In the *Proceedings of the IEEE Intl. Conf. on Robotics and Automation*, pp. 2138-2139, San Diego, 1994

Randomized Preprocessing of Configuration Space for Fast Path Planning

Lydia Kavraki
kavraki@cs.stanford.edu

Jean-Claude Latombe
latombe@cs.stanford.edu

Robotics Laboratory, Department of Computer Science,
Stanford University, Stanford, CA 94305, USA

Abstract

This paper presents a new approach to path planning for robots with many degrees of freedom (dof) operating in known static environments. The approach consists of a preprocessing and a planning stage. Preprocessing, which is done only once for a given environment, generates a network of randomly, but properly selected, collision-free configurations (nodes). Planning then connects any given initial and final configurations of the robot to two nodes of the network and computes a path through the network between these two nodes. Experiments show that after paying the preprocessing cost (on the order of hundreds of seconds), planning is extremely fast (on the order of a fraction of a second for many difficult examples involving a 10-dof robot). The approach is particularly attractive for many-dof robots which have to perform many successive point-to-point motions in the same environment.

Acknowledgments: This research was funded by ARPA grant N00014-92-J-1809. Early stages of this research benefited from discussions with Jérôme Barraquand.

1 Introduction

We present a new path planning method for robots with many degrees of freedom (dof). We approach the problem by defining a certain preprocessing of the configuration space (C-space), after which many difficult path planning problems can be solved in time of the order of a fraction of a second.[1] The preprocessing itself does not take very long: of the order of a few hundreds of seconds.

During the preprocessing stage a set of collision-free configurations (*nodes*) in the C-space are generated and interconnected into a network using very simple and fast planning techniques applied to pairs of neighboring nodes. The network produced has a large number of nodes (order of thousands). It may contain one or more connected components, depending on the robot's free space and the time spent on preprocessing.

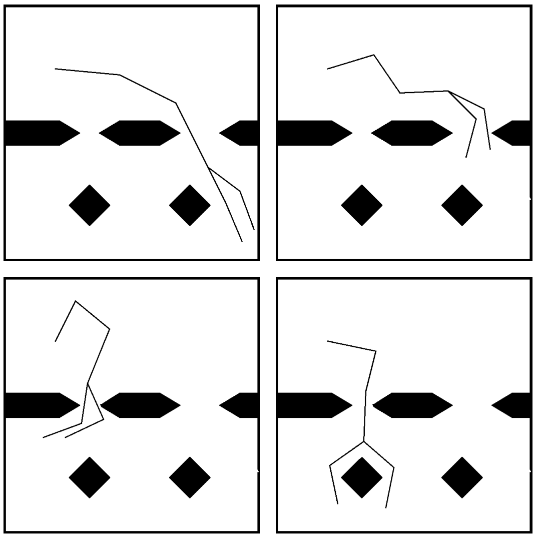


Figure 1: Snapshots along a path of a 10-dof robot

After preprocessing, planning a path between any two configurations is solved by connecting both configurations to some two nodes A and B in the network, and searching the network for a sequence of edges connecting A and B. The resulting path can be improved using any standard smoothing algorithm. The planner fails if it cannot connect any of the two input configurations to the network, or if A and B lie in two different connected components of the network.

We have implemented our method in a program written in C and running on a DEC Alpha workstation. We have conducted series of experiments with the planner. Difficult path planning problems like the one in Fig. 1 were solved in a fraction of a second after a preprocessing time of a few minutes. The robot used in this example has 3 prismatic and 7 revolute dof (see Fig. 2). To the best of our knowledge existing planners

[1] All the running times given in this paper were obtained by running our planner on a DEC Alpha workstation.

would take much longer to solve the same problem, if they succeed at all.

Our approach is particularly suitable for robots with many dof which perform several point-to-point motions in known static environments. Examples of tasks meeting these conditions include inspection and repair in constrained environments (e.g., nuclear plants), point-to-point welding operations to assemble the body of a car, and washing/cleaning airplane fuselages. In such tasks, many dof are needed to achieve some final configuration of the end-effector, while avoiding collisions of the rest of the arm with the complicated environment. Programming such robots is tedious and time consuming. An efficient planner can considerably reduce the programming burden.

Section 2 gives an overview of previous related research. Sections 3 and 4 describe the preprocessing and planning stages of our approach. Section 5 provides implementation details. In Section 6 we discuss a series of experiments conducted with the planner.

2 Relation to Previous Research

Path planning in a known environment has been studied extensively over the last two decades [14]. Recently there has been a renewed interest in developing heuristic, practical path planners. Very efficient path planning methods have been designed for robots with few dof (e.g., see [3, 8, 15]). An independently developed method that bares resemblances with our approach is described in [18]. In this work, a single shot planner is implemented for robots with few dof. The planner generates random via-points for guiding the robot from its initial to its final configuration. Carefully selected connections among the via-points are attempted and the results are retained in form of a graph. The planner finishes when the initial and the final configuration of the robot get connected through the graph. A planner for car-like robots [20] has also been implemented based on the same principles. Recently, the method has been extended to robots with many dof and transformed into a learning approach [19]. A combination of the common ideas with the approach presented here is attempted in [9].

Planners developed for many-dof robots include the planner proposed in [5]. This planner uses a learning scheme to avoid falling in the local minima of the potential field function. The quantity of the stored information however, makes the method impractical when the number of dof is large. Ways of computing potential functions for many-dof robots and randomized search techniques to escape local minima are introduced in [3]. Other potential field methods are described in [2]. Hierarchical cell decomposition techniques for path planning with 6-dof robots are investigated in [7, 13]. A planner based on variational dynamic programming is introduced in [1]. Genetic algorithms have been employed in [17] and use of parallel processing techniques is investigated in [4, 16].

Among the potential field planners, the Randomized Path Planner (RPP) [3] has been applied to robots having 3 to 31 dof and is often very efficient. It has also been used in practice with very good results [6]. However, several cases have been identified where RPP behaves poorly. The example of Fig. 1 is one of them, but there exist simpler ones [4, 21]. More generally, RPP may fail to find a path in reasonable time when:

- the robot's collision-free space in C-space consists of several regions (which we call "traps") connected through narrow passages,
- the boundaries of the attraction basins of the potential's local minima are located within or close to those passages, and
- the initial and final configurations lie in two different traps.

Then the potential function cannot help the planner to find a path between two traps. The search inevitably falls in the local minimum of the current trap. RPP attempts to escape this minimum by performing a series of random walks, but the probability that any of these walks finds its way through a narrow passage is almost zero. One idea to fix this problem is to use several potential functions, hoping that the boundaries of the basins of attraction for one potential will be significantly different from the boundaries for another potential. We have tried this idea and it works well in some cases. In other cases, it seems very difficult to generate an adequate set of potential functions. Furthermore, each failure with one potential takes a significant amount of time, so that the number of potentials that RPP may consider has to be small. The approach presented below tries to identify "difficult" regions in C-space and generate additional configurations in those regions in order to capture well the structure of the free C-space.

3 Preprocessing Stage

The preprocessing stage of our planner consists of the sequence of steps outlined below. In this section our description is for a general many-dof robot.

1. Generation of nodes. Random configurations of the robot (nodes) are generated over the free C-space.

Care is taken to produce a rather uniform distribution. For example, for the robot of Fig. 1, a node is generated by drawing each dof uniformly from its allowed range. After the 10 random choices have been made, the resulting configuration is tested for collision with obstacles and self-collision. We keep it only if it passes these tests. This step is repeated until a prespecified number N of nodes has been computed. We discuss later how the choice of N affects the algorithm. In our examples, N is in the order of a few thousands.

2. Interconnection of the nodes with a simple planner. We now have a set of N nodes. Given some metric in C-space, for each node x, all the other nodes are sorted according to increasing distance from x; a simple and fast planner (see Section 5) tries to connect x to each of the K closest nodes (K is a parameter). Each connection yields an edge of the network. Robot paths computed here are not recorded since they can easily be recovered. The connected components of the resulting network are computed by a straightforward breadth-first search algorithm.

3. Enhancement of "difficult" regions. Typically at the end of step 2 we have a few large components and several small ones. The purpose of the enhancement step is to add more nodes in a way that will facilitate the formation of a large component comprising as many of the nodes as possible and will also cover the more "difficult" (narrow) parts of the C-space. The identification of these difficult parts of the C-space is no simple matter and the heuristic that we propose below clearly goes only a certain distance in this direction. We present it in a general framework so that other heuristics can easily fit in.

For each node x that has been found during the generation step 1, we define a *weight* $w(x)$ which should be large whenever x "is in a difficult region". The weights for all x should add up to 1, as it will become apparent later. Let us also call *expansion* of node x the creation at random of another node y in the free part of the neighborhood of x. The simplest way to do this is to choose each of the parameters that describe the configuration uniformly at random from a small interval centered at the corresponding parameter of x.

The following is then the heuristic *scheme* that we propose. We add a user specified number M of new nodes to our collection. This time instead of choosing them at random as in step 1, we choose a node from among those that step 1 generated with probability

$$Pr(x \text{ is selected}) = w(x),$$

and we expand that node x. If y is the created node we denote by $p(y)$ the node x, that is the node responsible for its creation. We repeat this M times. If the function $w(x)$ adequately identifies the difficult parts of the C-space, our heuristic will tend to fill these more than others.

The essential parameter in our scheme is the function $w(x)$. In our experiments we use the results of step 2 to build $w(x)$. We define the *degree* d_x of a node x as the number of connections that x has with other nodes at the end of step 2, and

$$w(x) = \frac{1}{d_x + 1} \;/\; \sum_{t=1}^{N} \frac{1}{d_t + 1}.$$

We regard this number as a measure of the "difficulty" of the C-space region in which the node lies. Nodes with low degree are in "difficult" parts of the C-space. It is crucial to retain such nodes since they might lie in narrow passages of the C-space and may contribute significantly to producing a network that captures the connectivity of the free C-space.

We have experimented with other choices of $w(x)$ too. One of them is described in detail in [10] and its effect is the expansion of all small components produced at the end of step 2. This expansion (we consider as small any component with less than 1/2 the total number of nodes) can be formulated in the context of our scheme as follows: if S is the number of nodes belonging to small components, we let $w(x) = \frac{1}{S}$ if a node belongs to a small component and $w(x) = 0$ otherwise. Another possible choice of the $w(x)$ is the (normalized) reciprocal of the number of nodes generated in step 1 which fall within a certain distance from the node x. Many other choices are possible. Experiments showed that the degree heuristic of the previous paragraph, works well over a large range of examples. Let us finally point out that if the initial N is large, the choice $w(x) = 1/N$ should produce results that are almost identical to the choice of M random nodes in the manner of step 1.

After all M nodes have been produced (M is typically close to N), we test each node y of them for connection with its parent node $p(y)$. In the case of a successful connection, we record the fact that the new configuration y has been connected with $p(y)$ and thus with all the nodes in the component of $p(y)$; if the connection fails the new node is considered as not belonging to any component yet. Then, each node y of the M produced is tested for connection with the K closest among the $N + M - 1$ other nodes which lie in a different component than y itself. The effect of this addition of M nodes is a larger network, whose connected components are then recomputed. At this stage, components which contain a small fraction (usually less than 0.5% of the total nodes) are discarded.

4. Further reduction of the number of components. In many examples the enhancement of step 3 yields a connected component comprising most of the nodes when $N + M$ is large enough. Of course, this is not the case when the free C-space is not connected. There are also difficult examples where the free C-space is connected but our simple planner failed to achieve some crucial connections.

We attempt a further reduction of the number of components using a more sophisticated planner. For any two components, starting with the largest, we select a pair of nodes, x in the first and y in the second, which are close to each other (according to the C-space metric). In our implementation RPP [3] is called to connect x and y. If it produces a connection, the two connected components are merged into one, and a new pair of components is considered. Instead, if it fails to produce a connection within some short time bound, a different pair of x and y is chosen and a new connection is attempted. In this way we avoid getting stuck at a case that may be hard to solve by RPP, or even impossible (if the two components are not path-connected in free space). The process is repeated a few times for each pair of components. Eventually, if RPP fails to produce a connection, we consider that the components cannot be connected and we retain them as they are. Connection paths computed during that step are recorded, since their recomputation would be relatively expensive.

4 Path Planning Stage

Let x and y be the initial and final configurations of the robot and suppose that a single-component network was produced by preprocessing. We first connect x to a node of the network with the simple planner: we sort the network nodes in increasing distance from x and start with the closest nodes. If all these attempts fail we execute a random walk and try to connect the final configuration of the walk to the network with the simple planner. The length of the random walk can be chosen uniformly in the interval $[1, max\_length]$, where max_length is a constant. The above step can be repeated a few times if necessary. The same procedure is followed for y.

Let A and B be the nodes with which x and y get connected. A breadth-first search of the network constructs a path between A and B. This path is thus the shortest in the number of nodes. The robot paths connecting successive nodes along this path are recomputed, unless they were produced by RPP (see preprocessing step 4). The path between x and y can be smoothed using any standard smoothing technique.

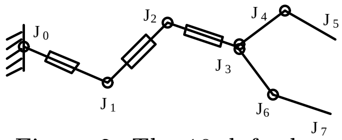


Figure 2: The 10-dof robot

If the network produced by preprocessing has more that one component, we try to connect both x and y to the same component, starting with the largest. If we cannot succeed, the planner declares failure. Since we bound the total amount of time spent in the connection of x and y to the network, we detect failure to connect them in a reasonable time.

5 Implementation Details

We now discuss some implementation details of our planner using the 10-dof robot of Fig. 2 as an example. This robot has 3 prismatic joints and 7 revolute joints located at J_0, J_1, J_2, J_3, J_4, and J_6. Two revolute joints are located at J_3. The description can be generalized easily to any articulated linkage [11].

1. Generation of random configurations. To create a random configuration of the robot we draw each dof uniformly from its allowed range. Then we check the resulting configuration for collision with the obstacles and self collision. Typically a very small percentage of the randomly guessed configurations are collision-free (less than 0.5%). Several optimizations can be applied in this step. For example if the robot has many links, we can draw the dof values in sequence and check for collision as soon as the location of a link gets determined. In our implementation the generation of the random configurations is very fast, so there was no need to improve on it.

2. Distance between two configurations. Let $J_i(x)$, $i = 0, \ldots, 7$ denote the position of the J_i (see Fig. 2), when the robot is at configuration x. We define the distance $d(x, y)$ between any two configurations x and y as

$$d(x,y) = \left(\sum_{i=0}^{7} \| J_i(x) - J_i(y) \|^2 \right)^{1/2}$$

where $\|J_i(x) - J_i(y)\|$ is the Euclidean distance between $J_i(x)$ and $J_i(y)$. The distance above is quick to compute, has an intuitive meaning and can be used for any planar articulated linkage. Other distances are possible. For example, it may be useful to weight the above sum. Larger weights should be assigned to the distances of the J_i's close to the base of the robot, since displacements of the links near the base have a more significant effect on the robot than displacements of the end effector.

3. Simple Path Planner. We have implemented a simple and fast path planner for the robot of Fig. 2. Let x and y be two configurations that we attempt to connect and $J_0, \ldots, J_7$ be the points of the robot as shown in Fig. 2. We simultaneously translate J_1, J_2, J_3, J_5, and J_7 along the straight line in the workspace that connects their workspace position at configuration x to the their workspace position at configuration y. Then we adjust the first prismatic dof and the positions of J_4 and J_6, by computing the inverse kinematics of the robot. If during the motion, the robot collides with an obstacle or with itself, or if a joint reaches a limit, or an adjustment is impossible, the planner fails. It is clear that this planner is weak. However, it is extremely fast and has high chances of success if x and y are close to each other. It tries to reduce the area swept by the robot when moving between two configurations and thus increase the chances of a collision-free path. Other planners are possible. We have observed that the simpler planner proposed performs better than the straight line in C-space, or a planner that changes one dof at a time. Finally we should note the the choice of the distance in C-space and that of the simple planner must be closely coupled: the distance should reflect the chances of the planner to connect two configurations.

4. Choosing N. For many-dof robots, this parameter should be set to at least a few thousands. Increasing N generally reduces the time needed for path planning and improves the quality of the path obtained, but it also increases preprocessing time. If N is set too small, no nodes may be generated in "difficult" regions of C-space, thus there will be no enhancement of these regions during step 3 for preprocessing. N should be determined by experimentation.

5. Choosing K. Preprocessing step 2 attempts to connect each node x to the K closest nodes in the network. On the one hand, K should not be too small, because we want to give our simple planner a good chance to make connections. On the other hand, making it too large increases the running time unnecessarily, since the simple planner cannot connect nodes that are far apart. A few successful connections per node are enough to ensure large connected components. In our experiments we used $K = 20$.

6. Choosing M. At preprocessing step 3 we add M nodes to enhance "difficult" regions of the C-space. Setting M close to N gives good results. We select a node to be expanded with the probability distribution function of step 3. To expand a configuration x we let each dof take a random value in an interval centered around its value at x. We set this interval to about 1/6 of the range of the dof. Also, we decrease this interval as we move towards the base of the robot. If we vary a lot a dof near the base we may create a configuration which is not close to the initial one.

7. Collision checking. Collision checking can be implemented analytically or with a discretized C-space bitmap for each link of the robot. To create a C-space bitmap we assume that each link of the robot is free to translate and rotate and we precompute for each link a three dimensional C-space bitmap that explicitly represents the free subset of the link's C-space (the "0"s) versus the part that gives rise to collision with an obstacle (the "1"s). When testing for collision, the planner tests each link against its C-space bitmap, which is very fast. Each telescopic link of our 10-dof robot can be modeled as two links. This technique is practical only for 2D workspaces, since 3D workspaces would require the generation of six-dimensional bitmaps. We use it in our planner (we discretize each dof to 128 values) and in particular we compute the C-space bitmaps with the use of the Fast Fourier Transform [12]. For self-collision, each link of the robot is tested against the others.

6 Experimental Results

The planner is implemented in C and we used a DEC Alpha workstation (Model Flamingo) running under DEC OSF/1 for our experiments. This machine is rated at 121.5 SPECmark89. In this section, we analyze in depth the performance of our method with a difficult example.

Fig. 3 shows 8 configurations of the robot of Fig. 2 and the environment in which the robot moves. Notice that the gates in its workspace are not of the same width: one is almost three times wider than the other. Path planning problems can be defined by selecting any two of these configurations. We summarize the results of our experiments in two tables. The first, given in Fig. 4(a), is obtained using our enhancement technique. The second, in Fig. 4(b), is given for comparison purposes and contains results obtained without enhancement. The sum of N and M is the same in the corresponding rows of the two tables.

The first column in the tables of Fig. 4 gives the time spent on preprocessing (in seconds) for computing networks with different number of nodes. Smaller networks in these tables are not part of larger net-

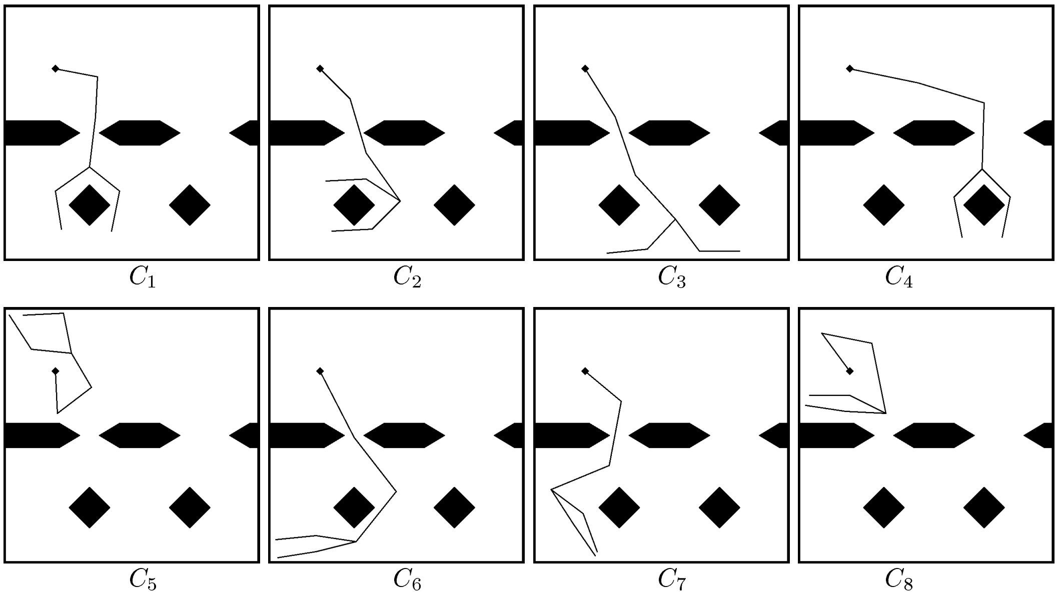


Figure 3: Various configurations of the 10 dof robot

(a)

Prepr. (sec)	Final nodes	N	M	C1	C2	C3	C4	C5	C6	C7	C8
183.81	1432	750	750	0.00	1.73	1.73	0.02	0.02	F	F	0.00
157.68	2805	1500	1500	0.00	0.27	3.17	0.00	0.02	F	1.53	0.12
172.53	3842	2000	2000	0.07	9.17	0.03	0.02	0.03	F	0.02	0.03
288.97	5903	3000	3000	0.02	0.02	0.03	0.02	0.18	8.52	0.02	0.03
483.98	8866	4500	4500	0.05	0.05	0.02	0.02	0.03	0.83	0.02	0.03
595.74	10901	5500	5500	0.02	0.02	0.02	0.03	0.05	1.48	0.03	0.03
785.43	12905	6500	6500	0.03	0.03	0.03	0.03	0.05	6.48	0.03	0.05
882.58	14905	7500	7500	0.05	0.08	0.03	0.03	0.07	0.33	0.03	0.07

(b)

Prepr. (sec)	Final nodes	N	M	C1	C2	C3	C4	C5	C6	C7	C8
73.43	1397	1500	0	F	F	F	0.02	0.02	F	F	0.00
159.13	2847	3000	0	F	F	F	0.02	0.02	F	F	0.00
216.32	3794	4000	0	F	F	F	0.02	0.02	F	F	0.00
342.62	5734	6000	0	F	F	F	0.02	0.03	F	F	0.02
568.44	8795	9000	0	0.02	1.25	0.02	0.02	0.05	F	0.02	0.03
747.82	10790	11000	0	0.02	0.98	0.03	0.02	0.03	5.25	1.82	0.03
891.85	12805	13000	0	0.02	0.03	0.03	0.03	0.05	F	0.02	0.05
1113.47	14691	15000	0	0.03	0.10	0.03	0.03	0.05	11.07	11.22	0.05

Figure 4: Preprocessing and connection to network times (a) with enhancement, (b) without enhancement $M = 0$

works; all networks were produced independently. Preprocessing time includes the generation of N initial network nodes, their interconnection with the simple planner, the addition of M nodes in the "difficult" regions of the C-space, and the reduction of the number of the resulting components using RPP. The bound on the running time of RPP (for each call) was set to 20 seconds. Column 2 shows the number of nodes in the largest component produced and columns 3 and 4 the values of the parameters N and M. In columns 5 through 12 we give the time required to connect configurations $C_1, \ldots, C_8$ of Fig. 3 to the precomputed networks. When this time is 0.00 this means that it took less than 0.01 seconds to connect the configuration to the network, and when it is more than 1 second, this indicates that a few random walks were performed. An 'F' denotes failure to obtain a connection of the configuration to the precomputed network after 35 random walks (their lengths are chosen uniformly in the interval $[100, 15000]$).

Prep. time	N+M	Gene ration	Conne ction	Expan sion	RPP Con.
183.9	750+750	1.5	30.9	15.7	135.9
288.9	3000+3000	5.9	152.7	103.2	26.9
595.7	5500+5500	11.0	312.3	260.3	12.1

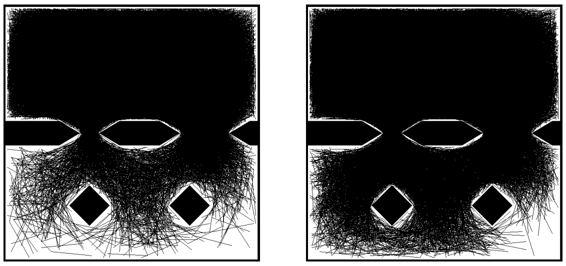
Figure 5: Breakdown of the preprocessing time (in seconds) for some of the networks of Fig. 4(a)

To estimate path planning time between two configurations we add the time needed to connect these to the network to the time required to obtain a path on the network. Typically, less than 0.1 seconds are spent searching the networks considered here. Our planner is not guaranteed to return a path. It fails to find a path between two configurations if it cannot connect them to the same component. But since the time allowed for these connections is bounded, the planner detects failure in a reasonable time.

A breakdown of the preprocessing time for some of the networks of Fig. 4(a) is given in Fig. 5. Notice from the latter figure that when $N+M$ is small, RPP may take long to connect components. This is because a few rather isolated components may be present at the end of enhancement. The time taken by RPP reductions gets smaller when the number of nodes increases.

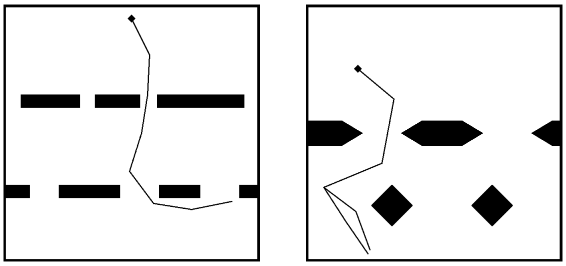
We observe from the table of Fig. 4(a) that if a small network is created, it does not capture very well the structure of the robot's free C-space and attempts to connect configurations $C_1, \ldots, C_8$ to it may fail or take a few seconds. However, when $N+M$ is sufficiently large (in this example when $M+N > 6000$), a large component comprising most of the generated configurations is formed and path planning is in the order of a small fraction of a second, which is quite impressive. Preprocessing itself is in the order of few minutes. In particular, 5 minutes are enough for our example as line 4 of Fig. 4(a) indicates.

Comparison between the corresponding lines of the tables in Fig. 4(a) and Fig. 4(b) shows that, in this case at least, the method does not perform very well when enhancement is omitted. More nodes are required to obtain a network which captures adequately the structure of the free C-space. This is inferred by the number of failures to connect to the network in Fig. 4(b), where no enhancement is done. As a result, more time is needed to create a network to which all $C1, \ldots, C_8$ can be connected in a few seconds. This time is 12 minutes in our example (line 6 in Fig. 4(b)). Still the result is not very stable: line 7 in Fig. 4(b) shows a larger network than the one of line 6, but we failed to connect configuration C_6 to it. Such behavior



(a) (b)

Figure 6: Networks produced in approximately 785 seconds (a) without enhancement, $N = 11500$, $M = 0$ and (b) with enhancement $N = M = 6500$



(a) (b)

Figure 7: Preprocessing time required to answer almost any path planning query in a fraction of a second (a) 70 seconds, (b) 150 seconds

is infrequent when enhancement is used.

Also, for fixed running time the network obtained with enhancement is usually denser. Fig. 6 illustrates this. It shows in (a) the component produced in 785 seconds whrn $N = 11500$, $M = 0$, and in (b) the one produced in the same time with enhancement ($N = M = 6500$). In these figures the robot configurations corresponding to the network nodes are drawn one on top of the other and it is impossible to distinguish them. But one can see which parts of the workspace have been attained and which ones have not.

We end this section with some general remarks. The example we used is difficult because it has a very narrow door and we ask the robot to go through it. Many interesting examples are not that difficult. Then our method spends less time in preprocessing. We show in Fig. 7 two such examples. The first of them involves an articulated linkage with 7 dof. The second shows the robot we considered above but in a workspace with a wider gate. For this case, after a preprocessing of only 150 seconds we can connect all the configurations of Fig. 3 to the network in less than 0.1 sec. Here, a single component is produced at the end of the enhancement step. Other examples can be found in [11].

7 Conclusion

We have described a new method for planning paths for robots with many dof. In this paper we analyze its performance on a difficult example. We have experimented with many 6 to 10 dof robots with very good results. With our technique, an initial cost is paid once in preprocessing the C-space. Afterwards, almost any path planning problem can be solved in a short time. An element of the success of the method is that it heuristically identifies the "difficult" regions of the C-space and enhances the information it has about them. Our approach is particularly useful in cases where repeated motions are to be carried out over the same environment, as is the case for many inspection, welding, and riveting tasks. Possible extensions of the method include:

• Some methods need to be devised to guess good values of the parameters of our algorithm. Adaptive/learning techniques may be useful.

• During path planning the network can be searched for the shortest path between two nodes, or the path that keeps a minimum clearance with the obstacles. In general, we may want to encode some features of the paths in the network edges and search for optimal paths for these features during path planning.

• After a path planning query, the network can be enhanced with the initial/configuration as new nodes, and the paths that connect these to the network.

• An extension of the method to changing environments is an interesting direction.

• Our experiments were conducted in 2D. Our planning approach would remain unchanged in 3D. The dimension of the workspace does not affect the number of generated nodes (this depends on the dimension of C-space), but expensive collision checking would increase preprocessing times.

References

[1] J. Barranquand and P. Ferbach, "Path planning through variational dynamic programming", TR 33, Paris Res. Lab., DEC, Paris, France, 1993.

[2] J. Barraquand, B. Langlois and J.C. Latombe, "Numerical Potential Field Techniques for Robot Path Planning", *IEEE Tr. on Syst., Man, and Cyb.*, 22(2):224-241, 1992.

[3] J. Barraquand and J.-C. Latombe, "Robot motion planning: A distributed representation approach", *Intl. J. of Rob. Res.*, 10(6):628–649, 1991.

[4] D. Challou and M. Gini, "Parallel Robot Motion Planning", *Proc. of IEEE ICRA*, GA, 46-51, 1993.

[5] B. Faverjon and P. Tournassoud, "A Practical Approach to Motion-Planning for Manipulators with Many Degrees of Freedom", *Robotics Research 5*, H. Miura and S. Arimoto, eds., MIT Press, Cambridge, MA, 424-433, 1990.

[6] L. Graux, P. Millies, P.L. Kociemba, and B. Langlois, "Integration of a Path Generation Algorithm into Off-Line Programming of AIRBUS Panels", *Aerospace Automated Fastening Conf. and Exp.*, SAE Tech. Paper 922404, Oct. 1992.

[7] K. Gupta and Z. Guo, "Sequential search with backtracking", *Proc. IEEE Intl. Conf. on Rob. and Autom.*, Nice, France, 2328-2333, 1992.

[8] Y. Hwang and N. Ahuja, "A potential field approach to path planning", *IEEE Tr. on Rob. and Autom.*, 8(1):23-32, 1992.

[9] L. Kavraki, J.-C. Latombe, M. Overmars and P. Švestka, "Probabilistic Roadmaps for Fast Path Planning", in preparation, Jan. 1994.

[10] L. Kavraki, J.-C. Latombe, "Randomized Preprocessing of Configuration Space for Fast Path Planning", Tech. Rep. STAN-CS-93-1490, Comp. Sci. Dept, Stanford Univ., Sept. 1993.

[11] L. Kavraki, J.-C. Latombe, "Randomized Preprocessing of Configuration Space for Path Planning: Articulated Robots", submitted to *IROS*, 1994.

[12] L. Kavraki, "Computation of Configuration-Space Obstacles using the Fast Fourier Transform", *Proc. IEEE Intl. Conf. on Rob. and Autom.*, Atlanta, GA, 255–261, 1993.

[13] K. Kondo, " Motion planning with six degrees of freedom by multistartergic bidirectional heuristic free space enumeration", *IEEE Tr. on Rob. and Autom.*, 7(3):267-277, 1991.

[14] J.-C. Latombe, *Robot Motion Planning*, Kluwer Academic Publishers, Boston, 1991.

[15] J. Lengyel, M. Reichert, B.R. Donald, and D.P. Greeberg, "Real-Time Robot Motion Planning Using Rasterizing Computer Graphics Hardware", *Proc. SIGGRAPH'90*, Dallas, 327-335, 1990.

[16] T. Lozano-Pérez and P. O'Donnell, "Parallel robot motion planning", *Proc. IEEE Intl. Conf. Rob. and Autom.*, Sacramento CA, 1000-1007, 1991.

[17] E. Mazer, J.M. Ahuactzin, G. Talbi and P. Bessiere, "The ariadne's clew algorithm", manuscript, 1992.

[18] M. Overmars, "A random approach to path planning", RUU-CS-92-32, Comp. Sci., Utrecht Univ., the Netherlands, October 1992.

[19] M. Overmars, P. Švestka,"A probabilistic learning approach to motion planning",RUU-CS-94-03, Comp. Sci., Utrecht Univ., the Netherlands, Jan. 1994.

[20] P. Švestka,"A probabilistic approach to motion planning for car-like robots", RUU-CS-93-18, Comp. Sci., Utrecht Univ., the Netherlands, April 1993.

[21] X. Zhu and K. Gupta, "On local minima and random search in robot motion planning", manuscript, 1993.