

# A Randomized Roadmap Method for Path and Manipulation Planning\*

NANCY M. AMATO  
amato@cs.tamu.edu

YAN WU  
yanwu@cs.tamu.edu

Department of Computer Science, Texas A&M University  
College Station, TX 77843-3112

## Abstract

*This paper presents a new randomized roadmap method for motion planning for many dof robots that can be used to obtain high quality roadmaps even when C-space is crowded. The main novelty in our approach is that roadmap candidate points are chosen on C-obstacle surfaces. As a consequence, the roadmap is likely to contain difficult paths, such as those traversing long, narrow passages in C-space. The approach can be used for both collision-free path planning and for manipulation planning of contact tasks. Experimental results with a planar articulated 6 dof robot show that, after preprocessing, difficult path planning operations can often be carried out in less than a second.*

## 1 Introduction

Automatic motion planning has application in many areas such as robotics, virtual reality systems, and computer-aided design. Although many different motion planning methods have been proposed, most are not used in practice since they are computationally infeasible except for some restricted cases, e.g., when the robot has very few degrees of freedom (dof) [7, 9]. In particular, it has become apparent that with existing technology the enormous complexity of the problem can only be overcome through the use of *probabilistic* or *heuristic* methods (as opposed to *complete* methods which are guaranteed to find a solution or determine that none exists).

Several promising heuristics for path planning have been proposed, of which we mention a few here. The Randomized Path Planner (RPP) of Barraquand and Latombe [2] is a potential field method that uses random walks to attempt to escape local minima. Although this method has been shown to work well for many dof robots, there exist simple situations in which it performs poorly (i.e., does not find a solution) [3, 8]. Researchers have proposed various potential functions (e.g., [1]) and other techniques,

such as learning [6], for escaping local minima. In general, potential field methods can be quite effective when the configuration space (C-space) is relatively uncluttered. However, they have not been as successful for planning in crowded C-space.

Independently, Kavraki and Latombe [8] and Overmars and Svestka [10, 11], proposed similar path planning methods which use randomization during preprocessing to construct a graph in C-space (often called a *roadmap* [9]) whose vertices correspond to collision-free configurations of the robot, and in which two vertices are connected by an edge if a path between the two corresponding configurations can be found by a simple, local, deterministic planner. Briefly, roadmap candidates are selected according to a uniform distribution over C-space and those found to be collision-free are retained. Next, connections between candidate pairs are attempted with a simple planner, and then ‘difficult’ regions of C-space are identified and the roadmap is enhanced in these regions with additional roadmap nodes. Planning involves connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. Planning can be done very fast in many situations. However, long, narrow passages between C-obstacles might be difficult to find. In [11], it is discussed how this approach can be used in ‘single shot’ mode, i.e., without constructing the entire roadmap first during preprocessing. Earlier, Donald [5] proposed the related idea of using free points drawn from a coarse grid in C-space to specify promising subgoals for motion planning.

### 1.1 Our Results

In this paper, we present a new randomized roadmap method for motion planning for many dof robots. The general approach follows traditional roadmap methods: during preprocessing a graph, or roadmap, is built in C-space, and planning consists of connecting the initial and goal configurations to the roadmap, and then finding a path in the roadmap between these two connection points. The main novelty in our approach is a new method for generating

---

\* This research supported in part by NSF Grant IRI-9304734.

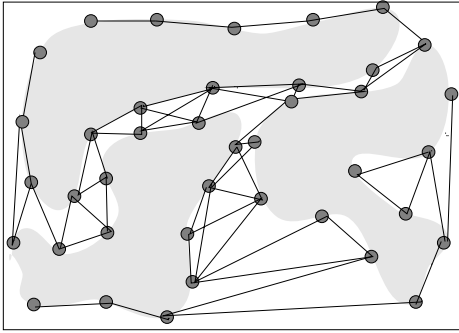


Figure 1: A roadmap in C-space that might be obtained when candidate nodes are uniformly distributed on constraint surfaces.

roadmap candidate points. In particular, we attempt to generate candidate points uniformly distributed on the surface of each C-obstacle. (See Figure 1.) Using this approach, high quality roadmaps can be obtained even when C-space is crowded. Experimental results with a planar articulated 6 dof robot show that, after preprocessing, difficult path planning operations can often be carried out in less than a second.

The approach extends fairly easily to dynamic environments. To update the roadmap when an object in the workspace moves we first remove current roadmap nodes in conflict with the new position, and then generate nodes reflecting the new position and connect them to the existing roadmap.

Our approach can be applied to some important situations that have so far not been satisfactorily solved by heuristic methods:

- Paths through long, narrow passages in a crowded C-space can be found since they are likely to appear in the roadmap.
- The method can be used for manipulation planning of contact tasks since the roadmap can be built on the constraint surfaces.

The previous approaches most closely related to ours are the path planning methods of Kavraki and Latombe [8] and Overmars and Svestka [10, 11] mentioned above. In fact, in [11] the authors describe a technique they call geometric node adding in which roadmap nodes are generated from robot configurations near obstacle boundaries, which is very similar in concept to the idea of generating nodes on C-obstacle boundaries. Moreover, they observe that this technique yields better roadmaps than uniform node generation. However, they state that geometric node adding (as they describe it) cannot be applied to articulated robots. Thus, a contribution of this paper is to provide methods for generating points on C-obstacle surfaces for general many dof robots. The randomized methods we propose

for finding these roadmap candidate nodes are more costly than in the techniques [8, 11] where candidate points are randomly selected according to a uniform distribution in C-space. However, our experimental results show that this additional computation in the node generation phase is negligible in comparison to the total preprocessing time, which in all the methods is dominated by the roadmap connection phase. Finally, in some sense, one can view our work as extending the strategy of [8, 11] to contact tasks.

We describe how the roadmap is constructed in Section 2, and how it is used for planning in Section 3. Implementation details and experimental results are presented in Sections 4 and 5, respectively. Future extensions of this work are discussed in Section 6.

## 2 Building the Roadmap

The tasks required for building the roadmap are generating the roadmap candidate nodes and connecting the candidates to form the roadmap. The description of these tasks below is for a general many dof robot.

### 2.1 Generating Candidate Nodes

Let  $d$  be the number of degrees of freedom of the robot, and let  $S = \{s_i | 1 \leq i \leq n\}$  be the objects in the workspace. Without loss of generality, assume that C-space has dimension  $d$ , and let  $S' = \{s'_i | 1 \leq i \leq n\}$  denote the set of C-objects corresponding to the objects in the workspace.

In this phase, we generate a set  $V$  of candidate roadmap nodes, each of which corresponds to a point in C-space. The general strategy of the node generation process is to construct a set  $V_i$  of candidate nodes for each object  $s_i$  such that each  $p \in V_i$  lies on the surface of  $s'_i$  and is not contained in the interior of any  $s' \in S'$ . The set of roadmap candidate nodes is the union of the candidate sets computed for each object, i.e.,  $V = \cup_i V_i$ .

We now consider how to compute the candidate set for an object. To obtain a high quality roadmap, we would like the nodes to be uniformly distributed (according to some metric in C-space) on the constraint surfaces that delineate the robot's free space. To simplify the exposition, in the following we assume the metric is the Euclidean distance in C-space; this and some other possible metrics are mentioned in Section 4. The easiest way to generate a candidate set for an object  $s \in S$  is to first compute a set of points uniformly distributed on the surface of  $s'$ , and then discard those points found to be internal to any other C-object. The process is sketched below for  $s_i \in S$ .

1. Generate  $m_i$  points  $p_1, p_2, \dots, p_{m_i}$  (approximately) uniformly distributed on the surface of  $s'_i$  and place these points in  $V_i$ .

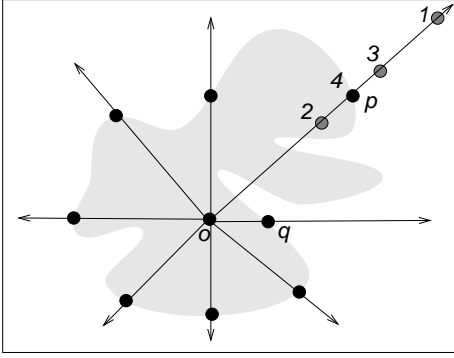


Figure 2: Generating points uniformly on a C-object in two-dimensional C-space with  $m = 8$ .

2. Remove all points from  $V_i$  that lie in the interior of some  $s'_j \in S'$ ,  $1 \leq j \leq n$ .

Briefly, the points in Step 1 are identified by binary searches which perform collision detection checks at each step. Ideally, the number  $m_i$  of points generated in Step 1 will depend upon the size and shape of  $s'_i$ , and thus may vary from C-object to C-object. In Step 2, collision detection is used to determine which points are to be removed from  $V_i$ . We next describe the point generation process in more detail. Collision detection is discussed with other implementation issues in Section 4.

### 2.1.1 Finding Points on C-objects

We now consider how to generate  $m$  points on the surface of a C-obstacle  $s' \in S'$ . For now, assume that we know the number of points we wish to obtain. Ideally, as mentioned above, these  $m$  points should be uniformly distributed on the surface of  $s'$ . However, since we wish to avoid the costly computation of the constraint surfaces, we propose using the heuristic method outlined below to generate the points.

1. Determine a point  $o$  (the origin) inside  $s'$ .
2. Select  $m$  rays with origin  $o$  and directions uniformly distributed in C-space.
3. For each ray identified in Step 2, use binary search to determine a point on the boundary of  $s'$  that lies on that ray.

An example of the method for two-dimensional C-space and  $m = 8$  is shown in Figure 2.

In Step 1, an internal point  $o$  can be determined by selecting any configuration in which the robot is known to collide with the object  $s$ . This is trivially done for a free-flying robot, and requires only slightly more effort otherwise. Of course, to obtain as uniform a distribution

as possible, one would like to select the origin somewhere near the center of the C-object. A simple heuristic is to select a configuration in which a central point of the object (e.g., an average of the coordinates of the object's vertices) coincides with a central point of the robot.

The rays in Step 2 can be selected according to a simple, regular partition of C-space. For example, in Figure 2, since  $m = 8$  and  $d = 2$ , the available 360 degrees are divided into eight equal regions.

Consider a ray  $r$  identified in Step 2. The binary search in Step 3 proceeds by continually narrowing the region on  $r$  known to contain a surface point of  $s'$ . Initially, we know that the origin  $o$  is internal to  $s'$ , and assume that we also know a point external to  $s'$ , e.g., the point labeled 1 in Figure 2. Next, the segment determined by these internal and external points is divided at its midpoint, e.g., the point labeled 2 in the figure. The search then proceeds recursively in the sub-segment which has one internal and one external endpoint. The process terminates when a point on the boundary of  $s'$  is found and/or the minimum step size is reached, e.g., the point labeled 4 in the figure. Before beginning the binary search we must find a point on  $r$  known to be external to  $s'$ . This could be done, for example, by choosing some point known to be at the edge of the relevant portion of C-space, or by iteratively testing points of increasing distance from the origin. Note that it is possible no external point is found, either because  $r$  may not pierce  $s'$ , or because our search for such a point is not successful.

**Potential problems and optimizations.** There are several situations in which the point generation scheme described above may not yield a good distribution on the surface of the C-object.

Clearly, the shape of the C-object and the selected origin both have a great influence on the quality of the resulting distribution. In particular, a distribution close to uniform can only be obtained if the C-object is roughly spherically shaped and the origin lies near its center. For example, if the C-object is long and narrow, then no choice of origin will yield a uniform distribution. In particular, the surface points that are farthest from the origin will have a much larger distance from their neighbors than those that are close to the origin. An optimization that can be applied in this case is to generate more points on the surface between points that are considered too far apart. An example is shown in Figure 3. Another heuristic is to compare the surface normal of a point with those of its neighbors: if they are very different, then the constraint surface must curve between them and we should generate more points.

Another difficulty arises if the C-object boundary has 'folds', i.e., a ray from the origin may intersect the boundary multiple times. In this case, one cannot determine if a good distribution has been obtained by inspecting the Eu-

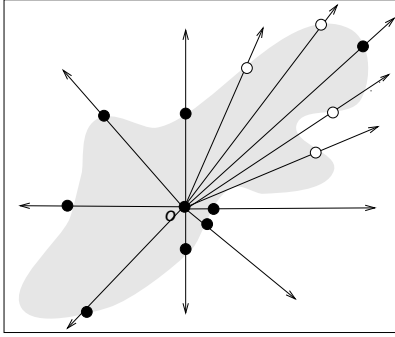


Figure 3: To get a better distribution on ‘oblong’ C-objects, additional points (lightly shaded) are generated near points with large nearest neighbor distance.

clidean distances between neighboring nodes. For example, in Figure 2, although the Euclidean distances between neighboring nodes are fairly uniform, the boundary distance between  $p$  and  $q$  is much larger than between other neighboring nodes. Unfortunately, this situation is difficult to identify without computing the C-object, which we hope to avoid. The heuristic mentioned above of comparing the surface normals of neighboring points can be used to identify some of these situations. Finally, even if one knows a priori that the boundary has ‘folds’, it is not clear how this information can be used to obtain a better distribution without performing a significant amount of computation.

## 2.2 Connecting Roadmap Candidates

We now consider how to connect the candidate nodes  $V = \cup_i V_i$  to create the roadmap. The basic idea is to use a simple, fast, local planner to connect pairs of roadmap candidate nodes. To save space, the paths found in this stage will not be recorded since they can be regenerated quickly. After the connections are made, the connected components in the roadmap are identified, e.g., by depth-first search.

Ideally, the roadmap will include paths through all corridors in C-space. The degree to which this goal can be met depends upon a number of factors: the number and distribution of the candidate nodes, the effectiveness of the simple planner, and the number of connections attempted for each candidate node. Thus, a trade-off exists between the quality of the resulting roadmap and the resources (computation and space) one is willing to invest in building it. Also, the amount of preprocessing required to achieve a given degree of connectivity will depend greatly on the complexity of the C-objects and the robot’s free-space.

Clearly, the method used to determine adjacencies in the roadmap will depend upon the intended use of the roadmap. In particular, different planners will be needed for path and

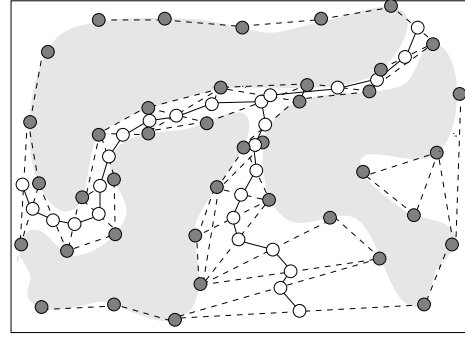


Figure 4: Improving the roadmap quality for path planning by selecting new roadmap nodes (lightly shaded) in C-space corridors.

manipulation planning. However, in all cases the planners should be simple and the paths easy to reconstruct. In addition, as outlined below, the connection strategy (i.e., determining which connections to attempt) may vary from application to application. In the following,  $k$  is a parameter, and distances are measured according to the chosen metric in C-space.

Many different connection strategies could be used in path planning applications. For example, the method used in [8] is to try to connect each node  $v \in V$  to its  $k$  nearest neighbors in  $V$ . A strategy that could improve the chance of finding connections across wide C-space corridors is as follows. For each node  $v \in V_i$ , attempt to connect it to its  $k$  nearest neighbors in  $V_j \in V$ , for  $1 \leq j \leq n$ . Note that both of these strategies require a non-trivial amount of computation to determine the  $k$  closest nodes. Thus, it might be desirable to use some heuristic method to identify the ‘close’ nodes.

The connection strategies for manipulation planning are more constrained since planning is restricted to the constraint surfaces. In this case a simple approach is to attempt to connect each node  $v \in V_i$  to its  $k_1$  nearest neighbors in  $V_i$  and also to its  $k_2$  nearest neighbors in  $V - V_i$ ; usually one would select  $k_1 > k_2$ .

**Possible optimizations.** Since the roadmap nodes are located on constraint surfaces (i.e., points of contact between the robot and an object), the paths encoded in the roadmap will ‘skip’ along the surfaces of C-objects and will not be very desirable for many path planning applications. In the planning phase, smoothing techniques can be used to improve the paths. We can also try to improve the quality of the paths in the roadmap by generating additional roadmap nodes near the center of C-space corridors. This could be done, for example, by adding an additional node near the midpoint of the path found by the simple planner when connecting two roadmap nodes (lying on different C-objects). Next, connections would be attempted between pairs of

these new ‘corridor’ nodes (see Figure 4).

Another potential optimization is to adaptively determine the number of roadmap candidate nodes to generate and the number of connections to attempt. For example, one could begin by attempting a small number of connections for each node. Then, if the resulting roadmap is not sufficiently connected, additional connections could be attempted. This process could be iterated until the roadmap is deemed to be sufficiently connected or until no further improvement is obtained.

### 3 Planning

Planning is carried out as in any roadmap method: we attempt to connect the nodes  $x_1$  and  $x_2$ , representing the start and goal configurations, respectively, to the same connected component of the roadmap, and then find a path in the roadmap between these two connection points. The following approach, proposed in [8], is well suited for our roadmap. First, the simple planner is used to try to connect the start and goal nodes to the roadmap; connections are attempted between  $x_i$  and the  $k$  closest roadmap nodes. If no connection is made for  $x_i$ , then we execute a random walk and try to connect the end node of the random walk to the roadmap. This can be repeated a few times if necessary. If we still can’t connect both nodes to the same connected component of the roadmap, then we declare failure. After both connections are made, we find a path in the roadmap between the two connection points using breadth-first search. Recall that we must regenerate the paths between adjacent roadmap nodes since they are not stored with the roadmap. Finally, smoothing techniques can be applied to improve the resulting path.

In many situations, some paths may be preferred to other paths. This information could be encoded in the roadmap by giving edges on desired (undesired) paths lower (higher) weights, and then using a shortest path computation to find the path between the start and goal connection points. Other methods for favoring certain paths have been proposed that could potentially be integrated into the roadmap (e.g., [4]).

### 4 Implementation Details

We implemented a path planner for a fixed-base segmented manipulator in a two-dimensional workspace. Generally, whenever there was a choice between implementation options, we chose the simplest method, which was often also the least efficient. This strategy was taken in order to produce a working prototype quickly and test the general concept of our approach. The descriptions below

are for a planar articulated robot with  $J$  joints; in our experiments  $J \geq 6$ .

**C-Space distance metric.** We used a simple Euclidean distance metric in C-space (also adopted in [8]). Let  $j_i(x)$  denote the position of the  $i$ th joint when the robot is in configuration  $x$ ,  $1 \leq i \leq J$ . The distance  $d(x, y)$  between configurations  $x$  and  $y$  is

$$d(x, y) = \left( \sum_{i=1}^J (d_i(x, y))^2 \right)^{1/2}.$$

where  $d_i(x, y)$  is the Euclidean distance between  $j_i(x)$  and  $j_i(y)$ . Our implementation uses this metric in both the roadmap connection and the planning phases to select ‘nearby’ candidate nodes. Although the above metric is simple and intuitive, in some cases a more sophisticated metric may be desirable. For example, one might want to assign greater (lesser) weight to joints controlling longer (shorter) links of the robot. Another possibility is to assign larger weights to the joints closer to the base of the robot.

**Number of candidate nodes.** Recall that we attempt to generate a set  $V_i$  of  $m_i$  roadmap candidate nodes for each obstacle  $s_i$  in the workspace. As previously mentioned, the number of candidate nodes that should be generated depends on both the amount of preprocessing one is willing to do and on the geometry of the C-obstacle. For simplicity, we attempted to generate the same number of candidate nodes for each C-obstacle. We did this by equally dividing the allowable range of each joint angle into  $c$  angles, and then considering all possible combinations. For example, in our experiments,  $c = 4$  and we attempt to generate about  $4^J$  candidate nodes for each C-obstacle.

**Collision detection.** The dominant operation in the creation of the roadmap is collision detection: it is heavily used both in the node generation and in the roadmap connection phases. Thus, its efficiency is of vital importance to the overall efficiency of the method. For simplicity, in our implementation, collision detection was implemented by checking each link of the robot with each obstacle, and also with the other links (for self-collision). For the two-dimensional workspace, better results might have been obtained if we had used C-space bitmaps for each link of the robot as was done in [8]. For a three dimensional workspace or complex objects it is not clear what method is best.

**Interconnection strategy.** In the interconnection phase, we would like to connect all the roadmap candidate nodes into a single connected component. Clearly, the interconnection strategy chosen can greatly affect how close we come to this goal. Recall that we have a set  $V_i$  of roadmap candidate nodes for each obstacle  $s_i$  in the workspace. The basic approach we have taken is to attempt to connect each

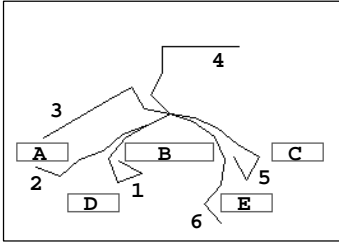


Figure 5: E1: five objects.

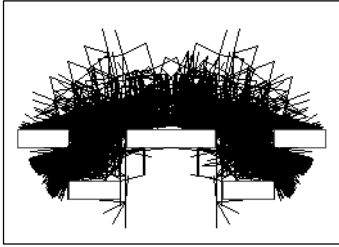


Figure 6: The single connected component for E1.

node  $v \in V_i$  to its  $k$  nearest neighbors in  $V_j \in V$ , for  $1 \leq j \leq n$ , where  $k$  is a constant. Clearly, the larger  $k$ , the greater chance of connection, but also the more computation involved; for our experiments  $k = 10$ .

Note also that this requires a non-trivial amount of computation to determine the  $k$  closest nodes for *every* candidate node in *every* candidate set. In an attempt to reduce the cost of finding the closest nodes, we tried a heuristic method to identify ‘close’ nodes. Briefly, we partitioned  $V_j$  into  $k$  equal-sized subsets, and found the closest node in each subset; in this way we are guaranteed to find the closest node, and will also probably get some other close nodes.

**Local planner.** The local planner is used in both the roadmap connection phase of the preprocessing and in the planning. Our experiments show that the efficiency of the local planner is of crucial importance since the roadmap connection phase required orders of magnitude more time than any other part of the preprocessing. On the other hand, the local planner must also be deterministic since we do not want to store the paths connecting nodes in the roadmap. Thus, great care should be taken in selecting this planner and different methods will be needed in different situations.

We used the following simple planner. Let  $x$  and  $y$  be two configurations we wish to connect. The local planner we tried was to move directly along the straight line segment connecting  $x$  and  $y$  in C-space, performing collision-detection checks at uniform intervals on the line segment. The distance between subsequent points checked is determined by the resolution needed for the current problem instance.

Environment	Generation		Connection time	Roadmap Structure		
	#nodes	time		#cc	sizes	time
E1	21342	33	3880	1	21342	109
E2	14017	10	786	5	14011, 2(2), 1(2)	39
E3	9046	9	405	6	9041, 1(5)	14
E4	7360	9	285	22	7339, 1(21)	8
E5	6969	9	287	1	6969	7

Figure 7: Preprocessing times (seconds) and statistics for the five environments.

Environment	Configuration Number					
	1	2	3	4	5	6
E1	Fail	.0028	.0027	.0027	.0027	.0163
E2	.0016	.0012	.0104	.0012	–	–
E3	.0015	.0012	.0056	.0011	–	–
E4	.0019	.0012	.0047	.0578	–	–
E5	.0014	.0012	.0747	.1960	–	–

Figure 8: Times (seconds) to connect configurations to the largest connected component using the straight-line planner. The failure to connect configuration #1 in E1 was declared in .1948 seconds.

## 5 Experimental Results

We implemented a path planner for a planar articulated manipulator in a two-dimensional workspace. The code was written in C. All measurements were taken on a SGI Indy workstation running IRIX Version 5.3; the CPU was a 100 Mhz MIPS R4600 with 32 MB of RAM. All I/O requests were serviced remotely using NFS. Clearly, better results would be obtained by a faster machine with more memory and a local disk.

In the following, we analyze the performance of the method in a few environments. In all cases, we used a fixed-base articulated robot with 6 dof (6 links). The various environments, and some representative configurations of the robot, are shown in Figures 5, 9, 10, 11, and 12.

The preprocessing times (seconds) and other statistics related to the roadmap construction are shown in the table in Figure 7. In this table, the second column lists the number of configurations generated as roadmap candidate nodes, the fifth column lists the number of connected components in the roadmap after the interconnection process (using only the straight-line in C-space planner), and the sixth column lists the number of nodes in each connected component (if needed, the number of connected components of a certain size is indicated in parenthesis). Note that the roadmap size is influenced by the number of obstacles in the workspace since a set of roadmap nodes is generated for each obstacle, i.e., the size of the network is related to the complexity of the environment. We used the heuristic interconnection strategy mentioned in Section 4 of picking  $k = 10$  candidates from each set  $V_i$  for every roadmap node. Note that even with this simple strategy, in

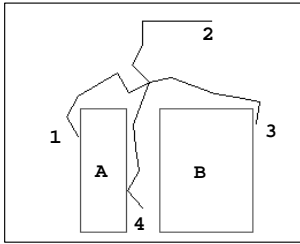


Figure 9: E2: two objects with a passage.

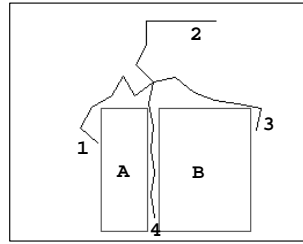


Figure 10: E3: two objects with narrow passage, base 1.

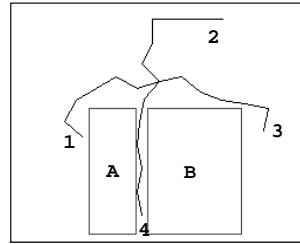


Figure 11: E4: two objects with narrow passage, base 2.

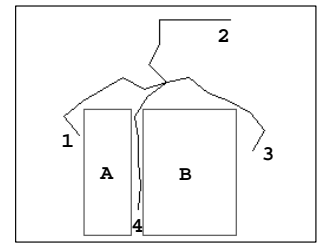


Figure 12: E5: two objects with narrow passage, base 3.

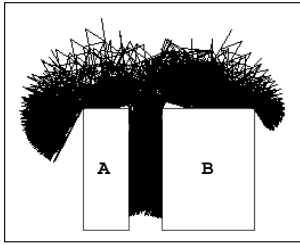


Figure 13: The largest connected component for E2.

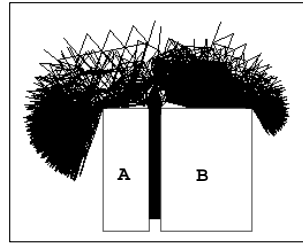


Figure 14: The largest connected component for E3.

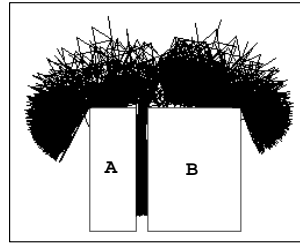


Figure 15: The largest connected component for E4.

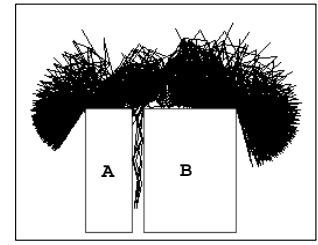


Figure 16: The single connected component for E5.

all cases almost all the nodes were contained in one large connected component. Finally, it is important to note that by far the most expensive phase of the preprocessing is roadmap interconnection, and thus fast local planners and collision detection routines are of great importance.

In Figures 6, 13, 14, 15, and 16 all the configurations in the largest connected component for that environment are drawn one on top of another in the workspace. The density of these figures indicates that most reachable regions of the workspace are well represented in the roadmap (recall that all roadmap nodes correspond to configurations in which the robot is in contact with at least one obstacle.)

The times (seconds) required to connect the configurations shown in Figures 5, 9, 10, 11, and 12 to the roadmap are shown in the table in Figure 8. The times include both the time of the simple planner and the time to search the roadmap for candidates for connection (again, using the heuristic method of Section 4). The only planner used in this stage was the straight-line in C-space planner, i.e., if the straight-line planner failed we did *not* try the approach mentioned in Section 3 of executing a random walk and then attempting to connect the terminus of the random walk to the roadmap. Even so, the only instance in which we failed to connect to the roadmap was configuration #1 in environment E1; in this case, failure was declared in .1948 seconds.

The time for path planning between any two configurations would add the time to find the path in the roadmap between the connection points to the connection times reported above. For the environments considered here, the search in the roadmap for these paths is very fast (e.g., less

than a second).

Below, we discuss the various environments in more detail.

**E1:** This environment contains five obstacles, and the base of the robot is fixed above the center of obstacle B. The roadmap for this environment is the largest of our examples and consequently also takes the most time to produce. Recall that in this simple implementation we attempted to generate the same number of roadmap candidate nodes on each C-obstacle. In this case we successfully generated 8124 nodes for obstacle A, 3832 nodes for obstacle B, 8124 nodes for obstacle C, 630 nodes for obstacle D, and 632 nodes for obstacle E. Fewer valid nodes are generated for obstacles D and E since they are farther from the base and most of the configurations in contact with these obstacles will collide with obstacle B. It is also worth noting that since the environment is symmetric with respect to the robot, essentially the same number of nodes are generated for obstacles A and C, and for D and E.

The only case in which we failed to connect a configuration to the roadmap was configuration #1 in E1 which lies in a region that is not well-represented in our roadmap. It is possible that a connection might have been achieved had we tried executed one or more random walks (see Section 3).

**E2:** This environment has a long passage between two obstacles, and the base of the robot is fixed above this passage. This is a difficult environment for most planning methods since the workspace passage corresponds to a long passage in C-space. As can be seen, our method handles this situation easily, and connects all four configurations to

the roadmap in a fraction of a second. Also note that the preprocessing time is less than that for E1. This is mainly because the roadmap contains fewer nodes since there are only two obstacles rather than five; we generate 7416 nodes for obstacle A and 6601 nodes for obstacle B.

**E3-E5:** These environments are similar to E2 except that the width of the passage is considerably narrower, which makes it even more difficult for planning since the corresponding passage in C-space is also narrower. In E3, the base of the robot is fixed above the passage, and in E4 and E5 the base is progressively translated to the right. Again, our method handles these situations very well, connecting all configurations to the roadmap with the simple straight-line in C-space planner in less than a second. Note that these roadmaps contain fewer nodes than the roadmap for E2. The reason for this is that, due to the narrower passage, fewer collision-free configurations are found in the node generation phase. In particular, we generate 5045 nodes for A and 4001 nodes for B in E3, 3730 nodes for A and 3630 nodes for B in E4, and 3584 nodes for A and 3385 nodes for B in E5. Note that as we generate fewer valid nodes the roadmap connection time, which is the most expensive phase of the preprocessing, is reduced. Thus, as C-space gets more cluttered, the preprocessing costs actually decrease.

A trend shown in this set of environments is that as the base of the robot is translated away from the passage, fewer collision-free configurations are found in the passage. This in turn makes connection between a configuration in the passage and the roadmap more difficult, as is evidenced in the differences in the times needed to connect configuration #4 to the roadmap. In order to improve the chance of connection in cases such as this, one could use one of the optimizations mentioned in Section 2.1.1 during the node generation phase.

## 6 Conclusion

We have described a new randomized roadmap method for motion planning that is applicable for both collision-free path planning and for manipulation planning of contact tasks. To test the concept, we implemented the method for path planning for a segmented robot in a two-dimensional workspace. The method was shown to perform well, even in crowded C-space.

In the future we plan to extend our implementation to manipulation planning for contact tasks, to a three-dimensional workspace, and to dynamic environments (in which obstacles may move). We also plan to investigate how the cost of building the roadmap can be reduced through the use of parallel processing, which is becoming increasingly available in very economical platforms (e.g.,

multiple processors on a single chip, and workstations with several processors). Another interesting problem is to modify the method for 'single-shot' planning, i.e., without first constructing the roadmap in preprocessing, as is done in [10, 11].

## Acknowledgement

We would like to thank the robotics group at Texas A&M, especially Jeff Trinkle, Peter Stiller, and David Stewart, for their suggestions regarding this work. We are also grateful to Lydia Kavraki and Shane Chang for their comments.

## References

- [1] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. Sys., Man, Cybern.*, 22(2):224–241, 1992.
- [2] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *Internat. J. Robot. Res.*, 10(6):628–649, 1991.
- [3] D. J. Challou, M. Gini, and V. Kumar. Parallel search algorithms for robot motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, volume 2, pages 46–51, 1993.
- [4] H. Chang and T. Y. Li. Assembly maintainability study with motion planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 1012–1019, 1995.
- [5] B. R. Donald. A search algorithm for motion planning with six degrees of freedom. *Artif. Intell.*, 31(3):295–353, 1987.
- [6] B. Faverjon and P. Tournassoud. A practical approach to motion-planning for manipulators with many degrees of freedom. In H. Miura and S. Arimoto, editors, *Robotics Research 5*. The MIT Press, 1990.
- [7] Y. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.
- [8] L. Kavraki and J. C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Internat. Conf. Robot. Autom.*, pages 2138–2145, 1994.
- [9] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.
- [10] M. Overmars. A random approach to path planning. Technical Report RUU-CS-92-32, Computer Science, Utrecht University, The Netherlands, 1992.
- [11] M. Overmars and P. Svestka. A probabilistic learning approach to motion planning. In *Proc. Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.