Proceedings of the 2004 IEEE
International Conference on Robotics & Automation
New Orleans, LA • April 2004

Motion Planning Using Dynamic Roadmaps

Marcelo Kallmann and Maja Mataric′
Interaction Lab, Computer Science Department
University of Southern California
Los Angeles, California 90089-0781, USA
{kallmann, mataric}@usc.edu

Abstract **- We evaluate the use of dynamic roadmaps for on-line motion planning in changing environments. When changes are detected in the workspace, the validity state of affected edges and nodes of a precomputed roadmap are updated accordingly. We concentrate in this paper on analyzing the tradeoffs between maintaining dynamic roadmaps and applying an on-line bi-directional Rapidly-exploring Random Tree (RRT) planner alone, which requires no preprocessing or maintenance. We ground the analysis in several benchmarks in virtual environments with randomly moving obstacles. Different robotics structures are used, including a 17 degrees of freedom model of NASA's Robonaut humanoid. Our results show that dynamic roadmaps can be both faster and more capable for planning difficult motions than using on-line planning alone. In particular, we investigate its scalability to 3D workspaces and higher dimensional configurations spaces, as our main interest is the application of the method to interactive domains involving humanoids.**

Index Terms - probabilistic roadmaps, motion planning, reaching, humanoids.

I. INTRODUCTION

Robotic algorithms are not yet capable of producing arm motions as efficiently as humans are, and on-line motion planning for complex manipulators, such as humanoid robots, is an open research problem.

The evidence that humans maintain egocentric spatial relationships between sensory signals and motor commands [1] motivates the idea that efficient on-line motion planning should make use of some kind of mapping indicating how occupied positions in space affect the built-in set of valid motions.

In this work we present results obtained with an implementation of a planner employing such a mapping. We follow a similar approach to that introduced by Leven and Hutchinson [2], where a precomputed Probabilistic Roadmap [3] [4] is used to encode valid motions, and a cell decomposition of the workspace is used to map, for each cell overlapped by obstacles, the edges and nodes of the roadmap that are affected. Each time obstacles changes are perceived, the affected workspace cells provide the corresponding edges and nodes of the roadmap to be re-validated on line. The obtained Dynamic Roadmap (DRM) is thus able to cope with dynamic changes in the environment.

The method is most useful when the static part of the environment can be well covered by the roadmap and few portions of the roadmap are invalidated on line, i.e., when the environment at query time does not vary much from the environment at precomputation time. When the roadmap is not capable of deriving a completely valid path, an on-line planner is used [5] to compute the missing parts of the desired path.

Several experiments were conducted in order to compare the performances obtained by using DRMs coupled with an on-line planner and using the on-line planner alone. Virtual environments with randomly displaced obstacles and various kinds of manipulators were considered. In particular we applied the method to the control of a 17 degrees of freedom model of NASA's Robonaut humanoid [6] (see Fig. 1).

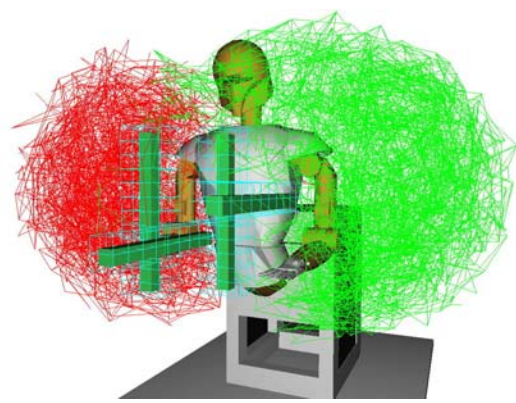

Figure 1. Portions of the roadmap are dynamically invalidated according to the obstacles inserted in Robonaut's workspace.

Our experiments simulate the conditions of a robot manipulating small objects in its workspace. Each required motion is planned and executed while the environment remains static. However, each manipulation implies changes in obstacles, which are considered in subsequent planned motions by updating the roadmap accordingly. Updates may also occur due to a relocation of the robot, or resulting from actions of any external agent sharing the workspace.

Although the described method can be employed in different kinds of motion planning problems, our main interest is the application to humanoids. We foresee that efficient on-line motion planning will be invaluable when humanoids become autonomous enough to replace or cooperate with humans in difficult and dangerous situations. This paper contributes to this end with the evaluation of the use of dynamic roadmaps.

II. RELATED WORK

Several techniques are available in the literature for solving robot motion planning problems [7]. For manipulators with several degrees of freedom (as in human-like arms) suitable techniques mainly draw on sampling-based motion planning, and can be classified into two main categories: multi-query and single-query methods.

Multi-query methods These approaches are based on Probabilistic Roadmaps (PRM) [3] that can be used for several different queries in a single environment. The basic procedure constructs a PRM to randomly sample the configuration space, creating nodes when sampled configurations are valid, and connecting pairs of sufficiently close nodes when connections are valid. Several variations of the basic PRM approach have been proposed [8] [9] [10]; a comparison of these variations is given in [4]. In particular, recent research indicates that better results may be obtained with deterministic instead of probabilistic sampling [11] [4].

Single-query methods Targeting time efficiency, single-query methods are based on roadmap trees, which grow with the sole goal of connecting the initial and final configurations of one given problem.

The Rapidly-exploring Random Tree (RRT) [12] is a very popular single-query method. Its basic idea is to expand nodes of the tree toward random samples, and its bi-directional version [5] is very efficient. Another efficient method is based on Expansive Spaces Trees [13], where nodes in low-density locations are locally expanded. An efficient bi-directional version [14] incorporates lazy collision detection [10] [15], which has been shown to significantly reduce computation time. The concept can also be applied to multi-query roadmaps.

Roadmap maintenance While multi-query roadmaps are generally too costly to be computed for a given environment, single-query methods may still require a considerable time to find motions in complex environments. A natural alternative is to maintain roadmaps in dynamic environments. The easiest approach is to simply delete all nodes that become invalid after an obstacle change [16] [17]. However, the only way to fill the created empty regions is to sample again for new configurations, which is a costly procedure.

The idea of using a workspace mapping to speed-up roadmap maintenance was introduced in [2]. The mapping allows to invalidate (not delete) nodes and edges affected by the occupied regions in the workspace, and also to validate them back when those regions become free, thus not producing empty regions.

In our work we use the same kind of workspace mapping, however we present new implementation solutions and focus on testing two aspects of the approach: 1) how the cost of roadmap maintenance compares against an on-line planner that does not require any maintenance; and 2) how it scales to 3D workspaces.

Application to humanoids Motion planning has been applied to humanoids in both robotics and animation domains. Work to date focuses on locomotion [18] [19] [20] and manipulation [21] [22] [17] problems. However, we believe that insufficient attention has been given to the development and testing of efficient motion planners for humanoids.

We contribute to the field by evaluating the use of dynamic roadmaps coupled with on-line motion planning and analyzing them on humanoids and other control problems.

III. METHOD OVERVIEW AND PAPER ORGANIZATION

Let d be the number of degrees of freedom (DOF) of a given robotics structure. Let C be the d-dimensional configuration space of the robot.

Let C_{free} denote the open subset of valid configurations in C. A configuration is considered valid if the corresponding robot posture is collision free and respects given articulation range limits. Other validity constraints, such as balance, are not considered in this work.

Two data structures are computed off line: the roadmap R and a grid-based cell decomposition of the workspace W. The roadmap is computed only considering the robot and the static obstacles in W. The grid G stores in each cell $c \in G$, all nodes and edges of R that are affected by c. We call this process *cell localization* and, coupling G with R, we obtain a Dynamic Roadmap (DRM). The entire precomputation phase is presented in Section IV.

During run time, dynamic obstacles are tracked and each time an obstacle appears, disappears, or moves, the affected nodes and edges of R are updated accordingly. Two strategies are presented. One simply updates reference counters; the other retests the validity of all the affected nodes and edges of R, which are efficiently identified from the information stored in the grid G. These maintenance procedures are presented in Section V.

As it is constantly updated, the roadmap is ready to answer queries with an A*-like graph search procedure [7]. However, in cases where large portions of the roadmap become invalid, the roadmap may fail to return a path, and an on-line planner is used to retrieve a valid one. Section VI describes the querying process.

We performed several experiments in order to assess the tradeoffs between maintaining DRMs and using on-line planning alone. The experiments are described in Section VII.

Section VIII presents and discusses the results, and Section IX concludes the paper.

IV. ROADMAP COMPUTATION

We start by defining the region of the workspace W where the robot will be working. For the manipulators examples in this paper we use the reachable space of the robot's arms.

Cell decomposition A cell decomposition of W is defined covering the reachable space of the robot. The decomposition need not be uniform. Indeed, it may be advantageous for some scenarios to have more cells in regions where dynamic obstacles appear most often. For simplicity, a grid G is used.

First, the axis-aligned box B delimiting the reachable space is determined. This can be done by hand. Alternatively, an iterative process can be used, which computes the union of the bounding boxes of a large number of random robot postures. B defines the extent of the grid.

Several tradeoffs influence the determination of G's resolution. In summary, fine resolutions allow for precisely describing affected locations in W, resulting in more portions of the roadmap remaining valid. However, they also result in more cells to be updated on line and more associated memory and precomputation time. We discuss grid resolution choices further in Section VIII.

Roadmap computation Our implementation is based on a standard PRM approach [3] [4], using a grid density control suited to our experiences with manipulators.

The configuration space C is randomly sampled and configurations lying in C_{free} are stored in a list L. We make use of the grid G to promote a uniform distribution of end-effector positions over W. Therefore, each cell c in G stores a counter of how many configurations with an end-effector joint inside c have been sampled. The sampling stops when the counters of most of the cells in G have reached their limits.

For each stored configuration q in L, we identify its k nearest configurations $q_i \in L$, $i \in \{1,\ldots,k\}$. Each valid connection between q and q_i, becomes an edge of the roadmap. A connection is considered valid if all configurations between q and q_i are also valid, within a desired precision. Linear interpolation between the parameters of q and q_i is used in order to determine configurations in between. Validity tests involve collision detection and are expensive. Faster tests are obtained with recursive bisection, allowing for early discarding invalid connections [4]. Alternatively, a precise method to test the validity of edges is available [23]. In the experiments presented here, k=6 was used.

Single nodes, those unconnected to another node in the roadmap, are discarded. The roadmap might contain several components, however as it is being constructed for a robot without considering many obstacles (only the static ones), a single connected component is expected.

In all performed validity tests, collision detection is performed using the VCollide package [24]. In order to detect if the robot at a given configuration is collision-free, all of its geometric parts and obstacles are pair-wise tested. As an optimization, we identify and deactivate pairs that are detected in advance to never intersect each other. Pair identification is determined by the user, observing the statistics of full collision tests over several random robot configurations.

Cell localization We say that a cell $c \in G$ invalidates a configuration $q \in R$, if, when c is considered as an obstacle in W, c collides with the robot at configuration q. Note that q might be a node of R, or in an edge of R.

For each cell $c \in G$, lists containing references to all elements (edges and nodes) of R containing configurations invalidated by c are stored. These lists are precomputed and not changed at run-time.

As precomputation time is not a primary issue, a simple brute force algorithm to localize cells was implemented. For each cell $c \in G$, collision detection is performed over the entire roadmap in order to determine all nodes and edges invalidated by c. A more efficient implementation could hierarchically test entire groups of cells, for instance following an *octree* hierarchical subdivision, or exploit cell adjacency coherence as proposed in [2]. Graphics hardware acceleration can also be employed, as viewing frustums can be defined having a cell shape. Alternatively, cells can be independently localized using parallel computation.

V. Roadmap Maintenance

During run-time, the environment is observed and all changes in the dynamic obstacles are tracked. Changes occur when obstacles appear, disappear, or change position. In all cases, two kinds of update operations are performed: cell occupation in case an object is detected to appear, and cell liberation in case an object is detected to disappear. Obstacle motions are treated with consecutive liberation and occupation operations.

Cells are updated per object. Let O_d be the set of objects disappearing and O_a be the set of objects appearing in W at a particular point in time. For each object in O_d and O_a, the cells occupied by the object are determined and sent to the respective update operation. Therefore, each update operation receives one cell as input.

In order to determine the cells occupied by an object, we simply take all cells intersected by the bounding box of the object. In case bounding boxes do not approximate dynamic objects well, precise and fast rasterization procedures can be devised using graphics hardware acceleration.

Reference counting For a given cell c being occupied, we increment the reference counter stored in each node and edge invalidated by c. Note that if the same cell is being occupied at the same time by n obstacles, this cell occupation procedure is called n times for the same cell. Fig. 2b and 2d show two examples of the effect of occupying cells.

Analogously, for a cell c being liberated, the reference counter stored in each node and edge invalidated by c are decremented.

When a node or edge has a reference counter greater than zero, it is considered not safe and thus not allowed to be traversed during the graph search procedure at query time.

Validity retesting A cell gives an approximation of a part of an obstacle and actually it may not be totally occupied by the obstacle. A more precise approach to cell occupation requires retesting the validity of all edges and nodes associated with the cell. Only when the edges or nodes actually induce a collision with obstacles (not the cells), they are invalidated and marked as unsafe. No reference counting is performed. Note that validity does not have to be tested for nodes and edges that were previously invalidated.

During the liberation of a cell c, validity retesting is also performed for all edges and nodes referenced by c, which are marked as unsafe. Nodes and edges that no longer induce collision with objects are marked again as safe.

Note that during validity retesting the collision detection procedure does not need to check for self-collisions in the robot. As nodes and edges belong to R, they are guaranteed to not contain any self-collisions. Only collision tests involving the dynamic obstacles need to be performed.

VI. ROADMAP QUERY

Given an initial configuration q_i, and a goal configuration q_g, the maintained roadmap R is used to derive a path P in C_{free} connecting q_i and q_g.

Let $N(q_i)$ and $N(q_g)$ be the nearest nodes in R to q_i and q_g respectively. Three tests are performed in order to determine if P can be successfully retrieved from R. The first two tests check the validity of connecting q_i with $N(q_i)$, and q_g with $N(q_g)$. The third test checks if there exists a safe path in R, i.e., trough valid edges, connecting $N(q_i)$ and $N(q_g)$. An A*-like graph search [7] is used to look for paths in R. If the three tests are successful, P is successfully found.

We say that the desired path is broken if any of the tests fail, meaning that R failed to solve the desired query alone. In such case, additional computation must be performed in order to try to solve the query. We make use of a bi-directional RRT planner to solve failed queries on line [5].

In case the path is broken in more than one location, we run the RRT with inputs q_i and q_g. If the path is broken only once, three cases can happen. If only the first test fails, we run the RRT with inputs q_i and $N(q_i)$. Analogously, if only the second test fails, we use as inputs q_g and $N(q_g)$. Finally, if the third test fails, the inputs are $N(q_i)$ and $N(q_g)$. The final path is a composition of paths retrieved from the roadmap and the RRT.

It is possible to implement strategies that attempt to reuse paths that are broken in several locations. A well-suited approach is to take the broken path, which has several edges marked as unsafe, and incrementally sample new nodes to connect to R until a safe path can be retrieved. Sampling can be biased to favor locations near to the unsafe nodes and edges, following the lazy evaluation approach proposed in [10]. We concentrate in this work on understanding the tradeoffs of maintaining DRMs and leave the investigation of such a method for future work.

Once a solution is found, a smoothing procedure is applied in order to improve the quality of the obtained path. A simple procedure iteratively replaces portions of the path by straight connections, always ensuring that the path remains valid.

VII. EXPERIMENTS

We performed experiments with three different scenarios. The first scenario used a 4 DOF manipulator arm (Fig. 2a), which was tested with grids of three different resolutions. In this scenario, 4 boxes were considered as dynamic obstacles. Fig. 2b illustrates the used DRM with the 24×32 grid. Each roadmap node drawn in the image represents the position of the wrist joint.

The second scenario used a manipulator with two arms (Fig. 2c), with a total of 7 DOF (the base has a 1 DOF rotation). Again 4 boxes were considered as dynamic obstacles. Two different grids were tested with this model. Figure 2d illustrates the used DRM with a grid of resolution of 20×20. In this figure, for each node of the roadmap, the two points corresponding to the position of each wrist joint are drawn in different colors.

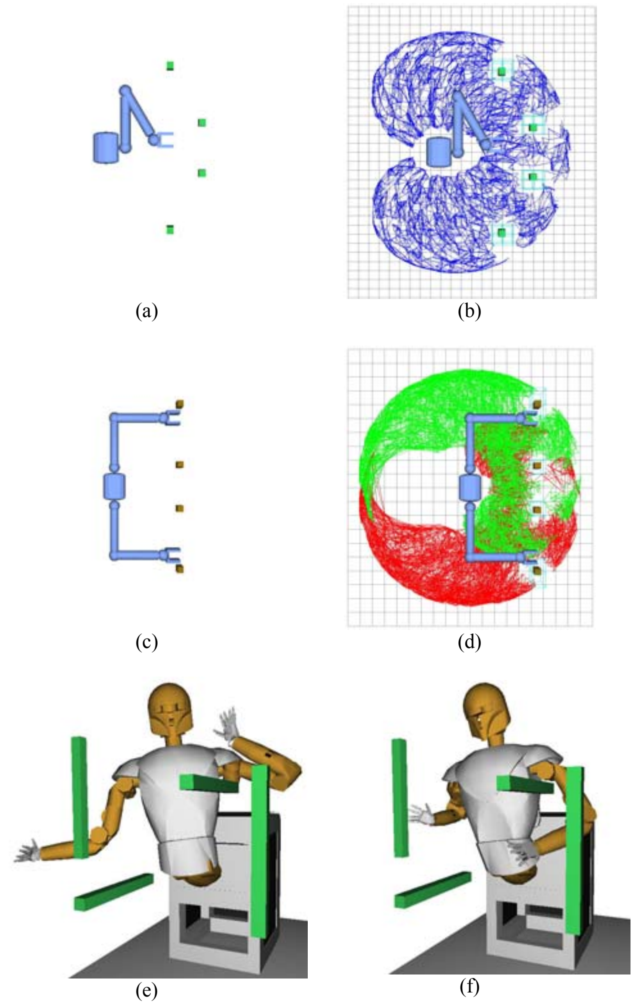

Figure 2. Scenarios used in our experiments.

The last scenario used the Robonaut humanoid model and 4 bar-like dynamic obstacles. The dynamic obstacles are allowed to randomly move in a defined region in front of the robot. Robonaut model has 17 DOF: 7 for each arm and 3 at the base. Two roadmaps with different sizes were tested (see Table I). The one with fewer nodes is illustrated in Fig. 1 (with the affected edges hidden according to the obstacles).

Each experiment consisted of solving 100 random problems with both DRM (coupled with RRT) and RRT (alone). Each random problem was defined by randomly positioning each obstacle in the workspace, and randomly generating initial and goal configurations for the robot. As an example, Fig. 2e and 2f show the initial and goal postures, respectively, for a problem generated randomly with the Robonaut model.

Full collision checking was performed in order to ensure that the problems are valid. Also, to have more meaningful

TABLE I. EXPERIMENTS RESULTS

Experiment		**DRM**						**RRT**		**Comparison**
scenario	***problems***	***grid***	***nodes***	***links***	***time(secs)***	***alone***	***accuracy***	***time(secs)***	***accuracy***	***RRT / DRM***
one arm	79	24×32	2218	7467	12.0	90%	100%	59.3	97%	4.9
one arm	66	64×70	2406	8069	17.1	92%	100%	146.8	68%	8.6
one arm	78	48×64	6661	22431	45.9	92%	99%	72.4	95%	1.6
two arms	65	20^2	3146	10629	18.2	86%	100%	151.3	86%	8.3
two arms	66	40^2	2898	9838	17.1	92%	100%	146.8	83%	8.6
Robonaut	99	24^3	3575	12312	47.9	36%	100%	63.1	98%	1.3
Robonaut	100	24^3	5144	17707	45.8	43%	99%	51.8	100%	1.1

problems, only those requiring planning were kept, i.e., those trivially solvable by direct interpolation were discarded.

The 100 problems generated per experiment were valid and non-trivial. However, some of them were impossible to solve. We considered a problem "impossible" when it could not be solved after 10 seconds by either of the methods. Table I specifies the actual number of problems solved per experiment.

In Table I, the DRM is said to solve one problem alone when it was capable of extracting the motion from the roadmap without any extra planning (see section VI). The times shown in the table for the DRM include the roadmap maintenance according to the motion of all dynamic obstacles and completely solving the query, including the time taken by extra planning when the DRM was not used alone.

An accuracy value for the methods was computed according to the number of times they failed. Failure was considered to occur when a method could not solve the problem before the 10 seconds limit. In each failure, the 10 seconds spent were included in the time measurement. When both methods failed, the problem was considered impossible.

VIII. ANALYSIS AND RESULTS

The validity retesting method is not a good option for the type of scenarios used in this work. In all experiments, it gave slower results than the reference counting update method. In some cases, it was even slower than applying the RRT planner alone. For this reason we focused on the reference counting method, which was used in all experiments reported in Table I. We believe however that validity retesting may provide good performance in more complex environments, where extra valid edges in the roadmap may avoid the need for planning difficult motions on line.

The first three rows in Table I show that a finer grid with a smaller roadmap was more efficient. Finer grids give a more precise approximation of obstacles. Smaller roadmaps have fewer nodes and edges to be updated. It is not worth using a huge roadmap if a smaller one is sufficient to represent the free space of a given dynamic scenario.

In the fourth and fifth rows, the finer grid also had better performance than the coarser grid, but the performance gain was much smaller. This indicates that there might be an optimum grid resolution and roadmap size to choose in each scenario. The right choice will depend on the complexity of the dynamic environment, number of updates, and number of queries required. Finer grids are more precise but require more cells to be validated. Note that a single cell may invalidate a very large portion of the roadmap, increasing the cost of maintaining several cells in finer grids.

The performed tests clearly demonstrate the superior performance of DRMs in the planar scenarios. However, in the scenarios using the Robonaut model, the DRM introduced only a modest speed gain. The main cause might be due to the higher number of DOF in the Robonaut model, requiring much larger roadmaps to adequately cover the volume of the free configuration space. However, if overly large roadmaps are used, the maintenance and query steps become too costly. For instance, we observed worse performances when using roadmaps with more than 5000 nodes. The use of 3D workspaces also increases the maintenance cost, as many more cells must be updated.

Contrary to the planar scenarios, the DRM was used alone in the Robonaut scenarios only in 36% and 43% of the tests. These low values confirm that the used roadmaps are not capable of adequately covering the free configuration space. However, even with few problems actually making use of the maintained roadmaps, the method was still advantageous. This fact indicates that optimizations in the roadmap query and maintenance are worth exploring in order to allow the use of larger roadmaps. The success of a DRM directly relates to the number of problems it can solve alone.

Another important observation is that the accuracy of the RRT in the Robonaut scenario was very high, showing its efficiency for finding solutions in that scenario. The planar scenarios were much harder for the RRT to solve. Indeed, in a 3D workspace, the manipulators have many more possibilities to turn around obstacles.

One important advantage of DRMs is to in finding difficult solutions. For example, the RRT needs several seconds to find motions for extreme configurations, such as for Robonaut with its two hands on its back. Extreme postures tend to be present in the precomputed roadmap with our sampling strategy, and can even be included by the user. Moreover, advanced techniques are available to build roadmaps able to efficiently recover the connectivity of difficult configuration spaces [9]. Note that comparing performances is a delicate task as a single difficult problem for one method can make the difference in the overall results. The relatively low limit of 10 seconds to decide if a problem is impossible was chosen in order to minimize the impact of such cases in our evaluation.

The planners were implemented in a fairly standard manner, without any specific optimizations. For instance, we did not make use of dedicated data structures for finding nearest neighbors (e.g., *k-d* trees) in any of the methods. The bi-directional RRT implementation followed the few modifications proposed by [9].

All computation was performed on a 2.8GHz Pentium 4 processor with 1GByte of memory. Although the memory requirements were high, the computer had no problems in storing our grids and we did not need any compression techniques as proposed in [2]. In general, the precomputation time of the DRM took from a few minutes to several hours. The three-dimensional grids, however, took approximately 3 days to compute. Most of the precomputation time was due to our brute force algorithm used in the cell localization phase. The alternatives discussed in Section IV can be applied and are being experimented on in order to perform further tests with finer grids.

IX. CONCLUSIONS

We have shown that the maintenance of precomputed roadmaps can lead to faster and more accurate results than performing single-query motion planning alone.

In the Robonaut scenario, the method had only a modest speed gain. However, the much better performance obtained in the planar scenarios motivates further experimentation with the method. Other types of scenarios can lead to better performances of DRMs. Factors such as the complexity of the environment, the number of DOF, and the number and size of the dynamic obstacles have an important impact on the performance of the method.

We believe that the maintenance of embedded motion knowledge is an important issue to be considered for interactive humanoid motion control.

The main contribution of this work is the implementation of several experiments that give more insight into the advantages and drawbacks of maintaining roadmaps, when compared against using a single-query motion planner alone.

As future work we intend to develop a faster cell localization procedure and to integrate procedures for the continuous insertion of temporary edges and nodes, better adapting the roadmap to the changing environment.

ACKNOWLEDGMENTS

This work was supported by DARPA grant DABT63-99-1-0015 and DARPA MARS 2020 grant NAG9-1444.

REFERENCES

[1] J. F. Stein, "The representation of egocentric space in the posterior parietal cortex", Behavioral and Brain Science, 15, 1992, pp. 691-700.

[2] P. Leven, and S. Hutchinson, "Toward real-time path planning in changing environments", Proc. of the fourth International Workshop on the Algorithmic Foundations of Robotics (WAFR), March 2000, pp. 363-376.

[3] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for fast path planning in high-dimensional configuration spaces", IEEE Transactions on Robotics and Automation, 12(4):566-580, June 1996.

[4] R. Geraerts and M. Overmars, "A comparative study of probabilistic roadmap planners", Proc. of the International Workshop on Algorithmic Foundations of Robotics (WAFR), Nice, France, 2002.

[5] J. J. Kuffner and S. M. La Valle, "RRT-connect: an efficient approach to single-query path planning", Proc. of the International Conference on Robotics and Automation (ICRA), San Francisco, 2000, pp. 995-1001.

[6] R. O. Ambrose, H. Alridge, R. S. Askey, R. R. Burridge, W. Bluethmann, M. Diltler, C. Lovchik, D. Magruder, and F. Rehnmark, "Robonaut: NASA's space humanoid", IEEE Intelligent Systems, 15(4):57-62, August 2000.

[7] J.-C. Latombe, "Robot motion planning", ISBN 0-7923-9206-X, Kluwer, Academic Publishers, 1991.

[8] C. Nissoux, T. Simeon and J. P. Laumond, "Visibility based probabilistic roadmaps", Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS), Korea, 1999, pp. 1316-1321.

[9] K. E. Bekris, B. Y. Chen, A. M. Ladd, E. Plakue, and L. E. Kavraki, "Multiple query probabilistic roadmap planning using single query planning primitives", Proc. of the International Conference on Intelligent Robots and Systems (IROS), Las Vegas, USA, 2003, pp. 656-661.

[10] R. Bohlin and L. Kavraki, "Path planning using lazy PRM", Proc. of the International Conference on Robotics and Automation (ICRA), San Francisco, USA, 2000, pp. 521-528.

[11] S. M. La Valle and M. S. Branicky, "On the relationship between classical grid search and probabilistic roadmaps", Proc. of the International Workshop on the Algorithmic Foundations of Robotics (WAFR), Nice, 2002.

[12] S. M. La Valle, "Rapidly-exploring random trees: a new tool for path planning", Technical Report TR 98-11, Computer Science Dept., Iowa State University, Oct. 1998.

[13] D. Hsu, J.-C. Latombe, and R. Motwani, "Path planning in expansive configuration spaces", International Journal of Computational Geometry and Applications, 9(4-5), 1999, pp. 495-512.

[14] G. Sanchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking", Robotics Research: The Tenth Int. Symp., R.A. Jarvis and A. Zelinsky (eds.), Springer Tracts in Advanced Robotics, Springer, pp. 403-417, 2003.

[15] C. Nielsen, and L. E. Kavraki, "A two-level fuzzy PRM for manipulation planning", Proc. of the International Conference on Intelligent Robots and Systems (IROS), Japan, 2000, pp. 1716-1722.

[16] T.-Y. Li, and Y.-C. Shie, "An incremental learning approach to motion planning with roadmap management", Proc. of the International Conference on Robotics and Automation (ICRA), Washington, 2002.

[17] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning collision-free reaching motions for interactive object manipulation and grasping", Proc. of Eurographics, Granada, Spain, 2003, pp. 313-322.

[18] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Footstep planning among obstacles for biped robots", Proc. of the International Conference on Intelligent Robots and Systems (IROS), Maui, 2001.

[19] M. G. Choi, J. Lee, S. Y. Shin, "Planning biped locomotion using motion capture data and probabilistic roadmaps", ACM Transactions on Graphics (TOG), 22(2), 2003, pp. 182-203.

[20] J. Pettre, J.-P. Laumond, T. Simeon, "A 2-stages locomotion planner for digital actors", Eurographics/SIGGRAPH Symposium on Computer Animation, San Diego, USA, 2003, pp. 258-264.

[21] J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue, "Dynamic humanoid H6: fast manipulation path planning for humanoid robots", Proceedings of the 18th Annual Conference of the Robotics Society of Japan (RSJ), 2000.

[22] Y. Koga, K. Kondo, J. Kuffner, and J. Latombe, "Planning motions with intentions", Proc. of SIGGRAPH, 1994, pp.395-408.

[23] F. Schwarzer, M. Saha, and J.-C. Latombe, "Exact Collision Checking of Robot Paths", In Algorithmic Foundations of Robotics V, J.D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson (eds.), Springer Tracts in Advanced Robotics, Springer, 2004, pp. 25-41.

[24] S. Gottschalk, M. C. Lin, and D. Manocha, "OBBTree: a hierarchical structure for rapid interference detection", Proc. of SIGGRAPH, 1996, pp. 171-180.