The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Path Planning in Changing Environments by Using Optimal Path Segment Search

Hong Liu, He Wen, Yan Li

Abstract— **This paper presents a novel planner for manipulators and robots in changing environments. When environments are complicated, it's always difficult to find a completely valid path solution, which is essential for many methods. However, our planner searches for several path segments to make robot move towards its goal as much as possible even though such a complete solution doesn't exist currently. In the learning phase, the planner begins by building a roadmap that captures the topological structure of the configuration space in a workspace without obstacles. In the query phase, the planner searches for a solution path in the roadmap with the A^* algorithm and performs roadmap updating using the lazy evaluation idea concurrently with the solution search process. If a completely valid solution is found, it will be adopted immediately. Otherwise the planner will collect a set of maximum valid path segments and then select the optimal one for planning in the execution process. The searching and execution process will be repeatedly performed until a goal configuration is reached. In plentiful experiments, our planner shows promising performances.**

I. INTRODUCTION

Over the past several decades, path planning in static environments has been studied extensively. The basic problem can be generally stated as to compute a collision-free and feasible motion sequence for an object to move from a given start configuration to a goal one without collision. Exact algorithms[1] which need to describe both the robot and its operation environment exactly, were first produced in the early research. Due to the exponential complexity property, they are infeasible for practical cases. After Lezano-Pérez introduced the concept of configuration space into the context of planning[2], methods building an approximate representation of the configuration space, C-space, caught the focus of research. However, the size of representation increases exponentially with C-space dimension. In order to avoid this drawback, the sampling-based philosophy was introduced to the path planning field. Among those sampling-based approaches, the Probabilistic Roadmap method (PRM)[4] is widely used and has led to many variants. Lazy PRM[5] is one of those variants whose key idea is to perform collision check unless it's necessary and our method also shares its Lazy Evaluation idea. Another category of sampling-based path planning methods utilizes an effective tree-like data structure called Rapidly-exploring Random Tree (RRT), which was first introduced in [6]. Since RRT biases the exploration toward unexplored portions of the space[7], it is an efficient tool for solving single query path planning problems.

Although planning problems in static environments can be solved elegantly, it is very common in such a real world scenario that the environment is not always static. There are a great number of dynamic planning problems involved in industry area and Human-Robot Interaction (HRI)[8] fields. For those purposes, how to extend the path planning to dynamically changing environments had led to extensive research. The Dynamic Roadmap method (DRM)[9][10] tailors the PRM framework to make it adapt to changes occurring to roadmap. It re-validates nodes and edges of roadmap obstructed by obstacles according to a precomputed mapping from workspace (W-space) to C-space. DRM is very effective when the environment at query time does not vary much from the one at precomputation time[11]. Due to the nice property mentioned above of the RRT, many methods based on it, such as Dynamic RRT (DRRT)[12] and Execution-extended RRT (ERRT)[13], have been introduced. Nevertheless, it's always impossible to find a solution with those methods when the environment is cluttered with moveable obstacles. Even though such a solution has been found currently, it gets invalidated soon by the moveable obstacles while planning.

In this paper, we aim to present a new planner tailored for dynamically changing environments. Like other methods, a completely valid path solution will be adopted immediately in the query phase. However, if such a solution doesn't exist currently due to moveable obstacles, our planner will try to collect several valid path segments of which only some nodes and edges are obstructed and then select the optimal one (if there is any) to control the motion of robot in the execution process. If all segments are infeasible, the robot will stop there until a complete solution or feasible segment is found. Meanwhile the roadmap is updated concurrently with the roadmap query process. Since the roadmap maintenance process determines the real-time performance of our planner, several optional update mechanisms will be discussed.

The rest of the paper is organized as follows. Section II describes an overview of some previous works closely related to our approach, plus the motivation of our method. General framework of the planner is presented in Section III and some important details are discussed as well. In Section IV simulation experiments are described. Finally, we draw a conclusion briefly.

H. Liu is with the Key Laboratory of Machine Perception and Intelligence and the Key Laboratory of Integrated Micro-system, Shenzhen Graduate School, Peking University, China hongliu@pku.edu.cn

H. Wen is with the Key Laboratory of Integrated Micro-system, Shenzhen Graduate School, Peking University, Shenzhen 518055 China wenh07119@szcie.pku.edu.cn

Y. Li is with the Key Laboratory of Integrated Micro-system, Shenzhen Graduate School, Peking University, Shenzhen 518055 China liy06040@szcie.pku.edu.cn

978-1-4244-3804-4/09/$25.00 ©2009 IEEE

II. RELATED WORK AND MOTIVATION

PRM is a successful framework which has been widely applied to many practical cases. The main idea behind PRM is to precompute a roadmap which presents the topological structure of C-space by sampling technique in a learning phase and then to search for a solution path in the roadmap in a query phase. Several methods tailor the original PRM framework to solve path planning problems in dynamic changing environments. In particular, DRM proposed in [9][10], and a hybrid method suggested in [14], which combines the merit of Lazy Evaluation and RRT are among those PRM-based methods.

A. DRM

The key idea of DRM to cope with dynamic changes is to build a mapping from W-space to C-space. In its learning phase the W-space is decomposed into unit cells with each one corresponding to some nodes and edges in the roadmap. In the query phase, nodes and edges corresponding to cells overlapped by obstacles are invalidated. The relationship between W-space unit cells and roadmap is called WC-mapping which consists of nodes mapping and edges mapping and can be defined as follows:

$$\Phi_n(w) = \{v \in G(V) \mid w \in \Omega(v)\}, \tag{1}$$

$$\Phi_a(w) = \{e \in G(E) \mid w \in \Omega(e)\}, \tag{2}$$

where $\Phi_n(w)$ and $\Phi_a(w)$ denote node mapping and edge mapping, respectively. The roadmap is represented as an undirectional graph $G(V,E)$, $G(V)$ is the set of roadmap milestones and $G(E)$ denotes the set of roadmap edges. $w \in \mathscr{W}$ is a unit cell of workspace, $\Omega(v)$ represents the subset of unit cells occupied by the robot at configuration v and $\Omega(e)$ denotes the swept cells of motion e.

However, it's difficult to compute Φ_n and Φ_a directly. But the inverse mappings, Φ_n^{-1} and Φ_a^{-1}, can be computed with less effort and the original mappings can be computed through them. The inverse mappings are defined as follows:

$$\Phi_n^{-1}(v) = \{w \in \mathscr{W} \mid w \in \Omega(v)\}, v \in G(V), \tag{3}$$

$$\Phi_a^{-1}(e) = \{w \in \mathscr{W} \mid w \in \Omega(e)\}, e \in G(E). \tag{4}$$

Although the WC-mapping is an effective roadmap maintenance mechanism, it will be always impossible for the DRM method to find a solution if the environment at query time varies a lot. Even though such a solution has been found, it will be obstructed soon by moveable obstacles. See Fig. 1 for example. DRM is not capable of solving planning problems in such environments.

B. Lazy Evaluation with RRT Reconnection

The efficient Lazy Evaluation idea is firstly introduced to dynamic planning field in [14]. In its learning phase, a roadmap is computed for robot in static environment without moveable obstacles. Then in the query phase, the planner first searches for a path in the roadmap and then checks it for collision. While checking the path, nodes and edges obstructed by moveable obstacles are set as invalid. If the path is collision-free, it will be the solution. Otherwise, a local connection with the RRT-connected algorithm is employed to reconnect those blocked edges. When all reconnections have been attempted without success, the roadmap is enhanced via inserting more nodes and edges into it.

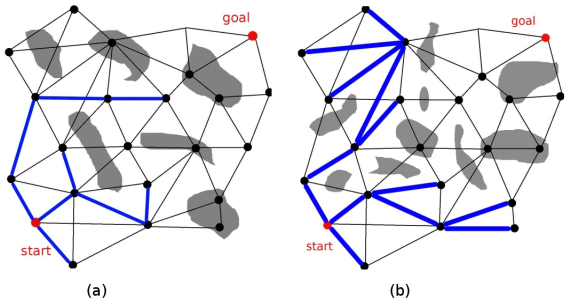

Fig. 1. (a) There is no solution in the roadmap. (b) There is still no path, though the obstacles have changed their configurations.

However, if environment is cluttered by moveable obstacles, the local connection and the roadmap enhancement mechanism won't achieve good performance.

C. Motivation and Overview of Our Method

The motivation of our method is to make robot move towards its goal configuration as much as possible, even though a completely valid solution path does not exit currently.

Our approach proceeds in two phases like PRM, a learning phase and a query phase. In the learning phase, a roadmap is constructed without considering any obstacles. Since only self-collision of robot should be avoided, the roadmap construction process is considerably fast. In the query phase, our planner first searches for a solution path with the A^* algorithm and then checks whether it is collision-free with obstacles. If all nodes and edges in the path belong to $\mathscr{C}_{free}$, the path will be adopted immediately. Otherwise, those nodes and edges obstructed by obstacles will be labeled and invalidated in the roadmap. At the same time the maximum valid segment of the path is extracted and saved into a container. The maximum valid path segment, $S(P)$, is defined as follows:

$$P = n_0 e_0 n_1 e_1 \cdots e_{m-1} n_m, \tag{5}$$

$$\begin{aligned} &S(P) = n_0 e_0 \cdots e_{k-1} n_k, 0 \le k \le m, \\ &\forall i (0 \le i \le k)\,(n_i \in \mathscr{C}_{free} \wedge (n_{k+1} \in \mathscr{C}_{obs} \vee e_k \subset \mathscr{C}_{obs})), \end{aligned} \tag{6}$$

where P is a solution path in the roadmap while n_0 and n_m are the start and goal, respectively. $n_i \in G(V)$, $e_j \in G(E)$, $e_j = (n_{j-1}, n_j)$. $\mathscr{C}_{free}$ is the open subset of collision-free configurations and $C_{obs} = \mathscr{C} - \mathscr{C}_{free} = \{c \mid c \in \mathscr{C} \wedge c \notin \mathscr{C}_{free}\}$.

Once there is no completely valid path in the roadmap, the planner will select the optimal segment from the container. In the motion execution process, when the path segment is obstructed by moveable obstacles or the endpoint is reached, the current configuration will be set as the start of the next roadmap query process until the robot reaches its goal.

III. DETAILS OF OUR APPROACH

A. Initial Roadmap Construction

The first step is to build a roadmap, $G(V,E)$. Nodes that encode feasible motion of a robot is sampled in its C-space without considering any obstacles. Many effective sampling techniques have been introduced to construct a robust roadmap in different tasks. The details of those techniques can be found in [15][16][17]. A comparative study on them has been presented in [18]. Moreover a method called random halton is recommended. After those sampling nodes are randomly generated in the C-space, a local planner is called to connect them. Because obstacles are not considered here, the local planner only needs to check whether self-collision happens. Roadmap construction is considerably fast accordingly. In our implementation, a binary local planner is employed with the nearest-k connection strategy. The rationale of the local connection strategy is also stated in [18]. After the construction process is accomplished, a given start and a goal configurations will be connected to the roadmap with the local planner.

Since the real-time performance is the most essential criterion, a robust roadmap that covers C-space adequately is needed and roadmap enhancement shouldn't be performed while planning.

B. Roadmap Query Process of the Planner

Algorithm 1 explains the principle of our planner in detail and the general process is illustrated in Fig. 2. Our planner begins by searching for a solution path between the given start and goal in the initial roadmap. If a collision-free solution path exists, the motion sequence encoded by the path will be adopted (see Fig. 2a). Otherwise, those

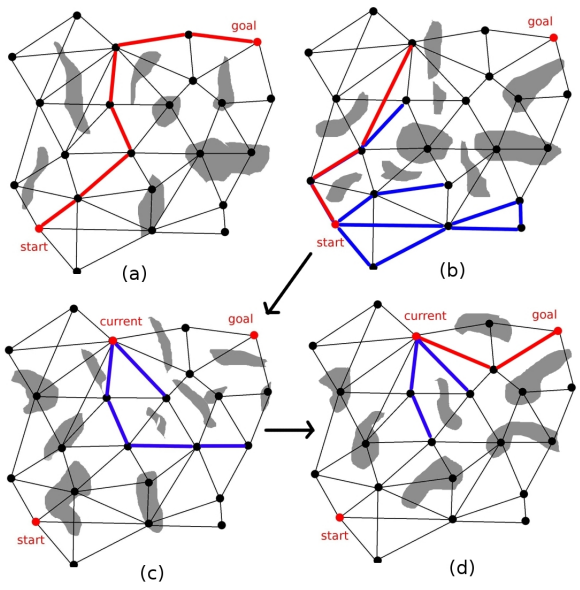

Fig. 2. (a) The completely path solution (red) is adopted immediately. (b) Several maximum path segments have been found but only the optimal one (red) of them is selected. (c) All segments are infeasible. (d) The robot reaches its goal.

Algorithm 1: Roadmap Query

Data: $G = G(V,E)$ is the roadmap. S is a maximum path segment. q_{init} is the initial state. q_{goal} is the goal state. $q_{endpoint}$ is the endpoint of the segment. S_{seg} is a set of maximum path segments.
Result: q_{goal} is reached.

```
/* S = n0 e0 ... ek nk+1, n0 = q_init, nk+1 = q_endpoint */
1  begin
2      S ← NULL;
3      S_seg = ∅;
4      while q_goal is not reached do
5          if S_seg ≠ ∅ then
6              S ← PopOptimalSegment(S_seg);
7              S_seg = ∅;
8              i := 0;
9              k ← GetSegmentLength(S);
10             while i ≤ k and S is not blocked do
11                 ExecuteMotion(e_i);
12                 i := i + 1;
13             if q_endpoint is reached then
14                 q_init := q_endpoint;
15             else
16                 q_init := q_current;
17         else
18             InitiateRoadmap(G);
19             S_seg ← CollectSegment(q_init, q_goal, G);
20 end
```

obstructed nodes and edges along the path will be labeled and invalidated in the roadmap and its maximum segment which has been defined in the previous section will be extracted and a container is used here to save the segment. The searching procedure is called repeatedly in the maintained roadmap. Until the start and goal are not connected with each other any more or the number of times that the path searching process is repeated exceeds a predetermined threshold, the optimal segment will be selected from the container if it is not empty (see Fig. 2b). Once the optimal segment is blocked or the endpoint is reached in the planning execution process, the current configuration will be returned and set as the start configuration of the next query. All those nodes and edges having been labeled will be cleared and set as valid again before next query. However, the robot won't execute any motion if the container is empty or all segments are infeasible (see Fig. 2c). Then the roadmap is re-initiated and searched again. When the robot reaches the goal, the planning task will terminate (see Fig. 2d).

The key step of Algorithm 1 is the maximum segment collection process (see the 19th line of the algorithm). The function `CollectSegment` is implemented to return a segment container. It searches in the roadmap for a path with the A^* algorithm (see the 3rd and 9th line in its pseudo-code) and calls the procedure `ExtractMaximumSegment`

Procedure `CollectSegment` (q_{init}, q_{goal}, G)

Data: $G = G(V,E)$ is roadmap. P is a path, if any. S is a maximum path segment. q_{init} is the initial state. q_{goal} is the goal state.
Result: S_{seg} is a set of maximum path segments.

```
1  begin
2      S_seg ← ∅;
3      P ← A* SearchPath(q_init, q_goal, G);
4      while P ≠ NULL do
5          S ← ExtractMaximumSegment(P);
6          S_seg ← S_seg ∪ {S};
7          if S = P then
8              return S_seg;
9          P ← A* SearchPath(q_init, q_goal, G);
10     return S_seg;
11 end
```

Procedure `ExtractMaximumSegment` (P)

Data: P is the path.
Result: S is the maximum path segment.

```
/* P = n_0 e_0 n_1 e_1 ... e_m n_{m+1}, n_0 = q_init, n_{m+1} = q_goal */
1  begin
2      S ← Add(S, n_0);
3      i := 0;
4      m ← GetPathLength(P);
5      while i ≤ m and n_{i+1} ∈ C_free ∧ e_i ⊂ C_free do
6          S ← Add(S, e_i, n_{i+1});
7          i := i + 1;
8      while i ≤ m do
9          if n_{i+1} ∈ C_obs then SetInvalid(n_{i+1});
10         if e_i ∩ C_obs ≠ ∅ then SetInvalid(e_i);
11         i := i + 1;
12     return S;
13 end
```

to extract its maximum path segment.

C. Maximum Path Segment Extraction

The segment extraction procedure is very important for our planner, thus it is discussed in detail here. The input of the procedure `ExtractMaximumSegment` is a path between two given configurations in the roadmap and the procedure returns the maximum valid segment. The key step (Line 6, 9, 10) of the procedure is to detect which nodes and edges are obstructed by obstacles. It determines the real-time performance of the proposed approach. Two different mechanisms, WC-mapping and online collision check, can be employed here.

*1) **WC-mapping**:* Like the DRM[9][10] method, our planner can utilize WC-mapping to cope with changes in the roadmap. However, the mapping is not employed to update roadmap directly. Instead it is only used to check whether nodes and edges along a path are obstructed by moving obstacles in the segment extraction process. The checking process can be defined as follows:

$$n_{i+1} \in \mathscr{C}_{obs} \Leftrightarrow n_{i+1} \in \Phi_n(\mathscr{W}_{obs}), \tag{7}$$

$$e_i \cap \mathscr{C}_{obs} \neq \emptyset \Leftrightarrow e_i \in \Phi_e(\mathscr{W}_{obs}). \tag{8}$$

Since $\mathscr{C}_{free} = \mathscr{C} - \mathscr{C}_{obs}$, it's obvious that

$$n_{i+1} \in \mathscr{C}_{free} \Leftrightarrow n_{i+1} \in \Big(G(V) - \Phi_n(\mathscr{W}_{obs})\Big), \tag{9}$$

$$e_i \subset \mathscr{C}_{free} \Leftrightarrow e_i \in \Big(G(E) - \Phi_e(\mathscr{W}_{obs})\Big), \tag{10}$$

where $\mathscr{W}_{obs}$ is the set of cells overlapped by obstacles.

*2) **Online Collision Check**:* After a path is found with the A^* algorithm, the planner checks nodes and edges for collision online with a collision checker. Since an invalid node implies that all its corresponding edges are invalid, it's reasonable to check endpoints of the edges for collision first. In other words, if $n_i \in \mathscr{C}_{obs}$ or $n_{i+1} \in \mathscr{C}_{obs}$, then $e_i \cap \mathscr{C}_{obs} \neq \emptyset$. There is no need to check feasibility of e_i any more accordingly.

After the segment is extracted, the remains of the path are also checked for collision. The obstructed residue will be labeled invalid in the roadmap as well (see line 8–11 in the pseudo-code).

D. Optimal Maximum Segment Selection

The procedure `PopOptimalSegment` greatly determines performance of our method. In the 6th line of Algorithm 1, the planner invokes it to select the optimal maximum path segment from the segment container S_{seg}: $S \leftarrow$ `PopOptimalSegment`(S_{seg}).

Once a maximum segment is extracted, an evaluation scheme is needed to determine its feasibility. Three evaluation criteria are employed here:

1) energy cost (minimize);
2) energy required for future execution (minimize);
3) execution safety.

*1) **Energy Computation**:* The first two criteria are both concerning energy consumption. According to the detailed mechanical parameters of robot, it's possible to calculate the accurate energy cost through a large number of mathematical operations. However, a simple but effective enough approximation is needed in experiments since the accurate energy consumption is not our interest.

After a segment is extracted, the changing quantity of mechanical parameters is computed for each link. As we all know that energy consumption varies directly with changes in parameter, so energy cost can be approximately represented with changing quantity of mechanical parameters.

Let c_1 denotes the start configuration of a collision-free motion e, while c_2 is the stop configuration. Assume that $c_1 = (x_1, x_2, \ldots, x_n)$, $c_2 = (x'_1, x'_2, \ldots, x'_n)$. The corresponding mechanical parameters of c_1 and c_2 are p_1 and p_2, respectively. We denote $p_1 = (\theta_1, \theta_2, \ldots, \theta_n)$ and $p_2 =$

$(\theta_1', \theta_2', \ldots, \theta_n')$. Here Ψ is the energy evaluation function defined as following:

$$\Psi : \mathscr{C} \times \mathscr{C} \to \mathbb{R},\ \Psi(c_1, c_2) = \sum_{i=1}^{n} \varepsilon_i \Delta\theta_i = \sum_{i=1}^{n} \varepsilon_i (\theta_i' - \theta_i), \quad (11)$$

where ε_i is the weight value of link i. Let's consider a path $P = n_0 e_0 n_1 e_1 \cdots e_{m-1} n_m$ and its maximum segment $S = n_0 e_0 \cdots n_{k-1} e_{k-1} n_k, 0 \le k \le m$. The energy consumption of the segment is $\Psi(S) = \sum_{i=1}^{k} \Psi(n_{i-1}, n_i)$. Since the C-space continuously changes its form with obstacles moving in the environment, it's impossible to compute the accurate energy required for future execution. In our method, it is estimated as $\Psi(P - S) = \sum_{i=k+1}^{m} \Psi(n_{i-1}, n_i)$.

*2) **Safety Degree Estimation**:* The safety degree can't be calculated easily as the energy cost. We even don't have a criterion in strict sense to judge whether a path segment is safer than an other one. A simple and approximate method is used to estimate safety degree here.

While building the roadmap, the planner gives each node and edge a reference counter with initial value 0. In the query phase, all nodes and edges obstructed by obstacles increment their reference counters. By contrast, the reference counter will be decremented once the node or edge is not blocked. The safety degree of the segment, $S = n_0 e_0 \cdots n_{k-1} e_{k-1} n_k, 0 \le k \le m$, can be estimated through the total sum of the reference counters:

$$\Gamma(S) \propto ref(n_m) + \sum_{i=0}^{m-1} \Big(ref(n_i) + ref(e_i) \Big), \quad (12)$$

where Γ is the safety degree estimated function. The larger $\Gamma(S)$ is, the more dangerous S is.

*3) **Segment Feasibility Computation**:* The value of energy cost, energy for future execution and safety degree determine feasibility of the segment. Let's consider the path $P = n_0 e_0 n_1 e_1 \cdots n_{m-1} e_{m-1} n_m$ and its maximum segment $S = n_0 e_0 \cdots n_{k-1} e_{k-1} n_k, 0 \le k \le m$. The feasibility can be computed as:

$$Feas(S) = \omega_1 \Psi(S) + \omega_2 \Psi(P - S) + \omega_3 \Gamma(S), \quad (13)$$

where $\omega_i, (i = 1, 2, 3)$ is the weight value that indicates the importance of corresponding factor. In fact the feasibility of segments, i.e. $Feas(S)$, is not calculated explicitly when the planner is implemented. Instead it embeds the computation of $\omega_1 \Psi(S)$ and $\omega_3 \Gamma(S)$ into the A^* path searching procedure. The optimal segment S is the one with minimum $Fea(S)$.

E. Real-time Performance Optimization

In some cases the roadmap is complicated. For example, the number of nodes, i.e. $|G(V)|$, is large or $K_{nearest}$ has a big value. In such cases, the procedure `CollectSegment` will search for path a great many times until no path exits any more, which makes the procedure `ExtractMaximumSegment` invoked over and over again. Moreover, after several segments have been extracted, those segments left over are scarcely feasible. Thus it's reasonable to limit the number of times that the path searching procedure could be executed.

IV. EXPERIMENTS AND ANALYSIS

For evaluating our method, we build a manipulator simulation for 6-DOF Kawasaki robot manipulator (FS03N). The planner has been implemented in C++ and the collision checks are handled with ColDet 1.2, an open-source 3D collision detection library. The simulation program (see Fig. 3) is running under Linux (Ubuntu 8.04) on an AMD 4000+ processor with 3 GByte of memory. In order to test the performance of the proposed method, experiments are performed within different scenarios.

A. Experiment Scenarios

In the first scenario, we compare the performance of our method with the DRM method. Just like DRM, our planner utilizes the WC-mapping to maintain the roadmap. The experimental scenario consists of two FS03N manipulators and several obstacles (see Fig. 4a). The two manipulators are treated as one entity. Thus the C-space is 12-dimensional. We approximately represent the reachable workspace as a cuboid with the size of $160 \times 280 \times 120\ cm^3$. For building the WC-mapping, the W-space is decomposed into $80 \times 140 \times 60$ grids. The initial state of manipulators is shown in Fig. 4a. The task of manipulators is to touch a target object (the purple cylinder) at the same time. The manipulators' goal state is given in Fig. 4b.

In the second scenario, there are four KS03N manipulators. The number of C-space dimension increases to 24. The volume of cuboid which roughly represents the W-space is $260 \times 240 \times 120\ cm^3$. Since the C-space is much more complicated than the first scenario, a larger roadmap is needed to cover the volume of the free C-space adequately. Experimental results in [10] show that the size of WC-mapping for maintaining roadmap grows dramatically with the roadmap size increasing. Accordingly only the Online Collision Check mechanism is tested. The Lazy Evaluation method is implemented in control experiments to show that our method can greatly increase the success rate for the manipulators operating in complicated environments. The initial and goal states are shown in Fig. 4c and Fig. 4d, respectively.

For both scenarios, it is assumed that the robots' operating velocity is not very fast and the acceleration is small. Taken the planning safety of HRI problems into consideration, those assumptions are acceptable.

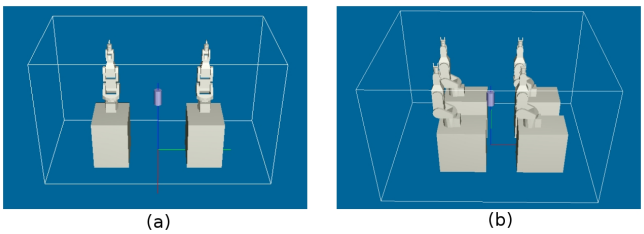

Fig. 3. (a) Experiment scenario 1 (b) Experiment scenario 2

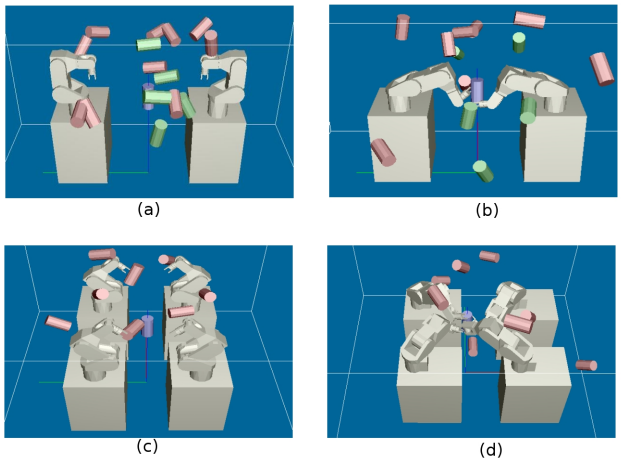

Fig. 4. (a) The given configuration in the scenario 1 (b) The given goal configuration in the scenario 1 (c) The initial configuration in the scenario 2 (d) The goal configuration in the scenario 2

B. Performance Results and Analysis

For each experiment, 100 tasks have been randomly generated. They are all valid, but some of them are impossible to solve under certain stipulation. The planner is deemed to fail to solve the problem when the manipulators can't reach the goal configurations within a time threshold by either of the methods. The time threshold for the first scenario is 30 seconds and since the tasks for the second scenario are much harder than those in the first one, the threshold for it is 60 seconds.

1) Comparison with DRM: Firstly, we compare the performance of our method with DRM. See Table I. The number of roadmap nodes, $|G(V)|$, is 10^3 and the $K_{nearest}$ is 5. Three kinds of experiments with different difficulty levels have been performed. The content of the third row presents the number of obstacles in form of $a+b$, which reflects the difficulty degree of the experiment. Here a is the number of moveable obstacles at normal speed and b represents the number of obstacles at a bit slower speed which could hinder the movement of manipulators more often. The experiment data shows that not only the success rate of our planner increases much with the maximum path segment searching algorithm (see the last row of Table I), but also the total time of planning is reduced (see the 6th row). Since our planner needs to search for several segments in one query, the A^* path searching procedure will be invoked several times and the average time for roadmap query is a bit more than DRM (see the 5th row).

TABLE I
COMPARISON WITH DRM

Experiment	task $1\sim100$		task $101\sim200$		task $201\sim300$	
Grids (cm^3)	$160\times280\times120$					
Obstacles	$10+0$		$5+5$		$10+5$	
Method	DRM	Our	DRM	Our	DRM	Our
Query Time (s)	0.39	0.47	0.32	0.41	0.56	0.69
Total Time (s)	24.84	**19.62**	27.18	**23.90**	29.90	**28.12**
Success Rate	48%	**80%**	27%	**51%**	2%	**28%**

2) Comparison with Lazy Evaluation: The comparative result is shown in Table II. Since the second scenario is more complicated than the first one, a large roadmap is needed. $|G(V)|$ is 10^4 and $K_{nearest}$ is 15. It's inaccessible to precompute a WC-mapping for the manipulators because the size of mapping will be intolerable. Accordingly the planner employs the Online Collision Check to maintain roadmap. We compared its performance with Lazy Evaluation method. From the table we can see that our planner is far better than its counterpart. It has higher success rate with much fewer collision checks and better real-time performance. The performance data also show that the Lazy Evaluation method is not suitable for solving dynamic planning problems in a complicated environment. Because $|G(V)|$ or $K_{nearest}$ has a large value, the roadmap $G(V,E)$ will have such a good connectivity that the A^* path searching procedure would be invoked for a great many times until no path exits in the roadmap. In some task the query time of Lazy Evaluation planner is up to 59.38 which is unacceptable.

3) Comparison between Different Roadmap Maintenance Mechanisms: In Table III we show the performance of our planner in detail and compare the two roadmap maintenance mechanisms—WC-mapping and Online Collision Check which we've discussed in section III-C. From the first four rows we find that the performance of WC-mapping declines sharply with the increase in roadmap size though the C-space is covered more adequately. For instance, the success solving rate declines from 80% to 27%, while $|G(V)|$ increases from 10^3 to 2×10^3 and the roadmap query time increases sharply. Because there will be more pairs of cell such that $\Phi_n(w_1)\cap\Phi_n(w_2)\neq\emptyset$ or $\Phi_e(w_1)\cap\Phi_e(w_2)\neq\emptyset$ if the size of roadmap increases. The intersecting cells make some nodes and edges in the roadmap labeled and re-validated for many times meaninglessly. If the WC-mapping mechanism is employed, collision check is not needed online. Therefore, some grids of the table are blank. We can see from the table that the Online Collision Check is more effective here. The time for roadmap query doesn't grow rapidly with the increase in the size of roadmap. More importantly the success rate is much higher. We also find that the average number of node and edge collision check declines with $|G(V)|$ increasing.

V. CONCLUSIONS

The main contribution of this paper is to introduce a novel planner for solving dynamic path planning problems. The proposed planner has been tested with simulations of manipulator on static pedestal in two scenarios and experiment data shows that the planner has following promising characteristics: 1) The planner can solve planning problems in high-dimensional configuration spaces. 2) Due to the good performance of maximum path segment searching algorithm, our rate of successful planning increases considerably, compared to those methods which require a completely valid path. 3) The safety of planner is considered as well.

TABLE II

COMPARISON WITH LAZY EVALUATION

Method	Experiment (Scenario 2)			Query Time (s)			Total Time (s)		Node Check			Edge Check			Success
	Task	Nodes	Obstacles	Min	Max	Avg	Min	Avg	Min	Max	Avg	Min	Max	Avg	Rate
Lazy	$001 \sim 100$	10^4	$5+0$	< 0.01	**17.17**	0.55	24.76	54.82	4	3862	1161.21	10	3601	808.32	**28%**
Lazy	$101 \sim 200$	10^4	$5+5$	0.02	**59.38**	0.71	28.65	58.84	4	2774	1159.73	10	1972	653.41	**7%**
Our	$001 \sim 100$	10^4	$5+0$	< 0.01	**0.36**	0.13	12.50	39.06	25	3341	643.98	16	2017	400.17	**85%**
Our	$100 \sim 200$	10^4	$5+5$	0.01	**4.02**	0.20	20.21	50.573	50	3919	1213.10	28	1709	393.13	**51%**

TABLE III

PERFORMANCE OF PROPOSED METHOD

Method	Experiment (Scenario 1)			Query Time (s)			Total Time (s)		Node Check			Edge Check			Success
	Task	Nodes	Obstacles	Min	Max	Avg	Min	Avg	Min	Max	Avg	Min	Max	Avg	Rate
Mapping	$001 \sim 100$	10^3	$10+0$	0.08	1.50	0.48	7.99	19.62							80%
Mapping	$101 \sim 200$	10^3	$5+5$	0.04	1.60	0.42	6.98	23.90							49%
Mapping	$201 \sim 300$	10^3	$10+5$	0.10	1.60	0.69	13.42	28.12							28%
Mapping	$001 \sim 100$	2×10^3	$10+0$	0.28	6.73	2.55	14.57	28.24							26%
Col-Check	$001 \sim 100$	10^3	$10+0$	< 0.01	0.10	0.047	6.95	14.92	5	1605	252.11	3	724	184.58	97%
Col-Check	$101 \sim 200$	10^3	$5+5$	< 0.01	0.13	0.05	6.62	18.74	9	2090	569.25	7	1225	287.43	78%
Col-Check	$201 \sim 300$	10^3	$10+5$	< 0.01	0.19	0.068	10.73	25.62	25	1749	664.46	19	895	316.31	48%
Col-Check	$001 \sim 100$	2×10^3	$10+0$	< 0.01	0.17	0.082	10.14	20.33	7	2108	281.11	4	1298	188.90	85%
Col-Check	$101 \sim 200$	2×10^3	$5+5$	< 0.01	0.18	0.093	9.66	19.91	9	1936	424.77	8	1030	239.07	73%
Col-Check	$201 \sim 300$	2×10^3	$10+5$	< 0.01	0.21	0.10	11.76	26.88	23	1523	613.73	22	760	313.23	40%
Col-Check	$001 \sim 100$	3×10^3	$10+0$	< 0.01	0.14	0.080	7.06	12.88	5	1280	193.13	3	626	153.19	93%
Col-Check	$101 \sim 200$	3×10^3	$5+5$	< 0.01	0.21	0.084	6.37	22.03	2	915	192.88	2	569	97.03	80%
Col-Check	$201 \sim 300$	3×10^3	$10+5$	< 0.01	0.24	0.096	8.97	25.97	25	1316	499.98	20	617	221.49	45%

VI. ACKNOWLEDGMENTS

This work is supported by National Natural Science Foundation of China (NSFC, No. 60675025,60975050) and National High Technology Research and Development Program of China (863 Program, No. 2006AA04Z247), Projects of Shenzhen Bureau of Science Technology and Information.

REFERENCES

[1] J.F. Canny, "The Complexity of Robot Motion Planning", MIT Press, Cambridge, MA.

[2] S.M. LaValle, "Planning Algorithm", Cambridge University Press.

[3] J.C. Latombe, "Robot Motion Planning", Kluwer, Academic Publishers, 1991.

[4] L.E. Kavraki, P. Svestka, J.C. Latombe, and M.H. Overmars, "Probabilistic roadmaps for fast path planning in high-dimensional configuration space", *IEEE Transactions on Robotics and Automation,* pp. 566-580, 1996.

[5] R. Bohlin and L.E. Kavraki, "Path planning using Lazy PRM", *IEEE International Conference on Robotics and Automation,* pp. 5215-28, 2000.

[6] S.M. LaValle, "Rapidly-exploring random trees: a new tool for path planning", *Technical Report TR 98-11, Computer Science Dept., Iowa State University,* 1998.

[7] J.J. Kuffner and S.M. LaValle, "RRT-connect: an efficient approach to single-query path planning", *IEEE International Conference on Robotics and Automation,* pp. 995-1001, 2000.

[8] H. Liu, X. Deng and H. Zha, "A planning method for safe interaction between human arms and robot manipulators", *IEEE International Conference on Intelligent Robots and Systems,* pp. 1814-1820, 2005.

[9] P. Leven and S. Hutchinson, "Toward real-time path planning in changing environments", *Proceedings of the Fourth International Workshop on the Algorithmic Foundations of Robotics,* pp. 363-376, 2000.

[10] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments", *The International Journal of Robotics Research,* Vol. 21, pp. 999-1030, 2002.

[11] M. Kallmann and M. Mataric, "Motion planning using dynamic roadmaps", *IEEE International Conference on Robotics and Automation,* pp. 4399-4404, 2004.

[12] D. Ferguson, N. Kalra and A. Stenz, "Replanning with RRTs", *IEEE International conference on Robotics and Automation,* pp. 1243-1248, 2006.

[13] J. Bruce and M. Veloso, "Real-time randomized path planning for robot navigation", *IEEE International Conference on Intelligent Robots and Systems,* pp. 2383-2388, 2002.

[14] L. Jaillet and T. Simeon, "A PRM-based motion planner for dynamically changing environments", *IEEE International Conference on Intelligent Robots and Systems,* pp. 1606-1611, 2004.

[15] V. Boor, M.H. Overmars and A.F. Van Der Stappen, "The Gaussian sampling strategy for probabilistic roadmaps planners", *IEEE International Conference on Robotics and Automation,* pp. 1018-1023, 1999.

[16] D. Hsu, T. Jiang, R. John and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners", *IEEE International Conference on Robotics and Automation,* pp. 4420-4426, 2003.

[17] N.M. Amato, O.B. Bayazit, L.K. Dale, C. Jones and D. Vallejo, "OBPRM: an obstacle-based PRM for 3D workspaces", *Proceedings of the Third International Workshop on the Algorithmic Foundations of Robotics,* pp. 155-168, 1998.

[18] R. Geraerts and M.H. Overmas, "A comparative study of probabilistic roadmap planner", *Proceedings of the Fifth International Workshop on the Algorithmic Foundations of Robotics,* pp. 249-264, 2002.

[19] H. Liu, X. Deng, H. Zha and D. Ding, "A path planner in changing environments by using W-C nodes mapping coupled with lazy edges evaluation" *IEEE International Conference on Intelligent Robots and Systems,* pp. 4078-4083, 2006.

[20] J.P. van den Berg and M.H. Overmars, "Roadmap-based motion planning in dynamic environments", *IEEE Transactions on Robotics,* pp. 885-897, Vol. 21, 2005.

[21] J.P. van den Berg, D. Nieuwenhuisen, L. Jaillet and M.H. Overmars, "Creating robust roadmaps for motion planning in changing environments", *IEEE International Conference on Intelligent Robots and Systems,* pp. 2415-2421, 2005.

[22] Y. Kitamura, F. Kishino, T. Tanaka and W. Yachida, "Real-Time Path Planning in a Dynamic 3-D Environment" *IEEE International Conference on Intelligent Robots and Systems,* pp. 925-931, 1996.