(sec) / Std	5.19 / 0.28	4.32 / 0.44	4.08 / 0.60	3.50 / 0.29	4.11 / 0.70	4.08 / 0.22	**2.44 / 0.12**
	Fail(%)	1	1	0	0	0	0	**0**
	DM(m) / Std	0.25 / 0.08	0.27 / 0.06	0.28 / 0.08	0.27 / 0.08	0.29 / 0.07	0.28 / 0.07	**0.29 / 0.03**
Scene 4	PL(m) / Std	47.44 / 3.19	45.19 / 5.22	40.40 / 5.00	42.99 / 4.13	42.36 / 3.15	41.82 / 3.04	**38.69 / 1.88**
	RT(sec) / Std	12.88 / 0.56	15.90 / 0 78	10.15 / 0.33	10.02 / 0.26	12.55 / 0.52	11.17 / 0.88	**8.77 / 0.14**
	Fail(%)	7	8	8	5	6	5	**2**
	DM(m) / Std	0.28 / 0.07	0.31 / 0.10	0.29 / 0.08	0.29 / 0.09	0.27 / 0.09	0.34 / 0.11	**0.35 / 0.09**

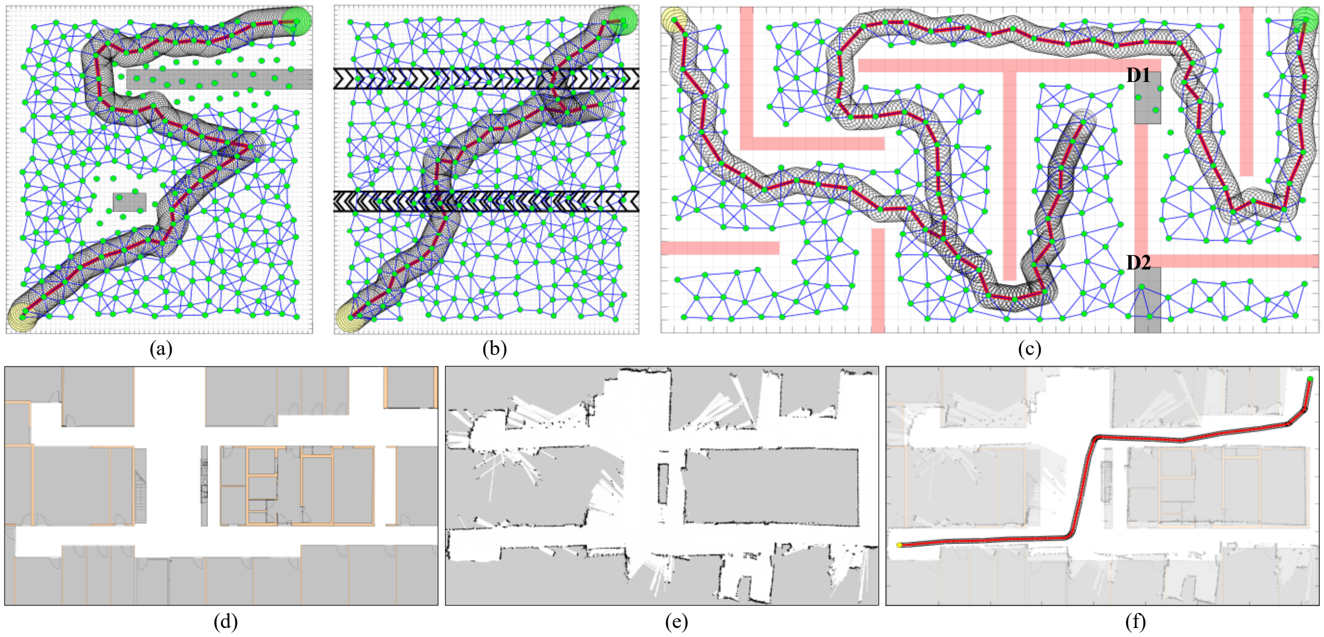

Figure 5. An instance of the simulation results. (a) A plain environment (3.73×4.03(m)) which contains two initially unknown static obstacle and 282 nodes in the graph, (b) same environment but with two dynamic obstacles and 294 nodes in the graph, (c) a maze (5.97×3.22(m)) where two doors will close (D1 and D2) as the robot starts to move and 266 nodes in the graph, (d) the actual map of an office (24×15(m)), (e) a noisy map of the same office, and (f) the solution provided by the proposed planner is shown on the combination of these two maps using a graph with 540 nodes. The robot is a circle with the radius of 0.177(m). The dimensions of the environments and the robot are chosen carefully to match the simulations with the experimental studies.

$$DM = \min_{\omega}\{\min_{\theta}(||(x_c, y_c)^T, obs.||)\}, \quad \omega = \frac{planning\ time}{\Delta t}$$

where θ is the sensing angle of the robot and Δt is the time between two consecutive scans of the environment by the robot. Table 1 shows the results when all planners used a set of 200 samples per run, Euclidean distance for heuristics and local planner, and uniform sampling with the sampling radius with the scaling factor of $\lambda = n^{1/2}$. Instead of using a fixed final configuration, each execution was concluded as successful if the distance of the robot to the goal was less than a fixed distance $D_f = 0.1\ (m)$. For tree-based planners, the fixed step size was replaced by the sampling radius $step\_size = R_s(n)$. During simulations, each actual obstacle was expanded by the size equal to the radius of the robot which for a TurtleBot, $R_{robot} \cong 0.18\ (m)$. Same radius was used against the boundaries of the environments. Since no post-processing was applied to the simulation results, the scanning of the planner was designed to take place each time the robot reaches a new node, which gives an equal number of scans and segments of the final path. The results indicate that the proposed planner outperforms each one of the studied algorithms in all performance variables. The planner maintains a stable path length and distance to the obstacles, while it significantly reduces the processing time and failure rate. As stated before, the processing time includes the initial sampling and roadmap construction time plus the computational cost related to the graph adjustment procedure. The failure rates also indicate the applicability of the planner to planning problems with uncertainty. The planner failed to guide the robot only in the last test environment and only two times out of 100 executions due to the elevated level of inaccuracy and noise in the given map.

B. *Experimental Setup*

To implement the proposed algorithm on a real robot, few modifications are required. Since one of the major drawbacks of sampling-based algorithms is their widely regarded suboptimal paths, we applied a post-processing procedure [26] on the results of the algorithm which can remove the redundant nodes from the final solution. Furthermore, a path smoothing technique was applied to refine the resulted paths by finding the inner circle of each three consecutive nodes on the post-processed path as presented in Figure 6.

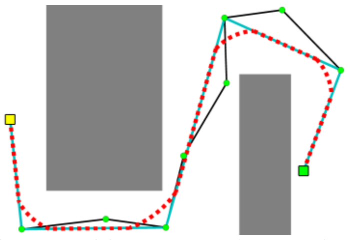

Figure 6. The performance of the post processing procedure for a given path. The original path between yellow and green squares is highly suboptimal (black line). Redundant nodes are removed and the rest are connected to provide a shortcut path (blue line). A smoothing technique is then employed to smooth the sharp edges of the final solution (red dotted line).

Next, the result of the algorithm after post-processing and path smoothing should be transferred to the robot. The final solution consists of three vectors, including FS, which stores the nodes on the final path, CU containing the curvature information when the robot is moving on a curve and finally D, which contains the travelled distance between any two consecutive notes of the final solution. These three vectors will be used later to compute the control vector of the robot, $Control$, which includes segmental linear (v_i) and angular (ω_i) velocity of the robot as well as the during of each segment (t_i).

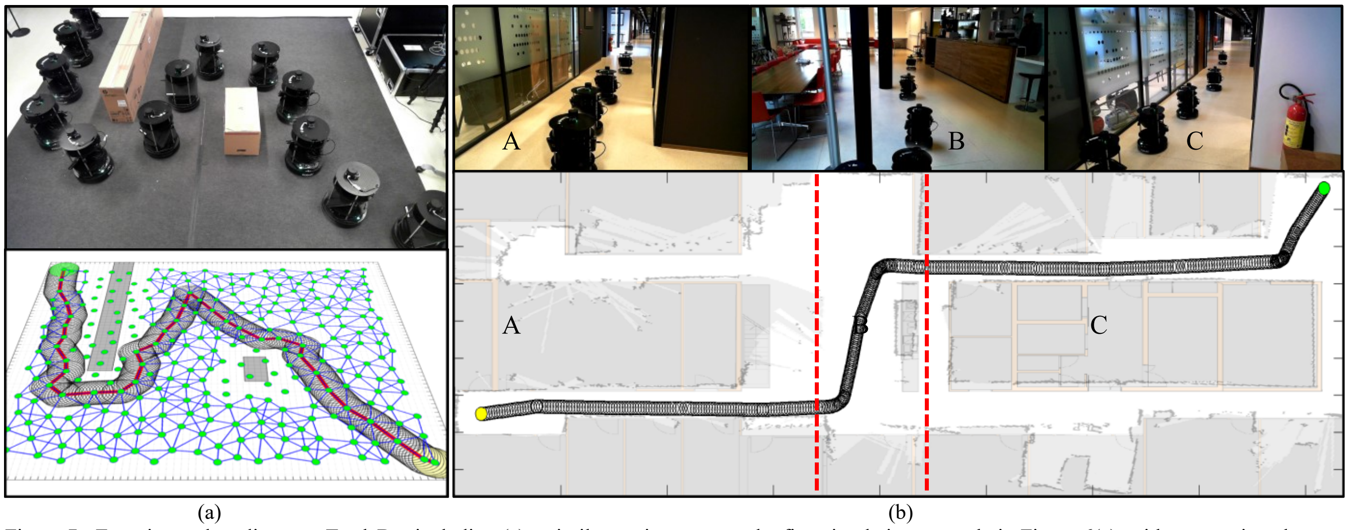

Figure 7. Experimental studies on a TurtleBot including (a) a similar environment to the first simulation example in Figure 6(a), with two static unknown obstacles where the robot only knows the boundaries of the environment and (b) the TurtleBot is moving in an office-like environment when the only available map is a noisy map generated by gmapping algorithm.

$$FS = \begin{bmatrix} x_1 & \dots & x_m \\ y_1 & \dots & x_m \end{bmatrix}, \; m = 2 \times ||path|| - 2$$

$$CU = \begin{bmatrix} r_1 & \dots & r_{m-1} \\ \alpha_1 & \dots & d_{m-1} \end{bmatrix}, \alpha_i \in \{-1, 0, +1\}$$

$$D = \begin{bmatrix} d_1 & \dots & d_{m-1} \\ \theta_1 & \dots & \theta_{m-1} \end{bmatrix}$$

$$Control = \begin{bmatrix} v_1 & \dots & v_{m-1} \\ \omega_1 & \dots & \omega_{m-1} \\ t_1 & \dots & t_{m-1} \end{bmatrix},$$

$$\omega_i = \frac{\alpha_i v_i}{r_i}, \quad t_i = \frac{\alpha_i \theta_i}{\omega_i} + \frac{(1-\alpha_i) d_i}{v_i}$$

where α_i shows the turning direction. The robot goes straight if $\alpha_i = 0$, turns right if $\alpha_i = +1$ and turns left if $\alpha_i = -1$. At the beginning, the robot is given an initial control vector based on the solution found by the original roadmap. As the robot starts to move, it scans the surrounding area on a fixed predefined frequency $\Delta t = 2$ (sec) which means if the robot is moving on a straight line with the linear speed of v_i, then is scans the surrounding area every $d = 2 \times v_i$. The sensing range of the robot was limited to one meter.

C. *Experimental Studies*

The performance of the algorithm was tested on a Turtlebot2 with an Asus Xtion Pro Live camera, an A1 RPLIDAR 360° laser range finder, and an onboard computer with a 2.60-GHz Intel Core i5 processor with 8 GB of memory in two different planning problems as shown in Figure 7. And Table 2. First, the robot is navigating in a 2D plain environment where two unknown static obstacle appear after the initial planning. This problem is similar to the first simulation scenario in Figure 6(a). Second, the robot is moving in an office with a highly inaccurate map and in the presence of unknown static and dynamic obstacles. During the experiments, the linear speed of the robot was set to be $0.2(m/sec)$ and the angular velocity was calculated accordingly. The definitions of PL, RT Fail and DM is same as described before. Even though the initial placement of the robot is important for successful implementation, but the planner could adopt to minor errors in the initial pose of the robot.

TABLE 2. EXPERIMENTAL RESULTS IN TWO DIFFERENT SCENARIOS.

n **=**	**50**	**100**	**500**	**1000**
		Experiment 1		
PL(m)/Std	12.13 / 0.18	12.08 / 0.13	11.62 / 0.12	11.94 / 0.07
RT(sec)/Std	0.27 / 0.08	0.38 / 0.08	0.48 / 0.11	0.63 / 0.14
Fail(%)	5	0	0	0
DM(m)/Std	0.23 / 0.13	0.24 / 0.12	0.21 / 0.08	0.20 / 0.09
		Experiment 2		
PL(m)/Std	49.67 / 2.85	48.40 / 2.97	46.18 / 2.65	43.19 / 2.98
RT(sec)/Std	1.34 / 0.08	2.78 / 0.08	4.25 / 0.17	7.25 / 0.15
Fail(%)	53	17	5	3
DM(m)/Std	0.38 / 0.10	0.37 / 0.13	0.38 / 0.07	0.35 / 0.11

Since the postprocessing and smoothing steps were implemented on the planner, one extra rule had to be added to the navigation. Whenever the environment scan resulted in graph adaptation and the solution path was repaired, an additional smoothing step takes place to prevent the robot from completely stopping and changing the orientation. Instead, the robot moves on a curve in order to follow the new path. Furthermore, Figure 8 shows the changes in the failure rates of the planner relative to the initial graph size. Having a too small graph leads to failure but as soon as few samples are added, the planner performs effectively. It also shows that after certain values, the size of the graph becomes affectless on the success or failure of the planner. The stability of the results presented in Figure 9 indicates that despite the randomized nature of the planner, it generates stable results with low variation over different runs. The stability of the results is because of the sampling-radius and the graph adjustment behavior. Since the samples are evenly distributed in the space, the resulted solution and corresponding processing time and distance to the obstacles change with lower variances. On the other hand, the graph adopts to the recent changes without adding new samples to the graph and this procedure keeps the appearance and behavior of the original graph. The failure rate was not included in stability analysis since the percentage of failure was averaged over 100 iterations.

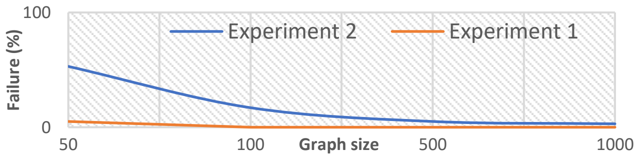

Figure 8. Performance of the proposed algorithm in terms of failure rate in during the experimental studies. The results are averaged over 100 different runs of the planner. The size of the graph plays an important role in the success rate of the planner.

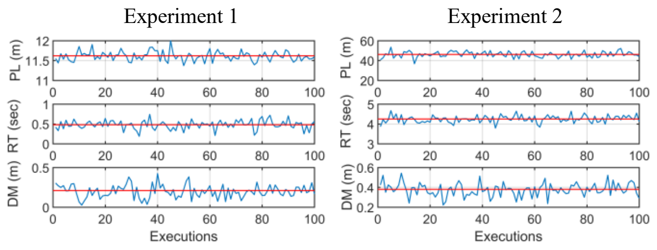

Figure 9. The performance of the proposed algorithm in generating stable results. For both experiments, the variations in path length, processing time and minimum distance to the obstacles are shown for 100 different runs.

VI. Conclusion

A multi-query planner was proposed to deal with uncertainty challenge in robotic motion planning. The proposed algorithm employs two new mechanism to deal with unknown changes. First, a sample classification component takes place parallel to the sampling procedure, which stores the generated samples in a grid-based matrix. This makes it computationally free to look for samples in any specific region of the configuration space during the planning. Since it requires only a simple calculation, it does not affect the overall processing time of the planner. Next, a graph adjustment procedure takes place during the execution of the initial solution to adapt the to the problem uncertainty. This mechanism detects the sensible grid cells and the corresponding nodes by means of a moderate cell recognition strategy that prevents too optimistic or too pessimistic cell recognition. Then the selected nodes are checked for collision and if they are in collision, the corresponding edges from current node to those will be removed. Furthermore, A second layer of nodes around the current node will be checked and in-collision edges are disconnected to reduce the response time of the planner to uncertainty in the planning. Several simulation and experimental tests have been conducted which show the efficient performance of the proposed planner in producing semi-optimal solutions with low computational cost and insignificant failure rates even when working with a small graph. The presented work could be further investigated for more complex problems when even the boundaries of the environment Is not known to limit the sampling domain.

Acknowledgment

This work is supported by the Research Council of Norway as a part of the Multimodal Elderly Care Systems (MECS) project, under grant agreement 247697.

References

[1] 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*, 12(4), 566-580, 1966.

[2] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” Dept. Comput. Sci., Iowa State Univ., Ames, IA, USA, Tech. Rep. TR 98-11, 1998.

[3] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, “Randomized kinodynamic motion planning with moving obstacles,” *The International Journal of Robotics Research*, 21(3), 233-255, 2002.

[4] S. Karaman, and E. Frazzoli, «Sampling-based algorithms for optimal motion planning,” *The international journal of robotics research*, 30(7), 846-894, 2011.

[5] H. Kurniawati, T. Bandyopadhyay, and N. M. Patrikalakis, “Global motion planning under uncertain motion, sensing, and environment map,” *Autonomous Robots*, 33(3), 255-272, 2012.

[6] H. M. Choset, *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[7] M. Elbanhawi, and M. Simic, “Sampling-based robot motion planning: A review,” *IEEE Access*, 2, 56-77, 2014.

[8] L. Jaillet, J, Hoffman, J. Van den Berg, P. Abbeel, J. M. Porta, and K. Goldberg, “EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles,” in *IROS* 2011, 2646-2652.

[9] K. Belghith, F. Kabanza, and L. Hartman, “Randomized path planning with preferences in highly complex dynamic environments,” *Robotica*, 31(8), 1195-1208, 2013.

[10] A. Bry, and N. Roy, “Rapidly-exploring random belief trees for motion planning under uncertainty,” in *ICRA* 2011, 723-730.

[11] M. W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, “Path planning for motion dependent state estimation on micro aerial vehicles,” in *ICRA* 2013, 3926-3932.

[12] P. Leven, and S. Hutchinson, “A framework for real-time path planning in changing environments,” *The International Journal of Robotics Research*, 21(12), 999-1030, 2002.

[13] L. Jaillet, and T. Siméon, “A PRM-based motion planner for dynamically changing environments,” in *IROS* 2004,1606-1611.

[14] J. P. van den Berg, D. Nieuwenhuisen, L. Jaillet, and M. H. Overmars, “Creating robust roadmaps for motion planning in changing environments,” in *IROS* 2005, 1053-1059.

[15] L. Jaillet, and T. Siméon, “Path deformation roadmaps: Compact graphs with useful cycles for motion planning,” *The International Journal of Robotics Research*, 27(11-12), 1175-1188, 2008.

[16] B. Burns, and O. Brock, “Sampling-based motion planning with sensing uncertainty,” in *ICRA* 2007, 3313-3318.

[17] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, “Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns,” *Autonomous Robots*, 35(1), 51-76, 2013.

[18] B. D. Luders, and J. P. How, “An optimizing sampling-based motion planner with guaranteed robustness to bounded uncertainty,” in *ACC* 2014, 771-777.

[19] A. A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, “FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements,” *The International Journal of Robotics Research*, 33(2), 268-304, 2014.

[20] W. Liu, and M. H. Ang, “Incremental sampling-based algorithm for risk-aware planning under motion uncertainty,” in *ICRA* 2014, 2051-2058.

[21] D. Li, Q. Li, N. Cheng, and J. Song, “Sampling-based real-time motion planning under state uncertainty for autonomous micro-aerial vehicles in GPS-denied environments,” *Sensors*, 14(11), 21791-21825, 2014.

[22] W. Sun, S. Patil, and R. Alterovitz, “High-frequency replanning under uncertainty using parallel sampling-based motion planning,” *IEEE Transactions on Robotics*, 31(1), 104-116, 2015.

[23] M. Otte, and E. Frazzoli, “RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning,” *The International Journal of Robotics Research*, 35(7), 797-822, 2016.

[24] V. Pilania, and K. Gupta, “Localization aware sampling and connection strategies for incremental motion planning under uncertainty,” *Autonomous Robots*, 41(1), 111-132, 2017.

[25] W. Khaksar, T. S. Hong, M. Khaksar, and O. Motlagh, “A low dispersion probabilistic roadmaps (LD-PRM) algorithm for fast and efficient sampling-based motion planning,” *International Journal of Advanced Robotic Systems*, 10(11), 397, 2013.

[26] R. Luna, I. A. Şucan, M. Moll, and L. E. Kavraki, “Anytime solution optimization for sampling-based motion planning,” in *ICRA* 2013, 5068-5074.