Proceedings of IROS 2000, In Press

A Two Level Fuzzy PRM for Manipulation Planning

Christian L. Nielsen
The Maersk Institute
University of Southern Denmark
5220 Odense SØ, Denmark
nielsen@mip.sdu.dk

Lydia E. Kavraki
Department of Computer Science
Rice University
Houston, TX 77005, USA
kavraki@cs.rice.edu

Abstract

This paper presents an algorithm which extends the probabilistic roadmap (PRM) framework to handle manipulation planning. This is done by using a two level approach, a PRM of PRMs. The first level builds a manipulation graph, whose nodes represent stable placements of the manipulated objects while the edges represent transfer and transit actions. The actual motion planning for the transfer and transit paths is done by PRM planners at the second level. The approach is made possible by the introduction of a new kind of roadmap, called the fuzzy roadmap. The fuzzy roadmap contains edges which are not verified by a local planner during construction. Instead, each edge is assigned a number which represents the probability that it is feasible. Later, if the edge is part of a solution path, the edge is checked for collisions. The overall effect is that our roadmaps evolve iteratively until they contain a solution. The use of fuzzy roadmaps in both levels of our manipulation planner offers many advantages. At the first level, a fuzzy roadmap represents the manipulation graph and addresses the problem of having probabilistically complete planners at the second level. At the second level, fuzzy roadmaps drastically reduce the number of collision checks. The paper contains experimental results demonstrating the feasibility and efficiency of our scheme.

1 Introduction

Problem Definition *The general manipulation planning problem* deals with path planning for robots manipulating movable objects among static obstacles. The goal is to bring the movable objects from a given start configuration to a given goal configuration. The objects are only able to move when they are grasped by a robot. Objects which a are not grasped must be at a stable position e.g., resting against the obstacles or other stable objects. For a more detailed description of the manipulation planning problem see [3].

The solution to the manipulation planning problem has the form of a *manipulation path* which is a sequence of *transit* and *transfer paths*. A transfer path is a subpath of the global plan where the object is grasped and moved by the robot. A transit path is a subpath where the object is left in a stable position while the robot changes grasp. Regrasping operations are planned automatically by a manipulation planner.

Importance Manipulation planning has resisted efficient solutions so far. This is not surprising given the complexity of the simple path planning problem [12] and the additional difficulties imposed by the planning of grasping and regrasping operations. In industrial studies manipulation is frequently ignored. For example, in assembly maintainability studies, objects are assumed capable of moving by themselves; it is computationally infeasible to consider both the objects and the tools/arms needed to manipulate them with the current state-of-the-art manipulation planning. Without doubt a reliable manipulation planner will permit further testing of setups and products and the development of realistic simulations for industrial settings. Novel application domains such as surgical simulations and animation will also benefit from a manipulation planner [11].

Related Work Different algorithms solving variations of the general manipulation planning problem have been proposed in the literature. The solution of Koga and Latombe [10, 11] is to first plan a path for the object, and then find a sequence of connected transfer and transit paths, that make the object trace this path. Another method proposed by Ferbach and Barraquand [7] is based on variational dynamic programming. The authors first plan a manipulation path in the unconstrained composite configuration space of the robots and objects. Then they iteratively enforce the manipulation constraints by making small perturbations to the path to minimize a gradually more and more strict penalty function, until the final path is collision free and meets the manipulation constraints. Ahuactzin, Gupta and Mazer [2] have proposed a method for manipulation planning which is an extension of the *Adriadne's Clew Algorithm* [5] based on genetic algorithms. Alami, Laumond and Simeon[3] presented a clear and general description of the manipulation planning problem. They gave solutions for both the case of discrete placements of the object and the case of an infinite set of grasps. In the case of discrete placements of the object, the building of the manipulation graph effectively decomposes the problem into a set of point-to-point path planning problems.

Overview of Our Approach In order to describe our work we first give a quick overview of PRM [9, 14]. PRM builds a roadmap of nodes in the configuration space (C-space) of the robot. To construct the roadmap, a number of random configurations are selected in the free C-space and are used as nodes. Close nodes are tested for connection with local planners. Each time a local planner succeeds, the corresponding edge is inserted in the roadmap. This random exploration of the C-space is often accompanied by a heuristic that adds extra nodes in "difficult" regions of the C-space, to improve the performance in cases where the solution path goes through narrow passages [4, 8, 14]. Individual planning queries are solved by adding the start and goal configuration as nodes in the roadmap and then using graph search to find a path connecting these nodes.

The application of the PRM framework to manipulation planning was possible because of the introduction of fuzzy roadmaps. Fuzzy roadmaps provide a simple way to deal with the problems associated with using probabilistic point-to-point path planners as local planners, and also provide efficient point-to-point planners.

The Fuzzy PRM introduced in Section 2 of this paper, builds a *fuzzy roadmap* whose edges are annotated by a probability. This probability is an estimate of the chance that the edge is actually feasible i.e. collision free. The planner drastically reduced the number of collision checks by minimizing the number of checks which are usually wasted while verifying edges that are not part of the final solution path. [1]

In Section 3 we show how to use the Fuzzy PRM to develop a manipulation planner. Our manipulation planner is itself a Fuzzy PRM planner which uses fuzzy point-to-point PRM planners as "local planners".

In Section 4 we present our experimental results. We conclude with a discussion in Section 5.

[1] Note that the Fuzzy PRM bears a strong resemblance to Lazy PRM developed simultaneously and independently by Bohlin and Kavraki [6].

2 Fuzzy PRM

In a traditional roadmap an edge between two nodes represents the fact that a particular local planner was able to connect the two nodes by a feasible path. The absence of an edge means that the connection failed or was never even tried (because the nodes were too far apart). We could say that if there exist an edge between two nodes, then the edge found by the local planner is feasible with probability 1, and if no edge exist then the probability is 0. The idea behind fuzzy roadmaps is to annotate all edges of the roadmap with such a probability, called the *edge probability.* Instead of restricting the probability to $\{0, 1\}$ we let it take any value in $[0, 1]$; hence the name "fuzzy" roadmap.

The edge probability is an estimate of the chance that a path will later be found by a local planner. In this context we are free to use a kind of local planner, which do not necessarily guarantee the feasibility of a given path, but which is able to make a fast estimate of the probability of its existence.

Just like the general PRM, the Fuzzy PRM has a learning phase and a query phase, but in the Fuzzy PRM we start by entering the query phase, and only if the roadmap (initially consisting of only the start and goal node) do not contain a possible solution path, then we jump to the learning phase to improve the roadmap. Figure 1 shows a flow chart outlining the algorithm. In the Fuzzy PRM we do not run the local planner to verify edges added during the learning phase. Instead of verifying the edges, we assign them an initial edge probability. Then later in the query phase, if an edge is part of a possible solution path, we apply the local planner to upgrade its probability.

The query phase The query phase is split into three steps: an update, a search, and an upgrade step.

In the update step we add nodes representing the start and goal configuration (if they are not already there) and connect these to the roadmap.

In the search step we find the most probable path from start to goal through the roadmap. The probability of a path τ is defined as the product of the probabilities of the edges $e_1, e_2, .., e_n$ along the path: $p(\tau) = \prod_{i=1}^{n} p(e_i)$.

We find this path by setting the edge weights to $w(e) = -log(p(e))$ and then by using Dijkstra's algorithm to find the path with the minimum sum of edge weights. If no path exist we return failure, else we proceed with the upgrade step.

The upgrade step handles the actual verification of the path. First we insert all edges into a priority

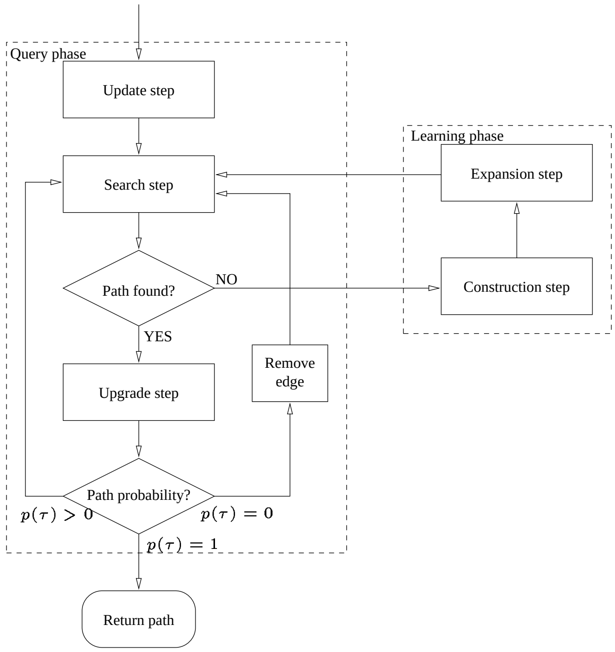


Figure 1: Flow chart with an overview of fuzzy PRM.

queue with their edge probability as their keys. Then we repeat the following loop until either all edges reach probability 1 or until the local planner discovers a collision along the path: Extract the edge with the lowest probability from the queue, make the local planner perform additional collision checks along the path, and assign the edge a new probability depending on the new collision check resolution. If the local planner discovers a collision, we delete the corresponding edge from the roadmap and return to the search step, else we reinsert the edge into the priority queue and continue the upgrade loop. If the probability of all the edges along the path is upgraded to 1 then the query is solved and we return the path.

When we need to upgrade a path we call the following local planner:

LOCAL PLANNER(q_{source}, q_{target}, $level$)
$r \leftarrow q_{target} - q_{source}$
$steps \leftarrow 2^{level-1}$
$i \leftarrow 0$
while $i < steps$
  $qt \leftarrow q_{source} + \frac{r}{2*steps} + \frac{r}{steps} i$
  if $qt \notin C_{free}$
    return false
  $i \leftarrow i + 1$
return true

The local planner performs collision checks on the straight line between the source and target configuration of the path e.i. the linear interpolation between q_{source} and q_{target}. The first time we need to upgrade a path we call the local planner with $level = 1$, and the local planner performs a single collision check at the middle of the path, thereby dividing the path into two unchecked sections. Next time we need to upgrade the path we increase $level$ by one and call the local planner again. The result is that we halve the distance between successive collision checks along the line each time we upgrade the edge.

We have chosen to make the simplifying assumption that the number of times a given C-space path passes the boundary of the C-obstacle only depends on the length of the path. We make this assumption even though it is not true for most structured environments, because it allows us to choose a simple probability model, relating the length of the path to the number of obstacle boundary penetrations, namely the Poisson distribution: $P\{N(l) = n\} = e^{-\lambda l}\frac{(\lambda l)^n}{n!}$, where l is the length of the path, $N(l)$ is a stochastic variable representing the number of times the path crosses the C-obstacle boundary, and λl is the mean value of $N(l)$.

If a path of length l has been successfully checked at level $level$ by the local planner outlined above, then it has been divided into 2^{level} subsections, each of length $\frac{l}{2^{level}}$. By using the Poisson probability model and the knowledge that N must be even (because the configuration in each end of the path is collision free) we can deduce the following expression for the probability that a path is feasible [13]:

$$p(e) = p(l, level) = \begin{cases} [cosh(\lambda \frac{l}{2^{level}})]^{-2^{level}} & : \frac{l}{2^{level}} > eps \\ 1 & : \frac{l}{2^{level}} \leq eps \end{cases}, \quad (1)$$

where eps is the smallest distance, with which we want to perform collision checks along paths. Note that it is possible to chose eps very small, since it is unlikely that the dense resolution will be reached for any path except the solution path.

λ can be estimated experimentally using the fact that $E[N(l)] = \lambda l$. By changing the λ parameter it is possible to control the planners preference toward fast planning vs. short solution paths, but this feature is beyond the scope of this paper.

Learning phase A construction step inserts N randomly chosen collision free robot configurations as nodes into the fuzzy roadmap. It then connects each of the N new nodes to its M closest neighbors with edges. Each of these edges represents a path between two nodes. Unlike in the general PRM we do not run a local planner to verify the edge, instead we assign it an initial probability given by equation (1) with $level = 0$.

An expansion step (similar in nature to [9]) chooses K nodes and performs a random walk starting from

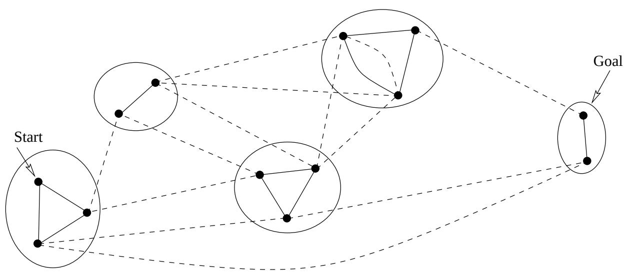


Figure 2: A manipulation graph. Solid lines represent transit paths and dotted lines represent transfer paths.

each of these nodes. Each of the K nodes are chosen randomly with a probability $p(v)$ equal to:

$$p(v) = \frac{kills(v)}{2 * total\_kills}, \qquad (2)$$

where $kills(v)$ is the number of edges starting at node v that were first added to the roadmap and later deleted during the query phase. $total\_kills$ is the total number of deleted edges. In case $total\_kills$ is 0 we skip the expansion step.

3 Fuzzy PRM for Manipulation

Our approach to manipulation planning is a two level method. In the first level we have a fuzzy roadmap which plays the role of a manipulation graph [3]. The purpose of the manipulation graph is to decompose the manipulation planning problem into multiple point-to-point path planning problems, which are solved by fuzzy point-to-point PRM planners at the second level.

Each node in the manipulation graph represents a landmark consisting of a triple containing a transformation matrix T_{obj}, a grasp number n_{grasp}, and a robot configuration vector q_{rob}. T_{obj} specifies a stable placement of the manipulated object, where the object can be left ungrasped by the robot. n_{grasp} is the number of a grasp relation of the object. A grasp relation specifies a legal position and orientation of the tool with respect to the placement of the object, at which the tool is able to grasp object. q_{rob} is a configuration vector for the robot, for which the robot tool is able to grasp the object placed at T_{obj} using the grasp specified by grasp relation n_{grasp}. The manipulation graph edges represents transit and transfer actions. Landmarks whose grasp relations are the same are connected by a transfer edge. Landmarks whose object positions are the same are connected by a transit edge.

Figure 2 shows a small manipulation graph. Note that two of the landmarks are connected by both a transfer and a transit edge. This is because the two landmarks share both the same object transformation and the same grasp number.

If we had complete path planners at the second level (as in [3]), we would be able to verify the existence of each of the transit and transfer paths in the manipulation graph during creation. We could the solve the manipulation planning problem by adding additional landmarks representing the start and goal configurations of the system, and search the manipulation graph for a path connecting the start and goal landmarks. Finally we could concatenate the individual transfer and transit edges along the path into the solution manipulation path.

A difficult problem is that we use only probabilistically complete planners at the second level. Probabilistically complete planners are unable to decide whether a given query is unsolvable: they will never terminate if queried for an unsolvable path. To use the above algorithm we need to decide on some maximal search time, such that if the second level planner uses more than this amount of time then we decide that the query is unsolvable. But a side-effect could be that difficult edges which are necessary in the solution path are wrongfully deleted from the roadmap.

We have chosen a different approach and we use a fuzzy roadmap to represent the manipulation graph. We let the edge probability of a given edge e depend on how much time the corresponding second level planner has spent searching for the path represented by e:

$$p(e) = 1 - \frac{time(e)}{total\_time}, \qquad (3)$$

where $time(e)$ is the time spent by the second level PRM searching for the path represented by e, and $total\_time$ is the total time spent by all second level planners.

The manipulation planner is again divided into a learning phase and a query phase, which are designed by analogy to the corresponding phases of Section 2, except that the manipulation graph is build during the an initial run of the learning phase.

The learning phase In the current version of the planner the user or the client software has to supply both T_{obj}, n_{grasp} and q_{rob} for each of the landmarks in the manipulation graph. It is possible to reduce this requirement to just T_{obj} by automatically calculating q_{rob} and n_{grasp} using inverse kinematics or a kinematic roadmap [1]. Choosing the stable object positions is a hard problem in itself. We currently rely solely on the user to add sufficient stable positions to make the problem solvable.

The landmarks of our roadmap are added one by one, and each landmark is connected to the other landmarks in the roadmap. If two landmarks have the same object configuration they are connected by a transit edge. If they have the same grasp relation they are connected by a transfer edge. Else, no edge is added. As in the Fuzzy PRM we do not verify the feasibility of the edges when they are created, instead all new edges are assigned an edge probability of 0.999 (since 1.0 means that the path has been found).

In case of a large number of landmarks it might be necessary to only connect each landmark to its M closest neighbors, chosen by some distance metric.

The query phase We start the query phase by updating the manipulation graph with the start and goal configuration of the system. If the object is ungrasped in either the start or goal configuration the corresponding landmark will have an undefined grasp number. In this case the node will only be connected to other nodes with the same object transformation by transit paths.

The local planners used during the upgrade step to find feasible transit or transfer paths for the individual edges along a manipulation path, are different instances of the fuzzy point-to-point PRM planner; One instance for each possible grasp of the object. These planners are used to plan the transfer paths. To upgrade a transfer edge with a particular grasp, we use the Fuzzy PRM associated with this grasp.

We could have included an instance of the fuzzy point-to-point PRM planner for each stable placement of the object, and then use these planners to plan the transit paths. Since the number of stable configuration is usually large (larger then the number of different grasps), and since it is usually much easier to plan a transit path than to plan a transfer path, we have chosen a different solution. We use a single fuzzy point-to-point PRM to solve all transit paths. We do this by associating a list of probabilities, one for each stable object position, with the edges, instead of just one probability. We choose the right entry in the list depending on the stable object placement associated with the transit path we are considering.

Edges which are assigned probability 0 for a given object placement are just hidden when we make a query for that placement. These edges are however kept in the roadmap until they are assigned probability 0 for all possible object placements. The probability of the edges in the manipulation graph is given by equation (3). From that equation it can be seen that an edge will never be assigned probability 0. Therefore we never delete edges from the manipulation graph. Suppose that the user or the client software has added sufficient nodes during the learning step such that there exist a path(s) from start to goal and at least one of these paths is actually feasible. Then our algorithm once in query phase, it will stay there until the manipulation problem is solved.

4 Experimental Results

The algorithm was implemented in C++ on a 400MHz Pentium II running Windows NT. Our program used 163 seconds on the average (10 subsequent runs) to solve the manipulation problem problem shown on Figure 3. This problem involves an L-shaped object and we initially had 7 stable placements of the object. The manipulation graph contained a total of 14 landmarks. λ was estimated to 0.22. The goal is to bring the object from the position shown at snapshot (a), where the object is behind the right obstacle, to the final position shown at snapshot (i), where the object is between the two obstacles on the left. Because of the large size and the strange shape of the object, the robot has to put the object down and regrasp it to complete the task.

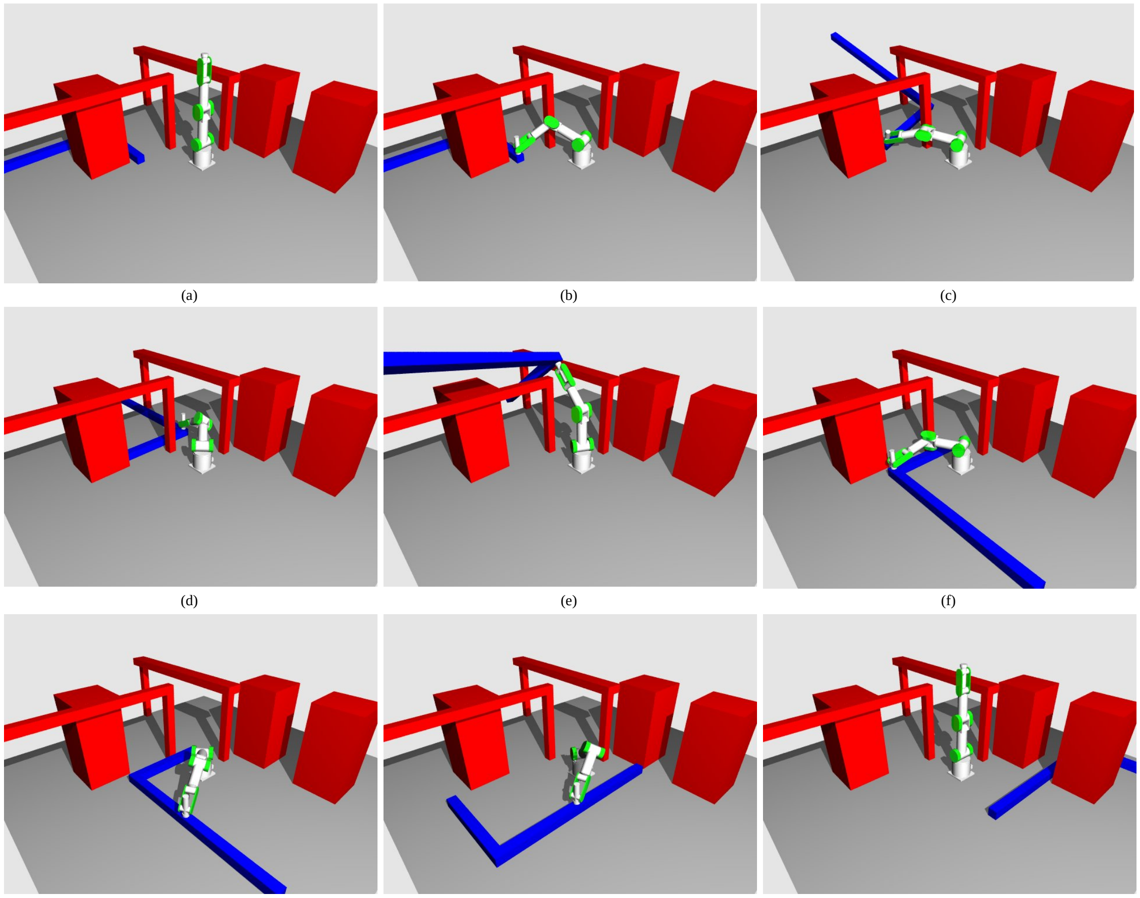
5 Discussion

We have successfully designed and implemented an algorithm capable of handling manipulation planning for a system consisting of a single redundant robot arm manipulating a single movable object with a finite set of stable placements. This was done by extending the PRM framework with an edge probability annotated roadmap, called a fuzzy roadmap.

Future improvements During the development of the current approach, we have discussed the possibility of moving the transfer paths from the second level PRMs to the first level manipulation graph. This will remove the time lost when querying the second level PRM for transfer paths that do not exist. Furthermore, in case we want to generate the stable object configurations automatically, it will be possible to deduce information from the manipulation graph about where in the workspace it would be advantageous to generate these configurations. This idea is under investigation.

Acknowledgements

This work was performed while Christian Nielsen visited the Physical Computing Laboratory at Rice University. Work on this paper by Lydia Kavraki has been supported in part by NSF CAREER Award IRI-970228 and NSF CISE SA1728-21122N. The authors would like to thank the rest of the Physical Computing Group at Rice University, Henrik Gordon Petersen, Lars Overgaard and Morten Lind Petersen at University of Southern Denmark for their comments and support.



(a) (b) (c) (d) (e) (f) (g) (h) (i)

Figure 3: An example of a solution path for the manipulation planning problem used in our experiments.

References

[1] J. M. Ahuactzin and K. Gupta. The kinematic roadmap. In *IEEE Transaactions of Robotics & Automation*, 1999.

[2] J. M. Ahuactzin, K. Gupta, and E. Mazer. Manipulation task planning for redundant robots: A practical approach. *Int. Journal of Robotic Research*, 17(7), 1998.

[3] R. Alami, J.P. Laumond, and T. Siméon. Two manipulation planning algorithms. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*, 1994.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3d workspaces. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*, pages 155–168, March 1998.

[5] P. Bessière, J. M. Ahuactzin, E.-G. Talbi, and E. Mazer. The "Aiadne's claw" algorithm: Global planning with local methods. In *IEEE/RSJ Conf. on Intelligent Robots and Systems*, 1993.

[6] R. Bohlin and L. E. Kavraki. A lazy prm for single query path planning. In *Proc. of the IEEE Int. Conf. on Robotics & Automation*, 2000.

[7] P. Ferbach and J. Barraquand. A penalty function method for constrained motion planning. Technical Report Research report 34, Paris Research Laboratory, Digital Equipment Corp, 1993.

[8] D. Hsu, L. E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. On finding narrow passages with probabilistic roadmap planners. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*, March 1998.

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

[10] Y. Koga. *On Computing Multi-Arm Manipulation Trajectories.* PhD thesis, Department of Mechanical Engineering, Stanford University, 1994.

[11] Y. Koga and J.-C. Latombe. On dual arm manipulation planning. In *Proc. of the IEEE Int. Conf. on Robotics & Automation*, 1994.

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

[13] C. L. Nielsen. Investigation of roadmap based methods in robot motion planning. Master's thesis, The Maersk Institute (MIP), University of Southern Denmark, 2000.

[14] M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In *Proc. of the Workshop on Algorithmic Foundations of Robotics*, 1994.