Robot Motion Planning Using Adaptive Random Walks

Stefano Carpin[†]
School of Engineering and Science
International University of Bremen
Germany
s.carpin@iu-bremen.de

Gianluigi Pillonetto
Department of Information Engineering
The University of Padova
Italy
giapi@dei.unipd.it

***Abstract*—We propose a novel motion planning algorithm based on adaptive random walks. The proposed algorithm turns out to be easy to implement and the solution it produces can be easily and efficiently optimized. Furthermore the algorithm can incorporate adaptive components, so that the developer is not required to specify all the parameters of the random distributions involved, and the algorithm itself can adapt to the environment it is moving in. Proofs of the theoretical soundness of the algorithm are provided as well as implementation details. Numerical comparisons with well known algorithms illustrate its effectiveness.**

I. Introduction

In the last years the problem of robot motion planning received further attention as a consequence of two possibly synergistic events. The massive introduction of randomized motion planners following the basilar work presented in [9] allowed to effectively face problems with a high number of degrees of freedom, not always solvable by formerly developed planners. Moreover, motion planners are being used for a number of applications that are beyond traditional robotics, like digital character motion generation, bioinformatics and many others (see [13]). The randomized approach appears to be the current main stream of research in motion planning and originated a wave of new algorithms which allows to study a wide variety of different problems. Indeed probabilistic planners are also being used to solve problems involving kinodynamic constraints ([15]). Recently there has also been an impulse to study algorithms not only probabilistic complete, but also resolution complete and to speculate on a possible turn back to efficient deterministic approaches ([4, 7]). The continuous flow of innovative tools for efficiently searching configuration state spaces confirms that as efficient planners are introduced, the frontier of applications is also being pushed further. Think for example at the problem of multi-robot motion planning ([6]), or protein folding and ligand binding, where problem instances with hundreds of degrees of freedom are common and are being tackled with randomized motion planners ([16]).
In this context we developed a new motion planner based on random walks. Preliminary results illustrate that, in spite of the widespread believe that random walks poorly perform in robot motion planning problems, by carefully choosing the random distributions involved it is possible to efficiently solve difficult problems with a good performance. In addition to that, the planner is well suited for the introduction of adaptive components, to let it adjust its random components in order to better address the environment where it moves. The paper is organized as follows. Section II illustrates the algorithm and section III provides details about simulations and numerical results. The theoretical soundness of the algorithm is sketched in IV, while conclusions are offered in section V.

[†]with the Department of Information Engineering of the University of Padova while doing this work

II. The Random Walk Algorithm

For clarity we formalize the motion planning problem we wish to solve. We are given an n-dimensional space of configurations C and let C_{free} be the subset of free configurations of C. Let $x_{start} \in C_{free}$ and let $X_{goal} \subset C_{free}$. Our goal is to find a path connecting x_{start} with X_{goal}, i.e. a continuous function $f : [0,1] \rightarrow C_{free}$ such that $f(0) = x_{start}$ and $f(1) \in X_{goal}$.
Most of the randomized planners used so far uses uniform sampling over the entire configuration space (see however [3] for an example of planner using a Gaussian distribution). Instead, the algorithm we propose tries to find a solution by building a random walk growing from x_{start}. At every step a new sample is generated in the neighborhood of the last point in accordance with a Gaussian distribution[1] and if the segment connecting them lies entirely in C_{free}, the point becomes the last point in the walk, otherwise it is discarded. The basic version of the algorithm is illustrated in algorithm 1.

A crucial point of the algorithm consists of establishing the probability distribution used to generate samples. For this aim, the v_k used in line 4 indicates a n-dimensional Gaussian vector with zero$-$mean vector and covariance matrix Σ_k. As the search of the path is carried out in a possibly high dimensional configuration space whose topological shape is usually unknown, it can be non trivial to fix the values of Σ_k. For this reason it is possible to let the algorithm start with arbitrary values of Σ_k and let them evolve while it runs. As it will be shown

[1]while using both uniform and Gaussian distribution yields effective planners, we will concentrate our discussion exclusively on Gaussian distributions. This because, in addition of exhibiting better performance and being more suited for adaptivity, it allows easier proofs of the probabilistic convergence of the algorithm (see section IV)

Algorithm 1 Basic Random Walk Based Motion Planner

```
1:  $k \leftarrow 0$
2:  $x_k \leftarrow x_{start}$
3:  while NOT $x_k \in X_{goal}$ do
4:      Generate a new sample $s \leftarrow x_k + v_k$
5:      if the segment connecting $x_k$ and $s$ lies entirely in $C_{free}$ then
6:          $k \leftarrow k + 1$
7:          $x_k \leftarrow s$
8:      else
9:          discard the sample $s$
10:     end if
11:     Update the covariance matrix $\Sigma$
12: end while
```

in section IV the conditions required for the probabilistic convergence of the algorithm are very mild so that a wide variety of update rules can be used while setting the values. The adaptive rule we are using is the following (see [8]). Let

$$\overline{x}_k = \frac{1}{H}\sum_{i=k-H}^{k-1} x_i \tag{1}$$

be the average of the last H accepted samples. Given a square p-dimensional matrix M let m_{ij} be its generic element in position i, j. We define $diag(M)$ as follows

$$diag(M) = \begin{bmatrix} m_{11} & 0 & \cdots & \cdots & 0 \\ 0 & m_{22} & 0 & \cdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \vdots \\ 0 & \cdots & 0 & m_{p-1,p-1} & 0 \\ 0 & \cdots & \cdots & 0 & m_{pp} \end{bmatrix} \tag{2}$$

i.e. the matrix obtained from M by setting to 0 all the elements outside the main diagonal. The update rule for Σ_k is then

$$\Sigma_k = \max\left(diag\left(\frac{1}{H}\left(\sum_{i=k-H}^{k-1} x_i x_i^T - H\overline{x}_k\overline{x}_k^T\right)\right), \Sigma_{MIN}\right) \tag{3}$$

where the function $\max$ returns a matrix whose generic element is the greatest of the corresponding elements in the two argument matrixes, and Σ_{MIN} is a diagonal constant matrix with strictly positive elements on the main diagonal. It then follows that Σ_k is diagonal too. In this preliminary stage, the choice of working with diagonal matrixes, i.e. to ignore correlations, has been done to get a simple implementation, but as illustrated in section IV the probabilistic completeness of the algorithm is guaranteed under less restrictive hypothesis. By dealing with diagonal matrix we get that the variance of Gaussian random vector used is the the variance of the last H accepted samples. Of course choosing a good value for H is important for getting a good algorithm performance and is definitely a point which needs more investigation.

One of the most important aspects of the random walk algorithm is that its performance is linear in the size of the number of samples generated, i.e. the time spent to generate a new sample is always the same and does not depend on the size of the previously generated samples set. This is different from most of the formerly developed algorithms, like probabilistic roadmaps (PRM) and rapidly exploring random trees (RRT), where the performance is usually quadratic in the number of generated samples (see however [1] for a version of the RRT algorithm where the time needed to generate a new sample is logarithmic in the number of already generated samples).

A natural technique to speed up the termination process is to let the algorithm perform a bidirectional search, with two random walks being expanded, one growing from the start point and the other from the goal region and to periodically verify if it is possible to join them. In order to avoid the examination of all the points, the connect trial is performed only between the last two generated points in the two walks. As already observed, the expedient of bidirectional search allows a substantial gain in the performance (see [11]). In addition to this, further opportunism can be introduced by letting the algorithm to periodically try to connect the last sample with the goal point (or the start point if in a bidirectional search we consider the walk growing towards the start point).

When the algorithm terminates, i.e. when the last generated point is in X_{goal}, the sequence of segments connecting x_i with x_{i+1} indeed makes up a path which solves the problem. However the quality of such path is extremely poor, because it includes a wide number of useless motions. Indeed the path obtained resembles a Brownian motion. For this reason a postprocessing stage is needed in order to smooth the generated trajectory. We use a *divide and conquer* algorithm similar to binary search (see algorithm 2). The *pushback* operations used therein append the given point to the end of the list.

Algorithm 2 Solution Smoothing

```
1:  SMOOTH($D, first, last, S$)
2:  INPUT: $D$ vector of points to smooth
3:  INPUT: $first, last$ extremes of $D$ to be optimized
4:  INPUT/OUTPUT: $S$ list with the smoothed sequence
5:  if $first = last$ then
6:      $S.pushback(D[first])$
7:  else if $first = last - 1$ then
8:      $S.pushback(D[first])$
9:      $S.pushback(D[last])$
10: else if the segment connecting $D[first]$ and $D[last]$ lies entirely $C_{free}$ then
11:     $S.pushback(D[first])$
12:     $S.pushback(D[last])$
13: else
14:     SMOOTH($D, first, (first + last)/2, S$)
15:     SMOOTH($D, (first + last)/2 + 1, last, S$)
16: end if
```

The smoothing procedure is iterated until it is not able to further smooth the trajectory. It is worth noting that due to the good performance of the algorithm itself, the smoothing step turns out to be extremely fast and few iterative steps are needed. This will be clearly illustrated in section III.

III. Simulation Details and Numerical Results

The proposed algorithm has been developed and integrated into the MSL software developed at the university of Illinois ([14]), in order to compare it with state of the art motion planning algorithms over a set of standard problems. The MSL includes a wide range of variants of RRT based motion planners as well as the basic PRM motion planner. Many of the predefined problems included in the MSL involve complicated three dimensional objects moving in difficult environments (see figure 1 for an example). MSL performs collision detection using the PQP library developed at the University of North Carolina ([12]).

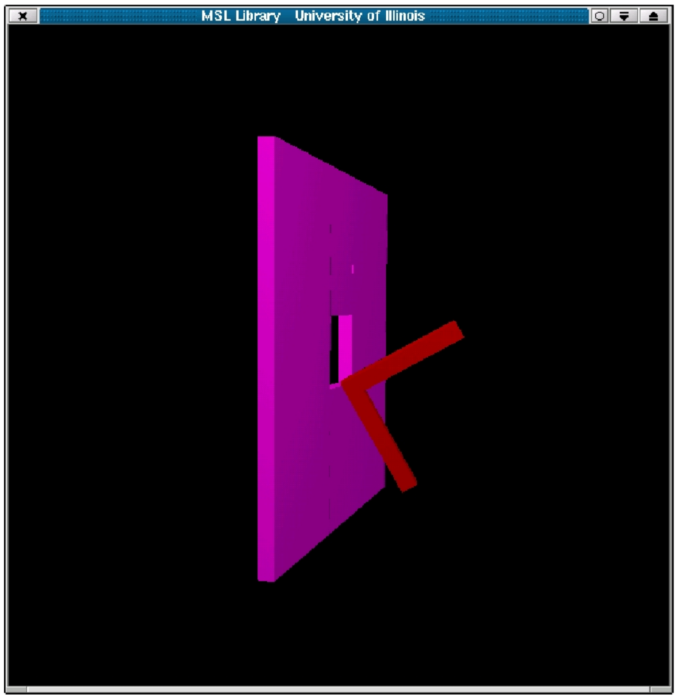

Fig. 1. Getting the L-shaped object from one side to the other side of the barrier is one of the many example problems provided with the MSL software. This example is called 3drigid3.

We first analyze the performance of the various steps of the proposed algorithm. All the numerical results concerning the random walk planner illustrated in this section refer to the bidirectional opportunistic planner previously described. The computer used is a Sun Ultra 10 workstation, working at 450 Mhz and with 256 Mbytes of RAM. History size for adaptivity, i.e. H, has been fixed to 10, while the square roots of diagonal values of Σ_{MIN} are set to one fifth of the difference between the maximal and minimal values which can be assumed by the corresponding degree of freedom.
Table I reports the data relative to 9 of the standard environments provided with the MSL. For lack of space we can not give an indepth description of every environment, but we use the same names given in the MSL, so the reader can refer to its documentation. We just state that the first 4 examples involve the model of a car with 3 degrees of freedom moving in a 2 dimensional environment (see figure 2), while the others refer to three dimensional objects moving in 3 dimensional environments (with 6 degrees of freedom). Column T_1 is the time spent to find the solution (seconds) and column N_1 is the number of samples in such solution, while column T_S is the time spent to smooth the solution and N_S is the number samples in the smoothed solution. T_{tot} is the sum of T_1 and T_s.

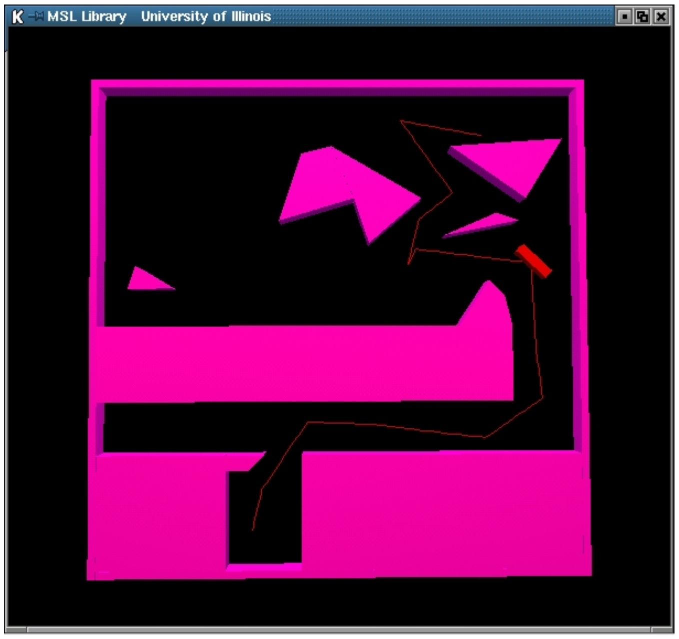

Fig. 2. Car2, one of the environments involving a car moving in a 3d environment

Environment	T_1	N_1	T_S	N_S	T_{tot}
Car 1	0.106	91.81	0.021	8.34	0.127
Car 2	1.419	311.92	0.154	16.14	1.566
Car 3	0.080	25.46	0.039	7.82	0.119
Car 4	0.013	22.24	0.006	7.76	0.019
Cage	26.54	512.84	2.22	14.32	28.76
Wrench	9.10	138.46	2.37	23.3	11.46
3drigid 1	0.236	896	0.133	5.94	0.369
3drigid 2	52.17	3940.7	1.36	15.27	52.53
3drigid 3	104.02	8247.2	0.453	7.15	104.47

TABLE I
Time spent in the various steps of the random walk algorithms. Data are averaged over 100 trials.

Two aspects are evident. First, the time spent by the algorithm is linear in the size of the generated random walk (compare columns T_1 and N_1). Second, even if the path produced by the random walk includes a great number of useless motions, significant improvements can be gained with the very fast post processing algorithm illustrated. We then compare the random walk algorithm with the basic PRM motion planner. Table II compares the overall time spent by both algorithms to find the solution. Again, time is expressed in seconds and data have been averaged over 100 trials.

It is well known that better versions of PRM exists (like lazy PRMs, see [2]), but it is however evident that adaptive random walks are indeed competitive. Finally we compare the perfor-

Environment	PRM	Random Walk
Car 1	0.369	0.127
Car 2	2.643	1.566
Car 3	0.858	0.119
Car 4	0.293	0.019
Cage	3442.76	28.76
Wrench	2526.26	11.46
3drigid 1	566.37	0.369
3drigid 2	871.96	52.53
3drigid 3	917.05	104.47

TABLE II
COMPARISON BETWEEN PRM AND RANDOM WALKS

mance of the random walk planner with the RRTConCon algorithm provided in the MSL. RRTConCon is an RRT bidirectional greedy planner based on RRT ([11]). It has to be pointed out that in this case we compare the performance just for the last 5 problems which are holonomic problems. In fact, while RRTs are well suited for kinodynamic motion planning problems, the current version of the planner we are proposing does not handle kinodynamic contraints. Further investigation on this aspect is needed. So it is not fair to compare them over the examples involving the car, which is nonholonomic, as either our algorithm ignores the kinodynamic constraints or RRTs perform poorly when ignoring such constraints. Instead, while comparing random walks with PRM the comparison has been possible since they both ignore kinodynamic constraints.

Environment	RRT	Random Walk
Cage	6.29	28.76
Wrench	3.46	11.46
3drigid 1	0.25	0.369
3drigid 2	69.01	52.53
3drigid 3	89.55	104.47

TABLE III
COMPARISON BETWEEN RRTCONCON AND RANDOM WALKS

In this case we can observe that in some cases RRTs are better, but there also exists environments where the opposite is true or where their performance is comparable. Of course no definitive conclusion can be drawn from a restricted set of examples. Moreover both algorithms have been run without trying to find out the optimal values for the many parameters involved.
We instead wish to outline that in spite of the wide skepticism against random walks, interesting performance emerges. Furthermore, by adding adaptive components it is possible to get a variable sampling resolution, which is indeed believed to be one of the most promising research direction. It should be outlined that in the case of single direction search adaptive random walks outperform RRT based motion planners. Finally, figure 3 illustrates an example of trajectory smoothing. It can be observed that even if the initial trajectory is extremely uneven, algorithm 2 succeeds in cutting out almost all useless motions.

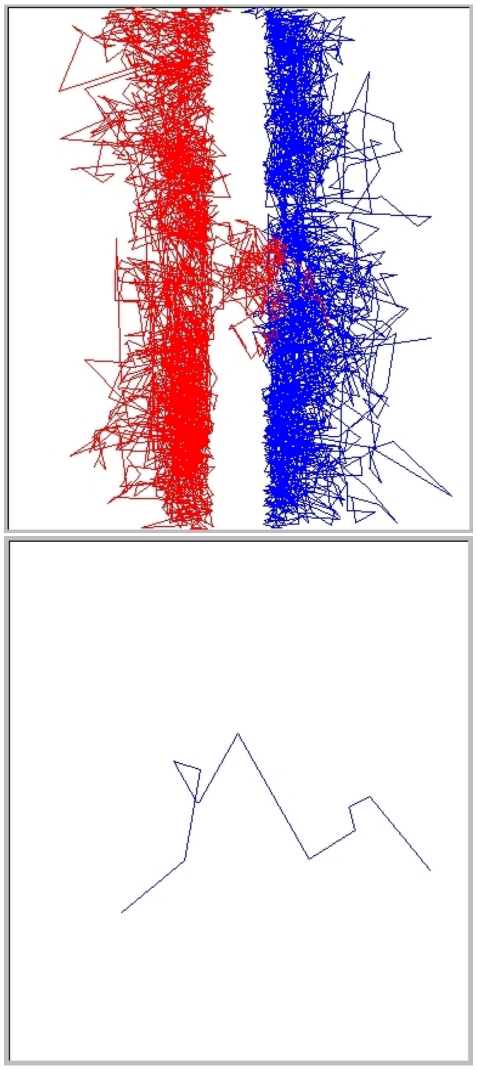
Fig. 3. Planned path before and after smoothing

IV. THEORETICAL FOUNDATIONS

In this section we provide the formalism and the proofs about the probabilistic convergence of the random walk algorithm introduced in the former sections. For lack of space not all the proofs are provided herein. The interested reader is referred to [5].

A. Preliminaries

We start defining a probability space as the triplet (Ω, Γ, η) where Ω is the sample space, whose generic element is denoted ω, Γ is a $\sigma - algebra$ on Ω and η a probability measure on Γ. Let C be $[0, 1]^n$, whose generic element is denoted x and equipped with the $\sigma - algebra$ $B(C)$ consisting of all the Borel sets in $\mathbb{R}^n$ which are contained in C. We will also denote with $\mu(A)$ the Lebesque measure of a generic set A in $B(C)$. Moreover, we will denote $N_{x,\Sigma}(y)$ the Gaussian probability density as function of y, having mean x and covariance matrix Σ.

Definition 1: We define $B^+(C)$ as corresponding to the sets in $B(C)$ whose measure is strictly positive with respect to μ. Hereby, we will denote with C_{free} a fixed open set belonging to $B^+(C)$.

Definition 2: We call $\{x_1, x_2\}$ an admissible couple if $tx_1 + (1-t)x_2 \in C_{free}$ for $t \in [0,1]$.
We end this sub-section by introducing the function $g : C \times C \to C$ such that:

$$\begin{cases} g(x_1,x_2) = x_1 + x_2 & \text{if } \{x_1, x_1+x_2\} \\ & \text{is an admissible couple} \\ g(x_1,x_2) = x_1 & \text{otherwise} \end{cases}$$

B. Adaptive Random Walk algorithm

We define the discrete time stochastic process $\{X_k\}_{k=0,1,2,\dots}$ on (Ω, Γ, η) and taking values on C, by the following recursive formulas:

$$\begin{cases} X_0(\omega) = x_{start} & \text{with } x_{start} \in C_{free} \\ X_k(\omega) = g(X_{k-1}(\omega), v_k(\omega)) & \text{for k=1,2,...} \end{cases} \quad (4)$$

where for every k the random vector $v_k(\omega)$ is normal, zero-mean with covariance matrix $\Sigma_k(\omega)$ (the dependence of a random variable on ω will be hereby implicit). We call the stochastic process in eq. 4, Adaptive Random Walk (ARW).

Definition 3: We define the random vector H_k as corresponding to $\{X_0, X_1, ..., X_k\}$. We also denote with $\sigma(H_k)$ the $\sigma - algebra$ generated by H_k ([10]).

Assumption 4: For every k, v_k is independent from $\sigma(H_{k-1})$ once Σ_k is known. Moreover, every entry of Σ_k is a random variable which is measurable with respect to $\sigma(H_{k-1})$
It comes from Assumption 4, that there exists for every k a function $L_k : C^k \to \mathbb{R}^{n\times n}$ such that $\Sigma_k = L_k(H_{k-1})$. We call L_k the learning rule of the random walk at instant k.

Assumption 5: For every k, $\varepsilon_1 I_n \le \Sigma_k \le \varepsilon_2 I_n$ where I_n is the $n \times n$ identity matrix and $\varepsilon_1, \varepsilon_2$ are strictly positive numbers.

Definition 6: Given H_{k-1} and assigned $x \in C$ and $A \in B(C)$, we denote with $P^m_{k,H(k-1)}(x,A)$ the m-step transition kernel of the chain, i.e. once known the history of the chain until instant k, $P^m_{k,H(k-1)}(x,A)$ provides the probability that ARW takes value in A at instant $k+m$.

Definition 7: Assigned a set q belonging to $B^+(C)$ and contained in C_{free}, we denote with $V(q)$ the maximal open set of points $x \in C_{free}$ such that for every y belonging to q, $\{x,y\}$ is an admissible couple.

We point out that the definition of $V(q)$ is similar to the one defined in [4].

C. Probabilistic completeness of the Adaptive Random Walk algorithm

Lemma 8: Let q a set belonging to $B^+(C)$ and contained in C_{free}. Then, given an arbitrary $D \in B^+(C)$ contained in $V(q)$, there exists a strictly positive m such that for every $x \in V(q)$, k and H_{k-1}:

$$P^2_{k,H_{k-1}}(x,D) \ge m$$

Proof: By definition of $V(q)$, we have that for every $x \in V(q)$, k and H_{k-1}:

$$P^1_{k,H_{k-1}}(x,q) \ge \int_q N_{x,\Sigma_k}(y)dy \ge l\mu(q) \doteq m_1$$

where l is a real number which uniformly bounds from below the function to be integrated on q (note that the existence of l comes from Assumption 5). Moreover, we also have that for every $x \in q$, k and H_{k-1}:

$$P^1_{k,H_{k-1}}(x,D) \ge \int_D N_{x,\Sigma_k}(y)dy \ge l\mu(D) \doteq m_2$$

Combining the two inequalities, we have that for every $x \in V(q)$, k and H_{k-1}:

$$P^2_{k,D}(x,D) \ge m_1 m_2$$

Remark 9: It is easy to note that when $V(q)$ is convex, we could replace $P^2_{k,D}(x,D) \ge m$ with $P^1_{k,D}(x,D) \ge m$ in the statement of Lemma 8.

Definition 10: Let T and U be subsets of C_{free}. We say that T and U are adjacent (and we write $Adj(T,U)$) when the following holds:

$$\overline{T} \cap \overline{U} \cap C_{free} \ne \emptyset$$

Lemma 11: Let A_i and A_j belong to $B^+(C)$ and to C_{free}, such that $Adj(V(A_i), V(A_j))$. Then, there exists a strictly positive m, such that for every $x \in V(A_i)$, k and H_{k-1}:

$$P^3_{k,H_{k-1}}(x, V(A_j)) \ge m$$

Proof: Since $Adj(V(A_i), V(A_j))$, there exists a point

$$x_0 \in \overline{A_i} \cap \overline{A_j} \cap C_{free}$$

Then, since C_{free} is open, there exists a ball $B_\varepsilon(x_0)$ with radius ε and center x_0 which is entirely in C_{free}. We define the set $D_1 = V(A_i) \cap B_\varepsilon(x_0)$. Clearly, we have $D_1 \in B^+(V(A_i))$. Then, by applying Lemma 8, there exists a strictly positive constant m_1 such that for every $x \in V(A_i)$, k and H_{k-1}:

$$P^2_{k,H_{k-1}}(x, D_1) \ge m_1$$

Let $D = V(A_j) \cap B_\varepsilon(x_0)$. Again, by applying Lemma 8 and by considering that $B_\varepsilon(x_0)$ is convex, there exists a strictly positive constant m_2, such that for every $x \in D_1$, k and H_{k-1}:

$$P^1_{k,H_{k-1}}(x, D) \ge m_2$$

Composing the two inequalities, we have that for every $x \in V(A_i)$, k and H_{k-1}:

$$P^3_{k,H_{k-1}}(x, D) \ge m_1 m_2$$

which completes the proof.

Definition 12: A triplet $\{C_{free}, x_{start}, x_{goal}\}$ is a solvable instance of the motion planning problem if the set

$$S(C_{free}, x_{start}, x_{goal}) = \{f : [0,1] \to C_{free}$$
$$\text{such that } f(0) = x_{start}, f(1) = x_{goal}, f \in C^0\}$$

is not empty.

Definition 13: Let $\{C_{free}, x_{start}, x_{goal}\}$ a solvable instance of the motion planning problem, let

$f \in S(C_{free}, x_{start}, x_{goal})$ and let A be a subset of C_{free}. We say that a solution f crosses A once if the set

$$t_a = \{t \in [0,1] \text{ such that } f(t) \in A\}$$

is an interval (either open, closed or half open).

Theorem 14: Let $\{C_{free}, x_{start}, x_{goal}\}$ be a solvable instance of the motion planning problem. Let us suppose that there exists a finite sequence $A_1, A_2, ..., A_k$, where every A_i belongs to $B^+(C)$ and is contained in C_{free}, such that

$$C_{free} = \bigcup_{i=1}^{k} V(A_i)$$

Then, there exists $f \in S(C_{free}, x_{start}, x_{goal})$ that crosses at most once $V(A_i)$ for $i = 1, 2, ..., k$.

Proof: Omitted.

Theorem 15: Let C_{free} be connected and such that there exists a finite sequence $A_1, A_2, ..., A_k$, where every A_i belongs to $B^+(C)$, is contained in C_{free} and

$$C_{free} = \bigcup_{i=1}^{k} V(A_i)$$

Then, for each $x_{start} \in C_{free}$ and X_{goal} belonging to $B^+(C)$ and to C_{free}, the algorithm ARW started in x_{start} will reach X_{goal} with probability 1.

Proof: Without loss of generality we suppose that X_{goal} is entirely included in one of the sets of the sequence $V(A_1), V(A_2), ..., V(A_k)$ denoted $V(A_l)$. In the light of Lemma 8 and 11, there exists a strictly positive m such that for every $V(A_i)$ and $V(A_k)$ with $Adj(V(A_i), V(A_k))$ and for every $x \in V(A_i)$, k and H_{k-1}:

$$P^3_{k,H_{k-1}}(x, V(A_k)) \geq m$$

and for every $x \in V(A_l)$, k and H_{k-1}:

$$P^2_{k,H_{k-1}}(x, X_{goal}) \geq m$$

This, in addition with Theorem 14, allows us to conclude that there exists constants h and s, independent from $x \in C_{free}$, k and H_{k-1} such that

$$\sum_{r=1}^{h} P^r_{k,H_{k-1}}(x, X_{goal}) \geq s > 0$$

It follows that the probability that ARW has never entered the set X_{goal} after Zh steps is less or equal to $(1-s)^Z$. Clearly, when Z diverges, this probability goes to zero.

V. Conclusions and Future Work

We introduced a random walk based adaptive motion planner. The algorithm builds a random walk according to a Gaussian sampling over the configuration space. As the produced path can be very uneven, an efficient post processing step is used, yielding a fast smoothing of the produced path. A simple but effective technique for letting the algorithm adapt the parameters of the random distribution used have been devised. Of course, it could also be possible to use more refined techniques, for example using correlation of the last samples, i.e. to use a non diagonal Σ_k. A sketch of the theoretical soundness of the algorithm has been provided. Numerical results illustrate that indeed for some problems the adaptive random walk planner appears to be competitive with previously developed algorithms. In the near future we plan to apply the algorithm to bioinformatics problems like protein folding. In that context, we expect the adaptivity to be better exploited, thanks to the continuous nature of the energetic levels concerning proteins instead of the boolean nature of the configuration space found in robot motion planning.

Acknowledgments

We thank professor Steve LaValle for making available the MSL software and Peng Cheng for useful hints.

References

[1] A. Atramentov and S.M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 632–637, Washington, May 2002.

[2] R. Bohlin and L.E. Kavraki. Path planning using lazy prm. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1469–1474, Seoul, May 2001.

[3] 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.

[4] M. Branicky, S. M. LaValle, K. Olsen, and L. Yang. Deterministic vs. probabilistic roadmaps. Submitted to IEEE Transactions on Pattern Analysis and Machine Intelligence.

[5] S. Carpin. *Advanced techniques for randomized robot motion planning*. PhD thesis, The University of Padova, December 2002.

[6] S. Carpin and E. Pagello. Exploiting multi-robot geometry for efficient randomized motion planning. In Maria Gini et al., editor, *Intelligent Autonomous Systems 7*. IOS Press, 2002.

[7] P. Cheng and S.M. LaValle. Deterministic resolution complete rapidly-exploring random tree. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Washington, May 2002.

[8] H. Haario, E. Saksman, and J. Tamminen. Adaptive proposal distribution for random walk metropolis algorithms. *Computational Statistics*, 14, 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 Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[10] B. Øksendal. *Stochastic Differential Equations*. Springer, 1998.

[11] J.J. Kufner and S.M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 995–1001, San Francisco, April 2001.

[12] 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.

[13] 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.

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

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

[16] G. Song and N.M. Amato. Using motion planning to study protein folding. In *Proceedings the 5th International Conference on Computational Molecular Biology (RECOMB)*, pages 287–296, 2001.