Merging the adaptive random walks planner with the randomized potential field planner

Stefano Carpin
School of Engineering and Science
International University Bremen
Germany

Gianluigi Pillonetto
Department of Information Engineering
University of Padova
Italy

***Abstract*— In this paper we investigate whether it is advantageous to merge some ideas formerly found in the randomized potential field planner with our recently introduced adaptive random walks planner. These aspects are biasing the generation of samples, an attractor for the samples generator, and the possibility to backtrack when the planner gets stuck while exploring the configuration space. We illustrate the numerical results of different experiments using these strategies one at the time, or combined together. It turns out that benefits of different amplitude can be obtained using them, but it is in general hard to incorporate these components in a general way independent from the problem instance to be solved.**

I. Introduction

Robot motion planning is one of the most widely studied aspects of robot algorithms and constantly benefits from a steady amount of research efforts. The reasons for this interest are both practical and theoretical. On the applied side, algorithms capable of efficiently determining the path a robot must follow to reach a certain goal position are required in industrial environments, as well as for mobile platforms used in research labs. On the theoretical side, the problem itself is intrinsically hard and calls for the development of algorithmic machinery beyond the classic tools. To spice up the scene, it recently became evident that robot motion planning algorithms can be used in many scenarios beyond robotics [11]. Computer graphics and computational biology [7] appear to be the most intriguing ones. Rather interestingly, these applications not strictly connected to robotics present problem instances which are usually much harder than robot specific ones. As it will be clarified later, the hardness of the problem at hand can be roughly estimated by the number of degrees of freedom of the system under study. While an industrial robotic manipulator has usually 6 degrees of freedom, problems from computational biology may have tenths or hundreds. The call for algorithms able to deal with many degrees of freedom is then still growing. Since the mid nineties [8], the paradigm dominating the motion planning scene has been the *randomized* or *sampling based*. According to this methodology, deterministic and exact algorithms doomed to incur in an exponential complexity are replaced by algorithms that build an approximated space representation based on random samples. These algorithms give up deterministic correctness for the weaker property of probabilistic completeness. This means that if a solution exists, the probability of finding it converges to 1 when the processing time tends to infinity. According to this paradigm, a great number of new algorithms have been devised, most notably [14], where an extremely fast algorithm able to deal with nonholonomic constraints was introduced.

In this framework we recently developed an algorithm for holonomic motion planning based on random walks [4],[6]. The algorithm explores the space of possible configurations using a random walk whose distribution's parameters are updated on the fly according to the recently generated ones. One of the appealing aspects of our approach is that for certain scenarios it is extremely fast, and it is almost parameters free. On the contrary, a tedious aspect of many motion planning algorithms is the large number of parameters to be tuned by hand. This activity is time consuming and requires significant expertise. The algorithm we developed, updates its few parameters while running, so that one has not to spend too much time to find good values. This updating is based on the recently accepted or discarded samples, which we call *history*. We also shown in [3] that history size is not a critical factor, i.e. there are pretty wide ranges yielding good results. In this paper we investigate whether the algorithm can be improved by introducing some algorithmic components already found in other algorithms formerly developed. In particular, in section II we will illustrate the adaptive random walks algorithm and we will relate it to the *randomized potential field* algorithm developed in the past [1]. In section III we discuss the ideas that can be borrowed from that planners and how they have been implemented. Section IV compares the results obtained by the different components implemented, and finally conclusions are offered in section V.

II. Formulation and motivation

From a formal point of view, the basic robot motion planning problem can be formulated as follows. Given a space of configurations $\mathcal{C}$ partitioned into the subsets $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$, and two points $x_s \in \mathcal{C}_{free}$ and $x_g \in \mathcal{C}_{free}$, determine a function

$$f : [0,1] \rightarrow \mathcal{C}_{free}$$

such that $f(0) = x_s$ and $f(1) = x_g$. The basic formulation of the problem is PSPACE hard and the best deterministic algorithm ever developed has time complexity exponential in

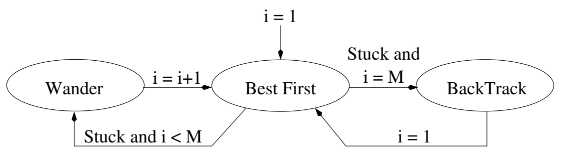

Fig. 1. The schematic behavior of the randomized potential field algorithm (this representation of the algorithm has been introduced in [13])

the dimension of the configuration space $\mathcal{C}$ (see [10] and [13] for comprehensive discussions of the subject). As anticipated in the introduction, the inherent complexity of the problem pushed the research community to develop algorithms based on randomization and sampling. One of the first algorithms using randomized components was presented in [1]. The algorithm performs a gradient descent search, and when it gets stuck in local minima it performs random motions to escape the potential well. Its overall behavior can be described with a state machine where three different states can be entered (see figure 1). Two states are used to distinguish the *gradient descend* and the *escape local minimum* behaviors. In figure 1 these are indicated as *best first* and *wander*, respectively. The third state is triggered when the algorithms gets stuck too many times. In this case the algorithm performs a backtrack, i.e. it restarts the gradient descent behavior from one of the points generated during the random motions to escape a local minimum. The devised algorithm performs very well for problems with many degrees of freedom, but needs some components to be tuned by hand. In particular, a good general purpose potential function is not immediate to determine. The adaptive random walk we recently developed can be seen as a special case of the randomized potential field planner. According to this analogy, the planner always stays in the *Wander* state, i.e. it keeps exploring the space with a random walk. Algorithm 1 illustrates the basic principle of the adaptive random walk algorithm. The exploration starts from x_s and continues until the goal is reached (note that rather than reaching a point, the algorithm stops when it reaches a region X_{goal} surrounding the goal point). One of the peculiarities of the algorithm is that samples are generated accordingly to a normal distribution (line 5) whose mean μ_k and covariance Σ_k are updated at every iteration (line 13). The update is based on the last accepted K samples. K is called *history size* and actually is the only parameter, though we already mentioned the algorithm's performance appears to be quite independent from it (provided it is not 0). This leverages the programmer from the task of parameters fine tuning. Paths generated by both ARW and by the randomized potential field planner are usually very jagged, but they can be smoothed very quickly (see formerly cited papers for details). The basic idea of the ARW algorithm can be greatly improved by using bidirectional search techniques and greedy strategies. Bidirectional search is a well known tool in artificial intelligence. Instead of using just a single random walk to explore the configuration space, two random walks are grown, one from the starting point x_s and one from the goal point x_g. Exploration terminates when

Algorithm 1 Basic ARW Motion Planner

```
1: $k \leftarrow 0$
2: $x_k \leftarrow x_{start}$
3: Arbitrarily initialize the covariance matrix $\Sigma_0$ and the mean vector $\mu_0$
4: while NOT $x_i \in X_{goal}$ do
5:     Generate a new sample $v_k \in N(\mu_k, \Sigma_k)$
6:     $s \leftarrow x_k + v_k$
7:     if the segment connecting $x_k$ and $s$ lies entirely in $\mathcal{C}_{free}$ then
8:         $k \leftarrow k + 1$
9:         $k_i \leftarrow s$
10:    else
11:        discard the sample $s$
12:    end if
13:    Update the covariance matrix $\Sigma_k$ and the mean vector $\mu_k$
14: end while
```

it is possible to join the two random walks, or when one of the two reaches its target point (x_g and x_s respectively). With the greedy strategy, when the segment connecting x_k and s (see line 7) does not completely lie in $\mathcal{C}_{free}$, it is not entirely discarded, but rather only the part in $\mathcal{C}_{obs}$ is eliminated. The above described techniques lead to a very competitive algorithm that can solve certain problems extremely quickly. It has of course to be mentioned that a malicious adversary could easily design a motion planning problem where the ARW algorithm will perform very poorly, but this is a problem common to most planners.
Based on the previously described algorithms, one could ask whether it is possible to combine them. Our overall goal is nevertheless to keep an algorithm which is parameters free as much as possible. In particular we believe that if parameters are introduced in the algorithm, they should be updated on the fly while the algorithm is being executed, so that a poor choice would not impact too much the performance. It is anyway necessary to point out that this goal is not always easy to obtain. We would also like to avoid designing complicated potential functions, because they are have to be environment oriented and their computation could impact the overall performance. In the next section we will discuss different strategies for implementing backtracking and behaviors like gradient descent in the context of the ARW planner.

III. Introducing Backtracking, Biased Components and Attractors

A. Backtracking

One of the main drawbacks experienced by the adaptive random walk algorithm is the presence of regions in the configuration space that have a narrow entrance. One could visually imagine exploring a long narrow corridor with a dead end, or entering a room with a very narrow door and no other exit. The random exploration process is likely to spend a lot of time before leaving such area. From a mathematical point of view this is not a problem, since convergence properties of the algorithm guarantee that if a solution exists it will be eventually found when the processing time diverges to infinity. Obviously from a practical point of view these situations should be quickly identified and a recover action be undertaken. The first problem is identifying the *stuck* situation. One way would be to monitor the last accepted samples and see whether they lie all in the same neighborhood. Obvious practical questions are: how many samples should be taken into consideration to determine if the planner is stuck or not? And how big should the neighborhood be? These would lead to the introduction of two further parameters that are environment dependent. A possible choice for the first parameter, could be K, i.e. the history size. K is already used to update sampling distribution's parameters and tells the algorithm how much of its recent processing it should take into consideration. We will indicate this number as S. The size of the neighborhood is even more problem dependent. In fact one should take into consideration that the different degrees of freedom may assume values in different ranges, treat differently rotations from translations and so on. To leave these issues out we adopt the following strategy. When a new sample is generated, the algorithm checks whether a straight line path can connect the last sample in the path with the newly generated sample (line 7 of algorithm 1). If this is not possible, the segment is not completely discarded, but rather the part lying in $\mathcal{C}_{free}$ is kept (see figure 2 for a simple bidimensional example). At every step we store a number telling us how much of the segment was accepted. For example if only half lies in $\mathcal{C}_{free}$ the value will be 0.5. We call this value *extension* at step i and we indicate it as e_i. To determine if the random walk is stuck, at each iteration the average of the last S values of e_i is computed, i.e.

$$\overline{e} = \frac{1}{S}\sum_{i=1}^{S} e_i.$$

We define the condition *being stuck* as $\overline{e} < T$, where T is a certain threshold. In fact T is a parameter, but its meaning is pretty qualitative and not dependent from the environment. For example, taking $T = 0.1$ means that we consider the random walk being stuck if in the last S attempts the average motion has been less than 10% of the tried distance. The parameter T in itself can be fixed or can be adjusted during the computation. When the random walk is stuck, the backtracking procedure is applied. The point to restart from is chosen randomly using a uniform distribution from the set of already accpted samples.

B. Biased components

As mentioned before, the randomized potential field planner uses a potential function, but we would like to avoid it. *Good* potential field functions are hard to design, environment dependent, and can add quite some overhead for their computation at each iteration. For these reasons, we have rather decided to introduce a *bias* in the sampling distribution. The

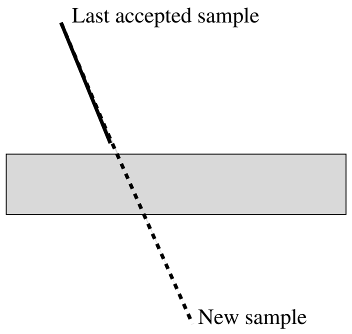

Fig. 2. In case the new sample cannot be connected with a straight line to the last accepted sample, because it intersects an obstacle (grey region in the middle). In this case only the fraction lying in $\mathcal{C}_{free}$ will be accepted (solid line). This would give a value for *extension* of about 0.35.

bias is handled as a non zero mean vector μ_k used in the sampling process (see algorithm 1). In fact, while the original ARW algorithm was developed using a zero mean vector, the overall convergence still holds with a non-zero mean vector (this was already somehow exploited in [5]). In case a single random walk is used to explore the configuration space, the obvious choice is to bias its mean towards the goal region. Anyway as we mentioned before, it is much more convenient using two random walks, one originating from x_s and the other from x_g. A path is found when either the two random walks meet each other or when the one originating from x_s can be connected to x_g (and viceversa). In this case one can bias the sampling process not only towards the final destination but also towards the other random walk. In fact, this approach takes into account that a solution can be found in different ways. In the experiments later illustrated we will compare both these choices. A practical aspect very important concerns the intensity of the biasing. How strong should this component be? Or, in other words, what is the ideal way to set the mean vector modulus? According to the framework developed in the original ARW algorithm we have chosen to use the covariance matrix Σ_k to modulate this intensity. In fact, Σ_k is updated at each iteration according to the last K accepted samples. The values on the main diagonal are the variances of the various degrees of freedom. These give a rough indication whether the random walk is exploring an open area or a cluttered narrow passage. In the former case variance values are big and then it makes sense to use a strong bias. In the latter case the values are small, and it is more appropriate to reduce the polarization effect. In addition, one could also think to apply the bias not always but only with a certain probability. The value of $\overline{e}$ could be used to take the decision. For example one could decide to introduce the bias with probability $\overline{e}$.

C. Attractors

In order to easily mimic the *best first* component of the randomzed potential field planner, *attractors* can be introduced. They could be used in alternative to the formerly discussed bias, or togheter. Informally speaking, an attractor is a sort of probability measure introduced over the configuration space

$\mathcal{C}$. According to this approach, when a sample is generated (line 6 of algorithm 1), it is accepted or refused with a certain probability. If the sample is refused, a new one is generated and the random selection is repeated. This probability depends on the sample location inside $\mathcal{C}$. In fact, as already envisioned in [2], one could think of spending more time to generate *good* samples, in order to reduce the number of calls to the collision checker, which is the really expensive component of sampling based motion planners. The concept of attractors is very well developed in dynamical systems theory, and benefits from a huge amount of formerly developed research. It then makes sense investigating whether it can be used in the ARW planner. On the other hand, we strive for not introducing complicated functions and additional parameters. In the preliminar experiments we will illustrate in the next section, the probability of accepting a new sample s is the following

$$e^{-\frac{dist(s,x_g)}{dist(x_s,x_g)}}$$

where *dist* is the distance between the two given points in $\mathcal{C}$. In this way, samples closer to the goal point x_g have a higher probability of being accepted.

IV. Experimental results

The effectiveness of the different enhancement has been tested using the MSL - Motion Strategy Library software [12]. MSL is general framework for developing, testing and comparing motion planning algorithms. It includes a number of benchmark problems with different hardness (different environments, single and multi-robot problems, holonomic and non-holonomic constrains and so on). Figures 3 and 4 show two of the motion planning problems included in the MSL distribution. By using a common underlying set of geometric subroutines, like the PQP collision detection algorithm [9], it is possible to perform fair quantitative comparisons between different algorithms. To compare the performance of the different versions, we compute the overall time spent to find the final solution, i.e. the smoothed one. We also logged the number of calls to the collision detector, which is the most time consuming subsystem and is independent from the host system load. Results discussion will be based in fact on this number.

We first run the formerly developed ARW algorithm, i.e. with no backtracking and no biasing. All over the experiments we always run a bidirectional search, since it is much faster. Also, for fairness of comparison we also always used the same formulas to update the covariance matrix Σ_k. Table I shows the results for the basic algorithm. In all the tables, displayed numbers are the average of 50 repeated executions. The first column displays the name of the benchmark, the second the number of degrees of freedom, the third the overall time spent (in seconds), and the fourth the number of call to the collision detector. In all the tables included in this section, we display with a bold font the number of calls to the collision checker performed by the best algorithm.

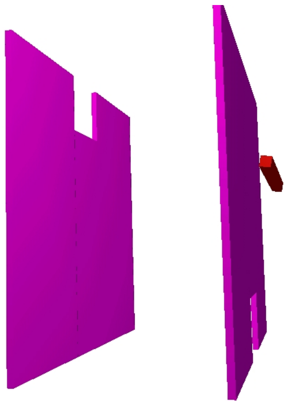

Fig. 3. The benchmark problem called *3drigid2*. The task is to move the object on the other side of the two obstacles. In order to pass through the two small openings some maneuvering is needed.

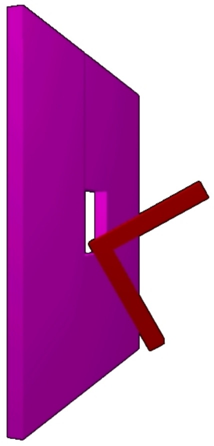

Fig. 4. The benchmark problem *3drigid3*. The task is to move the L-shaped object through the hole to the other side of the obstacle.

Problem	# dof	Time	# Calls
Car 1	3	0.167	3961
Car 2	3	0.833	10976
Wrench	6	2.728	22419
Cage	6	2.264	27467
3drigid2	6	14.468	197311
3drigid3	6	3.5632	44141
Cross	3	2.059	26986

TABLE I
Performance of the basic ARW algorithm

In the second set of tests we run the ARW algorithm introducing a biased component but without backtracking. The biasing is applied at every iteration and is not directed towards the goal state, but rather aims to connect the two random walks. Table II shows the results.

In the third test run we used a different biasing strategy, i.e. we biased the random walks towards their target points rather than towards each other. Results are shown in table III. Please note that no result is shown for the benchmark *3drigid2*

Problem	# dof	Time	# Calls
Car 1	3	0.183	4279
Car 2	3	0.762	12408
Wrench	6	1.747	20613
Cage	6	1.489	24622
3drigid2	6	9.7578	**169485**
3drigid3	6	2.212	36513
Cross	3	1.3454	24790

TABLE II
Performance of the ARW algorithm algorithm with a constant bias between the two random walks and no backtraking.

since the time spent is very high (two orders of magnitude of difference).

Problem	# dof	Time	# Calls
Car 1	3	0.3634	3490
Car 2	3	0.459	**4458**
Wrench	6	3.50	26366
Cage	6	2.457	27249
3drigid2	6	–	–
3drigid3	6	3.447	28777
Cross	3	1.98	**22276**

TABLE III
Performance of the ARW algorithm algorithm with a constant bias towards the target points.

We have then run the ARW algorithm introducing a backtracking recover behavior but with no biasing. In this case we have set the threshold T to the value 0.1. The value of $\overline{e}$ was computed taking into consideration the last 20 samples (i.e. $S = 20$). Results are displayed in table IV.

Problem	# dof	Time	# Calls
Car 1	3	0.1148	3532
Car 2	3	0.757	13906
Wrench	6	2.065	25752
Cage	6	1.755	29305
3drigid2	6	15.8282	313213
3drigid3	6	3.124	53837
Cross	3	1.917	35359

TABLE IV
Performance of the ARW algorithm algorithm with backtracking and no bias.

This set of tests was concluded including both extensions, i.e. we have run the ARW algorithm with both the backtracking behavior and the bias towards between the two random walks being built. Table V shows the results

The next set of tests aimed to evaluate the utility of attractors. Again, we first run the ARW algorithm using only the attractor strategy described in the former section. Table VI shows the results we obtained.
As we did for the bias, in the subsequent test we have put together both the attractor and the backtracking component. Table VII presents the results we obtained.

Problem	# dof	Time	# Calls
Car 1	3	0.151	2428
Car 2	3	1.618	17832
Wrench	6	1.79	18204
Cage	6	1.312	19662
3drigid2	6	80.387	892379
3drigid3	6	1.643	**21546**
Cross	3	2.863	35666

TABLE V
Performance of the ARW algorithm algorithm with backtracking and bias between the two random walks.

Problem	# dof	Time	# Calls
Car 1	3	0.239	4759
Car 2	3	0.671	8726
Wrench	6	2.624	22250
Cage	6	2.028	24123
3drigid2	6	16.059	215067
3drigid3	6	4.384	53800
Cross	3	2.146	27541

TABLE VI
Performance of the ARW algorithm algorithm with the addition of an attractor.

Problem	# dof	Time	# Calls
Car 1	3	0.216	4211
Car 2	3	0.832	10559
Wrench	6	3.211	25542
Cage	6	2.245	25839
3drigid2	6	16.697	218113
3drigid3	6	5.036	59581
Cross	3	2.191	27819

TABLE VII
Performance of the ARW algorithm algorithm including both an attractor and a backtracking component.

In the very last test, we joined together all the three proposed techniques, i.e. backtracking, bias and the attractor. The results are included in table VIII (again, no result is displayed for the benchmarck *3drigid3* because of the very poor performance).

Problem	# dof	Time	# Calls
Car 1	3	0.226	**2242**
Car 2	3	1.441	10080
Wrench	6	2.822	**18080**
Cage	6	1.740	**17616**
3drigid2	6	–	–
3drigid3	6	2.874	25059
Cross	3	3.013	28773

TABLE VIII
Performance of the ARW algorithm algorithm including a backtracking component from random position, a constant bias between the two random walks and an attractor.

V. Lesson learned and conclusions

In this paper we have discussed how we borrowed some ideas developed in the randomized potential field planner and how we have integrated them into the adaptive random walk planner. The integration between the two is somehow natural, since the latter can be seen as a special case of the former. In particular, we introduced the capability to backtrack when the random walk gets stuck, and a bias in the sampling distribution to mimic a sort of attractive behavior. All over the process we tried to introduce as few parameters as possible. This because in our opinion one of the attractive characteristics of the ARW algorithm is the very low number of parameters that have to be tuned by hand. It turned out that the proposed extensions give benefits in some benchmark problems, but none of them show a gain over all the investigated benchmarks. This can somehow be explained in two ways. The ARW algorithm performs a purely random exploration. As known [15], this behavior reduces the possibility that a *malicious adversary* could design a problem instance that negatively affects the algorithm, though it is still possible. The three added components, instead have a two fold aspect. On the one hand there exist some problems where they really pay off. Backtracking, is obviously useful if the configuration space exhibit many dead ends. As we pointed out in the introduction, this is one of the difficult scenarios for the ARW algorithm. On the other hand, finding the solution for certain problem instances indeed requires getting through narrow passages where many proposed motions will result in very short steps. In this case the backtracking component will restart the algorithm many times, even if it is nevertheless necessary to pass through that region. In this case, environment specific knowledge would be needed in order to set a good value for the threshold T. Biasing the sampling distribution and using attractors have more or less the same nature. If there are widely open areas, they clearly boosts the exploration process. On the other hand, if the solution involves finding a path taking a long detour from the obvious straight line trajectory, they have a detrimental effect, as they pushes the random walk very often into $\mathcal{C}_{obs}$. When used in combination with the backtracking component, the negative effect can even be overemphasized, since this can trigger backtracking when not needed. It should nevertheless be outlined that for each of the studied benchmarks the use of the proposed extensions gave benefits when comparing with the basic ARW algorithm.

In the future we plan to investigate whether it is possible to determine on-the-fly which technique should be used and which one should not. In fact it is acknowledged that there exists no *best planner*, but rather each one has its own strengths and weaknesses. It is our goal to embed different techniques, even beyond the above described ones, and to develop a selection mechanism that chooses on the fly which one should be included and which one should not. We do indeed believe that these adaptive subsystems will be very useful in the process of developing parameters free motion planners.

References

[1] J. Barraquand and J.C. Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[2] V. Boor, M.H. Overmars, and A.F. van der Stappen. The gaussian sampling theory for probabilistic roadmap planners. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1018–1023, Detroit, May 1999.

[3] S. Carpin and G. Pillonetto. Learning sample distribution for randomized robot motion planning: role of history size. In *Proceedings of the 3rd International Conference on Artificial Intelligence and Applications*, pages 58–63. ACTA press, 2003.

[4] S. Carpin and G. Pillonetto. Robot motion planning using adaptive random walks. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 3809–3814, 2003.

[5] S. Carpin and G. Pillonetto. Centralized multi-robot motion planning: a random walks based approach. In F. Groen, N. Amato, A. Bonarini, E. Yoshida, and Ben Krose, editors, *Intelligent Autonomous Systems 8*, pages 610–617. IOS press, 2004.

[6] S. Carpin and G. Pillonetto. Motion planning using adaptive random walks. *IEEE Transactions on Robotics*, 21(1):129–136, 2005.

[7] Greg Chirikjian, Nancy Amato, and Lydia Kavraki (Eds.). Special issue: Robotics techniques applied to computational biology. *International Journal of Robotics Research*, 2004–to appear.

[8] 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, 1996.

[9] E. Larsen, S. Gottschalk, M.C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical Report TR99-018, Department of Computer Science, University of N. Carolina, Chapel Hill, 1999.

[10] J.C. Latombe. *Robot Motion Planning*. Kluver Academic Publishers, 1990.

[11] J.C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *The International Journal of Robotics Research - Special Issue on Robotics at the Millennium*, 18(11):1119–1128, 1999.

[12] S.M. LaValle. Msl - the motion strategy library software, version 2.0. http://msl.cs.uiuc.edu.

[13] S.M. LaValle. *Planning Algorithms*. Available Online.

[14] S.M. LaValle and J.J. Kufner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, 2001.

[15] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995.