Path Planning in Repetitive Environments

Jur van den Berg Mark Overmars
Department of Information and Computing Sciences
Universiteit Utrecht, The Netherlands
{berg, markov}@cs.uu.nl

***Abstract*— In this paper we discuss path planning in a special class of dynamic environments, namely *repetitive environments*, in which the motions of the moving obstacles are periodic (i.e. repetitive). For such environments it is possible to generate a roadmap in a preprocessing stage, which can be used multiple times to quickly solve individual planning queries. This in contrast to general dynamic environments, whose *transitoriness* invalidates the premise that a preprocessed roadmap can be used multiple times. Our approach is suitable for any robot with any number of degrees of freedom in two- and three dimensional workspaces, and applications can be found, amongst others, in virtual environments and computer games.**

I. Introduction

Path planning is of great importance in various fields such as robotics and virtual environments. The problem is generally formulated as finding a path for a robot between two query configurations that avoids collisions with the obstacles in the environment. Most research has focused on the problem of path planning in *static* environments, where the geometry of the environment is known in advance and the obstacles do not move. Many solutions to this problem have been presented, most of them being *probabilistic* [8], [9]. They can roughly be subdivided into two classes: single-shot and multiple-shot. Single-shot approaches aim to solve a single planning query as fast as possible. The most popular method is the Rapidly-exploring Random Tree (RRT)-approach [7], where a random tree of possible paths is grown outward from the start until it can be connected to the goal configuration. Multiple-shot approaches do some (relatively expensive) preprocessing on the environment, such that (multiple) individual path planning queries can be answered quickly. They are useful when expectedly many queries will be done in the same environment. The most well-known approach is the Probabilistic Roadmap (PRM)-method [6], [12]. In the preprocessing stage, it generates a roadmap that represents the connectivity of the free space by connecting randomly sampled configurations with straight-line paths. Individual queries are then answered by connecting the start and goal to the roadmap, and searching for a shortest path (using e.g. Dijkstra's). Both PRM and RRT are proven to be *probabilistically complete*, and have successfully been used in complex, high-dimensional configuration spaces.

The extension of the path planning problem to dynamic environments, where some of the obstacles are moving, has been extensively studied as well [3], [5], [11], [1]. The static-environment approaches can be applied to dynamic environments when the absolute notion of time is incorporated in the configuration space as an additional dimension.

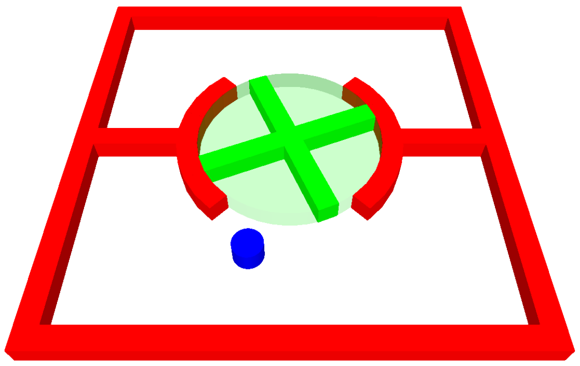

Fig. 1. A repetitive environment with an automatically revolving door.

Moving obstacles in the workspace then transform to static obstacles in the configuration-time space. However, the nature of configuration-time spaces differs considerably from the nature of regular configuration spaces (without time). Firstly, configuration-time spaces are *transitory*, as the notion of time in these spaces corresponds to the real-world time. Because of this, multiple-shot approaches are *not* applicable here: it is possible to construct a roadmap, but it can only be used once – for a query of which the real-world time corresponds to the time interval modeled in the configuration-time space. Therefore, single-shot approaches have been the methods of choice in these spaces [5], [11].

A second problem of the inherence of the real-world time and the time modeled in the configuration-time space, is that planning a path takes time itself, and must be finished before the path is executed. So, when a path planning query is formulated in a dynamic environment, one must attach a time value to the start configuration. This value should not be chosen too close to the current real-world time, because that would leave too little time for planning the path. On the other hand, the start time should not be chosen too far from the current real-world time, because this would result in a latency that is unacceptable for most applications. Ideally, the start time is chosen equal to the current real-world time plus the time needed for planning, where the planning time is preferably as short as possible. However, an RRT-approach (like in [5]) does not give any guarantee on the amount of time needed to plan a path, which makes it hard to pick a suitable start time. Also, the planning may take quite long, particularly in cluttered spaces with many narrow passages [1].

The above problems can be overcome if the moving obstacles have *periodic motions*. That is, after some time π the obstacles return to their initial configuration and execute the same motion again. Such *repetitive environments* do not have the problem of transitoriness, so we *can* apply a multiple-shot approach and build a roadmap in a preprocessing stage. We exploit this roadmap to tackle the second problem, as finding a path in the query stage can be done very fast and within an accurately estimated amount of time.

Repetitive environments are often observed in practice, particularly in virtual environments and computer games. Examples are automated revolving doors (see Fig. 1), automated sliding doors, regular bus services, etc. Environments in which a subset of the free space has a periodic motion also fall within the definition, such as automated platforms, elevators, etc. Yet, we are unaware of any approaches to date dealing specifically with this class of dynamic environments.

In this paper we present a framework for path planning in repetitive environments. Our method is suitable for any robot with any number of degrees of freedom, both in two- and three dimensional workspaces. Once a roadmap has been constructed, queries can be answered within real-time requirements. The rest of the paper is organized as follows. In Section II we will describe a naive approach to the problem. Section III describes a more advanced approach which tackles the problems raised by the naive approach, and in Section IV we will discuss experiments performed with our method proving the claims we make in the paper.

II. Naive Approach

A. Problem Specification

The static path planning problem is generally formulated in terms of the configuration space $\mathcal{C}$, the set of all possible configurations of the robot. The dimension of the configuration space corresponds to the number of the robot's degrees of freedom. To extend the definition to motion planning in dynamic environments, time is added as a dimension. Let $\mathcal{T} \subset \mathbb{R}$ be the time interval of interest, then the *configuration-time space* $\mathcal{CT}$ is formed as $\mathcal{C} \times \mathcal{T}$. It consists of pairs $\langle c, t \rangle$, where c is an element of $\mathcal{C}$ denoting the robot's configuration, and t a scalar in $\mathcal{T}$ denoting the time. The robot is represented by a point in the configuration-time space, and both static and dynamic obstacles in the workspace transform to static obstacles in $\mathcal{CT}$.

For repetitive environments, let π be the period of the motions of the moving obstacles. Then the time interval $\mathcal{T}$ is defined as $[0, \pi)$. Further, it is combined with the modulo-π operator, making configuration-time $\langle c, \pi \rangle$ equal $\langle c, 0 \rangle$, or in general: $\langle c, t \rangle = \langle c, t \bmod \pi \rangle$. Hence, the space is periodically connected through the boundaries $t = \pi$ and $t = 0$.

Given a start configuration $c_s \in \mathcal{C}$ and a start time $t_s \in \mathbb{R}$, and a goal configuration $c_g \in \mathcal{C}$, the problem to solve is to find a valid path $\Pi : [t_s, t_g] \to \mathcal{C}$, such that $\Pi(t_s) = c_s$, and $\Pi(t_g) = c_g$, for some arrival time $t_g \in \mathbb{R}$. Hence, we must find a collision-free path in the configuration-time space from $\langle c_s, t_s \bmod \pi \rangle$ to $\langle c_g, t_g \bmod \pi \rangle$. However, this path should obey some additional constraints posed by the time dimension. Firstly, time marches forward, so all paths should be monotonic in the time-dimension. Further, we would like the paths satisfy some maximum velocity $\hat{v}$ of the robot. So, the slope of valid paths in the configuration-time space is bounded: $\forall (t_1, t_2 \in [t_s, t_g] : t_2 > t_1 : d_{\mathcal{C}}(\Pi(t_1), \Pi(t_2)) \leq (t_2 - t_1)\hat{v})$, where $d_{\mathcal{C}} : \mathcal{C}^2 \to \mathbb{R}$ is the distance measure on the configuration space $\mathcal{C}$.

Given the above constraints, we can define a distance measure $d : \mathcal{CT}^2 \to \mathbb{R}$ on the configuration-time space $\mathcal{CT}$:

$$d(\langle c_1, t_1 \rangle, \langle c_2, t_2 \rangle) = \left\lceil \frac{t_1 + d_{\mathcal{C}}(c_1, c_2)/\hat{v} - t_2}{\pi} \right\rceil \pi + t_2 - t_1.$$

It measures the *time* needed to go from one state-time to the other without violating the maximum velocity constraint. Note that since paths in $\mathcal{CT}$ are monotonic in the time dimension, this distance measure is *asymmetric*.

B. PRM *for Static Environments*

A PRM-approach aims to create a roadmap that represents the connectivity of the free configuration space, so that it can be used effectively for path planning. The roadmap is constructed by sampling configurations randomly from the configuration space. If a configuration is collision-free (determined by a collision-checker [2]), it is added as a node to the roadmap. Subsequently, it is attempted to connect the node to other nodes already present in the roadmap. If such a connection succeeds –that is when the straight-line segment in the configuration space between the two nodes is collision-free– it is added as an (undirected) edge to the roadmap. This procedure is repeated until some stop-criterion is met, usually that is when some predefined set of configurations is connected via the roadmap.

Since collision-checking is expensive, connections are only attempted to a specific set of *potential neighbors* for which there is a high probability that a connection succeeds. Usually, this set is formed by the nodes that are closer than some preset maximum neighbor distance $\hat{d}$. Further, connections to nodes in the same *connected component* of the roadmap are omitted. This is because they would not extend the connectivity of the roadmap. To fully exploit the latter rule, the neighboring nodes are considered in the order of increasing distance to the newly added node.

C. PRM *for Periodic Configuration-Time Spaces*

To implement a PRM for periodic configuration-time spaces, we can fairly straightforwardly follow the PRM-approach for static environments. We start with sampling configuration-times from $\mathcal{CT}$, i.e. a configuration c is randomly picked from $\mathcal{C}$ and a moment in time t is chosen randomly out of the interval $[0, \pi)$ to form a random configuration-time $\langle c, t \rangle$. If this configuration-time is collision-free, i.e. the robot configured at c at time t does not collide with any static or dynamic obstacle, it is added as a node to the roadmap.

Since the distance measure on $\mathcal{CT}$ is asymmetric, the roadmap must be *directed*. Therefore, a new node now has

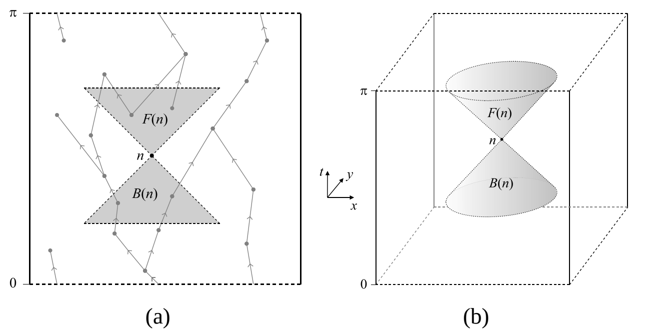

Fig. 2. The neighbor region of a node n in a 2-D (a) and a 3-D (b) $\mathcal{CT}$-space. The horizontal boundaries are induced by the maximum neighbor distance $\hat{d}$ and the diagonal boundaries are induced by the maximum velocity $\hat{v}$.

two sets of potential neighbors: potential forward neighbors (for outgoing edges) and potential backward neighbors (for incoming edges). In both cases we choose a maximum neighbor distance $\hat{d}$. So, if the newly added node is denoted by n, its sets $F(n)$ and $B(n)$ of potential forward and backward neighbors, respectively, are defined as:

$$\begin{aligned} F(n) &= \{n' \in N - \{n\} \mid d(n, n') \leq \hat{d}\}, \\ B(n) &= \{n' \in N - \{n\} \mid d(n', n) \leq \hat{d}\}, \end{aligned}$$

where N denotes the set of nodes in the roadmap. The region in the configuration-time space containing the potential neighbors is hourglass-shaped (see Fig. 2).

Having determined the set of potential neighbors of a newly added node n, we try to connect to them in the order of *increasing* distance. Just as in the static case, we like to avoid redundant connections to nodes to which a connection already exists. However, in directed roadmaps there is no such thing as connected components. Therefore, we maintain for the newly added node n two auxiliary sets, FC and BC, containing the nodes to which n is *transitively forward connected* and to which it is *transitively backward connected*, respectively. Initially these sets are empty, but each time an outgoing edge from n to another node n' is added to the roadmap, we add n' to the set of forward connected nodes FC, and do this recursively for the outgoing edges of n'. The recursion stops when a node is already in FC, or when the node is not contained in the potential forward neighbor set $F(n)$. Now, each time we consider an edge (n, n') for addition to the roadmap, we first check whether n' is already in FC. If so, we can omit the edge. For backward neighbors we proceed similarly. When all the potential neighbors have been considered, a new node is sampled and the above procedure is repeated. The complete algorithm is given in Algorithm 1.

We consider the nodes in the potential neighbor sets in the order of increasing distance to n. This may look trivial, but actually we can only do this because of the property that all edges are oriented in the same direction along the time axis. For general directed roadmaps the definition of potential neighbor sets is much more complicated (as is the evaluation whether or not a node is a potential neighbor), and the nodes

Algorithm 1 ConstructPeriodicPrmNaive

```
1:  repeat
2:    Sample a configuration-time $\langle c, t \rangle \in \mathcal{CT}$
3:    if configuration-time $\langle c, t \rangle$ is collision-free then
4:      Add $\langle c, t \rangle$ as a node $n$ to the roadmap
5:      $FC \leftarrow \emptyset$; $BC \leftarrow \emptyset$
6:      for all $n' \in F(n)$ in order of increasing $d(n, n')$ do
7:        if $n' \notin FC$ and the straight-line between $n$ and
          $n'$ in $\mathcal{CT}$ is collision-free then
8:          Add edge $(n, n')$ to the roadmap
9:          Update $FC$
10:     for all $n' \in B(n)$ in order of increasing $d(n', n)$ do
11:       if $n' \notin BC$ and the straight-line between $n'$ and
          $n$ in $\mathcal{CT}$ is collision-free then
12:         Add edge $(n', n)$ to the roadmap
13:         Update $BC$
14: until some stop-criterion is met
```

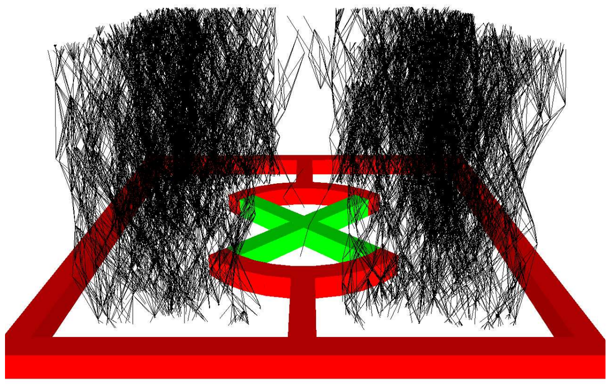

Fig. 3. An example roadmap in the configuration-time space of the revolving door scene of Fig. 1. The third dimension is the time axis.

in the potential neighbor set are considered in the order of *decreasing* distance [12].

We applied the above algorithm on the scene of Fig. 1, where the period π covers one quarter-revolution of the door. The construction of the roadmap stopped when two specific configurations on either side of the door were mutually forward connected. To this end, we maintained the forward connected set of these two nodes during the entire algorithm. The maximum neighbor distance $\hat{d}$ was set to $\pi/4$, and the maximum velocity $\hat{v}$ of the robot was set in accordance with the angular velocity of the door.

Averaged over 100 runs, it took 1.52 seconds to create a roadmap for this scene on a Pentium IV 3.0GHz with 1 GByte of memory. The average roadmap contained 2276 nodes (see Fig. 3 for an example).

D. Query Stage

A roadmap created as shown above can be used to quickly answer queries for a path in the environment. Suppose we want to find a path from some configuration c_s to some configuration c_g, and suppose that the query is posed at real-world time

t_w. Then, we should find a path from $\langle c_s, (t_w + \Delta t) \bmod \pi \rangle$ to $\langle c_g, t_g \bmod \pi \rangle$ in $\mathcal{CT}$, where t_g is some preferred arrival time. Δt is preferably an as small as possible amount of time in which we should (1) connect both query configuration-times to the roadmap as if they were newly added nodes, (2) find a path in the roadmap from the start to the goal, for instance using Dijkstra's algorithm, and (3) do some optional post-processing on the path, like smoothing. The value of Δt should be chosen before step 1 is carried out, and the path can only be executed after step 2 is finished. Hence, we must have an accurate estimate of how long these steps take to choose a good value for Δt.

Experiments on the above scene indicate that connecting the query configurations takes on average 0.006 seconds and performing a shortest path query (using Dijkstra's in our case) takes 0.003 seconds. So, within only 0.01 seconds we can successfully answer a query. Step 3, smoothing the path, is not necessary for answering the query, but it makes the path look nicer and more natural. An appealing property of most smoothing algorithms is that they are incremental [4], so we can spend as much time as we like on smoothing the path. So if, for instance, we choose Δt to be 0.1 seconds, we have enough time to perform the necessary steps 1 and 2, and can spend the remaining time on step 3.

III. Advanced Approach

Although we showed some promising results in the previous section, the method described so far has some clear drawbacks:

- Large parts of the environment are usually time-invariant. Yet, samples that are picked from these regions include a time-value, while they could have been treated as samples in a static environment. This could substantially lower the number of nodes in the roadmap (see e.g. Fig. 3).
- If there is more than one periodic obstacle, the overall period π is calculated as the *least common multiple* of the periods of the individual obstacles. This may cause π, and thus the number of nodes necessary to adequately cover the $\mathcal{CT}$-space, to become very large.
- If a query is posed on the repetitive environment, one has to attach some time-value to the goal configuration (t_g in the previous section). Usually we do not want this, but rather reach the goal as soon as possible.

In this section we propose a more advanced approach in which we resolve the above issues. The main idea of the advanced approach is that local periodicities are considered instead of one global period.

A. Advanced PRM for Repetitive Environments

In the advanced method we define each configuration $c \in \mathcal{C}$ to have its own *local period* $\pi(c)$. If the configuration c lies in a (time-invariant) part of the space where it cannot possibly be intersected by a periodic obstacle, we define $\pi(c)$ to be zero. If c does possibly intersect a periodic obstacle, the local period $\pi(c)$ of the configuration equals the period of this obstacle. It may also be possible that a configuration intersects more than one obstacle, but for now we assume that this will not happen.

To determine the local period of a configuration c, we check of which periodic obstacles the *sweep volumes* in workspace intersect with the robot configured at c. To this end, we assume that the sweep volumes of the periodic obstacles are modeled as objects in the collision-checker (see for instance Fig. 1; the sweep volume is indicated as a transparent disc), along with information about the period of the obstacle.

Sampling is done as follows. First, a random configuration $c \in \mathcal{C}$ is picked. Then we determine the local period $\pi(c)$ of c. If $\pi(c)$ is zero, we can treat the sample as a node in a static environment and need not attach a time value to it. Otherwise, we should randomly pick a time-value for the sample from the range $[0, \pi(c))$. However, this yields relatively little samples in time-variant parts of the space. Therefore, we let the number of time-values m that are attached to a configuration c be a function of the local period: $m(c) = \max(1, \lceil \nu\pi(c) \rceil)$, where ν is a parameter controlling the ratio between the number of samples in time-variant and time-invariant parts of the space.

As there are different local periods, the distance measure becomes somewhat more complicated:

If $\pi(c_1) = 0 \vee \pi(c_2) = 0$, then:

$$d(\langle c_1, t_1 \rangle, \langle c_2, t_2 \rangle) = d_{\mathcal{C}}(c_1, c_2)/\hat{v}$$

else if $\pi(c_1) = \pi(c_2)$ then:

$$d(\langle c_1, t_1 \rangle, \langle c_2, t_2 \rangle) = \left\lceil \frac{t_1 + d_{\mathcal{C}}(c_1, c_2)/\hat{v} - t_2}{\pi(c_2)} \right\rceil \pi(c_2) + t_2 - t_1$$

else the distance is undefined.

The above distance measure implies that we can connect two nodes when (1) they are both in a time-invariant part of the space, (2) one node is time-variant and the other is not, or (3) they are both time-variant but have the same local period. Nodes that are both time-variant but have different local periods cannot be connected, but this is not a problem as indirect connections are possible via time-invariant parts of the space. In general, we do not allow edges that cross more than one border between regions of the space with different local periods. So, for example an edge between two nodes in the time-invariant part of the space that goes through a time-variant part is not admitted.

The definitions of the potential neighbor sets $F(n)$ and $B(n)$ remain the same under the new distance measure. However, the three types of edges identified above differ in nature, affecting the way in which we can discard edges for which a connection already exists. For instance, if we consider an edge between two time-variant nodes (with local period π_ℓ) for addition to the roadmap, we know that if some other path through the roadmap (with length $< \pi_\ell$) already exists between these nodes, the amount of time it takes to travel from one node to the other is not decreased when we add the new edge. For an edge between two time-invariant nodes on the other hand, this is not the case, as such edges are always traversed at maximal velocity and the nodes do not have a time-value. Hence, a direct connection between two nodes in time-invariant parts of the space always shortcuts possible existing paths in the roadmap (in terms of time). Yet, we do not want all of these edges to be added, for a slightly longer detour may also be acceptable. The same holds (more or less) for edges connecting one time-variant node and one time-invariant node. (Note that such edges can only be traversed at times in accordance with the time-value attached to the time-variant node of the edge.)

Therefore, we adopt a variant of the method of [10] for deciding whether or not the edge is added in these cases: An edge is added between two nodes *unless* there already exists a path in the roadmap that is shorter than k times the length of the direct connection between the two nodes, for some parameter $k \geq 1$. To this end, we maintain for a newly added node a *partial* shortest path tree up to some distance horizon. This horizon equals k times the length of the edge that is considered for addition. Since we consider possible edges in order of increasing length, the shortest path tree is incrementally grown each time a new edge is considered.

The complete advanced algorithm is given in Algorithm 2. The function PARTIALINCRDIJKSTRA(n, h) grows the current shortest path tree of n up to horizon h. The graph distance between two nodes n and n' is denoted $G(n, n')$. It is defined to be ∞ if n' is not in the partial shortest path tree of n.

Algorithm 2 CONSTRUCTPERIODICPRMADVANCED

```
1:  repeat
2:    Sample a configuration $c \in \mathcal{C}$
3:    for $\max(1, \lceil \nu\pi(c) \rceil)$ times do
4:      Sample a time $t \in [0, \pi(c))$.
5:      if configuration-time $\langle c, t \rangle$ is collision-free then
6:        Add $\langle c, t \rangle$ as node $n$ to the roadmap
7:        for all $n' \in F(n)$ in order of incr. $d(n, n')$ do
8:          PARTIALINCRDIJKSTRA$(n, k \cdot d(n, n'))$
9:          if $k \cdot d(n, n') < G(n, n')$ and the edge $(n, n')$
            is collision-free in $\mathcal{CT}$ then
10:           Add edge $(n, n')$ to the roadmap
11:       Do the same for backward neighbors in $B(n)$
12: until some stop-criterion is met
```

B. Query Stage

Suppose we want to find a path from a configuration c_s to a configuration c_g in a repetitive environment using the advanced method. If c_g is time-invariant, i.e. $\pi(c_g) = 0$, we do not need to attach a time value to c_g, and thus do not need to predetermine some 'preferred arrival time' like in the previous section. Also, if c_s is time-invariant as well, we can connect both query configurations to the roadmap *before* determining the start time of the query. Let t_w be the real-world time after we have connected the query configurations to the roadmap, then the start time t_0 of the query will equal $t_w + \Delta t$, for an as small as possible Δt. The only thing we have to do in this time lapse Δt is finding a path in the roadmap starting from c_s at time t_0 and arriving at c_g as quickly as possible (and optionally some path smoothing afterwards).

For this, we slightly adapt Dijkstra's algorithm, as we should be careful if we encounter edges from a time-invariant node

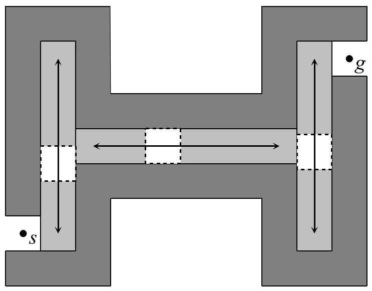

Fig. 4. A repetitive environment with three sliders. The white squares are free space that slide back and forth along the arrows.

n to a time-variant node $n' = \langle c', t' \rangle$. Such an edge is only valid if we leave n at times t_ℓ for which: $t_\ell \bmod \pi(c') = (t' - d(n, n')) \bmod \pi(c')$. Hence, if we arrive at n at time t, we must wait at n for $(t' - d(n, n') - t) \bmod \pi(c')$ time, before we can proceed to n'.

C. Overlapping Dynamic Obstacles

In the above advanced method we have not dealt with cases in which the sweep volumes of the repetitive obstacles overlap in the *configuration space*. That is, at some configurations the robot intersects more than one sweep volume in *workspace*. Let us first discuss the case that a *pair* of obstacles intersect. To create a roadmap in such environments, we should treat these two obstacles as one obstacle with a period equal to the *least common multiple* of the periods of the individual obstacles. Having defined two intersecting obstacles as one obstacle, we can thus handle the case in which three or more obstacles (either directly or indirectly) intersect as well. Eventually, each *connected component* of intersecting obstacles is treated as one obstacle with a period equal to the least common multiple of the periods of the constituting obstacles.

To show that we cannot do any better than this, consider Fig. 4. Suppose that we want to find a path from s to g through the series of sliders (the white squares are free space that slide back and forth along the arrows). Obviously, if the robot has area, the sweep volumes of the three obstacles (indirectly) intersect. Now, if each slider has a different period, and the motions of the sliders are out of phase, the robot has to stay in the horizontal slider for a couple of periods (i.e. moving back and forth multiple times) until it can switch to the right vertical slider. Even though the periods of the individual sliders may be small, the combined period of the sliders can be large, and the path that is found through these sliders can indeed be of the complexity of this combined period.

IV. EXPERIMENTS

We implemented both the naive and the advanced method for creating a roadmap in a repetitive environment. The advanced method has two parameters, ν and k, which the naive method does not have. Note that if we define the local period $\pi(c)$ to equal the global period π for all configurations c, and we set $\nu = 0$ and $k = 1$, the advanced method has become equivalent to the naive approach.

TABLE I

RESULTS FOR VARIOUS PERIODS OF THE REVOLVING DOORS (SEE FIG. 5)

	Periods		Naive approach					Advanced approach				
	local	**global**	**#vertices**	**#edges**	**roadmap**	**add query**	**dijkstra**	**#vertices**	**#edges**	**roadmap**	**add query**	**dijkstra**
1	1, 1	1	665	9079	4.84s	0.017s	0.001s	711	2193	1.57s	0.031s	0.001s
2	5, 5	5	4010	59058	28.23s	0.031s	0.010s	1499	6021	7.07s	0.053s	0.002s
3	2, 2.5	10	3198	30403	16.66s	0.018s	0.005s	897	3110	2.55s	0.034s	0.001s
4	2.5, 4	20	6975	69186	39.27s	0.037s	0.012s	1099	4073	3.91s	0.042s	0.001s
5	1.3, 3.5	45.5	11404	95235	64.02s	0.059s	0.016s	899	3230	2.90s	0.038s	0.001s
6	2.9, 3.7	107.3	22177	167791	140.44s	0.135s	0.032s	1146	4237	4.03s	0.041s	0.001s

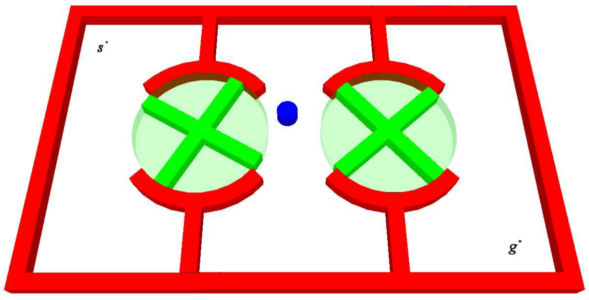

Fig. 5. A repetitive environment with two revolving doors.

In this section we compare the performance of both approaches on the environment of Fig. 5 for various periods of the revolving doors. That is, we measure the time needed for creating the roadmap, the time needed to attach the query configurations to the roadmap, and the time needed to perform Dijkstra's shortest path algorithm. Since randomness in involved in creating a roadmap, these figures are averaged over 100 runs. Parameters such as maximal velocity $\hat{v}$ of the robot, and maximal neighbor distance $\hat{d}$ are set equal for both methods. Throughout the experiments, we fix $\nu = 1$ and $k = 3$ for the advanced method. In all cases, the stopping criteria is that both query configurations are mutually forward connected. The experiments were performed on a Pentium IV 3GHz with 1 GByte of memory. The results are summarized in Table I. The first column gives the periods of both doors and the global period, i.e. the least common multiple of both.

The results clearly show that the time it takes for the naive approach to generate a roadmap grows more or less proportional with the global period of the environment. This relation is not strict however, as is indicated by the second experiment. Although the global period is smaller than in the third experiment, it still takes more time to generate a roadmap. This is because the individual periods of the doors are larger in the second experiment, giving less opportunities to pass the revolving doors. The running times of the advanced approach are not affected by the global period. Only the local periods of the doors play a role, which is indicated by experiments 2 and 4, which feature the largest local periods and hence require the most time for a roadmap to be generated.

Adding the query configurations to the roadmap takes more time (relative to the roadmap size) in the advanced method than in the naive approach, because the technique used to determine whether or not an edge should be added between two nodes is more expensive in this case. However, for the advanced method this step is not part of the planning time that needs to be pre-estimated. Only the Dijkstra-step is, and because the roadmaps stay small in the advanced method, this can be performed within insignificant amounts of time. Also, the amount of time needed can be estimated accurately; it scales more or less linearly with the number of edges in the roadmap (see the results of the naive approach). We can conclude that after a query is posed, the path can be executed with negligible latency, well within real-time constraints.

V. CONCLUSION

In this paper we presented a probabilistically complete approach for path planning in repetitive environments. We showed that these environments need a different treatment than general dynamic environments, as they are not transitory. This allows for a multiple-shot approach in which a roadmap is created in a preprocessing phase. Experiments presented in this paper prove that, using such a roadmap, path planning queries can be answered within minimal and predictable amounts of time. Hence, our approach can be used effectively in real-time applications, such as computer games or virtual worlds.

REFERENCES

[1] J. van den Berg, M. Overmars; Roadmap-based motion planning in dynamic environments. *IEEE Trans. on Robotics* 21(5), pp. 885-897, 2005.
[2] G. van den Bergen; *Collision detection in interactive 3D environments*. Morgan Kaufmann Publishers, San Francisco, 2004.
[3] K. Fujimura; *Motion planning in dynamic environments*. Springer-Verlag, Tokyo, 1991.
[4] R. Geraerts, M. Overmars; Clearance based path optimization for motion planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 2386-2392, 2004.
[5] D. Hsu, R. Kindel, J.-C. Latombe, S. Rock; Randomized kinodynamic motion planning with moving obstacles. *Int. J. Robotics Research* 21(3), pp. 233-255, 2002.
[6] L. Kavraki, P. Švestka, J.-C. Latombe, M. Overmars; Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. on Robotics and Automation* 12(4), pp. 566-580, 1996.
[7] J. Kuffner, S. LaValle; RRT-Connect: An efficient approach to single-query path planning. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 995-1001, 2000.
[8] J.-C. Latombe; *Robot motion planning*, Kluwer Academic Publishers, Boston, 1991.
[9] S. LaValle; *Planning algorithms*, chapter 5. Cambridge University Press, 2006.
[10] D. Nieuwenhuisen, M. Overmars; Useful cycles in probabilistic roadmap graphs. *Proc. IEEE Int. Conf. on Robotics and Automation*, pp. 446-452, 2004.
[11] S. Petty, T. Fraichard; Safe motion planning in dynamic environments. *Proc. IEEE Int. Conf. on Intelligent Robots and Systems*, pp. 3726-3731, 2005.
[12] P. Švestka; *Robot motion planning using probabilistic road maps*. PhD thesis, Universiteit Utrecht, 1997.