Anytime Dynamic Path-Planning with Flexible Probabilistic Roadmaps

Khaled Belghith*, Froduald Kabanza*, Leo Hartman† and Roger Nkambou‡
*Université de Sherbrooke
Email: khaled.belghith@USherbrooke.ca, kabanza@USherbrooke.ca
†Canadian Space Agency
Email: Leo.Hartman@space.gc.ca
‡Université du Québec À Montréal
Email: nkambou.roger@uqam.ca

***Abstract*— Probabilistic roadmaps (PRM) have been demonstrated to be very promising for planning paths for robots with high degrees of freedom in complex 3D workspaces. In this paper we describe a PRM path-planning method presenting three novel features that are useful in various real-world applications. First, it handles zones in the robot workspace with different degrees of desirability. Given the random quality of paths that are calculated by traditional PRM approaches, this provides a mean to specify a sampling strategy that controls the search process to generate better paths by simply annotating regions in the free workspace with degrees of desirability. Second, our approach can efficiently re-compute paths in dynamic environments where obstacles and zones can change shape or move concurrently with the robot. Third, it can incrementally improve the quality of a generated path, so that a suboptimal solution is available when required for immediate action, but get improved as more planning time is affordable.**

I. Introduction

Path planning has been extensively studied in robotics over the past two decades and is increasingly becoming important in other areas such as computer animation, computer aided design, and medical robotics equipments (e.g., for surgery). In its traditional form, the path planning problem is to plan a path for a moving body (typically a robot) from a given start configuration to a given goal configuration in a workspace containing a set of obstacles. The basic constraint on solution paths is to avoid collision with obstacles, which we call hereby a hard constraint. There exist numerous approaches for path planning under this constraint (e.g., see [1]–[5]).

In many complex applications, however, in addition to obstacles that must be avoided, we may have areas that must be avoided as much as possible. That is, a path going through these areas is not highly desirable, but would be acceptable if no better path exists or can be computed efficiently. The danger concept is relevant for example in military applications. Some path planning techniques that deal with it have been proposed, including [6], [7]. Conversely, it may be desirable for a path to stay close to certain areas as much as possible. Even if there is no explicit notion of degree of desirability for a given path planning problem, as we demonstrate later, introducing it provides a mean to control the quality of a path generated by a PRM method. Indeed, PRM paths are obtained by connecting milestones that are randomly sampled in the free workspace, and this tends to yield awkward paths, requiring heuristic post-processing operations to smooth them. We can influence the sampling strategy to generate less awkward paths by specifying zones we prefer them to go through. We call such constraints on desirable zones (or undesirable ones) soft constraints on the robot path.

Our approach consists in defining a sampling strategy that implements a balance between a random sampling of the free workspace, keeping the robot configuration in desirable zones, and minimizing the distance to the goal. More specifically, the strategy emulates an anytime dynamic A* (AD* algorithm [8]). We apply AD* to implement a sampling strategy for fast replanning and anytime response in PRM approaches and extend it further to zones with degrees of desirability.

In the next section we give the most relevant background. We then describe our PRM approach, followed with experiments that illustrate some of its merits.

II. Background

A configuration q of an articulated robot with n degrees of freedom (dof) is an n element vector of the robot joint positions. Since the robot moves by changing its joint rotations or translations, a path between two points is a sequence of configurations sufficiently close together connecting the two points. A path is collision-free if the robot does not collide with any obstacle in the workspace in any of the configurations on the path.

Grid-based approaches provide one approach for implementing path planning algorithms. The configuration space is decomposed into an occupancy grid of cells; a path from an initial cell to a goal cell is then found by searching a sequence of moves between adjacent free cells, connecting the initial configuration to the goal [4], [9], [10]. These moves correspond to possible edges in a graph with nodes corresponding to free cells in the grid. Graph-search algorithms such as AD* [8] can be used to compute a path between two configurations.

To obtain correct solutions, grid decompositions must use a sufficient level of precision; the smaller the cell size is, the more precise the solution is. This results in large search spaces, particularly for high dimensional configuration spaces. In this case, a better exploration strategy for the occupancy grid, maybe to use random search [11]. While this may help coping with the complexity of the configuration space, in very large configuration spaces the planner spends a large amount of time generating the occupancy grid [12].

Probalistic roadmaps (PRM), introduced in [1], avoid this overhead of the grid decomposition by sampling configuration space probabilistically. Given a 3D model of obstacles in the workspace, a 3D model of the robot, a PRM planner builds a graph of nodes corresponding to robot configurations in the free workspace, by picking configurations randomly and checking that they are collision-free, using a fast collision detection method (called a local planner) to check that an edge between two adjacent nodes is also collision-free; each time a local planner succeeds, the corresponding edge (i.e., local path or path segment) is inserted into the roadmap. The graph built that way is called a probabilistic roadmap and is a simplified representation of the free configuration space. The graph can thus be explored using familiar graph-search methods.

In the original PRM approach, given that a roadmap was pre-computed before searching for a path, it could be sorted for multiple queries. However, it was later observed that PRM planners spent most of their time performing collision checks for edges between nodes [2], [3], therefore it was a waste of time to pre-compute a complete roadmap. This prompted for lazy collision PRM approaches, which delay the collision detection along edges until a solution path is found [2], [3]. Lazy approaches re-build the roadmap for each query, that is, they are single query approaches.

A PRM planner selects a node to expand in the free configuration space according to some given sampling measure. The efficiency of PRM approaches significantly depends on this measure [12]. A good measure should take visibility properties into account but such properties can be very difficult to formalize, particularly in dynamic environments.

Another important issue is dealing with the quality of paths, for instance paths that optimize degrees of desirability of regions in the free configuration space. Given that PRM approaches are today considered the most efficient path planning approaches for high dimension configuration spaces [12] as we consider, we decided to explore a solution within the PRM framework. We thus developed a sampling method that takes into account degrees of desirability of regions in the workspace.

The applications we consider involve a dynamically changing workspace. Such changes may invalidate a previously calculated path, either because it is no longer optimal, or simply because it now leads to a dead-end. From the roadmap perspective, this means that the cost of a segment between two milestones can change dynamically. We thus need to keep track of milestones in an optimal solution to the goal. When changes are noticed, edges costs are updated, and a new roadmap is re-computed fast, starting from the goal, taking into account previous traces of the path-calculation. This brings us back to a method in between the multiple query approach and the single query approach. The difference with a multiple query approach is that we are just concerned with the roadmap to a given goal, that is, the goal the robot is trying to reach in a dynamic environment. Our new PRM path-planning algorithm can be stopped at any time and give a solution (more precisely anytime after the time necessary for a first solution), and the more time it is given, the better the solution is; that is, it has an anytime planning capability [13].

III. FADPRM Planner Algorithm

Our approach, called Flexible Anytime Dynamic PRM (FADPRM), uses a sampling strategy that implements a balance between: an anytime dynamic A* (AD*) exploration of the roadmap; and a random sampling as in a normal PRM. AD* [8] is a generalization of the familiar A* graph search algorithm, to deal with dynamically changing edges; changes of edges in robot navigation problems correspond to changes in the workspace. AD* has also the anytime planning capability to provide a solution path quickly and incrementally improve its quality if more planning time available.

A. Algorithm Sketch

FADPRM works with a free workspace that is segmented into zones, each zone being assigned a degree of desirability (dd), that is, a real number in the interval [0 1]. The closer is dd to 1, the more desirable the zone is. Every configuration in the roadmap is assigned a dd equal to the average of dd of zones overlapping with it. The dd of a path is an average of dd of configurations in the path. An optimal path is one having the highest dd.

The input for FADPRM is thus: an initial configuration, a goal configuration, a 3D model of obstacles in the workspace, a 3D specification of zones with corresponding dd, and a 3D model of the robot. Given this input:

1) To find a path connecting the input and goal configuration, we search backward from the goal towards the initial (current) robot configuration. Backward instead of forward search is done because the robot moves, hence its current configuration is not the initial configuration, but the goal remains the same; we want to re-compute a path to the same goal but from the current position whenever the environment changes before the goal is reached.
2) A probabilistic priority queue $OPEN$ contains nodes on the frontier of the current roadmap (i.e., nodes that need to be expanded because they have no predecessor yet; or nodes that have been previously expanded but are not update anymore) and a list $CLOSED$ contains non frontier nodes (i.e., nodes already expanded)
3) Search consists in repeatedly picking a node from $OPEN$, generating its predecessors and putting the new ones and the not updated ones in $OPEN$.
    a) Every node n in $OPEN$ has a key priority proportional to the node's density and best estimate to the goal. The density of a node n, $density(n)$, reflects

the density of nodes around n and is the number of nodes in the roadmap with configurations that are a short distance away. The estimate to the goal, $f(n)$, takes into account the node's dd and the Euclidean distance to the goal configuration as explained below. Nodes in $OPEN$ are selected for expansion in decreasing priority. With these definitions, a node n in $OPEN$ is selected for expansion with priority proportional to

$$(1-\beta)/density(n) + \beta * f(n),$$

β is the inflation factor with $0 \leq \beta \leq 1$.

b) To increase the resolution of the roadmap, a new predecessor is randomly generated within a short neighborhood radius (the radius is fixed empirically based on the density of obstacles in the workspace) and added to the list of predecessors in the roadmap generated so far; then the entire list of predecessors is returned.

c) Collision is delayed: detection of collisions on the edges between the current node and its predecessors is delayed until a candidate solution is found; if colliding, we backtrack and rearrange the roadmap by eliminating nodes involved in this collision.

4) The robot may start executing the first path found.
5) Concurrently, the path continues being improved.
6) Changes in the environment (moving obstacles and zones or changes in dd for zones) cause update of the roadmap and replanning.

With β equal to 0, the selection of a node to expand is totally blind to zone degrees of desirability and to edges costs (Euclidian distance). Assuming $OPEN$ is the entire roadmap, this case corresponds to a normal PRM, and the algorithm probabilistically converges towards an optimal solution as is the case for a normal PRM [3]. With $\beta = 1$, the selection of a node is a best-first strategy and by adopting an A*-like $f(n)$ implementation, we can guarantee finding an optimal solution within the resolution of the roadmap sampled so far. Therefore the expression $(1-\beta)/density(n) + \beta * f(n)$ implements a balance between fast-solution search and best-solution search by choosing different values for β.

Values of β closer to 1 give better solutions, but take more time. An initial path is generated fast assuming a value close to 0, then β is increased by a small quantity, a new path is computed again, and so on. At each step, we have a higher probability of getting a better path (probability 1 when β reaches 1). This is the key in the anytime capability of our algorithm.

The heuristic estimate is separated into two components $g(n)$ (the quality of the best path so far from n to the goal configuration) and $h(n)$ (estimate of the quality of the path from n to the initial configuration), that is, $f(n) = g(n) + h(n)/2$; we divide by 2 to normalize $f(n)$ to values between $[0, 1]$. This definition of $f(n)$ is as in a normal A* except that:

- We do backward search, hence $g(n)$ and $h(n)$ are reversed (here we follow the AD* [8] strategy).
- The quality of a path is a combination of its dd and its cost in terms of distance traveled by the robot. Given $pathCost(n, n')$ the cost between two nodes, $g(n)$ is defined as follows:
  $g(n) = pathdd(n_{goal}, n)/(1 + \gamma.pathCost(n_{goal}, n))$
  with $0 \leq \gamma \leq 1$.
- The heuristic $h(n)$ is expressed in the same way as $g(n)$ and estimates the cost of the path remaining to reach n_{start}:
  $h(n) = pathdd(n, n_{start})/(1 + \gamma.pathCost(n, n_{start}))$

The factor γ determines the influence of the dd on $g(n)$ and on $h(n)$. With $\gamma = 0$, nodes with high dd are privileged, whereas with $\gamma = 1$ and with the dd of all nodes equal to 1, nodes with least cost to the goal are privileged. In the last case, if the cost between two nodes $pathCost(n, n'))$ is chosen to be the Euclidean distance, then we have an admissible heuristic and the algorithm guarantees converge to the optimal solution (this follows from a property of AD*, which in turn is inherited from A*). When dds are involved, and since zones can have arbitrary configurations, it's difficult to define admissible heuristics. The algorithm guarantees improvement of the solution, but it's impossible to verify optimality. Since the dd measures the quality of the path, the idea is to run the algorithm until a satisfactory dd is reached. The functions $pathdd$ and $pathCost$ are implemented by attaching these values to nodes and updating them on every expansion.

B. Detailed Algorithm

The detailed description of FADPRM is shown in Algorithm 1, adopting the same structure as AD* [8].

FADPRM stores a one-step lookahead cost $rhs(n)$ which satisfies the following two conditions:

$$rhs(n) = max_{n' \in Succ(n)}(c(n, n') + g(n'))$$
$$rhs(n_{goal}) = 0,$$

with $Succ(n)$ the set of successors of n in the roadmap. A node n is then consistent if $g(n)$ equals $rhs(n)$. Otherwise it is overconsistent (if $g(n) < rhs(n)$) or underconsistent (if $g(n) > rhs(n)$).

Nodes in $OPEN$ are expanded in decreasing priority in order to update their g-values and their predecessors's rhs-values. When edge costs change within the roadmap, FADPRM updates the rhs-values of the nodes affected by these changes and put those that have been made inconsistent into $OPEN$, so that they become considered again by further state expansions, and hence have their optimal path to the goal updated. The ordering of nodes in $OPEN$ is based on a node priority $key(n)$, which is a pair $[k_1(n), k_2(n)]$ defined as follows:

$$key(n) = [(1-\beta)/density(n) + \beta \times f(n),\ max(g(n), rhs(n))],$$

with $f(n) = (max(g(n), rhs(n)) + h(n_{start}, n))/2$ and $key(n) \leq key(n')$ if $k_1(n) \leq k_1(n')$ or $(k_1(n) = k_1(n')$ and $k_2(n) \leq k_2(n'))$.

Algorithm 1 The FADPRM Algorithm

```
01. KEY($n$)
02.   $f(n) = [max(g(n), rhs(n)) + h(n)]/2$;
03.   if($g(n) < rhs(n)$)
04.     return $[(1-\beta)/density(n) + \beta.f(n); max(g(n), rhs(n))]$
05.   else
06.     return $[f(n); max(g(n), rhs(n))]$

07. UPDATESTATE($n$)
08.   if $n$ was not visited before
09.     $g(n) = 0$;
10.   if $(n \neq s_{goal}) rhs(n) = max_{s' \in Succ(n)}(c(n, n') + g(n'))$;
11.   if $(n \in OPEN)$ remove $n$ from $OPEN$;
12.   if $(g(n) \neq rhs(n))$
13.     if $(n \notin CLOSED)$
14.         insert $n$ into $OPEN$ with $key(n)$;
15.     else
16.         insert $n$ into $INCONS$;

17. COMPUTEORIMPROVEPATH()
18.     while $(NoPathfound)$
19.       remove $n$ with the maximum key from $OPEN$;
20.       if $(Connect(u, n_{start}))$
21.         return $\beta$-suboptimal path ;
22.         break;
23.       else
24.         $expandnode(n)$;
25.         if $(g(n) < rhs(n))$
26.             $g(n) = rhs(n)$;
27.             $CLOSED = CLOSED \bigcup \{s\}$;
28.             For all $n' \in Pred(n)$
29.               UpdateState($n'$);
30.         else
31.             $g(n) = 0$;
32.             For all $n' \in Pred(n) \bigcup \{s\}$
33.               UpdateState($n'$);

34. MAIN()
35.     $g(n_{start}) = rhs(n_{start}) = 0; g(n_{goal}) = 0$;
36.     $rhs(n_{goal}) = \infty; \beta = \beta_0$;
37.     $OPEN = CLOSED = INCONS = \emptyset$
38.     insert $n_{goal}$ into $OPEN$ with $key(n_{goal})$;
39.     while (Not collision-free Path)
40.         Rearrange Tree;
41.         ComputeorImprovePath();
42.     publish current $\beta-$suboptimal solution;
43.     while ($n_{start}$ is not in the neighborhood of $n_{goal}$)
44.         if $n_{start}$ changed
45.             if addtoTree($n_{start}$)
46.                 publish current $\beta-$suboptimal solution;
47.         if changes in edge costs are detected
48.             for all edges $(u, v)$ with changed edge costs
49.                 Update the edge cost $c(u, v)$;
50.                 UpdateState($u$);
51.         if significant edge cost changes were observed
52.             decrease $\beta$ or replan from scratch;
53.         if $\beta < 1$
54.             increase $\beta$;
55.         Move states from $INCONS$ to $OPEN$;
56.         Update the priorities for all $n \in OPEN$
57.             according to Key(n);
58.         $CLOSED = \emptyset$;
59.         while (Not collision-free Path)
60.             Rearrange Tree;
61.             ComputeorImprovePath();
62.         publish current $\beta-$suboptimal solution;
63.         if $\beta = 1$
64.             wait for changes in edges cost;
```

The function Main in FADPRM first sets the inflation factor β to a low value β_0, so that a suboptimal plan can be generated quickly (lines 35-42). Then if no changes in edge costs are detected, β is increased to improve the quality of its solution (lines 53-62). This will continue until the maximum of optimality is reached with $\beta = 1$ (lines 63-64).

According to the function UpdateState(n), when a node has been chosen for expansion during a particular search, if it becomes inconsistent due to a cost change in a neighboring node, it is not inserted into $OPEN$. Instead, it is placed into $INCONS$ which contains all inconsistent states. Then, when the current search terminates, all nodes in $INCONS$ are moved to a new $OPEN$ list (lines 55-56).

In the function ComputeorImprovePath(), when a node n with maximum key is extracted from $OPEN$, we first try to connect it to n_{start} using a fast local planner as in SBL [3]. If it succeeds, a path is then returned (Line 21). The expansion on a node n with maximum key from the $OPEN$ (Line 24) consists, as we said in the previous section, in sampling a new collision-free node in the neighborhood of n [3], then the sampled node takes apart in the set $Pred(n)$.

Every time a β-suboptimal path is returned by ComputeorImprovePath(), it is checked for collision. If a collision is detected on one of the edges constituting the path, a rearrangement of the roadmap is then needed to eliminate nodes involved in this collision (Lines 39-41 and 59-61). This refers to the concept of Lazy collision checking as defined in PRM approaches (in instance with SBL [3]: the collision checking on edges is delayed until a solution path is found).

FADPRM also handles the case of floating n_{start} where the robot is moving towards n_{goal}: since all the nodes in $OPEN$ have their key values recalculated each time β is changed, the updated new n_{start} will be automatically taken into account.

IV. Experimental Results

The following experiments were run on a 2 GHZ Pentium IV with 512 MB of RAM. We consider paths with a dd of 0.5 to be neutral, below 0.5 to be dangerous and above to be desirable. More specifically, dangerous zones are given a dd of 0.2 and desirable ones a dd of 0.8. A free configuration of the robot not having any contact with zones is assigned a dd of 0.5. We use $path - dd$ as a measure for path quality. We assume $\gamma = 0.7$ in all experiments. PRM refers to MPK, the implementation of SBL [3]. The implementation of FADPRM is built on top of MPK.

We did experiments on a simulation of the Space Station Remote Manipulator System (SSRMS), that is, a 17 meter long articulated robot manipulator with seven rotational joints, currently mounted on the International Space Station (ISS) [14]. This is a very complex environment: SSRMS has 7 degrees of freedom and our ISS model contained almost 50 obstacles and 85000 triangles.

The concept of dangerous and desirable zones is motivated by a real-world application dealing with teaching astronauts to operate the SSRMS in order to move payloads or inspect the ISS using a camera mounted at the end effector. In this

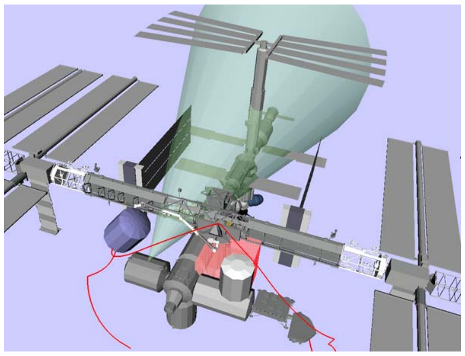
Fig. 1. SSRMS with a dangerous zone (cube) and a desired zone (cone)

case, astronauts have to move the SSRMS remotely, within safe corridors of operations. The definition of a safe corridor is that it must of course avoid obstacles (hard constraint), but also go as much as possible within regions visible through cameras mounted on the ISS exterior (so the astronaut can see the manipulations through a monitor on which the cameras are mapped). Hence, safe corridors depend on view angles and lighting conditions for cameras mounted on the ISS, which change dynamically with the orbit of the ISS by modifying their exposure to direct sunlight. As safe corridors are more complex to illustrate in paper, we just picked conical zones approximating cameras view regions and polygonal zones at arbitrary locations. Fig. 1 illustrates the robot carrying a load on a path calculated by FADPRM avoiding a dangerous zone (cube on the right) and going through a desirable zone (cone on the left), the latter corresponding to a camera field of view.

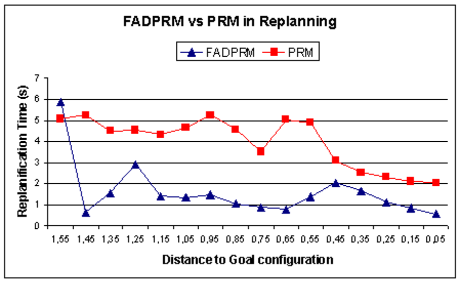

Fig. 2. FADPRM Vs PRM in Replanning

The first experiment illustrates the situation in which a human operator is learning to manipulate the SSRMS from a given start configuration to a given goal configuration. To provide feedback on whether he is on the right track, from every current configuration, we call FADPRM to calculate a path with a high dd to the goal. If no such a path exists, we notify the learner that he is moving the SSRMS to a dead end. Replanning occurs because the current configuration is moved arbitrary by the learner as he tries to reach the goal. Paths are computed to confirm the learner is on the right track, but they are not displayed to him. Hence the learner is not following a previously calculated path, but a path in his mind. Fig. 2 shows the time taken for replanning in the scenario of Fig. 1. Except for the first few iterations, FADPRM takes less re-planning time than PRM. In the first few iterations, the overhead incurred by the AD*-based sampling method dominates the planning time. In later iterations, it is outweighed by the savings gained by re-planning from the previous roadmap.

In Fig. 3, we compare the time needed for FADPRM and PRM to find a solution for 15 arbitrary queries in the ISS environment. As the time (and path quality) for a path is a random variable given the random sampling of the free workspace, for each query we ran each of the planners 10 times and reported the average planning time (in this case FADPRM was used in a mode not storing the roadmap between successive runs). Before displaying the results, we sorted the PRM setting in increasing order of complexity, starting with queries taking less time to solve. For FADPRM, we show results with $\beta = 0$, and $\beta = 0.4$. With $\beta = 0$, FADPRM behaves like PRM, which validates our previous analysis. With $\beta = 0.4$, FADPRM takes more time. On the other hand, Fig. 4 shows that $\beta = 0.4$ yields higher quality paths than $\beta = 0$. This validates another previous analysis higher β values yield better paths, but take more time to compute. On Fig. 4, FADPRM($\beta = 0$) setting is sorted in increasing order of $path - dd$.

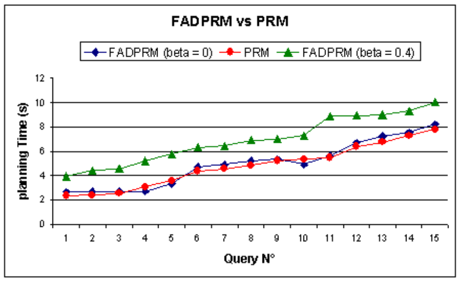

Fig. 3. FADPRM ($\beta = 0$, $\beta = 0.4$) Vs PRM in Replanning

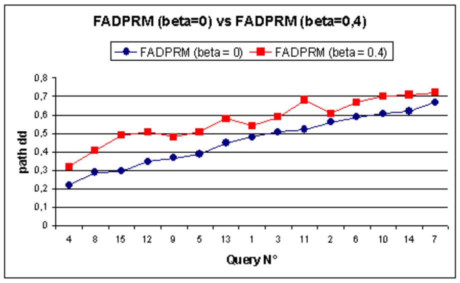

Fig. 4. FADPRM ($\beta = 0$) Vs FADPRM ($\beta = 0.4$) in Path Quality

We also did experiments on a simulated Puma robot (Fig. 5), operating on a car (from [3]), and the results are similar to those of the SSRMS experiment. Here too the environment is very complex: 6 degrees of freedom for the robot and approximately 7000 triangles. To illustrate the "search control" capability of FADPRM, in one experiment, we specified a small desirable zone (see Fig. 5-a), and in another, we specified a wider desirable zone (on the right) and a wide undesirable zone (on the left) (see Fig. 5-b). In both set experiments, we wanted to influence the sampling of the free workspace to yield paths that move the robot in front of the car (from left side, to the front, then to the right side). The paths shown on Fig. 5 illustrate the ideal solutions. By specifying a desirable zone on the right as in Fig. 5(a), and running FADPRM many times on the same query (input/goal configuration), FADPRM yielded better paths, on average, than PRM. On the other hand, by enlarging the size and coverage of the desirable zone and adding a undesirable zone (right, on the back of the car), as shown in Fig. 5(b), we noted that the quality of paths increased by a percentage of 50% over 100 trials.

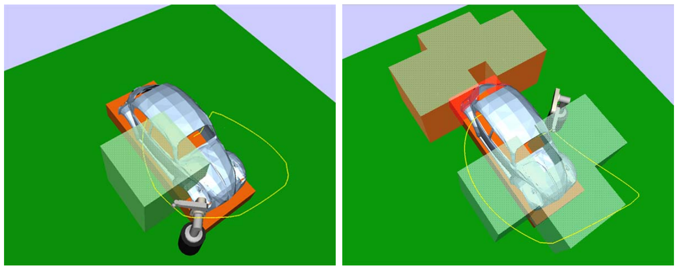
(a) (b)

Fig. 5. Puma robot and a car

The second experiment succeeds more often because the path is more constrained; a wider desirable zone on the front of the car together with an undesirable zone on the back of the car, make the probability of sampling a configuration along the desirable region higher than in the first set-up. Given the random quality of paths that are calculated by traditional PRM approaches, zones with degrees of desirability provide a mean to specify a sampling strategy that controls the search process to generate better paths by simply annotating the 3D workspace with region's degrees of desirability.

Fig. 6 illustrates the anytime capability of FADPRM on these two experiments. We notice the continuous improvement of the path quality ($path - dd$) for the two settings. The more time it is given, the better the path provided by FADPRM will be.

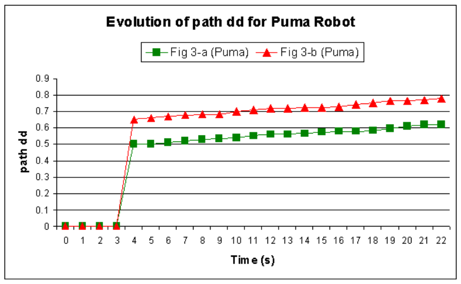

Fig. 6. Path quality evolution with FADPRM

V. Conclusion

We have just explained improvement to PRM path-planning approaches, along three dimensions: (1) modeling zones in the robot workspace with different degrees of desirability, (2) efficiently re-computing paths in dynamic environments and (3) anytime planning capability for real-time applications. Besides extending the range of problems solvable by PRM approaches along these dimensions, we have demonstrated that even for traditional problems addressed by PRM approaches, our extension can improve the quality of generated paths and compensate for the random search by providing desirable regions as a mean of controlling search.

In the Background section we mentioned that one key to the efficiency of PRM approaches is the sampling measure. This is a challenging problem and recent approaches include [5], [15], [16]. We also plan to address this problem by developing new sampling strategies within FADPRM.

References

[1] L. Kavraki, P. Svestka, J. C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," in *IEEE Transactions on Robotics and Automation*, vol. 12, 1996, pp. 566–580.
[2] R. Bohlin and L. Kavraki, "Path planning using lazy PRM," in *IEEE International Conference on Robotics and Automation*, 2000, pp. 521–528.
[3] G. Sanchez and J.-C. Latombe, "A single-query bi-directional probabilistic roadmap planner with lazy collision checking," in *Ninth International Symposium on Robotics Research*, 2001, pp. 403–417.
[4] D. Ferguson, M. Likhachev, and A. Stentz, "A guide to heuristic-based path-planning," in *ICAPS Workshop on Planning under uncertainty for Autonomous Systems*, 2005, pp. 9–18.
[5] M. Saha and J.-C. Latombe, "Finding narrow passages with probabilistic roadmaps : The small step retraction method," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2005.
[6] D. Sent and M. Overmars, "Motion planning in environments with dangerzones," in *IEEE International Conference on Robotics and Automation*, 2001, pp. 1488–1493.
[7] P. Melchior, B. Orsoni, O. Lavialle, A. Poty, and A. Oustaloup, "Consideration of obstacle danger level in path planning using A* and fast-marching optimisation: comparative study," in *Signal Processing*, vol. 83, no. 11, 2003.
[8] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic A*: An anytime, replanning algorithm," in *International Conference on Automated Planning and Scheduling (ICAPS)*, 2005.
[9] S. Koenig and M. Likhachev, "Improved fast replanning for robot navigation in unknown terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2002.
[10] M. Likhachev, G. Gordon, and S. Thrun, "ARA*: Anytime A* search with provable bounds on sub-optimality," in *Conference on Neural Information Processing Systems (NIPS)*, 2003.
[11] S. M. LaValle, M. S. Branicky, and S. R. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *International Journal of Robotics Research*, no. 23, pp. 673–692, 2004.
[12] D. Hsu, J.-C. Latombe, and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," in *12th Int. Symp. on Robotics Research, San Francisco*, Oct. 2005.
[13] T. Dean and M. Boddy, "An analysis of time-dependent planning," in *Proc. of National Conference on Artificial Intelligence*, 1988.
[14] N. Currie and B. Peacock, "International space station robotic systems operations: A human factors perspective," in *Habitability and Human Factors Office (HHFO), NASA Johnson Space Center*, 2002.
[15] D. Hsu, G. Sanchez-Ante, and Z. Sun, "Hybrid PRM sampling with a cost-sensitive adaptive strategy," in *Proc. IEEE. Int. Conf. on Robotics and Automation*, 2005.
[16] B. Burns and O. Brock, "Sampling-based motion planning using predictive models," in *Proc. IEEE. Int. Conf. on Robotics and Automation*, 2005.