In the Proc. of IEEE/RSJ/GI International Conf. on Intelligent Robots and Systems (IROS), 1764-1772, 1994

Randomized Preprocessing of Configuration Space for Path Planning: Articulated Robots

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 describes the application of a recent approach to path planning for robots with many degrees of freedom (dof) to articulated robots moving in two or three dimensional static environments. The planning approach, which itself is not restricted to articulated robots, consists of a preprocessing and a planning stage. Preprocessing is done only once for a given environment and generates a connected 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. We show that after paying the preprocessing cost (on the order of minutes on a DEC Alpha workstation), planning is extremely fast (ranging from a fraction of a second to, at most, a few seconds) for many difficult examples involving 7-dof and 12-dof robots. 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.

1 Introduction

We describe the application of a recently developed path planning method to many-dof articulated robots operating in two or three dimensional environments. The planning method carries out a preprocessing of the configuration space (C-space), after which many difficult path planning problems can be solved in time ranging from a fraction of a second to a few seconds in the worst case.[1] The preprocessing itself is not very long: in the order of a few minutes.

During the preprocessing stage a set of collision-free configurations (*nodes*) are generated in the free C-space of the robot and interconnected into a network using very simple and fast path 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 several connected components, depending on the robot's free C-space and on the time spent on preprocessing.

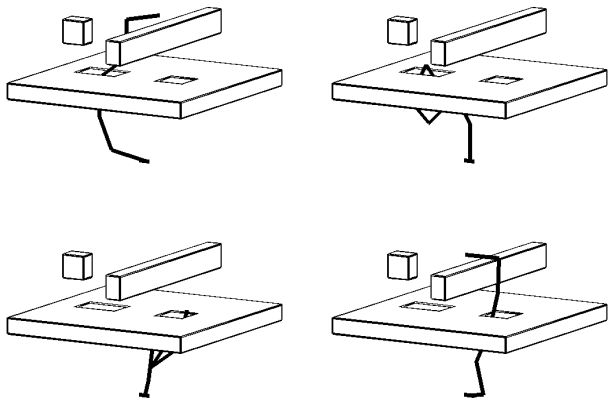

Figure 1: Snapshots along a path of a 12-dof robot

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 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.

Extensive experiments have been carried out with an implementation of the method on a DEC Alpha workstation. In this paper we report our results for articulated robots moving in 2D and 3D workspaces. Difficult planning problems like the one in Fig. 1 are solved in a fraction of a second after a preprocessing

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

step of 14 minutes. The robot used in this example has a fixed base and 6 spherical joints, for a total of 12 dof. Once preprocessing has been done, almost every path planning query in the same environment can be solved in a few seconds.

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, redundant 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. In Section 5 we discuss the implementation of the method for articulated robots and in Section 6 we report a series of experiments conducted with the implemented planner.

2 Relation to Previous Research

Path planning in a known environment has been studied extensively over the last years [13]. There exists a strong interest in developing efficient heuristic planners for many-dof robots, since complete methods have overwhelming complexity in high dimensions.

Heuristic planners for many-dof robots include the potential field method proposed in [5], where a learning scheme is used 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 grows too 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]. In [7] a sequential framework with backtracking is described for serial manipulators. Various search techniques that limit the C-space explored to find a path are investigated in [12] for 6-dof robots. A planner based on variational dynamic programming is introduced in [1]. Genetic algorithms have been employed in [16] and use of parallel processing techniques is investigated in [4, 15].

Potential field methods seem to be the most efficient techniques so far for dealing with many dof. Among them, the Randomized Path Planner (RPP) [3] has been experimented with for robots having 3 to 31 dof and is often very efficient. It has also been used in practice with good results [6]. However, several cases have been identified where RPP behaves poorly [4, 20]. RPP may fail to find a path in reasonable time if the robot falls into a local minimum of the potential field function and the way out of the local minimum is through narrow passages in the C-space which can not be easily located with the guidance of the potential function. 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.

The approach discussed in this paper is a learning scheme that scales efficiently in many dimensions and it is not restricted to any particular type of robot. Previous work on the approach is described in [10, 9]. A similar method developed independently is presented in [17, 18]. The same authors discuss an application of their approach to non-holonomic robots in [19]. A combination of the common ideas in [17, 18] with the method presented here is attempted in [8].

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. Graph Construction. During this step random configurations of the robot (nodes) are generated and are interconnected with a simple and fast planner.

When generating nodes, care is taken to produce a rather uniform distribution in the free C-space. For example, for the robot of Fig. 1, a node is generated by drawing a value for each of the 12 dof uniformly from its allowed range. After the 12 random choices have been made, the resulting configuration is tested for collision with obstacles and itself. We keep it only if it passes this test. A prespecified number N of nodes are computed in the above way. We discuss later how the choice of N affects the algorithm. In our many-dof examples, N is in the order of a few thousands.

We now try to interconnect the nodes obtained so far with a simple and fast planner. Given some metric in C-space, for each node x, we sort all other according to increasing distance from x. Then a simple planner (see Section 5) tries to connect x to each of the K closest nodes (K is a parameter). Each successful connection yields an edge of the network. Robot paths computed here are not stored since they can easily be recovered. The connected components of the resulting network are computed by a breadth-first search.

2. Graph Enhancement. Typically at the end of

step 1 we have a few large components and several small ones. The purpose of the enhancement 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 help 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.

For each node x found during graph construction, we define a *weight* $w(x)$, which should be large whenever x "is in a difficult region". Additionally, the weights for all x should add up to 1. Let us now call *expansion* of node x the creation at random of another node y in the free 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 value of 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, 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 $p(y)$ node x, that is the node responsible for y's creation. We repeat this M times. If function $w(x)$ adequately identifies the difficult parts of C-space, our heuristic will tend to fill these more than others.

The essential parameter in our scheme is the function $w(x)$. A description of possible weight functions is given in [9]. In our experiments we use the results of step 1 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 1, 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 may lie in narrow passages of the C-space and may contribute to producing a connected network in the free C-space.

After all M nodes have been produced (good values of M are between $N/2$ and 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 y is tested for connection with the K closest among the $N + M - 2$ other nodes which lie in a different component than y itself. The effect of the addition of the M nodes is a larger network, whose connected components are recomputed. At this stage, components which contain a small fraction (usually less than 0.5% of the total nodes) are discarded.

3. Further reduction of the number of components. For the examples of this paper, graph enhancement yields one connected component comprising most of the nodes, when $N + M$ is large enough. This component is stored for use during planning. There are difficult examples where a few large components remain at the end of step 2. A powerful planner can then be used to attempt connections among these components. This option is explained in depth in [9].

4 Path Planning Stage

Let x and y be the initial and final configurations of the robot. We first connect x to a node of the precomputed network. To do this we sort the network nodes in increasing distance from x and try to connect x with them, starting with the closest nodes, using the simple planner. If all these attempts fail, we execute a random walk of certain length 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.

Let A and B be the nodes with which x and y get connected respectively. 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. The path between x and y can be smoothed using any standard smoothing technique.

4. Implementation: Articulated Robots

We now discuss the implementation of our method for 2D and 3D articulated robots. We base our discussion on the robot of Fig. 2. This planar robot has 5 revolute joints. If its *base*, that is J_1, can move it has a total of 7 dof, otherwise it has 5 dof.

1. Generation of random configurations. To create a random configuration of the robot we draw each dof uniformly from its allowed range. Typically a very small percentage of the randomly guessed configurations are collision-free. Several optimizations can be

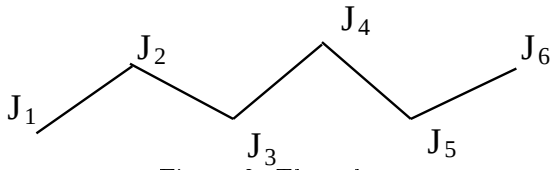

Figure 2: The robot

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 such optimizations are not performed.

2. Simple Planner. During the graph construction phase of preprocessing, a planner is needed to obtain thousands of connections. It is crucial that a simple and fast planner is used at that stage. The planner should have high chances of success for connecting configurations that are close together.

A possible candidate planner is the straight line in C-space. We have tried this planner and it gives reasonably good results. It is also very simple to implement for robots moving in 3D workspaces.

Another planner that proved useful for articulated robots is the following. Let x and y be the two configurations we attempt to connect and $J_1, \ldots, J_k$ be the points on the robot as shown in Fig. 2. We simultaneously translate every second J_i, that is J_{2*i+1}, $i = 0, \ldots k/2$, along the straight line in the workspace that connects its workspace position at configuration x to its workspace position at configuration y. Then we adjust the positions of J_{2*i}, $i = 1, \ldots k/2$, by computing the inverse kinematics of the robot. In this way the J_{2*i}'s follow the motion initiated by the J_{2*i+1}'s. If k is even then the position of J_k is not determined by the above. It can be determined by moving the last dof of the robot (the one that specifies the position of J_k) on a straight line in the C-space so that it reaches its desired value in configuration y. 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. The planner generalizes directly to 3D articulated robots and any kind of robot for which we can move a few of its dof at a certain direction and adjust the others.

3. Distance between two configurations. We would like to avoid invoking the simple planner in situations where it is unlikely to succeed. That is why we define a distance in C-space and call the planner only for nodes that are close together.

Let $J_i(x)$, $i = 1, \ldots, k$ denote the position of 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

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

where $\| J_i(x) - J_i(y) \|$ is the Euclidean distance between $J_i(x)$ and $J_i(y)$. This distance is quick to compute and has an intuitive meaning for articulated robots. Other distances can be used here. For example, for fixed-base robots it may be useful to weight the above sum with large weights assigned to the distances of the J_i's close to the base and small weights assigned to the distances of the J_i's towards the free end of the robot. This is because displacements of the links near the base have a more significant effect on the robot than displacements of the end effector.

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 2 for preprocessing. N should be determined by experimentation. For example, if path planning often fails, this is an indication that the network does not capture well the connectivity of the free C-space and thus N must be increased.

5. Choosing K. During graph construction, we attempt 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 preprocessing 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 $K = 30$.

6. Choosing M. During graph enhancement, we add M nodes in the "difficult" regions of the C-space. Setting M between 1/3 and 1/2 of the initial nodes gives good results. We select a node to be expanded with the probability distribution function of step 2. 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, for fixed-base robots we decrease this interval as we move towards the base of the robot. If we considerably vary a dof near the base we may create a configuration which is not close to the initial one.

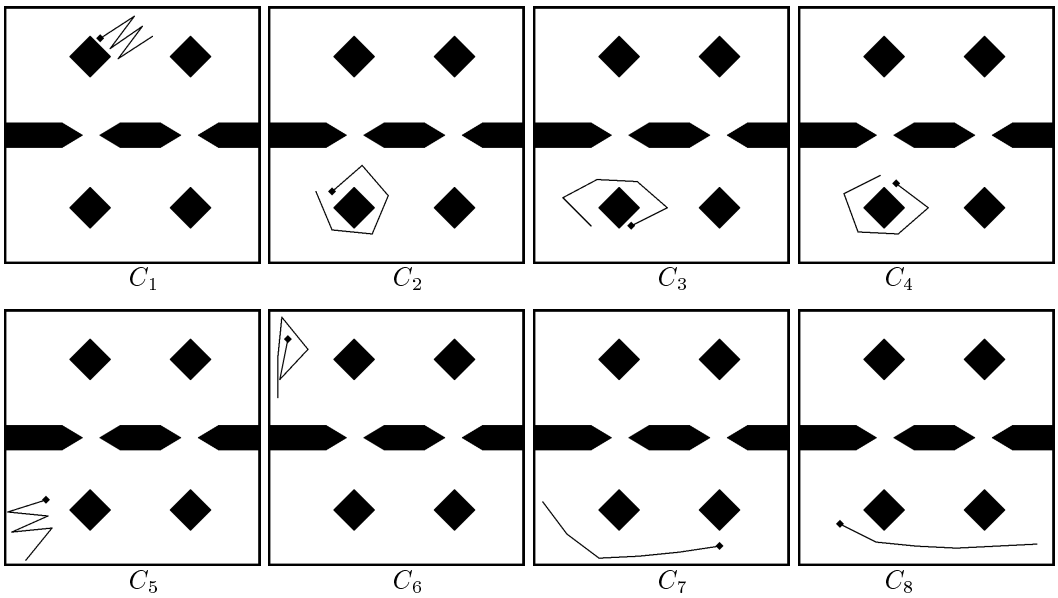

Figure 3: The robot is an articulated linkage with a moving base and 5 revolute joints (7dof)

N	M	Final nodes	Prepr. (sec)	C1	C2	C3	C4	C5	C6	C7	C8
600	300	236	12.68	0.00	F	F	F	F	1.12	F	F
800	400	470	19.67	0.00	F	F	F	F	0.17	F	F
1000	500	516	26.50	F	0.02	0.00	0.00	0.33	F	0.00	0.00
1200	600	779	35.18	0.03	F	F	F	F	F	F	F
1400	700	1695	42.20	0.02	0.02	0.00	0.02	0.12	2.42	0.00	0.00
1600	800	1931	49.79	0.02	0.02	0.00	0.02	0.12	1.15	0.00	0.00
1800	900	2287	58.63	0.12	0.00	0.02	0.00	0.25	0.40	0.02	0.00
2000	1000	2630	69.40	0.03	0.02	0.00	0.02	0.27	1.25	0.00	0.02

Figure 4: Preprocessing and time for connection to networks for $C_1, \ldots, C_8$ of Fig. 3

7. Collision checking. Collision checking for planar robots can be implemented using a discretized C-space bitmap for each link of the robot. We assume that each link of the robot is free to translate and rotate and we precompute for it a 3D 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. This technique is practical only for 2D workspaces, since 3D workspaces would require the generation of six-dimensional bitmaps. We use it in our implementation for planar robots (we discretize each dof to 128 values) and in particular we compute the C-space bitmaps with the use of the Fast Fourier Transform [11]. For self-collision, each link of the robot is tested against the others.

For robots moving in 3D workspaces, we check (analytically) each of their links against the obstacles and against other links. In this case, collision checking is more expensive and it increases preprocessing times.

5 Experimental Results

The planner is implemented in C and we used a DEC Alpha workstation (Model Flamingo rated at 121.5 SPECmark89) running under DEC OSF/1 for our experiments. We present experiments with a free-flying articulated robot and a fixed-base manipulator. In all figures, we draw a square at the base of the robot to distinguish it from the other end of the robot.

Free-flying Articulated Linkage. Fig. 3 shows a free articulated robot with 5 revolute joints (7 dof) and the environment in which it moves through a set of 8 different configurations. The configurations were specified manually. Path planning problems can be defined by selecting any two of these configurations.

Columns 1 and 2 of the table of Fig. 4 show the values of the parameters N (initial nodes) and M (nodes added in "difficult" regions). We set $M = N/2$. Column 3 gives the number of configurations in the largest component produced and column 4 the time spent on preprocessing in seconds. Smaller networks in this ta-

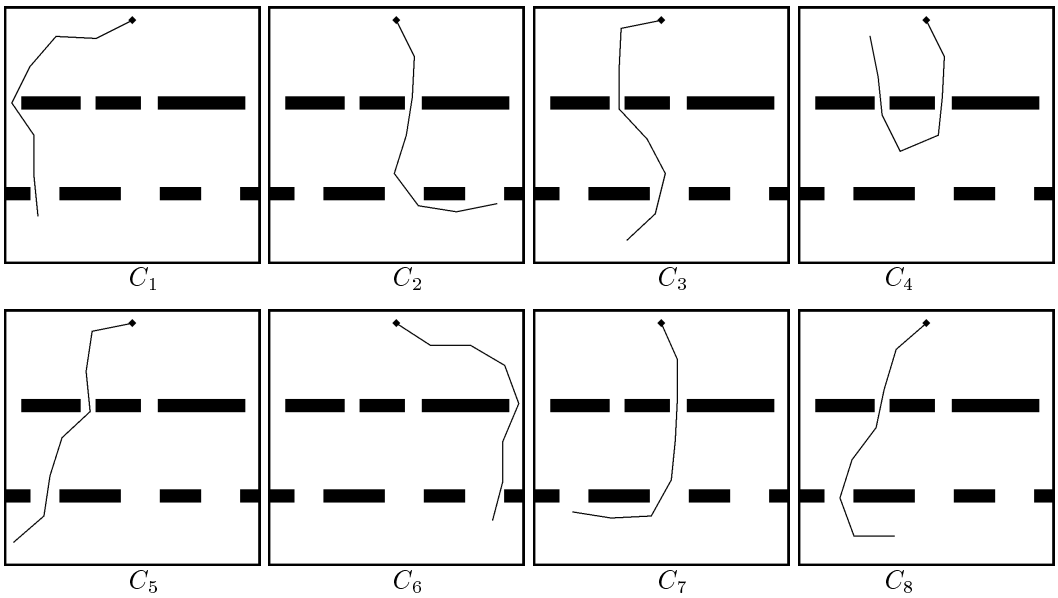

Figure 5: The robot is an articulated linkage with a fixed base and 7 revolute dof

N	M	Final nodes	Prepr. (sec)	C1	C2	C3	C4	C5	C6	C7	C8
800	400	583	25.69	0.30	F	F	F	F	F	F	F
1000	500	977	36.06	0.02	0.00	F	0.58	F	F	0.03	F
1200	600	1150	44.83	0.00	F	0.00	F	4.08	F	F	4.02
1400	700	1151	60.43	0.02	F	F	F	F	0.57	F	F
1600	800	2084	69.66	0.02	0.07	0.02	0.47	0.00	0.05	0.02	1.37
1800	900	2371	80.74	0.02	0.02	0.02	2.87	0.00	0.02	0.02	0.03
2000	1000	2646	92.78	0.00	0.02	0.18	2.25	0.02	0.02	0.52	0.02
2200	1100	2940	104.45	0.02	0.03	0.02	0.20	0.00	0.02	0.02	0.02

Figure 6: Preprocessing and time for connection to networks for $C_1, \ldots, C_8$ of Fig. 5

N	M	Final nodes	Prepr. (sec)	C1	C2	C3	C4	C5	C6	C7	C8
1200	0	615	33.43	0.30	F	F	F	F	F	F	F
1500	0	827	48.58	0.00	F	F	F	F	F	F	F
1800	0	1478	61.08	0.02	1.37	0.00	0.55	0.43	F	0.58	2.08
2100	0	1197	77.76	0.00	F	F	F	F	0.13	F	F
2400	0	2061	91.18	0.00	0.07	0.02	0.48	0.02	0.00	0.02	0.53
2700	0	1578	106.58	0.00	F	F	F	F	0.00	F	F
3000	0	2615	122.71	0.00	1.12	0.02	1.43	0.18	0.00	0.30	0.00
3300	0	2609	137.73	0.02	0.02	F	0.42	F	0.02	0.02	F

Figure 7: Preprocessing and time for connection to networks for $C_1, \ldots, C_8$ of Fig. 5, when M=0

ble are not part of larger networks; all networks were produced independently. Preprocessing time includes the graph construction and graph enhancement time. 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' indicates failure to obtain a connection of the configuration to the precomputed network after 30 random walks (their lengths are chosen uniformly in the interval $[100, 10000]$). 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. On the average, less than 0.1 seconds are spent searching the networks considered here.

It is clear from the table of Fig. 4 that if a small network is created, it does not capture well the struc-

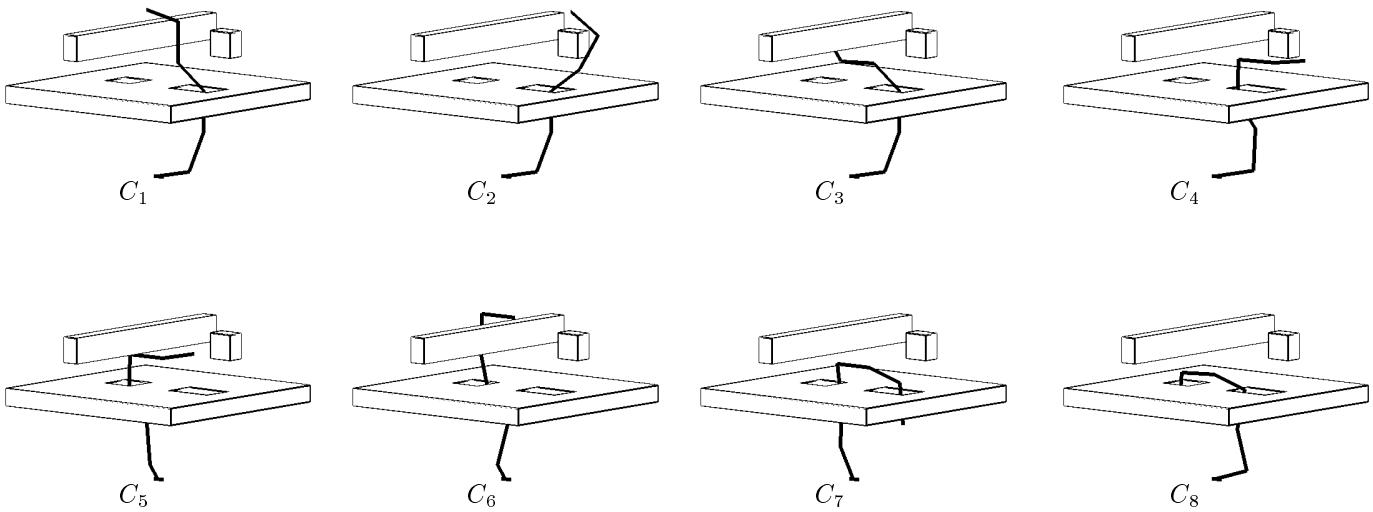

Figure 8: The robot has a fixed base and 6 spherical joints (12 dof)

N	M	Final nodes	Prepr. (sec)	C1	C2	C3	C4	C5	C6	C7	C8
500	500	906	134.33	11.93	F	F	6.22	12.10	0.07	F	F
800	800	1480	290.84	0.05	0.82	3.53	0.82	F	F	19.15	0.08
1100	1100	2112	475.66	0.07	0.20	4.05	0.07	F	F	F	4.90
1400	1400	2724	686.21	0.08	1.53	6.37	0.28	9.87	0.83	20.52	5.92
1700	1700	3376	884.23	0.05	0.23	7.88	0.08	3.08	0.17	7.33	4.60
2000	2000	3945	1051.84	0.05	0.08	3.00	0.10	8.88	0.07	20.58	3.85
2500	2500	4973	1432.89	0.07	0.70	0.05	0.50	0.05	0.07	6.30	7.02
3000	3000	5995	1837.21	0.10	0.10	3.65	0.83	0.17	0.87	5.32	13.53

Figure 9: Preprocessing and time for connection to network for $C_1, \ldots, C_8$ of Fig. 8

ture of the robot's free C-space and attempts to connect configurations $C_1, \ldots, C_8$ to it often fail. Path planning between any two of the above configurations will fail if either of them cannot be connected to the network. However, when $N + M$ is sufficiently large, 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. A preprocessing of 40 seconds is sufficient for this example.

Fixed-Base Articulated Linkage in 2D. We report preprocessing and connection to network times for another experiment with an 7-dof articulated robot with a fixed base. A set of 8 different configurations of the robot is shown in Fig. 5. The results of the method are given in Fig. 6. Again path planning takes a fraction of a second for most pairs of configurations of Fig. 5 after a preprocessing step of 70 seconds.

We show in Fig. 7 a similar table to that of Fig. 6 but with results obtained without the enhancement of the "difficult" regions ($M = 0$). Comparison between the corresponding rows of the two tables is possible because the sum of $N + M$ is the same. We observe that, in this case at least, the method is not very stable when enhancement is omitted. For example, for a small N ($N = 1600$) and enhancement ($M = 800$), we managed to connect all our configurations to the network produced, but for $N = 3300$ and without enhancement (last row) this did not happen.

For a different seed for the random number generator, we may obtain a better component for $N = 3300$ and good timings for the connection of the considered configurations. For this example, enhancement helps significantly improve the quality of the results of the preprocessing phase. It seems that some positions of the fixed-base robot are very constrained (especially the ones where the robot goes through one of the openings in the workspace walls). Enhancement of these configurations is crucial. For instance, with $N = 1600$, $M = 800$ we obtained one major component in 29 out of 40 trials. The corresponding success for the same total number of nodes and without en-

hancement ($N = 2400$, $M = 0$) was only 14 out of 40.

Fixed-Base Articulated Linkage in 3D. We show in Fig. 8 a 12-dof articulated robot. The robot has a fixed base and 6 spherical joints. We report in Fig. 9 the performance of our method. The simple planner used in this case is the straight line in C-space. Note that preprocessing time increases significantly. This is due to the fact the collision checking is more expensive in 3D. Apart from that, the behavior of the method is the same and planning takes a few seconds, in the worst case, after a preprocessing of 884 seconds.

6 Conclusion

We have described a new method for planning paths for many-dof robots. In this paper we analyze the performance of the method for articulated robots with 7 to 12 dof. 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 approach 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.

• Finally, it is interesting to extend the method to changing environments.

References

[1] J. Barraquand 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.*, 10:628–649, 1991.

[4] D. Challou and M. Gini, "Parallel Robot Motion Planning", *Proc. of IEEE ICRA*, GA, (2):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 Paper 922404, Oct. 1992.

[7] K. Gupta and Z. Guo, "Sequential search with backtracking", *Proc. IEEE ICRA*, Nice, 2328-2333, 1992.

[8] L. Kavraki, P. Švestka, J.-C. Latombe and M. Overmars, "Probabilistic Roadmaps for Fast Path Planning in High Dimensional Configuration Spaces", in preparation, April 1994.

[9] L. Kavraki, J.-C. Latombe, "Randomized Preprocessing of Configuration Space for Fast Path Planning", *Proc. IEEE ICRA*, San Diego CA, 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, "Computation of Configuration-Space Obstacles using the Fast Fourier Transform", *Proc. IEEE ICRA*, Atlanta, GA, 255–261, 1993.

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

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

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

[15] T. Lozano-Pérez and P. O'Donnell, "Parallel robot motion planning", *Proc. IEEE ICRA*, Sacramento CA, 1000-1007, 1991.

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

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

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

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

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