J Intell Robot Syst (2016) 84:829–858
DOI 10.1007/s10846-016-0362-z

A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density

E. G. Tsardoulias · A. Iliakopoulou · A. Kargakos · L. Petrou

Received: 21 September 2015 / Accepted: 10 March 2016 / Published online: 23 May 2016

Abstract Path planning constitutes one of the most crucial abilities an autonomous robot should possess, apart from Simultaneous Localization and Mapping algorithms (SLAM) and navigation modules. Path planning is the capability to construct safe and collision free paths from a point of interest to another. Many different approaches exist, which are tightly dependent on the map representation method (metric or feature-based). In this work four path planning algorithmic families are described, that can be applied on metric Occupancy Grid Maps (OGMs): Probabilistic RoadMaps (PRMs), Visibility Graphs (VGs), Rapidly exploring Random Trees (RRTs) and Space Skeletonization. The contribution of this work includes the definition of metrics for path planning benchmarks, actual benchmarks of the most common global path planning algorithms and an educated algorithm parameterization based on a global obstacle density coefficient.

E. G. Tsardoulias (✉) · A. Iliakopoulou · A. Kargakos · L. Petrou
Faculty of Engineering, Department of Electrical and Computer Engineering, School of Electrical and Computer Engineering, Aristotle University of Thessaloniki, 54124 Thessaloniki, Greece
e-mail: etsardou@eng.auth.gr

L. Petrou
e-mail: loukas@eng.auth.gr

Keywords Path planning · Probabilistic RoadMaps · Visibility graph · Generalized Voronoi graph · Rapidly exploring random trees · Occupancy grid maps

1 Introduction

One of the most widely researched fields in robotics is a field agent's ability to operate autonomously, without an external operator's intervention. An autonomous robot should possess the following attributes:

- A way to represent the environment
- A target selection method, i.e. to have an exploration strategy
- An efficient method to move from its current pose to the next goal

The environment representation problem is solved by SLAM algorithms, which concurrently perform mapping of the environment and the robot's localization in it. The produced maps have three basic types of representation: metric maps, feature maps and hybrid that essentially are a combination of the previous two. Concerning the goal selection, an abundance of methods have been proposed, which of course vary according to the map representation. Finally, the problem of moving from one point to another in a specific environment is solved by path planning and

navigation algorithms. As expected, no universal path planning method exists, able to operate in all map representations. Thus, in this work, only path planning algorithms that can be deployed in metric maps are considered.

The type of metric representation assumed is the Occupancy Grid Maps, where the map is stored as a grid of cells, each of which represents a small fraction of the environment. Additionally, each cell holds a value, corresponding to the occupancy probability of the specific element. In order to evaluate the performance of each path planning method, five different OGMs were constructed: a totally obstacle-free space, an area with sparse obstacles, an environment of high obstacle density, one that combines dense obstacles with corridors and finally a maze. The selection of heterogeneous environments is essential towards extracting reliable results concerning the method's performance. Furthermore, a number of performance metrics are introduced: the Mean and Relative Standard Deviation of the execution time, the Mean and Relative Standard Deviation of path distance, the Path Anomaly, i.e. the amount of rotations the robot has to perform, the Estimated Time of Traversal and the Success Rate.

This paper contains a survey of the most common methods that can be used on OGMs, which are the Probabilistic RoadMaps, Visibility Graphs, Rapidly exploring Random Trees and Space Skeletonization. These methods are evaluated for their performance via the described metrics. It should be stated that in this work, we do not look into search algorithms for graphs or grids, such as Dijkstra, A*, D* and others, as we are interested in methodologies that can be efficiently applied in an Occupancy Grid Map. Obviously, the aforementioned search algorithms can be applied too, but they usually are time inefficient due to the small dimensions of each cell, thus due to the large number of grid nodes. Nevertheless, as will be presented in the next chapters, A* is deployed in most of the described algorithms, after preprocessing the grid. Finally, these methods were enhanced via automatic parameterization based on a global obstacle Density Coefficient.

Regarding the document's structure, the state of the art description exists in chapter 2, whereas in chapter 3 the experimental environments, as well as the metrics used to evaluate the maps are presented. Chapter 4 is the core of the document, as the path planning algorithms and their automatic parameterization are described. Finally, in chapter 5 the experimental results are presented and chapter 6 includes the conclusions and future work.

2 Related Work

Many path planning approaches exist, suitable for two dimensional to N dimensional spaces, for deployment in ground, aerial or underwater vehicles or even robotic arms. The first path planning method, proposed in 1979 by Losano-Perez and Wesley is the Visibility Graph [1]. There, an algorithm for path creation into a convex environment is described. The obstacles are transformed, so as to depict untraversable areas due to constraints enforced by the robot's footprint. Then, a graph is created, containing the transformed obstacles' vertexes as nodes. This is a *visibility graph* where the path creation is performed by search algorithms on the graph structure. In [2] Ghosh states that even though this method is comprehensively simple, the high computational resources needed to apply it in a large space makes it unrealistic. Thus, many of the algorithms described in the aforementioned work, aim to reduce the computational or memory resources needed for the algorithm to be efficiently executed. Nonetheless, in [3] Ghosh and Goswami describe the difficulties existent during the computation of Visibility Graphs in obstacles of arbitrary shape. Jungtae et al. tackle this problem by proposing the Quantized Visibility Graph (QVG) where the obstacles of arbitrary type are represented as polygons [4]. In our work, a variation of the Visibility Graph method has been created, as well as a hybrid algorithm, incorporating Visibility Graphs in conjunction with a sparse uniform space sampling, approach that increases the initial success rate.

Regarding skeletonization techniques, in [5], Bhattacharya and Gavrilova suggest the computation of collision - free and smooth paths using the Generalized Voronoi Diagram (GVD), as well as ways to store and update the diagram's structure in case of dynamic environments. Furthermore in [6] Garrido et al., use the GVD to discover safe areas and thereafter Fast Marching algorithm is applied, aiming at minimizing the length of the path. Another approach is presented in [7], where the Voronoi Uncertainty Fields (VUFs) are proposed in order to perform path planning under

uncertainty. This approach is two-stage, where the high-level planer employs the GVD in order to construct a collision-free path and a low-level planner adds repulsive forces deriving from the environmental obstacles.

One of path planning's most established algorithmic families are the Probabilistic Roadmaps (PRMs). There, a sampling is applied to the environmental representation and a graph is created, the edges of which are considered safe for robot traversing. Since our desire is to create a path from an origin to a goal, these two points are integrated in the graph and the path is computed via search algorithms such as Dijkstra or A*. There are many surveys comparing sampling methods for PRMs and introducing improvements [8–11]. PRMs were initially introduced in [12] from Kavraki et al. In [13] Laumond and Nissoux use the Visibility Graph's creation procedure to compute the connections of the PRM graph, whereas in [14] the Lazy PRM algorithm is described, where the checks for valid connections between the PRM nodes are minimized by initially assuming that all connections are valid. Then the minimum path is computed and is gradually checked for collisions. If a collision exists, the participant nodes are removed from the PRM structure and the next iteration is performed. Finally in [15] Hybrid PRM is proposed, combining different PRM sampling methods, according to the environment's formation.

Even though PRMs constitute a fairly popular algorithmic category regarding path planning, they pose a drawback: their size can grow too large, depending on the application space, thus making them unable to maintain low execution times. For this reason, several algorithms were proposed aiming to produce sparser graphs that create near-optimal solutions in favor of low execution times. One such method is k-PRM [16], where only the k nearest "visible" neighbors of a node are connected to it, in contrast to the vanilla version where all of them are assigned with neighborhood relations. In the same work, PRM* is proposed where the neighbors are selected based on a distance radius, nonetheless this radius is dynamically altered according to the total number of nodes. This method is similar to our approach but in our case the number of nodes depends on the environmental properties, i.e. how many are needed in order to successfully create paths under specific spatial limitations. Furthermore, in [17] the Incremental Roadmap Spanner (IRS) is proposed, which incrementally constructs an asymptotically non-optimal roadmap, resulting in a sparse graph. IRS2, the method's continuation, is proposed in [18], where redundant samples are rejected according to specific criteria in order to further lower the graph's complexity. Conclusively, K-order Surrounding Roadmap (KSR) [19] constructs a roadmap in an incremental manner by creating a tree while answering to a query, selects a part of the tree according to quality measures and adds this part to an existing roadmap.

Finally, Rapidly exploring Random Trees (RRTs) were proposed in 1998 from La Valle [20]. RRTs constitute tree-like structures created iteratively. Each RRT initiates from the starting pose (called root) and terminates when a leaf approaches the current target. RRTs have many properties such as the ability to be deployed in multidimensional spaces, to apply non-holonomic constraints, to incorporate motion constraints, they are uniformly expanded to unoccupied space and others, providing fertile ground for further research. Indeed, many surveys aim to improve the final result either regarding the path length or the execution time. Some of them are Execution Extended RRT (ERRT) [21], the combination of genetic procedures with bi-directional RRTs [22], RRT* [23], RRT*-Smart [24], T-RRT [25] and Cell-RRT [26]. An interesting approach is presented in [27] where the RRT construction is parallelized using two methods. First a batch parallelization is performed using the original RRT algorithm and next, the configuration space is divided into regions. Finally, the RRT computation is distributed in each region to available processes.

In this work, a number of the prior methods are implemented, as well as some methods' combinations and enhancements that improve their performance, minimizing the execution time, the memory required or increasing the success rate.

3 Metrics - Experiment Environments

In order to check each algorithm's ability to perform global path planning and measure its performance in an objective way, eight different metrics are introduced. The experiments were performed using five heterogeneous environments, so as to achieve maximum spatial diversity and therefore more accurate

results. As noted, the environments are metric and specifically Occupancy Grid Maps. In addition, each method is executed K times for N successive targets in each environment. Next, the performance metrics and the experiments' environments follow.

3.1 Performance Metrics

3.1.1 Mean Execution Time - t_m

The mean execution time of each algorithm in milliseconds. This metric is crucial as the optimal algorithm must construct paths efficiently, thus in a minimum amount of time.

3.1.2 Relative Standard Deviation (RSD) of Execution Time - RT

The relative standard deviation of each algorithm's execution times. The mathematical formula for its computation is:

$$RT = \frac{\sqrt{\frac{1}{N \cdot K} \cdot \sum_{i=1}^{N \cdot K} (t_i - t_m)^2}}{t_m} \quad (1)$$

Here, t_i is the specific method's i^{th} execution time during the experiments. RSD is a statistical metric used to measure the accuracy and repeatability of a certain variable. Its value is representative of each method's reliability as far as execution times are concerned, i.e. by how much the individual variables values diverge from the mean value.

3.1.3 Mean Path Distance - D_m

The mean distance of the produced paths in cells. This information is quite important, as the mean path distance directly relates to the path's traversing time, which is of maximum interest in an exploration involving situation. It is assumed that each **Path** consists of subgoals, where the i^{th} subgoal is denoted as $Path_i$. Also, the cardinality of **Path** measured in subgoals is denoted as $PathSize = |\mathbf{Path}|$. D_m is computed as the mean distance D of each path. The computation of D follows:

$$D = \sum_{i=1}^{PathSize-1} Dist(Path_i, Path_{i+1}) \quad (2)$$

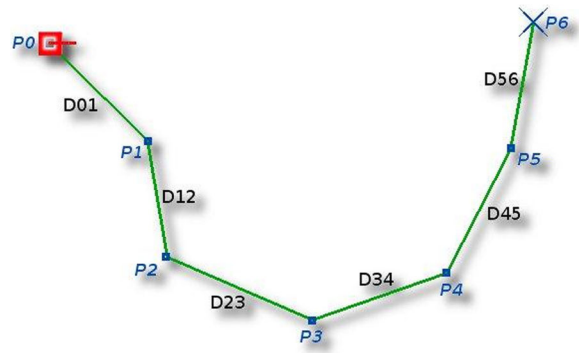

Fig. 1 Example of a simplified path, depicting its subgoals

Here, $Dist(p, q) = \sqrt{(x_p - x_q)^2 + (y_p - y_q)^2}$ is the euclidean distance between points p and q. A simplified example of a path is depicted in Fig. 1.

3.1.4 Relative Standard Deviation (RSD) of Path Distance - RD

The relative standard deviation of each path's distance. Similarly to the RT metric, the mathematical computation formula follows:

$$RD = \frac{\sqrt{\frac{1}{N \cdot K} \cdot \sum_{i=1}^{N \cdot K} (D_i - D_m)^2}}{D_m} \quad (3)$$

Here, D_i is the i^{th} path's distance in cells. This metric checks the punctuality of each method concerning its produced paths, i.e. if an algorithm produces the same path for different experimental conditions.

3.1.5 Path Anomaly - RC

The path anomaly metric measures the smoothness of each path, i.e. the amount of rotational movement the robot has to perform during the path's traversal. In our implementation RC is bounded in the [0, 1] range. The first computation step is to measure the total rotational motion a robot would do if it had traversed the specific path. In order to simplify this calculation, we assumed that the vehicle turns only at the path's subgoals trying to align with the next one. Furthermore, the robot is assumed to perform the minimum rotational movement in each turn, so the rotation is bounded in $[0^o, 180^o]$. A visual depiction of the metric's computation is shown in Fig. 2, where θ_i denotes the turn at the i^{th} subgoal.

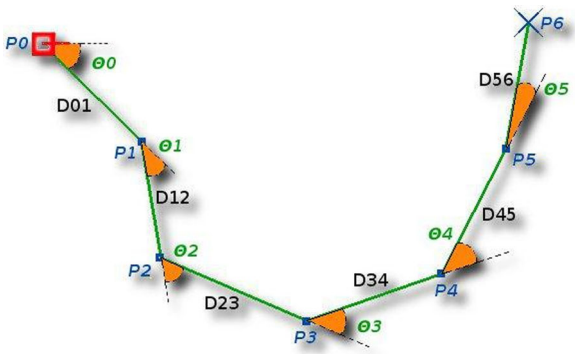

Fig. 2 Example of a RC computation - The sum of the orange angles is the total turn of the robot for traversing the specific path

Obviously the best case scenario is when the robot is aligned to the final goal, where no rotational movement is needed, resulting in an angle sum of 0^o. On the other hand, the worst case scenario is to perform an 180^o turn at each of the subgoals, something unrealistic, but introduces an upper limit to RC, which is $(PathSize - 1) \cdot 180°$. Finally, RC is calculated as follows:

$$RC = \frac{\sum_{i=1}^{PathSize} \theta_i}{(PathSize - 1) \cdot 180^o}, \in [0, 1] \tag{4}$$

This metric is of equal importance to D_m, as it directly affects the time needed by a vehicle to traverse the corresponding path.

3.1.6 Estimated Time of Traversal - ETT

The estimated time of traversal for each path. Obviously, this metric includes D_m and the total rotational movement the vehicle has to perform. The following assumptions are made: a) the vehicle performs either pure linear or rotational movement and not a combination of them, b) the linear velocity is constant and equal to $LS\ m/sec$ similarly to rotational, which is equal to $RS\ rads/sec$, c) the transition from linear to rotational motion and vice versa is instant, d) no friction or slippage phenomena occur during each movement and e) the Occupancy Grid Cell Dimension (a cell's side in meters) is denoted as $OGCD$ and the accumulated rotational move as $AngleSum = \sum_{i=0}^{PathSize-1} \theta_i$. Finally, the ETT is measured in seconds and is calculated as follows:

$$ETT = \frac{D_m \cdot ODGD}{LS} + \frac{AngleSum}{RS} \tag{5}$$

3.1.7 Success Rate - SR

The final and most important metric is each algorithm's success rate, i.e. its capability to calculate paths regardless of a) the environmental conditions and b) the robot's and target's poses. A path planning algorithm can fail for two main reasons : a) due to the transgression of an execution time limit and b) due to environmental limitations, i.e. narrow passages or lack of traversable areas. As aforementioned, each method's executions number for an environment is $N \cdot K$, where K is the experiment's ID and N the number of successive goals the algorithm must produce paths for. Thus, the success rate of each algorithm for a specific environment, if S is the number of successfully created paths, is calculated as follows :

$$SR = \frac{S}{N \cdot K} \cdot 100\ \% \tag{6}$$

3.2 Experimental Environments

In order to obtain reliable results regarding the ability of each algorithm to produce paths, the environments in which the experiments will be performed must be as heterogeneous as possible. Thus five distinct environmental setups were selected, each of which has unique spatial characteristics. The assumption is made that $OGCD = 0.02m$, so each cell has a size of $0.02 \cdot 0.02m^2$. In addition, it should be noted that the robot visits three successive targets in each environment (except for the maze-like environment). The robot's pose is denoted as R, the targets' as T_1, T_2, T_3 and as a result the vehicle performs the following transitions: $R \rightarrow T_1 \rightarrow T_2 \rightarrow T_3$.

3.2.1 Environment 1 - Free Space

The first environment is the simplest possible in terms of path creation, as it consists solely of free space. This environment is depicted in Fig. 3. Its total area is $1000 \cdot 1000\ cells^2 = 20 \cdot 20m^2 = 400m^2$. The ideal paths

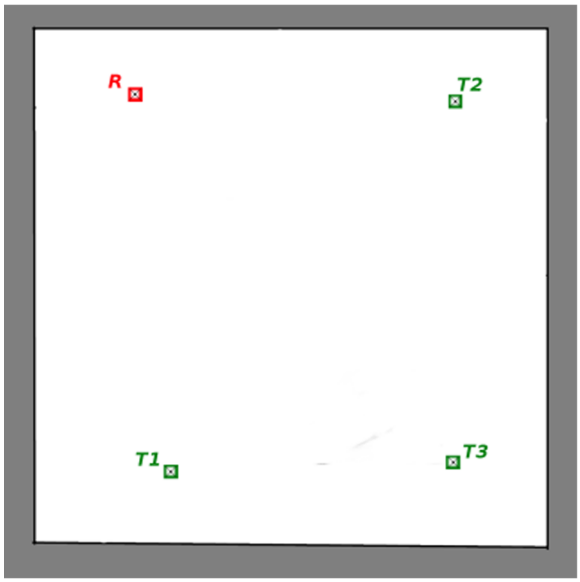

Fig. 3 First environment - Free space

for this environment are straight lines connecting the successive targets.

3.2.2 Environment 2 - Sparse Obstacles

The second environment's main characteristic is the existence of sparse obstacles. This environment is depicted in Fig. 4, it is more demanding in comparison

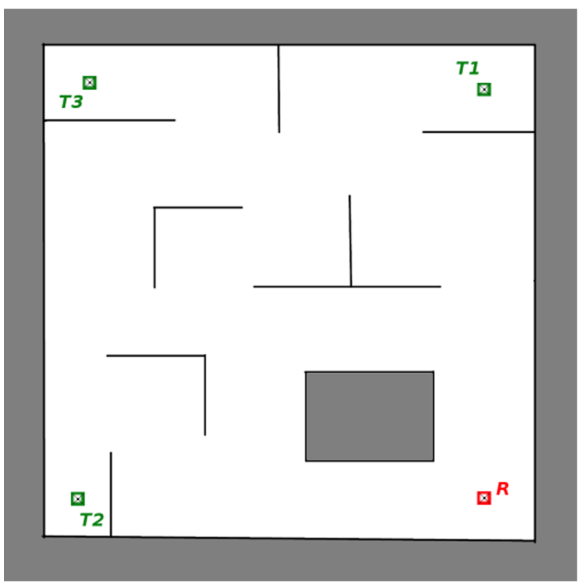

Fig. 4 Second environment - Sparse obstacles

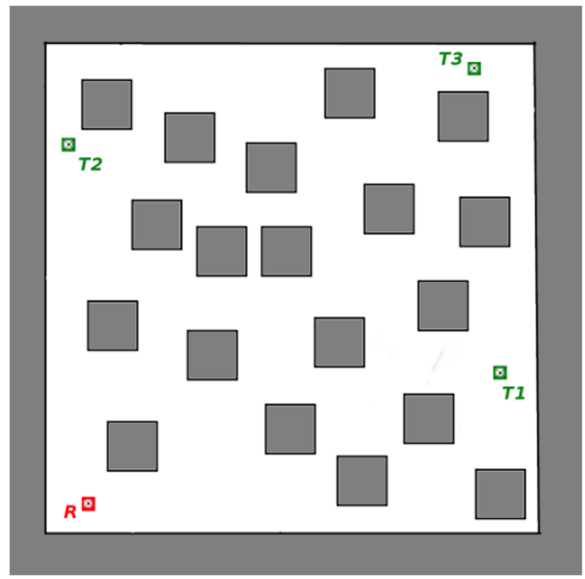

Fig. 5 Third environment - Dense convex obstacles

to the first one, but still the challenge degree is quite low for the path planning algorithms. The total area is $1000 \cdot 1000\ cells^2 = 20 \cdot 20m^2 = 400m^2$.

3.2.3 Environment 3 - Dense Convex Obstacles

The third environment consists of dense convex square obstacles and is depicted in Fig. 5. The total area of this space is $1000 \cdot 1000\ cells^2 = 20 \cdot 20m^2 = 400m^2$. This environment is characterized by increased difficulty as the dense obstacles create narrow passages.

3.2.4 Environment 4 - Dense Convex Obstacles with Corridors

The fourth environment contains not only dense obstacles, but also corridors of various widths. The environment is depicted in Fig. 6 and its total area is $1000 \cdot 1000\ cells^2 = 20 \cdot 20m^2 = 400m^2$. This space is characterized by increased traversal difficulty as the dense obstacles, combined with narrow corridors, constitute it quite demanding.

3.2.5 Environment 5 - Maze - Like Environment

The fifth and final environment is maze - like and is depicted in Fig. 7. The total area of the space is $902 \cdot 1237 px^2 = 18.04 \cdot 24.74m^2 = 446.3m^2$. The current environment was not included for simulating

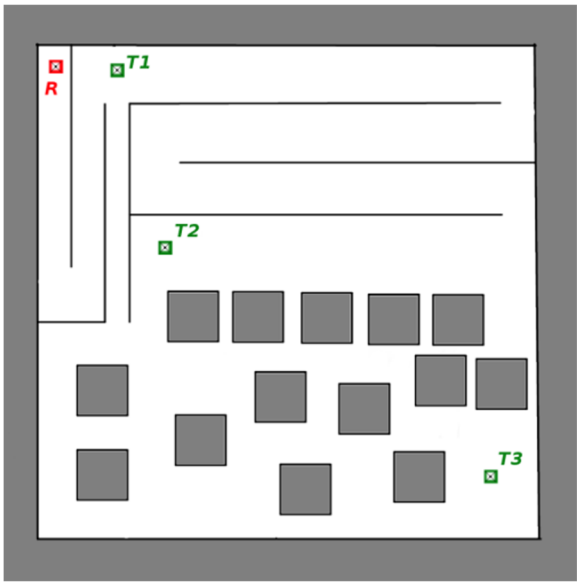

Fig. 6 Fourth Environment - Dense Convex Obstacles with Corridors

real-life applications, but to determine the capability of each algorithm to perform well in space where the path to goal is not obvious. This is the main reason why there is no succession of targets, but a sole goal, as the question asked is whether or not the specific algorithms can produce a path from the start to the goal point.

3.3 Map Container Specifications

In the following methods' descriptions, it is assumed that the maps described above ($M_i, 1 \leq i \leq 5$), are stored in a square container denoted as M_C, of size $MAP_{size} \times MAP_{size}$, where MAP_{size} is significantly larger than the actual map's size. We suppose that these maps were created by a robotic vehicle, whose initial pose was in the map's center, having coordinates $(\frac{MAP_{size}}{2}, \frac{MAP_{size}}{2})$. Furthermore, since M_i's center was the initial robot pose, this cell is always unoccupied. With X_{min}, X_{max}, Y_{min}, Y_{max}, the limits of M_i in M_C are denoted. These limits can easily be computed, as they denote a rectangle in which the entirety of the produced OGM exists. The limits of M_i will be used in order to confine the necessary path planning procedures in the "useful" space, aiming to increase the performance. They are calculated via an iterative procedure which initiates from the map's center and sweeps towards the four map directions. The limits are the vertical or horizontal lines closest to the map's center, which split the map in two half-planes, where the explored portion of the map exists entirely in one of them. Finally, is must be noted that these assumptions are also valid during an exploration procedure, when the OGM is incomplete, thus the path planning algorithms can be executed whenever the developer needs them.

These concepts are depicted in Fig. 8. For clarification purposes, the X axis is horizontal with increasing values from left to right, whereas the Y axis is vertical with increasing values from top to bottom.

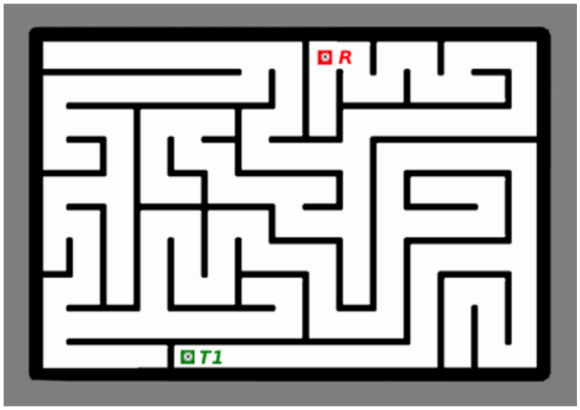

Fig. 7 Fifth environment - Maze - like environment

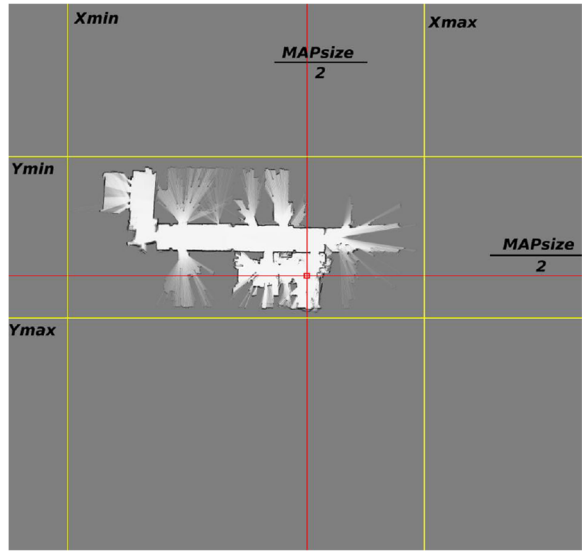

Fig. 8 Map container specifications

4 Planning Methods

As stated in the introduction, one of the contributions of this paper is the automatic parameterization of path planning methods, in order to increase their success percentage and consequently their performance. For this reason the introduction of a new metric, indicative of the obstacle density is necessary. This metric is denoted as Density Coefficient (D_c) and is computed using the values of a distance transform [28]. The selected distance transform calculates the Manhattan Distance form the obstacles and is computed via a BrushFire algorithm [29], initiating from the occupied cells and propagating in the unoccupied space. An example of a distance function calculated for an occupancy grid map is depicted in Fig. 9.

If X_{free} is the set of all OGM cells corresponding to unoccupied space, $x_i \in X_{Free}$ a single cell and $B[x_i]$ the distance function value of x_i, D_c is calculated as the difference of the mean value of distance function values (μ_B) minus its standard deviation (σ_B):

$$D_c = \mu_B - \sigma_B \tag{7}$$

There,

$$\mu_B = \frac{1}{|X_{Free}|} \cdot \sum_{\forall x_i \in X_{Free}} B[x_i] \tag{8}$$

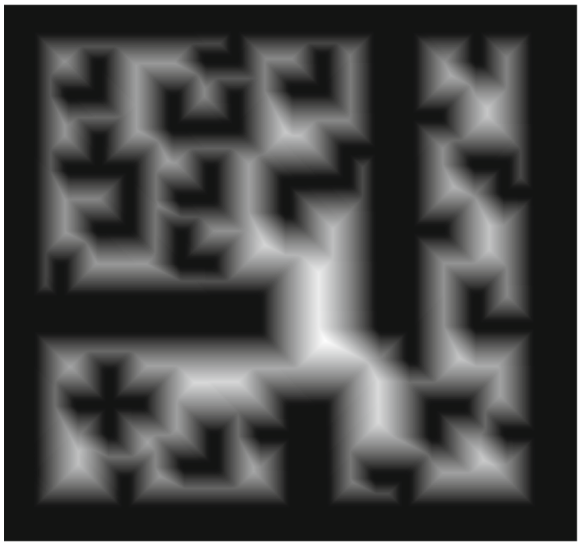

Fig. 9 Example application of BrushFire algorithm in an OGM

Table 1 μ_B, σ_B and D_c values of each environment

Environment	μ_B	σ_B	$D_c = \mu_B - \sigma_B$
Environment 1	165.919	116.969	48.95
Environment 2	54.6265	33.8798	20.746
Environment 3	35.12.88	23.0628	12.066
Environment 4	27.7183	18.7098	9.0085
Environment 5	14.3008	8.4105	5.8903

$$\sigma_B = \sqrt{\frac{1}{|X_{Free}|} \cdot \sum_{\forall x_i \in X_{Free}} (B[x_i] - \mu_B)^2} \tag{9}$$

In Table 1 the values of μ_B, σ_B and D_c are presented for all five environments.

As the results indicate, the difference between the distance function's mean and standard deviation values, constitutes a valid measure of the environmental obstacle density. The D_c metric can estimate the obstacle density assuming their dispersion is close to uniform. It is important to emphasize that the proposed enhancements will indeed produce better results if the environment is homogeneous concerning its structural properties. On the opposite occasion the computed D_c parameter will mistakenly describe the environment, fact that can lead to the path planning algorithm's failure. Next, the global path planning methods will be described. For each method its standard version and the one utilizing the D_c coefficient in order to increase their success rate will pe presented.

It should be stated that no modifications were made to them (like post-processing smoothing or introduction of motion constraints), in order to be able to perform benchmarks on their standard versions.

4.1 Probabilistic RoadMaps (PRMs)

The basic concept behind the PRM approach is the construction of a set of points usually produced by spatial sampling. This set contains the nodes of a graph **G**, which can have neighborhood relations if the nodes' corresponding connections are collision-free. The initial point n_R (the robot's pose) and the target n_G are inserted in **G** and are connected to the appropriate neighbors. Finally the path from n_R to

n_G is calculated by deploying the A* algorithm on **G**. There are many sampling alternatives proposed by the scientific community. Here four methods will be described: the *uniform* and *random* space sampling, the *sampling with Halton Sequences* and the *uniform incremental sampling* that improves the performance of the uniform one.

4.1.1 Uniform Space Sampling - USS

As aforementioned, the path extraction's first step is to create the **G** graph. This procedure is described in Algorithm 1. To do so, a sampling procedure initiates from the point (X_{min}, Y_{min}) (lines 6–7). The sampling step is denoted as S_{UNI} and the sampling procedure is performed by rows. Each sample n_k is tested for obstacle proximity by checking if $B[n_k] > MWD$, where MWD stands for Minimum Wall Distance (line 13). This constant is essential for the graph creation and subsequently for paths that are collision safe. If the prior condition holds, n_k is inserted in **G** (line 14). Assuming that an arbitrary state of the algorithm is reached where the point $n_k = (i, j)$ and $(i = X_{min} + p \cdot S_{UNI}, j = Y_{min} + q \cdot S_{UNI})$, the graph edges are created on the fly by checking if the points $(i - S_{UNI}, j - S_{UNI})$, $(i, j - S_{UNI})$, $(i + S_{UNI}, j - S_{UNI})$, $(i - S_{UNI}, j) \in \mathbf{G}$ (Fig. 10, lines 17–28). If this condition is valid, the proper edge connections are created. Assuming that $S_{UNI} < 2 \cdot MWD$, no further checks concerning edge collisions with obstacles are needed, as this is geometrically impossible. Furthermore, it should be noted that only four connectivity checks are needed instead of eight, as only the aforementioned nodes could be created due to the sampling process.

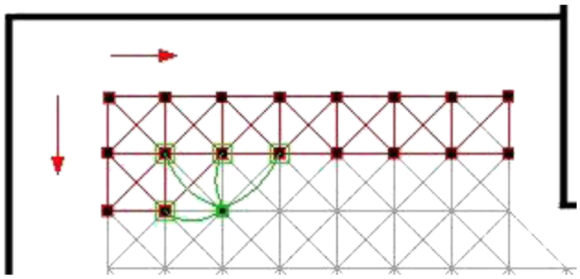

Fig. 10 Connectivity checks during uniform sampling

Algorithm 1 USS algorithm

```
1: procedure COLLISION(n_p, n_q)
2:     True if the line from n_p to n_q collides with an obstacle
3: ---------------------------------------------------------------
4: procedure CREATEUSSGRAPH( )
5:     G = ∅
6:     x_t = X_min
7:     y_t = Y_min
8:     while y_t < Y_max do
9:         y_t += S_UNI
10:        while x_t < X_max do
11:            x_t += S_UNI
12:            n_k = (x_t, y_t)
13:            if B[n_k] > MWD then
14:                Add n_k in G
15:            else
16:                continue to next iteration
17:            n_p = (x_t − S_UNI, y_t − S_UNI)
18:            if not Collision(n_k, n_p) then
19:                Connect n_k to n_p in G
20:            n_p = (x_t − S_UNI, y_t)
21:            if not Collision(n_k, n_p) then
22:                Connect n_k to n_p in G
23:            n_p = (x_t, y_t − S_UNI)
24:            if not Collision(n_k, n_p) then
25:                Connect n_k to n_p in G
26:            n_p = (x_t + S_UNI, y_t − S_UNI)
27:            if not Collision(n_k, n_p) then
28:                Connect n_k to n_p in G
29:    return G
30: ---------------------------------------------------------------
31: procedure USSPATHPLANNING( )
32:    G = CreateUSSGraph()
33:    for all n_k ∈ G do
34:        if Distance(n_k, n_R) < S^ext_UNI then
35:            Connect n_R to n_k in G
36:        if Distance(n_k, n_G) < S^ext_UNI then
37:            Connect n_G to n_k in G
38:    Path = A*(n_R, n_G, G)
39:    return Path
```

Up to this point, **G** has been created, so n_R and n_G must be inserted in the graph. Generally, the potential goals can exist in locations whose distance function value to the closest obstacle is less than MWD.

In order to improve the connectivity of the initial and goal points towards the rest of the graph, a new constant is created, S^{ext}_{UNI}, such that $S^{ext}_{UNI} > S_{UNI}$ but still $S^{ext}_{UNI} < 2 \cdot MWD$. S^{ext}_{UNI} must be larger than S_{UNI}, as the robot pose can exist out of **G** (Fig. 11, green pose), since the robot can be located in close proximity to an obstacle (even if it should not). Thus peculiar cases can arise when the robot pose is far from any element of **G** due to the local OGM morphology, constituting the path planning algorithm bound to fail.

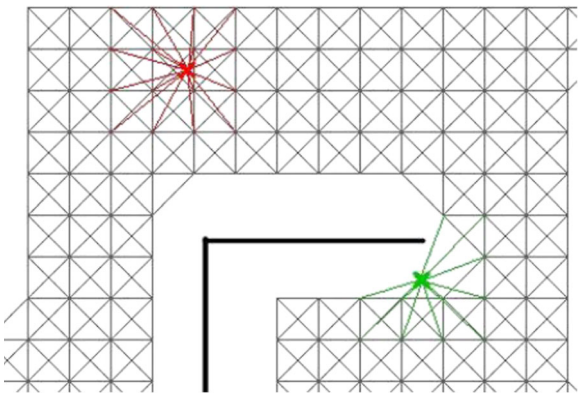

Fig. 11 Extended connectivity for initial and goal points

Conclusively, the initial and goal points are connected to each $n_R \in \mathbf{G}$ for which their euclidean distance is less than S_{UNI}^{ext} (Fig. 11, lines 34-37). Finally, graph **G**'s creation has been concluded and A* is deployed from n_R to n_G through **G** (line 38).

This method's drawback is that the S_{UNI} constant must be chosen arbitrarily. If S_{UNI} is assigned a high value, the algorithm will be faster but will fail in narrow areas. On the contrary, a low S_{UNI} value results in a high success ratio but increased execution times. In order to achieve a balance, S_{UNI}'s value is being set according to the environmental obstacle density, i.e. based on the D_c metric. Thus $S_{UNI} = 1.5 \cdot D_c$, $MWD = 1.2 \cdot D_c$ (since S_{UNI} must be lower than $2 \cdot MWD$) and $S_{UNI}^{ext} = 2 \cdot S_{UNI}$. It must be noted that in both simple and enhanced version of Uniform space sampling, MWD must be assigned a value higher than the robot's width, in order for the traversing to be safe and collision-free. Figures 12 and 13 illustrate two examples of paths created by Uniform Space Sampling.

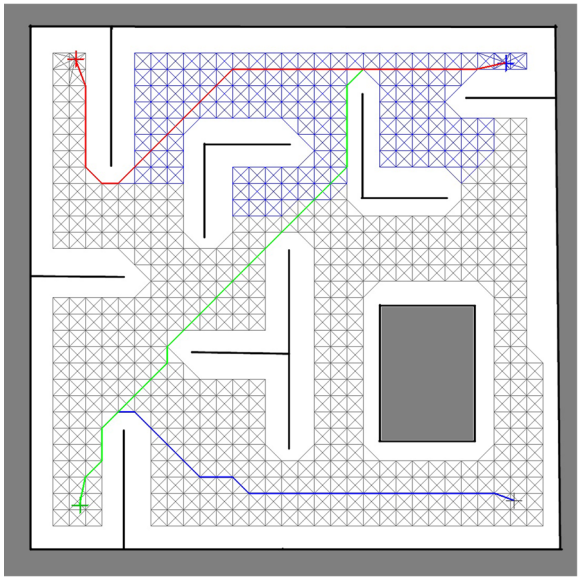

Fig. 12 Example paths using USS for Environment 2

4.1.2 Random Space Sampling - RSS

In Random space sampling, as the name suggests, **G**'s construction involves a random sampling procedure in the unoccupied space. The theoretical advantage of random sampling in comparison to the uniform one is that random samples can potentially be created in narrow areas, where uniform sampling constantly fails due to the S_{UNI} constrain. This method is described in Algorithm 2. Hereinafter only the $CreateGraph$ procedure is going to be presented, since the $Collision$ and $PathPlanning$ procedures are common for all methods.

Algorithm 2 RSS algorithm

```
1:  procedure CreateRSSGraph( )
2:      G = ∅
3:      N_Rand = A_{X_Free} / R_Spar
4:      counter = 0
5:      while counter < N_Rand do
6:          counter+ = 1
7:          n_k = Random(M_i)
8:          if B[n_k] > MWD then
9:              Add n_k in G
10:             for all n_p ∈ G do
11:                 if Distance(n_p, n_k) < R_max then
12:                     Connect n_p, n_k in G
13:     return G
```

In order to perform random sampling, the number of needed random samples must be specified. As sampling is performed in the unoccupied space, it makes sense for the samples number N_{Rand} to be proportional to the free space's area, denoted as $A_{X_{Free}}$. Constant R_{Spar} (Random sampling Sparseness) is introduced in order to specify the exact number of samples needed: $N_{Rand} = A_{X_{Free}}/R_{Spar}$ (line 3). R_{Spar} is arbitrarily chosen based on the desired density of sampling points per space unit. Similarly to the previous method, the samples are tested for proximity by $B[n_k] > MWD$ (line 8). The neighboring relations of **G** are defined via the samples' proximity status. Specifically an edge is created between n_p and n_q samples if $Dist(n_p, n_q) < R_{max}$, where R_{max} is the maximum circle radius of center n_p, for an element

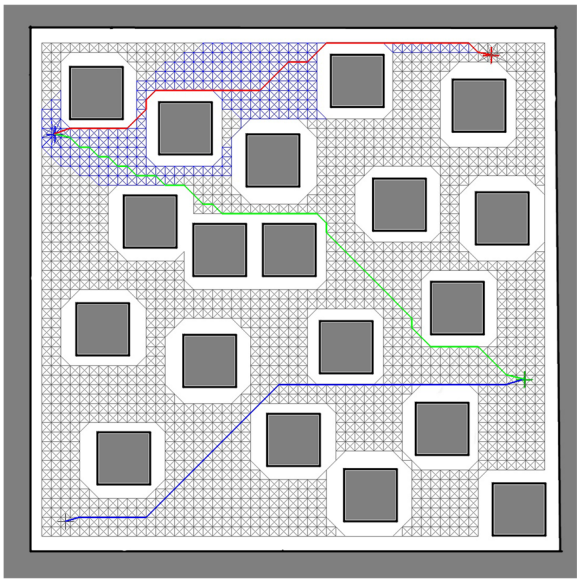

Fig. 13 Example paths using USS for Environment 3

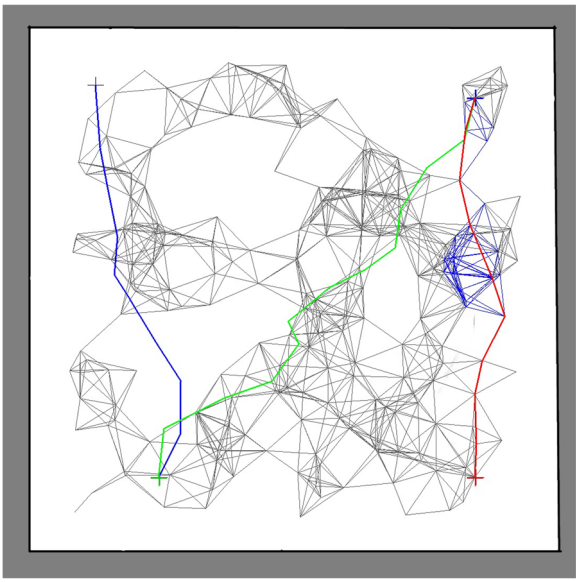

Fig. 14 Example paths using RSS for Environment 1

n_q to be connected to it (lines 10-12). Again, R_{max} must be smaller than $2 \cdot MWD$ in order not to check for connection collisions. Finally the initial and goal points are inserted into **G** and connected to the other samples if their proximity distance R^{ext}_{max}, is less than R_{max}, where $R^{ext}_{max} < 2 \cdot MWD$.

The enhanced version of the algorithm suggests that $R_{max} = 1.5 \cdot D_c$, $MWD = 1.2 \cdot D_c$ and $R^{ext}_{max} = 2 \cdot R_{max}$. In addition R_{Spar} must be automatically set in order to have less samples in sparse environments and more to complex ones. Since R_{Spar} implies spatial dimensionality (2D), it is expected to select $R_{Spar} = (D_c)^2$. In Figs. 14 and 15 two path examples are depicted.

4.1.3 Space Sampling with Halton Sequences - HSS

Due to Random space sampling's randomness, the final graph **G** might be unconnected in certain narrow passages of the environment. Thus, it would make sense to exploit the advantages of both Uniform and Random sampling methods. A *Halton sequence* is characterized as a low-discrepancy sequence. It's basic property is that the sequences samples' $x_1, x_2, ..., x_N$ successive differences tend to be low, or in other words it approaches the Uniform space sampling, as far as discrepancy is concerned. Another name for these sequences is "Quasi-random", since the samples produced are not random, but possess some properties of random sequences.

Given a prime number p, a sample i of Halton sequence is computed on the basis of p as follows: $i = a_0 + a_1 \cdot p + a_2 \cdot p^2 + ...$, where a is a set containing integer numbers. The desired sequence sample $r(i, p)$, belonging to the [0, 1] range, is calculated by

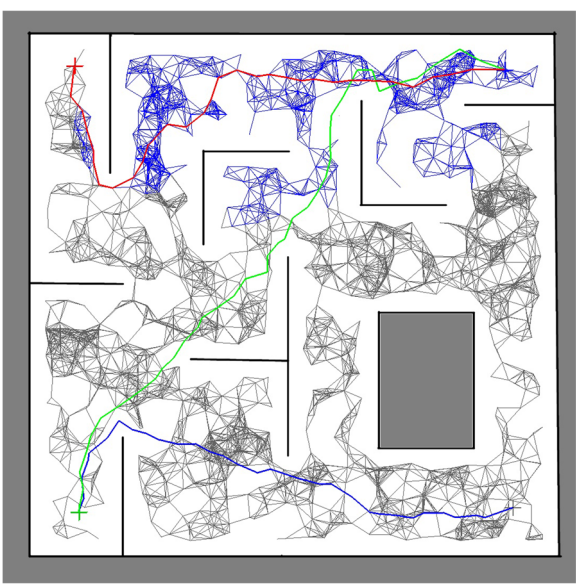

Fig. 15 Example paths using RSS for Environment 2

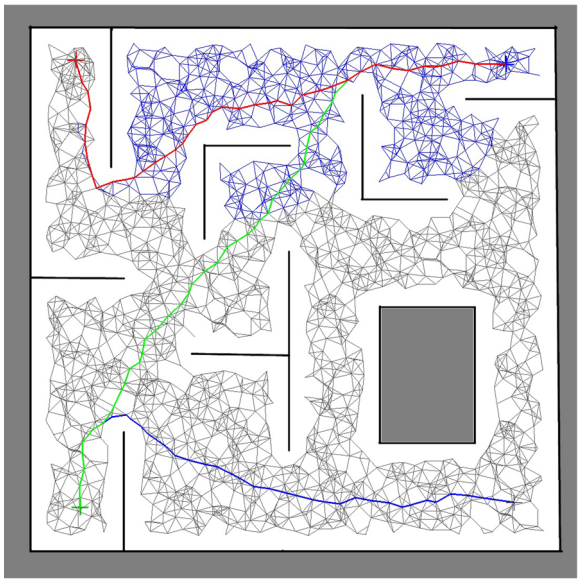

Fig. 16 Example paths using HSS for Environment 2

reversing the number's "bits" and moving the decimal point:

$$r(i,\, p) = \frac{a_0}{p} + \frac{a_1}{p^2} + \frac{a_2}{p^3} + \ldots \tag{10}$$

Thus, a node $n_k \in \mathbf{G}$ is a point with coordinates produced by two Halton sequences with different bases (2 and 3 in our work): $n_k = [r(k, 2), r(k, 3)]$.

This method is presented in Algorithm 3.

Algorithm 3 HSS algorithm

```
1:  procedure CreateHSSGraph( )
2:      G = ∅
3:      N_Halton = A_{X_Free} / H_Spar
4:      counter = 0
5:      while counter < N_Halton do
6:          counter+ = 1
7:          n_k = (r(counter, 2), r(counter, 3))
8:          if B[n_k] > MWD then
9:              Add n_k in G
10:             for all n_p ∈ G do
11:                 if Distance(n_p, n_k) < R_max then
12:                     Connect n_p, n_k in G
13:     return G
```

Again, the parameters $R_{max} = 1.5 \cdot D_c$, $MWD = 1.2 \cdot D_c$ and $H^{ext}_{max} = 2 \cdot R_{max}$ are used and $H_{Spar} = (D_c)^2$, where the samples' number equals to $N_{Halton} = A_{X_{Free}}/H_{Spar}$ (line 3). In Figs. 16 and 17 two examples of path creation using Halton sequences sampling are depicted.

4.1.4 Uniform Incremental Space Sampling - UISS

Even though the Uniform space sampling method is quite straightforward and time efficient, it has a major drawback: it is a batch method, i.e. every time a new path is needed, the whole graph needs to be recreated. Thus, the vanilla method is altered in order to be incremental, aiming at storing the sampling graph and only update it when required. This method is described in Algorithm 4.

Algorithm 4 UISS algorithm

```
1:  procedure CreateUISSGraph( )
2:      Remove n_R, n_G from G
3:      x_t = MAP_size/2 − ⌊(MAP_size/2 − X_min)/S_UNI⌋ · S_UNI
4:      y_t = MAP_size/2 − ⌊(MAP_size/2 − Y_min)/S_UNI⌋ · S_UNI
5:      counter = 0
6:      while y_t < Y_max do
7:          y_t+ = S_UNI
8:          while x_t < X_max do
9:              x_t+ = S_UNI
10:             n_k = (x_t, y_t)
11:             if UISS_HIST[n_k] not equal to −1 then
12:                 if B[n_k] < MWD then
13:                     Remove n_k from G
14:                     Remove n_k's connections from G
15:             else
16:                 if B[n_k] > MWD then
17:                     Add n_k in G
18:                     UISS_HIST[n_k] = counter
19:                     counter+ = 1
20:                 else
21:                     Continue
22:                 for i in [−1, 1] do
23:                     for j in [−1, 1] do
24:                         n_p = (x_{n_k} + i · S_UNI, y_{n_k} + j · S_UNI)
25:                         if n_p == n_k then
26:                             continue
27:                         if not Collision(n_p, n_k) then
28:                             Connect n_k, n_p in G
29:     return G
```

The first noteworthy difference between the simple (USS) and the incremental uniform space sampling (IUSS) methods, is the initial sampling point. In USS, the sampling procedure initiated from the upper left map corner (X_{min}, Y_{min}), which was in general variable, due to the map expansion during the robot exploration. Consequently, the whole graph would be shifted, making the incremental approach impossible. Thus, a constant reference must be created, which in our case was the center of the map container $(\frac{MAP_{size}}{2}, \frac{MAP_{size}}{2})$ (which is the initial pose of the robot, as described in Section 3.3). This fact ensures that it will be unoccupied, thus a node can be placed there. The initial sampling point is calculated

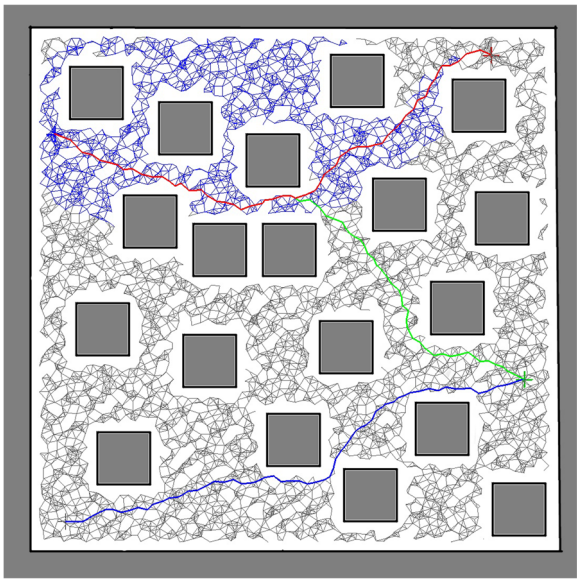

Fig. 17 Example paths using HSS for Environment 3

as the first cell in the map's limits whose coordinates' distance from the map's center is an integer multiple of S_{UNI} (lines 3-4). This way, it is ensured that the cell existing at the map's center will be in **G**, thus all potential **G**'s elements have predefined poses.

Furthermore, a matrix $UISS\_HIST$ of the same size as the OGM container is stored, holding each graph node's id (if such node exists) or -1 in contrary. This way, checking the graph for nodes in an area of interest is quite efficient (lines 20–26). Thus, each time a new path must be created, the whole graph structure is checked for unoccupied areas that do not contain nodes. If such areas exist, the graph is updated by inserting nodes and connections to their neighbors are created. Furthermore, the already existent elements of **G** are checked for proximity to the obstacles, as it is possible for the B field to have erroneous values, that are corrected in a future timestamp. If such occurrences are found, the corresponding nodes are removed from **G** (lines 11–13). Finally, the old poses of robot and goal are removed and the current ones are inserted, in order for the A^* algorithm to construct the path. Since the paths are visually similar to the USS method's ones, no figures are presented.

4.2 Visibility Graphs

Visibility graphs comprise nodes whose connectivity relies on *visibility*, i.e. if two nodes can be connected with a straight line segment, without crossing any obstacles. Of course, this condition must be valid in both space sampling and RRT methods with the difference of the nodes' selection. Visibility graphs usually contain nodes which are the edges of environment's convex obstacles.

4.2.1 Simple Visibility Graph - VG

Many different approaches exist concerning the detection of convex spatial obstacles' edges. The most common ones involve line sweeps and the identification of edges from spatial points with certain characteristics. The first approach utilizes a straight line segment that sweeps the environment. When the line's body collides with an obstacle, the collision point is considered a visibility graph's node. The second proposition applies ray casting on a specific set of points. When two successive virtual rays have a large length difference (a ray's length is the distance from the origin to the obstacle that the ray eventually hits), the collision point of the shortest ray is considered to be a graph node. In the current approach the second method is employed. Since the final path consists of lines segments connecting the points which lay on the obstacles' surfaces, it is understandable that the path is neither safe nor collision free. Thus, virtual obstacles are created by "inflating" the real ones by MWD cells (the distance considered safe for the robot's traversal). This method is described in Algorithm 5.

Algorithm 5 VG algorithm

```
1:  procedure CreateVisibilityGraph( )
2:      o_inf = InflateObstacles(MWD)
3:      G = {n_R, n_G}
4:      Open = ∅
5:      Closed = ∅
6:      Insert n_R and n_G in Open
7:      while Open is not ∅ do
8:          n_p = any element of Open
9:          Remove n_p from Open
10:         Insert n_p in Closed
11:         P_q = RayCasting(n_p, o_inf)
12:         for all i ∈ P_q do
13:             if Distance(P_q^i, P_q^{i+1}) > D_ray^max then
14:                 if ||r_q^i|| < ||r_q^{i+1}|| then
15:                     if P_q^i not in Closed then
16:                         Insert P_q^i in G
17:                         Connect P_q^i, n_p in G
18:                 else
19:                     if P_q^{i+1} not in Closed then
20:                         Insert P_q^{i+1} in G
21:                         Connect P_q^{i+1}, n_p in G
22:     return G
```

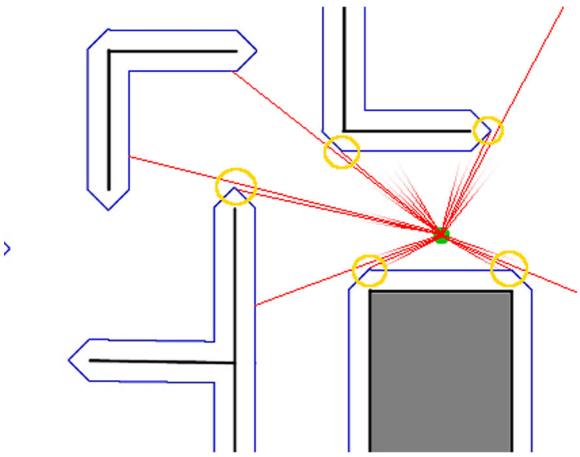

Fig. 18 Example of ray casting

The visibility graph's construction initiates by inserting n_R and n_G to **G** (line 2). Additionally, the **Open** and **Closed** sets exist, where **Open** contains the nodes to be expanded, whilst **Closed** contains the already expanded nodes. Initially, n_R and n_G are inserted in **Open** (line 5). At any iteration, the nodes existing in **Open** set are expanded, meaning that from each node $n_q \in$ **Open** a ray casting procedure is performed, creating circumferentially N_{ray} rays ($N_{ray} = 360$ in our implementation), where each ray is denoted as r_q^i. Each ray is being cast till its collision with an inflated obstacle (at the point P_q^i), or till a maximum ray length is exceeded (line 11). The maximum length was arbitrarily set to 300 cells. If two successive rays' collision points abstain more than D_{ray}^{max}, i.e. if $Dist(P_q^i, P_q^{i+1}) > D_{ray}^{max}$, the shortest ray's collision point is inserted in **G** (lines 16 and 20). Additionally, the new node is inserted in **Open**, unless it already exists in **Open** or **Closed** (lines 14–21). A ray casting example is depicted in Fig. 18 where the virtual obstacles are presented in blue and the collision points which are stored as Visibility Graph's nodes in orange.

The method terminates when **Open** $= \emptyset$. To construct the path, A* is applied. In order to enhance the method, MWD is parameterized as $MWD = 1.5 \cdot D_c$. Two paths' examples created by the Visibility Graph method are depicted in Figs. 19 and 20.

4.2.2 Visibility Graph with Sparse Uniform Sampling - VGSS

The Visibility Graph method's performance is acceptable in purely convex environments. Unfortunately, the experimental environments are not convex, as they contain circumferential obstacles. For that reason, Visibility Graph methods do not have a high success rate, especially in corridor-like spaces. In order to overcome this problem, a sparse uniform sampling is

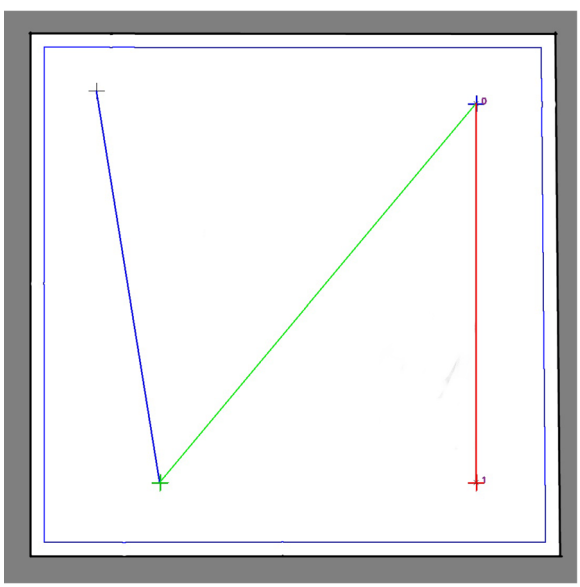

Fig. 19 Example paths using VG for Environment 1

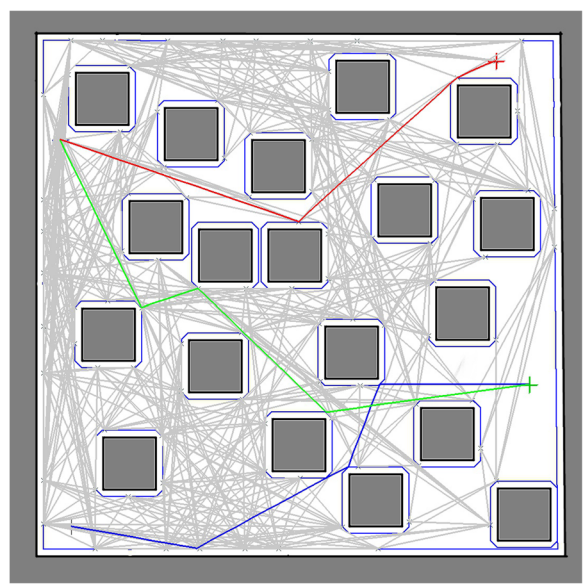

Fig. 20 Example paths using VG for Environment 3

introduced, whose nodes are inserted in **Open** with the algorithm's initiation, aiming at improving the graph's connectivity. This method is presented in Algorithm 6.

Algorithm 6 VGSS algorithm

```
1: procedure CreateVisibilityGraph( )
2:     o_inf = InflateObstacles(MWD)
3:     G = {n_R, n_G}
4:     G_ss = Sparse sampling in X_free with step S_VG
5:     Open = G_ss
6:     Closed = ∅
7:     Insert n_R and n_G in Open
8:     while Open is not ∅ do
9:         n_p = any element of Open
10:        Remove n_p from Open
11:        Insert n_p in Closed
12:        P_q = RayCasting(n_p, o_inf)
13:        for all i ∈ P_q do
14:            if Distance(P_q^i, P_q^{i+1}) > D_ray^max then
15:                if ||r_q^i|| < ||r_q^{i+1}|| then
16:                    if P_q^i not in Closed then
17:                        Insert P_q^i in G
18:                        Connect P_q^i, n_p in G
19:                else
20:                    if P_q^{i+1} not in Closed then
21:                        Insert P_q^{i+1} in G
22:                        Connect P_q^{i+1}, n_p in G
23:    return G
```

The sampling step used is denoted as S_{VG}. In order for the algorithm to adjust to various environments, $MWD = 1.5 \cdot D_c$ and $S_{VG} = 7.5 \cdot D_c$. Two example paths created by Visibility Graph with Sparse Uniform Sampling are depicted in Figs. 21 and 22.

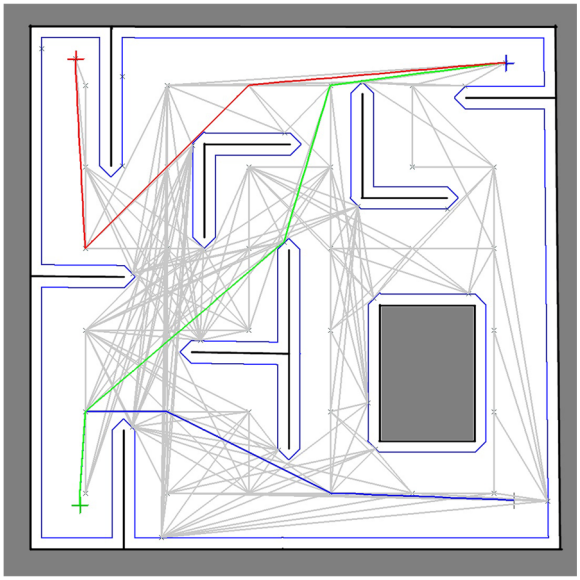

Fig. 21 Example paths using VGSS for Environment 2

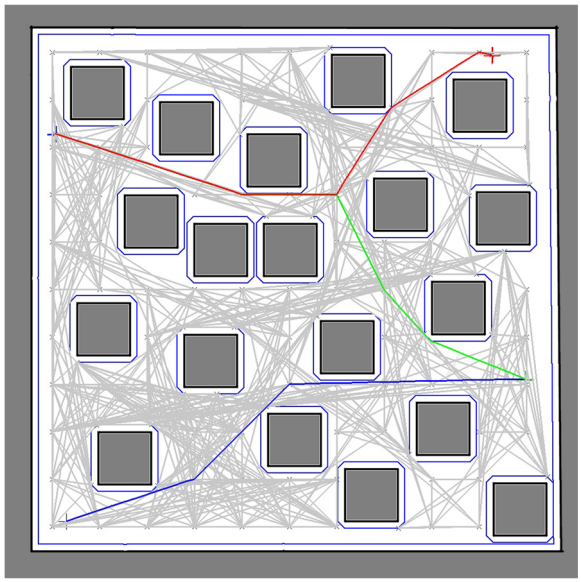

Fig. 22 Example paths using VGSS for Environment 3

4.3 Rapidly Exploring Random Trees (RRTs)

RRTs stands for *Rapidly exploring Random Trees*, a data structure designed to perform efficient searches in high dimensional and generally non-convex spaces. This algorithm involves the creation of a tree structure, in a way that each point belonging to X_{Free} is connected to the closest tree member. The RRT algorithm is repetitive and its main characteristic is that it rapidly minimizes one random point's distance from the tree structure. RRTs are very efficient in path creation problems including obstacles or kinematic restrictions and are considered a technique able to construct open loop paths for non-linear problems. It is important to state that even though RRTs can be created with kinematic constraints, in our work we chose to follow the simplest assumption, i.e. that our robot can perform only pure rotational or pure linear motion. This way, the RRTs were constructed without kinematic constraints, even though if they did their final metrics would probably be more favorable (specially the path anomaly coefficients). A typical RRT structure is depicted in Fig. 23.

The tree structure initiates from n_R and propagates through X_{Free} until n_G is reached. Then, the path is constructed by reverse traversing of the tree, initiating from n_G and finalizing at n_R. Four RRTs variations will be presented: the standard RRT, RRT *, multiple RRTs and Multiple incremental RRTs.

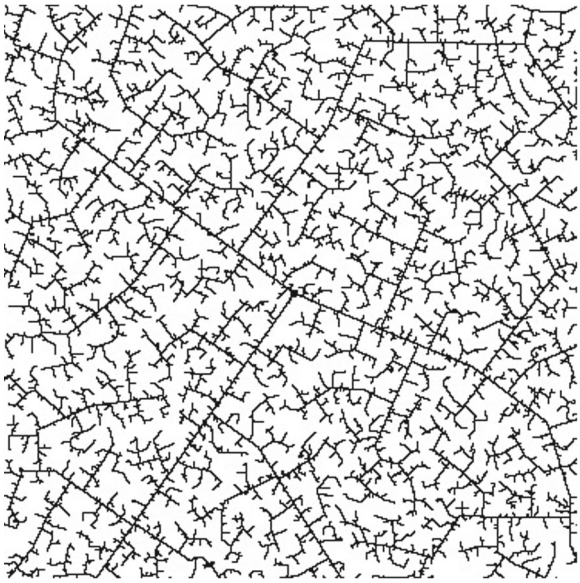

Fig. 23 RRT structure[1]

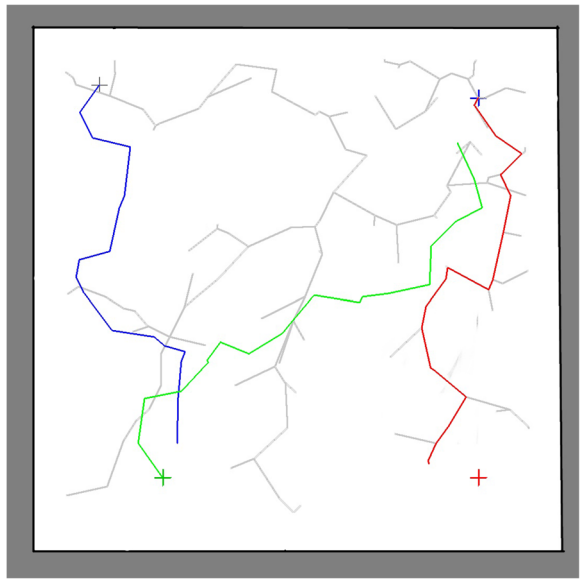

Fig. 24 Example paths using RRT for Environment 1

4.3.1 Standard RRT

The construction of the standard RRT structure is presented in Algorithm 7 and initiates by the selection of an unoccupied cell, which will be the tree root. In our case, this point is n_R, i.e. the robot's pose (line 3). Then, at each iteration a random point $n_{rand} \in X_{Free}$ is selected (line 7). Next, the tree ($\mathbf{G}$) is searched to find the closest node $n_{near} \in N$ (line 8). The algorithm proceeds by selecting a point n_{new} that belongs on the virtual line that connects n_{rand} with n_{near}, located at a random distance D_r from n_{near} (line 9). D_r is bounded to [0, D^{exp}_{max}], where D^{exp}_{max} denotes the maximum expansion distance possible. If the connectivity between n_{new}, n_{near} is possible in accordance to spatial restrictions, an edge is created and n_{near} becomes n_{new}'s parent (lines 10–12). On the contrary, n_{rand} is rejected and the next algorithmic iteration is performed (lines 14–15). The algorithm ends when n_G can be connected with any node $n_k \in \mathbf{G}$ (lines 16–17). Similarly to the other methods, if $B[n_{new}] > MWD$, n_{new} is rejected.

In the enhanced version we automatically assign the D^{exp}_{max} and MWD values. Specifically $D^{exp}_{max} = 1.5 \cdot D_c$ and since D^{exp}_{max} must be less than $2 \cdot MWD$, $MWD = D_c$. Two path creation examples are depicted in Figs. 24 and 25.

[1]Source: http://planning.cs.uiuc.edu/img2043.gif

Algorithm 7 RRT algorithm

```
1: procedure CreateRRTGraph( )
2:     G = ∅
3:     Insert n_R in G
4:     counter = 0
5:     while counter < K_max and n_G ∉ G do
6:         counter+ = 1
7:         Select n_rand from X_free
8:         Find n_near ∈ G :
                argmin_{n_near}(distance(n_near, n_rand))
9:         Pick n_new = Expand(n_near, n_rand, D^max_exp)
10:        if n_new ∈ X_free and B[n_new] > MWD then
11:            Insert n_new in G
12:            Connect n_new, n_near in G
13:        else
14:            counter− = 1
15:            continue
16:        if Distance(n_new, n_G) < D_goal then
17:            return G
18:    return G
```

*4.3.2 RRT **

A major problem of the standard RRT method is the output path randomness concerning its length. An example is evident in Fig. 25 where the created paths are far from optimal in length. RRT* was proposed in order to tackle this drawback by introducing the length cost concept. Specifically, during the tree's creation, the tree is reordered, aiming for each new node to have a minimum length cost to the tree root. This method is described in Algorithm 8.

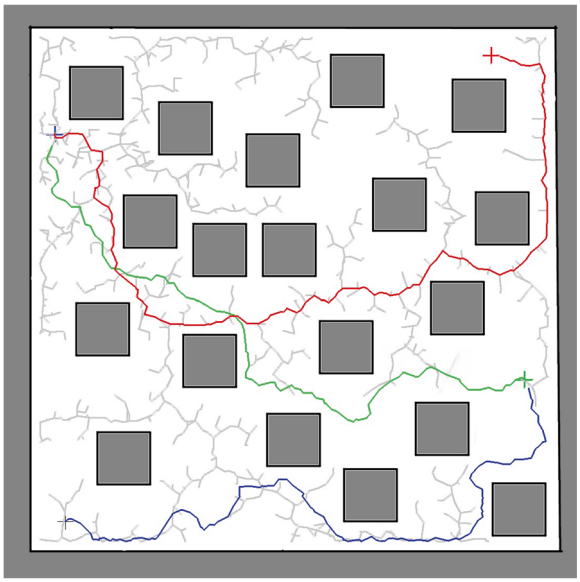

Fig. 25 Example paths using RRT for Environment 3

Algorithm 8 RRT* algorithm

```
1: procedure CreateRRTGraph( )
2:     G = ∅
3:     Insert n_R in G
4:     counter = 0
5:     while counter < K_max and n_G ∉ G do
6:         counter+ = 1
7:         Select n_rand from X_free
8:         Find n_near ∈ G :
               argmin_{n_near}(distance(n_near, n_rand))
9:         Pick n_new =
               Expand(n_near, n_rand, D_exp^max)
10:        if n_new ∈ X_free and B[n_new] > MWD then
11:            Insert n_new in G
12:            for all n_p ∈ G :
                   Distance(n_p, n_new) < D_MinRadius do
13:                if Cost(n_new) > Cost(n_p)
                           +Distance(n_new, n_p) then
14:                    n_nearest = n_p
15:            Parent(n_p) = n_new
16:            for all n_p ∈ G :
                   Distance(n_p, n_new) < D_MinRadius do
17:                if Cost(n_p) > Cost(n_new)
                           +Distance(n_p, n_new) then
18:                    Cost(n_p) =
                           Cost(n_new)+Distance(n_p, n_new)
19:                    Erase n_p, Parent(n_p) connection
20:                    Parent(n_p) = n_new
21:        else
22:            counter− = 1
23:            continue
24:        if Distance(n_new, n_G) < D_goal then
25:            return G
26:    return G
```

As before, a random point n_{rand} is chosen and the closest node $n_{near} \in \mathbf{G}$ is picked (line 8). Then a point n_{new} is selected on the virtual line connecting n_{rand} with n_{near}, located at a random distance $D_r \in [0, D_{max}^{exp}]$ from n_{near} (line 9). In contrast to the standard RRT algorithm, $n_{nearest} \in \mathbf{G}$ is computed in a radius of $D_{MinRadius}$ such that n_{new} has the minimum distance cost to tree root if connected to $n_{nearest}$ (lines 12–14). $D_{MinRadius}$ is the maximum search radius of nodes regarding n_{new}, for connection's redistribution to be considered. So eventually, n_{new}'s father is not selected by minimum distance but by minimum length cost to the tree's root (line 12-15). Finally, for each node existent in a radius of $D_{MinRadius}$ from n_{new}, if its cost is greater than the sum of n_{new}'s cost plus the distance from the specific node to n_{new}, its father's connection is deleted and n_{new} takes the father's place (lines 16–20).

Similarly to the vanilla RRT, $D_{max}^{exp} = 1.5 \cdot D_c$ and since D_{max}^{exp} must be less than $2 \cdot MWD$, $MWD = D_c$. In addition, $D_{MinRadius} = 2 \cdot D_{max}^{exp}$.

An important fact is that this method does not terminate when the RRT reaches n_G, but continues for the same number of iterations needed to converge, in order to improve the path's total length via rearranging the node connections. This fact is described in the RRT* algorithm, but the selection of the number of extra iterations resides on the corresponding developer's judgment.

Algorithm 9 MRRT algorithm

```
1: procedure CreateMRRTGraph( )
2:     Create G_i = ∅, 0 ≤ i < N_RRT
3:     Insert n_R in G_0
4:     Insert n_G in G_1
5:     Select n_{r_i} ∈ X_free, 2 ≤ i < N_RRT
6:     Insert n_{r_i} in G_i
7:     counter = 0
8:     while counter < K_max and G_0, G_1 not merged do
9:         counter+ = 1
10:        Select n_rand from X_free
11:        for i ∈ [0, N_RRT − 1] do
12:            Find n_near^i ∈ G_i :
                   argmin_{n_near^i}(distance(n_near^i, n_rand))
13:            Pick n_new^i = Expand(n_near^i, n_rand, D_exp^max)
14:            if n_new^i ∈ X_free and
                           B[n_new^i] > MWD then
15:                Insert n_new^i in G_i
16:                Connect n_new^i, n_near^i in G_i
17:            else
18:                continue
19:        for all G_i do
20:            for all G_j ≠ G_i do
21:                for all n_p ∈ G_i do
22:                    for all n_q ∈ G_j do
23:                        if distance(n_p, n_q) ≤ D_merge then
24:                            Connect n_p, n_q
25:                            Change parenthood relations in
                                   G_j, starting from n_q
```

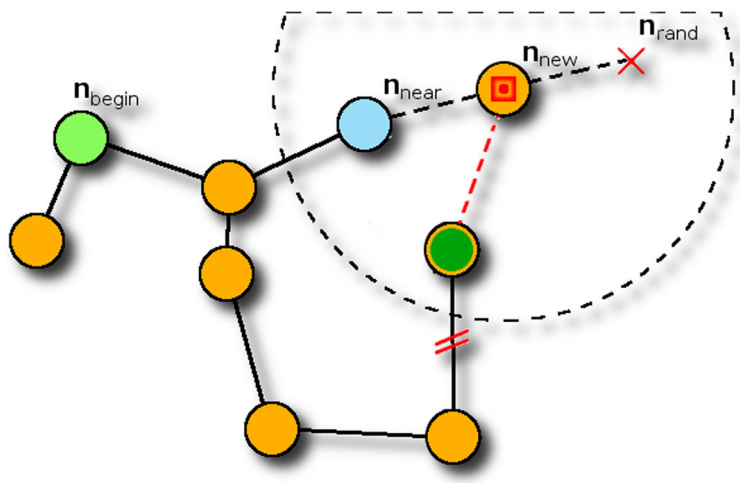

Fig. 26 Example of a RRT* iteration. Here n_{near} becomes father of n_{new} but n_k rearranges its parenthood relation due to lower length cost if connected through n_{new}

In Fig. 26, a method's iteration example is presented and in Figs. 27 and 28 two paths examples created by the RRT* method are depicted.

4.3.3 Multiple RRTs - MRRT

The multiple RRTs method suggests the employment of more than one simultaneously expanded RRTs, in order to increase the algorithmic performance and to overcome expansion issues the standard RRT method has. This method is presented in Algorithm 9. Here, an arbitrary number N_{RRT} of RRTs (each one denoted as $\mathbf{G_i}$) is created, two initiating from the robot's and goal's pose and $N_{RRT} - 2$ initiating from random spatial unoccupied cells (lines 2–6).

As before, a random point n_{rand} is chosen (line 10). Then for all existent trees, the nearest node n^i_{near} is computed, n^i_{new} is created and the connectivity between them is checked (lines 10–18). The method differentiates from the prior RRT algorithms, as the possible merging of trees is investigated in each iteration. If nodes $n_p \in \mathbf{G_i}$ and $n_q \in \mathbf{G_j}$ exist, such that $Dist(n_p, n_q) \leq D_{merge}$, the trees $\mathbf{G_i}$, $\mathbf{G_j}$ are merged (lines 19–25). Obviously, D_{merge} is the maximum distance between two nodes of different trees, in order for these trees to be merged. A problem this approach

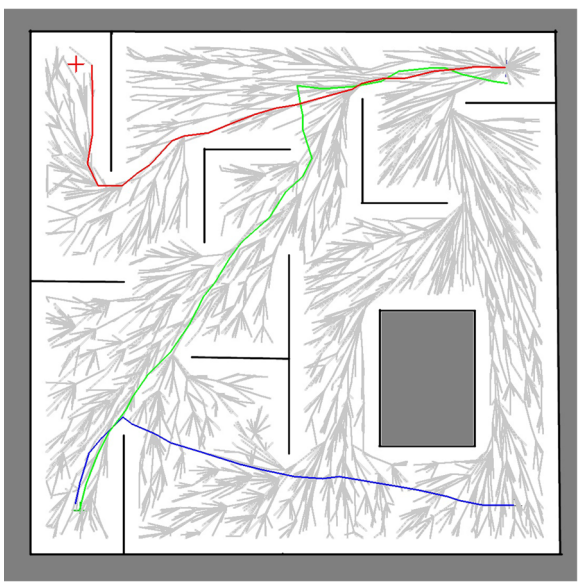

Fig. 27 Example paths using RRT* for Environment 2

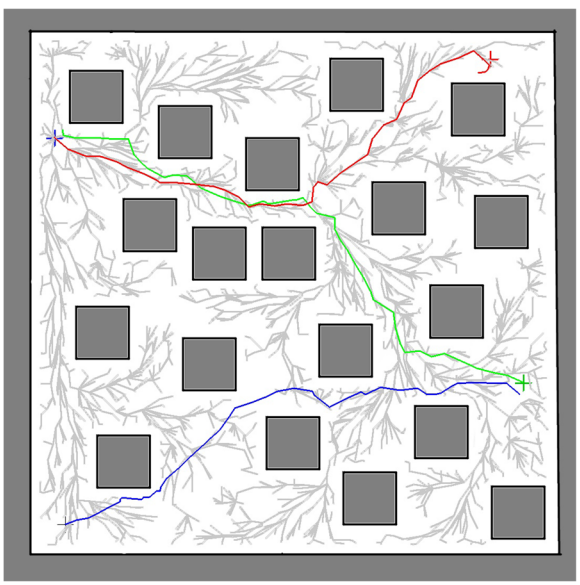

Fig. 28 Example paths using RRT* for Environment 3

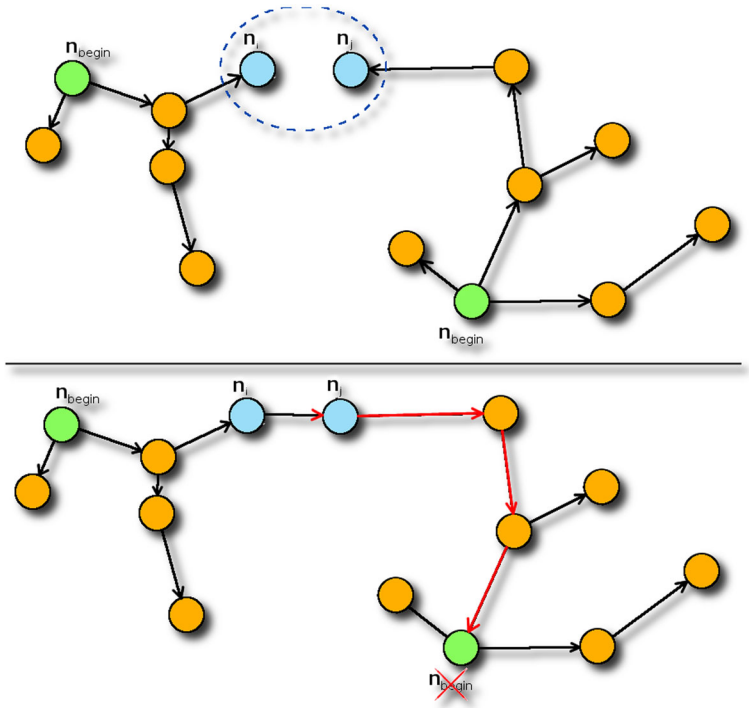

Fig. 29 Procedure of merging two RRTs in order to obtain an RRT-like structure

has is that the merged structure $\mathbf{G_{ij}}$ is not tree-like, as it has two roots. In order to overcome this drawback, the parenthood relations are shifted initiating from n_p (or n_q) and traversing the tree towards the root, i.e. a node's parent becomes its child. This procedure is iterated till the tree's root is reached (line 25). An optical description is presented in Fig. 29. Finally, the algorithm terminates when the trees initiated from the robot and the goal poses are merged, i.e. a path from the robot to the goal exists.

The enhanced version of the algorithm is achieved by automatic value assignment of D_{max}^{exp} and MWD.

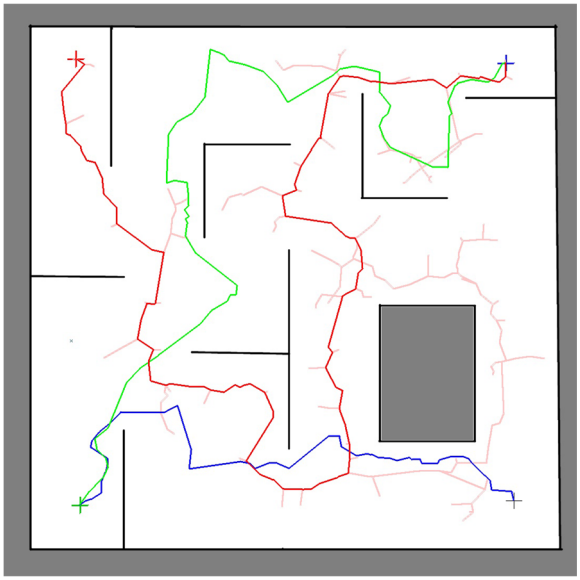

Fig. 30 Example paths using MRRTs for Environment 2

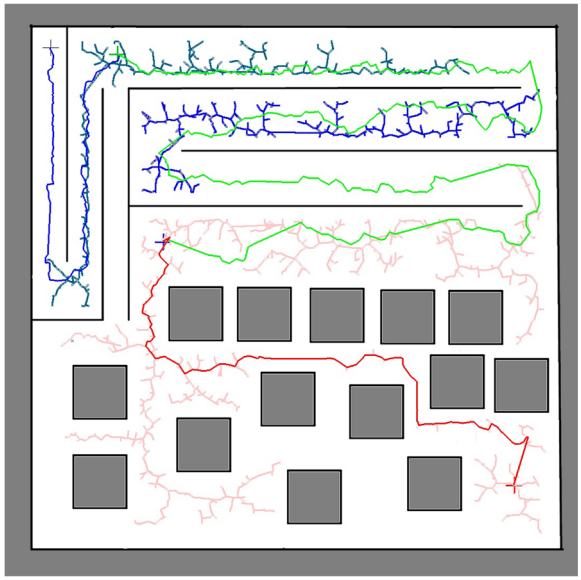

Fig. 31 Example paths using MRRTs for Environment 4

Specifically $D_{max}^{exp} = 1.5 \cdot D_c$ and since D_{max}^{exp} must be less than $2 \cdot MWD$, $MWD = 0.8 \cdot D_c$. Additionally $D_{merge} = D_{max}^{exp}$. Finally, the number of trees N_{RRT} must be specified. In our implementation, we selected to have one tree per 10000 unoccupied cells, i.e. $N_{RRT} = A_{X_{Free}}/10000$. Two path examples created by the multiple RRT method are depicted in Figs. 30 and 31.

4.3.4 Multiple Incremental RRTs - MIRRT

The multiple incremental RRTs extension was created to maintain the already formed tree structures in the environment, in order to use them in future path planning requests. Similarly to the MRRT method, N_{RRT} trees are initially created. After a path planning procedure, we assume that N'_{RRT} trees remain, where $1 \leq N'_{RRT} \leq N_{RRT} - 1$. N'_{RRT} is equal to 1 if all the trees are merged before the path is created and $N'_{RRT} = N_{RRT} - 1$ when the trees initialized from the robot and goal are the first to be merged, thus a path was directly created.

For each new path request, $N_{RRT} - N'_{RRT}$ new trees are constructed in the unoccupied space, in order for the total tree number to be constant and proportional to the unoccupied area. At the same time, the merging conditions are checked during the trees' expansion. The enhanced version of the algorithm automatically assigns values to the same constants as the MRRT method does. Since the paths created from MRRT and MIRRT methods are structurally similar, no figures will be presented here. Additionally, since the only difference between the MRRT and MIRRT is the different number of created trees in every iteration, no need exists for presenting the algorithm.

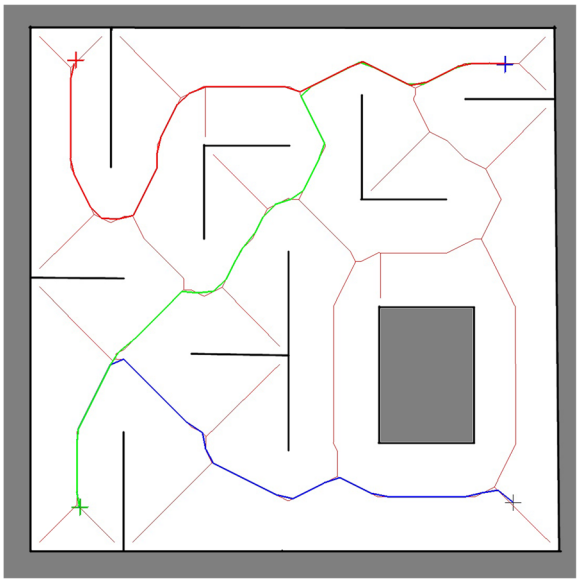

Fig. 33 Example paths using GVD for Environment 2

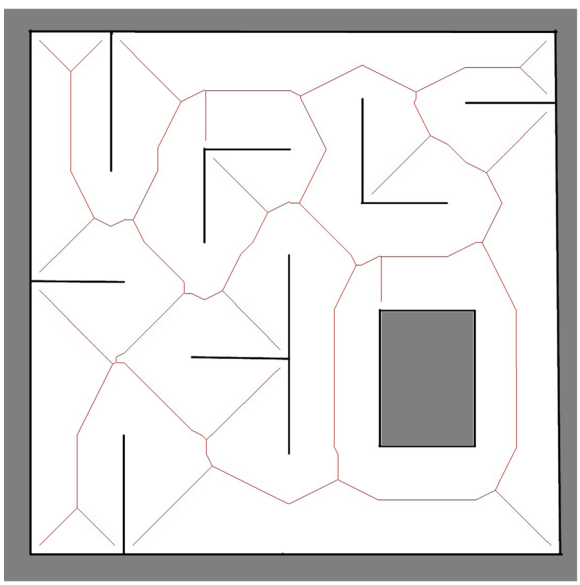

Fig. 32 Example of GVD constructed in Environment 2

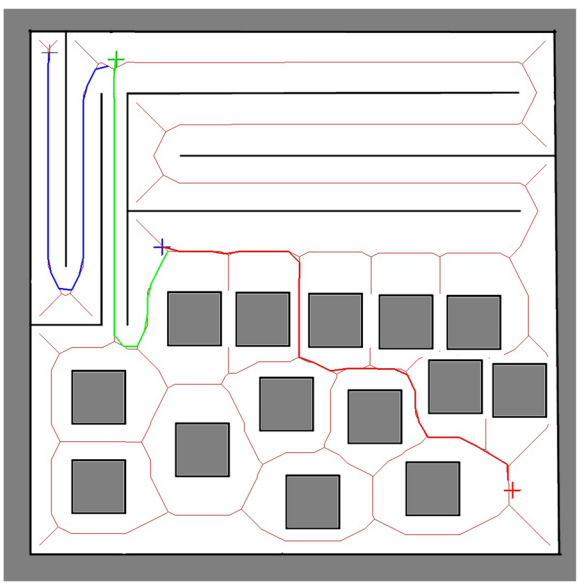

Fig. 34 Example paths using GVD for Environment 2

Table 2 Constant values for simple and enhanced methods for all environments

Method	Constant's Name	Computation	Constant's Value	Env.1 Enh.	Env.2 Enh.	Env.3 Enh.	Env.4 Enh.	Env.5 Enh.
USS, UISS	S_{UNI}	$1.5 \cdot D_c$	25	73.43	31.12	18.1	13.51	8.84
	MWD	$1.2 \cdot D_c$	20	58.74	24.89	14.48	10.81	7.07
	S^{ext}_{UNI}	$2 \cdot D_c$	35	97.9	41.49	24.13	18.02	11.78
RSS	S_{UNI}	$1.5 \cdot D_c$	25	73.43	31.12	18.1	13.51	8.84
	MWD	$1.2 \cdot D_c$	20	58.74	24.89	14.48	10.81	7.07
	R^{ext}_{max}	$2 \cdot D_c$	35	97.9	41.49	24.13	18.02	11.78
	R_{spar}	D_c^2	300	2396	430.3	145.5	81.15	34.69
HSS	S_{UNI}	$1.5 \cdot D_c$	25	73.43	31.12	18.1	13.51	8.84
	MWD	$1.2 \cdot D_c$	20	58.74	24.89	14.48	10.81	7.07
	H^{ext}_{max}	$2 \cdot D_c$	35	97.9	41.49	24.13	18.02	11.78
	H_{spar}	D_c^2	300	2396	430.3	145.5	81.15	34.69
VG	MWD	$1.5 \cdot D_c$	25	73.43	31.12	18.1	13.51	8.84
VGSUS	MWD	$1.5 \cdot D_c$	25	73.43	31.12	18.1	13.51	8.84
	S_{VG}	$7.5 \cdot D_c$	100	367	155.6	90.49	67.56	44.17
RRT	MWD	D_c	20	48.95	20.75	12.07	9.01	5.89
	D^{exp}_{max}	$1.5 \cdot D_c$	30	73.43	31.12	18.1	13.51	8.84
RRT*	MWD	D_c	20	48.95	20.75	12.07	9.01	5.89
	D^{exp}_{max}	$1.5 \cdot D_c$	30	73.43	31.12	18.1	13.51	8.84
	$D_{MinRadius}$	$3 \cdot D_c$	60	146.8	62.24	36.2	27.02	17.67
MRRT,MIRRT	MWD	D_c	20	48.95	20.75	12.07	9.01	5.89
	D^{exp}_{max}	$1.5 \cdot D_c$	30	73.43	31.12	18.1	13.51	8.84
	D_{Merge}	$1.5 \cdot D_c$	30	73.43	31.12	18.1	13.51	8.84
	N_{RRT}	$A_{X_{Free}}/10000$	10	9	9	7	8	5
GVD	–	–	–	–	–	–	–	–

Algorithm 10 Path planning via GVD

1: **procedure** CREATEGVDPATH( )
2: $\mathbf{GVD} = CalculateGVD()$
3: Calculate $P_{in} \in \mathbf{GVD}$: $argmin_{P_{in}}(ManhattanDist(P_{in}, n_R))$
4: Calculate $P_{out} \in \mathbf{GVD}$: $argmin_{P_{out}}(ManhattanDist(P_{out}, n_G))$
5: GVD_{DT} = create distance transformation on $\mathbf{GVD}$ from P_{in} to P_{out}
6: GVD_{path} = back traversal of GVD_{DT} from P_{out} to P_{in}
7: $Path = Line(n_R, P_{in}) + GVD_{path} + Line(n_G, P_{out})$

4.4 Space Skeletonization

The space skeletonization algorithmic family differentiates from the previous three by not using a graph containing the robot and the goal poses. Instead, it performs a space skeletonization which results in a "skeleton", i.e. a continuous line running through the entirety of the unoccupied space. Each point of the line – ideally – has the property of being equidistant to its two nearest obstacles. In the current paper the Generalized Voronoi Diagram method is employed to perform skeletonization. Another common skeletonization method is the application of the thinning morphological operator in the unoccupied space.

4.4.1 Generalized Voronoi Diagram - GVD

The Generalized Voronoi Diagram is computed via the deployment of a Manhattan distance transformation of the unoccupied space. In this work the distance transformation is created utilizing the Brushfire algorithm. Brushfire is an iterative algorithm initiating from a set of points, propagating in a specific space and terminating under some conditions, creating the Manhattan

distance function stored in a container denoted as B. In this work, a Brushfire algorithm initiates from the obstacles applying the 0 value, expands through the unoccupied space by increasing the neighbors' values by one and terminates when all the unoccupied space elements have an updated Brushfire value. The fact that the algorithm operates by updating a set of points that at any time are equidistant to their closest obstacles, can be perceived as a wave-like value propagation. During this propagation, the cells existing where two propagation "waves" collide, participate in the Generalized Voronoi Diagram, as they are equidistant to the obstacles, the two waves initiated from. An example of a GVD is depicted in Fig. 32.

The path planning procedure is described in Algorithm 10. Obviously, a part of the GVD will be used as the path from the robot pose (n_R) to the goal (n_G). Since n_R, n_G are generally not elements of the GVD, the cells closest to them that belong on the GVD must be computed. These are denoted as P_{in} and P_{out}, the first being the entry point to the GVD and the second the exit. In order to efficiently calculate these points, a Brushfire algorithm initiates from n_R and n_G, propagate through the unoccupied space and terminate when a GVD's element is visited (lines 3–4). Then a Brushfire algorithm is applied from P_{in} to P_{out} via GVD only, which if backtracked produces the desirable path (lines 5–6). Finally, the total path consists of a straight line segment between n_R and P_{in}, the GVD part from P_{in} to P_{out} and the straight line segment from P_{out} to n_G (line 7). Since the GVD path planning method does not contain any parameters, no enhanced version of the algorithm is possible. Two path examples produced by the GVD method are depicted in Figs. 33 and 34.

Table 3 Experimental results for simple and enhanced versions of algorithms for Environment 1

Environment/Method	t_m	RT	D_m	RD	RC	ETT	SR
Env 1 - Simple USS	14.34	3.87 %	826.86	0 %	5.91 %	248.35	100 %
Env 1 - Enhanced USS	0.41	17.46 %	819.39	0 %	10.13 %	246.32	100 %
Env 1 - Simple RSS	75.92	2.6 %	817.38	0.22 %	7.52 %	245.59	100 %
Env 1 - Enhanced RSS	1.89	11.28 %	831.39	1.06 %	13.57 %	250.09	100 %
Env 1 - Simple HSS	42.97	2.69 %	836.77	0 %	11.26 %	251.59	100 %
Env 1 - Enhanced HSS	1.12	10.69 %	816.8	0 %	13.19 %	245.7	100 %
Env 1 - Simple UISS	0.97	12.24 %	824.28	0 %	6.45 %	247.61	100 %
Env 1 - Enhanced UISS	0.11	48.93 %	820.49	0 %	9.71 %	246.63	100 %
Env 1 - Simple VG	7.01	8.74 %	804.02	0 %	3 %	241.36	100 %
Env 1 - Enhanced VG	6.58	5.4 %	804.02	0 %	3.02 %	241.35	100 %
Env 1 - Simple VGSS	4501.1	3.65 %	804.02	0 %	3 %	241.36	100 %
Env 1 - Enhanced VGSS	21.35	6.87 %	804.02	0 %	3.02 %	241.36	100 %
Env 1 - Simple RRT	71.41	18 %	1074.53	5.35 %	20.01 %	323.36	100 %
Env 1 - Enhanced RRT	62.84	11.6 %	1090.03	5.81 %	21.12 %	328.06	100 %
Env 1 - Simple RRT*	99.37	46.25 %	849.05	3.02 %	12.51 %	255.34	100 %
Env 1 - Enhanced RRT*	75.94	4.58 %	788.92	2.64 %	12.47 %	237.3	100 %
Env 1 - Simple MRRT	5.06	22.17 %	1554.13	8.72 %	29.25 %	467.70	100 %
Env 1 - Enhanced MRRT	3.73	11.08 %	1727.88	12.69 %	33.1 %	520.02	100 %
Env 1 - Simple MIRRT	8.43	27.16 %	1790.97	26.17 %	31.29 %	538.86	100 %
Env 1 - Enhanced MIRRT	5.98	25.57 %	1893.22	14.61 %	34.84 %	569.71	100 %
Env 1 - GVD	54.47	10.31 %	967.65	0 %	6.5 %	290.62	100 %

5 Experimental Results

In this section the experimental results for all methods and environments are presented. Specifically, in Table 2 the values of each parameter used in the methods (both originals and enhanced), are provided for all environments. Additionally in Tables 3, 4, 5, 6 and 7 the experimental results are depicted.

As the amount of information is quite large for an effortless evaluation, an attempt will be made to create a rational way to automatically compare the methods' performance. The way to do this is to group the different environments' results into a single unified representation. For that reason, the mean value of metrics for all environments is calculated. If a method fails to produce results in a specific environment, its metrics are assigned the value of the worst method in the specific environment. Next, the metrics that do not come in a percentage form are normalized to [1, inf) by dividing them by the minimum value of the specific metric. The results are presented in Table 8. It should be noted that the mean value of metric M_i is denoted as $\bar{M}_i$ and the mean normalized value as $\hat{\bar{M}}_i$.

In order to decide which path planning method has the best performance when applied in occupancy grid maps, the performance weight W_p is introduced, consisting of three separate parts whose values are bound in [0, 1], in order for the final coefficient to be also bound in [0, 1]. The first part, denoted as P_c, is a performance coefficient. P_c takes under consideration the method's execution time, the mean path distance and the rotation coefficient, as they are crucial metrics for the method to be efficiently used. Instead of using $\hat{\bar{t}}_m$ and $\bar{RC}$ separately, the ETT can be utilized, as it combines them. Of course, the preferred values of both $\hat{\bar{t}}_m$ and $\bar{RC}$ should be low and specifically equal to unity. Thus the use of the inverse value of $\hat{\bar{t}}_m$ and $\bar{ETT}$ was selected, in order to achieve a high value

Table 4 Experimental results for simple and enhanced versions of algorithms for Environment 2

Environment/Method	t_m	RT	D_m	RD	RC	ETT	SR
Env 2 - Simple USS	10.63	4.9 %	1148.42	0 %	8 %	344.93	100 %
Env 2 - Enhanced USS	2.87	9.43 %	1131.55	0 %	10.07 %	339.97	100 %
Env 2 - Simple RSS	79.27	1.43 %	1092.12	0.62 %	8.78 %	328.07	100 %
Env 2 - Enhanced RSS	13.74	3.87 %	1145.83	1.96 %	11.48 %	344.32	100 %
Env 2 - Simple HSS	30.65	2.29 %	1129.17	0 %	12.77 %	339.39	100 %
Env 2 - Enhanced HSS	13.35	5.13 %	1129.09	0 %	10.63 %	339.26	100 %
Env 2 - Simple UISS	1.4	10.99 %	1149.49	0 %	8.57 %	345.28	100 %
Env 2 - Enhanced UISS	0.57	9.26 %	1153.1	0 %	8.87 %	346.37	100 %
Env 2 - Simple VG	–	–	–	–	–	–	–
Env 2 - Enhanced VG	–	–	-	–	–	–	–
Env 2 - Simple VGSS	1036.19	3.43 %	1154.33	0 %	4.93 %	346.55	100 %
Env 2 - Enhanced VGSS	193.12	2.51 %	1211.75	0 %	4.98 %	363.77	100 %
Env 2 - Simple RRT	199.9	27.46 %	1501.45	7.68 %	19.19 %	451.39	100 %
Env 2 - Enhanced RRT	113.29	22.82 %	1539.67	7.89 %	20.07 %	462.91	100 %
Env 2 - Simple RRT*	759.58	59.17 %	1104	2.70 %	12.94 %	331.85	100 %
Env 2 - Enhanced RRT*	285.23	44.24 %	1186.32	5.41 %	13.05 %	356.55	100 %
Env 2 - Simple MRRT	10.45	24.54 %	2238.91	12.02 %	26.34 %	672.99	100 %
Env 2 - Enhanced MRRT	6.89	20.91 %	2495.98	18.69 %	27.62 %	750.17	100 %
Env 2 - Simple MIRRT	17.87	47.73 %	2306.8	12.20 %	28.41 %	693.46	100 %
Env 2 - Enhanced MIRRT	12.65	110.73 %	2291.14	8.73 %	29.56 %	688.82	100 %
Env 2 - GVD	52.72	3.32 %	1292.69	0 %	9.21 %	388.29	100 %

Table 5 Experimental results for simple and enhanced versions of algorithms for Environment 3

Environment/Method	t_m	RT	D_m	RD	RC	ETT	SR
Env 3 - Simple USS	6.14	5.18 %	1017.63	0 %	8.69 %	305.72	100 %
Env 3 - Enhanced USS	12.42	4.43 %	1004.52	0 %	6.93 %	301.7	100 %
Env 3 - Simple RSS	80.22	2.77 %	978.84	0.65 %	7.23 %	294.01	100 %
Env 3 - Enhanced RSS	85.57	2.34 %	989.48	1 %	8.87 %	297.29	100 %
Env 3 - Simple HSS	16.47	3.29 %	1042.18	0 %	13 %	313.30	100 %
Env 3 - Enhanced HSS	71.53	1.53 %	993.01	0 %	10.87 %	298.45	100 %
Env 3 - Simple UISS	0.75	11.86 %	1023.36	0 %	8.04 %	307.41	100 %
Env 3 - Enhanced UISS	1.32	10.43 %	1000.29	0 %	7.85 %	300.48	100 %
Env 3 - Simple VG	27.17	11.52 %	1670.45	0 %	6.42 %	501.62	66.67 %
Env 3 - Enhanced VG	46.03	8.94 %	1080.22	0 %	5.08 %	324.45	66.67 %
Env 3 - Simple VGSS	395.02	2.75 %	1192.22	0 %	4.62 %	357.9	100 %
Env 3 - Enhanced VGSS	1071.16	3.09 %	1001.65	0 %	4.75 %	300.73	100 %
Env 3 - Simple RRT	101.51	17.67 %	1294.14	7.42 %	19.49 %	389.22	100 %
Env 3 - Enhanced RRT	84.64	14.71 %	1387.2	8.97 %	20.39 %	417.18	100 %
Env 3 - Simple RRT*	218.95	55.85 %	1040.19	3.62 %	12.55 %	312.68	100 %
Env 3 - Enhanced RRT*	260.75	46.3 %	1178.09	7.08 %	14.97 %	354.18	100 %
Env 3 - Simple MRRT	9.42	17.03 %	1797.69	14.11 %	24.28 %	540.52	100 %
Env 3 - Enhanced MRRT	13.14	19.34 %	1663.6	13.11 %	24.68 %	500.31	100 %
Env 3 - Simple MIRRT	7.78	29.03 %	1800.97	15.79 %	26.35 %	541.61	100 %
Env 3 - Enhanced MIRRT	11.76	74.78 %	1739.95	13.53 %	25.16 %	523.24	100 %
Env 3 - GVD	53.39	3.85 %	1036.19	0 %	10.75 %	311.39	100 %

when the two metrics are equal (or close) to one. Since the desired behavior of P_c is to be increased when the corresponding method is both execution and path efficient, P_c is formulated using the statistical measure F1 score ($F1 = 2 \cdot \frac{a \cdot b}{a+b}$). F1 is bounded to [0, 1] and produces high values when both of its variables are high. The two metrics of F1 should have, if not an identical, an almost equal range. From the experimental results of Table 8, $\hat{\bar{t}}_m$ is bounded to $[1, 1252.06]$ and $E\bar{T}T$ in $[1, 1.835]$. For them to be used in F1, $E\bar{T}T$ must be scaled down to an approximate range of $[1, 2]$. For this reason, instead of $\hat{\bar{t}}_m$, $\hat{\bar{t}}^{0.1}_m$ is used, with a range of $[1, 2.09]$. Finally P_c is computed by the following formula:

$$P_c = 2 \cdot \frac{\frac{1}{\hat{\bar{t}}^{0.1}_m} \cdot \frac{1}{E\bar{T}T}}{\frac{1}{\hat{\bar{t}}^{0.1}_m} + \frac{1}{E\bar{T}T}} \tag{11}$$

The second part of W_p is a measure of the methods' reliability and repetitiveness, i.e. the R_c. For that reason the $R\bar{T}$ and $R\bar{D}$ metrics will be used. Since the maximum value of 1 is desired for R_c, its form must be constructed in a way that when $R\bar{T}$ and $R\bar{D}$ increase, R_c will decrease and when $R\bar{T} = R\bar{D} = 0$ the R_c value will be equal to 1. The following formula is used for the computation of R_c:

$$R_c = \frac{1}{1 + R\bar{T} + R\bar{D}} \tag{12}$$

Finally, the third coefficient S_c is a measure of success, thus $S\bar{R}$ can be directly used. Since this coefficient is of major importance, instead of $S\bar{R}$, $S\bar{R}^2$ will be used:

$$S_c = S\bar{R}^2 \tag{13}$$

Table 6 Experimental results for simple and enhanced versions of algorithms for Environment 4

Environment/Method	t_m	RT	D_m	RD	RC	ETT	SR
Env 4 - Simple USS	4.51	4.44 %	1691.32	0 %	5.96 %	507.69	100 %
Env 4 - Enhanced USS	47.11	2.41 %	867.34	0 %	6.33 %	260.52	100 %
Env 4 - Simple RSS	80.44	3.08 %	1651.56	6.67 %	7.33 %	492.43	96.67 %
Env 4 - Enhanced RSS	243.59	1.62 %	1371.71	27.56 %	9.15 %	411.92	96.67 %
Env 4 - Simple HSS	13.96	4.22 %	2159.05	0 %	13.19 %	648.7	66.67 %
Env 4 - Enhanced HSS	246.91	11.28 %	1651.46	0 %	11.28 %	496	100 %
Env 4 - Simple UISS	0.61	13.68 %	1674.15	0 %	5.71 %	502.53	100 %
Env 4 - Enhanced UISS	2.05	8.98 %	855.84	0 %	6.32 %	257.06	100 %
Env 4 - Simple VG	–	–	–	–	–	–	–
Env 4 - Enhanced VG	–	–	–	–	–	–	–
Env 4 - Simple VGSS	–	–	–	–	–	–	–
Env 4 - Enhanced VGSS	3613.32	3.43 %	936.07	0 %	5.76 %	281.11	100 %
Env 4 - Simple RRT	999.38	67.81 %	1457.34	33.44 %	17.38 %	438.07	100 %
Env 4 - Enhanced RRT	890.14	20.91 %	2587.55	2.9 %	21.8 %	777.9	66.67 %
Env 4 - Simple RRT*	5304.37	86.73 %	1197.92	30.13 %	16.31 %	360.19	100 %
Env 4 - Enhanced RRT*	15214.6	122.31 %	1256.1	24.59 %	15.78 %	377.62	100 %
Env 4 - Simple MRRT	495.22	49.11 %	2137.4	4.96 %	22.66 %	642.35	100 %
Env 4 - Enhanced MRRT	3112.36	55.15 %	2126.3	6.35 %	21.09 %	637.44	98.33 %
Env 4 - Simple MIRRT	265.09	82.74 %	2269.37	6.44 %	22.62 %	679.75	98.33 %
Env 4 - Enhanced MIRRT	1478.82	65.82 %	2247.17	9.59 %	21.57 %	669.17	96.67 %
Env 4 - GVD	54.68	6 %	897.26	0 %	9.39 %	269.65	100 %

Conclusively, the performance weight W_p is calculated as follows:

$$W_p = P_c \cdot R_c \cdot S_c = 2 \cdot \frac{\frac{1}{\hat{t}_m^{0.1}} \cdot \frac{1}{\bar{ETT}}}{\frac{1}{\hat{t}_m^{0.1}} + \frac{1}{\bar{ETT}}} \cdot \frac{1}{1 + \bar{RT} + \bar{RD}} \cdot \bar{SR}^2 \tag{14}$$

The performance metric for all methods, as well as their rank are presented in Table 9.

Comments for each path planning method follow. Initially, the simple and enhanced USS showcased relatively small RT and RD values and their SRs are 80 % and 100 % respectively. These results indicate that the USS is a stable and trustworthy method for deployment in OGMs. Specifically, the enhanced USS has a normalized mean value of D_m equal to 1, indicating that (in average) this method achieved the minimum path, in comparison to all others. Additionally, the small RC value proves that the produced paths contain no redundant turns. On the other hand, RSS methods achieved similar results regarding SR but had much worse RDs, due to the methods' random nature. Additionally, RC is slightly worse than USS's, as the samples are not uniformly dispersed in space. The simple RSS execution times are lower than simple USS's, something not evident in their enhanced versions, since the enhanced RSS method requires more time in obstacle dense environments for creating the initial graph due to the random sampling. HSS, the third space sampling method, achieved similar results to the previous two methods. It should be noted that the RD metric is 0 (or close to 0 in simple HSS), as the method does not create random, but pseudo-random samples, resulting in identical paths. RC and ETT are slightly elevated in the simple HSS case, something fixed in the enhanced HSS. Finally, as UISS is concerned, the results are almost identical to the ones of the USS method. The only difference is detected in the t_m and RT metrics. Specifically, the enhanced

Table 7 Experimental results for simple and enhanced versions of algorithms for Environment 5

Environment/Method	t_m	RT	D_m	RD	RC	ETT	SR
Env 5 - Simple USS	–	–	–	–	–	–	–
Env 5 - Enhanced USS	83.32	2.89 %	4721.9	0 %	4.31 %	472.41	100 %
Env 5 - Simple RSS	45.8	0 %	4878.26	0 %	11.01 %	486.53	5 %
Env 5 - Enhanced RSS	624.47	1.26 %	4696.86	0.59 %	8.97 %	470.13	100 %
Env 5 - Simple HSS	–	–	–	–	–	–	–
Env 5 - Enhanced HSS	473.52	1.24 %	4741.95	0 %	11.74 %	474.78	100 %
Env 5 - Simple UISS	–	–	–	–	–	–	–
Env 5 - Enhanced UISS	9.47	4.88 %	4752.06	0 %	4.34 %	475.42	100 %
Env 5 - Simple VG	–	–	–	–	–	–	–
Env 5 - Enhanced VG	–	–	–	–	–	–	–
Env 5 - Simple VGSS	–	–	–	–	–	–	–
Env 5 - Enhanced VGSS	–	–	–	–	–	–	–
Env 5 - Simple RRT	–	–	–	–	–	–	–
Env 5 - Enhanced RRT	–	–	–	–	–	–	–
Env 5 - Simple RRT*	–	–	–	–	–	–	–
Env 5 - Enhanced RRT*	–	–	–	–	–	–	–
Env 5 - Simple MRRT	–	–	–	–	–	–	–
Env 5 - Enhanced MRRT	–	–	–	–	–	–	–
Env 5 - Simple MIRRT	–	–	–	–	–	–	–
Env 5 - Enhanced MIRRT	–	–	–	–	–	–	–
Env 5 - GVD	69.86	29.81 %	5203.62	0 %	11.94 %	520.95	100 %

UISS achieved the lowest execution time comparing to the other methods, fact that in conjunction to the 100 % SR and the very low ETT, predispose positively about the method's inferiority. The increased RT values can be explained by the extremely small execution times.

As far as VGs are concerned, the results are disappointing. Both simple and enhanced versions resulted in high execution times and large path distances, elevated RC and consequently high estimated time of traversal. The worst is that the method succeeded only once in three times. The enhanced method's inferiority in comparison to the simple one is mainly detected in the ETT's slight decrease. The VG's failure can be attributed to the selected environments, as they are not purely convex. On the other hand, the incorporation of a sparse uniform sampling in VG improved the results. Except from the simple VGs, where the execution times were raised, D_m, RT, RD and ETT decreased by far. Furthermore, the success rate almost doubled, increasing from 33.33 % to 60 %. The enhanced version of VGSS led to much better results in all metrics when compared to the simple one. Specifically the execution times decreased to 1/4 of the initial value, the path distance and RC were lower (fact that decreased the ETT) and the success rate increased to 80 %. It must be stated that the ETT's normalized mean is close to 1 and therefore the produced paths approach the optimal solution.

Proceeding to the RRTs, the simple RRT method had higher t_m than the PRMs but lower than VGs. Additionally the success rate has reached 80 %. An important observation is that for the first time RC surpasses 15 %, something expected due to the tree's structure. Additionally, even though the enhanced RRT achieved slightly lower execution times, it failed to increase the success rate and decrease ETT. The RRT* method reached an 80 % success rate both in the simple and enhanced form. Its specific characteristic is the high execution times needed for the path improvement step, as described in paragraph 4.3.2, though the values were excessive in

Table 8 Normalized mean values of metrics on all five environments

Method	$\hat{\bar{t}}_m$	$\bar{RT}$	$\hat{\bar{D}}_m$	$\bar{RD}$	$\bar{RC}$	$\hat{\bar{ETT}}$	$\bar{SR}$
Simple USS	48.82	9.64 %	1.157	0.118 %	8.1 %	1.189	80 %
Enhanced USS	10.81	7.32 %	1	0 %	7.55 %	1	100 %
Simple RSS	26.75	1.97 %	1.102	1.632 %	8.37 %	1.139	80.33 %
Enhanced RSS	71.69	4.07 %	1.057	6.434 %	10.41 %	1.094	99.33 %
Simple HSS	53.88	8.46 %	1.214	0.118 %	12.43 %	1.279	73.33 %
Enhanced HSS	59.65	5.97 %	1.092	0 %	11.54 %	1.144	100 %
Simple UISS	46.46	15.72 %	1.156	0.118 %	8.142 %	1.187	80 %
Enhanced UISS	1	16.49 %	1.004	0 %	7.418 %	1.003	100 %
Simple VG	1250.69	56.62 %	1.585	10.54 %	14.716 %	1.722	33.33 %
Enhanced VG	1252.06	55.44 %	1.516	10.54 %	14.452 %	1.613	33.33 %
Simple VGSS	1610.31	32.39 %	1.372	6.81 %	9.43 %	1.385	60 %
Enhanced VGSS	408.54	9.142 %	1.072	0.118 %	6.09 %	1.054	80 %
Simple RRT	147.68	32.15 %	1.232	10.896 %	15.6 %	1.309	80 %
Enhanced RRT	131.32	19.97 %	1.382	5.232 %	19.06 %	1.547	73.33 %
Simple RRT*	518.25	55.56 %	1.099	8.012 %	13.25 %	1.099	80 %
Enhanced RRT*	1217.53	49.45 %	1.125	8.062 %	13.64 %	1.139	80 %
Simple MRRT	84.66	28.53 %	1.513	8.08 %	22.89 %	1.755	80 %
Enhanced MRRT	278.15	27.26 %	1.547	10.28 %	23.68 %	1.807	79.67 %
Simple MIRRT	68.32	43.29 %	1.694	12.24 %	24.12 %	1.835	79.67 %
Enhanced MIRRT	157.82	61.34 %	1.565	9.41 %	24.61 %	1.833	79.33 %
GVD	21.089	10.66 %	1.099	0 %	9.558 %	1.099	100 %

the enhanced approach. Nevertheless, t_m's increase assisted towards constructing a nearly-optimal path, as can be derived from the D_m and ETT metrics. As far as MRRTs are concerned, we initially observe that the execution times decreased in comparison to RRT and RRT* (at least in their simple forms), something expected. The SRs were kept to a 80 % level but RT, RD and RC were increased. Furthermore, the mean path length was quite high and inductively EET increased too. The high values of RT and RD can be explained if we consider the method's randomness and the high values of D_m and RC exist due to the new structure characteristics, produced by the merging of different RRTs. Specifically, the final structure is highly "unstructured" regarding its creation due to the randomness of the individual RRTs merging. The same comments apply for the MIRRTs, even though the execution time decreased compared to MRRTs.

Finally, as the space skeletonization method is concerned, GVD had almost constant execution times for all environments, indicative of the method's independence to the environmental obstacle density. GVD's t_m is quite lower than RRTs, VGs and some of the PRM algorithms. Additionally $RD = 0$, showcasing the method's stability regarding the produced path's distance. Its two more noteworthy advantages are the 100 % success rate, as well as the low estimated time of traversal resulted from the low RC and D_m values. On the other hand, the method's main drawback is the increased D_m in cases of low environmental obstacle density, as the produced GVD is created in the middle of the unoccupied space.

Generally, the evaluation coefficients showcase the fact that the Space Sampling methods are the best choice for path planning in two dimensional occupancy grid maps. Specifically, the enhanced uniform incremental space sampling achieved the 1^{st} rank among all methods, as it has a success rate of 100 %, small path length and no unnecessary path turns. In the 2^{nd}, 4^{th} and 5^{th} place, the enhanced versions of Uniform, Random and Halton space sampling

Table 9 Evaluation coefficients and rank of path planning methods

Method	P_c	R_c	S_c	W_p	$Rank$
Enhanced UISS	0.9984	0.8584	1.0000	0.8571	1^{st}
Enhanced USS	0.8815	0.9318	1.0000	0.8214	2^{nd}
GVD	0.8146	0.9037	1.0000	0.7362	3^{rd}
Enhanced HSS	0.7550	0.9436	1.0000	0.7124	4^{th}
Enhanced RSS	0.7612	0.9049	0.9867	0.6797	5^{th}
Simple RSS	0.7910	0.9642	0.6454	0.4927	6^{th}
Simple USS	0.7506	0.9111	0.6400	0.4377	7^{th}
Simple UISS	0.7534	0.8633	0.6400	0.4162	8^{th}
Enhanced VGSS	0.6949	0.9152	0.6400	0.4070	9^{th}
Simple HSS	0.7222	0.9210	0.5378	0.3577	10^{th}
Simple RRT	0.6762	0.6990	0.6400	0.3025	11^{th}
Simple MRRT	0.6036	0.7320	0.6400	0.2828	12^{th}
Enhanced RRT	0.6299	0.7987	0.5378	0.2705	13^{th}
Simple RRT*	0.6741	0.6113	0.6400	0.2637	14^{th}
Enhanced MRRT	0.5614	0.7270	0.6347	0.2590	15^{th}
Enhanced RRT*	0.6301	0.6349	0.6400	0.2560	16^{th}
Simple MIRRT	0.5951	0.6430	0.6347	0.2428	17^{th}
Enhanced MIRRT	0.5727	0.5856	0.6294	0.2111	18^{th}
Simple VGSS	0.5751	0.7184	0.3600	0.1487	19^{th}
Enhanced VG	0.5474	0.6025	0.1111	0.0366	20^{th}
Simple VG	0.5315	0.5982	0.1111	0.0353	21^{st}

follow. This fact indicates that the enhanced versions increased the methods' potential of creating efficient paths by far. It should be noted that GVD holds the 3^{rd} place as it is an extremely stable method with 100 % success rate. It must also be noted that enhanced VGs were placed in the 9^{th} position due to their low path distance and rotation coefficient. From 11^{th} to 18^{th} place, the RRT methods exist. In these cases, the enhanced methods did not manage to improve the results. The main reason is that the R_c coefficient decreased the RRTs expansion distance according to the environmental obstacle density. Thus, the RRT expansion itself was significantly delayed, as more iterations were needed towards for the algorithm's convergence, i.e. to create the path from the robot pose to the goal. Finally, VGs hold the lowest rank, as their SR was very low and the path length quite high.

6 Conclusions - Future Work

In this paper, path planning methods for deployment on two dimensional Occupancy Grid Maps were presented. The methods described belonged to four heterogeneous algorithmic families, the Space Sampling (PRMs), the Visibility Graphs, the Rapidly exploring Random Trees and the Space skeletonization. The algorithms were tested in five OGMs with different characteristics and seven performance and quality metrics were used. The contributions of this paper include the definition of several path planning benchmarking metrics, several path planning methods' comparison, and their enhancement via a environmental obstacle density coefficient. Furthermore, some novelties were introduced regarding in specific methods (such as UISS) which accelerated their execution, while preserving their quality or results.

The results indicated that the Space sampling methods are the optimal choice for performing path planning procedures in two dimensional OGMs, since they had a great success rate, small execution times and the paths produced were short in length and lacked excessive turns. The space skeletonization method (GVD) produced great results, as it had a 100 % success rate and constructed efficient paths, though its

use is not suggested in obstacle free environments. Additionally, the RRT family resulted in medium quality results, specially in obstacle dense spaces. Thus it can be inferred that the RRTs are not optimal for path creation in two-dimensional OGMs, in opposition with high dimensional spaces, where RRTs are primarily used. Finally, Visibility Graphs resulted in poor quality paths and low success rates, as the experimental environments were not sparse, nor purely convex. The introduction of the density coefficient D_c enhanced the efficiency of the Space Sampling and Visibility Graphs algorithms, but resulted in quality reduction in the RRTs cases. Finally, the performance weight (W_p) succeeded in objectively evaluate the methods, as the final weights were in compliance with both the experimental results, as well as the intuitive evaluation performed by observation.

Regarding future work, many expansions in this paper can be made. Initially, the survey can be expanded to three, or even N dimensional spaces, where the specific methods' properties change. Furthermore, the algorithms can be tested in a greater variety of environments, some of which should have different obstacle densities presented in the same areas. This would affect the enhanced methods' approach, as new and more intelligent metrics should be introduced. A way to eliminate our major assumption (and at the same time limitation) concerning the environmental obstacle homogeneity, would be to preprocess the map at hand and extract local Density Coefficients, in order for the methods to have different properties in each region.

Also it would be useful to introduce alteration procedures like final path smoothing in the PRMs case or create the RRTs using motion constraints, in order to expand the survey. Finally, different approach must be followed in the VG and RRT families in order for the enhanced methods to produce much better results in comparison to the standard ones.

References

1. Lozano-Prez, T., Wesley, M.A.: An algorithm for planning collision-free paths among polyhedral obstacles. Commun. ACM **22**(10), 560–570 (1979)
2. Ghosh, S.K.: Visibility Algorithms in the Plane, ISBN:9780521875745. Cambridge University Press (2007)
3. Ghosh, S.K., Goswami, P.P.: Unsolved problems in visibility graphs of points, segments and polygons. In: Proceedings of India-Taiwan Conference on Discrete Mathematics, Taipei, pp. 44–54 (2009)
4. Kim, J., Kim, M., Kim, D.: Variants of the quantized visibility graph for efficient path planning. Adv. Robot. **25**(18), 2341–2360 (2011)
5. Bhattacharya, P., Gavrilova, M.L.: Voronoi diagram in optimal path planning. In: 4th International Symposium on Voronoi Diagrams in Science and Engineering, 2007. ISVD '07, pp. 38–47
6. Garrido, S., Moreno, L., Abderrahim, M., Martin, F.: Path Planning for Mobile Robot Navigation using Voronoi Diagram and Fast Marching. In: 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2376–2381 (2006)
7. Ok, K.-C., Ansari, S., Gallagher, B., Sica, W., Dellaert, F., Stilman, M.: Path planning with uncertainty: Voronoi uncertainty fields. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), pp. 4596-4601. IEEE (2013)
8. Geraerts, R., Overmars, M.H.: A comparative study of probabilistic roadmap planners. In: Workshop on the algorithmic foundations of robotics, pp. 43–57 (2002)
9. Geraerts, R.J., Overmars, M.H.A.: Sampling Techniques for Probabilistic Roadmap Planners. Intelligent Autonomous Systems **8**, 600–609 (2004)
10. Kavraki, L.E., Latombe, J.-C.: Probabilistic Roadmaps for Robot Path Planning, Practical Motion Planning in Robotics: Current Approaches and Future Directions, pp. 33–53. Wiley (1998)
11. Kavraki, L.E., Kolountzakis, M.N., Latombe, J.-C.: Analysis of probabilistic roadmaps for path planning. IEEE Trans. Robot. Autom. **14**(1), 166–171 (1998)
12. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation **12**(4), 566–580 (1996)
13. Laumond, J.-P., Nissoux, C.: Visibility-based probabilistic roadmaps for motion planning. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 1999. IROS '99, vol. 3, pp. 1316–1321 (1999)
14. Bohlin, R., Kavraki, L.E.: Path planning using lazy PRM. In: Proceedings of the IEEE International Conference on Robotics and Automation, 2000. ICRA '00, vol. 1, pp. 521–528 (2000)
15. Hsu, D., Sanchez-Ante, G., Sun, Z.: Hybrid PRM Sampling with a Cost-Sensitive Adaptive Strategy. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation 2005, ICRA 2005, pp. 3874–3880
16. Karaman, S., Frazzoli, E.: Sampling-based algorithms for optimal motion planning. Int. J. Robot. Res. **30**(7), 846–894 (2011)
17. Marble, J.D., Bekris, K.E.: Asymptotically near-optimal is good enough for motion planning. In: International Symposium on Robotics Research (2011)
18. Marble, J.D., Bekris, K.E.: Towards small asymptotically near-optimal roadmaps. IEEE (2012)
19. Li, Y., Li, D., Maple, C., Yue, Y., Oyekan, J.: K-Order Surrounding roadmaps path planner for robot path planning. J. Intell. Robot. Syst. **75**(3-4), 493–516 (2014)

20. LaValle, S.M.: Rapidly-exploring random trees: A New Tool for Path Planning. TR 98–11, Computer Science Dept., Iowa State University (1998)
21. Bruce, J., Veloso, M.: Real-time randomized path planning for robot navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 3, pp. 2383–2388 (2002)
22. Martin, S.R., Wright, S.E., Sheppard, J.W.: Offline and Online Evolutionary Bi-Directional RRT Algorithms for Efficient Re-Planning in Dynamic Environments (2007)
23. Karaman, S., Frazzoli, E.: Incremental Sampling-based Algorithms for Optimal Motion Planning coRR (2010)
24. Nasir, J., Islam, F., Malik, U., Ayaz, Y., Hasan, O., Khan, M., Muhammad, M.S.: RRT*-SMART: a rapid convergence implementation of RRT*. Int. J. Adv. Robot. Syst., 10 (2013)
25. Jaillet, L., Corts, J., Simon, T.: Sampling-based path planning on configuration-space costmaps. IEEE Transactions on Robotics **26**(4), 635–646 (2010)
26. Guitton, J., Farges, J.-L., Chatila, R.: Cell-RRT: Decomposing the environment for better plan. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009. IROS 2009, pp. 5776–5781
27. Jacobs, S.A., Stradford, N., Rodriguez, C., Thomas, S., Amato, N.M.: A scalable distributed RRT for motion planning. In: 2013 IEEE International Conference on Robotics and Automation (ICRA), pp. 5088–5095, p. 2013. IEEE
28. Felzenszwalb, P., Huttenlocher, D.: Distance transforms of sampled functions Cornell University (2004)
29. Barraquand, J., Latombe, J.-C.: Robot motion planning: A distributed representation approach. Int. J. Robot. Res. **10**(6), 628–649 (1991)

E. G. Tsardoulias has obtained his doctorate and engineering diploma from the Department of Electrical and Computer Engineering at Aristotle University of Thessaloniki (AUTH), Greece, in 2013 and 2007 respectively. His working experience includes participating in the "Eudoxus" project, a web application that unifies the Greek University secretariats, enables the easy distribution of books and the on-line payment of the publishing houses, with the collaboration of Ministry of Education. His research interests are focused in Robotics and specifically in Autonomous Robots. Some of the topics involved are autonomous navigation, SLAM (Simultaneous Localization And Mapping) and multi-robot exploration / full coverage. In addition, from 2009 till now, is the head manager of the artificial intelligence group of the robotics team P.A.N.D.O.R.A., which operates at the Department of electrical and computer engineering, Aristotle University of Thessaloniki, and is an IEEE branch also. Team P.A.N.D.O.R.A. has participated in the national competition RoboCup- RoboRescue in China, Graz-Austria, Istanbul-Turkey and Eindhoven-Netherlands. Finally, he has published scientific papers in magazines such as Microprocessors and Microsystems and Journal of Intelligent Robotics, as well as presentations in conferences like ICAPS. His current occupation is in the FP7 funded RAPP project, involving technical coordination, robotics expertise and development.

A. Iliakopoulou holds a Master's degree in Computer Science from the Fu Foundation of Engineering and Applied Sciences at Columbia University in the city of New York and an engineering diploma from the Department of Electrical and Computer Engineering at Aristotle University of Thessaloniki (AUTH), Greece. Her working experience includes both industry driven and research projects. As an undergrad, she participated in the robotics team P.A.N.D.O.R.A. developing algorithms for autonomous robot navigation in unknown spaces. She participated in the national competition RoboCup-RoboRescue in Istanbul-Turkey in 2010. After her graduation from Aristotle University of Thessaloniki (AUTH), Greece, she worked as a Research Assistant in Center for Research & Technology Hellas (CERTH), Greece, where she contributed to "SocialSensor", an international project, which aimed at building an application that discovers trends, events, and interesting media content in social media. Ms Iliakopoulou has also been a Technology Intern at the The New York Times, developing tools for the newspaper's newsroom that facilitate computing data analytics of their content. In May 2016, she received the award of excellence in Computational Journalism from the Brown Institute for Media Innovation and the Tow Center for Digital Journalism at Columbia University for her Master's thesis, "Building an Automated Tagger System." Ms. Iliakopoulou will continue her career at The New York Times in the position of Software Engineer.

A. Kargakos graduated from Electrical and Computer Engineering Dpt. of Aristotle University of Thessaloniki (AUTH). He is a research assistant in the Information Technologies Institute of Centre for Research and Technology Hellas (CERTH-ITI) since 2013. His research interests include fields such as robotics and artificial intelligence. He participated in robotics team P.A.N.D.O.R.A. (AUTH) and European research projects as CloPeMa and Ramcip (CERTH-ITI).

L. Petrou received his Diploma of Electrical Engineering from the University of Patras, Greece, in 1973, the M. Sc. Degree in Theory and Practice of Automatic Control from U.M.I.S.T., Manchester, U.K. in 1975 and the Ph.D. in Electrical Eng., from Imperial College, London, U.K. in 1979. Currently, Dr. Petrou is an Associate Professor at the Electrical and Computer Engineering Department of the Aristotle University of Thessaloniki, Greece. His research interests include Microprocessor applications, autonomous mobile robots, intelligent control of waste water treatment plants, reconfigurable embedded systems, evolutionary computation, fuzzy logic and neural network control systems, sensor networks. His work has been published in over 40 papers, book chapters, and conference publications. Currently he is Team Leader of the PANDORA (Program for the Advancement of Non Directed Operating Robotic Agents) Robotics Research Team of the Electrical and Computer Engineering Department of the Aristotle University of Thessaloniki, (http://pandora.ee.auth.gr) and participates in the EDUSAFE Marie Curie ITN project of CERN.