Real-Time Motion Planning in Changing Environments: Some Preliminary Results

Peter Leven, *University of Illinois, Urbana, IL, USA*
Seth Hutchinson, *University of Illinois, Urbana, IL, USA*

We present a new method for generating collision-free paths for robots operating in changing environments. Our approach is closely related to recent probabilistic roadmap approaches. These planners use preprocessing and query stages, and are aimed at planning many times in the same environment. In contrast, our preprocessing stage creates a representation of the configuration space that can be easily modified in real-time to account for changes in the environment. As with previous approaches, we begin by constructing a graph that represents a roadmap in the configuration space, but we do not construct this graph for a specific workspace. Instead, we construct the graph for an obstacle-free workspace, and encode the mapping from workspace cells to nodes and arcs in the graph. When the environment changes, this mapping is used to make the appropriate modifications to the graph, and plans can be generated by searching the modified graph.

After presenting the approach, we address a number of performance issues via extensive simulation results for robots with as many as twenty degrees of freedom. We evaluate memory requirements, preprocessing time, and the time to dynamically modify the graph and replan, all as a function of the number of degrees of freedom of the robot.

1 Introduction

In this paper, we present a new method for generating collision-free paths for robots operating changing environments. Our approach is closely related to the probabilistic roadmap planners [3, 5, 6, 8, 9, 14, 16]. These planners were originally used for robots that would construct many plans in the same environment, the idea being that the cost of planning will be amortized over many planning episodes justifies spending extensive amounts of time during a preprocessing stage, provided the resulting representation can be used to generate plans very quickly during a query stage. During the preprocessing stage, the planner generates a set of nodes that correspond to random configurations, connects the nodes using a motion planner to form a roadmap, and, if necessary, uses a subsequent sampling stage to enhance the roadmap. Planning is then reduced to a query processing stage, in which the initial and final configurations are connected to the roadmap[1], and the resulting graph is searched for a feasible path.

The sample configurations generated during the construction of a probabilistic roadmap can be thought of as landmarks in the configuration space. A family of approaches based on the Ariadne's Clew algorithm generates landmarks and search for paths between landmarks in a deterministic fashion by solving optimization problems. This method has been used to plan for manipulators [4, 11, 12]. and for manipulation planning (i.e., constructing a sequence of transfer and transit paths)[1].

Our new approach is a direct descendant of the probabilistic roadmap methods. However, unlike these, our method is aimed at changing environments, and therefore cannot exploit the premise that planning will occur many times in the same environment. Note that we are not concerned here with the problem of motion planning among moving obstacles whose trajectories are known a priori. We are interested in motion planning for environments in which obstacle locations are unknown a priori, or in which obstacle motion is unpredictable, such as would be the case for a mobile robot equipped with a manipulator operating in a previously unknown environment.

Our method begins, as do the probabilistic roadmap planners, by constructing a graph that represents the configuration space. Nodes are generated by a random sampling scheme, and connections between nodes are generated using a simple, straight-line planner. Unlike the probabilistic roadmap planners, we generate a roadmap that corresponds to an obstacle-free environment. Then, in a second phase of the preprocessing stage, we generate a representation that encodes the mapping from cells in the discretized workspace to nodes and arcs in the graph. These two phases are specific to the robot, but are independent of the environment in which the

[1] In our approach, connecting the initial and goal configurations to the graph is exactly the same as for the other roadmap-based approaches, and therefore we do not discuss this aspect further in the present paper.

robot will operate. The fact that the preprocessing is completely independent of the robot's target environment removes constraints on preprocessing time. Indeed, with our approach, it is feasible that when a new robot is designed, an extended period of preprocessing could be performed, at the end of which, the robot would essentially be preprogrammed to construct motion plans in any environment that it might encounter.

In the online planning phase, the planner first identifies the cells in the discretized workspace that correspond to obstacles, and then deletes the corresponding nodes and arcs from the graph, using this encoded mapping. Planning is then reduced to connecting the initial and final configurations to the graph (again, as is also the case with the probabilistic roadmap planners), and then searching the graph for a path between these newly added nodes. Of course it is possible to add obstacles to the environment in such a way that the graph becomes disconnected, but this is true for any of the probabilistic roadmap planners (once one knows how samples are selected, and how these samples are connected by local planners, it is fairly straightforward to construct environments that will thwart them).

In the online planning stage, our method runs in real time; plans are generated in less than one second. Thus, it is feasible to use the planner even in the case when obstacles are moving in the environment with unknown trajectories, provided a sensing system can identify in real-time those regions of the workspace that are occupied by the obstacles.

There have been a number of previous approaches to path planning in changing environments. In some cases, planners have execution times that make it feasible to directly use them in some kinds of changing environments with no modifications. This is the case, e.g., for the Ariadne's Clew algorithm reported in [4, 11]. The Ariadne's Clew algorithm operates by generating landmarks (during an exploration phase) and then connecting them to the existing network (the search phase). Variations of this algorithm can be obtained by varying the search phase, and by using different optimization criteria to select candidate landmarks [1, 12]. The idea of incrementally expanding a network for single query planning has also been used in [6] and [9]. In both of these, networks are grown from both the initial and goal configurations until they can be connected. In [6] the notion of expansive configuration spaces is used, while in the latter, random trees are used. In [15] an adaptable approach that uses multiple local planners is described. At runtime, characteristics of the problem are used to determine which (combination of) local planners will be most effective. We also note here that in two of these previous approaches ([5] and [12]), the idea of somehow representing the mapping from the workspace to the configuration space was incorporated.

We believe that our planner is the first to construct a representation that can be used to replan in real time in changing environments. In the remainder of the paper we describe the details of the algorithm and give an evaluation of our methods via simulation results.

2 Constructing the Roadmap

Our construction of a roadmap of the configuration space is very similar to methods used in previous probabilistic roadmap planers [7]. Nodes are generated by generating sample configurations, and these nodes are then connected to form a graph. We will denote this graph by $\mathcal{G} = \langle \mathcal{G}_n, \mathcal{G}_a \rangle$, in which $\mathcal{G}_n$ is the set of nodes in the graph, and by $\mathcal{G}_a$ the set of arcs in the graph. Since the graph is constructed without the presence of obstacles in the workspace, local planning to connect the nodes is trivial.

Since there are no obstacles to consider, it is fairly easy to generate samples in the configuration space. The only hard constraint is that self-collision (i.e., collision between distinct links of the robot) is prohibited. At present, we generate configuration nodes by sampling from a uniform distribution on the configuration space [7].

To create a graph, pairs of nodes are connected by arcs that correspond to trajectories of the arm. Like several of the previously mentioned approaches, we connect each node to its k-nearest neighbors. In our current implementation, the paths corresponding to arcs are not tested to ensure that no self-collisions take place along the path.

The distance metric used for determining the neighbors of a node plays a very important role in the probabilistic roadmap methods [2]. For our application, the swept volume of a path in the workspace connecting two configurations would be an ideal metric. Unfortunately, as noted by others [2, 7], this metric is very expensive to compute, and therefore, in our current implementation, we have opted for approximations that are faster to compute. In [2], a number of metrics were evaluated for efficiency and effectiveness for the case of a rigid object translating and rotating in a three-dimensional workspace. In particular, we have performed extensive experimental evaluation with the following four distance metrics:

the 2-norm in C-space

$$\mathcal{D}_2^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \|\mathrm{q}' - \mathrm{q}\| = \left[\sum_{i=1}^{n} (q_i' - q_i)^2 \right]^{\frac{1}{2}}$$

the ∞-norm in C-space

$$\mathcal{D}_\infty^{\mathcal{C}}(\mathrm{q}, \mathrm{q}') = \max_n |q_i' - q_i|$$

the 2-norm in workspace

$$\mathcal{D}_2^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \left[\sum_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|^2\right]^{\frac{1}{2}}$$

the ∞-norm in workspace

$$\mathcal{D}_\infty^{\mathcal{W}}(\mathrm{q},\mathrm{q}') = \max_{\mathrm{p}\in\mathcal{A}} \left\|\mathrm{p}(\mathrm{q}') - \mathrm{p}(\mathrm{q})\right\|.$$

In each of these equations, the robot has n joints, q and q$'$ are the two configurations corresponding to different nodes in the graph, q_i refers to the configuration of the ith joint, and p(q) refers to the workspace reference point p of the set of reference points $\mathcal{A}$ at configuration q. Versions of $\mathcal{D}_\infty^{\mathcal{W}}$ and $\mathcal{D}_2^{\mathcal{W}}$ were also used in [7]. Once a node's k-nearest neighbors have been identified, they are connected by a straight-line planner.

3 Workspace to C-space mapping

The ability to plan in real time in changing environments comes from our encoding of the relationship between the workspace and the C-space. To represent the workspace, we use a uniform, rectangular decomposition, which we denote by $\mathcal{W}$. We denote the C-space by $\mathcal{C}$, and define the mapping $\phi : \mathcal{W} \to \mathcal{C}$ as

$$\phi(w) = \{\mathrm{q} \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\},$$

in which w is cell of the workspace and $\mathcal{A}(\mathrm{q})$ denotes that subset of $\mathcal{W}$ occupied by the robot at configuration q. We note that $\phi(w)$ is exactly the $\mathcal{C}$ obstacle region (often denoted by $\mathcal{CB}$) if w is considered as a polyhedral obstacle in the workspace. In our approach, we do not explicitly represent the $\mathcal{C}$, but instead use the roadmap $\mathcal{G}$. Therefore, we define two additional mappings, one from the workspace to the nodes in the graph, and one from the workspace to the arcs in the graph:

$$\phi_n(w) = \{\mathrm{q} \in \mathcal{G}_n \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset\},$$
$$\phi_a(w) = \{\gamma \in \mathcal{G}_a \mid \mathcal{A}(\mathrm{q}) \cap w \neq \emptyset \text{ for some } \mathrm{q} \in \gamma\}.$$

In terms of the implementation, it is much easier to compute the inverse maps ϕ_n^{-1} and ϕ_a^{-1}. Therefore, we use the inverse maps for the construction of our representation of the mapping.

The construction of the representation for ϕ_n^{-1} is straightforward. For each $\mathrm{q} \in \mathcal{G}_n$ we note the mapping from each $w \in \phi_n^{-1}(\mathrm{q})$ to the corresponding q. The set of cells in $\phi_n^{-1}(\mathrm{q})$ is computed by expanding a from seed cell. The seed for fixed-base articulated robots is the origin of the robot, which is not stored as part of the node obstacle data structure. The shell expands in each direction in the workspace until the collision tester determines that a cell is outside the robot. We use the collision-checking package V-Clip [13] to test for cell intersection with the robot.

The computation of ϕ_a^{-1} is more complex and time-consuming. It involves computing the swept-volume of the robot as it traverses a path computed by the local planner between two configurations. In our implementation, cells that are occupied by the robot at the endpoints of the arc γ (i.e., the configurations that correspond to the two nodes connected by the arc) are not included in $\phi_a^{-1}(\gamma)$. Computing the swept volume for a robot trajectory is not a trivial problem. A method for swept volume computation for three dimensional objects that can only translate is presented in [17]. An extension to this method was presented that could also accommodate rotation about a single axis, but no results were given. This approach could be used to simplify the computation of ϕ_a^{-1}.

We have developed a method for computing $\phi_a^{-1}(\gamma)$ as follows. First, ϕ_n^{-1} for the two endpoints of γ is computed. Then, the path corresponding to γ is sampled using a recursive bisection method, which proceeds as follows. First, the configuration corresponding to the midpoint of the segment connecting the two nodes is computed, and ϕ_n^{-1} for that configuration is computed. If this set contains any cells not already in ϕ_n^{-1} of the endpoints, these new cells are added to the swept volume, and the path is subdivided again on both sides of the midpoint. This process is repeated until no new cells of $\mathcal{W}$ are added.

4 Empirical evaluation

For a real-time motion planner, we are interested in three parameters: graph update time, planning time, and data structure size. We are also interested in the preprocessing time, though this is less important. In the this section, we evaluate a preliminary version of the planner, studying the case of serial-link manipulators with revolute joints operating in a two-dimensional workspace. Note that our implementation is not limited to revolute joints; prismatic joints can be used as well. We limit the following discussion to revolute joints to simplify the analysis.

4.1 Experimental set-up

The following experimental set-up was used to evaluate the approach. The robots tested are all serial-link manipulators with two to 20 revolute joints in a two-dimensional workspace. The workspace cells and the robot are given three-dimensional polyhedral descriptions to allow the use of standard collision checking packages to be used for collision testing. Each cell of the workspace is modeled as a unit cube, and each link of

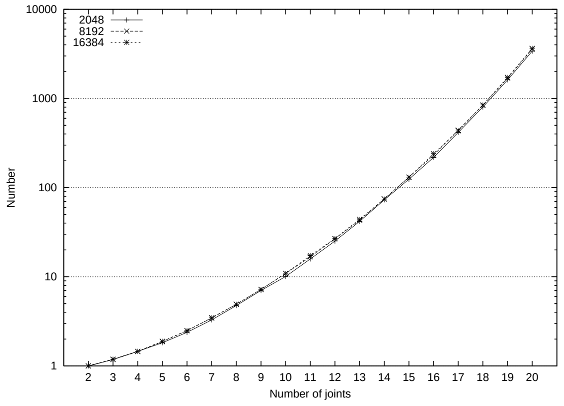


Figure 1: Number of samples of $\mathcal{C}$ generated per node accepted.

the robot is modeled as a rectanguloid with a width of 2.1 units and a height of 1 unit. The total length of the robot was fixed at 70 units, with each link length being scaled appropriately. Each joint of the robot was given the full revolute range of motion, and the joints were allowed to wrap around. In addition, the robot was not tested for self-collision, except when the samples of $\mathcal{C}$ were generated. To evaluate how the data structures and computation time grow with the size of the graphs, we tested graphs with 2048, 8192, and 16384 nodes.

4.2 Results for serial-link manipulators in a 2D workspace

In this section we present some results from using the above setup. We start with an analysis of the data structures that are computed to implement our planning approach. Each of the four data structures—$\mathcal{G}_n$, $\mathcal{G}_a$, ϕ_n, and ϕ_a—was computed and stored separately to simplify analysis. We then follow the analysis with some planning examples.

4.2.1 Sampling $\mathcal{C}$ For each of the robots, $\mathcal{C}$ was sampled uniformly at random, except for the first 129 samples. The first 129 samples were chosen such that the robot covers as much of the workspace as possible. For the robots tested, this means that 129 uniformly-spaced samples were taken of the range of the first joint, and the remaining joints were held fixed at a position that maximized the length of the robot. Of the random samples, samples in which the robot collides with itself were rejected.

As seen in Figure 1, when sampling $\mathcal{C}$ uniformly, it became exponentially more difficult as the number of joints increases to find configurations in which the robot does not collide with itself. Notice that the y-axis scale of this graph is logarithmic. This exponential effect is a consequence of a robot consisting of revolute joints operating in the plane: for each set of three links, there is a region of $\mathcal{C}$ in which the third link is in collision with the first. For each additional link beyond three, there is a combinatoric effect with combinations of links in collision, as well as an interaction among these links that tends to make the collision regions larger. This effect is expected to be drastically reduced for non-planar robots operating in 3D.

4.2.2 Graph construction As described above, arcs in the graph are constructed for the k-nearest neighbors of each node. For our evaluation, we tested the case of $k = 5$ nearest neighbors. We computed the nearest neighbors using the simple $O(n^2)$ method of calculating the distance between all pairs of nodes and keeping the closest five for each node.

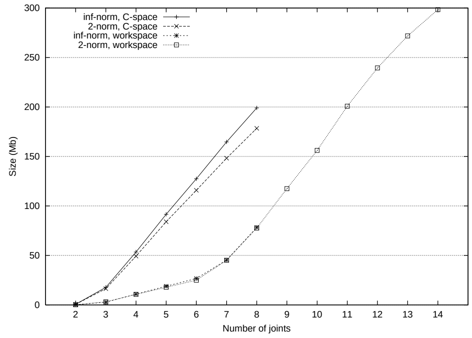
There is not much difference between the computation time between the 2-norm and the ∞-norm, except that the computations involving the ∞-norm tend to take somewhat more time to perform (this implies a processor architecture in which a branch is more costly to compute than a floating-point multiply). There is more of a difference between the metric defined in $\mathcal{C}$ versus the metric defined in $\mathcal{W}$, where the latter appears to have a more linear increase in computation time as the number of joints increases. Overall, for robots with more than five joints, the workspace metrics take more time to compute.

The size of the graph varies linearly with the number of nodes, with an average size of 163kB for 2048 nodes, 654kB for 8192 nodes, and 1309kB for 16384 nodes. The variance in size from the average was less than 15%.

4.2.3 Computing ϕ_n The overall size of ϕ_n is the product of the number of nodes and the average number of cells the robot covers in the workspace. For a robot with a fixed maximum length, this means that the size of the data structure is largely independent of the number of joints, as we have observed in our simulations.

4.2.4 Computing ϕ_a The size of the ϕ_a for graphs with 16384 nodes nodes is shown in Figure 2. As can be seen in the figure, this data structure can become quite large. This is the reason that the computation of ϕ_a stops at 13 joints for the 8192 node case, and 8 joints for the 16384 node case. The limit is the process size of the program computing ϕ_a; in order to keep the computation times comparable, the memory size of the program computing ϕ_a was limited to the physical memory available in the computer. More examples were computed for the $\mathcal{D}_2^{\mathcal{W}}$ metric for the 8192 and 16384 node cases in order to evaluate the compressibility of the resulting data files (see [10]).

Shown in Figure 3 is the time it takes to compute the ϕ_a for the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ metrics (the $\mathcal{D}_\infty^{\mathcal{C}}$ and the $\mathcal{D}_\infty^{\mathcal{W}}$ produce similar results to the $\mathcal{D}_2^{\mathcal{C}}$ and $\mathcal{D}_2^{\mathcal{W}}$ metrics, re-



16384 nodes

Figure 2: Size of ϕ_a.

spectively). These times correlate well with the increase in the size of the respective data structures: the larger the data structure is, the longer it takes to compute.

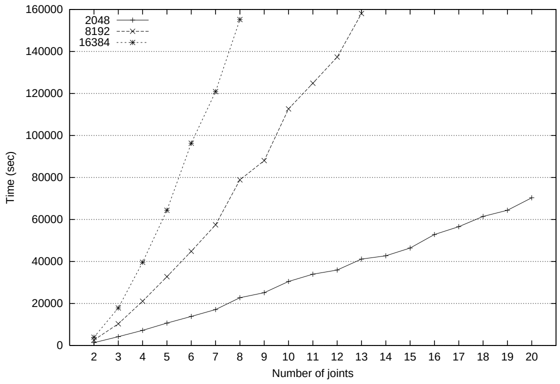
An interesting feature of both Figures 2 and 3 is the apparent inflection point at six joints in the graphs for the $\mathcal{D}_2^{\mathcal{W}}$ and $\mathcal{D}_\infty^{\mathcal{W}}$ metrics. This inflection point is more obvious with the 8192 and 16384 node graphs. An explanation for this is that, given the resolution of the workspace that we are using, the random sampling is better able to "fill" $\mathcal{C}$ with samples for the robots with fewer joints. This results in the nodes being closer together, and, therefore, the paths between them do not cover as much of $\mathcal{W}$.

4.2.5 Planning examples Our motion planner tests the connectivity of the graph between two selected nodes in the modified graph. We have run our program on many examples, and in all of the planning runs tested, it takes less than a second to update $\mathcal{G}_n$ and $\mathcal{G}_a$ using ϕ_n and ϕ_a and to find a path in the modified graph.

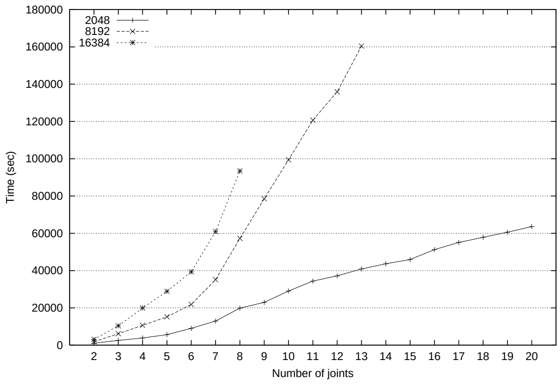
Shown in Figure 4 is a 19-joint robot negotiating between two obstacles (black objects in the figure). The size of a grid cell in the figure is 11 units on a side. This plan was generated from the graph that used the $\mathcal{D}_2^{\mathcal{W}}$ distance metric with 2048 nodes. The fan-like parts of the plan come from the set of nodes created to span the workspace of the robot, as described in Section 4.2.1. Figure 5 shows a plan for a 5-joint robot. For this environment and 2048 nodes, only about one-third of the 76 graphs had a connection that could solve this problem (note that, for robots with 2 to 4 joints, the length of the first link precludes the possibility of a plan). When the number of nodes is increased to 8192, most of the graphs had a connection that could solve the problem.

5 Conclusions

We have presented what we believe to be the first planner that is fully able to plan in real time in changing environments. We have presented extensive analysis of



$\mathcal{D}_2^{\mathcal{C}}$ metric



$\mathcal{D}_2^{\mathcal{W}}$ metric

Figure 3: Computation time for ϕ_a for two metrics.

the planner via planning simulations. Although our preliminary results are quite promising, there are a number of open issues that remain; many of these are outlined in [10].

References

[1] Juan Manuel Ahuactzin, Kamal Gupta, and Emmanuel Mazer. Manipulation planning for redundant robots: A practical approach. *International Journal of Robotics Research*, 17(7):731–747, July 1998.

[2] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proc. IEEE Conf. Robotics and Automation*, pages 630–637, 1998.

[3] Nancy M. Amato, O. Burchan Bayazit, Lucia K. Dale, Christopher Jones, and Daniel Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[4] Pierre Bessière, Juan-Manuel Ahuactzin, El-Ghazali Talbi, and Emmanuel Mazer. The "Ariadne's Clew" algorithm: Global planning with local methods. In *Pro-*

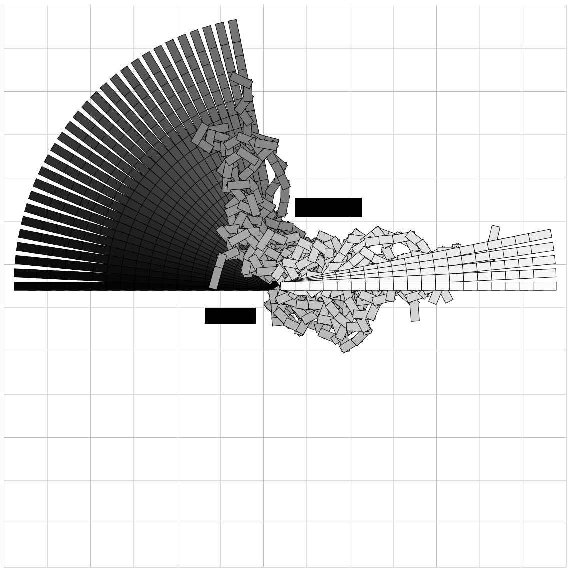


Figure 4: A plan for a 19-joint robot passing through a (relatively) narrow corridor. The dark blocks are the obstacles.

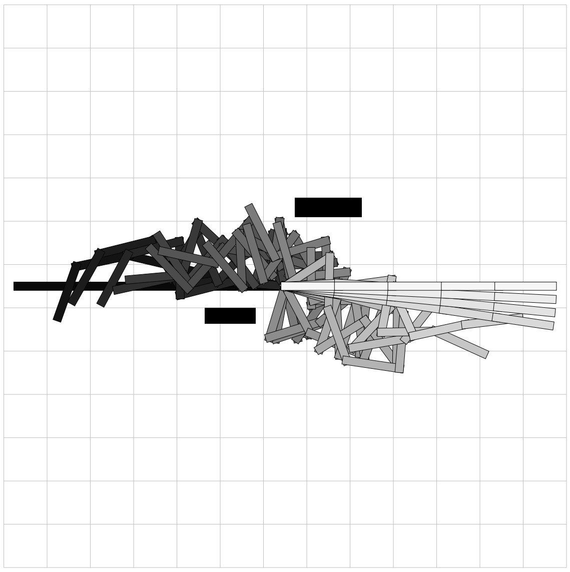


Figure 5: A plan for a 5-joint robot passing through the same corridor.

ceedings of the Workshop on Algorithmic Foundations of Robotics, pages 39–47, 1994.

[5] T. Horsch, F. Schwarz, and H. Tolle. Motion planning with many degrees of freedom — random reflections at c-space obstacles. In *Proc. IEEE Conf. Robotics and Automation*, pages 3318–3323, 1994.

[6] David Hsu, Jean-Claude Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Proc. IEEE Conf. Robotics and Automation*, pages 2719–2726, 1997.

[7] L.E. Kavraki, P. Švestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, August 1996.

[8] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proc. IEEE Conf. Robotics and Automation*, volume 3, pages 2138–2145, 1994.

[9] James J. Kuffner, Jr. and Steven M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Conf. Robotics and Automation*, 2000. To appear.

[10] Peter Leven and Seth Hutchinson. Toward real-time path planning in changing environments. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, 2000. To appear.

[11] Emmanuel Mazer, Juan Manuel Ahuactzin, and Pierre Bessière. The ariadne's clew algorithm. *Journal of Artificial Intelligence Research*, 9:295–316, 1998.

[12] Alistair McLean and Isabelle Mazon. Incremental roadmaps and global path planning in evolving industrial environments. In *Proc. IEEE Conf. Robotics and Automation*, pages 101–107, 1996.

[13] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. Technical Report TR97-05, Mitsubishi Electric Research Laboratory, 201 Broadway, Cambridge, MA 02139, June 1997.

[14] Mark H. Overmars and Petr Švestka. A probabilistic learning approach to motion planning. In *Proceedings of the Workshop on Algorithmic Foundations of Robotics*, pages 19–37, 1994.

[15] Daniel Vallejo, Christopher Jones, and Nancy M. Amato. An adaptive framework for 'single shot' motion planning. Technical Report 99-024, Department of Computer Science, Texas A&M University, College Station, TX, October 1999.

[16] Steven A. Wilmarth, Nancy M. Amato, and Peter F. Stiller. Motion planning for a rigid body using random networks on the medial axis of the free space. In *Proceedings of ACM Symposium on Computational Geometry*, pages 173–180, 1999.

[17] Patrick G. Xavier. Fast swept-volume distance for robust collision detection. In *Proc. IEEE Conf. Robotics and Automation*, pages 1162–1169, 1997.