A Dynamic Subgoal Path Planner for Unpredictable Environments

Hong Liu, Weiwei Wan and Hongbin Zha

Abstract—Although lots of planning algorithms have focused on the planning of fixed manipulators and mobile robots in moderate dynamic environments, seldom planning algorithms can be employed to deal with mobile agents in the presence of large scenario scales and unpredictable changing obstacles. Path planning for mobile robots in unpredictable environments would be an extreme challenge since computational complexity increase dramatically with high dimensionality, unpredictability and large scales. In this paper, a novel and real-time approach is proposed to solve this problem by generating subgoals dynamically according to time and potential values. This dynamic subgoal based approach includes two procedures, the subgoal generator and the inter-subgoal replanner. On the one hand, a set of high-level subgoals is generated dynamically by an improved single shot strategy that could tailor itself adaptively. On the other hand, a roadmap is built during the preprocessing phase by employing a localized Dynamic Roadmap Mapping (local-DRM) for inter-subgoal replanning. Finally these two procedures will collaborate according to the potential field criterion to ensure completeness. Our approach can not only generate paths rapidly enough to satisfy the requirements of an anytime planner but also work for large scenario scales. Experimental results on different kinds of mobile agents, in large scenario scales and in the presence of unpredictable changing obstacles show that our approach can find out a collision free path on an average of 0.11s for a single planning, indicating an anytime planner.

I. INTRODUCTION

Since the presence of random algorithms, such as the popular probabilistic roadmap method (PRM)[1] and the rapidly-exploring randomized tree method (RRT)[2] method, path planning study has improved significantly. The derivatives of these algorithms can not only solve the traditional piano mover’s problems, but also be competent for path planning in moderate dynamic environments[3][4]. However, this is not always the case in unpredictable environments. Recent progresses[5][6] seek to solve such problems by the idea of anytime planning. Their works are promising because their realtime performance. However, such a problem remains challenging due to high DOFs, large scenario scales and unpredictable changing obstacles.

In this paper, we seek an anytime solution based on the idea of dynamic subgoals where single query and multiple query primitives are borrowed as subgoal generator and inter-subgoal replanner respectively. Single query strategy[7] will find a collision free path for robots without the preprocessing phase. Generally speaking, it is much faster than a multiple query strategy taking into account the consumptive preprocessing phase of the later one. Like heuristic searching algorithms, single query strategy tries to consider as less redundant configurations as possible to generate feasible paths. This is usually realized by a goal-biased sampling scheme since there is no need for a complete planner in most scenarios. References [2] and [8] are examples of this strategy. Although single query strategy plays an important role in changing environments where multiple query strategy (PRM and its variants[9][10]) seems infeasible, it is not efficient enough for anytime requisites. On the contrary, multiple query strategy can take advantage of the prebuilt roadmaps to rapidly replan motions. DRM [11][12][13], which is a derivative of PRM, can effectively plan for fixed base manipulators or high-DOF robots by employing a mapping between W space and C space. It can find a collision-free path in the presence of both stationary and changing obstacles and can satisfy the requisites of an anytime planner. Nevertheless, the planner encounters great challenges when these manipulators or robots are mounted on mobile bases. (1) The increased C Space dimensions require more samples or a more sophisticated sampling schema. (2) The movements of obstacles become more drastic due to relative motion. (3) The extension of W Space makes \( W-C \) mapping more complicated.

The collaboration of these primitives helps a lot in overcoming those shortages. For one thing, subgoals from single query primitives can help reduce the size of \( W \) space and the amount of configurations required for mapping. For another, multiple query primitives can replan rapidly in an inter-subgoal space and improve the efficiency of tree exploring. At first, the single query based procedure explores in relatively large steps with a adaptively increasing biased probability and generates pivots that are called subgoals without checking configurations along the edges between them. Then, these subgoals will be treated as query configurations and submitted to the multiple query based procedure for inner replanning. From another viewpoint, this is like the lazy evaluation strategy[10] where lazy collision detection is innovated by local planning. This idea is from what people do in their daily motion planning. People would not or cannot plan specific motions far away when one is in a new environment. Usually, what they are going to do is moving toward their destination or destination lists with only local planning. Here the subgoals play the role of destination lists. The global director is the single query procedure that moves a man toward the goal direction, while the inter-subgoal
planner is the multiple procedure. Note that the subgoals in our approach are always changing according to time and potential values (refer to Section V), we name them dynamic subgoals.

Our main contributions are planning in unpredictable environments as following.

- Mobile robots usually encounter relatively large scales due to their mobility. The planner in this paper can dynamically generate paths in large scenarios.
- Changes of obstacles become unpredictable due to their movements and relative motions. The planner can avoid the obstacles and reach the destination tactically.

The rest of this paper is organized as follows. Related works are presented in Section II. Section III discusses anytime planning. In Section IV and Section V, details and the overall framework of the planner are presented respectively. Experiments and analysis are introduced in Section VI. Section VII draws the final conclusions followed by acknowledgement.

II. BACKGROUND WORKS

The idea of subgoals is not new and relates to many previous works and the concept of subgoals has been employed to reduce computational complexity of planning paths for long[15][16]. However, subgoals in these articles cannot be transplanted into our scenario directly due to the requirement of efficiency for an anytime planner. Typical approaches usually store those pregenerated subgoals for future usage. Yet such storage helps little in changing environments. For instance, suppose that there is a sequence of subgoals \( G_i \) in the subgoal list \( L_{sg} \). Then there will be a time period \( T_i \) for each subgoal position \( G_i \) that a robot cannot arrive at in \( T_i \). Due to the motions of obstacles, \( G_i \) may become obstructed after such a \( T_i \) (or when the robot arrives at \( G_i \)). Consequently, it is helpless to store \( G_i \) and replanning should be performed in \( T_i \) to generate new subgoals for replanning. In this case, the subgoal problem in this paper becomes different from those related works and we must attempt new solutions. These subgoals are not employed to store information but act as pivots to the final aim. Since the environments change along time, these subgoals are regenerated according to the change, namely dynamic subgoals.

Another idea is the collaboration of single query and multiple query primitives. Sampling based Roadmap of Trees(SRT)[14] is a pioneer of this idea. In SRT, configurations in the preprocessed roadmap are substituted by single query trees. SRT can better model \( C_{free} \) spaces and does well in complicated environments (narrow spaces). However, SRT is not suitable for realtime applications although it can refer to parallelized computing to improve its efficiency. Unlike SRT, the approach in this paper is fast enough on ordinary processors and seeks a balance between completeness and efficiency.

III. ANYTIME PLANNING

The key point of a planner for unpredictable changing environments is rapid replanning at any necessary time. And the key point of an anytime planner is to plan online with as less time as possible. In this section requisites of such a planner will be discussed. In reality, when obstacles are perceived at a specified time \( t_w \), planners could usually start the computation immediately. However, motions cannot be carried out until \( t_w + \tau \) where \( \tau \) is the period of time required for planning the path. In a scenario \( s \), \( \tau \) must satisfy the inequation \( \tau < t_s \) where \( t_s \) implies the dynamic attribute of the current scenario1, namely how the obstacles are changing. \( \tau \) and \( t_s \) are illustrated in detail as following.

- \( \tau \) The smaller \( \tau \) is, the faster the planner will be.
- \( t_s \) A smaller \( t_s \) denotes an environment with more drastically moving obstacles.

In summary, an anytime planner should be able to generate a path as quickly as possible to lower \( \tau \) and to satisfy a small enough \( t_s \) on the premise of a given completeness. See Fig.1 to fix the idea.

![Fig. 1. An illustration of CT space obstacles and the idea of ts.](image)

When trajectories of obstacles are known or predictable, \( CT \) obstacles is like an extracted \( C \) space obstacle along the time dimension. The left object in Fig.1 demonstrates such an obstacle. When the environment is unpredictable, \( CT \) obstacles cannot be represented in a particular shape anymore. We could only roughly say that the obstacles in such case are contained in a truncated cone (see the right object of Fig.1). Here the slope of the frustum surface is subject to the differential constraints of the environment. In reality, the higher the surface slope is, the smaller the \( t_s \) should be. See the distance along the time dimension between the two red lines (the distance indicates \( t_s \) here) in Fig.1 to fix the idea. It lies in the fact that regarding a scenario with drastically moving unpredictable objects, the textured volume of the truncated cone becomes much larger and only a small enough \( t_s \) can guarantee safe motions. Another way to explicate \( t_s \) is a smaller \( t_s \) promises more drastic changes of an unpredictable obstacle.

In our approach, the subgoal generator and inner replanner are mainly from RRT and DRM. Raw RRT structures are developed to quickly explore \( C \) space. They can rapidly select larger Voronoi regions for expansion. The complexity of RRT depends on the length of the solution path while the length of the solution path depends on the selection of parameters \( \delta_q \) and \( P_b \). Here, \( \delta_q \) is the step mounted to \( q_{near} \).
on the direction to \( q_{\text{rand}} \) and \( P_b \) is the probability of a new random sample being the goal configuration (refer to reference[2]). Formula (1) shows the role of \( \delta_q \):

\[
q_{\text{new}} = q_{\text{near}} + \delta_q
\]  

Here, \( q_{\text{new}} \) will be added to the tree if it is collision free. Despite the fact that larger \( \delta_q \) and \( P_b \) would lower the complexity of RRT significantly, they should not be tuned arbitrarily. Firstly, if a step \( \delta_q \) is too large, some obstacles that obstruct the edge between \( q_{\text{near}} \) and \( q_{\text{rand}} \) in the \( C \) space may be overlooked and the path may become invalid. Secondly, \( P_b \) should not be too high either. Although higher \( P_b \)'s give strong heuristics to goal points, they should be carefully chosen. In the worst case of \( P_b = 1 \), the RRT algorithm degenerates into a segment connector between the initial configuration and the goal one. Indeed, a higher \( P_b \) may lead RRT to local minima with a higher probability and hinder its application in generalized scenarios.

In this paper, we try to combine single query and multi-query primitives to make up respective drawbacks and improve performance. In order to improve the efficiency of RRT, we can tune \( \delta_q \) and \( P_b \). In order to apply DRM to mobile robots, we can lay limitations on the accessible range of \( W \) space. In our work, RRT primitives play the role of a high level guide as subgoal generators while DRM primitives will perform a detailed planning between those subgoals.

IV. SUBGOAL GENERATOR AND INNER REPLANNER

Primitives from RRT and DRM play the role of the two procedures employed in our approach, namely the subgoal generator and the inner replanner respectively. Details of them will be shown in the following subsections.

A. Subgoal Generator

The subgoal generator is a modified RRT based on the raw structure. RRT has many derivatives, for example RRT-connect[17] or those stores previous experiences[18][19][20]. Nevertheless, employing a more complicated variation here is unnecessary. As configurations \( G_i \) at a long distance away are not going to be arrived before time \( T_i \), samples at these \( G_i \) might become invalid due to the unpredictable movements of obstacles. Therefore, it is a waste of time to employ those variations that utilizes 'far away' information and only the elementary primitives are considered.

The modified RRT generator in this work is as following. Each time when replanning is invoked, the generator...
regenerates a path with a large $\delta_q$ and an adaptively tuned $P_b$ dynamically. See Algorithm 1 for details. The tuning of $P_b$ is highly dependent on the core of collaboration (namely when to replan), and it will be explicated in Section V.

Algorithm 1: Subgoal Generator

Input: $C_{global}^{init}$
Output: $L_{globalpath}^{init}$

1. $G_{global}.init(C_{global}^{init})$
2. while True do
   3. $C_{rand} \leftarrow \text{biased_rand_conf}(P_b)$
   4. $C_{near} \leftarrow \text{nearest_vert}(C_{rand}, G_{global})$
   5. $C_{vertex} \leftarrow \text{new_conf}(C_{new}, \delta_q)$
   6. if not check_collision($C_{new}$) then
      7. $G_{global}.add\_vertex(C_{new})$
      8. if check_dist($C_{new}, G_{goal}^{init}) \leq \delta_q$ then
         9. $L_{globalpath} \leftarrow \text{back\_trace}(C_{global}, G_{global})$
      10. return $L_{globalpath}$
   11. end
12. end

The input of the algorithm is $C_{global}^{init}$ which denotes the configuration when replanning is required. The output $L_{globalpath}^{init}$ is a list of configurations indicating the path generated by this global search strategy. $G_{global}$ denotes the exploring tree built during the generating of $L_{globalpath}^{init}$. $C_{rand}$, $C_{near}$ and $C_{new}$ are the same as those variables defined in a raw RRT planner, meaning different configurations employed when building the searching tree. A $C_{rand}$ is acquired by a biased random sampling strategy with the probability $P_b$, that is the function $\text{biased_rand_conf}()$. The nearest vertex to $C_{rand}$ in tree $G_{global}$ is chosen as $C_{near}$ in function $\text{nearest\_vert}()$. Function $\text{new\_conf}()$ moves $C_{near}$ along the edge between $C_{near}$ and $C_{rand}$ with a step $\delta_q$ to generate $C_{new}$.

When replanning is required, the planner will first regenerate a global path $L_{globalpath}$ with input of the current configuration, namely $C_{init}$ in Algorithm 1. Step $\delta_q$ in this subgoal generator is relatively large to improve the efficiency of raw RRT. Lines 8-10 show the criterion for termination. A new $L_{globalpath}$ is returned by back tracing the exploring tree when $C_{new}$ is in the vicinity of $C_{goal}^{init}$. The back tracing procedure is implemented by function $\text{back\_trace}()$.

B. Inner Replanner

The inner replanner is a localized DRM. Although the primitives (the mapping strategies, $A^*$ search) of DRM are employed in our local procedure, the local DRM focuses on different aspects.

- How to scale DRM to mobile robots working in arbitrary large scenarios
- How to make up the incompleteness of DRM mapping

This subsection will concentrate on the first aspect and the second aspect will be discussed in Section V.

1) Localizing: A local DRM tries to plan a local path for mobile robots locally. It only plans paths in a local $W$ space without considering the whole $W$ space which is neither possible to be known in advance nor solved or mapped in polynomial time. In this paper, DRM $W$ space is localized to solve this problem. See the shadow region in the right of Fig.2 to fix the idea of a localized $W$ space.

When a global path is generated by the subgoal generator, local DRM will employ these subgoals to plan locally. Local DRM will first update the invalidation of its $V$ and $E$ in the prebuilt local roadmap $G_{DRM}$. Then the two configurations will be inserted into $G_{DRM}$ for $A^*$ search. Note that the insertion of query configurations is carried out after updating $G_{DRM}$ and this indicates that the edges between query configurations and $G_{DRM}$ are supposed to be collision-free. Lazy evaluation between the query configurations and $G_{DRM}$ will be performed online when execution starts. Although this is a relatively time consuming procedure, it does little harm to our ‘anytime’ timer since only two edges at most are detected finally in one local search.

Besides, note that there is a high probability a path may not be found by the succeeding $A^*$ algorithm on the localized roadmap. This is highly relevant to the replanning schema and will be discussed in Section V.

2) Implementation of mapping: Instead of computing the complex mapping $\Phi_\omega(\omega)$ and $\Phi_\omega^{-1}(\omega)$, the inverse mapping $\Phi^{-1}_\omega$ and $\Phi^{-1}_\omega$ (or $\Omega(v_i)$ and $\Omega(e_{ij})$) are generated.

In the preprocessing phase, the roadmap $G$ is built without any obstacles in the predefined or local $W$ space ($W_l$ space in the following context). Note that only collision of inner robot is detected for each $v_i$ and $e_{ij}$ in this period. After that, focus goes to mappings $\Omega(v_i)$ and $\Omega(e_{ij})$.

Take computing $\Omega(v_i)$ for example, the robot in the $W_l$ space is first set to the configuration $v_i$ in $C$ space, and then the surfaces of the robot model in that configuration will be sampled with small vertices to locate the voxels obstructed by these surfaces. Compared with the traditional expansive seed method (a seed cell is put inside the robot and expanded in each direction until all cells $\Omega(v_i)$ occupied by the robot are found by collision checker), this strategy avoids explicit collision detection, lowers the amount of data of a specific mapping and makes the mapping of a relatively large $G$ possible. Fig.3 illustrates the idea of surface sampling.

The upper image in Fig.3 shows the results of surface sampling on two manipulators. The mapping results at a certain configuration $v_k$, namely $\Omega(v_k)$, is shown in the right image of Fig.3. Here voxels occupied by manipulators and obstacles are rendered with red and green cubes respectively. The reason why mappings of a robot could be substituted by the mappings of the robot surface lies in that when collision happens there is sure to be overlapped surface mappings before the overlapping of inner mappings. Thus it is unnecessary to map those space-consuming inner voxels. By employing this strategy, a lot of time and space can be saved that make the mapping of a larger $G$ pragmatic.

After the generation of these samples, voxels occupied by model surfaces can be easily indexed by the coordinates.
of them and the relations between $W$ space and $C$ space could be easily generated. Then these relations are stored as mappings for future usage.

V. OVERVIEW OF THE ALGORITHM

The overview of the anytime planner is shown in Fig.4. The two dash boxes in Fig.4 indicate the roles of subgoal generator and inner replanner respectively.

In the beginning, a mapping between $\omega$ and $G$ is generated in the preprocessing phase. The $W$-$C$ mapping box shown in Fig.4 demonstrates this idea. Then these mappings are employed by local DRM to search new paths online. As illustrated previously, when operation of a robot starts, the subgoal generator firstly generates a global path list $L_{globalpath}$. The first two configurations of $L_{globalpath}$ will then be sent to local DRM as $C_{init}^{local}$ and $C_{goal}^{local}$ to plan a collision free local path.

A. $W_i$ Space and $\delta_q$

One problem during this procedure is how to choose the size of $W_i$ space and the length of step $\delta_q$. Generally speaking, a smaller $W_i$ space implies a faster local DRM. But this is not always the case for the overall planning strategy. If $W_i$ space is too small, the planner may go into local minima easily and have to refer frequently to global planner for help. Also in general cases a larger $\delta_q$ could save the cost of a global planner significantly. However, if the step is too large, local mapping may become infeasible.

In reality, a robot should not move out of the current local region with a single step as show in formula (4). Or else it is possible that a subgoal cannot be inserted into the prebuilt roadmap of local DRM and the planner fails to generate a feasible local path.

$$d(q_{new}, base) \leq \text{sizeof}(W_i \text{ space})$$  (4)

In formula (4), $d()$ means the distance between the given configurations and $q_{new}$ denotes the configurations in a RRT. Finally, those $q_{new}$ along the feasible path will serve as subgoals.

$$\min(t(\delta_q, W_i))$$  (5)

s.t.  
$$\delta_q = f_q(p_{env})$$  (6)

$$W_i = f_w(p_{env})$$  (7)

In fact, how to choose $q_{new}$ and $W_i$ is highly relevant to parameters of the environments $p_{env}$ by function $f_q$ and $f_w$ respectively (constraints (6) and (7)). However, $p_{env}$ cannot be perceived accurately. In the worst case it is even unpredictable. In realization, $\delta_q$ and $W_i$ are chosen empirically through lots of experiments in specific scenarios.

B. Adaptive RRT and Replanning

Another problem is when to replan (this is also related to the dynamicity of subgoals). The blue area in Fig.4 demonstrates the replanning strategy employed in our realization. A key point in this course is the adaptive tuning of $P_b$. In the beginning $P_b$ is set to 1. When planning fails, our planner will increase $P_b$ by a certain amount to make a more randomized expanding direction. When a new planning (this happens at subgoals) is required, $P_b$ is set back to 1 and readapt itself. This helps a lot in saving energy (or making the results smooth) and getting out of local minima. In fact, thanks to the employment of adaptive $P_b$ tuning, our planner is as probability complete as raw RRT.

As shown in the blue area of Fig.4. The planner replans each time it encounters a perspective collision in the next step. In the following part we will demonstrate the idea lies behind the scheme.

Generally speaking, whether to replan for a new path depends on the requirements of safety or the distance between
obstacles and robots. Based on the idea of $W_I$ space division, the distance between obstacles and robots can be alternatively evaluated by potential fields. Fig.5 illustrates this idea. In Fig.5 the cube indicates the robot while the cuboid indicates the obstacles. Immediate $\omega$s of them are denoted with stroked grids respectively. The white segments show the generated global path and the red segments show the local details. The bounding box is $W_I$ space. When the obstacles in $W_I$ space is perceived, the planner will compute the potential fields based on the $\omega$s in $W_I$ space. At a specific configuration $v_i$, the alternative evaluation will be the largest potential at all $\Omega(v_i)$, refer to (8).

$$s(R) = \max(p(\Omega(v_i)))$$

(8)

In formula (8), $R$ denotes the robot, $s(\cdot)$ denotes the evaluation and $p(\cdot)$ denotes the potential. Since this $W_I$ space potential field approach is carried out in a limited workspace, computational complexity is fine enough for real-time application.

$$s(R) \leq 1 \iff \text{0-hard problem} \iff \text{exact CD}$$

(9)

In another view, $s(R)$ is like the $\theta$-hard [21]. In the hardest situation (0-hard problem), the calculation of $s(R)$ degenerates into an exact collision detection (CD), refer to formula (9).

Now the approach becomes seeking a minimum $t(\delta_q, W_I, s(R))$. Note that $\delta_q$ and $W_I$ are independent of $s(R)$ and they are chosen according to the strategy illustrated in the previous section. Here the discussion will focus on $s(R)$.

In order to ensure completeness, our approach takes planning problems as 0-hard ones and carries out replanning when $s(R) \leq 1$. This schema is implemented as following. $s(R)$ is not calculated explicitly while the potential evaluation approach is substituted with the combined exact collision detection and rough voxel testing to ensure a successful planner in the most rigid environments. The blue area in Fig.4 (the planner replans each time it encounters a perspective collision in the next step) demonstrates the specific steps of this idea.

VI. EXPERIMENTS

In order to evaluate the proposed method, hundreds of simulation experiments are implemented in 3D workspace with different scenarios. The experimental design mainly focuses on the ability to plan in the presence of large scenarios and unpredictable changing obstacles.

We ascribe these experiments into three groups where each group of experiments includes randomly moving obstacles and a different scale of scenario. Our experimental obstacles move and rotate by a random step along with time. Since the step is random, the obstacles become unpredictable.

All our experiments are carried out on an ordinary personal computer with Pentium D 2.80GHz CPU and 2GB memory. Dynamics in these experiments is implemented with the Open Dynamic Engine. Experimental results are based on an average of 100 executions.

Fig.6 demonstrates the scenarios of our environments.

(a) Scenario of experiment group I

(b) Scenario of experiment group II

(c) Scenario of experiment group III

Fig. 6. The scenario of experiment group I, II and III

The first group of experiments is to plan a 3 DOF vehicle among other unpredictable obstacles. All these objects are supposed to move on the ground (they cannot fly). In the second group of experiments, a 6 DOF space robot is planned to go through randomly rotating lattice obstacle. This scenario forms 'narrow passages' in dynamic environments. In the last group the algorithm is tested against a more complicated environment with stationary/randomly changing obstacles and a 9 DOF mobile manipulator. The manipulator in experiment group III is modeled by parameters of a practical 6 DOF Kawasaki manipulator (FS03N). The original manipulator is mounted on a vehicle base for mobility.

Settings of these experimental environments are listed in Table I.

Here, $r$, $o_i$, $s$ and $n_o$ in Table I represent robot size, the
size of the $i_{th}$ obstacle, scene size and number of obstacles respectively. The sizes shown in Table I are the AABB boundbox of the objects. For example, the robot in group I is a triangle cuboid, but only the AABB boundbox $7 \times 6 \times 2$ is given here to describe the shape roughly. Note that this is only for the convenience of showing the dimensions, collision detection itself is tested at each mesh exactly.

The motions of the unpredictable obstacles are defined according to reality. They are depicted by six parameters indicating the random walk steps along and rotation steps around the three Cartesian coordinates. The random walk steps are randomly chosen in (-5, 5) while the random rotation steps are chosen in (-60, 60). Note that the lattice obstacles in Group II are special. They rotate with a random step in (-30, 30).

### Table I

<table>
<thead>
<tr>
<th>Sizes</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$7 \times 6 \times 2$</td>
<td>$5 \times 5 \times 1$</td>
<td>$160 \times 244 \times 160$</td>
</tr>
<tr>
<td>$o_0$</td>
<td>$5 \times 10 \times 2$</td>
<td>$10 \times 15 \times 10$</td>
<td>$10 \times 15 \times 10$</td>
</tr>
<tr>
<td>$o_1$</td>
<td>$- \times - \times -$</td>
<td>$- \times 6 \times 2$</td>
<td>$- \times 6 \times 80$</td>
</tr>
<tr>
<td>$o_2$</td>
<td>$- \times - \times -$</td>
<td>$- \times 6 \times 80$</td>
<td>$- \times 6 \times 120$</td>
</tr>
<tr>
<td>$o_3$</td>
<td>$- \times - \times -$</td>
<td>$- \times 6 \times 120$</td>
<td>$- \times 720 \times 20 \times 140$</td>
</tr>
<tr>
<td>$s$</td>
<td>$240 \times 120 \times 2$</td>
<td>$60 \times 50 \times 200$</td>
<td>$720 \times 1440 \times 130$</td>
</tr>
<tr>
<td>$n_o$</td>
<td>45</td>
<td>2</td>
<td>15, 15, 15, 3</td>
</tr>
</tbody>
</table>

The experimental results confirm our algorithm. Since the roadmap in Group II and Group III is complicated (5000 and 3000 vertices respectively as shown in Table II), the A* takes more time and results in an average local planning cost of 0.101s and 0.144s. The local planner takes little time ($0.007$s) on the 100-vertex roadmap of Group I.

Time cost of the planner in Group III becomes higher than 0.15s. This is mainly caused by three factors. The first one is that each new sampled point of the global planner has to do an extra test of self collision. The second factor is the crowded obstacles, and the third one is the large dimension of these scenarios. Although the environment is unpredictable and relatively large, the planner is still sufficient to be employed in real-time (about 0.18s per planning).

The last row $c_{\text{replan}}$ in Table III denotes the average times of replanning invoked during each execution (the whole procedure that a robot moves from initial configuration to goal configuration). $c_{\text{replan}}$ reflects the robustness of our planner. For instance, even if a planner is fast enough for anytime replanning, it may still fail due to redundant replannings or traps of local minima. In the worst case, $c_{\text{replan}}$ goes infinite that the robot cannot find any of the feasible paths. In our experiments, the randomize DRM planner could return the paths in limited replanning times (average 62.50s in the worst case) indicating its completeness.

### Table II

<table>
<thead>
<tr>
<th>Settings</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\delta_q$</td>
<td>30, 120</td>
<td>15, 60</td>
<td>120, 120</td>
</tr>
<tr>
<td>$l_{\text{size}}$</td>
<td>$60 \times 60$</td>
<td>$25 \times 25 \times 25$</td>
<td>$240 \times 240 \times 136$</td>
</tr>
<tr>
<td>$l_{\text{verts}}$</td>
<td>100</td>
<td>5000</td>
<td>3000</td>
</tr>
<tr>
<td>$l_{\text{mapping}}$</td>
<td>0.44MB</td>
<td>63.72MB</td>
<td>25.51MB</td>
</tr>
<tr>
<td>$\delta_q$</td>
<td>20, 90</td>
<td>15, 60</td>
<td>120, 120</td>
</tr>
</tbody>
</table>

### Table III

<table>
<thead>
<tr>
<th>-</th>
<th>Group I</th>
<th>Group II</th>
<th>Group III</th>
</tr>
</thead>
<tbody>
<tr>
<td>$l_{\text{ave}}$</td>
<td>0.007</td>
<td>0.101</td>
<td>0.144</td>
</tr>
<tr>
<td>$l_{\text{min}}$</td>
<td>0.002</td>
<td>0.006</td>
<td>0.014</td>
</tr>
<tr>
<td>$l_{\text{max}}$</td>
<td>0.011</td>
<td>0.233</td>
<td>0.299</td>
</tr>
<tr>
<td>$g_{\text{ave}}$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.039</td>
</tr>
<tr>
<td>$g_{\text{min}}$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.005</td>
</tr>
<tr>
<td>$g_{\text{max}}$</td>
<td>0.0</td>
<td>0.0</td>
<td>0.071</td>
</tr>
<tr>
<td>$c_{\text{replan}}$</td>
<td>0.007</td>
<td>0.101</td>
<td>0.184</td>
</tr>
<tr>
<td>$c_{\text{ave}}$</td>
<td>47.36</td>
<td>19.96</td>
<td>62.50</td>
</tr>
</tbody>
</table>

Table III shows the result of our approach by comparing the three groups together. In a low dimensional or moderately crowded environment (Group I and Group II), the global planner in our approach can rapidly generate a high-level guide path ($g_{f} < 0.001s$ and shown as 0.0 in Table III). Even in a high dimensional and drastically crowded environment (Group III), the global planner in our approach is able to return a path in less than 0.1s.

As explicated previously, the computational complexity of our local planner mainly depends on the A* search algorithm. Since the roadmap in Group II and Group III is complicated (5000 and 3000 vertices respectively as shown in Table II), the A* takes more time and results in an average local planning cost of 0.101s and 0.144s. The local planner takes little time ($0.007$s) on the 100-vertex roadmap of Group I.

Figure 7. A detail view of Group III

Table IV shows results of different parameters applied to third experimental group (Fig.7 shows a detail view). In the worst cases, the planner degenerates into a raw RRT planner (first row) or degenerates into a raw DRM planner (last row).

Parameter selection seeks the balance between the step length and local size. Note that $c_{\text{replan}}$ is shown by $c_{\text{min}}, c_{\text{max}}$ and $c_{\text{ave}}$ in detail.

We also carried out experiments to compare our approach with a collaborative RRT-RRT planner that employs global-RRT as the subgoal generator while employs localRRT as the inner planner for comparison. See Table V.

Although local RRT strategy reduces $c_{\text{replan}},$ RRT-RRT is not a competent planner. In the worst case of a local RRT, the planner takes more than 2.0s to generate a feasible path and it is not qualified for anytime invoking.

In these experiments we can see that even in the most complicated scenario, the cost of a global planner is still less than 0.10 seconds. At the same time, the cost of a local planner does not increase much owing to the prebuilt mappings of DRM. The experimental results confirm our
assumption that the collaboration of subgoal generator and inner replanner is mighty. The subgoal based planner can be employed in relatively large scenarios with unpredictable obstacles at anytime.

VII. CONCLUSIONS

In this paper a novel path planning algorithm is proposed aiming at generating a collision free path for mobile agents in unpredictable environments. The planner is composed of a subgoal generator and an inner replanner based on the idea of RRT and DRM respectively. Potential field based collaboration between the multiple shot and single shot primitives from RRT and DRM helps to generate subgoals dynamically and plan paths rapidly. Experimental results and analysis show that our method can not only perform path planning rapidly while avoiding unpredictable obstacles but also be competent for relatively large scenario scales.

ACKNOWLEDGEMENTS

This work is supported by National Natural Science Foundation of China (NSFC, No.60875050, 60675025), National High Technology Research and Development Program of China (863 Program, No.2006AA04Z247), Shenzhen Scientific and Technological Plan and Basic Research program (No.JC200903160369A), Natural Science Foundation of Guangdong(No.9151806001000025).

REFERENCES


