Finding Critical Changes in Dynamic Configuration Spaces

Yanyan Lu Jyh-Ming Lien

Abstract— **Given a motion planning problem in a dynamic but fully known environment, we propose the first roadmap-based method, called critical roadmap, that has the ability to identify and exploit the critical topological changes of the free configuration space. Comparing to the existing methods that either ignore temporal coherence or only repair their roadmaps at fixed times, our method provides not only a more complete representation of the free (configuration-time) space but also provides significant efficiency improvement. Our experimental results show that the critical roadmap method has a higher chance of finding solutions, and it is at least one order of magnitude faster than some well-known planners.**

I. Introduction

In this paper, we consider the problem of navigating an object from an initial configuration to a goal configuration in the workspace consisting of both static and dynamic obstacles. We assume that each dynamic obstacle moves along some known trajectory with bounded velocities. Although this assumption was considered impractical in the past, there are recent evidences showing that considering the motion of the dynamic objects can increase long-term optimality, such as energy efficiency and safety [1]. Recent advances in motion and behavior prediction [2] also provide opportunities for longer planning horizon. In addition, there are more and more examples of mobile factory floor robots, such as KIVA [3], whose motions are fully known. Virtual prototyping, such as assembly/disassembly and part removal, is another important domain that usually considers the motion of the moving parts to be known or predictable [4].

Since the 1980s, there have been extensive work on planning motion in the environment whose state is fully known at any given time. A classical approach is to formulate the problem in configuration-time space (CT-space) by incorporating the time dimension $\mathcal{T}$ to the configuration space $\mathcal{C}$ (C-space), denoted as $\mathcal{X} = \mathcal{C} \times \mathcal{T}$. Earlier planners that work in $\mathcal{X}$ usually do not consider temporal coherence. For example, Erdmann and Lozano-Perez [5] proposed to slice $\mathcal{X}$ into a series of configuration spaces and connect these slices via visibility graphs. Probabilistic methods such as Probabilistic Roadmap Methods (PRM) [6] and Exploring Random Tree (RRT) [7] greatly enhance the ability of the planners to tackle problems with high degrees of freedom. However, the idea of temporal coherence is still largely missing from the direct application of the probabilistic methods by sampling and connecting configurations in $\mathcal{X}$.

This work is supported in part by NSF IIS-096053.

Both authors are with the Department of Computer Science, George Mason University, Fairfax, VA 22030, USA, {ylu4, jmlien}@cs.gmu.edu

There do exist several probabilistic planners that consider temporal coherence, e.g., [8], [9], [10], [11]. The main idea of these planners is to repair the invalid portion of the (tree-based or graph-based) roadmaps due to the motion of the obstacles. Since the changes in the configuration space is usually small from frame to frame, these repairs can usually be done locally and efficiently. However, as far as we known, all these methods repair periodically *at fixed time interval*. That is, even if there are no changes in the configuration space, the planner will still check every node and edge in the roadmap. The situation is even worse when the repair is not done frequently enough. Edges and nodes that are believed to be valid may be unsafe to traverse. Ideally, the repair interval should be determined adaptively based on the motion of the obstacles.

Motivated by this observation, we propose a new approach, called critical roadmap method, that detects all the "critical moments" when the topology of free configuration space $\mathcal{C}_{free}$ changes. At every such a moment, our method updates the roadmap in an efficient way to reflect the current topology of $\mathcal{C}_{free}$. More specifically, our roadmap in $\mathcal{C}$ is composed of several small roadmaps (called local roadmaps) with each constructed for one obstacle [12]. A local roadmap can be reused after it is transformed based on the obstacle's motion. Then, the critical moments of $\mathcal{C}_{free}$ are approximated by determining the status changes of the nodes in local roadmaps.

The new approach has the following nice properties. First, the proposed method provides better efficiency. Compared to PRM and RRT, it takes advantage of temporal coherence and reuses the valid edges and nodes. Compared to the methods which update roadmaps at fixed times, our method performs updating only at critical moments and therefore avoids redundant computation. As shown in the experimental results, the new method is at least one order of magnitude faster than PRM, RRT and a *fixed time interval* method. Second, our method provide better completeness because it focuses on capturing the changes of the topology of $\mathcal{C}_{free}$. Finally, like most probabilistic methods, our method can handle problems with high dimensional $\mathcal{C}$.

II. Related Work

Motion planning problems involving changing environments can be roughly classified into three categories: (1) The trajectory of every moving obstacle is fully known in advance, (2) the future trajectory for a moving obstacle can only be estimated based on the acquired sensor data, and (3) the trajectory of a moving obstacle is completely unpredictable. Because our method falls into the first category, we will focus on reviewing recent works considering fully known environments. More detailed surveys can be found from the references therein.

A common approach to handle moving obstacle with known trajectory is to plan in the configuration-time (CT) space $\mathcal{X}$ [5]. The CT-space $\mathcal{X}$ is usually approximated by a sequence of configuration spaces at fixed times. In [13], visibility-graph algorithm is applied to generate all path segments between adjacent slices and join adjacent solutions. Multi-robot motion planning is an important field in this category. Erdmann and Lozano-Perez introduce prioritized path planning for multiple robots in an environment with static obstacles [5]. It assigns priorities to all the robots and motion planning is performed for one robot at a time in order of decreasing priority. For each robot, its path should be collision-free with respect to the obstacles as well as previously solved robots.

To handle problems in higher dimensional space and kinodynamic constraints, probabilistic methods are usually applied to generate a graph or tree structure to approximate the CT-space. For example, Hsu and Kindel [14] produce a tree-based roadmap in configuration-time space. The tree is rooted at the initial configuration-time point and grows along the time-axis. It terminates when it falls into a region from which it is known how to get to the goal. Because the state of $\mathcal{X}$ usually does not change much during a short period of time, one of the main drawbacks of these planners is that the computation is not reused between slices.

More recent approaches consider temporal coherence or computation reuse by updating the roadmap. Many methods employ two-phase approaches: they first compute a roadmap with respect only to the static obstacles; when given a query during run time, edges are checked based on the location of the dynamic obstacles at given time [8], [15], [9]. For example, Leven and Hutchinson [16] construct a regular grid in workspace that maps each of its cell to the roadmap nodes and edges. Once obstacles move, [16] quickly checks occupied cells and invalidates the associated nodes and edges. Jaillet and Simeon [8] fixe the invalid portion of the roadmap by applying RRT to quickly check the possible reconnections along the invalid edge. If reconnection fails, new nodes and edges are added. van den Berg and Overmars [15] proposed a two-level search strategy. At the local level, they find a feasible local path along a single edge of the roadmap. At the global level, they apply an A*-like search to coordinate the local paths. Li and Shie [10] propose to reuse RRT via a new data structure called Reconfigurable Random Forest (RRF) which combines all the RRTs from previous queries. Ferguson et al. propose the idea of Dynamic RRT [11] that keeps a single RRT and detects the parts of the tree invalidated by configuration space changes. Then these invalid parts are removed and the modified RRT is regrown until the goal is reached again. Our method is closer to the Elastic roadmap [17], where the configurations are sampled around obstacles and are moved along with the obstacles. These methods, either explicitly or implicitly, depend on the idea that the moving obstacles do not significantly affect the roadmap connectivity (at least for a short period of time), thus the planner should be able to quickly repair and replan. However, unlike the proposed method, these methods are not capable of determining the next critical changes in C-space even though the trajectories of the obstacles are fully known, thus can only update the roadmap at fixed times.

III. Preliminaries

In this section, we define important notations used throughout the paper. In our problem, we are given a set of moving obstacles $\mathcal{O}$ and a single robot $\mathcal{R}$ with bounded velocity. The trajectory of each obstacle $O_i \in \mathcal{O}$ is known. The motions of O_i are piecewise linear. During a given time interval, it maintains a constant linear velocity $\vec{v}$ and a constant angular velocity $\vec{\omega}$. In some applications such as prioritized path-planning for multiple robots, O_i can be the robots with priorities higher than $\mathcal{R}$. Our goal is to find a sequence of collision-free configurations so that $\mathcal{R}$ can reach its destination before a user-defined time t_f. We use $\mathcal{R}_c$ to denote the robot configured at configuration c.

Our strategy to solve this problem is to represent the free CT-space $\mathcal{X}_{free}$ by identifying the topological changes in the free C-space $\mathcal{C}_{free}$. Because constructing a complete representation of $\mathcal{C}_{free}$ and $\mathcal{X}_{free}$ is intractable, the data structure that we will be using is a roadmap $\mathcal{M}$. The roadmap $\mathcal{M}$ in $\mathcal{X}_{free}$ is composed a sequences of *critical roadmaps* $\mathcal{G}_t$ constructed at critical times t. Each critical roadmap is then composed of a set of local roadmaps M_i that are constructed at $t = 0$, one for each obstacle O_i, and are transformed along with the motion of O_i at $t > 0$. An example of the local roadmap is shown in Fig. 2.

We say that a time t is a critical moment if the topology of $\mathcal{C}_{free}$ changes at t. The central idea in our strategy is to determine the critical moments E through the given time interval $[0, t_f]$. These critical moments E include (1) the time of collision (toc) and (2) the time of separation (tos). The critical times toc and tos are when a node of the local roadmaps becomes in-collision and collision free, respectively. An iterative method to predict toc and tos will be discussed in Section V.

To estimate toc and tos, our method performs continuous collision detection [18] and penetration depth estimation [19]. Therefore, our geometric model is represented by a convex hull hierarchy in which the root is the convex hull of an entire model and the leaves are its all convex components. Each inner node is the convex hull of its children nodes. We denote the convex hull hierarchy of $\mathcal{R}$ be $CH_{\mathcal{R}}$ and the convex hull hierarchy of O_i be CH_i.

IV. An Overview of Our Method

This section provides an overview of the proposed framework. Algorithm 1 sketches our ideas.

Algorithm 1 CriticalRoadmap($\mathcal{O}$,$\mathcal{R}$)

```
1:  for all $O_i \in \mathcal{O}$ do
2:      $M_i \leftarrow$ ComputeLocalRoadmap($O_i$)
3:      $E_i \leftarrow$ IdentifyCriticalMoments($M_i$)
4:      $E \leftarrow E \bigcup E_i$
5:  $\mathcal{G}_t =$ Connect($M_i$)
6:  while $E \neq \emptyset$ do
7:      $e \leftarrow E.pop()$
8:      $\mathcal{G}_{t'} \leftarrow$ Update($\mathcal{G}_t, e$)
9:      Connect($\mathcal{G}_t, \mathcal{G}_{t'}$)
10:     $\mathcal{G}_t = \mathcal{G}_{t'}$
11: return
```

There are two main stages in Algorithm 1. In the first stage, critical moments are identified using local roadmap M_i for each obstacle $O_i \in \mathcal{O}$. The second stage creates the critical roadmap $\mathcal{G}_t$ at the critical moment t and connects the critical roadmaps to approximate $\mathcal{X}_{free}$. The subroutines that update and connect the critical roadmaps will be discussed in Section VI and Section VII, respectively.

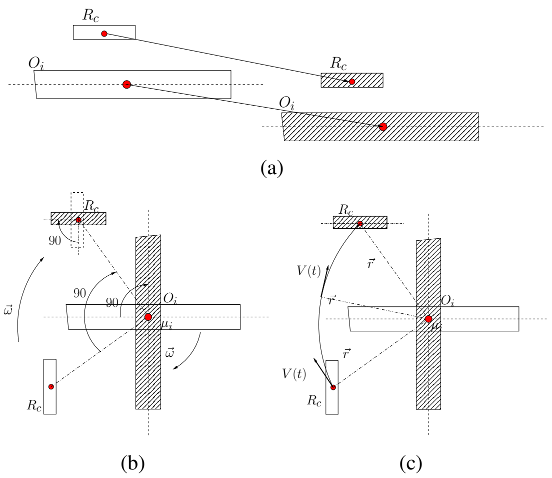

Fig. 1. $\mathcal{R}_c$ and O_i are colored in shadow after transformations. (a). $\mathcal{R}_c$ moves along with O_i when O_i has translational velocity. (b). Since $c \in M_i$, $\mathcal{R}_c$ can be treated as a feature on O_i. While O_i is rotating around μ_i, $\mathcal{R}_c$ is also rotating around μ_i and its orientation is changing. If O_i has angular velocity $\vec{\omega}$, $\mathcal{R}_c$'s angular velocity for rotation around μ_i is $\vec{\omega}$ too. (c). $\mathcal{R}_c$ has the translational velocity $V(t)$ introduced by rotating around μ_i. The instant direction of $V(t)$ is always perpendicular to $\vec{r}$.

Local roadmaps. The local roadmap M_i is constructed via the obstacle-based roadmap methods around the obstacle O_i [20], [21], and M_i always moves along with O_i. Since M_i moves along with its associated obstacle O_i, any $c \in M_i$ also moves along with O_i. That means c can be treated as a point on O_i. In general, $\mathcal{R}_c$ undergoes two types of transformations: translation with O_i, and rotation around μ_i where μ_i is O_i's center of mass. As illustrated in Fig. 1(b), $\mathcal{R}_c$'s angular velocities for self-rotation and rotation around μ_i are both $\vec{\omega}$. Its translational velocity consists of two parts: $\vec{v}$ (see Fig. 1(a)) and $V(t)$. $V(t)$ is introduced by $\mathcal{R}_c$ rotating around μ_i and its magnitude is constant but the direction keeps changing all the time: the direction of $V(t)$ is always perpendicular to $\vec{r} = c - \mu_i$ (see Fig. 1(c)).

Two local roadmaps M_i and M_j are connected via their *boundary nodes* $B_{i,j}$ [12]. For a configuration $c \in M_i$, we say that c is in $B_{i,j}$ if and only if $\mathcal{R}_c$ does not collide with O_j but at least one of c's neighbors in M_i does.

Critical moments. A necessary condition for the critical time t is when at least one of the following two events, i.e., toc and tos, happens at t.

1) $\exists c \in M_i \wedge \exists O_{j\neq i} \in \mathcal{O}$, $\mathcal{R}_c$ and O_j begin to contact.
2) $\exists c \in M_i \wedge \exists O_{j\neq i} \in \mathcal{O}$, $\mathcal{R}_c$ and O_j begin to separate.

Based on the collision states of the local roadmaps, we classify the position relationships of any two $\mathcal{C}$-obstacles into three categories, as shown in Fig. 2. Finally, we know that $\mathcal{C}_{free}$ changes when there are at least one pair of $\mathcal{C}$-obstacles having their position relationship changed. In Section V, we will discuss how to identify toc and tos in details.

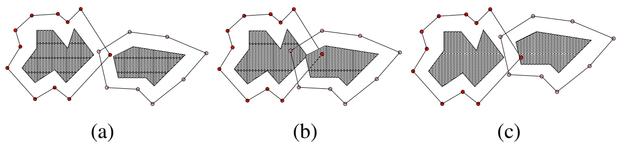

Fig. 2. The position relationships of two obstacles. (a). M_i does not overlap O_j and M_{O_j} does not overlap O_i. (b). M_i overlaps O_j ***and*** M_j overlaps O_i. (c). M_i overlaps O_j ***or*** M_j overlaps O_i.

V. Identify Critical Moments

A. *Time of Contact (toc)*

Given a configuration $c \in M_i$ and an obstacle $O_{j\neq i}$, when $\mathcal{R}_c$ and O_j do not intersect, we want to compute the time of contact toc within a time interval $[t_1, t_2]$.

We extend the iterative method [18] to estimate toc by generalizing the idea of *conservative advancement*. In each iteration, we estimate the advancing time δ_{toc} during which $\mathcal{R}_c$ will be safely moved toward O_j without causing any collision. The estimated advancing time δ_{toc} is calculated based on a tight lower bound of the closest distance between $\mathcal{R}_c$ and O_j and an upper bound of the motions of $\mathcal{R}_c$ and O_j. The process repeats until the distance between $\mathcal{R}_c$ and O_j is under some user-defined threshold ε or t_2 is reached.

1) Estimation of Advancing Time: The result from [18] is only applicable to constant velocities. To estimate δ_{toc}, we extend [18] to consider models that do not have constant velocities. Let d be the shortest distance between $\mathcal{R}_c$ and O_j with direction $\vec{n}$ ($\vec{n}$ is normalized). Note that d is actually the distance between the closest pair of features of $\mathcal{R}_c$ and O_j. Without loss of generality, assume the closest features are two points: p on $\mathcal{R}_c$ and q on O_j (see Fig. 3(a)). Let ρ be an upper bound of motions of $\mathcal{R}_c$ and O_j. Then calculation of ρ is answered by computing the distance traveled by p and q along $\vec{n}$ in unit time.

Let $\mathcal{T}_p$ be p's trajectory and $\mathcal{T}_q$ be q's trajectory. Let R_c and O_j's translational velocity be $\vec{v}$ and $\vec{v'}$ and their angular velocity be $\vec{\omega}$ and $\vec{\omega'}$, respectively. Recall that while $\mathcal{R}_c$ is rotating around μ_i, its orientation is also changing. As a result, each of its surface point has an additional transformation: rotation around $\mu_{\mathcal{R}_c}$ with angular velocity $\vec{\omega}$ where $\mu_{\mathcal{R}_c}$ is $\mathcal{R}_c$'s center of mass. Therefore p's velocity $\dot{\mathcal{T}}_p(t)$ is $\vec{v} + V(t) + \vec{\omega} \times \vec{r_1}$. The projection of p's velocity onto $\vec{n}$ is $\dot{\mathcal{T}}_p(t) \cdot \vec{n}$. q's velocity $\dot{\mathcal{T}}_q(t)$ is $\vec{v'} + \vec{\omega'} \times \vec{r_2}$ and q's velocity along $\vec{n}$ is $\dot{\mathcal{T}}_p(t) \cdot \vec{n}$. Finally, the upper bound of the distance traveled by p and q is given by the following lemma.

Lemma 5.1: The upper bound ρ of the distance traveled by p and q along $\vec{n}$ in unit time is
$|\vec{n} \times \vec{\omega}||\vec{r}| + (\vec{v} - \vec{v'}) \cdot \vec{n} + (|\vec{n} \times \vec{\omega}||\vec{r_1}| + |\vec{n} \times \vec{\omega'}||\vec{r_2}|)$
where

- $|\vec{n} \times \vec{\omega}||\vec{r}|$ is the upper bound of $V(t)$,
- $(\vec{v} - \vec{v'}) \cdot \vec{n}$ is the distance R_c travels towards O_j,
- $\vec{r} = \overrightarrow{pos_p} - \mu_i$, where $\overrightarrow{pos_p}$ stands for the position of the point p,
- $\vec{r_1} = \overrightarrow{pos_p} - \mu_{\mathcal{R}_c}$ is self-rotation radius, and
- $\vec{r_2} = \overrightarrow{pos_q} - \mu_j$, where $\overrightarrow{pos_q}$ stands for the position of point q.

These notations are illustrated in Fig. 3(a). Note that the magnitudes of $\vec{r}$, $\vec{r_1}$, $\vec{r_2}$ are constant but their directions change over time.

Proof: Due to space limit, please see proof in [22]. ∎

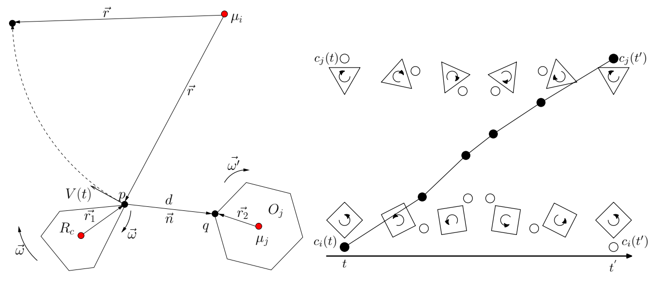

Fig. 3. (a) This figure illustrates the translational velocities of p and q which are caused by rotation. For p, its translational velocity introduced by self-rotation of R_c and rotation of O_i is $\vec{\omega} \times \vec{r} + \vec{\omega} \times \vec{r_1}$. For q, its translational velocity caused by rotation of O_j is $\vec{\omega'} \times \vec{r_2}$. (b) The obstacles are two moving polygons. The square is rotating counterclockwise while the triangle is rotating clockwise. Both of them are translating from left to right. A point robot travels from the bottom-left corner to top-right corner.

B. Time of Separation (tos)

When $\mathcal{R}_c$ and O_j intersect, we apply the similar idea to estimate tos. At each iteration, we first compute the penetration depth pd between $\mathcal{R}_c$ and O_j. Then advancing time δ_{tos} is estimated based on pd and motions of $\mathcal{R}_c$ and O_j. Let $\vec{n}$ be the direction that realizes this pd and ρ be the upper bound of the velocity along $\vec{n}$. Since we know that pd is the shortest distance to separate $\mathcal{R}_c$ and O_j, there is a simple relationship between δ_{tos} and pd:

$$\rho \times \delta_{tos} \leq pd \ .$$

Therefore, we can ensure that $\mathcal{R}_c$ and O_j will remain in collision after advancing them by the estimated time δ_{tos}. This continues until they are still intersecting but the penetration depth is under some very small user-defined tolerance τ or t_2 is reached. Fig 4 shows a simple example.

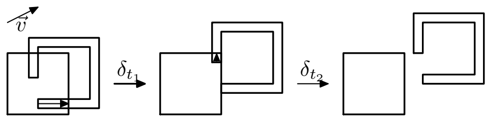

Fig. 4. Assume the square is fixed while the tube is moving with velocity $\vec{v}$. The magnitude of an arrow shows the penetration depth and its direction is the corresponding penetration direction. δ_{t_1} and δ_{t_2} are the estimations of tos at the first and second iteration, respectively.

We apply the idea in Section V-A to compute the upper bound ρ and use DEEP [19] to compute penetration depth between two convex polyhedra. Instead of checking all pairs of convex pieces from $\mathcal{R}_c$ and O_j, one more efficient way is to use a convex hull traversal tree ($CHTT$) [23]. $CHTT$ is built while traversing $CH_{\mathcal{R}}$ and CH_j. The traversal starts with the root nodes of $CH_{\mathcal{R}}$ and CH_j and it is performed recursively on both trees simultaneously. Each node in $CHTT$ corresponds to an intersection test on a single pair of convex pieces. The root node of $CHTT$ is the intersection test on the roots of $CH_{\mathcal{R}}$ and CH_j and its leaf nodes are either the intersection test on the leaf nodes of $CH_{\mathcal{R}}$ and CH_j or a pair of convex pieces which do not intersect. For the intersection test between some pair of nodes in $CH_{\mathcal{R}}$ and CH_j, if they intersect, the penetration depth is computed. Then collision detection will be performed on their child nodes. If they do not overlap, their children are guaranteed to be collision free.

C. Implementation Details

For the computation of toc, conservative advancement continues until the distance d between $\mathcal{R}_c$ and O_j is smaller than a user-defined threshold ε. Because of the underestimation of advancing time in each iteration, the two models are not guaranteed to be in contact and there might be always some very small distance between them. To make the two models collide at time toc, we increase d by ε (we use $\varepsilon = 0.001$) in the last iteration and estimate the advancing time with this new d. Adding a small value to d makes sure that two models are in contact at toc and the reported toc is still very close to its true value. Similarly, for time of separation, δ_{tos} is estimated with $\rho \times \delta_{tos} \leq (pd + \varepsilon)$ in the last iteration.

VI. Update Roadmap at Each Critical Moment

A critical moment implies potential changes of the topology of $\mathcal{C}_{free}$. So for any given critical moment t, the critical roadmap $\mathcal{G}_t$ needs to be updated to reflect these changes. As [12], all local roadmaps are transformed based on obstacles' motions, and then the local roadmaps are merged into $\mathcal{G}_t$ by adding connections between the boundary nodes of pairs of local roadmaps.

More specifically, for each configuration $c \in M_i$ and an obstacle $O_{j\neq i} \in \mathcal{O}$, we maintain a list of critical times $T_{\{c,O_j\}}$. To find the boundary nodes $B_{i,j}$ of M_i, we determine a set of configurations $CD_{i,j}$ from the local roadmap M_i that will make $\mathcal{R}$ collide with O_j. Because for any time t, if t is between a toc and a tos in $T_{\{c,O_j\}}$, $\mathcal{R}_c$ must intersect O_j at t. Therefore, $CD_{i,j}$ be easily collected for any critical time without additional collision detection tests. If $CD_{i,j}$ and $CD_{j,i}$ are nonempty, M_i and M_j are merged by adding connections between their boundary nodes $B_{i,j}$ and $B_{j,i}$.

VII. Connect Critical Roadmaps

Given two consecutive critical times, we need to connect their critical roadmaps. The following observation allows us to make the connections efficiently.

Observation Consider two consecutive critical times t and t' with $t < t'$. For a valid edge c_1c_2 in critical roadmap $\mathcal{G}_t$, let c'_1 be the transformed c_1 at t' and c'_2 be the transformed c_2 at t'. All the connections $c_1c'_1$, $c_1c'_2$, $c_2c'_1$ and $c_2c'_2$ are guaranteed to be collision free if $c'_1c'_2$ is also valid.

Proof: Due to space limit, please see proof in [22]. ■

Note that due to the motion of the local roadmap, the connection between two nodes, e.g., c_1 and c'_2, might not be a straight line. Consider a configuration c from M_i at time t. Let c' be its new position at t' after transformation. Recall that as the obstacle O_i moves, c moves along with O_i. Therefore, the connection between c and c' in $[t, t']$ is exactly the same as O_i's trajectory over this time interval.

Let us now consider a more general example. Let $c_i(t)$ be a node of M_i at time t and $c_j(t')$ be a node of M_j at time t'. The connection (denoted as $\mathcal{T}$) between $c_i(t)$ and $c_j(t')$ over the time interval $[t, t']$ is interpolated as follows. First, we know that $\mathcal{T}(t) = c_i(t)$ and $\mathcal{T}(t') = c_j(t')$. For any other time τ, $t < \tau < t'$, we want to compute $\mathcal{T}(\tau)$. Using linear interpolation, $\mathcal{T}(\tau) = (1 - s) \times c_i(t) + s \times c_j(t + 1)$, where $s = \frac{\tau - t}{t' - t}$. A simple example is shown in Fig. 3(b).

VIII. Experiments and Results

Experiment Set Up. All the experiments are performed on an Intel Core i7 M620 CPU at 2.67GHz with 4GB RAM. The implementation is coded in C++. Our new method is tested on three environments: Hole (Fig. 5(a)), Ball (Fig. 5(b)) and Table (Fig. 5(c)). They share the same robot which is a long and thin rod. The maximum time limits t_f are 5.0, 10.0 and 2.0 time units, respectively. Each environment is assigned 30 queries. Although these queries are generated with some randomness, we try to make each as hard as possible. Take Hole as an example. For each query, the rod robot has to pass through at least one hole to reach its destination.

Table I shows the running times of our method in different phases. All running times are measured in seconds. The numbers of critical times for the Hole, Ball and Table environments are 63, 16, and 7, respectively.

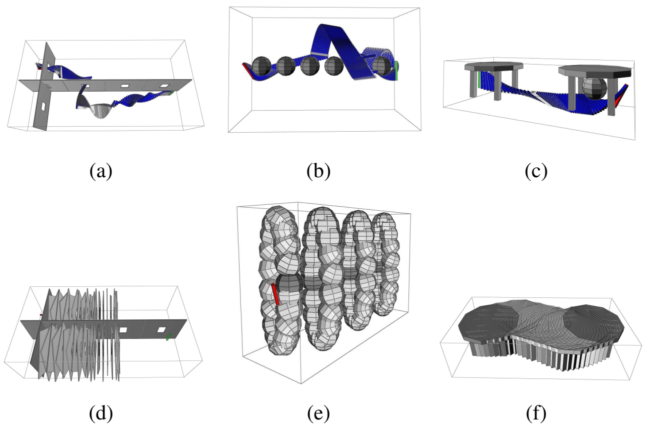

Fig. 5. (a), (b) and (c) show testing environments with a collision-free path connecting start and goal. (d), (e) and (f) show the swept volumes of the dynamic obstacles in (a), (b) and (c), respectively.

TABLE I
RUNNING TIMES (SEC) FOR EVERY PHASE OF OUR METHOD

	Hole	Ball	Table
Create Local Roadmaps	3.182	2.574	2.449
Detecting Critical Moments	4.399	0.375	4.758
Update Global Roadmaps	6.022	0.484	0.031
30 Queries	40.185	7.129	1.669
Total	53.835	10.592	8.939

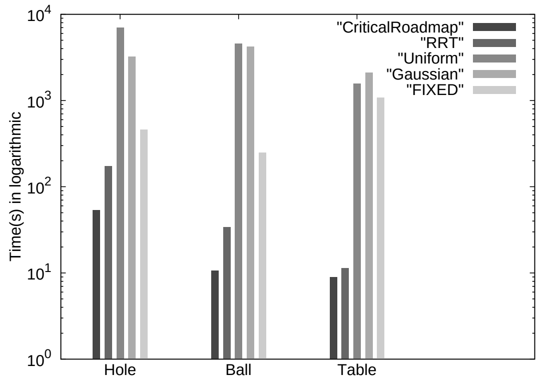

Fig. 6. Experimental results for the environments in Fig. 5. The y-axis is in logarithmic scale.

Comparison to Previous Work. The expected running times for each method are shown in Fig. 6. The expected running time is the averaged running time of all successful queries weighted by the reciprocal of the successful rate. The successful rate is the number of successful queries divided by the total number of queries (which is 30 $\times$ 10 in our experiments). The proposed method can find a valid path for every query, i.e., 100% successful rate.

The proposed method is first compared to several classic planners: Uniform [6], Gaussian [20] and RRT [7]. Like in $\mathcal{C}$, those planners samples a number of $\langle$configuration,time$\rangle$ pairs, keep the valid ones and connect them into a $\mathcal{M}$ in $\mathcal{X}$. As shown in Fig. 6, our method has significant efficiency improvements: 2-3 orders of magnitude faster over Uniform and Gaussian and noticeably faster than RRT. Note that RRT is a single-shot planner and it generates a new tree for each query, so we expect its performance to degrade when more queries are given. All the others are multi-query planners.

As for the method which updates a critical roadmap at fixed time intervals, we set the intervals to 0.02, 0.1 and 0.00001 for the Hole, Ball and Table environments, respectively. The running times for these intervals are plotted in Fig. 6. Although each time interval is very small, it cannot guarantee to find a valid path for each query. For Hole, 5 out of 30 queries cannot be solved and for Table, there is a chance of 18/30 that a query could not be solved. This is because it misses some critical moments when the topology of free configuration space changes. Some connections between two consecutive critical roadmaps are detected invalid at the query stage. Moreover, it may update a roadmap at a moment which does not involve the changes of $\mathcal{C}_{free}$. Our method can avoid these unnecessary update and is at least one order of magnitude faster.

IX. Conclusion and Future Work

In this paper, we presented a method called "critical roadmap method" for planning motion in dynamic but fully known environments. Our method can detect all the critical times when the free configuration space changes. The critical times are estimated via the time of separation and the time of contact. Our method provides a more complete representation of free configuration-time space compared to the existing methods that only use fixed time resolution. We also propose an efficient way to assemble the roadmap at each critical time by reusing the most of the existing roadmap and repair only the invalid part caused by obstacles' motions. Compared to the previous methods, our strategy is more complete and provides significant efficiency improvements.

Limitations and Future Works. While using local roadmaps provides a more efficient way of updating roadmaps for each critical time, it requires the robot to be free flying. Therefore, the new method does not work for a robot with a fixed base. Moreover, our current penetration depth computation relies on decomposing each object into a convex hull hierarchy. This approach does not provide a tight lower bound of the penetration depth between two non-convex shapes therefore increases the number of iterations for conservative advancement. In the future, we will look for a direct way to compute penetration depth between two non-convex objects.

References

[1] T. Fraichard, "A short paper about motion safety," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, pp. 1140–1145.

[2] J. Lee and H. Hashimoto, "Intelligent Space-Its concept and contents," *Advanced Robotics Journal*, vol. 16, no. 4, pp. 265–280, 2002.

[3] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *Spectrum, IEEE*, vol. 45, no. 7, pp. 26 –34, 2008.

[4] M. Garber and M. Lin, "Constraint-based motion planning for virtual prototyping," in *Proceedings of the seventh ACM symposium on Solid modeling and applications*. ACM, 2002, pp. 257–264.

[5] M. Erdmann and T. Lozano-Perez, "On multiple moving objects," *Algorithmica*, vol. 2, pp. 1419–1424, 1986.

[6] L. E. Kavraki, P. Svestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, August 1996.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 1999, pp. 473–479.

[8] L. Jaillet and T. Simeon, "A prm-based motion planner for dynamically changing environments," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004, pp. 1606–1611.

[9] M. Kallman and M. Mataric, "Motion planning using dynamic roadmaps," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 5. IEEE, 2004, pp. 4399–4404.

[10] T.-Y. Li and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2002, pp. 3411–3416.

[11] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1243–1248.

[12] J.-M. Lien and Y. Lu, "Planning motion in similar environments," in *Proceedings of the Robotics: Science and Systems Conference (RSS)*, Seattle, Washington, Jun 2009.

[13] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, October 1979.

[14] D. Hsu, R. Kindel, J. C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," in *Proc. Int. Workshop Alg. Found. Robot.(WAFR)*, 2000, pp. SA1–SA18.

[15] J. van den Berg and M. Overmas, "Roadmap-based motion planning in dynamic environments," in *Proc. IEEE Int. Conf. Intel. Rob. Syst. (IROS)*, 2004, pp. 1598–1605.

[16] P. J. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *Int. Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[17] Y. Yan and O. Brock, "Elastic roadmaps: Globally task-consistent motion for autonomous mobile manipulation in dynamic environments," in *Proc. Robotics: Sci. Sys. (RSS)*, 2006.

[18] X. Zhang, M. Lee, and Y. J. Kim, "Interactive continuous collision detection for non-convex polyhedra," *Vis. Comput.*, vol. 22, no. 9, pp. 749–760, 2006.

[19] Y. J. Kim, M. C. Lin, and D. Manocha, "DEEP: Dual-space expansion for estimating penetration depth between convex polytopes," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2002)*, 2002.

[20] V. Boor, M. H. Overmars, and A. F. van der Stappen, "The Gaussian sampling strategy for probabilistic roadmap planners," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, vol. 2, May 1999, pp. 1018–1023.

[21] J.-M. Lien, "Hybrid motion planning using Minkowski sums," in *Proc. Robotics: Sci. Sys. (RSS)*, Zurich, Switzerland, 2008.

[22] Y. Lu and J.-M. Lien, "Finding critical changes in dynamic configuration spaces," George Mason University, Tech. Rep. GMU-CS-TR-2011-9, 2011.

[23] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha, "Fast proximity queries with swept sphere volumes," in *Proc. of IEEE Int. Conf. on Robotics and Automation*, 2000, pp. 3719–3726.