Creating High-Quality Roadmaps for Path Planning

Roland Geraerts and Mark H. Overmars
Institute of Information and Computing Sciences, Utrecht University
3508 TB Utrecht, the Netherlands, Email: {roland,markov}@cs.uu.nl

Abstract

A central problem in robotics is planning a collision-free path for a robot in a static environment with obstacles. Such a path is usually extracted from a roadmap that represents the connectedness of the free space. Contemporary algorithms generally do not give guarantees on the quality of the roadmap, and, hence, the quality of the paths. While techniques exist for optimizing the paths in the on-line query phase, they are too slow to be applied in real-time applications. In this paper we present a technique for creating high-quality roadmaps, shifting the optimization process to the off-line construction phase. As a result, high-quality paths can be extracted from these roadmaps at interactive speeds.

Our method creates roadmaps from which paths can be extracted satisfying the following criteria. First, we require that a path for a query can always be found if one exists (at a given resolution). In addition, we require low query times and low memory consumption. These requirements are met by the *Reachability Roadmap Method* (RRM), which is an efficient new algorithm that creates small, resolution complete roadmaps for problems involving two or three degrees of freedom such as mobile robots. Then, we require that short and alternative paths can be obtained which is met by adding useful cycles to the roadmap created by the RRM. When safety is of major importance, we require that paths can be obtained having a large clearance to the obstacles. This criterion is met by retracting the local paths corresponding to the edges of the roadmap to the medial axis. Since all operations are performed in a preprocessing phase, high-quality paths can now be extracted from the roadmap in real-time.

1 Introduction

Motion planning is one of the fundamental problems in robotics. The motion planning problem can be defined as finding a path between a start and goal placement of a robot in an environment with obstacles. The past fifteen years, efficient algorithms have been devised to tackle this problem. They are successfully applied in fields such as mobile robots, manipulation planning, CAD systems, virtual environments, protein folding and human robot planning. See the books of Choset *et al.* [10], Latombe [22] and LaValle [23] for an extensive overview.

Many problems in these fields have been successfully solved by the *Probabilistic Roadmap Method* (PRM), see e.g. [1, 21, 30]. The PRM consists of two phases: a construction and a query phase. In the construction phase, a roadmap graph is built, capturing the connectivity of the free space with a set of one-dimensional curves. Globally speaking, the PRM samples the configuration space $\mathcal{C}$ (that is, the space of all possible placements for the robot in the

environment) for collision-free placements. These are added as nodes to the graph. Pairs of promising nodes are chosen in the graph and a simple local planner is used to try to connect such placements with a path. If they can be connected, then an edge between the nodes is added to the graph. This process continues until the graph reflects the connectivity of the free configuration space $\mathcal{C}_{\text{free}}$ (that is, the space of all possible *free* placements for the robot). In the query phase, the start and goal placements are connected to the graph. The path is usually obtained by a Dijkstra's shortest path query.

A drawback of the PRM is that a resulting roadmap often contains many redundant nodes and edges, in particular when the environment contains one or more narrow passages [14]. Over the years, many improvements have been suggested to tackle the narrow passage problem (see e.g. [5, 18, 25]) but those solutions often lead to large roadmaps. Such large roadmaps increase the time needed to extract a path and can require a vast amount of memory which may not always be available.

A variant of the PRM, which aims at keeping the roadmap small, is the *Visibility* PRM proposed by Nissoux *et al.* [28]. This technique only connects new nodes to useful nodes. Usefulness is determined as follows: when a new node cannot be connected to other nodes it forms a new connected component and is labeled useful. If it connects two or more components it is also labeled useful. If it can be connected to just one component it is not labeled useful. Although this approach prunes the roadmap a lot, it is often slower than other variants of the PRM as the approach is too strict in rejecting nodes [17].

While these roadmap-based method have been successfully applied to a broad range of motion planning problems, they generate ugly paths. That is, the paths are only piecewise linear, they have many redundant motions, and they have little clearance to the obstacles, resulting in unnatural looking motions. While techniques exist for optimizing the paths [13, 15, 26], they are too slow to be applied in real-time, i.e. in the query phase. To improve the query time, the optimization process should be shifted to the preprocessing phase. A first step in this direction was carried out by Nieuwenhuisen *et al.* [26]. The authors improve a roadmap generated by the PRM. Their method guarantees that the paths are short, have enough clearance from the obstacles, and are C^1 continuous, leading to natural looking motions. However, their method only works for problems involving two degrees of freedom.

Another drawback of these algorithms is that a path may not always be found in practice, in spite of its existence. This can occur for example when the roadmap is not dense enough.

In this paper, we are interested in creating high-quality roadmaps for problems involving two or three DOFs. Hence, we can employ a more specialized method, allowing the extraction of high-quality paths in real-time.

We aim at generating roadmaps with the following five properties:

1. The roadmap is *resolution complete*. This means that a valid query (which consists of a start and a goal configuration) can always be connected to the roadmap. If the start and goal belong to the same connected component of the free space, then a corresponding path can always be found (at a given resolution).

2. The roadmap is *small*. A small roadmap assures low query times and low memory consumption. In addition, when a roadmap must obey other criteria, a small roadmap eases manual tuning.

3. The roadmap contains *useful cycles*. These cycles provide short paths and alternative routes which allow for variation in the routes that entities take. For example, van

den Berg *et al.* [2] exploit cycles in dynamic environments where additional obstacles might appear, and to avoid deadlock situations when multiple robots move in the same environment.

4. The roadmap provides *high-clearance paths*. By retracting the roadmap to the medial axis, paths with much clearance can be extracted in real-time. High-clearance paths are more natural and safe. In addition, they are perfectly suitable for guiding the motions of a group of entities [20].

5. The roadmap provides paths in *real-time*. By shifting the optimization process to the off-line construction phase (at the cost of longer preprocessing times), no smoothing is required in the query-phase, providing high-quality paths at interactive speeds.

The paper is organized as follows: In Section 2, we discuss our new *Reachability Roadmap Method* (RRM) which creates the initial roadmap satisfying the first and second property. It may be surprising to see how small covering roadmaps can be. We compare the Reachability Roadmap Method with the PRM and visibility PRM. Experiments will show that roadmaps created by the PRM can have many nodes and edges. The visibility PRM creates reasonably small roadmaps but this method has great difficulties in creating roadmaps that can solve any query. In contrast, the RRM creates small covering roadmaps for all tested environments. In Section 3, we enhance the roadmap by presenting an algorithm that adds useful cycles and clearance to the roadmap. After performing experiments with 2D and 3D environments, we conclude in Section 4 that our algorithm successfully creates roadmaps satisfying the five properties.

2 Creating small, resolution complete roadmaps

In this section, we will discuss the Reachability Roadmap Method which is a new and efficient method that creates small roadmaps. It is particularly suited for problems involving two or three DOFs. The method ensures that a valid query can always be connected to the roadmap and that a path is always found (if one exists) at a given resolution. We will discuss the properties of the approach in Section 2.1. In Section 2.2, we show the outline of the method. Details will be given in Section 2.3. In Section 2.4, we show experiments on different environments and conclude that the method successfully creates small roadmaps for these environments.

2.1 Reachability

An important issue is how to determine when a roadmap is dense enough, i.e. when should the construction algorithm terminate? Many motion planning techniques such as the PRM and Visibility PRM are probabilistically complete, i.e. whenever a path exists, the probability that it will be found converges to one as the computation time goes to infinity. As there is no guaranteed upper bound, the running time may not be a practical termination criterion. Many authors terminate the method when a path between a specified start and goal is found. However, this does not guarantee that this roadmap can solve every query.

In [16], we used the *reachability* criterion to determine when a problem has been solved, i.e. a problem has been solved if a roadmap $G = (V, E)$ covers the free configuration space ($\mathcal{C}_{\text{free}}$) and captures its connectivity. We defined *coverage* and *maximal connectivity* as follows:

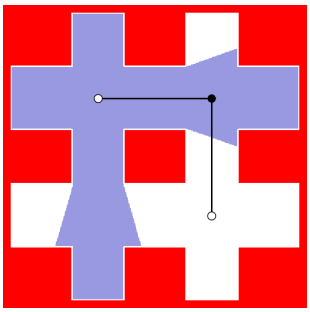


Figure 1 The coverage and maximal connectivity criteria have been met. The reachability regions of the white nodes cover the complete free space and are connected via the black node.

Definition 1 (coverage)**.** *G covers $\mathcal{C}_{\text{free}}$ when each configuration $c \in \mathcal{C}_{\text{free}}$ can be connected using the local planner to at least one node $\nu \in V$.*

Definition 2 (maximal connectivity)**.** *G is maximally connected when for all nodes $\nu', \nu'' \in V$, if there exists a path in $\mathcal{C}_{\text{free}}$ between ν' and ν'', then there exists a path in G between ν' and ν''.*

Coverage ensures that every query (which consists of a start and goal configuration) can be directly connected to the roadmap, as is required to solve the problem. If there exists a path (in $\mathcal{C}_{\text{free}}$) between the start and goal configuration, then maximal connectivity ensures that a path between them can be found in the roadmap.

Figure 1 shows an environment whose free space is covered by two (white) nodes and is connected via one extra (black) node. The reachability region for the upper left node has been drawn. Each configuration in this region can be connected with a straight-line local planner (i.e. a straight-line connection [17]) to the node, and a three-node graph suffices to solve this problem. In Section 2.3, we will explain how such a reachability region can be computed.

2.2 Reachability Roadmap Method

Our goal is to create a small roadmap $G = (V, E)$ that satisfies both the coverage and maximal connectivity criteria. The idea is to compute a small set of *guards* that cover the complete free space. These guards are then connected via *connectors* to fulfill the maximal connectivity criterion. The resulting roadmap is then pruned to obtain an even smaller roadmap (see Algorithm 1).

Computing the minimum number of guards corresponds to the well known art gallery problem which is NP-hard. See the book of O'Rourke which elaborates on this problem [29]. As we want to create a roadmap for a possibly large environment with many obstacles, exact algorithms will take too much time. Therefore, we will use a heuristic to create and place these guards (which are added as nodes to the roadmap).

Globally speaking, the guards are chosen as follows. To get the free space $\mathcal{C}_{\text{free}}$ covered as fast as possible, we place the guard nodes $V^g \subseteq V$ on locations where they probably cover a large part of the free space. Locations on the medial axis M are good candidates as they have a large clearance from the obstacles and thus have an increased probability of having a large reachability region [31]. In addition, placing guards on the medial axis produces 'good' roadmaps [25]. To prune the number of candidate guards, a new guard ν^g is only accepted if

Algorithm 1 ReachabilityRoadmap ($\mathcal{C}$-space $\mathcal{C}$)

Output: Graph $G(V, E)$, where $V = V^g \cup V^c$

covered space $\mathcal{C}' \leftarrow \emptyset$
$M \leftarrow$ locus of configurations of the medial axis
while $\mathcal{C}' \neq \mathcal{C}_{\text{free}}$ **do**
  if $M \backslash \mathcal{C}' \neq \emptyset$ **then**
    guard node $\nu^g \leftarrow$ the node corresponding to a configuration $c \in M \backslash \mathcal{C}'$
  else
    configuration $c \leftarrow$ a configuration in $\mathcal{C}_{\text{free}} \backslash \mathcal{C}'$
    guard node $\nu^g \leftarrow$ node corresponding to RetractConfiguration(c)
  $\mathcal{C}' \leftarrow \mathcal{C}' \cup$ ReachabilityRegion(ν^g)
  $V^g \leftarrow V^g \cup \nu^g$
for all $\nu_i, \nu_j \in V^g : i < j$ **do**
  $\mathcal{C}' \leftarrow$ ReachabilityRegion(ν_i) $\cup$ ReachabilityRegion(ν_j)
  if $\mathcal{C}' \neq \emptyset$ **then**
    $\nu^c \leftarrow$ connector node that corresponds to a configuration in $\mathcal{C}'$
    $V^c \leftarrow V^c \cup \nu^c$
    $E \leftarrow E \cup \epsilon(\nu_i, \nu^c)$
    $E \leftarrow E \cup \epsilon(\nu^c, \nu_j)$
PruneRoadmap($G, \mathcal{C}$)

its location c is not covered by guards that have already been placed. That is, $c \in M \backslash \mathcal{C}'$ where $\mathcal{C}' \subseteq \mathcal{C}_{\text{free}} = \bigcup_{\nu \in V^g}$ ReachabilityRegion(ν). Nonetheless, it can occur that all locations on the medial axis have been covered by the guards, i.e. $M \backslash \mathcal{C}' = \emptyset$, while the free space has not been fully covered yet, i.e. $\mathcal{C}' \neq \mathcal{C}_{\text{free}}$. In such a situation we choose a configuration c that has not been covered yet by other guards, i.e. $c \in \mathcal{C}_{\text{free}} \backslash \mathcal{C}'$. Then we retract this configuration to the medial axis by the technique from [31] to increase the expected size of its reachability region. (Note that the location of this guard will be covered by other guards.) We keep adding new guards to the roadmap until $\mathcal{C}_{\text{free}}$ has been fully covered, i.e. $\mathcal{C}' = \mathcal{C}_{\text{free}}$.

We connect the guards by placing connectors in the overlapping reachability regions of the guards. We consider all pairs of guards whose regions overlap. For each pair $(\nu_i, \nu_j) : i < j$, we add a connector node ν^c and its connections (edges $\epsilon(\nu_i, \nu^c)$ and $\epsilon(\nu^c, \nu_j)$) to the roadmap.

Finally, we prune this roadmap by transforming it to a Steiner Tree. Such a tree is a network that does not throw away guards (but is allowed to remove connectors) and keeps the connected components connected. To prune the roadmap even more, we add all remaining connections and create a minimal spanning tree.

Theorem 1 (Coverage of $\mathcal{C}_{\text{free}}$)**.** *Algorithm 1 creates a roadmap that satisfies the coverage criterion.*

Proof: It is well known that each configuration in $\mathcal{C}_{\text{free}}$ can be connected to the medial axis in a straight line. Hence, we can limit ourselves to consider only configurations lying on the medial axis. We start with selecting configurations on the medial axis and add them as nodes to the roadmap. If all configurations on the medial axis are covered by the guards but there are still configurations in the free space that have not been covered yet, we select such a configuration which we retract to the medial axis. As this retraction can be performed in a straight line [31], the original configuration will be covered by the retracted configuration.

Because we keep adding new guards until the free space is completely covered, the coverage criterion eventually will be met. This criterion will remain satisfied when the roadmap is pruned as these heuristics never remove guards. ■

Theorem 2 (Maximal connectivity of $\mathcal{C}_{\text{free}}$)**.** *Algorithm 1 creates a roadmap that satisfies the maximal connectivity criterion.*

Proof: Choset and Burdick show in [9] that the medial axis is a connected structure if the free space in which a robot operates is also connected. Hence, the medial axis is a complete representation for motion planning purposes. Let $\nu', \nu'' \in V$ be two random nodes in the graph (corresponding to configurations c' and c'') for which there exists a continuous path Π in $\mathcal{C}_{\text{free}}$ between c' and c''. As the algorithm satisfies the coverage criterion, there exists for each configuration $c \in \Pi$ a guard node $\nu^g \in V^g \subseteq V$ that covers c. Now consider the sequence of different guard nodes $\nu_i \in V^g$ that covers the configurations on Π. (If several guard nodes in V^g cover a particular configuration c, we consider the guard node having the smallest index.) Let now configurations c_a and c_b correspond to guard nodes ν_i and ν_{i+1}. Since there exists a path in $\mathcal{C}_{\text{free}}$ between configurations c_a and c_b, the reachability regions of nodes ν_i and ν_{i+1} have some overlap. (This may be a point in the extreme case.) As the algorithm chooses a connector node $\nu^c \in V^c \subseteq V$ in this overlapping part, edges $\epsilon(\nu_i, \nu^c), \epsilon(\nu^c, \nu_{i+1}) \in E$ will be part of the graph. Hence, we can construct a path Π' in graph G between nodes ν' and ν'' as follows: $\Pi' \leftarrow \bigcup_i \epsilon(\nu_i, \nu^c) \cup \epsilon(\nu^c, \nu_{i+1})$.

The maximal connectivity criterion will remain satisfied when the roadmap is pruned as the heuristics only remove edges that are part of cycles. ■

2.3 Algorithmic details

The most time-consuming operation in our technique is the computation of the reachability regions. In general, an exact computation of these regions would involve intricate and practically infeasible calculations in the configuration space $\mathcal{C}$. To make the calculations feasible, we approximate the $\mathcal{C}$-space by discretizing it. Because of memory limitations, this approach is restricted to low-dimensional $\mathcal{C}$-spaces. We will perform experiments with both two- and three dimensional $\mathcal{C}$-spaces.

We discretize a d-dimensional $\mathcal{C}$-space as follows. First, we create a d-dimensional grid. For each cell in this grid we check whether the configuration that corresponds to the cell collides with the obstacles. We update the cell appropriately, i.e. we put the value 0 in the cell if it collides and 1 otherwise. We will use this grid in the remainder of the algorithm.

In theory, the grid itself could be used for motion planning purposes. However, in this paper, we are dealing with environments that contain narrow passages and/or environments that are large. We need a huge grid for both types. In Section 3.3, we will show that such a grid will be too large to use as a roadmap, i.e. the time for extracting a query will be too high for real-time usage. Besides, our goal is to create small roadmaps.

We will now show for a given resolution how to get the free space covered and maximally connected. Then, we show how the roadmap can be pruned.

Coverage

We need a set of cells M that reside on the medial axis. Such a set can be computed efficiently by the medial axis transform (MAT). We use the algorithm from Lee *et al.* in [24].

(a) $\mathcal{C}$-space (b) MAT (c) DT (d) MAT ∩ DT (e) guards

Figure 2 The creation of candidate guards. The white area of (a) corresponds to the discretized $\mathcal{C}$-space of Figure 1. Fig. (b) shows the cells of the medial axis transform and (c) the distance transform. Dark cells correspond to a large distance to the closest obstacle while light cells correspond to a small distance. In (d), the intersection of the MAT and DT is shown. Finally, (e) shows the four guards on the medial axis having the largest distance in the distance transform.

This algorithm is a two-pass dynamic program that computes the MAT in $O(n)$ time where n is the number of cells in the grid. See Figure 2(b) for an example.

We select guards on the medial axis having a large distance to the obstacles. The distance can be computed efficiently by the distance transform (DT) in $O(n)$ time (see Figure 2(c)). Like the MAT, the DT is also computed by a two-pass dynamic program (see e.g. [24]). By combining the MAT and DT, we know for each cell on the medial axis the distance to the closest obstacles (see Figure 2(d)). These cells are then sorted by decreasing distance, using bucket sort in linear time. We store these cells (candidate guards) in a list M. Figure 2(e) shows an example of how such candidate guards are obtained.

We keep adding guards from this list to the roadmap until the free space has been fully covered (or when all candidate guards corresponding to the cells in M have been handled). As we have already mentioned, we initially only add guards as nodes to the graph which have not been covered yet by other guards. If the space is not covered when all cells in M have been handled, we consider uncovered cells (in order of decreasing distance) and add their corresponding configurations retracted on the medial axis (see [31] for details).

There are two issues we have to resolve: how to determine the cells in the reachability region of a guard and how to determine which cells have not been covered yet.

We consider two ways to compute the reachability region of a guard placed in cell $g \in G \subseteq M$. The first algorithm is analogous to a FLOOD-FILL algorithm. Its usual purpose is to fill an arbitrary bounded space in a picture with a given color. The boundary of a reachability region consists of cells that cannot be reached anymore from cell g. We start with adding cell g to a queue. The algorithm terminates looping when the queue is empty. In each loop, we extract a cell c from the queue. If the straight-line connection from g to c is not in $\mathcal{C}_{\text{free}}$ (which can be calculated efficiently by the Bresenham line-drawing algorithm [7]), then we know that g does not belong to the region. Hence, we can continue taking cells from the queue. In the other case, the region will be extended by cell c. Then we check for each of the facing neighbors of c whether such a neighbor is in $\mathcal{C}_{\text{free}}$ and whether the neighbor has not been covered yet. (A cell in a d-dimensional grid has $2d$ facing neighbors.) If the two conditions are satisfied, then we add the neighbor cell to the queue. This approach can be applied in a grid of any dimension d. Its complexity is $O(d|g|l)$, where $|g|$ is the number of cells in the region and l is the number of cells that intersects with the line from r to the furthest cell in the reachability region of g.

The second algorithm, which we call WedgeFill, improves this bound to $O(|g|)$ when the grid is two-dimensional. The algorithm subdivides the 2D space in four quadrants originating from cell g. In each quadrant, it fills cells within the boundaries of one or more wedges. For each wedge, the cells on the current scan line are added to the region if they are visible from g, i.e. the cell lies within the boundaries of the wedge. If an obstacle cell is encountered, the wedge is split or its boundaries are narrowed. This process is repeated until each wedge is too narrow to contain a cell or until all cells on the current scan line are obstacle cells.

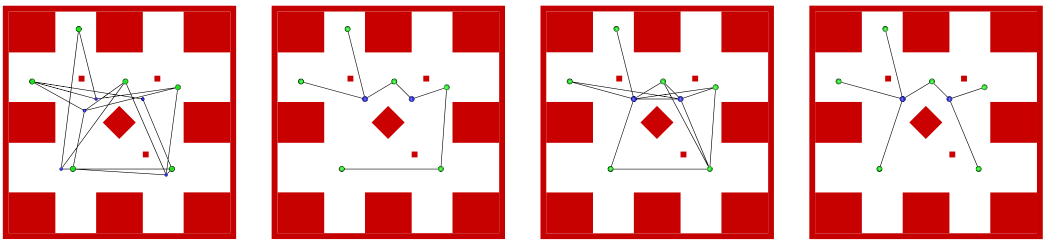
To determine which cells have not been covered yet, we store for each cell in the grid the set of guards that cover the cell, i.e. when guard g_i is added to the roadmap, index i is inserted to the sets in the grid corresponding to all cells that are covered by this guard. A set is a data structure that allows insertions in $\mathrm{O}(\log k)$ time [11], where k is the number of guards that cover this cell. Checking whether the cell has been covered by a guard also takes $\mathrm{O}(\log k)$ time. Storing all covering guards might seem an overkill but we need this information in the next phase. In addition, k will generally be very small.

Maximal connectivity

As most guards usually do not cover any other guards, we have to calculate a set of connectors to which the guards can be connected such that the maximal connectivity criterion is satisfied. These connectors will be placed in the overlapping reachability regions of the guards. For each pair of guards that share cells in the grid, we place one connector in a cell that lies on the medial axis. As the number of shared cells is generally larger than one, we have to choose one of those cells. Since we prefer a large clearance, we choose the one that has the largest value in the distance transform (DT). When the set of shared cells contains no medial axis cells, we choose a cell that has the largest distance in the DT. If more than one cell satisfies this condition, we choose the one that minimizes the connection distance of the connector to the two guards. Possibly, different connectors share the same cell in the grid. These connectors are then merged. This part of the algorithm can be computed in linear time in the number of elements of the sets in the grid. We obtain a roadmap that satisfies the maximal connectivity criterion by adding all collision-free connections between the guards and connectors.

Roadmap pruning

If we allow all collision-free connections between guards and connectors, the number of connections can become quadratic in the number of nodes in the roadmap. Figure 3(a) shows such a fully connected roadmap. As our objective is to create small roadmaps, we want to solve the following problem. Given a roadmap $G = (V, E)$ that consists of a set of guards $V^g \subseteq V$, a set of connectors $V^c \subseteq V$, and all feasible connections E between the joint set of guards and connectors, create a shortest roadmap $G' = (V', E')$ (in terms of total edge length) that spans the guards. This problem is known as the discrete Euclidean Steiner problem which is NP-complete. (See the book of Hwang *et al.* [19] which elaborates on Steiner Tree problems.) Consequently, no polynomial time algorithm for this problem is likely to exist. We handle this problem by using the *shortest path heuristic*, described in the same book. Figure 3(b) shows such a tree. Algorithm 2 outlines the approach. First, a priority queue Q is initialized (sorted on increasing path length) with all shortest paths between the guards in V^g. Then, a graph G' is created that consists of all guards V^g. For each path Π in Q, an edge $\epsilon(\nu', \nu'')$ of Π is added to the edges E' if there exists no path between nodes ν'



(a) Initial roadmap (b) Steiner Tree (c) All connections (d) MST

Figure 3 Roadmap pruning. Figure (a) shows the initial roadmap. From this roadmap, an approximated Steiner Tree is calculated in (b). In (c), all collision-free connections are added. Figure (d) shows the minimal spanning tree (MST) of the roadmap in (c).

and ν'' in G'.

Algorithm 2 SteinerTree (Graph $G(V = V^g \cup V^c, E)$)

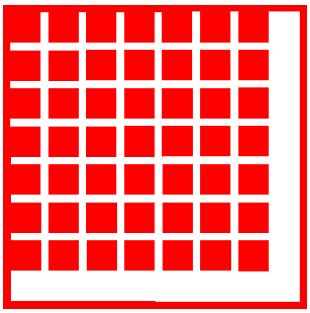
Output: Graph $G'(V' = V^{g\prime} \cup V^{c\prime}, E')$

```
1: PriorityQueue Q {sorted on increasing path length}
2: for all distinct pairs of guards (ν', ν'') do
3:    node path N ← shortest path between ν' and ν''
4:    if |N| > 0 then Q.push(N)
5: V^{g'} ← V^g
6: while not Q.empty() do
7:    Π ← Q.front()
8:    Q.pop()
9:    for i ← 0 to |N| − 2 do
10:      if ν_{i+1} ∈ V^c and ν_{i+1} ∉ V^{c'} then
11:         V^{c'} ← V^{c'} ∪ ν_{i+1}
12:      if not G'.sameConnectedComponent(ν_i, ν_{i+1}) then
13:         E' ← E' ∪ ε(ν_i, ν_{i+1})
14: remove all connectors ν^{c'} ∈ V^{c'} with degree 1
```

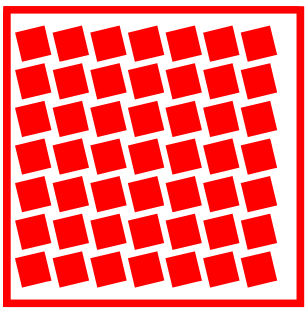
The total edge length of G' can be decreased further by the following approach. First, all collision-free connections between the nodes in G' are added to G'. Then, its minimal spanning tree (MST) is calculated by Kruskal's algorithm [11]. It can occur that by reconnecting the edges, there are connectors in the graph having degree 1. These are removed as they do not contribute to the coverage and/or maximal connectivity. Figure 3 shows these pruning steps.

2.4 Experiments

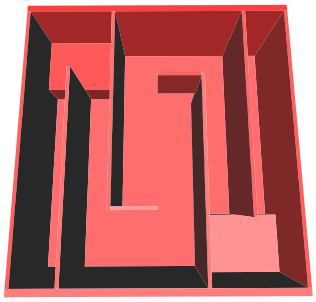
In this section, roadmaps are created by appling the Reachability Roadmap Method (RRM) to five environments. These are depicted in Figure 4. We compare the RRM with the PRM and Visibility PRM with regard to the size of the roadmaps as well as the required computation time.



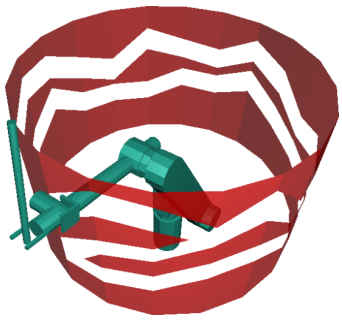
(a) Grid



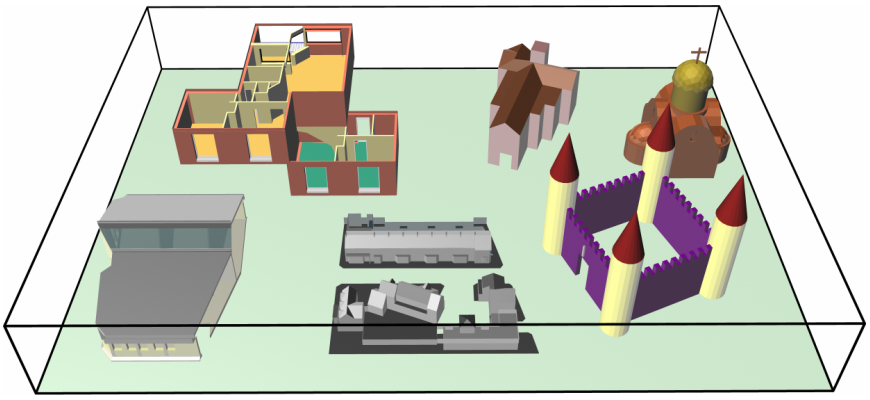
(b) Rotated grid



(c) Corridor



(d) Manipulator



(e) Toy village

Figure 4 The five test environments.

Experimental setup

All techniques were integrated in a single motion planning system called SAMPLE (System for Advanced Motion PLanning Experiments), implemented in Visual C++ under Windows XP. SAMPLE automates conducting experiments, i.e. statistics are automatically generated and processed, decreasing the chance for errors. All experiments were run on a 3GHz Pentium 4 processor with 1 GB internal memory. We used Solid as basic collision checking package [4].

We use a square-shaped robot for the 2D problems and a box-shaped robot for the 3D problems. We perform experiments on five environments. They have the following properties:

Grid This is a simple 2D environment that contains 49 squares. The optimal solution is a roadmap containing seven guards, six connectors and twelve edges. We want to know whether the RRM can be competitive compared to the optimal solution.

Rotated grid We rotated the 49 squares which resulted in many narrow passages. We use this environment to study the effect of the reduced visibility on the size of the roadmaps.

Corridor This environment consists of a 3D winding corridor with four hairpins. While we expect that the RRM creates a small roadmap, we expect that the PRM will be faster than the RRM because this is an easy problem with no narrow passages.

Manipulator This 3D environment features a robot arm with three rotational degrees of freedom that can move through a long passage. We selected this environment to show that our algorithm can handle complex reachability regions in $\mathcal{C}$-space.

Toy village This large 3D environment contains seven buildings (>10,000 objects). There are many scale differences in this environment, i.e. the environment has ample free spaces (outside) and small rooms (inside). We want to know whether our algorithm can handle such large environments.

For each environment, we create a roadmap with the Reachability Roadmap Method. Information on the environments and robots is listed in Table 1. To decide which cells are free, for each cell we check a square (for the 2D grids) or a box (for the 3D grids) for collisions with the environment. The reachability regions are computed by the WEDGEFILL algorithm for 2D-problems and by FLOODFILL algorithm for the 3D-problems.

We compare this technique with the PRM (as this is a widely used motion planning method) and the Visibility PRM (as this method creates small roadmaps). They have some parameters that have to be set. We use the Halton sampling strategy with a random seed [6]. The step size of the straight-line local planner corresponds to the size of a cell in the grid. We choose the *nearest-k* node adding strategy as this method performed reasonably well on different environments [17]. This strategy tries to connect the new configuration to the nearest k nodes in the graph that lie close enough. For both techniques, we perform 100 independent runs. Such a run has been solved if both the coverage and maximal connectivity criteria have been satisfied. When we report the construction time, we do not include the time needed to check these two criteria.

In the first batch of experiments, we focus on the size of the roadmaps. We set the maximum connection distance and maximum number of connections to infinity. This will minimize the number of nodes and edges needed to fulfill the coverage and maximal connectivity criteria. These choices have a negative impact on the running times for the PRM and Visibility PRM [17]. Nonetheless, these choices minimize the size of the roadmap.

Table 1 Information on the environments and robots.

	Bounding boxes		
	number of cells	**robot**	**# objects**
Grid	80×80	1×1	53
Rotated grid	200×200	$1 \times 1 \times 1$	53
Corridor	$40 \times 8 \times 40$	$1 \times 1 \times 1$	13
Manipulator	$60 \times 60 \times 60$	$1 \times 1 \times 1$	61
Toy village	$180 \times 22 \times 160$	$1 \times 1 \times 1$	10,155

Table 2 The optimal values used in the neighbor selection strategy. $^{(*)}$We did not constrain the maximum connection distance for the Visibility PRM in the Grid environment.

	Neighbor selection parameters	
	max. connection distance	**max. number of connections**
Grid$^{(*)}$	10	75
Rotated grid	5	75
Corridor	15	75
Manipulator	2	75
Toy village	50	75

In the second batch of experiments, we focus on the running times of the three methods. To provide a fair comparison, we use the optimal connection parameters (which have been derived from preliminary experiments) for the PRM and Visibility PRM. These are stated in Table 2. Using optimal values for the parameters will cause the PRM and Visibility PRM to terminate faster, but larger graphs will be produced.

We record the following statistical data: the construction time of the roadmap, the number of nodes $|V|$ and the number of edges $|E|$ in the roadmap. Furthermore, for each environment and technique we measure the length of the roadmap $|G|$ which we calculate as the sum of the lengths of the edges in the roadmap.

Experimental results

When we conducted the experiments, we experienced that the PRM and the Visibility PRM in particular were not always able to find a solution within the maximum running time that had been set to two hours. In those cases we conducted only 10 experiments. (The PRM could not find any solution for the Manipulator and Toy village environments. The Visibility PRM only found solutions for the Grid and Corridor environments.) The roadmaps created by the RRM are visualized in Figure 5. The results are stated in Table 3.

Grid The RRM computed a roadmap that has the minimum number of nodes. The roadmap length is close to optimal. Hence, our algorithm is competitive compared to the optimal solution. While the PRM created much larger roadmaps, the Visibility PRM produced reasonably small ones but this took on average more time than the other two techniques. As this technique sometimes creates artificial narrow passages [28], it may be difficult to solve the problem. When the optimal connection parameters were used for the PRM, its running times were comparable to the times of the RRM, but the PRM produced considerably larger roadmaps.

Rotated grid As the reachability regions are much smaller in this environment than the regions in the Grid environment, more nodes are required to get the free space covered. The RRM needed far less nodes and edges than the PRM, because the RRM tries to minimize the amount of double coverage. In addition, the resulting roadmap length was much smaller. The Visibility PRM was unable to solve the problem within two hours.

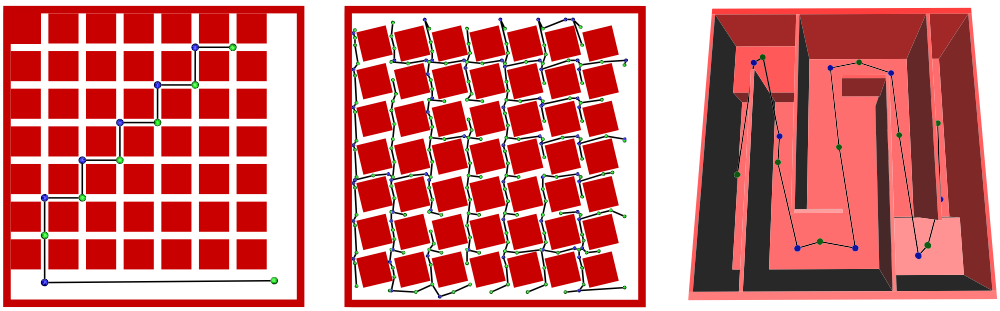
Corridor As expected, the RRM created a small roadmap, consisting of 17 nodes and 16 edges. By lack of narrow passages, the PRM was considerably faster than the RRM. However, the PRM created relatively large roadmaps. The Visibility PRM created small roadmaps but was the slowest technique.

Manipulator The RRM required 313 nodes and 306 edges to cover and connect the three-dimensional configuration space. The PRM and Visibility PRM were unable to solve the problem within two hours. This is because the free $\mathcal{C}$-space contains small regions (which complicates getting $\mathcal{C}_{\text{free}}$ covered) and narrow passages (which complicates getting the nodes connected).

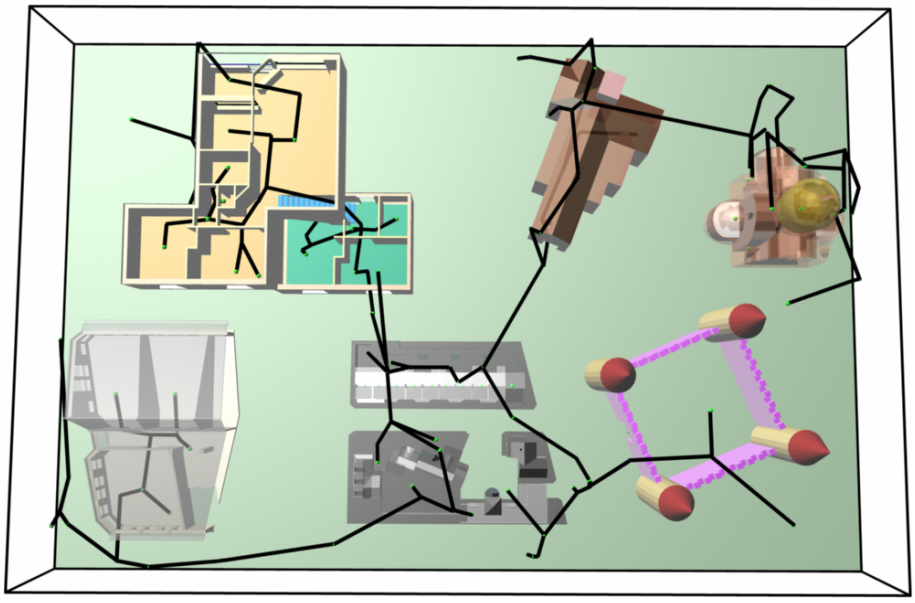
Toy village As this environment has a rather non-uniform distribution of the obstacles, the PRM had difficulties in covering and connecting the small areas. The Visibility PRM was unable to solve the problem within two hours as well. In contrast, the RRM solved the problem within six minutes. The resulting roadmap concentrated the nodes in regions where much detail was present in the environment, while nodes in ample free space were relatively sparse. This resulted in a surprisingly small roadmap.

The experiments showed that roadmaps created by the PRM can have many nodes and edges. The visibility PRM created reasonably small roadmaps but this method had great difficulties in creating covering roadmaps. In contrast, the RRM created small covering roadmaps for all tested environments. It was surprising to see how small covering roadmaps can be.

In conclusion, the RRM is well suited for creating small roadmaps for problems involving two or three DOFs with many scale differences or narrow passages. More general techniques, such as the PRM and Visibility PRM, are better suited for problems with many DOFs and problems having favorable visibility properties. That is, problems whose free configuration space can be captured by few nodes where each node can reach a large portion of the free space using a local planner.



(a) Grid (b) Rotated grid (c) Corridor



(d) Toy village

Figure 5 Resulting roadmaps projected onto the environments.

Table 3 Results for the five environments. The left column shows the results of the experiments that were focused on the *size* of roadmaps, i.e. we put no limits on the connection parameters for the PRM and Visibility PRM in the experiments. The right column shows the results of the experiments that were focused on the *construction time* of roadmaps, i.e. we used the optimal connection parameters for the PRM and Visibility PRM. $^{(*)}$The optimal maximum connection distance for the Visibility PRM was ∞. Hence, the results are the same as in the corresponding results in the left column. $^{(+)}$Ten out of ten runs were not solved within two hours.

	Results – Grid (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.25	**13**	**12**	**88**
PRM	0.72	127	125	629
Vis. PRM	0.86	26	25	331

	Results – Grid (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**0.25**	13	12	88
PRM	0.27	192	193	582
Vis. PRM$^{(*)}$	0.86	26	25	331

	Results – Rotated grid (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	2.49	**252**	**251**	**388**
PRM	19.50	2,594	2,593	1,614
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Results – Rotated grid (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**2.49**	252	251	388
PRM	3.40	2,551	2,550	1,493
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Results – Corridor (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.22	**17**	**16**	**156**
PRM	0.09	46	45	324
Vis. PRM	0.74	18	17	204

	Results – Corridor (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.22	17	16	156
PRM	**0.05**	56	55	3270
Vis. PRM	0.28	24	23	207

	Results – Manipulator (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	9.94	**313**	**306**	**861**
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Results – Manipulator (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**9.94**	313	306	861
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Results – Toy village (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	351.64	**177**	**143**	**1,278**
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Results – Toy village (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**351.64**	177	143	1,278
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

3 Enhancing the roadmap

Since the RRM concentrates on creating small roadmaps, the extracted paths may be long. In Section 3.1, we propose an algorithm that adds useful cycles to the roadmap to decrease the length of the extracted paths, satisfying our third property. We meet the fourth property in Section 3.2 which shows how to retract a roadmap to the medial axis. We perform experiments with 2D and 3D environments in Section 3.3 and conclude that our algorithm successfully creates roadmaps satisfying the five properties.

3.1 Adding useful cycles

In [13], we discussed three methods for decreasing the path length. Although these methods can decrease the path length considerably, they usually do not remove the detours around obstacles. These detours can be avoided by adding cycles to the roadmap. Besides obtaining shorter paths, cycles provide alternative routes for an entity.

In the following sections, we will show how to add useful cycles to the roadmap. Our strategy is partly based on the work of Nieuwenhuisen and Overmars [27] which adds *useful edges* to the roadmap. An edge is useful if it introduces a cycle that improves the roadmap according to some criterion. As we are working with small roadmaps, unfavorably placed queries can still lead to long paths. We will show that these can be avoided by adding *useful nodes* and their corresponding edges to the initial roadmap. The final roadmap will then be composed of all nodes from the initial roadmap, as well as the added useful nodes and useful edges between those nodes.

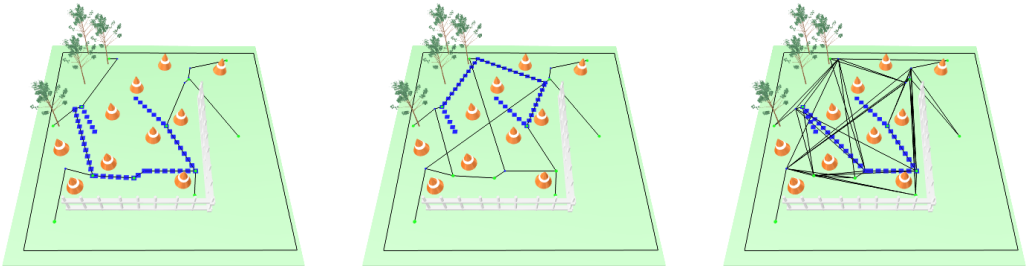
Useful edges

Nieuwenhuisen and Overmars [27] propose a technique that adds useful cycles to the roadmap. The goal is to add only those edges that have a high probability of introducing a path that cannot be continuously deformed into an existing path.

A *useful edge* is defined as follows:

Definition 3 (Useful edge)**.** *Let ν be the node that corresponds to configuration c which has been added to the graph and V^n its set of neighbors. Let ν' be a node in V^n and $d(\nu, \nu')$ be the distance between ν and ν'. The graph distance between ν and ν' is $G(\nu, \nu')$ which is the length of the shortest path in the graph from ν to ν'. If there is no path from ν to ν', $G(\nu, \nu')$ is ∞. Then edge $\epsilon(\nu, \nu')$ is K-useful if $K * d(\nu, \nu') < G(\nu, \nu')$.*

This definition only adds an edge to the graph between ν and ν' if their graph distance improves by a factor K. A small value of K adds more edges than a large value of K. Figure 6 shows that no cycles are added if K is set to ∞. If $K \leq 1$, then all collision-free edges (i.e. local paths) are allowed. The authors use a pruned version of Dijkstra's shortest path algorithm to efficiently determine whether a particular edge is useful. They also show, when time goes to infinity, that their approach will find a path with a length converging to $K * |\Pi|$, where $|\Pi|$ denotes the length of shortest possible path Π. Hence, the larger the number of nodes in the roadmap (and the smaller the value of K), the shorter the expected length of a path. As one of our criteria is obtaining a small roadmap, these two conflicting criteria (short path length and small roadmap) need to be balanced.



(a) $K = \infty$ (b) $K = 1.5$ (c) $K = 1$

Figure 6 A roadmap that contains few nodes can lead to long paths for unfavorably placed queries. Even when parameter K is set to one, i.e. when all collision-free connections are added as edges to the roadmap, the extracted path can be long compared to the optimal path.

Adding useful nodes

In Figure 6, unfavorably placed start and goal positions were added and connected to the roadmap. As few nodes were placed in the middle of the environment, such a small roadmap can yield long paths, making detours around the obstacles. We handle this problem by adding *useful nodes* (and useful edges) to the roadmap, while attempting to keep it small. Accordingly, we define a useful node as follows:

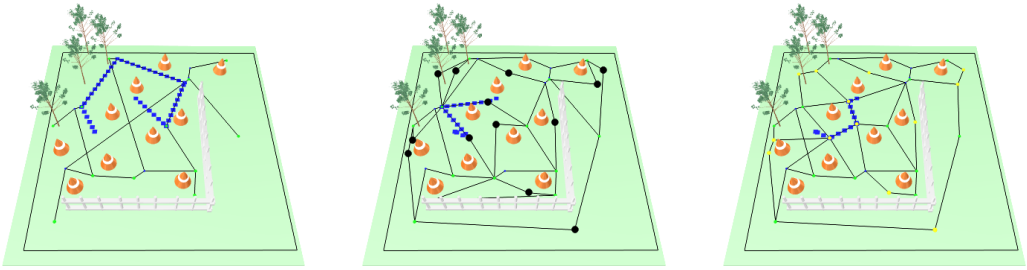
Definition 4 (Useful node)**.** *Let $c \in \mathcal{C}_{\text{free}}$ be a configuration and ν be its representing node. Let $\{\nu', \nu''\}$ be its two closest neighbors in the graph to which a collision-free connection exists, i.e. edge $\epsilon(\nu, \nu') \in \mathcal{C}_{\text{free}}$ and edge $\epsilon(\nu, \nu'') \in \mathcal{C}_{\text{free}}$. Let Π be the shortest path in G between ν' and ν''. If no path exists, then $\Pi = \emptyset$. Then node ν is useful if $\exists \nu_i \in \Pi : \epsilon(\nu, \nu_i) \notin \mathcal{C}_{\text{free}}$.*

As one of our goals is to obtain high-clearance paths, we only select candidate useful nodes that lie on the medial axis. Definition 4 says that a node ν is useful if ν can be connected to two neighbors and the new cycle guides the entity around an obstacle, i.e. there must be at least one connection from node ν to the nodes describing the shortest path Π which causes the moving object to collide with an obstacle. We limit ourselves to checking connections between nodes because checking all connections from node ν to each configuration on path Π will consume too much time.

As a clarification, we apply Algorithm 3 on our running example. Figure 7(b) shows the resulting roadmap. (The black discs represent the useful nodes.) It can be noticed that the useful nodes have a tendency to spread. This can be explained as follows. Suppose that a candidate node ν is very close to a previously added useful node ν' which has already been added and connected to the roadmap. Then it is likely that ν has the same two neighbors as ν' to which free connections exist. (Node ν will not be connected to ν' as useful nodes are only connected to nodes from the initial roadmap to keep the roadmap small.) As their edges will lie close together, it is unlikely that ν is part of a connection that collides with an obstacle. As a result, node ν is not labeled useful according to Definition 4.

Reconnecting the edges

The roadmap quality can be further improved by rearranging its edges. Our approach creates a graph G' that consists of all nodes from graph G. Then, for each pair of nodes, we check



(a) Useful cycles: length=63.8

(b) Useful nodes: length=25.9

(c) Useful nodes and cycles, edges reconnected: length=18.3

Figure 7 Path lengths of an unfavorably placed query. The factor K for adding useful cycles is 1.5.

Algorithm 3 AddUsefulNodes (graph $G(V, E)$)

```
1:  MA ← list of configurations, sampled on the medial axis
2:  for all c ∈ MA do
3:      ν ← node that represents configuration c
4:      ν', ν'' ← the two closest neighbors of the input graph to ν for which edges ε(ν, ν') and
        ε(ν, ν'') ∈ C_free
5:      Π ← shortest path in G from ν' to ν''
6:      for all ν_i ∈ Π do
7:          if ε(ν, ν_i) ∉ C_free then
8:              V ← V ∪ ν
9:              E ← E ∪ ε(ν, ν')
10:             E ← E ∪ ε(ν, ν'')
11:             break
```

if the connection between them is collision-free. Such a connection is added as an edge to G' if the edge is K-useful. The approach is outlined in Algorithm 4. First, we create a priority queue (sorted on increasing edge length) and fill it with all collision-free connections between pairs of nodes from graph G. The improved graph G' will have the same nodes as graph G. The edges of G' consist of edges extracted from the queue if the two nodes do not belong to the same connected component or if the edge introduces a K-useful cycle. Due to the reconnection of the edges, it can occur that a (useful) node is no longer part of a cycle. If such a node has degree one, than this node is removed as our goal is to keep the roadmap small. Figure 7(c) shows the roadmap whose edges have been reconnected.

Figure 7 gives an indication of changes in path length of the unfavorably placed query in several phases of the running example. In each picture, the same initial roadmap was used (which was produced by the RRM). Figure 7(a) shows the path for the roadmap to which useful cycles were added. This path is rather long. A shorter path was found in (b), which shows the roadmap after adding useful nodes and their corresponding connections. Yet a shorter path was found after reconnecting the edges, which is shown in (c).

Algorithm 4 ReconnectEdges (graph $G(V, E)$, factor K)

Output: graph $G'(V', E')$

```
1: PriorityQueue Q {sorted on increasing edge length}
2: for all pair of nodes ν', ν'' ∈ V : ν' ≠ ν'' do
3:    if edge ε(ν', ν'') ∈ C_free then Q.push(ε(ν', ν''))
4: V' ← V
5: E' ← ∅
6: while not Q.empty() do
7:    edge ε(ν', ν'') ← Q.top()
8:    Q.pop()
9:    if K * d(ν', ν'') < G(ν', ν'') then E' ← E' ∪ ε
10: Remove all useful nodes from V' with degree 1
```

3.2 Providing high-clearance paths

The fourth criterion which a roadmap must obey is that high-clearance paths can be obtained in real-time. A path has a high clearance if it follows the medial axis of the free configuration space. We use our retraction algorithm from [13] to retract the paths to the medial axis.

Rather than retracting paths, our goal now is to retract the entire roadmap to the medial axis. To ensure that the complete roadmap will be retracted to the medial axis, we require that its nodes initially lie on the medial axis. The Reachability Roadmap Method already satisfies this requirement. For each edge ϵ, let Π be the local path that corresponds to the edge. We apply the retraction algorithm to each local path Π.

The result of this approach can be viewed in Figure 9. The input graph was created in three steps. A small covering roadmap was produced by the Reachability Roadmap Method. Then useful nodes and edges were added. These edges were then successfully retracted by the appropriate algorithm. Note that some retracted edges have overlapping parts. We do not attempt to merge them as this will increase the number of nodes in the roadmap.

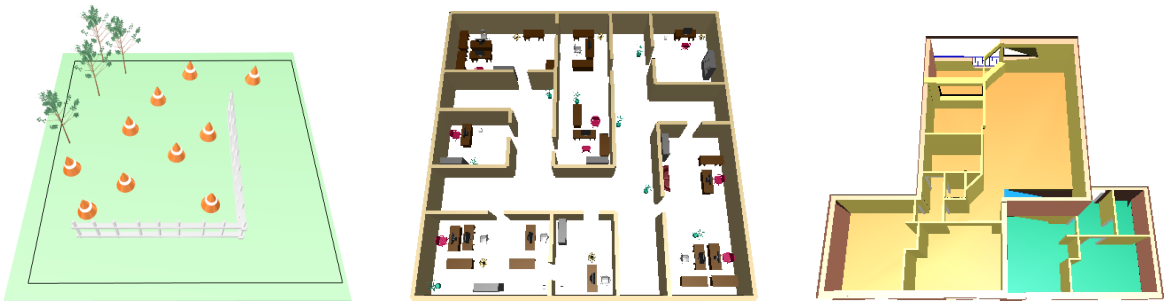
3.3 Experiments

In this section, we test our approach on three virtual environments. In the first part of the experiments, we compare the roadmaps produced by the following three algorithms. The first algorithm, which we refer to as the *Grid Roadmap Method* (GRM), creates a grid. The GRM is constructed by placing a node on each corner of each free grid cell. Edges are added for each boundary and each diagonal of a free cell. This algorithm is often used in the game community. The second algorithm is the Reachability Roadmap Method (RRM). The third algorithm (RRM*) uses the RRM as input and adds useful nodes and edges. Its edges are then rearranged.

In the second part of the experiments, we retract the edges of the roadmaps produced by the RRM* to the medial axis of the free space to obtain high-clearance paths. We refer to this combination of methods as the Retract RRM* (RRRM).

Experimental setup

We conduct experiments with the environments depicted in Figure 8. Information on the environments and robots is listed in Table I. The environments have the following properties:



(a) Field (b) Office (c) House

Figure 8 The three test environments.

Table 4 Information on the environments and robots.

	Bounding boxes		
	number of cells	**robot**	**# objects**
Field	94×94	1×1	16,000
Office	160×160	$1 \times 1 \times 4$	79,000
House	$57 \times 20 \times 40$	$3 \times 3 \times 3$	1,000

Field This small environment contains ten cones, two fences and four trees. These obstacles are cluttered in a large part of the environment. The other part is rather empty. There are many alternative routes. We will test whether our algorithm can capture most of them.

Office This large environment with more than 80 pieces of furniture (79,000 geometrical objects) has a rather non-uniform distribution. There are large open spaces and many narrow passages, requiring a large grid to capture the connectivity of $\mathcal{C}_{\text{free}}$. Also this environment contains many alternative routes. The results will show whether our algorithm can capture them. We also investigate how well our algorithm deals with large environments containing many obstacles.

House This environment has twelve rooms. There are few alternative routes from one room to another room. Hence, we expect that few cycles will be added to the reachability roadmap. Each edge in the roadmap will be retracted to the medial axis of the environment. We will investigate how much the clearance improves along the roadmap.

When we add cycles to a roadmap, we need a way to measure the improvement. For this purpose we define the *Shortest path factor* (SPF):

Definition 5 (Shortest path factor)**.** *Let $G' = (V', E')$ be the graph created by the* GRM*. Let $G = (V, E)$ be a graph for which $V \subseteq V'$. Furthermore, let Ψ be the set of shortest paths in G between each pair of nodes in V, n be the number of paths in Ψ and Ψ' be the set of shortest paths in G' between each pair of nodes in V. Finally, let $|\cdot|$ denote the length of a path. Then the* Shortest path factor *is defined as follows:*

$$\text{SPF} = \sum_{i=1}^{n} |\Psi_i| / \sum_{i=1}^{n} |\Psi'_i|.$$

This definition provides a factor that describes how much longer the expected length of

a path (between two nodes of graph G) is compared to the optimal path length in the grid. If the factor equals one, then each extracted path will be the shortest one in the grid. The larger this factor, the larger the detour made by the moving object.

For all environments and techniques, we define 100 random queries and report how many seconds it takes to solve them. Regarding the Grid Roadmap Method, finding the closest free neighbor can be done in $O(1)$ time. Hence, we only report the time needed for running Dijkstra's shortest path algorithm. In contrast, the other two methods require finding the closest free neighbors *and* running Dijkstra's algorithm.

We also keep track of the sizes of the roadmaps, the construction times, the query times and clearance information (i.e. minimum, average and maximum clearance of configurations in the roadmap). The clearance is measured as the Euclidean distance between the pair of closest points on the moving object and obstacles.

Experimental results

In the following paragraphs, we will describe the results of adding useful cycles and nodes, and adding clearance to the roadmap. See Figure 9 for visualizations of these results.

Adding useful cycles and nodes For each environment, we created a roadmap by applying the Grid Roadmap Method (GRM), the Reachability Roadmap Method (RRM) and the modified RRM (RRM*) method. We set parameter K to 1.5. The results are stated in Table 5 and are discussed below:

Field Even for this relatively small environment, the GRM produced a huge roadmap. As a result, computing 100 random queries took 1.36 seconds. These running times may be acceptable for real-time behavior. An advantage of the method is that a shortest path in the grid will always be found, but as indicated above, the path lacks clearance as it runs very close to obstacles. The RRM created a small roadmap consisting of 29 nodes and 18 edges. As the graph was small, connecting a query to the roadmap and running Dijkstra's algorithm on average took 4.3 ms. Hence, this roadmap can be used in real-time situations. However, due to the small size of the roadmap, the off-center placement of the nodes in the environment, and the fact that the roadmap does not have cycles, the shortest path factor is rather high (1.57). This means that an extracted path between two nodes in the roadmap is on average 57% larger than the corresponding shortest path in the grid of the GRM roadmap. The RRM* added 14 useful nodes and 29 useful edges to the roadmap. As a result, the SPF decreased to 1.137 which means that an extracted path will be much shorter than an extracted path in the RRM roadmap. This did not have a negative impact on the extraction times of the queries.

Office Again, the GRM created a huge roadmap. Finding a shortest path between existing nodes in this roadmap took 34 ms on average. The RRM created a small roadmap. The RRM* added 15 nodes and 33 edges to this roadmap, improving the SPF with 53 percent points. Hence, an extracted path will make less detours. In addition, a close inspection of this roadmap in Figure 9 shows that many alternative routes have been introduced. This did not result in longer query times. The RRM and RRM* methods function well in environments with a lot of detail and many obstacles as the three methods needed only a few seconds.

House A roadmap produced by the GRM for a 3D grid will inevitably become huge, even for a relatively small environment such as this one. Running a query on average took 135 ms, which is far too long in real-time situations. It was surprising to observe that the RRM only needed 34 nodes and 33 edges to get the free space covered and connected. Apparently, the

Table 5 Results for the three environments. Three methods were compared. We collected the following statistical data: the construction time of the roadmap, its number of nodes $|V|$ and edges $|E|$. Then we mention the shortest path factor (SPF) and the summed running time of 100 random queries.

Field	Graph results			Path results	
	time (s)	$\|V\|$	$\|E\|$	**spf**	**100 queries (s)**
grm	0.88	7,350	27,741	1.000	1.36
rrm	0.75	29	18	1.570	0.43
rrm*	0.91	43	47	1.137	0.23

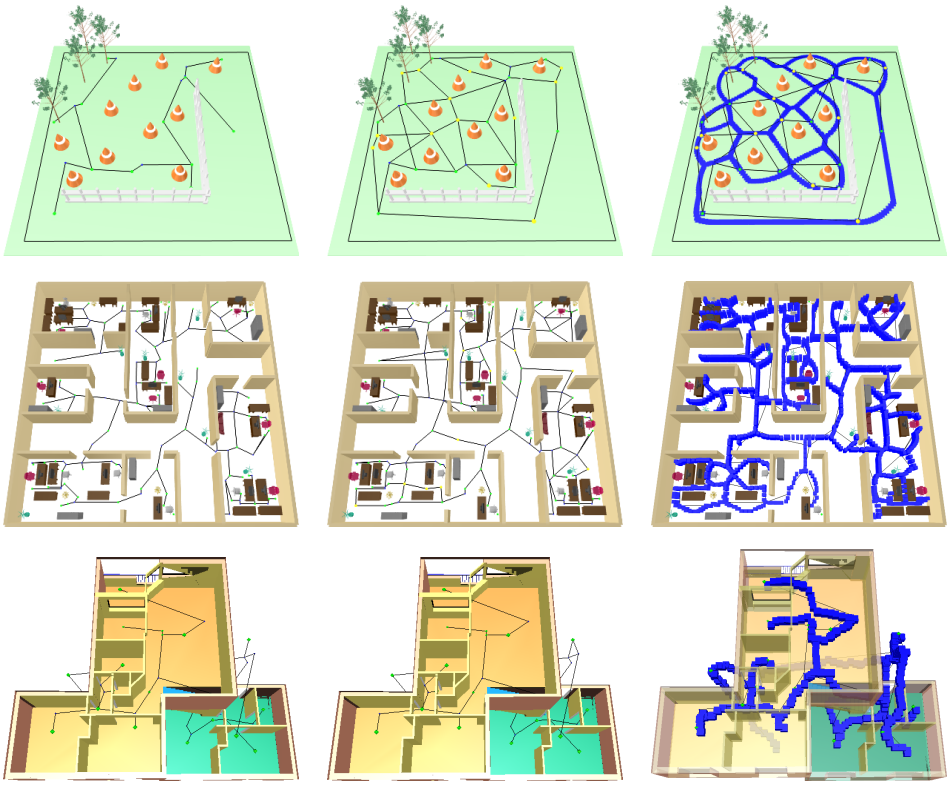
Office	Graph results			Path results	
	time (s)	$\|V\|$	$\|E\|$	**spf**	**100 queries (s)**
grm	0.89	16,917	62,297	1.000	3.42
rrm	1.10	154	147	1.812	0.38
rrm*	2.70	167	180	1.181	0.36

House	Graph results			Path results	
	time (s)	$\|V\|$	$\|E\|$	**spf**	**100 queries (s)**
grm	12.91	40,088	454,250	1.000	13.48
rrm	11.67	34	33	1.225	0.82
rrm*	18.68	34	34	1.224	0.82

nodes were properly placed in each room, as the RRM* did not add any useful node to the roadmap. Only one edge was added. Running a query took on average 8 ms. An extracted path will be short, because, on average, it will be only 23% larger than the corresponding optimal path in the grid.

In conclusion, roadmaps created by the GRM rapidly become huge, especially when 3D grids are required, resulting in query times that are too high for real-time performance. In practice, much smaller grids are used but this can be problematic when there are narrow passages. In contrast, the RRM creates small roadmaps that completely cover the free space and have low query times. However, the extracted paths can be long. The RRM* combines the advantages of GRM and RRM, i.e. reasonably short paths are produced while the extraction times are kept relatively low. The resulting paths were on average 14 to 23 percents larger than the optimal paths in a corresponding grid. Since the RRM* placed the nodes on the medial axis, a large clearance caused the nodes to lie far away from the obstacles, increasing the path length.

Adding clearance As indicated above, high-clearance paths are often preferred. Such paths can be obtained by retracting each of the three roadmaps created by the RRM* to the medial axis. We refer to this method as the Retracted RRM* (RRRM). We compare the clearance of roadmaps produced by the RRM* and RRRM. Table 6 shows the corresponding results. In all retracted roadmaps, the (minimum, maximum and average) clearance was improved. The times needed for the retraction were reasonably fast for the Field and House



(a) RRM (b) RRM* (c) RRRM

Figure 9 The results for the Field, Office and House environments. The left column shows the initial roadmaps created by the Reachability Roadmap Method. The middle column shows the roadmaps to which useful nodes and cycles were added (RRM*). The right column shows the roadmaps to which clearance was added (RRRM).

environments. The roadmaps for Office and Quake environments were retracted within six minutes. The main reason for this difference is that the latter two roadmaps are considerably larger than the other two. We expect that the running times of the retraction algorithms can be dramatically decreased by incorporating learning techniques.

4 Conclusion

We presented a method that automatically computes high-quality roadmaps for 2D and 3D environments.

The initial roadmap was created by the *Reachability Roadmap Method* (RRM), which is a new and efficient method that creates small roadmaps for two- and three-dimensional motion planning problems. The RRM ensures that every valid query can be connected to the roadmap, as is required to solve the problem. If there exists a path in the (discretized) free space that connects the query, the algorithm ensures that a path can be found in the roadmap.

We compared the RRM with the PRM and the Visibility PRM. The RRM created roadmaps with the fewest number of nodes and edges in all environments. While the PRM produced large roadmaps, the Visibility PRM created relatively small roadmaps but had great difficulties

Table 6 Clearance and time statistics for RRM* and retracted RRM* (RRRM) roadmaps. Results corresponding to the RRRM are the averages over 100 independent runs.

Field	Clearance			Time
	min	**avg**	**max**	**s**
rrm*	0.03	2.71	6.44	
rrrm	0.34	3.08	6.46	24

Office	Clearance			Time
	min	**avg**	**max**	**s**
rrm*	0.00	1.60	6.82	
rrrm	0.01	1.77	7.53	320

House	Clearance			Time
	min	**avg**	**max**	**s**
rrm*	0.00	2.17	5.64	
rrrm	0.13	3.33	10.41	49

Quake	Clearance			Time
	min	**avg**	**max**	**s**
rrm*	0.00	2.90	9.45	
rrrm	0.05	3.28	9.75	343

in getting the free space covered and connected. It was surprising to see how few nodes are actually required for covering roadmaps. For difficult environments containing narrow passages, the RRM was faster than the other two techniques. In these environments, the PRM and Visibility PRM consumed much time before obtaining a path through the narrow passages. For easy environments, the RRM will be outperformed by the PRM as all samples will reach a large portion of the free space. When the number of dimensions increases and the problem does not contain very narrow passages, the RRM will be outperformed (in running time) by the PRM, as the PRM is less sensitive to the dimensionality of the problem. In addition, full coverage of the $\mathcal{C}$-space is not required in many motion planning problems as queries will be 'reasonable'. In such situations the preprocessing times for the PRM and the Visibility PRM will be much lower. But the roadmaps they produce will still be large, leading to increased query times.

As paths, extracted from RRM roadmaps, can make long detours around obstacles, we provided a method that adds useful cycles to the roadmap. Experiments showed that alternative and reasonably short paths can be extracted from the enhanced roadmap. The query times on this roadmap are low which enables real-time extraction of paths. Another criterion the roadmap should satisfy is that high-clearance paths can be extracted without extra computation time in the query phase. This criterion was met by retracting the edges of the roadmap to the medial axis of the free space. While this may take a reasonable amount of time, we believe that the running times of the retraction algorithm can be improved dramatically by incorporating learning techniques.

An interesting topic for future research is how to efficiently extend the RRM to higher dimensions. A roadmap that is built for a lower dimensional subspace could be used to guide the motions for an object operating in a high-dimensional $\mathcal{C}$-space (see e.g. [8, 3, 12]). We believe that the method presented in this paper will enhance the quality of motion planners, in particular when query time is the major concern. The method could further be extended to incorporate extra constraints that level designers put on the roadmaps. For example, one could incorporate non-holonomic constraints, walkable surfaces, surface properties and incorporate safety information in the roadmap.

Acknowledgments

Part of this research has been funded by the Dutch BSIK/BRICKS Project.

References

[1] J. Barraquand, L.E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16:759–744, 1997.

[2] J.P. van den Berg, D. Nieuwenhuisen, L. Jaillet, and M.H. Overmars. Creating robust roadmaps for motion planning in changing environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2515–2521, 2005.

[3] J.P. van den Berg and M.H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *International Journal of Robotics Research*, 24:1055–1072, 2005.

[4] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2003.

[5] V. Boor, M.H. Overmars, and A.F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[6] M. Branicky, S.M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation*, pages 1481–1487, 2001.

[7] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.

[8] O. Brock and L.E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 1469–1475, 2001.

[9] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19:96–125, 2000.

[10] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, first edition, 2005.

[11] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press/ McGraw-Hill Book Company, second edition, 2001.

[12] M. Foskey, M. Garber, M. Lin, and D. Manocha. A Voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 55–60, 2001.

[13] R. Geraerts and M.H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2386–2392, 2004.

[14] R. Geraerts and M.H. Overmars. Creating small roadmaps for solving motion planning problems. In *IEEE International Conference on Methods and Models in Automation and Robotics*, pages 531–536, 2005.

[15] R. Geraerts and M.H. Overmars. On improving the clearance for robots in high-dimensional configuration spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4074–4079, 2005.

[16] R. Geraerts and M.H. Overmars. Reachability analysis of sampling based planners. In *IEEE International Conference on Robotics and Automation*, pages 406–412, 2005.

[17] R. Geraerts and M.H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous System*, 54:165–173, 2006.

[18] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 4420–4426, 2003.

[19] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, 1992.

[20] A. Kamphuis and M.H. Overmars. Finding paths for coherent groups using clearance. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 19–28, 2004.

[21] L.E. Kavraki, P. Švestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[22] J.-C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[23] S.M. LaValle. *Planning Algorithms*. http://msl.cs.uiuc.edu/planning, 2005.

[24] Y.-H. Lee, T.-W. Kao, and S.-S. Lee. Optimal parallel algorithms for computing the chessboard distance transform and the medial axis transform on RAP. In *IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pages 22–28, 1996.

[25] J.-M. Lien, S. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, pages 4439–4444, 2003.

[26] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M.H. Overmars. Automatic construction of roadmaps for path planning in games. In *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 285–292, 2004.

[27] D. Nieuwenhuisen and M.H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation*, pages 446–452, 2004.

[28] C. Nissoux, T. Siméon, and J.-P. Laumond. Visibility based probabilistic roadmaps. *Advanced Robotics Journal*, 14:477–493, 2000.

[29] J. O'Rourke. *Art Gallery Theorems and Algorithms*. New York: Oxford University Press, 1987.

[30] M.H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, 1992.

[31] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.