*Int. J. Appl. Math. Comput. Sci., 2019, Vol. 29, No. 4, 641–654*
DOI: 10.2478/amcs-2019-0047

amcs

# MULTIQUERY MOTION PLANNING IN UNCERTAIN SPACES: INCREMENTAL ADAPTIVE RANDOMIZED ROADMAPS

WERIA KHAKSAR [a,*], MD ZIA UDDIN [a], JIM TORRESEN [a]

[a]Robotics and Intelligent Systems Group (ROBIN), Department of Informatics
University of Oslo, Ole Johan Dahls hus, Gaustadallen 23 B, N-0373 Oslo, Norway
e-mail: {weriak,mdzu,jimtoer}@ifi.uio.no

Sampling-based motion planning is a powerful tool in solving the motion planning problem for a variety of different robotic platforms. As its application domains 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 its 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 an 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 resulting 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 for problems with various sources of uncertainty show a significant improvement compared with similar algorithms in terms of the failure rate, the processing time and the minimum distance from obstacles. The planner is also successfully implemented and tested on a TurtleBot in four different scenarios with uncertainty.

**Keywords:** motion planning, uncertainty, roadmaps, sampling, obstacle avoidance.

## 1. Introduction

An essential component for any robotic system is the ability of the robot to move inside its environment (Jafarzadeh and Fleming, 2018). Such an ability is provided using a motion planning procedure. Motion planning can be defined as moving a mobile robot between a pair of start and goal configurations in an environment filled with obstacles, while avoiding any collision with obstacles and the environments boundaries (Kingston *et al*., 2018; Klaučo *et al*., 2016). It has been proven that the motion planning problem in its simplest form is NP-complete (Canny, 1988; González *et al*., 2016), meaning that the running time of the algorithm is exponential in the degree of freedom which makes the motion planning problem a challenging research area. Based on the increasing application of motion planning in various areas such as automation, animating digital characters, computer games, architectural design, assembly planning, robotic surgery, and computational biology, there have been an extensive amount of research in the field of motion planning algorithms, resulting in various algorithms with different characteristics, advantages and drawbacks (Choset *et al*., 2005; Khaksar *et al*., 2012; Elbanhawi and Simic, 2014; Przybylski and Putz, 2017).

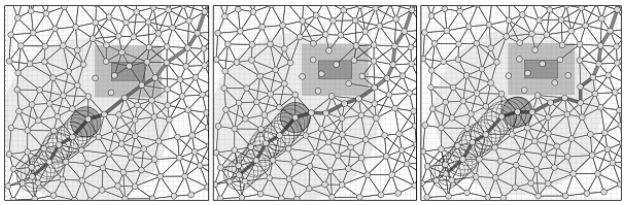
In the field of motion planning, sampling-based planners have been successfully applied to solve difficult problems in high-dimensional spaces (Ha *et al*., 2018). These algorithms are unique in the fact that planning occurs by sampling the configuration space. Original sampling-based planners such as probabilistic roadmaps (PRMs) (Kavraki *et al*., 1996), rapidly-exploring random trees (RRTs) (LaValle and Kuffner, 2001), and expansive space trees (ESTs) (Hsu *et al*., 2002), are proved to be probabilistically complete as the probability of finding a

*Corresponding author

solution in these planners is one when the input size goes to infinity. These algorithms have been improved further to achieve some forms of optimality in the generated solutions. Optimal sampling-based planners such as PRM* and RRT* (Karaman and Frazzoli, 2011) are asymptotically optimal as the solutions found by these algorithms converge asymptotically to the optimum, if one exists, with the probability of 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 robots 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 (Kurniawati *et al*., 2012).

Despite the proven advantages of sampling-based algorithms in path planning and even in other fields such as computer games and drug design (Choset *et al*., 2005), 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 to prevent them from happening. 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 the randomized algorithms, a local planner should be available to detect possible collision-free connection between any 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 errors, uncertain environment maps 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 compared with their counterparts that do not consider uncertainty. Furthermore, focusing on one aspect of uncertainty normally requires deterministic knowledge on other aspects. For instance, having an efficient path planner to deal with dynamic obstacles requires a very accurate sensory system. In other words, the cumulative effect of all sources of uncertainty can be difficult to model in the planning phase before task execution.



(a) (b) (c)

Fig. 1. Part of the solution computed by the proposed planner in a simple 2D environment with a static unknown obstacle: the robot is following the preplanned path (a), as the robot senses the new obstacle, the graph is adjusted and the path is repaired accordingly (b), as the robot keeps moving, the graph keeps being adjusted and the generated path is improved during the planning (c). The obstacle and its expanded version are shown with dark and light squares, respectively. The robot is the dark circle with the light circle around it as the vision range, and the generated path is the thick line.

In this paper, an extension of the optimal probabilistic roadmaps (PRM*) (Karaman and Frazzoli, 2011) is proposed which is able to handle different types of planning uncertainty in a single package without a considerable increase in the computational cost. The algorithm is based on an adaptive randomization concept in which the entire randomized structure obtained from the traditional sampling mechanism is incrementally and frequently updated to handle any type of uncertainty in the planning space and the robot's sensory system. 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, $\Delta t$, which directly updates itself based on the information provided by the onboard sensor(s).

The most important part of the proposed algorithm is a graph-adjustment component which adjusts the resulting 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 in the same way since regardless of the source of uncertainty, it results in encountering an obstacle when it was not 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 nonholonomic mobile robot moving in a 2D indoor environment. This implementation is tested on a TurtleBot in four different path planning scenarios with uncertainty.

The remainder of this paper is organized as follows: Section 2 is a summary of the related literature. Section 3 contains formal definitions and the notation regarding the problem while Section 4 describes the proposed algorithm. The simulation and experimental results are provided in Section 5 and, finally, the work is concluded and potential future work is discussed in Section 6.

## 2. Related work

Sampling based planning is by no means a novel concept in robotics (Janson *et al*., 2018). It was proposed to overcome the complexity of deterministic robot planning algorithms for a robot with six degrees of freedom. The use of random computations to solve otherwise rather difficult problems have been immensely successful.

A common restrictive assumption in sampling-based algorithms is that the environment is well defined such that the relative location of the robot with respect 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 (Elbanhawi and Simic, 2014). In the past years, sampling-based algorithms have been updated to deal with various sources of uncertainty. Based on a single or a 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 tuning appropriate parameters and various heuristics in different instances.

There are several extensions of the RRT algorithm to deal with uncertainty (Jaillet *et al*., 2011; Belghith *et al*., 2013; Bry and Roy, 2011; Achtelik *et al*., 2013) 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 the PRM planner have been introduced to deal with uncertainty. A PRM was proposed for dynamic motion planning based on regenerating a roadmap on the assumption of an obstacle-free space (Leven and Hutchinson, 2011) while the data structure of the PRM was improved to accommodate changes in the environment and consequently, in the roadmap. However, this algorithm only handles dynamic environments.

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 (Jaillet and Simeon, 2004). A generalized PRM was introduced in surroundings where obstacle movements are restricted to local sectors (Choset *et al*., 2005). The PDR maintains a roadmap whose paths can be deformed. Thus, numerous paths can be obtained between two configurations (Jaillet and Simon, 2008).

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 (Burns and Brock, 2007). 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 (Kurniawati *et al*., 2012). 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 among dynamic obstacles with uncertain motion patterns (Aoude *et al*., 2013). This method builds a learned motion pattern model by combining the flexibility of the Gaussian process with the efficiency of the RRT planner.

BU-RRT* (Luders and How, 2014) is a novel optimizing sampling-based motion planner that guarantees the feasibility of linear systems subject to a bounded uncertainty. FIRM (Agha-Mohammadi, 2014) 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 this preserves the optimal substructure property. FIRM also relies on a 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 (Liu and Ang, 2014) 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 the RRT* framework to compute risk-aware and asymptotically optimal trajectories under motion uncertainty.

A sampling-based real-time motion planning algorithm was proposed (Li *et al*., 2014) for planning under state uncertainty which is an extension of the closed-loop rapid belief tree. An RRT-based planner (HFR) was reported that is able to perform high-frequency replanning under uncertainty using

parallel sampling-based planners (Sun *et al*., 2015). RRT$^{\text{X}}$ (Otte and Frazzoli, 2016) 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 (Pilania and Gupta, 2017) 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 are able to achieve higher uncertainty reduction while maintaining adequate samples in regions where uncertainty reduction is poor. A holistic approach for 3D object reconstruction with a mobile manipulator robot with an eye-in-hand sensor has been proposed (Vasquez-Gomez *et al*., 2017) considering uncertainty in both observation and control. In this work, a set of candidate views/states is directly generated in the state space, and later only a subset of these views is kept by filtering the original set.

Few advances in the concept of belief roadmap search have been proposed (Shan and Englot, 2017) to plan under localization uncertainty based on a best-first strategy to improve the computational cost of the search. A sampling-based mobile manipulator planner has been introduced (Pilania and Gupta, 2018) that considers the base pose uncertainty and the effects of this uncertainty on manipulator motions by having a hierarchical planner that plans for both the base and the arm in a judicious manner, a localization-aware sampling and connection strategies, and also by incorporating base pose uncertainty along the edges. A distributionally robust incremental sampling-based method has been introduced (Summers, 2018) for kinodynamic motion planning under uncertainty based on the RRT planner. In this method, unlike many approaches that assume Gaussian distributions for uncertain parameters, moment-based ambiguity sets of distributions with given mean and covariance are considered.

Finally, a framework has been proposed (Axelrod *et al*., 2018) to compute shadows as the geometric equivalent of a confidence interval around observed geometric objects which introduces tighter bounds than those of previous methods and the tightness of the bounds does not depend on the number of obstacles by relying on computing a bound specific to a trajectory instead of trying to identify a generic safe set.

Most of the above mentioned 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, the 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 a mobile robot is considered that stops for a minute each time and senses a new obstacle.

## 3. Problem formulation

In this section, basic definitions and descriptions of the motion planning problem and the proposed algorithm are provided. A mobile robot is moving in a $d$-dimensional state space. Initially, there is a map of the environment that only specifies the space boundaries. The only requirement of the planner is to have or to be able to generate an initial solution before the navigation starts. In other words, initially the robot follows the direct path towards the goal position until it encounters an obstacle.

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

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

$$(x_i, y_i)^T \in Q_{\text{free}}, \quad i = 1, \ldots, n, \tag{1}$$

$$||(x_i, y_i)^T, (x_j, y_j)^T|| \leq R_S(n), \quad \forall (x_j, y_j)^T \in S, \tag{2}$$

$$R_S(n) = [L(Q_{\text{free}}(n - \lambda)/\pi n^2]^{1/2}, \tag{3}$$

$$S = \begin{bmatrix} x_1 & \ldots & x_n \\ y_1 & \ldots & y_n \end{bmatrix}, \tag{4}$$

where $||A, B||$ denotes the Euclidean distance between two points $A$ and $B$ and $R_S(n)$ is the sampling radius based on the number of samples, $n$. $L$ is the Lebesgue measure (i.e., the volume) and $\lambda$ is a positive scaling constant. More details about the sampling radius can be found in the work of Khaksar *et al*. (2013).

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

$$\sigma(\tau) \in Q_{\text{free}}, \quad \forall \tau \in [0, 1], \tag{5}$$

$$\sigma(\tau) = (x_i, y_i)^T, \quad \sigma(1) = (x_j, y_j)^T. \tag{6}$$

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

path, $\sigma_D$, between $(x_i, y_i)^T$ and $(x_j, y_j)^T$.

**Definition 3.** Let $CON_{n\times n}$ be a matrix defining the *connection graph* between all $(x_i, y_i)^T$ and $(x_j, y_j)^T \in S$. Any two pairs of samples in $S$ are connected if there exist a direct collision-free direct path between them and the length of this path is less than a given connection radius $R_c(n)$. Specifically,

$$CON_{n\times n} = [con_{i,j}], \quad i, j = 1, \ldots, n, \tag{7}$$

$$con_{i,j} = \begin{cases} |\sigma_D(i,j)||, , & \sigma_D(i,j) \neq \emptyset \\ & \wedge \quad ||\sigma_D|| \leq R_c(n), \\ 0, & \text{otherwise}, \end{cases} \tag{8}$$

$$R_c(n) = \gamma \Big[\frac{\log(n)}{n}\Big]^{1/n}, \tag{9}$$

$$\gamma > \gamma^* = 2(1 + 1/d)^{\frac{1}{d}} \Big[L\Big(\frac{Q_{\text{free}}}{\zeta_d}\Big)\Big]^{1/d}, \tag{10}$$

$$S = \begin{bmatrix} con_{1,1} & \ldots & con_{1,n} \\ \ldots & con_{i,j} & \ldots \\ con_{n,1} & \ldots & con_{n,n} \end{bmatrix}, \tag{11}$$

where $d$ is the dimension of the configuration space, $\zeta_d$ is the volume of the unit ball in the $d$-dimensional Euclidean space and $L(Q_{\text{free}})$ denotes the Lebesgue measure (i.e., the volume) of the obstacle-free space.

The concept of connection radius was taken from the PRM* algorithm (Karaman and Frazzoli, 2011) to guarantee asymptotically optimal solutions.

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

**Definition 5.** Let $COL_{n\times n}(t)$ be a matrix that represents the *uncertainty* in the planning problem as a function of time. This uncertainty could be from unknown obstacles, noisy sensing devices, or dynamic obstacles. We have

$$COL_{n\times n}(t) = [col_{i,j}(t)], \quad i, j = 1, \ldots, n, \tag{12}$$

$$col_{i,j}(t) = \begin{cases} 0 & \text{in time } (t) \to \sigma_D(i,j) \neq \emptyset, \\ 1 & \text{otherwise}, \end{cases} \tag{13}$$

$$COL(t) = \begin{bmatrix} col_{1,1}(t) & \ldots & col_{1,n}(t) \\ \ldots & col_{i,j}(t) & \ldots \\ col_{n,1}(t) & \ldots & col_{n,n}(t) \end{bmatrix}. \tag{14}$$

The matrix of uncertainty shows whether or not, in a specific time $t$, a given configuration $(x_i, y_i)^T$ is in collision with obstacles by the value of $col_{i,i}(t)$. It also shows if in time $t$, there is a direct path $\sigma_D(i,j)$ between any two configurations $(x_i, y_i)^T$ and $(x_j, y_j)^T$, by the value of $col_{i,j}(t)$. This matrix will be updated continuously during the navigation by analyzing the readings of the robots sensory system. Any noise in the robot's sensing devices can be formulated using this matrix as it gets updated frequently and imprecise sensory readings are not able to cause the planning to fail. the main contribution of the proposed structure to sensing uncertainty is how the algorithm looks at the sensory information and, more importantly, how the planner records the information.

## 4. Proposed algorithm

In this section, the proposed algorithm is presented in detail which includes the graph construction and graph adjustment. The flow chart of the proposed algorithm is presented in Fig. 2.

In the flow chart of Fig. 2, the "low dispersion sampling" generates the collision/free samples inside the configuration space while the "connection matrix" connects the samples considering the sampling radius introduced in the previous section. "Uncertainty update" checks the current visible samples for collision duo to uncertainty, and "find shortest path" uses the A algorithm for realizing the shortest path from the start to the goal position. "Remove in-collision nodes" updates the graph by discarding the colliding nodes, and finally, the "add collision-free nodes" updates the graph by adding previously removed nodes.

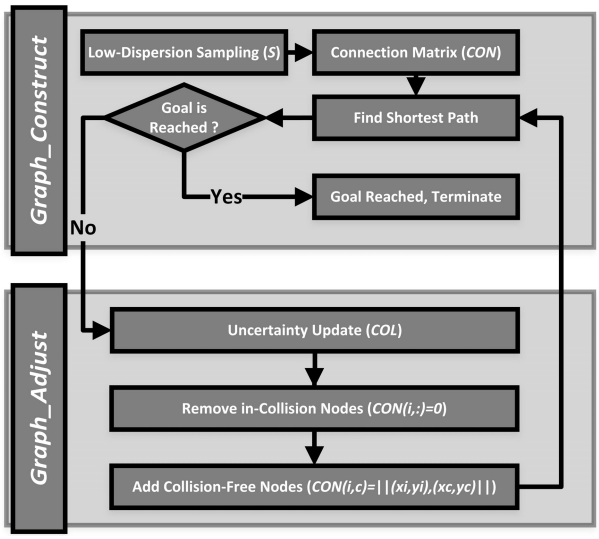


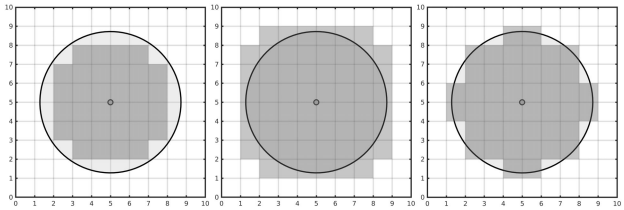

Fig. 2. Flow chart of the proposed algorithm including the $Graph\_Construct$ and $Graph\_Adjust$ procedures.

Like any multi-query planner, the graph construction phase starts by learning the configuration space through sampling. Initially, it requires an approximation of the space boundaries. Having any additional information is optional and does not affect the planner performance. According to the initial available map, the sampling takes place and the set of all samples $S = \{(x_i, y_i)^T, i = 1, \ldots, n\}$ 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 $GR$, $(\alpha, \beta)$ can be calculated as follows:

$$\alpha = \lceil x_i/\Delta_{GR} \rceil, \quad \beta = \lceil y_i/\Delta_{GR} \rceil \tag{15}$$

where $\lceil \alpha \rceil$ means the smallest positive integer, which is greater than or equal to $\alpha$. Having a grid resolution smaller than the sampling radius $R_S(n)$ guarantees that any given cell in the grid matrix includes at most one sample. Using this simple structure makes it computationally cheap to search the visible area around the robot and find the neighbor nodes without searching the whole graph. In the current position of the robot, the surrounding grid cells are considered as visible if the cell center is within the sensing range. This strategy provides enough visible grid cells without being pessimistic or optimistic as presented in Fig. 3.

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.



(a) (b) (c)

Fig. 3. Different strategies for recognizing a grid cell as visible: a pessimistic strategy that accepts a cell if the entire cell is within the sensing range (a), an optimistic strategy that accepts a cell if it is partially visible (b), and the proposed strategy that recognizes a cell if the center of the cell is visible (c). Unrecognized cells are shown in white.

**Algorithm 1.** $Graph\_Construct$.

**Require:** $S \leftarrow \{(x_i, y_i)^T\}, \quad GR \leftarrow \{(\alpha_i, \beta_i)^T\}$
**Require:** $CON \leftarrow \{con_{i,j}\}, \quad i, j = 1, \ldots, n$
**while** $reach = false$ and $fail = false$ **do**
  $time = time + \Delta t$
  Scan: $\rho(\theta, time)$
  Update: $COL_{n \times n}$
  $Graph\_Adjust$
  $Graph\_Shortest\_Path \leftarrow [dist, path]$
  $Sparse\{(CON), (x_c, y_c)^T, (x_f, y_f)^T\}$
  **if** $||(x_c, y_c)^T, (x_f, y_f)^T|| \leq \epsilon$ **then**
    $reach \leftarrow true$
  **end if**
  **if** $path = \emptyset$ **then**
    $fail \leftarrow true$,
    **return**
  **end if**
  $Move \leftarrow d = V \times \Delta t$
**end while**

In the next phase of the algorithm, as the robot starts to move, the surrounding area is scanned and visible grid cells, as shown in Fig. 3(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 values of the corresponding nodes in the uncertainty matrix will be updated. If a node was defined as free before but now, the corresponding grid cell to that node is not reachable, i.e., occupied, the status of that node will be updated to occupied, $col_{i,i}(t) = 1$. On the other hand, if a node was marked as occupied before and now it is reachable, its 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 Fig. 4, where the graph adjustment is shown in two different positions and some vertices are removed or added back to the roadmap.

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 the 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 $col_{i,i}(t)$ has been updated to 1, then all the connections of that node will be removed in the graph connection matrix, $CON(i, :) = 0$, and $CON(:, i) = 0$. The opposite procedure applies to the nodes that have been disconnected before and now have a collision-free connection to the current node. Next, for all other nodes within the sensing range and connected to

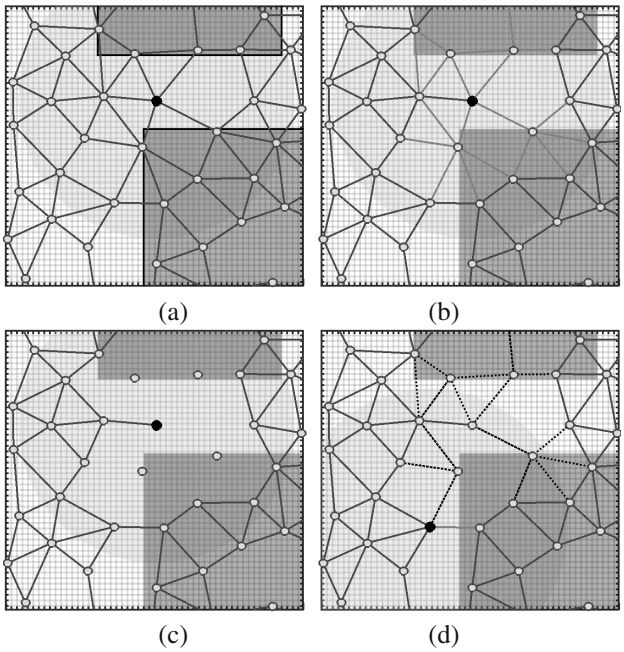


Fig. 4. $Graph\_Adjust$ procedure: the original graph before sensing the obstacle (a), the obstacles are detected and in-collision edges are determined (b), the adjusted graph after removing the in-collision edges (c), and after the robot moves to another position with a different sensing outcome, all of the previously removed edges are added back to the graph if the corresponding uncertainty values are not zero (d). The black point is the robot's current position and the light gray circle represents the robot's vision range.

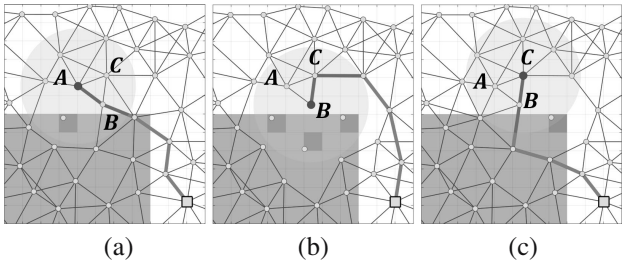



Fig. 5. Local minimum trap where the robot moves between two local optima forever: the robot is at point A and the current shortest path goes through point B (a), when the robot reaches point B, the graph is adjusted and now the path goes through point C (b), as the robot reaches point C, some parts 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 B (c). This loop continues forever.

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.

**Algorithm 2.** $Graph\_Adjust$.

**Require:** $COL_{n\times n}(t) = [col_{i,j}(t)], \quad i,j = 1,\ldots,n$

1: **for all** $(x_i, y_i)^T \in visible\_range$ **do**
2: **if** $col_{i,c} = 1$ and $con_{i,c} \neq 0$ **then** $CON(i,:) = 0, \quad CON(:,i) = 0$
3: **end if**
4: **if** $col_{i,c} = 0$ and $con_{i,c} \neq 0$ **then**
5: **for all** $(x_i, y_i)^T \in visible\_range$ **do**
6: **if** $col_{i,j}(t) = 0$ $CON(i,j) = 0, \quad CON(j,i) = 0$ **then**
7: **end if**
8: **end for**
9: **end if**
10: **if** $col_{i,c} = 0$ and $con_{i,c} = 0$ **then**
11: $CON(i,c) = ||(x_i, y_i)^T, (x_c, y_c)^T||$
12: $CON(c,i) = ||(x_i, y_i)^T, (x_c, y_c)^T||$
13: **for all** $(x_i, y_i)^T \in visible\_range$ **do**
14: **if** $col_{i,j}(t) = 0$ and $con_{i,j} \neq 0$ **then**
15: $CON(i,j) = ||(x_i, y_i)^T, (x_j, y_j)^T||$
16: $CON(j,i) = ||(x_i, y_i)^T, (x_j, y_j)^T||$
17: **end if**
18: **end for**
19: **end if**
20: **end for**

Applying the $Graph\_Adjust$ procedure has another benefit that improves the planning efficiency. According to lines 13–17 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 any more may cause a local minimum in which the robot keeps moving between two positions forever. Figure 5 shows an example of the situation with local minima. Limiting the graph adaptation to the visible region avoids local minima. If the graph is adjusted, the shortest path from the robots current position to the final configuration will be calculated and the robot continues moving but along the latest generated path. This procedure repeats with a constant frequency $\Delta t$ [sec.] until the robot reaches the obstacle or concludes that no solution exists. As presented in Fig. 3, the proposed planner is capable of disconnecting the in-collision nodes and reconnecting free nodes.

## 5. Results and a 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.

**5.1. Post-processing.** To make the results of the proposed algorithm suitable for a real robot, few modifications are required. Since one of the major drawbacks of sampling-based algorithms are their widely regarded suboptimal paths, we applied a postprocessing procedure (Luna *et al.*, 2013) to the results of the algorithm which can remove the redundant nodes from the final solution. From a given path, a shorter path could be obtained by checking whether nonadjacent configurations $(x_i, y_i)^T$ and $(x_j, y_j)^T$ along the path can be connected with the local planner. The points $(x_i, y_i)^T$ and $(x_j, y_j)^T$ could be chosen randomly. Another alternative would be a greedy approach. Starting from $(x_{\text{init}}, y_{\text{init}})^T$, the planner tries to connect directly to the final configuration $(x_f, y_f)^T$. If this step fails, it starts from the configuration after $(x_{\text{init}}, y_{\text{init}})^T$ and tries again. This process repeats until a connection can be made to $(x_f, y_f)^T$, say, from the point $(x_1, y_1)^T$. Now the planner sets the target to $(x_1, y_1)^T$ and begins again, trying to connect from $(x_{\text{init}}, y_{\text{init}})^T$ to $(x_1, y_1)^T$, and repeats the procedure. This procedure can also be applied in the opposite direction. Figure 6 illustrates this greedy approach in the forward direction to shorten a path in a two-dimensional Euclidean space.

Next, a path smoothing technique was applied to refine the resulting paths by finding the inner circle of each three consecutive nodes on the postprocessed path as presented in Fig. 7. The proposed path smoothing technique works by receiving a set of three consecutive configurations in the final postprocessed path and calculating the center coordinate of the inner circle accordingly,

$$x_c = \frac{l_i \times x_i + l_{i+1} \times x_{i+1} + l_{i+2} \times x_{i+2}}{l_i + l_{i+1} + l_{i+2}}, \quad (16)$$

$$y_c = \frac{l_i \times y_i + l_{i+1} \times y_{i+1} + l_{i+2} \times y_{i+2}}{l_i + l_{i+1} + l_{i+2}}, \quad (17)$$

$$r^2 = \frac{(l_i + l_{i+1}) \times (l_{i+1} + l_{i+2}) \times (l_i + l_{i+2})}{l_i + l_{i+1} + l_{i+2}}, \quad (18)$$

where $(x_i, y_i)^T$, $(x_{i+1}, y_{i+1})^T$, and $(x_{i+2}, y_{i+2})^T$ are the vertices of the triangle and $l_i$, $l_{i+1}$, and $l_{i+2}$ are the lengths of the opposite sides, respectively. The overall performance of the postprocessing method is illustrated in Fig. 8 where the original solution, the shortened path, and the final smoothed path are shown for a given planning problem.

**5.2. Simulation studies.** The planner was simulated in Matlab R2018b to perform in four different planning scenarios as presented in Fig. 9. All simulations were run on a desktop with a 3.40-GHz Intel Core i7 processor with 32 GB of memory.

In the first problem as in Fig. 8(a), the environment includes four simple obstacles in a symmetric

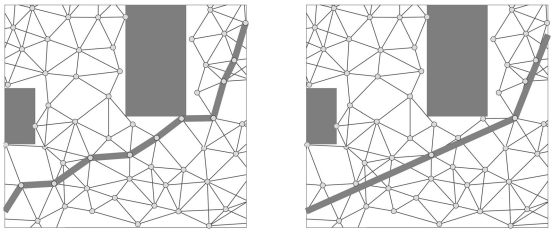



Fig. 6. Implementation of the greedy forward post-processing approach for removing the redundant segments of the generated solution. The original (a) and shortened (b) paths after the postprocessing.

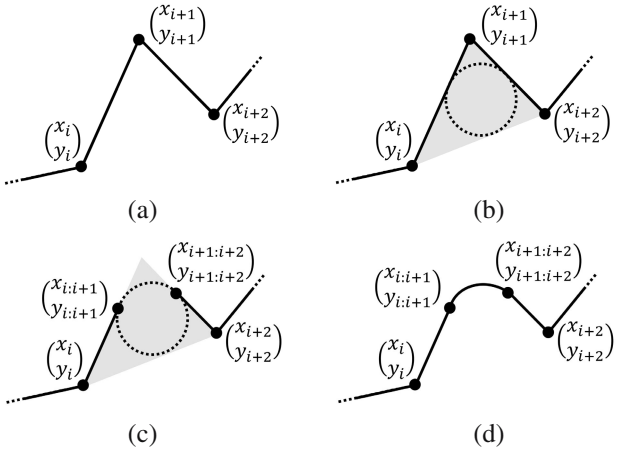



Fig. 7. Performance of the proposed path smoothing method: the original path (a), finding the inner circle (b), removing the sharp edge (c), and the final smooth path (d).

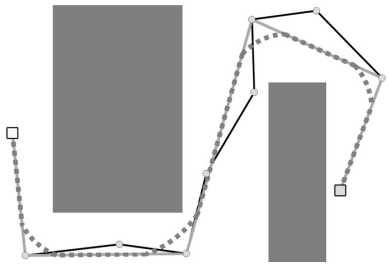


Fig. 8. Performance of the postprocessing and path smoothing procedures for a given path. The original, shortened, and smoothed paths are represented by thin, thick, and dashed lines, respectively.

Table 1. Performance comparison based on the simulation results with 500 nodes in all initial graphs.

| Parameter | Scene | GCS | RR-GP | BU-RRT* | FIRM | CC-RRT* | RRBT-LAS | Proposed |
|---|---|---|---|---|---|---|---|---|
| PL [std] (meter) | 1 | 72.1 [4.8] | 71.2 [5.6] | **64.1** [3.5] | 68.1 [6.1] | 65.5 [4.4] | 72.1 [6.2] | **64.1 [2.5]** |
| | 2 | 80.5 [6.4] | 81.9 [7.2] | 84.4 [8.1] | 80.5 [9.2] | 84.8 [7.6] | 81.9 [7.9] | **79.4 [3.2]** |
| | 3 | 93.2 [9.7] | 95.6 [8.5] | 91.7 [8.6] | 93.3 [7.9] | 94.6 [9.3] | 96.8 [8.1] | **91.5 [4.6]** |
| | 4 | 94.3 [9.1] | 95.6 [9.9] | 87.9 [8.1] | 99.8 [9.1] | 90.1 [8.7] | 96.6 [9.2] | **85.1 [5.3]** |
| RT [std] (sec) | 1 | 18.6 [2.6] | 12.4 [4.0] | 11.7 [2.6] | 15.9 [4.1] | 12.4 [1.8] | 12.0 [5.7] | **4.7 [0.1]** |
| | 2 | 5.8 [1.3] | 5.2 [2.7] | 4.9 [1.9] | 6.2 [1.3] | 4.2 [1.8] | 6.0 [2.6] | **2.1 [0.2]** |
| | 3 | 5.2 [0.3] | 4.3 [0.4] | 4.1 [0.6] | 3.5 [0.3] | 4.1 [0.7] | 4.0 [0.2] | **2.3 [0.1]** |
| | 4 | 12.9 [0.6] | 15.9 [0.8] | 10.2 [0.3] | 10.0 [0.3] | 12.6 [0.5] | 11.2 [0.9] | **8.6 [0.2]** |
| Failure (%) | 1 | 1 | 2 | 2 | 1 | **0** | 2 | **0** |
| | 2 | 6 | 5 | 5 | 4 | 2 | 3 | **0** |
| | 3 | 1 | 1 | **0** | **0** | **0** | **0** | **0** |
| | 4 | 7 | 8 | 8 | 5 | 6 | 5 | **2** |
| DM [std] (meter) | 1 | 0.3 [0.1] | **0.4 [0.1]** | 0.3 [0.1] | 0.3 [0.1] | 0.3 [0.1] | **0.4 [0.1]** | **0.4 [0.1]** |
| | 2 | 0.4 [0.1] | 0.4 [0.1] | 0.3 [0.1] | 0.4 [0.2] | 0.4 [0.1] | 0.4 [0.1] | **0.5 [0.1]** |
| | 3 | 0.1 [0.0] | 0.2 [0.0] | 0.2 [0.1] | 0.2 [0.0] | 0.2 [0.1] | 0.1 [0.0] | **0.3 [0.0]** |
| | 4 | 0.3 [0.0] | 0.3 [0.1] | 0.4 [0.1] | 0.2 [0.0] | 0.3 [0.0] | 0.4 [0.1] | **0.6 [0.1]** |

arrangement, and the robot has to pass through the narrow passage formed by the two middle obstacles. The second case is an environment filled with a random number of polygonal obstacles in random positions. The next problem is a maze where the robot needs to find its way avoiding the walls of the maze. The last problem is an office-like environment where the robot has to move from one room in one corner of the office to another room.

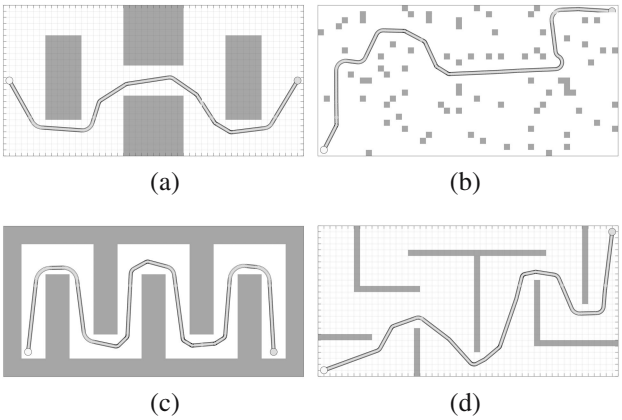


Fig. 9. Four planning problems used in the simulation studies with an instance of the generated results in each problem: a plain environment with four simple obstacles (a), an environment with 15% random obstacle occupancy (b), a maze where the robot should follow the walls to reach the goal (c), and an office-like environment where the robot has to move from one room to another in order to reach the final position (d). All environments have the same dimension (50 m×25 m), and the solutions were created with similar graph sizes ($n = 500$).

In all four cases, the robot does not possess any information about the obstacles' arrangements, and the only available information are the coordinates of the planning query pair of the start and goal configurations. The results displayed in Fig. 9 are from a similar graph with size of 500 nodes. 500 nodes were used since this worked for all algorithms in all test problems.

The performance of the planner is compared with six similar algorithms as presented in Table 1 and Fig. 10, for dealing with different types of planning uncertainty as described in Section 2 including GCS (Kurniawati *et al.*, 2012), RR-GP (Aoude *et al.*, 2013), BU-RRT* (Luders and How, 2014), FIRM (Agha-Mohammadi, 2014), CC-RRT* (Liu and Ang, 2014) and RRBT-LAS (Pilania and Gupta, 2017). The results are described based on path length ($PL$) in meters, which is the total distance travelled by the robot, the processing time ($RT$) in seconds, which is the total planning time minus the navigation time, the failure rate ($FL$), which is the percentage of failures, and the minimum shortest distance to the obstacles ($DM$) in meters, which is calculated using

$$DM = \min_{\omega}\{\min_{\theta}(||(x_c, y_c)^T, obs.||)\}, \tag{19}$$

$$\omega = \frac{\text{planning\_time}}{\Delta t}, \tag{20}$$

where $\theta$ is the sensing angle of the robot, and $\Delta t$ 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 500 samples per run, the Euclidean distance for heuristics and the 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 rated 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, $R_{\text{robot}} \cong 0.18$ [m]. This will allow having a point robot instead of the actual robot and performing the simulation in the configuration space rather than the actual work space. The same expansion strategy was applied on the boundaries of the environments. Since no postprocessing 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 of the studied algorithms in all performance variables. The planner maintains a stable path length and a stable distance to the obstacles, while it significantly reduces the processing time and the 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. The comparative results obtained from the simulation studies are presented in Fig. 10 where the performances are averaged over all four test problems.

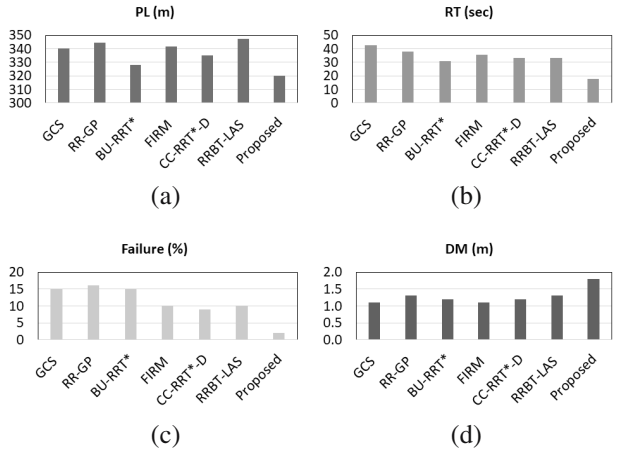



Fig. 10. Comparative simulation results based on the average performances in all four test problems. The results are averaged over 100 executions with 500 nodes in the initial graph for the path length (a), the run time (b), the failure rate (c), and the minimum distance to the obstacles (d).

The proposed planner yields higher safety, while the solution cost, runtime, and the failure rate are superior to other planners.

**5.3. Experimental setup.** In order to implement the proposed method on a real robotic platform, the result of the algorithm after postprocessing and path smoothing should be transferred into the robot in the form of control commands. 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 $(\omega_i)$ velocity of the robot as well as the during of each segment $(t_i)$,

$$FS = \begin{bmatrix} x_1 & \dots & x_m \\ y_1 & \dots & y_m \end{bmatrix}, \quad m = 2 \times ||path|| - 2, \tag{21}$$

$$CU = \begin{bmatrix} r_1 & \dots & r_m \\ \alpha_1 & \dots & \alpha_m \end{bmatrix}, \quad \alpha \in [-1, 0, +1], \tag{22}$$

$$D = \begin{bmatrix} d_1 & \dots & d_m \\ \theta_1 & \dots & \theta_m \end{bmatrix}, \tag{23}$$

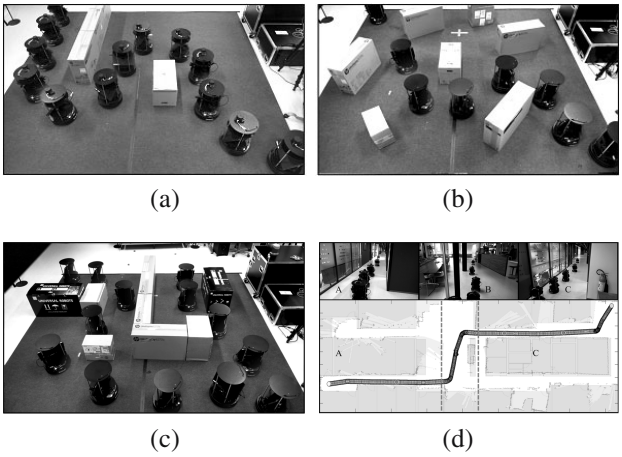
$$control = \begin{bmatrix} v_1 & \dots & v_{m-1} \\ \omega_1 & \dots & \omega_{m-1} \\ t_1 & \dots & t_{m-1} \end{bmatrix}, \tag{24}$$

$$\omega = \alpha_i \frac{v_i}{r_i}, \tag{25}$$

$$t_i = \frac{\alpha_i \theta_i}{\omega_i} + (1 - \alpha_i)\frac{d_i}{v_i}, \tag{26}$$

where $\alpha_i$ 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 road map. As the robot starts moving, it scans the surrounding area with a fixed predefined frequency $\Delta t = 2$ [sec] which means if the robot is moving along a straight line with a linear speed of $v_i$, then it scans the surrounding area every $d = 2 \times v_i$. The sensing range of the robot was limited to one meter. These values have been fund to work well with the planner during experimental trials.

**5.4. Experimental results.** The performance of the algorithm was tested on a Turtlebot2 with an Asus Xtion Pro Live camera, an A1 RPLIDAR $360^o$ laser range finder, and an onboard computer with a 2.60-GHz Intel Core i5 processor with 8 GB of memory. The details of the used robotic platform are presented in Table 2. The robot is used for navigation in four different planning problems as shown in Fig. 11 and Table 3.



(a) (b) (c) (d)

Fig. 11. Instances of the performance of the algorithm on a TurtleBot in four different experiment scenarios including simple (a), scattered (b), maze (c), and office environments (d). All presented results were obtained with $n = 500$ nodes in the final graph.

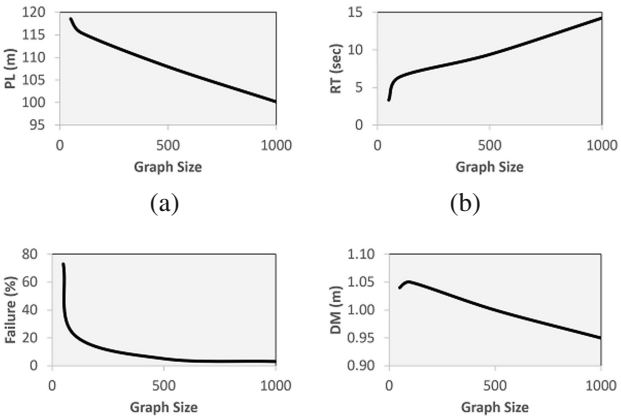



Fig. 12. Corresponding performance of the proposed planner with different numbers of nodes in the final graph in terms of the path length (a), the run time (b), the failure rate (c), and the minimum distance to the obstacles (d).

First, the robot is navigating in a 2D plain environment where two unknown static obstacles appear after the initial planning. Next, the same environment is filled with seven simple polygonal obstacles in a scattered arrangement. In the third environment, the robot is moving in a simple maze problem where there is a local minimum and avoiding it is an additional objective for the planner. Finally, the robot is moving in an office with a highly noisy map and in the presence of unknown static and dynamic obstacles. During the experiments, the linear speed of the robot was set to be $0.25$ [m/sec] and the angular velocity was calculated accordingly. The definitions of $PL$, $RT$ $Fail$ and $DM$ are the same as before. Figure 12 illustrates the average performance of the planner based on the number of nodes in the final graph.

Even though the initial placement of the robot is important for successful implementation, the planner could adopt to minor errors in the initial pose of the robot. 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 resulting in graph adaptation and the solution path is updated, an additional smoothing step takes place to prevent the robot from completely stopping and changing its orientation. Instead, the robot moves on a curve in order to follow the new path. Furthermore, Fig. 12 shows the changes in the performance of the planner relative to the initial graph size in the experimental studies. 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 ineffective on the success or failure of the planner (Fig. 12(c)). As the number of nodes in the initial graph increases, the length of the final solution and the minimum distance to the obstacles decrease since the solution keeps improving towards the optimal solution which is shorter and is very close to the boundary of the obstacles. As usual, the bigger the graph, the longer it takes to solve the problem.

Table 2. Experimental setup configuration.

| Component | Specifications |
|---|---|
| Robot | TurtleBot3 |
| Size | 354 mm × 354 mm × 420mm |
| Linear Velocity | $v_{\max} = 0.65$ [m/sec] |
| Angular Velocity | $\omega_{\max} = 180$ [deg/sec] |
| Sensor | 360 Laser Distance Sensor LDS-01 |
| SBC | Intel Core i3-4010U |
| Power Source | Lithium-Ion, 14.8V, 4400 mAh (4S2P) |

The algorithm also produces stable results with low standard deviations mainly because of the sampling radius and the graph adjustment behavior as presented with the standard deviation (std) values in Table 3. Since the samples are evenly distributed in the space, the resulting solution and the 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. It should also be mentioned that the proposed planner successfully handles sensing uncertainty by frequently updating the uncertainty matrix and accounting for unforeseen, missed or wrongly detected obstacles during the actual execution of the motion planning.

Table 3. Performance comparison based on the experimental studies for different graph sizes ($n$).

| $n =$ | 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 (0.12v | 0.21 (0.08) | 0.20 (0.09) |
| | | Experiment 2 | | |
| PL [m]/std | 21.68 (0.36) | 20.69 (0.25) | 18.17 (0.19) | 16.28 (0.11) |
| RT [sec]/std | 0.89 (0.09) | 1.56 (0.66) | 1.73 (0.79) | 2.18 (0.81) |
| Fail [%] | 6 | 1 | 0 | 0 |
| DM [m]/std | 0.21 (0.08) | 0.23 (0.11) | 0.21 (0.08) | 0.19 (0.07) |
| | | Experiment 3 | | |
| PL [m]/std | 35.11 (1.85) | 34.30 (2.15) | 31.95 (1.54) | 28.72 (0.79) |
| RT [sec]/std | 0.84 (0.12) | 1.70 (0.35) | 2.93 (0.68) | 4.18 (0.65) |
| Fail [%] | 9 | 4 | 0 | 0 |
| DM [m]/std | 0.22 (0.14) | 0.21 (0.18) | 0.20 (0.15) | 0.21 (0.11) |
| | | Experiment 4 | | |
| 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) |

## 6. Conclusion

A multiquery planner was proposed to deal with the uncertainty challenge in robotic motion planning. The proposed algorithm employs two new mechanisms 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 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 to the problem uncertainty. This mechanism detects the visible 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 the 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 even when the boundaries of the environment are not known to limit the sampling domain.

## Acknowledgment

This work is supported by the Research Council of Norway as part of the Multimodal Elderly Care Systems (MECS) project, under the grant agreement no. 247697.

## References

Achtelik, M.W., Weiss, S., Chli, M. and Siegwart, R. (2013). Rapidly-exploring random belief trees for motion planning under uncertainty, *IEEE International Conference on Robotics and Automation, ICRA 2013, Karlsruhe, Germany*, pp. 3926–3932, DOI: 10.1109/ICRA.2013.6631130.

Agha-Mohammadi, A.A. and Chakravorty, S.A.N.M. (2014). FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements, *International Journal of Robotics Research* **33**(2): 268–304, DOI: 10.1177/0278364913501564.

Aoude, G.S., Luders, B.D., Joseph, J.M., Roy, N. and How, J.P. (2013). Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns, *Autonomous Robots* **35**(1): 51–76, DOI: 10.1007/s10514-013-9334-3.

Axelrod, B., Kaelbling, L.P. and Lozano-Perez, T. (2018). Provably safe robot navigation with obstacle uncertainty, *International Journal of*

*Robotics Research* **37**(13-14): 1760–1774, DOI: 10.1177/0278364918778338.

Belghith, K., Kabanza, F. and Hartman, L. (2013). Randomized path planning with preferences in highly complex dynamic environments, *Robotica* **31**(8): 1195–1208, DOI: 10.1017/S0263574713000428.

Bry, A. and Roy, N. (2011). Rapidly-exploring random belief trees for motion planning under uncertainty, *IEEE International Conference on Robotics and Automation, ICRA 2011, Shanghai, China*, pp. 723–730, DOI: 10.1109/ICRA.2011.5980508.

Burns, B. and Brock, O. (2007). Sampling-based motion planning with sensing uncertainty, *IEEE International Conference on Robotics and Automation, ICRA 2007, Rome, Italy*, pp. 3313–3318, DOI: 10.1109/ROBOT.2007.363984.

Canny, J. (1988). *The Complexity of Robot Motion Planning*, MIT Press, Cambridge, MA.

Choset, H.M., Hutchinson, S., Lynch, K.M., Kantor, G., Burgard, W., Kavraki, L.E. and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*, MIT Press, Cambridge, MA.

Elbanhawi, M. and Simic, M. (2014). Sampling-based robot motion planning: A review, *IEEE Access* **2**: 56–77, DOI: 10.1109/ACCESS.2014.2302442.

González, D., Pérez, J., Milanés, V. and Nashashibi, F. (2016). A review of motion planning techniques for automated vehicles, *IEEE Transactions on Intelligent Transportation Systems* **17**(4): 1135–1145, DOI: 10.1109/TITS.2015.2498841.

Ha, J.S., Choi, H.L. and Jeon, J.H. (2018). Iterative methods for efficient sampling-based optimal motion planning of nonlinear systems, *International Journal of Applied Mathematics and Computer Science* **28**(1): 155–168, DOI: 10.2478/amcs-2018-0012.

Hsu, D., Kindel, R., Latombe, J.C. and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles, *International Journal of Robotics Research* **21**(3): 233–255, DOI: 10.1177/027836402320556421.

Jafarzadeh, H. and Fleming, C.H. (2018). An exact geometry-based algorithm for path planning, *International Journal of Applied Mathematics and Computer Science* **28**(3): 493–504, DOI: 10.2478/amcs-2018-0038.

Jaillet, L., Hoffman, J., Van den Berg, J., Abbeel, P., Porta, J.M. and Goldberg, K. (2011). EG-RRT: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles, *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2011, San Francisco, CA, USA*, pp. 2646–2652, DOI: 10.1109/IROS.2011.6094802.

Jaillet, L. and Simeon, T. (2004). A PRM-based motion planner for dynamically changing environments, *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2004, Sendai, Japan*, pp. 1606–1611, DOI: 10.1109/IROS.2004.1389625.

Jaillet, L. and Siméon, T. (2008). Path deformation roadmaps: Compact graphs with useful cycles for motion planning, *The International Journal of Robotics Research* **27**(11).

Janson, L., Ichter, B. and Pavone, M. (2018). Deterministic sampling-based motion planning: Optimality, complexity, and performance, *International Journal of Robotics Research* **37**(1): 46–61, DOI: 10.1177/0278364917714338.

Karaman, S. and Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning, *International Journal of Robotics Research* **30**(7): 846–894, DOI: 10.1177/0278364911406761.

Kavraki, L.E., Svestka, P., Latombe, J.C. and Overmars, M.H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces, *IEEE Transactions on Robots and Automation* **12**(4): 566–580, DOI: 10.1109/70.508439.

Khaksar, W., Tang, S.H., Ismail, N. and Arrifin, M. (2012). A review on robot motion planning approaches, *Pertanika Journal of Science & Technology* **20**(1): 15–29.

Khaksar, W., Tang, S.H., Khaksar, M. and Motlagh, O. (2013). A low dispersion probabilistic roadmaps (LD-PRM) algorithm for fast and efficient sampling-based motion planning, *International Journal of Advanced Robotic Systems* **10**(11): 1–10, DOI: 10.5772/56973.

Kingston, Z., Moll, M. and Kavraki, L.E. (2018). Sampling-based methods for motion planning with constraints, *Annual Review of Control, Robotics, and Autonomous Systems* **1**: 159–185, DOI: 10.1146/annurev-control-060117-105226.

Klaučo, M., Blažek, S. and Kvasnica, M. (2016). An optimal path planning problem for heterogeneous multi-vehicle systems, *International Journal of Applied Mathematics and Computer Science* **26**(2): 297–308, DOI: 10.1515/amcs-2016-0021.

Kurniawati, H., Bandyopadhyay, T. and Patrikalakis, N.M. (2012). Global motion planning under uncertain motion, sensing, and environment map, *Autonomous Robots* **33**(3): 255–272, DOI: 10.1007/s10514-012-9307-y.

LaValle, S.M. and Kuffner, J.J. (2001). Randomized kinodynamic planning, *The International Journal of Robotics Research* **25**(5): 378–400, DOI: 10.1177/02783640122067453.

Leven, P. and Hutchinson, S. (2011). A framework for real-time path planning in changing environments, *The International Journal of Robotics Research* **21**(12): 999–1030, DOI: 10.1177/0278364902021012001.

Li, D., Li, Q., Cheng, N. and Song, J. (2014). Sampling-based real-time motion planning under state uncertainty for autonomous micro-aerial vehicles in GPS-denied environments, *Sensors* **14**(11): 21791–21825, DOI: 10.3390/s141121791.

Liu, W. and Ang, M.H. (2014). Incremental sampling-based algorithm for risk-aware planning under motion uncertainty, *IEEE International Conference on Robotics and Automation, ICRA 2014, Hong Kong, China*, pp. 2051–2058, DOI: 10.1109/ICRA.2014.6907131.

Luders, B.D. and How, J.P. (2014). An optimizing sampling-based motion planner with guaranteed robustness to bounded uncertainty, *American Control Conference, ACC 2014, Portland, OR, USA*, pp. 771–777, DOI: 10.1109/ACC.2014.6859383.

Luna, R., Şucan, I.A., Moll, M. and Kavraki, L.E. (2013). Anytime solution optimization for sampling-based motion planning, *IEEE International Conference on Robotics and Automation, ICRA 2013, Karlsruhe, Germany*, pp. 5068–5074, DOI: 10.1109/ICRA.2013.6631301.

Otte, M. and Frazzoli, E. (2016). RRT$^{X}$: Asymptotically optimal single-query sampling-based motion planning with quick replanning, *International Journal of Robotics Research* **35**(7): 797–822, DOI: 10.1177/0278364915594679.

Pilania, V. and Gupta, K. (2017). Localization aware sampling and connection strategies for incremental motion planning under uncertainty, *Autonomous Robots* **41**(1): 111–132, DOI: 10.1007/s10514-015-9536-y.

Pilania, V. and Gupta, K. (2018). Mobile manipulator planning under uncertainty in unknown environments under uncertainty, *International Journal of Robotics Research* **37**(2–3): 316–339, DOI: 10.1177/0278364918754677.

Przybylski, M. and Putz, B. (2017). D* Extra Lite: A dynamic A* with searchtree cutting and frontiergap repairing, *International Journal of Applied Mathematics and Computer Science* **27**(2): 273–290, DOI: 10.1515/amcs-2017-0020.

Shan, T. and Englot, B. (2017). Belief roadmap search: Advances in optimal and efficient planning under uncertainty, *IEEE International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada*, pp. 5318–5325, DOI: 10.1109/IROS.2017.8206425.

Summers, T. (2018). Distributionally robust sampling-based motion planning under uncertainty, *IEEE International Conference on Intelligent Robots and Systems, IROS 2017, Vancouver, BC, Canada*, pp. 6518–6523, DOI: 10.1109/IROS.2018.8593893.

Sun, W., Patil, S. and Alterovitz, R. (2015). High-frequency replanning under uncertainty using parallel sampling-based motion planning, *IEEE Transactions on Robotics* **31**(1): 104–116, DOI: 10.1109/TRO.2014.2380273.

Vasquez-Gomez, J.I., Sucar, L.E. and Murrieta-Cid, R. (2017). View/state planning for three-dimensional object reconstruction under uncertainty, *Autonomous Robots* **41**(1): 89–109, DOI: 10.1007/s10514-015-9531-3.

**Weria Khaksar** is a postdoctoral research fellow at the Department of Informatics, the University of Oslo, Norway. His main research interests include robotics, AI, and machine learning with a special focus on motion planning and probabilistic robotics. He received a PhD in industrial engineering from University Putra Malaysia in 2013.

**Md Zia Uddin** obtained his PhD degree in biomedical engineering in 2011. He is a post doctoral research fellow in the Robotics and Intelligent Systems (ROBIN) research group at the Department of Informatics, University of Oslo, Norway. His research interests include sensors, robotics, artificial intelligence, deep learning, computer vision, and pattern recognition. He has authored more than 90 research publications, including those in international journals, conferences and book chapters. He is a senior member of the IEEE.

**Jim Torresen** is a professor of computer science at the University of Oslo, Norway. His research interests include nature-inspired computing, adaptive systems, reconfigurable hardware, and robotics and their use in complex realworld applications. He received a PhD in computer science from the Norwegian University of Science and Technology. He is a senior member of the IEEE.

Received: 27 November 2018
Revised: 22 May 2019
Accepted: 18 July 2019