Creating Robust Roadmaps for Motion Planning in Changing Environments*

Jur P. van den Berg[1] Dennis Nieuwenhuisen[1] Léonard Jaillet[2] Mark H. Overmars[1]

[1]*Institute of Information and Computing Sciences*
Utrecht University, The Netherlands
{berg,dennis,markov}@cs.uu.nl

[2]*LAAS-CNRS*
Toulouse, France
leonard.jaillet@laas.fr

***Abstract*— In this paper we introduce a method based on the Probabilistic Roadmap (PRM) Planner to construct robust roadmaps for motion planning in changing environments. PRM's are usually aimed at static environments. In reality though, many environments are not static, but contain moving obstacles as well. Often the motion of these obstacles is not unconstrained, but is restricted to some confined area, e.g. a door that can be open or closed or a chair which is bounded to a room. We exploit this observation by assuming that a moving obstacle has a predefined set of potential placements. We present a variant of PRM that is robust against placement changes of obstacles. Our method creates a roadmap that is guaranteed to contain a path for any feasible query when time goes to infinity, i.e. the method is probabilistically complete. Our implementation shows that after a roadmap is created in the preprocessing phase, queries can be solved instantaneously, thus allowing for on-the-fly replanning to anticipate changes in the environment.**

***Index Terms*— motion planning, changing environments, PRM**

I. Introduction

Motion planning is of great importance in the field of robotics. Much research has been done on motion planning in static environments, and one of the most successful methods that emerged the last decade is the Probabilistic Roadmap (PRM) Planner [1], [2], [7], [11], [12]. The general idea behind it is that a roadmap is created in a preprocessing phase that well covers the free space of the robot, such that individual motion plans can be generated very quickly in a query phase. The roadmap is constructed by connecting randomly sampled collision-free configurations by a local planner (which typically creates straight-line motions). A query is processed by connecting the start and goal configurations to the roadmap and searching the augmented roadmap for a feasible path.

In reality though, many environments are not static, but contain dynamic obstacles as well that change their positions over time. For such environments the PRM method does not work, because the premise that planning will occur many times in the same environment does not hold anymore. However, in many applications the environment contains information about which dynamic changes are possible (see Fig. 1). For example, it is normally known

*This research was supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2001-39250 (MOVIE - Motion Planning in Virtual Environments).

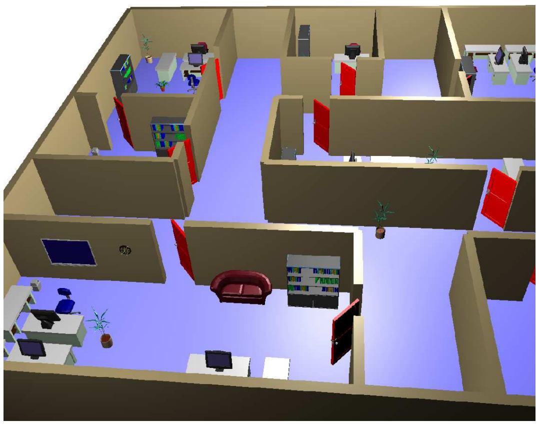


Fig. 1. A typical example of a scene containing moving obstacles. Although most objects can change their position, it is expected that they are always situated in the same room.

where the doors are that can be opened, and we normally know in what states the door can be (anywhere between open and closed). Another example is a chair of which we know that it is located 'somewhere' in a room. We refer to obstacles for which the set of *potential* placements is known a priori as *moving obstacles*. These include obstacles that can be removed from (and re-added to) a scene.

Note that we are not concerned here with motion planning in *dynamic* environments where continuously moving obstacles are avoided using temporal coordination (see e.g. [3]). We rather focus on *changing* environments, in which obstacles occasionally change their placements. We assume that we know for each moving obstacle its set of potential placements, and exploit this knowledge specifically. Yet, we put no constraints on the nature of this set.

The knowledge of the potential placements of the moving obstacles can be employed to create roadmaps that incorporate the possible changes of the environment, such that during the query phase the changes can be anticipated very quickly. A straightforward approach would be to create the roadmap as if none of the moving obstacles is present and then mark parts of the roadmap as temporarily blocked as long as they are occupied by a moving obstacle. This approach is roughly followed by [10]. A problem with this approach is that it requires the roadmap to cover the topology of the free space independent of the placements of the moving obstacles, yet [10] does not guarantee this.

In this paper we introduce a method that *does* guarantee the above, and produces roadmaps for changing environments that are *robust*. By robust we mean that for any placement of the moving obstacles the roadmap contains a path between any pair of query configurations, provided that such a path actually exists.

Our approach directly follows the PRM paradigm. To save time in the construction of the roadmap, standard PRM implementations do not add edges between nodes that were already in the same connected component of the roadmap. To guarantee the creation of a robust roadmap however, our method does add such edges, but only if they are *necessary*, that is when the already present paths between the nodes may possibly be invalidated by some placements of the moving obstacles, while the new edge is not.

Of all edges in the roadmap we know for which placements of the moving obstacles they are valid. This ensures that queries can be performed very quickly, and facilitates on-the-fly replanning to anticipate changes in the environment.

Our method is as generic as PRM, which means that it can be used for a broad range of robot types in configuration spaces of any dimension, and in combination with any sampling scheme or local planner. We implemented our method and experiments show that in reasonable construction time, robust roadmaps can be produced for realistic environments. Queries are solved instantaneously, allowing for real-time on-line planning.

A. Related work

Only a few papers have been written on motion planning in changing environments that use roadmaps. The most important one is [10], from which some principles are used in our method as well. It uses a voxel grid covering the workspace, and stores for each roadmap edge which voxels are swept by the robot. When some obstacle changes position, it is easy to identify the edges invalidated by the obstacle. This is done by checking the voxels covered by the obstacle for overlap with the swept voxels of each of the edges. In the remaining roadmap a path can then quickly be found.

A more ad-hoc approach is presented in [6]. A normal cycle-free PRM-roadmap is constructed for the static part of the environment. If during a query, an edge crucial for finding a path appears to be in collision with a moving obstacle, a variant of RRT (Rapid-exploring Random Trees) [8] is used to connect the nodes of the invalidated edge. If this fails, the two disconnected components of the roadmap are attempted to be reconnected by additional global sampling.

The main difference between these methods and our method is that the above methods do not concern the creation of a robust roadmap, but rather focus on the problem of how to deal with the changed obstacles in the query phase. They do, however, not assume any knowledge of the potential placements of the moving obstacles. We do assume this, allowing us to create roadmaps that are guaranteed to contain a path for any feasible query. Also, obstacle changes can be anticipated very fast during runtime, as we store for each edge for which placements of the obstacles it is valid.

B. Organization of this paper

The rest of this paper is organized as follows. In Section II we give a general solution for creating robust roadmaps. We discretize the problem in Section III, and give some simple examples concerning one moving obstacle. In Section IV we extend the method to work with multiple moving obstacles. The query phase is discussed in Section V, and we show experiments and results in Section VI.

II. PRM FOR CHANGING ENVIRONMENTS

A changing environment consists of a robot, static obstacles and a set of moving obstacles. For now, we restrict ourselves to one moving obstacle O (in Section IV we lift this restriction). For O, a set of potential placements $P(O)$ is defined. This set contains, in general, an infinite number of continuous placements, but it may also contain a finite number of discrete placements. Each $p \in P(O)$ maps the moving obstacle to a placement in the scene.

A query is defined as a triple (s, g, p), where s and g are the start and goal configurations of the robot, and p denotes a placement of the moving obstacle. Here, we devise an algorithm, based on the PRM-planner, which creates a roadmap (composed of a set N of nodes and a set E of edges) that is robust against the placement of the moving obstacle. The algorithm is probabilistically complete, i.e. if the method runs long enough, the roadmap contains a solution for every feasible query.

A. General solution

A PRM planner randomly samples configurations in the configuration space C of the robot. They are added as nodes to the roadmap if the robot is collision free for that configuration. Since, in changing environments, we also have the moving obstacle O to take into account, we extend the definition of 'collision free' to the case of changing environments:

Definition II.1 (collision_free)**.** *For a configuration $c \in C$ of the robot and a placement $p \in P(O)$ of the moving obstacle, we define the function $\mathrm{collision\_free}(c, p)$ to be true if the robot at configuration c does not collide with the static obstacles and the moving obstacle placed at p.*
For two configurations $c_0, c_1 \in C$, and a placement $p \in P(O)$ of the moving obstacle, we define the function $\mathrm{collision\_free}(c_0, c_1, p)$ to be true if the path generated by the local planner between c_0 and c_1 does not collide with the static obstacles and the moving obstacle placed at p.

A configuration c is added as a node to the roadmap if $\mathrm{collision\_free}(c, p)$ evaluates to true for at least one placement p of the moving obstacle.

Subsequently, in standard PRM connections are tried to other nodes that are not in the same *connected component* as c. Here, we need new definitions for 'connected' and 'component' in order to take the moving obstacle into account.

Definition II.2 (connected)**.** *For two nodes n_0 and n_1 in the roadmap and a placement $p \in P(O)$ for the moving obstacle, we define the function* $\text{connected}(n_0, n_1, p)$ *to be true if there is a valid path in the roadmap connecting n_0 and n_1 when the moving obstacle is placed at p.*

Instead of connected components, we use the notion of *labeled components*. They consist of nodes that are connected regardless of the placement of the moving obstacle.

Definition II.3 (labeled component)**.** *Two nodes n_0 and n_1 belong to the same* labeled component L *if* $\forall(p \in P(O) \mid \text{connected}(n_0, n_1, p))$.

The collection $\{L_0, L_1, \ldots\}$ of all labeled components is a *partition* of the set of nodes N in the roadmap, so:

- $L_0 \cup L_1 \cup \ldots = N$
- $L_i \cap L_j = \emptyset$ when $L_i \neq L_j$

Note that two nodes that belong to different labeled components can still be connected by an edge. This means that such an edge only connects the labeled components for some placements of O and does not connect them for other placements. For each pair of labeled components, we need to maintain the set of placements of O for which they are connected. Such a set is called a *connection set*.

Definition II.4 (connection set)**.** *For every pair of labeled components L_0 and L_1, a* connection set $F(L_0, L_1) \subset P(O)$ *is defined as the set of placements of O for which the two labeled components are connected:*

$$F(L_0, L_1) = \{p \in P(O) | \text{connected}(n_0, n_1, p)\}, \quad \text{where } n_0 \in L_0, n_1 \in L_1$$

In standard PRM, an edge is added if it connects two different connected components. In our algorithm we only add an edge if it connects two different labeled components *and* if it extends the connection set of the two labeled components. To state this formally, we introduce the notion of *necessary* edges.

Definition II.5 (necessary)**.** *For two nodes $n_0 \in L_0$ and $n_1 \in L_1$, we define the edge (n_0, n_1) to be* necessary *if it extends the connection set between the two:*

$$\{p \in P(O) | \text{collision\_free}(n_0, n_1, p)\} \not\subset F(L_0, L_1)$$

If a necessary edge is added, then by definition it extends the connection set of its two incident labeled components, so we need to update this connection set. This update may extend all other connection sets as well, because new routes through the roadmap have become available. The new information needs to be *propagated* through the roadmap to update the other connection sets. This is discussed in Section II-C.

We now have all the ingredients for an algorithm that creates a robust roadmap for changing scenes (Algorithm 1). Just as with standard PRM, the algorithm repeats until some stop-criterion is met, for example a pre-specified amount of time.

Algorithm 1 shows a general solution to our problem. It is not yet a practical solution, as in lines 3 and 9 a possibly infinite set is evaluated. In Section III we resolve this by discretizing the set of placements of the moving obstacle.

In line 12 the propagation algorithm is called to update the connection sets. Because of this, some labeled components may need to be merged. This is done in line 13. Both algorithms are detailed in Section II-C.

Algorithm 1 CONSTRUCTROBUSTROADMAP

```
1:  repeat
2:      sample a configuration $c \in C$
3:      if $\exists(p \in P(O) \mid \text{collision\_free}(c, p))$ then
4:          add $c$ as a node $n$ to the roadmap
5:          initialize labeled component $L \leftarrow n$
6:          for all labeled components $L_i$, $L_i \neq L$ do
7:              initialize $F(L, L_i) \leftarrow \emptyset$
8:          for all nodes $n_i \in N, n_i \neq n$ do
9:              if edge $(n, n_i)$ is necessary (see Definition II.5) then
10:                 Add edge $(n, n_i)$ to the roadmap
11:                 $P_f \leftarrow$ the placements of $O$ for which $(n, n_i)$ is free
12:                 PROPAGATE($L_n, L_{n_i}, P_f$) ($L_n$ and $L_{n_i}$ are the labeled components of nodes $n$ and $n_i$ respectively)
13:                 MERGELABELEDCOMPONENTS
14: until some stop-criterion is met
```

B. Probabilistic completeness

Our method is probabilistically complete. That is, if time goes to infinity the roadmap is guaranteed to contain a solution path for any feasible query. We sketch a proof here, which roughly follows the proof for standard PRM as given in [12].

Given is an environment with static obstacles, a moving obstacle and a query (s, g, p) for which there exists a path. We cover this path with overlapping free balls. The connection made by the local planner between configurations in two neighboring balls is collision free with respect to the static obstacles and to placement p of the moving obstacle. There are two possibilities: either this connection is necessary and added to the roadmap, or this connection is not necessary because there already exists a path in the roadmap between the two configurations that is collision free with respect to p (see Definition II.5). In either case the two configurations will be connected. As the probability that each ball contains a configuration approaches 1 when the running time goes to infinity, it is guaranteed that a path will be found between s and g for placement p.

C. Updating the connections sets

As stated previously, after adding an edge between two labeled components, we need to update their connection set and propagate the new information to all other connection sets (line 12 of Algorithm 1). For this purpose we need the notion of *neighboring* labeled components.

Definition II.6 (neighboring)**.** *Two labeled components L_0 and L_1 are* neighboring *each other if there exists an edge between the two labeled components:*

$$\exists(n_0 \in L_0, n_1 \in L_1 \mid (n_0, n_1) \in E)$$

If we add an edge between labeled components L_0 and L_1 we first update the connection set $F(L_0, L_1)$. Then, to propagate this new information to the other connection sets, we recursively update all neighbors of L_0 and then all neighbors of L_1. The propagation algorithm is shown in Algorithm 2. Its input consists of the two labeled components for which connectivity is added and the placements $P_f \subset P(O)$ for which the edge is collision free. Every connection set is updated at most once.

Algorithm 2 PROPAGATE (L_a, L_b, P_f)

```
1: $F(L_a, L_b) \leftarrow F(L_a, L_b) \cup P_f$
2: for all neighbors $L_i$ of $L_a$ do
3:     $P_f \leftarrow F(L_a, L_b) \cap F(L_a, L_i)$
4:     if $P_f \not\subset F(L_b, L_i)$ then
5:         PROPAGATE $(L_b, L_i, P_f)$
6: for all neighbors $L_i$ of $L_b$ do
7:     $P_f \leftarrow F(L_a, L_b) \cap F(L_b, L_i)$
8:     if $P_f \not\subset F(L_a, L_i)$ then
9:         PROPAGATE $(L_a, L_i, P_f)$
```

After running Algorithm 2 it is possible that nodes that used to belong to different labeled components now belong to the same labeled component according to Definition II.3. Stated differently, for every placement of the moving obstacle there is a connection between the labeled components of these nodes. We call the connection set of these labeled components *complete*.

Definition II.7 (complete)**.** *The connection set $F(L_0, L_1)$ of two labeled components L_0 and L_1 is* complete *if for every placement of the moving obstacle O, L_0 and L_1 are connected:*

$$F(L_0, L_1) = P(O)$$

If a connection set is complete, we need to *merge* the associated labeled components. This process is detailed in Algorithm 3.

Algorithm 3 MERGELABELEDCOMPONENTS

```
1: for all sets $F(L_0, L_1)$ that have been updated in
   PROPAGATE do
2:     if $F(L_0, L_1) = P(O)$ then
3:         $L_0 \leftarrow L_0 \cup L_1$.
4:         Remove the connection sets incident to $L_1$
5:         Append neighbor list of $L_1$ to neighbor list of $L_0$
6:         Relabel the nodes in $L_1$.
```

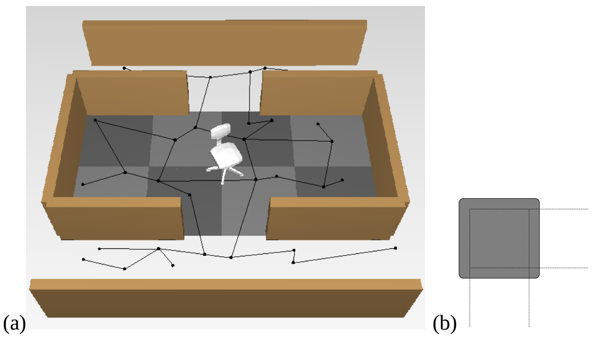


Fig. 2. (a) The subdivision of the set of placements of a moving obstacle (the chair) in 8 chunks, and the associated robust roadmap for a cylindrical robot. (b) The sweep volumes of the chunks overlap, this is shown for the upper left chunk.

III. DISCRETIZATION

In our algorithm, we need to check for which placements of the moving obstacle O the nodes and edges are collision free (see lines 3 and 9 of Algorithm 1), provided that they do not collide with the static obstacles (see Definition II.1). In case of a discrete set of placements $P(O)$, we can check against each individual element of this set. For continuous sets however, it is practically impossible to check for which placements the nodes and edges collide. Therefore, we partition $P(O)$ into a finite set of *chunks* $X(O)$.

Definition III.1 (chunk)**.** *A* chunk *$\chi \in X(O)$ is a subset of the placements of obstacle O: $\chi \subset P(O)$.*

The definition of $X(O)$ as a partition of $P(O)$ implies the following concerning the chunks:

- $\bigcup X(O) = P(O)$
- $\chi_i \cap \chi_j = \emptyset$ when $\chi_i \neq \chi_j$

Instead of checking for which *placements* of the moving obstacle a node or edge is collision free, we now check for which *chunks* of the moving obstacle the node or edge is collision free.

We observe that, in order to preserve completeness, we need to choose the chunks such that if the problem is solvable for a placement $p \in P(O)$ of the moving obstacle, the problem is also solvable for the chunk that contains p. If we would partition $P(O)$ in an infinite number of chunks, then every chunk contains only one placement, which would restore the undiscretized situation. In most cases, however, it is sufficient to partition $P(O)$ into only a small number of chunks.

Fig. 2 shows an example of the subdivision of the potential placements of a moving obstacle. The obstacle is a chair with two degrees of freedom that can be placed freely inside a room. The robot is a small cylinder. The set of placements of the chair is subdivided into 8 different chunks. A resulting roadmap is shown. The two parts of the roadmap above and below the room are always connected regardless of the chunk the chair is in, so they belong to the same labeled component. The sweep volumes of the chunks in workspace will overlap. This overlap is omitted in Fig. 2a, but is shown in Fig. 2b.

Nodes that collide with one of the chunks never have free connections to other nodes for all moving obstacle placements. This means that every node that collides with the sweep volume of the moving obstacle is a labeled component by itself for which we need to maintain connection sets to all other labeled components. Luckily we can merge certain colliding nodes to one labeled component by relaxing Definition II.3: If the sets of chunks for which two nodes are free, are both equal to the set of chunks for which the nodes are connected, we can merge their labeled components. To this end, line 2 of Algorithm 3 should be adapted. Note, that this new definition does not change the definition of labeled components for nodes that are free for all chunks.

Definition III.2 (labeled component)**.** *Two nodes* n_0 *and* n_1 *belong to the same* labeled component L *if:*

$$\begin{aligned}&\{\chi \in X(O) | \text{collision\_free}(n_0, \chi)\} = \\ &\{\chi \in X(O) | \text{collision\_free}(n_1, \chi)\} = \\ &\{\chi \in X(O) | \text{connected}(n_0, n_1, \chi)\}\end{aligned}$$

A. Implementation issues

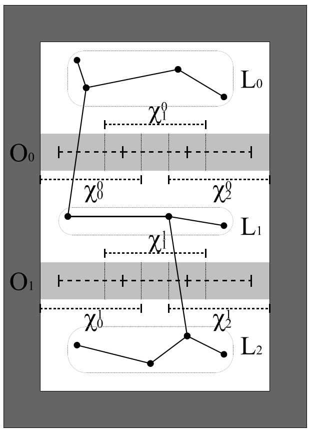
To find out whether an edge between two labeled components is necessary (see Definition II.5), we do not need to collision-check the edge with all chunks. Only the chunks for which there is no connection yet need to be checked. If for one or more of those the edge is free, the edge extends the connection set and hence it is necessary.

An important issue is how to create the chunks. A straightforward solution is to partition the range of every degree of freedom of the moving obstacle into a collection of segments. The Cartesian products of these segments then form the chunks. To be able to collision-check against a chunk, we must create an object that represents (a superset of) the workspace sweep volume of the chunk. For this purpose we use principles from [10], by covering the workspace with a voxel grid. The object is then formed by the collection of voxels that is swept by a number of obstacle placements sampled evenly over the chunk. The sampling density should correspond to the resolution of the voxel grid, which determines the accuracy of the representation.

IV. Multiple obstacles

The example we saw in the previous section concerned only one moving obstacle, but the generic description of our solution applies to the case of multiple moving obstacles as well. Multiple obstacles can be regarded as one composite obstacle whose set of degrees of freedom is just the concatenation of the degrees of freedom of each individual obstacle.

Let us assume that we have k obstacles $O_0, O_1, \ldots, O_{k-1}$ and that the set of placements $P(O_i)$ of each individual obstacle O_i is partitioned into a set of chunks $X(O_i) = \{\chi_0^i, \chi_1^i, \ldots\}$, just as we did in the previous section. The set of chunks $\mathcal{X}$ of the composite obstacle is then the Cartesian product of the sets of chunks of the individual obstacles: $\mathcal{X} = X(O_0) \times X(O_1) \times \cdots \times X(O_{k-1})$. An element of $\mathcal{X}$, a composite chunk, is denoted by $(\chi_{j_0}^0, \chi_{j_1}^1, \ldots, \chi_{j_{k-1}}^{k-1})$, where the j's are indices of individual obstacle chunks.

(a) 

(b)

$$\begin{array}{c|ccc} & & F(L_0, L_2) & \\ \chi_2^1 & 0 & 0 & 0 \\ \chi_1^1 & 0 & 0 & 0 \\ \chi_0^1 & 0 & 1 & 1 \\ \hline & \chi_0^0 & \chi_1^0 & \chi_2^0 \end{array}$$

Fig. 3. (a) An example environment with two moving obstacles (square pillars that can slide along the dashed lines), each partitioned into three chunks (the gray regions are the workspace sweep volumes of the chunks). (b) A representation of the connection set $F(L_0, L_2) \subset \mathcal{X}$ of labeled components L_0 and L_2 of Fig. a. A 0 means that there is no connection, a 1 means that there is a connection.

Using this set $\mathcal{X}$, we can use the algorithm from the previous section without any changes, that is, for each combination of individual obstacle placements (the composite chunks) we maintain whether two labeled components are connected. In Fig. 3 an example representation is given of a connection set in an environment with two moving obstacles, of which each placement set is partitioned into three chunks. Since two moving obstacles are involved, we can think of the connection sets as two-dimensional grids of booleans, where the chunks of each individual obstacle are listed along the axes. A grid cell models a composite chunk of the respective individual chunks along the axes. For instance the connection set depicted in Fig. 3b indicates that there is a connection between labeled components L_0 and L_2 of Fig. 3a when moving obstacle O_0 is positioned inside chunk χ_1^0 or χ_2^0, and O_1 is positioned inside chunk χ_0^1; i.e. $\text{connected}(L_0, L_2, (\chi_1^0, \chi_0^1))$ and $\text{connected}(L_0, L_2, (\chi_2^0, \chi_0^1))$ are true.

If we have k moving obstacles, the grids become k-dimensional and if the placement set of each obstacle O_i is partitioned into m_i chunks, the total number of chunks of the composite obstacle is $\prod_{i=0}^{k-1} m_i$. However, when an edge is considered for addition to the roadmap, we do not need to perform collision checks with all chunks of the composite obstacle to check whether the edge is free. It suffices to check whether the edge is free with respect to the chunks of each of the individual obstacles. So in total only $\sum_{i=0}^{k-1} m_i$ collision checks have to be performed. From these checks, we can easily deduce for which composite chunks the edge is free: $\text{collision\_free}(n_0, n_1, (\chi_{j_0}^0, \chi_{j_1}^1, \ldots, \chi_{j_{k-1}}^{k-1})) = \text{collision\_free}(n_0, n_1, \chi_{j_0}^0) \wedge \text{collision\_free}(n_0, n_1, \chi_{j_1}^1) \wedge \cdots \wedge \text{collision\_free}(n_0, n_1, \chi_{j_{k-1}}^{k-1})$.

A. Implementation issues

For each connection set $F(L_0, L_1)$ we need to maintain for all $\prod_{i=0}^{k-1} m_i$ composite chunks whether the two labeled components L_0 and L_1 are connected. A connection set is straightforwardly represented by an array of booleans, and the operations on connection sets we need in the algorithm (union, intersection, test for subset and completeness) are easily implemented for such a representation.

Another issue concerns how to store the total collection of connection sets. In principle a connection set is maintained for every pair of labeled components. From analysis on the standard PRM method we know that the number of connected components increases quickly in the first part of the execution of the algorithm, and decreases later on as the components get connected. We are likely to see the same behavior for labeled components in our method. When the number of components is maximal, many of them are not connected to each other. This also means that the associated connection sets are empty. Therefore, we choose to only store non-empty connection sets. A convenient data structure for this purpose is a map. Identifying an empty connection set is then done by finding it absent in the map.

In the current algorithm, connection sets are also maintained for labeled components that are ‘far away’ from each other, but are interconnected via many other labeled components. These connection sets are not so interesting, as the probability that a direct edge is added between the labeled components is small. Yet, storing such connection sets takes as much memory as others.

We can repair this deficiency by changing the propagation algorithm (Algorithm 2): instead of propagating the information through the roadmap until no expansions of connection sets are achieved any more, we may put a maximum $D_{\max}$ on the depth of the propagation, i.e. connection sets between labeled components that are connected to each other via more than $D_{\max}$ other components are not updated and propagated. This means that these connection sets remain empty, and therefore do not need to be stored. Also, the propagation algorithm becomes faster. The drawback is that the information stored in all connection sets is not complete anymore. This may cause some extra edges to be added, but we expect this number to be very limited.

B. An example

A nice example is the puzzle environment depicted in Fig. 4. It has four doors that all have two potential placements, each of them blocking a different path through the environment. Although not trivial to observe, the goal position g indicated in the figure can be reached from the start position s for every combination of placements of the doors. The algorithm will quickly detect this because s and g eventually belong to the same labeled component.

The region around the top-left static obstacle in the environment can completely be sealed off from the rest of the environment. Hence, a different labeled component will be formed in this region. Yet, a path from s to g for the placement combination indicated in the figure (the dark

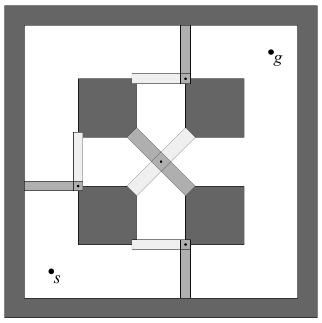


Fig. 4. A puzzle environment. The goal position g can be reached from the start position s for every placement combination of the doors. Each of the two placements of every door is modeled as a chunk.

grey doors are the actual placements) goes through this top-left region. This means that if two nodes belong to the same labeled component, their connection may go through other labeled components as well.

V. Query phase

The roadmaps created by our algorithm are robust against changes of placements of obstacles. During the construction of the roadmap we stored for each edge for which placements of the moving obstacles it is free (see line 11 of Algorithm 1). We exploit this property to quickly compensate for placement changes of obstacles during the execution of a query. Experiments show that after preprocessing, queries can be performed instantaneously. To find a path between a given start and goal position, we first search for a path in the roadmap using a shortest path algorithm, given the current placements of the obstacles. Next, the robot starts executing the resulting path. If, during the execution of the query an obstacle changes its placement, we immediately know if one of the edges of the query path is invalidated. If this is the case, we replan by querying from the current robot position to the goal position. Since our roadmap is robust, we are again guaranteed to find a new path if the query is feasible.

VI. Experiments

We implemented our algorithm in C++ using SOLID [4] as a collision checker, and conducted experiments on 3 changing environments each representing a realistic example. We will now describe these experiments in detail.

The scene of the first experiment (Fig. 2) consists of a room in which a chair can be freely placed. A robot needs to move from the lower corridor to the upper corridor. Recall that we need to choose the chunks such that if the problem is solvable for a particular placement, the problem is also solvable for the chunk that contains this placement. With this in mind, we partitioned the set of placements of the chair in 8 chunks. In workspace, each chunk overlaps with its neighboring chunks as shown in Fig. 2b.

The second scene (shown in Fig. 4) consists of 4 moving obstacles that can all be placed in two different positions, hence the number of chunks is 8. In contrast to the first experiment, the connection sets consist of composite

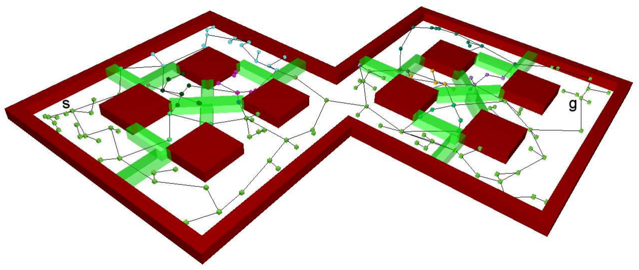


Fig. 5. A more complicated puzzle environment, consisting of 256 potential combinations of the moving obstacle placements.

chunks. There are 16 combinations of moving obstacle placements (2^4), and thus we have 16 composite chunks. The robot needs to move from one corner to another.

The third experiment (Fig. 5) is a variation of the second, but since the scene is repeated twice, the number of chunks is 16 and the number of composite chunks is $2^8 = 256$. Again the query points are chosen far apart.

For each scene we conducted 25 experiments for which the running times were averaged. We ran the algorithm until a roadmap was created that contains a path between the query points for all potential obstacle placements. In these experiments this is a good criterion because the query points are far apart and the robot has to pass many moving obstacles on its path. For all experiments we used a cylinder as the robot, representing a moving body.

To collision-check a node or an edge, we always first check the robot against the static obstacles. If there is a collision, we reject the edge or node immediately. For collision-checking an edge we used standard binary interpolation [5]. Nodes are only connected to other nodes that are at most half the size of the scene away. We implemented the described algorithms including all suggested optimizations, except that we did not use a limited depth on the propagation. Experiments were conducted on a Pentium 3.0GHz equipped with 1Gb of memory.

The results of our experiments are shown in Table I.

TABLE I

ROADMAP CREATION TIMES FOR THE THREE EXPERIMENTS

Scene	Average running time (s)
Office with chair	0.23
Puzzle scene	0.19
Extended puzzle scene	1.94

As can be seen from the results, in the first two experiments, preprocessing the scene to create a robust roadmap took only very little time. In the third scene (where there are 16 times more composite chunks compared to the second scene), preprocessing took a little longer but still less than 2 seconds. After preprocessing, queries can be performed instantaneously.

VII. CONCLUSION

In this paper a novel approach for creating robust roadmaps in changing environments was presented. We proved that our method is probabilistically complete; if the set of potential placements of the moving obstacles is known beforehand, our algorithm creates a roadmap that eventually contains a solution path for any feasible query.

Our method is a generalization of standard PRM, i.e. if there are no moving obstacles, our algorithm behaves equivalent to the PRM method. The algorithm can be used in combination with any sampling strategy and local planner. In addition it is applicable to a broad range of robot types in configuration spaces of any dimension. The method can even be extended to the case of non-holonomic robots that require the use of *directed* roadmaps.

The results we obtained with our implementation of the method show that the approach performs well within run-time demands for realistic environments. A great advantage of our method is that we leave the lion's share of the effort to the preprocessing phase. Since we do not require any additional collision checks in the query phase, interactive performance can be achieved. This property is also exploited to replan if the obstacles change their placement during query time. In addition, several heuristics to improve preprocessing time were proposed. These provide a trade-off between efficiency and completeness.

ACKNOWLEDGMENT

The authors like to thank Marin Hekman for constructing the scenes used in this paper.

REFERENCES

[1] N. Amato, Y. Wu; A randomized roadmap method for path and manipulation planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 113-120, 1996.
[2] J. Barraquand, L. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, P. Raghavan; A random sampling scheme for path planning. *Int. Journal of Robotics Research* 16, pp. 759-774, 1997.
[3] J. P. van den Berg, M. H. Overmars; Roadmap-based Motion Planning in Dynamic Environments. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1598-1605, 2004.
[4] G. van den Bergen; *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann Publishers, San Francisco, 2004.
[5] R. Geraerts, M. H. Overmars; A comparative study of probabilistic roadmap planners. *Proc. Workshop on the Algorithmic Foundations of Robotics*, pp. 40-54, 2002.
[6] L. Jaillet, Th. Siméon; A PRM-based Motion Planner for Dynamically Changing Environments. *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1606-1611, 2004.
[7] L. Kavraki, P. Švestka, J.-C. Latombe, M. H. Overmars; Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces. *IEEE Trans. on Robotics and Automation 12*, pp. 566-580, 1996.
[8] J. J. Kuffner, S. M. LaValle; RRT-connect: An efficient approach to single-query path planning. *Proc. IEEE International Conference on Robotics and Automation*, pp. 995-1001, 2000.
[9] J.-C. Latombe; *Robot motion planning*. Kluwer Academic Publishers, Boston, 1991.
[10] P. Leven, S. Hutchinson; A Framework for Real-time Path Planning in Changing Environments. *Int. Journal Robotics Research*, 21(12):999-1030, 2002.
[11] M. H. Overmars; A random approach to motion planning. *Technical Report RUU-CS-92-32, Dept. Comput. Sci., Utrecht Univ.*, Utrecht, the Netherlands, 1992.
[12] P. Švestka; Robot motion planning using probabilistic roadmaps. *PhD thesis, Utrecht Univ.*, 1997.