iARW: An incremental path planner algorithm based on adaptive random walks

Bruno Vilhena Adorno and Geovany Araújo Borges

***Abstract*— This paper presents a new path planning algorithm that uses adaptive random walks to incrementally construct a roadmap in the robot's free configuration space. This algorithm, named Incremental Adaptive Random Walks (iARW), uses a modified version of the ARW algorithm proposed by Carpin and Pillonetto [1] for exploring the configuration space and storing the discovered path in a roadmap. Thus, the main idea is to use bidirectional adaptive random walks to explore the configuration space but also to use and expand the roadmap whenever possible. With this approach it is possible to construct a roadmap that captures the connectivity of the free configuration space without a preprocessing phase. A comparison of our approach with other state of the art path planners illustrates the good performance of the proposed method.**

I. INTRODUCTION

The path planning problem consists in finding a sequence of movements for enabling a robot to leave a start configuration $\mathbf{q}_{start}$ and arrive at the goal $\mathbf{q}_{goal}$ without collision with the obstacles in the workspace $\mathcal{W}$. To tackle this problem, several feasible algorithms have been proposed in the last years. For instance, some of them are: Probabilistic Roadmaps (PRM) [2], Rapidly-Exploring Random Trees (RRT)[3], Expansive Space Trees (EST) [4], Randomized Potential Field Planner (RPP) [5] and a less known but nonetheless important, Adaptive Random Walks (ARW) [1]. More general surveys on path planning algorithms can be found in [6], [7], [8]. The main idea of PRM, RRT and EST algorithms is to build a roadmap in the free configuration space $\mathcal{C}_{free}$ in order to capture its connectivity, although PRM is a multiple query algorithm and RRT and EST are primarily designed as single query algorithms. Furthermore, RPP uses the classical approach of defining an attractive potential to the goal configuration and a repulsive potential for the obstacles. However, RPP employs random walks to escape local minima. Indeed, even in PRM, random walks can be used for expanding the roadmap [2]. The former algorithm, ARW, is made entirely of random walks but its main strength relies on the fact that the random walks adapt themselves accordingly to the configuration space, refining the sampling and thus accelerating the convergence of the process. The main drawback of these algorithms is, excepting the RPP, the poor quality of the resultant paths. However, the final path can be smoothed on a post-processing phase [9].

B. V. Adorno is PhD candidate at the Département Robotique, Laboratoire d'Informatique, Robotique et Microélectronique de Montpellier (LIRMM), UMR 5506 - CC 477 161 rue Ada, 34392 Montpellier Cedex 5 - France. e-mail: `adorno@lirmm.fr`

G. A. Borges is supported by CNPq under grant number 305710/2005-2. He is adjunct professor at Departamento de Engenharia Elétrica, Universidade de Brasília, Caixa Postal 4386, CEP 70919-970, Brasilia, DF - Brazil. e-mail: `gaborges@ene.unb.br`

The simplicity and good performance of ARW [10] together with its probabilistic completeness [1] have attracted our attention, although this algorithm has been disregarded by the path planning community. This motivated the investigations and developments presented in this work.

Our main contribution concerns the development and evaluation of a hybrid algorithm capable of representing incrementally the connectivity of free configuration space without a preprocessing phase. Furthermore, we have proposed a simple measure for explorability, better suited for small dimensional configuration spaces, enabling us to replace the original distribution of ARW by a biased one, improving the overall performance of the algorithm.

The paper is organised as follows: the standard ARW is briefly described in the next section, Section III reviews previous attempts to improve the original algorithm and proposes a simple method to augment the explorability capabilities of the planner. Section IV presents a new algorithm that incrementally uses adaptive random walks and roadmaps to explore and capture the connectivity of the free configuration space. Comparisons between this novel approach and state of the art algorithms are made in Section V. Finally Section VI presents the conclusions.

II. DESCRIPTION OF ARW PLANNER

In the Adaptive Random Walk path planner [1], given a configuration $\mathbf{q}_k$, a neighbor configuration $\mathbf{q}_{k+1}$ is generated accordingly to a normal distribution, $\mathbf{q}_{k+1} \sim N(\mathbf{q}_k, \mathbf{\Sigma}_k)$. The sample is accepted if it is generated in $\mathcal{C}_{free}$ and otherwise is rejected. In the case of an accepted sample, it is appended to the chain of intermediate configuration and a new sample $\mathbf{q}_{k+2}$ is generated using as reference $\mathbf{q}_{k+1}$. The process is repeated until the last accepted sample can be connected to the goal configuration by the local path planner.

The key point here is the calculation of the covariance matrix $\mathbf{\Sigma}_k$ as

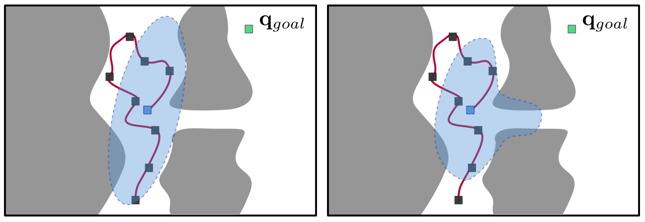
$$\mathbf{\Sigma}_k = \max\left(\mathbf{P}_k, \mathbf{\Sigma}_{min}\right) \tag{1}$$

where the matrix metric is its trace, and

$$\mathbf{P}_k = \left(\frac{1}{H}\sum_{i=k-H}^{k-1} \mathbf{q}_i \cdot \mathbf{q}_i^T\right) - \bar{\mathbf{q}}_H \cdot \bar{\mathbf{q}}_H^T, \tag{2}$$

with $\bar{\mathbf{q}}_H$, the average of the last H configurations in the

```
Algorithm 1: Unidirectional ARW
Input  : $\mathcal{W}$, $\mathbf{q}_{start}$, $\mathbf{q}_{goal}$
Output: List containing the path connecting $\mathbf{q}_{start}$ to $\mathbf{q}_{goal}$
1  $k \leftarrow 0$; $\mathbf{q}_k \leftarrow \mathbf{q}_{start}$;
2  $\mathbf{\Sigma}_0 \leftarrow \mathbf{\Sigma}_{min}$;
3  while not local_planner($\mathbf{q}_k$,$\mathbf{q}_{goal}$) do
4      Sample a new configuration $\mathbf{v}_k$ from $N(0, \mathbf{\Sigma}_k)$;
5      $\mathbf{s} \leftarrow \mathbf{q}_k + \mathbf{v}_k$;
6      if local_planner($\mathbf{q}_k$,$\mathbf{s}$) then
7          $k \leftarrow k + 1$;
8          $\mathbf{q}_k \leftarrow \mathbf{s}$;
9          Update the covariance matrix $\mathbf{\Sigma}_k$ using (1);
10         Put $\mathbf{q}_k$ in the list of intermediate configurations;
```



(a) 3σ ellipsis of adaptive Gaussian distribution. (b) Biased distribution.

Fig. 1. Biasing the sampling towards less explored regions.

chain, given by

$$\bar{\mathbf{q}}_H = \frac{1}{H} \sum_{i=k-H}^{k-1} \mathbf{q}_i. \tag{3}$$

Equation (1) indicates that the covariance matrix is computed using only the last H configurations stored in the chain. If the last H samples are far from each other, probably the random walk is in a open region and the trace of the covariance matrix will be larger. Thus, the next sample will have a higher probability of being generated farther from the reference configuration, enabling the planner to explore $\mathcal{C}_{free}$ using larger steps. Conversely, if the last H samples are closer to each other, it indicates that the random walk is in a narrow space and the trace of the covariance matrix will be smaller. In this manner, smaller steps will be taken between two configurations, leading to a more refined sampling.

Algorithm 1 summarizes the evolution of the random walk previously described. In general the path planning is solved using two ARW. One leaves the start configuration and the other one leaves the goal configuration. If a random walk can be connected to the other or if one them can be connected to its target configuration, the query is solved.

Unlike PRM that uses a graph to represent the roadmap or RRT and EST that use a tree to represent the connectivity of the free configuration space, the main idea of ARW is to generate a chain of intermediate configurations in $\mathcal{C}_{free}$. This leads to a simpler data structure representation, once the relevant portion of $\mathcal{C}_{free}$ with respect to the current query can simply be represented as a list.

III. IMPROVEMENTS ON ARW PLANNER

The first attempt to improve the performance of the ARW algorithm was made by its original developers in [11]. They investigated three extensions: biasing the generation of samples; using an attractor for the samples generator; and the possibility to backtrack when the planner gets stuck. They use backtracking as a solution to free the random walk when it gets stuck in dead ends. For instance, if the last H motions couldn't be bigger than 10% of the tried distance, the random walk is considered stucked and a random configuration is chosen uniformly from the chain and then the random walk is restarted from there. The attractor was defined so that the probability of accepting a sample close to the goal configuration is higher than the probability of accepting a farther one. Furthermore, they evaluated the biasing of the random walks towards each other and toward their goals.

When comparing with a standard bidirectional ARW, the results in [11] show that the only method that seems to improve the performance of the resultant algorithm is the addition of a constant bias between the two random walks. Adding an attractor or using backtracking presented no important change in results and an arbitrary combination of the methods can significantly degrade the overall performance, besides the additional complexity of the final algorithm.

Once we understand that the effect of bias in [11] is to modify the sampling distribution of the random walks, one question naturally arises: is the Gaussian distribution the best one? Carpin and Pillonetto [1] show that the Gaussian distribution leads to a planner that is probabilistic complete. Furthermore, it is cheap to generate samples from a normal distribution. But all this does not imply that the Gaussian distribution is the best one. However, it seems that the great idea behind ARW is its adaptivity, and maybe a simpler distribution can work as well, like the uniform distribution. Moreover, by taking into account the results in [11], it seems a good idea to bias the distribution towards less explored areas of $\mathcal{C}_{free}$.

Fig. 1 shows an example on the effect of biasing towards less explored regions. Although the adaptivity of the Gaussian distribution improves the sampling in $\mathcal{C}_{free}$, it is still difficult to sample the narrow corridor, as it can be seen in Fig. 1(a). A biased distribution, as shown in Fig. 1(b) can improve the convergence rate of the algorithm due to a higher explorability.

One practical way of biasing the distribution is to generate multiples candidates and choose the one from the less explored region. A rough method that should work for small degrees of freedom is to impose a grid on the configuration space and associate for each cell the number of configurations in its interior. This can be made incrementally for each new configuration appended to the chain. A region with small number of generated configurations is considered less explored. Thus, for each candidate generated in $\mathcal{C}_{free}$ we can associate it with a region and, as a consequence, we can choose the one from the region with the smallest counter.

IV. INCREMENTAL ADAPTIVE RANDOM WALKS

Probably the greatest advantage of roadmap methods is to capture the global connectivity of $\mathcal{C}_{free}$ whatever may be the dimension of configuration space. Indeed, all information about the dimension of $\mathcal{C}$ can be embedded in the nodes, without changing the graph topology. Thus, the graph can be regarded as a "compression" of the search space. For that reason, the query phase is generally much faster than the construction phase in roadmap methods. On the other hand, it is usually computationally expensive to build the roadmap. Therefore single query algorithms can achieve better performance if it is not necessary to capture the connectivity of whole $\mathcal{C}_{free}$ to solve a specified query [4], [3], [12].

Single query algorithms, however, do not keep any information of the configuration space for subsequent queries. Hence, for the same queries they will have to repeat the search in the configuration space. Some extensions have been made to RRT [13], [14], [15] and to PRM [16] aiming to solve this problem.

Here we present an incremental algorithm that explores $\mathcal{C}_{free}$ using adaptive random walks. For each query, the relevant pieces of the generated path are stored in a roadmap in a way that it can be used in subsequent queries. The main goal is to explore $\mathcal{C}_{free}$ and also to expand and use the roadmap whenever possible.

iARW is shown in Algorithm 2. The inputs are the start and goal configurations and the roadmap, if any is available for the query. As in the default bidirectional ARW, the planner merges the two random walks or connects them to the target points. If any of these can be done, the query is answered and the path is stored in the roadmap. Otherwise, each random walk tries to connect itself with new components of the roadmap. If there is some common component attached to both chains, the planner has found the solution. It is the concatenation between the first random walk (just from $\mathbf{q}_{start}$ up to the configuration that is connected to the common component), the path found in the roadmap and the second random walk (again, only the part that goes from the configuration connected to the common component up to $\mathbf{q}_{goal}$). If there is no common component, new samples are generated for each chain and the process continues until a path is found or a maximum number of nodes is generated on $\mathcal{C}_{free}$.

Algorithm 3 presents the process of searching new components and expanding the roadmap. Given the last configuration $\mathbf{q}_{last}$ of the chain generated by a random walk, the roadmap and the list of components to which the random walk is connected, it tries to connect $\mathbf{q}_{last}$ to all configurations of the roadmap that belong to a component different of the ones to which the random walk has already been previously connected, which are in the component list. When the first new component is found, the relevant part of the random walk is stored in the graph and if there is another component to which $\mathbf{q}_{last}$ can be connected its just a matter of adding a edge in the graph, once the relevant part of

Algorithm 2: iARW

```
Input  : Start and goal configurations and the roadmap $\mathcal{G}$.
Output: Configurations of the resultant path.
1  for at most k configurations do
2      if merge_ARW(ARW_s,ARW_g) then
3          store(path) and return path
4      find_new_component(ARW_s,$\mathcal{G}$,components_start);
5      find_new_component(ARW_g,$\mathcal{G}$,components_end);
6      if exist_common(components_start,components_end)
       then
7          return path
8      else
9          generate_new_configuration(ARW_s);
10         generate_new_configuration(ARW_g);
```

Algorithm 3: Find a new component in the roadmap.

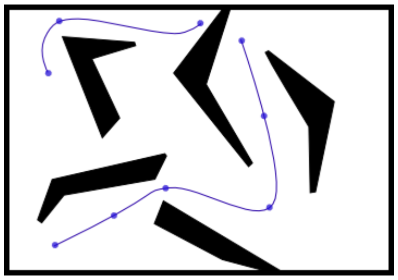
```
Input  : Configuration $\mathbf{q}_{last}$, roadmap $\mathcal{G}$ and a list containing the
         connected components to the random walk associated to
         $\mathbf{q}_{last}$.
Output: 1 if it has found a new component and 0 otherwise
1  find ← 0;
2  forall nodes $\mathbf{q}_i$ on $\mathcal{G}$ do
3      if not same component and local_planner($\mathbf{q}_{last}$,$\mathbf{q}_i$) then
4          if not find then
5              entry_points ← store_partial_path();
6              find ← 1;
7          update_list(components_list);
8          add_edge(entry_points,$\mathbf{q}_i$);
9  return find
```

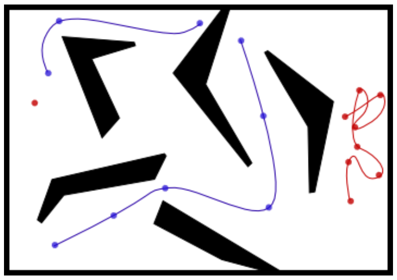
the random walk has just been stored. We keep track of the entry points in an individual query for two reasons: the first one is to know which are the relevant parts that will be stored (generally are the configurations between the last entry point and the current entry point); the second reason is to know which pieces of the random walk have to be taken into consideration when the iARW finds a common component.

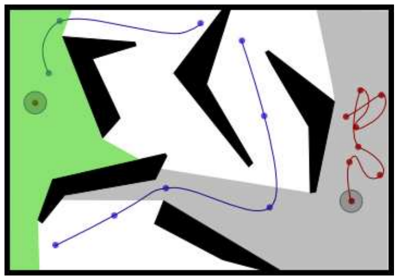
The overall process described by Algorithms 2 and 3 can be illustrated by Fig. 2. Fig. 2(a) starts ilustrating two components previously stored in the roadmap, indicating that this is not the first query, once in the first one iARW acts exactly like a standard ARW. In Fig. 2(b) two random walks start on the opposite sides of the scene. They are not necessarily symmetric, because their evolutions are subject to local characteristics of the environment. Fig. 2(d) indicates that the relevant part of the chain is smoothed before it is stored in the roadmap. For this smoothing we decided to use the standard divide and conquer algorithm [1]. Finally, Fig. 2(f) shows the final path given by the hatched part. In order to avoid too much redundance, the pieces past the entry point are not stored in the graph.

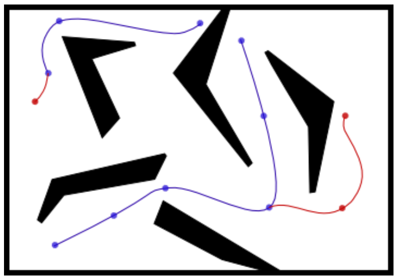
V. RESULTS

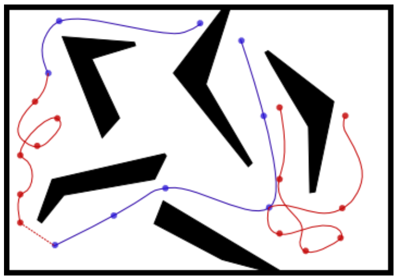
iARW performance has been evaluated and compared with both single and multiple query algorithms. We first compare several variants of ARW with the original one, as well as with EST and RRT. We used the test environments shown in Fig. 3. From these environments, LIRMM 1 and


(a) Previously stored roadmap.


(b) Two random walks started on the right and left.


(c) Visibility of each last node.


(d) The random walks are connected to roadmap components that were in their visibility regions. Unnecessary nodes are removed by a divide and conquer algorithm.


(e) The search occurs until both random walks can be connected to the same component in the roadmap.

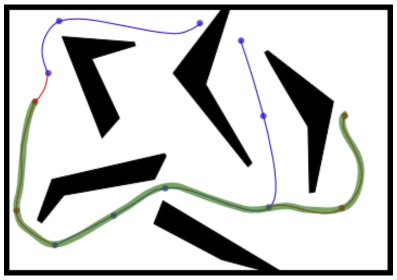

(f) The final path is given by the hatched piece. Only the relevant paths are stored in the roadmap.

Fig. 2. iARW evolution showing how adaptive random walks can benefit from a previously stored roadmap.

LIRMM 2 are occupancy grids obtained from real data gathered in experiments with the Omni robot [17]. The other tests environments are simulated. This figure also shows the reference paths used in the first evaluation. The square robot used for the tests could only translate on the plane, thus having two degrees of freedom. For tunning the ARW, EST and RRT parameters, tests were previously conducted in the three environments shown in Fig. 3(a)–(c). It has led to the following parameter values (considering the side of the square robot equals to 10 units):

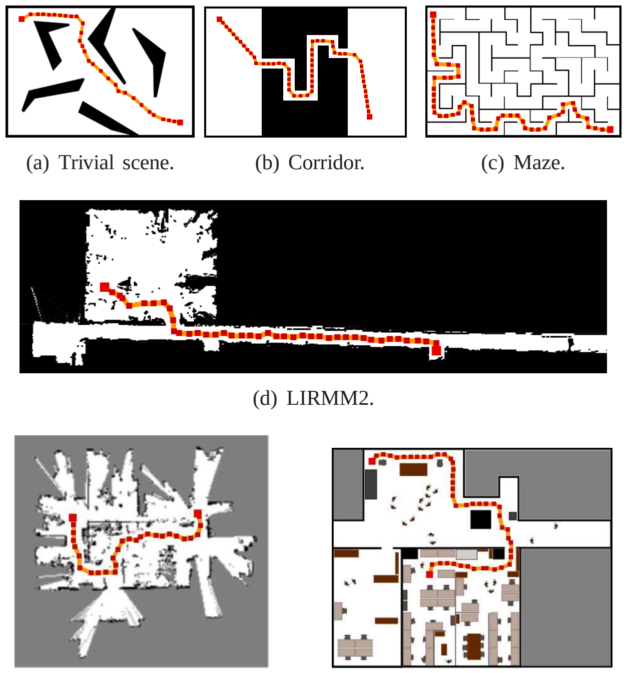
- ARW: $H = 50$;
- EST: Mass function for neighbor choosing $\pi(q) = \frac{1}{(n_v+1)^3}$, with n_v the number of neighbors of the configuration q, and the size of the interval for the uniform distribution equals to 70 units;
- RRT: Steps $\epsilon = 40$ units and merging made by RRT–connect.

In the second evaluation we compare the performance of iARW (using a biased ARW) with a standard PRM and the Gaussian one [18] in the same environments of the first evaluation.

It is important to point out that although we don't use the original implementation of the reference algorithms (we used our own implementations), all planners use the same local path planner, the same collision detection and graph search algorithms (when applicable).

Fig. 4 shows the results for the first comparison. The time indicated in the ordinate axis is the total time used to find and smooth the reference paths and it is in logarithmic scale. The simpler uniform distribution did not improve the overall performance of ARW. On the contrary, in most scenes the average performance has been degraded. However, the biasing mechanism using selection of candidates has lowered significantly the average time to solve the queries. Indeed, with exception of the maze, in all environments a larger number of candidates implied in less time to find the reference path. In general, we have seen that about five candidates are enough to improve the performance of the algorithm. However, when the number of candidates turn out to be bigger than ten, perfomance starts to degrade. This can be explained by two reasons: the first one is regarding to the overhead of generating more samples and their collision checking; the second one is that more candidates implies more bias and the random walk can gets temporally stuck in the local minima generated by the rough metric for the explorability criteria presented in Section III.

When compared with both EST and RRT, the biased ARW seems to be very competitive. Indeed, in four of the six test environments, the biased ARW presented the best performance, regardless of the number of candidates used. A special attention should be given to the results in LIRMM 1. This environment is very cluttered and confined, and one can expect that all the algorithms would be penalized due to very narrow free spaces. Indeed, given a occupancy grid representing LIRMM 1, it was transformed in a bitmap representation using a very conservative threshold. Thus, if a given cell had an occupancy probability greater than zero, we have considered it as occupied. In the end, it leads to a still more confined scene than the original one shown in Fig. 3(e). In this scene, both EST and RRT were more penalized


(a) Trivial scene. (b) Corridor. (c) Maze.

(d) LIRMM2.

(e) LIRMM1. (f) LARA.

Fig. 3. Test environments.

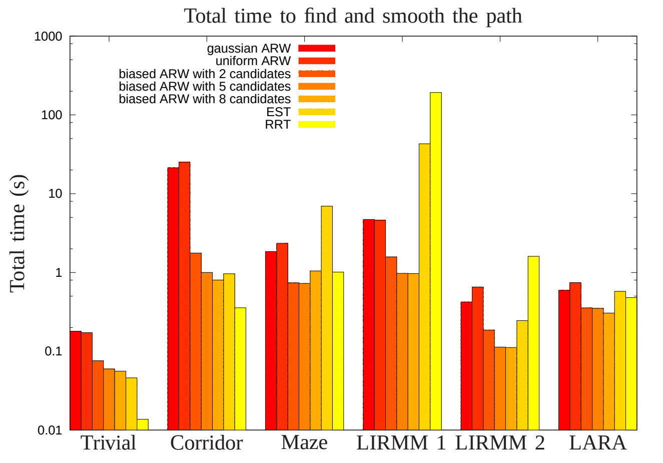


Fig. 4. Comparison between the single query algorithms. The results are the average time between 100 executions.

than all other kinds of ARW, biased or not. As explained before, the parameters of all algorithms were tunned based on previous test conducted using the first three scenes of Fig. 3. When confronted with a new environment, all flavors of ARW maintained a good performance, while EST and RRT did not. This highlights the strongest characteristic of ARW: its adaptability to the environment, hence implying good performance in very different workspaces. Furthermore, the biasing scheme improves greatly the performance of the algorithm when compared with the original ARW. However, it is important to note that in Figure 4 we only show the results of the biased ARW, not of the iARW, due to the fact that we want to evaluate individual queries. If we had conducted incremental queries in the experiment, iARW would have greatly outperformed both ARW, RRT and EST once these algorithms do not use previous informations for new queries. Thus, it would be unfair to conduct such a comparison.

The second evaluation concerns iARW (using biased ARW with three candidates), standard PRM and gaussian PRM. As roadmaps methods are more suited to multiple queries, for each scene four non–trivial paths were chosen as references. The Corridor is an exception: only the path that passes through the corridor is non–trivial. Thus, for this specific environment, one can expect that the planner that first succeeds to connect both sides of the scene takes less time to find all four routes, since the first one is the bottleneck for this map. For each environment and for each path planner, we performed 100 successful executions. We considered an execution as successful when all four reference routes could be found. Besides, as the number of nodes necessary to capture the connectivity of a scene is not known *a priori* when working with PRM, we proceeded as follows: for each map, at first $n_0 = 100$ nodes were generated. If all four reference routes could be found, this execution was considered successful and other iterations could be made with the same number of nodes. Otherwise, if any of them could not be found, the execution was restarted with $n_k = 2n_{k-1}$. Thus, the number of generated nodes was doubled until all four paths were found. The execution time considered for performance evaluation was the average of 100 successful iterations.

Fig. 5 shows the results for time performance in the comparison between iARW and the PRMs. As expected, the Gaussian PRM was superior than uniform PRM in all environments with narrow passages. In the trivial scene, that only has open spaces, both PRMs achieved nearly the same performance. On most scenes, iARW was much better than both PRMs. However, in the Corridor both Gaussian PRM and iARW required almost the same average time to find all routes. This was also expected, once this is the kind of map for which Gaussian PRM is tailored. Also, in the Maze both iARW and Gaussian PRM had the same performance.

The other advantage of iARW is highlighted on Fig. 6. The number of nodes necessary to capture the connectivity of $\mathcal{C}_{free}$ is smaller on iARW. This is due to the fact that iARW only stores relevant portions of $\mathcal{C}_{free}$, whilst the standard PRM samples the configuration space uniformly. The Gaussian PRM also does a great selection when sampling, but not as good as iARW. It occurs due to the fact that gaussian PRM strongly samples narrow spaces, but also the boundary of obstacles, which is not necessarily a relevant part of $\mathcal{C}_{free}$.

VI. CONCLUSIONS AND FUTURE WORKS

This paper presents a new path planner, iARW, that uses biased adaptive random walks to solve a query and simultaneously expand a roadmap in $\mathcal{C}_{free}$. It uses the biased bidirectional ARW to explore the configuration space and, as the queries are solved, the relevant parts of the

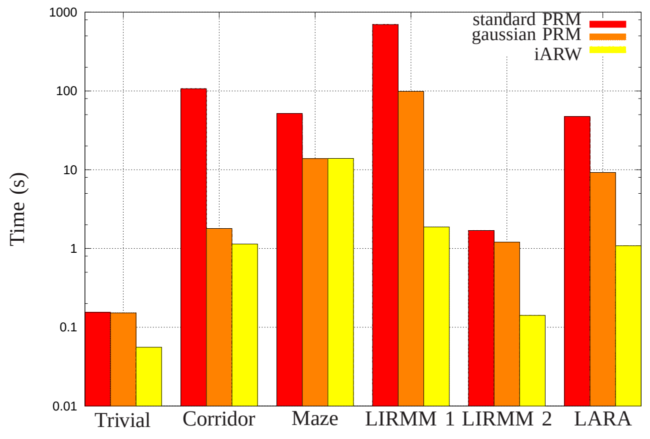


Fig. 5. Comparison between iARW and PRM: total time for finding four reference paths.

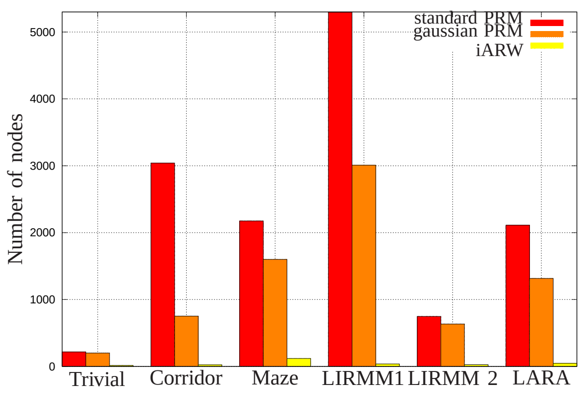


Fig. 6. Comparison between iARW and PRM: number of nodes in the final roadmap.

path are stored in the roadmap. However, when searching the configuration space, each random walk tries to connect itself to as many different components of the roadmap as possible, augmenting the connectivity of the graph. Besides, the roadmap is used whenever possible, improving the algorithm's perfomance. Furthermore, a simple approach to bias the random walks is presented. The method consists in taking a configuration, from a set of candidates, that maximizes an explorability criteria. A first evaluation showed that our biased ARW greatly outperforms both standard Gaussian ARW and the uniform one. Nevertheless, we must point out that the heuristic used in the explorability criteria should work well only for a small and medium number of dimensions. When compared with RRT and EST, the biased ARW showed a very competitive performance, with the additional advantage of using a simpler data structure. In the second evaluation, we have compared iARW with both the standard PRM and the Gaussian one. In general, iARW has greatly outperformed both PRMs and in only one case the Gaussian PRM had the same perfomance of iARW. In despite of the good perfomance achieved by the proposed method, we believe that further improvements could be achieved by using other kinds of adaptive distributions. Finally, our next step is to conduct comparisons between iARW and other incremental algorithms such as Lazy PRM [16], LRF [15] and Multipartite RRTs [14]. Also, an evaluation using high dimensional configuration spaces should give more insights on the overall performance of the proposed algorithm.

ACKNOWLEDGEMENTS

We would like to thank the helpful suggestions and feedbacks of the anonymous reviewers. Also, we would like to thank Rogerio Richa for helping to improve the text of the final manuscript.

REFERENCES

[1] S. Carpin and G. Pillonetto, “Motion planning using adaptive random walks,” *IEEE Transactions on Robotics*, vol. 21, no. 1, 2005.

[2] L. E. Kavraki, M. N. Kolountzakis, J. C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, 1996.

[3] J. J. J. Kuffner and S. M. LaValle, “RRT-connect: An efficient approach to single-query path planning,” in *IEEE International Conference on Robotics and Automation*, vol. 2, 2000, pp. 995–1001.

[4] D. Hsu, J.-C. Latombe, and R. Motwani, “Path planning in expansive configuration spaces,” in *Proceedings of the 1997 IEEE International Conference on Robotics and Automation*, 1997, pp. 2719–2796.

[5] J. Barraquand and J. Latombe, “Robot motion planning: A distributed representation approach,” *The International Journal of Robotics Research*, pp. 628–649, 1991.

[6] H. Choset, K. M. Lynch, S. Hutchinson, W. B. G. Kantor, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

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

[8] S. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.

[9] R. Geraerts and M. H. Overmars, “Clearance based path optimization for motion planning,” New Orleans, LA, 2004, pp. 2386–2392.

[10] B. V. Adorno and G. A. Borges, “Planejamento de caminho usando biarw melhorado e mapa de rotas,” in *Simpósio Brasileiro de Automação Inteligente*, 2007, pp. 1–6.

[11] S. Carpin and G. Pillonetto, “Merging the adaptive random walks planner with the randomized potential field planner,” *Fifth International Workshop on Robot Motion and Control*, pp. 151–156, 2005.

[12] B. V. Adorno, “Planejamento probabilístico de rotas no espaço de configuração e sua aplicação em robótica móvel,” Master's thesis, Universidade de Brasília, 2008.

[13] T.-Y. Li and Y.-C. Shie, “An incremental learning approach to motion planning with roadmap management,” in *Proceedings IEEE International Conference on Robotics & Automation*, 2002.

[14] M. Zucker, J. Kuffner, and M. Branicky, “Multipartite rrts for rapid replanning in dynamic environments,” 2007.

[15] R. Gayle, K. R. Klingler, and P. G. Xavier, “Lazy reconfiguration forest (lrf) - an approach for motion planning with multiple tasks in dynamic environments,” in *IEEE International Conference on Robotics and Automation*, Rome, April 2007, pp. 1316–1323.

[16] R. Bohlin and L. E. Kavraki, “Path planning using lazy prm,” in *Lydia E. Kavraki lntemational Conference on Robotics & Automation*, San Francisco, April 2000, pp. 521–528.

[17] G. A. Borges, “Cartographie de l'environnement et localisation robuste pour la navigationde robots mobiles,” Ph.D. dissertation, Université Montpellier II, LIRMM, 161 rue ADA, 34392, Montpellier, Cedex 5, France., May 2002.

[18] V. Boor, M. H. Overmars, and A. F. Stappen, “The gaussian sampling strategy for probabilistic roadmap planners,” 1999, pp. 1018–1023.