Analytical Potential Fields and Control Strategies for Motion Planning*

Seung-Woo Kim and Daniel Boley

Department of Computer Science and Engineering
University of Minnesota
Minneapolis, Minnesota 55455

Abstract. We present a novel method for robot motion planning that constructs a network of collision free paths using a randomized search over a potential field in Configuration Space. Our method finds local minima and then connects them to form a graph, which we call a roadmap. We use a gradient search scheme to find the local minima very efficiently and accurately. To find a path between two configurations, it is then a simple matter to connect given start and goal configurations to the roadmap and to use a standard graph search algorithm to search the roadmap. The construction of the roadmap can be done in parallel with very little communication.

1 Introduction and Background

Motion planning for multi-jointed robotic manipulators is an important area of research for which a large number of algorithms have been developed [11]. Application domains include industrial robots, teleoperation [12], and control of redundant robots [10]. However, the computational complexity of many of these algorithms can be quite substantial, severely limiting their use in practice [6]. The major objective of the present paper is to propose a novel method for computing paths for multi-jointed robot arms. The method proposed is fast even with a large number of degrees of freedom and can easily be parallelized.

Robot motion planning is based on the use of two spaces, the Workspace (W-space), which is the physical space through which the robot moves, and the Configuration space (C-space), which is the space of all robot configurations. The C-space has dimension equal to the number of degrees of freedom (*dof*) of the robot. Many algorithms operate by computing the set of infeasible configurations (obstacles) and then searching the remaining C-space for feasible paths. To make the methods tractable, various devices have been proposed in the past, including discretizing the C-space into cells [2], computing an artificial potential function to avoid collisions with obstacles or to find the goal [2, 7], and random search [8, 9]. Using a discretized C-space requires the pre-computation of a large number of potential values at the cells, and suffers from artifacts such as missing or

* This research was partially supported by NSF grants CCR-9405380 and CCR-9628786.

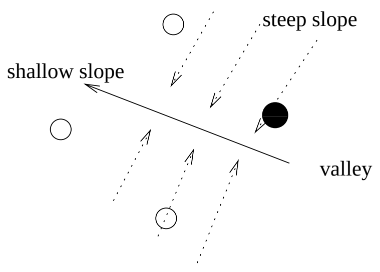

Fig. 1. Valley: arrows point down hill, circles mark cells, black is discrete minimum.

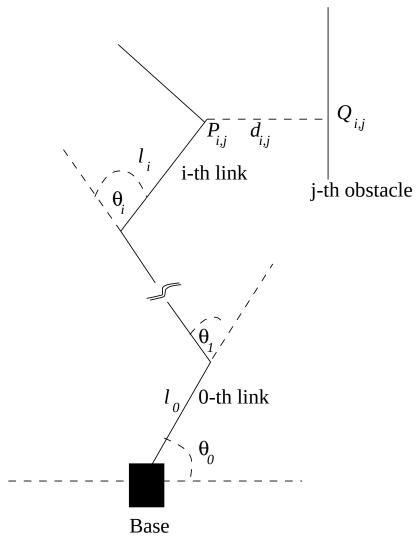

Fig. 2. Computing Potential Fields

spurious local minima over the cells that do not correspond to local minima of the continuous potential (e.g. fig. 1).

To obtain acceptable performance, some methods do a significant amount of preprocessing of the configuration space (C-space) [8], or place landmarks in C-space that are then used by a local planner [3, 5]. In Kavraki's approach [8, 9], a large number of randomly computed straight line collision-free partial paths are generated until the C-space is covered sufficiently, and the individual partial paths are connected into a single graph. This method requires a very large number of partial paths, and may have difficulties in traversing narrow passageways in the C-space. Other methods make assumptions on the type of robot (for instance, [1] takes advantage of the symmetry of the workspace with respect to the first axis of the robot), or use a coarse discretization of C-Space.

The method proposed here uses artificial potential fields and performs a substantial amount of computation during a preprocessing phase using a gradient based search algorithm for both descent and ascent, combined with a randomized search strategy.

1.1 Analytical Potential Fields

Analytical potential fields were used in [7] to allow the use of traditional gradient descent algorithms to find the goal and stay clear of obstacles. We use a similar idea in our work. A set of panels (line segments in 2D workspace) were used to represent obstacles. Each joint was represented by a point robot, and the potential was computed for each joint independently. The total potential was a linear combination of the individual potentials. The potential also included a "sink" at the goal to promote movement toward the goal.

In our work, we consider the entire robot as a point in C-space. At any given configuration (point in C-space), we compute the potential using repulsive forces between the obstacles and the robot in the workspace. This potential is computed between each link and each obstacle, both of which are represented by line segments in a 2D workspace (an obstacle is a panel in a 3D workspace). For every link-obstacle pair, there is a closest point $P_{i,j}$ on the link i and a corresponding closest point $Q_{i,j}$ on the obstacle j, and the distance between those points determines this pair's contribution to the overall potential value. This is illustrated in Figure 2. The details are given in section 2.1.

2 Current Work

We propose a method to build a collision free roadmap for a given static environment and an articulated robot. Once this roadmap is built, we attach any initial and goal configurations to this roadmap. Having attached these configurations to the roadmap, we can find a path between them extremely fast since the roadmap has been precomputed. The roadmap consists of nodes and edges which are collision free paths between adjacent nodes.

Rather than using thousands of random configurations for the nodes in the graph, we use the local minima for a potential field which is used to guide an articulated robot. We use an algebraic formula to compute the energy function which is differentiable with respect to joint angles in order to follow the negative gradient of the potential field. As mentioned above, we represent links as line segments and obstacles as line segments in 2D workspace and as flat panels in 3D workspace. An analytical approach is used in order to calculate clean local minima. By choosing the local minima as the nodes in the roadmap, we have been able to reduce the number of nodes under 1000, substantially less than in [8, 9]. A strategy to connect adjacent local minima to form the edges in parallel is also investigated. In our method, we (a) limit the situations where a random movement is necessary, (b) we use a potential field to guarantee collision-free paths, (c) we have a systematic strategy for climbing out of local minima to find a path to another, and (d) we use a Gauss-Newton direction to follow a shallow slope very accurately.

2.1 Basic Methods

Generating Potential Fields In order to compute the potential for a given configuration, Euclidean geometry is used to compute a pair of closest points for each link and obstacle segment. We denote those points as $P_{i,j}$ (on the link) and $Q_{i,j}$ (on the obstacle) as the closest points between the i-th link and the j-th obstacle segment. Then the distance between them is $d_{i,j} = \|P_{i,j} - Q_{i,j}\|$. If we put m = number of links, n = number of obstacles, and define $r_{i+j*m} = 1/d_{i,j}$, then the overall potential E for the configuration is as follows:

$$E = \frac{1}{2} \sum_{i=0}^{mn-1} r_i^2 \qquad (1)$$

This potential function enjoys several useful properties. (1) As the robot gets close to the obstacles, the potential becomes infinitely large. Therefore, it is collision free as long as it follows a gradient descent direction. (2) The gradient of the potential field can be derived directly from the above equation by an analytic formula. (3) Equation (1) is a nonlinear least squares problem for which there are many efficient and well-studied algorithms [4].

Computing the Gradient The gradient is computed directly from the above equation. Even though the link length l_i and closest obstacle point (p, q) change as the robot moves, they are fixed for the simplicity of the computation of the potential. Let us put $P_{i,j} = (x_{i,j}, y_{i,j})$ and $Q_{i,j} = (p, q)$. Also, the k-th joint is at (x_k, y_k). Then the gradient of the potential at $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{m-1})^T$ is

$$\nabla E(\boldsymbol{\theta}) = \boldsymbol{\mathcal{J}}^T \mathbf{r} \tag{2}$$

where $\boldsymbol{\mathcal{J}} \subseteq \mathbf{R}^{mn \times m}$ is the Jacobian matrix of $\mathbf{r}$, defined by $\boldsymbol{\mathcal{J}}_{i,j} = \frac{\partial r_i}{\partial \theta_j}$. The computation of $\boldsymbol{\mathcal{J}}$ is done as follows:

$$\begin{aligned}\boldsymbol{\mathcal{J}}_{i+jm,k} &= \frac{\partial r_{i+jm}}{\partial \theta_k} = \frac{\partial}{\partial \theta_k}\left(\frac{1}{d_{i,j}}\right) = \frac{\partial}{\partial \theta_k}\|P_{i,j} - Q_{i,j}\|^{-1} \\ &= \frac{\partial}{\partial \theta_k}\left[(x_{i,j} - p)^2 + (y_{i,j} - q)^2\right]^{-1/2} \\ &= -\frac{(x_{i,j} - p)\frac{\partial x_{i,j}}{\partial \theta_k} + (y_{i,j} - q)\frac{\partial y_{i,j}}{\partial \theta_k}}{\left(\sqrt{(x_{i,j} - p)^2 + (y_{i,j} - q)^2}\right)^3}\end{aligned} \tag{3}$$

$\frac{\partial x_{i,j}}{\partial \theta_k}$ is calculated as follows (see Fig. 2):

$$\begin{aligned}\frac{\partial x_{i,j}}{\partial \theta_k} &= \frac{\partial}{\partial \theta_k}\left[x_0 + \sum_{u=0}^{i} l_u cos(\sum_{v=0}^{u} \theta_v)\right] \\ &= -\sum_{u=k}^{i} l_u sin(\sum_{v=0}^{u} \theta_v) \\ &= (y_k - y_{i,j})\end{aligned} \tag{4}$$

Similarly,

$$\frac{\partial y_{i,j}}{\partial \theta_k} = x_{i,j} - x_k \tag{5}$$

Therefore,

$$\boldsymbol{\mathcal{J}}_{i+jm,k} = \frac{(x_{i,j} - p)(y_{i,j} - y_k) - (y_{i,j} - q)(x_{i,j} - x_k)}{\left(\sqrt{(x_{i,j} - p)^2 + (y_{i,j} - q)^2}\right)^3} \tag{6}$$

Computation of the terms in the above equation is straightforward and is already needed for collision checking. However, the dimension of the Jacobian matrix increases with the number of obstacles, resulting in higher computation costs.

Computing the Descent Direction The simplest way to obtain a descent direction is simply to follow the direction of the negative gradient (2). This yields the method of steepest descent. However it is well known that simple steepest descent, though robust, can be very slow [4]. We use the Gauss-Newton method [4] to speed up the process. The Gauss-Newton method is based on a sequence of linear approximations of $\mathbf{r}(\boldsymbol{\theta})$. If $\boldsymbol{\theta}_k$ denotes the current approximation, then a correction $\mathbf{p}_k$ is computed as a solution to the linear least squares problem

$$\min_{p} \|\mathbf{r}(\boldsymbol{\theta}_k) + \boldsymbol{\mathcal{J}}(\boldsymbol{\theta}_k)\mathbf{p}\|_2, \qquad \mathbf{p} \subseteq \mathbf{R}^m, \tag{7}$$

and the new approximation is $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{p}_k$, where α_k is a step length to be determined. This linear least squares problem is solved using the QR decomposition of $\boldsymbol{\mathcal{J}}(\boldsymbol{\theta}_k)$ [4]. One of the important properties of the Gauss-Newton direction is that if $\boldsymbol{\theta}_k$ is not a critical point, then $\mathbf{p}_k$ is a descent direction [4].

Determining the Step Length Because we want to find a path to the local minima, α_k must be determined in such a way to limit the maximum step movement of the robot as well as not to overshoot over the hills in $\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha_k \mathbf{p}$, where $\mathbf{p}$ is search direction obtained by any descent method mentioned above. At first, α_0 is set to $1/\|\mathbf{p}\|$ and the difference in energy is computed, $E(\boldsymbol{\theta}_{\mathbf{k}}) - E(\boldsymbol{\theta}_{\mathbf{k+1}})$. If it is negative, we divide it by two and repeat the process until it becomes positive. If it is positive at the first step, we multiply it by two until the maximum movement of each joint in workspace reaches a certain limit, or the difference becomes negative. Calculating the Euclidean distance in workspace is necessary because a small amount of rotation at the base results in a much larger movement at the tip than the same rotation in a joint near the tip.

Locating Local Minima In order to find a local minimum from any initial configuration, a combination of the methods stated above are used to take advantage of each method's particular advantages while mitigating each method's disadvantages. Steepest descent method works well to repel the robot from the obstacles. Gauss-Newton method may not be accurate where it is too close to the obstacle. Steepest Descent method is used first in order to place the robot relatively far away from the obstacles. Once it is away from obstacles, Gauss-Newton converges rapidly to the nearby local minimum. In most cases, this combination works well except a few special cases, where we must use random probing of nearby configurations to find a lower potential.

In our actual experimentation, we added more constraints to avoid self collision. The repulsive forces between the robot links are computed and added to Equation (1). These terms are treated similarly because they are also represented as a nonlinear least squares problem, and omitted for the brevity of the paper.

2.2 Building the Graph

Our goal is to build a connected graph whose vertices are all the local minima as well as possible way points in between. To do this, we use an upstream ascent process to climb away from a given local minimum and find a path to another local minimum. Our process continues to find new local minima, connecting them to the local minima just passed through to form a partial path in the overall graph, in such a way as to avoid doubling back to a local minimum already visited. This search process terminates when a local minimum is reached that is in a previously found partial path or an obstacle is encountered. A new local minimum is found by descending from a randomly chosen configuration, and a new partial path is constructed using the same upstream ascent process starting from this new local minimum.

Moving Upstream in the Potential Field The task here is to find a collision free path from one local minimum to another. This path will form part of a partial path, which will eventually be connected to other partial paths to form one overall connected graph encompassing all the local minima. Starting at a local minimum, we select a random coordinate direction to move along. While moving in this direction, we also move "laterally" to avoid obstacles. The basic step is summarized as follows:

1. Select a coordinate direction.
2. Follow the selected direction for a small distance.
3. Find the local minimum in the hyperplane normal to the chosen coordinate direction starting from the current configuration (the "lateral" movement).
4. If the hyperplane local minimum in step 3 has a potential value less than the potential at the previous hyperplane local minimum, we have found a "hill" from which we can use a normal descent method to find the neighboring overall local minimum.
5. Steps 2, 3 are repeated from the point in step 3 using the same selected coordinate direction to find new hills until an obstacle is encountered.

As the above process proceeds, we obtain a sequence of points on the "hills" connected by the path we just traversed. We continue the search from each "hill" to find the next "hill." From each "hill" we also use a descent method to find the nearby local minimum. All these local minima and "hills" are connected to form a partial path of the overall graph. If a local minimum already in the overall graph is encountered, then the partial path currently being constructed can be connected to the rest of the graph. When a partial path ends because an obstacle is encountered, a new random starting configuration and randomly selected coordinate direction is selected and a new partial path is constructed.

Parallel Implementation of Generating the Graph In order to construct the roadmap, we need to glue the generated partial paths together. Each partial path can be generated independently. The only communication required occurs when a partial path is complete and must be connected to the existing graph. Hence this process is quite suitable for parallel processing. We use master-slave scheme for the parallel implementation of constructing the graph. One master program is responsible for accepting partial paths and glueing the nodes and edges to the graph, and the slave programs keep generating and sending partial paths to the master program. Generating each partial path takes a few seconds for a single processor, while glueing a path to the graph takes on the order of milliseconds. The messages from the slaves are very short. Therefore we expect linear speedup for up to at least a few hundred processors.

A variation of the Graph Build method After a few minutes of generating and glueing the partial paths to the graph, there emerges a large graph consisting of more than 70% of all the nodes, while a few dozen other graphs have just a couple of nodes. If we detect this situation, it is much more efficient to switch to concentrating on connecting the small graphs to the largest one. However, if we switch prematurely, the resulting graph may be too sparse, making it more difficult to find paths when the input query is given. The appropriate switching percentage depends on the specific W-space configuration.

2.3 Finding a Path from Start to Goal Configuration

Once the roadmap is constructed, the start and goal configurations are attached to the graph by using the descent algorithm to locate their respective local minima. Once they are attached to the graph, we can use any standard graph search algorithm to find a path. We are using breadth-first search, whose memory requirements are feasible in our cases, given that we have on the order of a few hundred to one thousand nodes. We also tried iterative deepening depth-first search, but performance suffered due to the high branching factor.

3 Experimental Results

We are using SGI Challenge Clusters consisting of 3 Challenge L machines and 1 Challenge XL machine. Each Challenge L machine has 4 R10000 processors and Challenge XL has 8 R10000 processors. We used PVM3 (Parallel Virtual Machine) for the parallel processing.

In Table 1, each experiment was run 10 times. Along with the average value, standard deviation and worst case are shown. P represents the ratio of the largest graph vs the total number of nodes, at which we stop random generation and try to connect disconnected components. LM is the total number of nodes. AT_b is the graph build time, AT_s is the average time required to connect Figure 3(a) and Figure 3(h). AT_r is the average time to actually regenerate the path. It took about 7 minutes to compute the graph for Figure 3 with the switch-over percentage of 60%. The time for connecting the Figure 3(a) and Figure 3(h) to the graph after pre-processing was 2 seconds. Regenerating the path in W-space

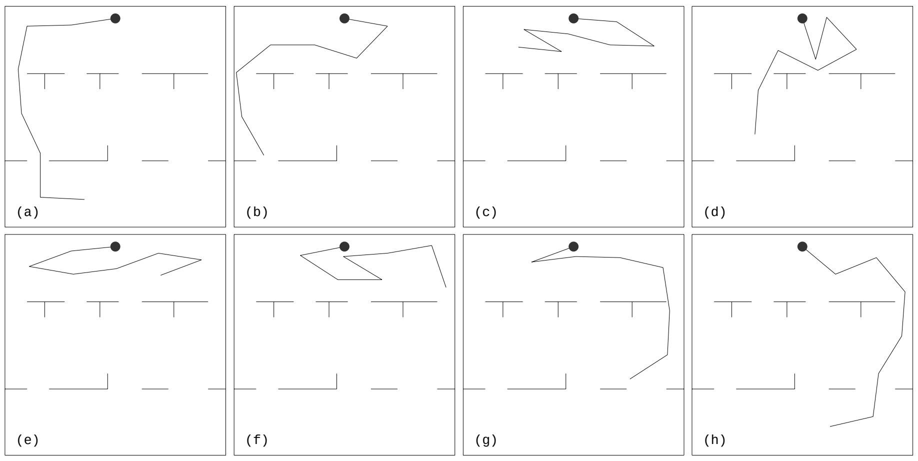

Fig. 3. a 7 *dof* robot example from start(a) to goal(h) configuration

P	LM	SD_{lm}	AT_b	SD_b	WT_b	AT_s	SD_s	WT_s	AT_r	SD_r	WT_r
ratio	*node count*		*graph build time*			*graph search time*			*path regen. time*		
30%	698	105	237	85	350	3	2	6	10	10	21
60%	954	162	463	144	634	2	2	5	8	6	15
80%	993	83	482	97	620	2	1	3	8	2	11

Table 1. performance for Figure 3 wrt switch-over ratio with 16 processors, SD =standard deviation, WT =worst case; times are in seconds.

took about 8 seconds because the partial paths are not stored when the graph is generated to reduce the memory requirement. If the partial paths were stored during pre-processing, the time required for regenerating the path would be 0. The unit of each column except LM is in seconds. SD and WT represents standard deviation and worst case respectively. For this example, the performance of actually finding the path is quite consistent because obstacles are rather evenly distributed in the W-space. Changing switch-over ratio directly affects the number of nodes generated and it also affects the path regeneration time. If the ratio is set too low, the resulting graph has fewer nodes and the path generated may not be optimal.

For the second example(Figure 4), the fastest pre-processing time for Figure 4 was 47 seconds when the switch-over ratio was set to 60%. However, the performance of finding the actual path varied more depending on where the input configuration is set because if the input configuration is placed where it's difficult to find by randomization, the corresponding local minimum may not be in the graph. In that case, the planner must search for the nodes that are in the graph from that point. Nevertheless, it never exceeded more than 10 seconds to connect to the graph because the algorithm's ability to connect local minima is

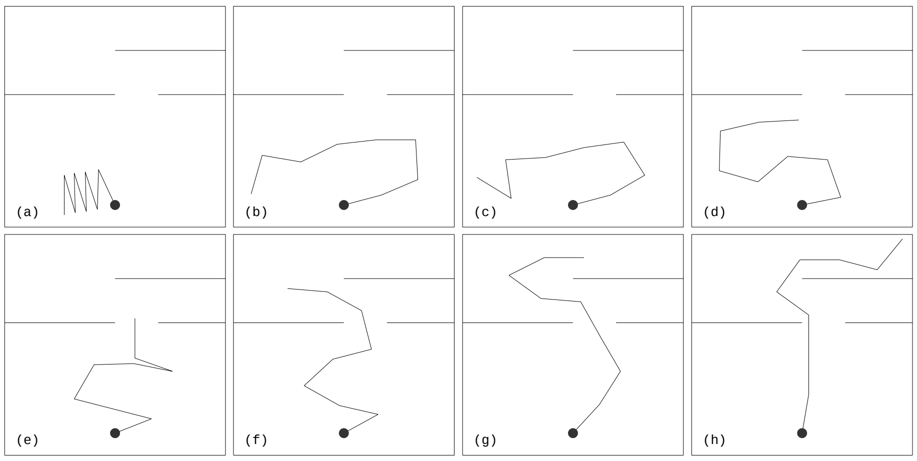

Fig. 4. an 8 *dof* robot example from start(a) to goal(h) configuration

P	LM	SD_{lm}	AT_b	SD_b	WT_b	AT_s	SD_s	WT_s	AT_r	SD_r	WT_r
ratio	*node count*		*graph build time*			*graph search time*			*path regen. time*		
60%	182	25	87	11	96	12	10	23	6	7	16
70%	193	26	93	14	105	11	8	22	5	3	10
80%	257	171	209	155	442	10	3	14	5	1	7

Table 2. performance for Figure 4 wrt switch-over ratio with 16 processors, SD =standard deviation, WT =worst case; times are in seconds.

NP	AT_b	SD_b	WT_b	SU	SC
no. procs	*graph build time*			*speedup*	*efficiency*
1	6684	913	7692	1.00	1.00
4	1750	280	2120	3.82	0.95
8	921	153	1100	7.26	0.91
16	482	97	620	13.87	0.87

Table 3. graph build time for Figure 4 wrt number of processors

very robust.

Table 3 shows the graph build time for Figure 4 with the varying number of processors used. The switch-over ratio was set to 80%. NP is the number of processors, SU represents the speed-up, and $SC = SU/NP$. The scalability is quite good as expected up to 16 processors. More processors will be tried to measure the scalability in the future.

4 Conclusion and Future Work

In this paper, we presented a novel method to combine randomized search and potential fields to solve motion planning problems in 2 dimensional W-space and high dimensional C-space. The experimental results show that this method is very efficient even for very difficult problems with a short period of preprocessing. The differentiable potential function and the upstream ascent strategies used in this paper are easily adaptable to 3D W-spaces. Hence we expect that our methods to be easily extended to a 3D workspace environment in a very straighforward way. This is focus of our future research.

References

1. P. Adolphs and H. Tolle. Collision-free real-time path-planning in time varying environment. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, pages 445–452, 1992.
2. J. Barraquand, B. Langlois, and J. C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-22(2):224–241, March/April 1992.
3. Pierre Bessiere, Juan-Manuel Ahuactzin, El-Ghazali Talbi, and Emmanuel Mazer. The Ariadne's Clew algorithm: Global planning with local methods. In *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 1993.
4. Å. Björck. *Numerical Methods for Least Squares Problems.* SIAM, 1989.
5. Pang C. Chen and Yong K. Hwang. SANDROS: a motion planner with performance proportional to task difficulty. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2346–2353, 1992.
6. Y.K. Hwang and N. Ahuja. Gross motion planning – a survey. *ACM Computing Surveys*, 24(3):219–291, 1992.
7. Jin-Oh Kim and Pradeep K. Kohsla. Real-time obstacle avoidance using harmonic potential functions. *IEEE Trans. Robotics and Automation*, 8(3), June 1992.
8. L. Kavraki. Randomized preprocessing of C-space for fast path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2138–2145, 1994.
9. Lydia Kavraki, J.C. Latombe, Petr Svestka, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Robotics and Automation*, 12(4):566–580, 1996.
10. Thierry Laliberte and Clement Gosselin. Efficient algorithms for the trajectory planning of redundant manipulators with obstacle avoidance. In *Proc. IEEE Int'l Conf. on Robotics and Automation*, pages 2044–2049, 1994.
11. J. C. Latombe. *Robot Motion Planning.* Kluwer Academic Publ., Norwell, MA, 1991.
12. V. Lumelsky and Edward Cheng. Real-time collision avoidance in teleoperated whole sensitive robot arm manipulators. *IEEE Trans. Systems, Man, and Cybernetics*, SMC-23(1):194–203, Jan/Feb 1993.