Robotica (2013) volume 31, pp. 1195–1208. © Cambridge University Press 2013
doi:10.1017/S0263574713000428

Randomized path planning with preferences in highly complex dynamic environments

Khaled Belghith†*, Froduald Kabanza† and Leo Hartman‡

†*University Of Sherbrooke, 2500, boul. de l'Université Sherbrooke, Québec J1K 2R1, Canada*
‡*Canadian Space Agency, John H. Chapman Space Centre, 6767 Route de l'Aéroport Saint-Hubert, Québec J3Y 8Y9, Canada*

(Accepted April 15, 2013. First published online: May 22, 2013)

SUMMARY

In this paper we consider the problem of planning paths for articulated bodies operating in workplaces containing obstacles and regions with preferences expressed as degrees of desirability. Degrees of desirability could specify danger zones and desire zones. A planned path should not collide with the obstacles and should maximize the degrees of desirability. Region desirability can also convey search-control strategies guiding the exploration of the search space. To handle desirability specifications, we introduce the notion of flexible probabilistic roadmap (flexible PRM) as an extension of the traditional PRM. Each edge in a flexible PRM is assigned a desirability degree. We show that flexible PRM planning can be achieved very efficiently with a simple sampling strategy of the configuration space defined as a trade-off between a traditional sampling oriented toward coverage of the configuration space and a heuristic optimization of the path desirability degree. For path planning problems in dynamic environments, where obstacles and region desirability can change in real time, we use dynamic and anytime search exploration strategies. The dynamic strategy allows the planner to replan efficiently by exploiting results from previous planning phases. The anytime strategy starts with a quickly computed path with a potentially low desirability degree which is then incrementally improved depending on the available planning time.

KEYWORDS: Robotics; Path planning; Replanning; Anytime planning; Dynamic planning.

1. Introduction

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.[3, 8, 12, 17, 24]

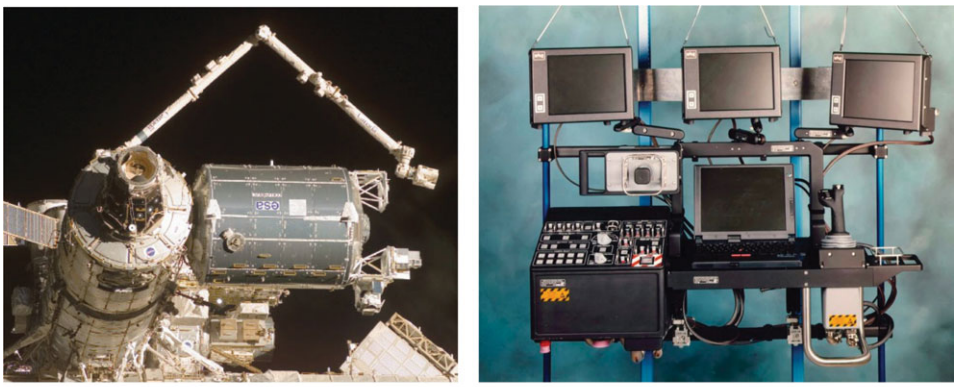
In many complex applications, however, in addition to obstacles that must be avoided, we may have dangerous areas that must be avoided as much as possible. That is, a path going through these areas is not highly desirable, but would be acceptable if no better path exists or can be computed efficiently. The danger concept is relevant, for example, in military applications. Some path planning techniques that deal with it have been proposed, including refs. [22, 26]. Conversely, it may be desirable for a path to stay close to certain areas as much as possible. Even when a path planning problem has no explicit notion of region desirability, introducing the notion provides a way to control the quality of a path generated by a randomized path planning method. Indeed, 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. In this paper we demonstrate that one can influence the sampling strategy to generate less awkward paths by specifying zones the path is preferred to go through. We also show that region desirability specifications can also help control the exploration of the sampled search space and make the path planner more efficient.

Our path planning approach builds flexible roadmaps by extending existing sampling techniques, including delayed collision checking, single query, bi-direction and adaptive sampling.[25] Desirable and undesirable workspace regions are soft constraints on the robot path, whereas obstacles are hard constraints. The soft constraints convey preferences for rating solution paths which must avoid obstacles. The more a path avoids undesired zones and goes through desired zones, the better it is.

The exploration of the sampled configuration space is done using dynamic and anytime space exploration methods.[14, 19, 20] In dynamic environments, a path planner can adapt a previously computed path to dynamic changes in obstacle configurations, goals or region desirability by computing a new path. Dynamic state space exploration strategies reuse the results obtained from previous searches to achieve better performance compared to re-searching from scratch. In addition, an anytime search strategy proceeds incrementally, starting with a path having a low desirability degree and then improving it incrementally. In this way at "any time," the planner has a plan with some degree of satisfaction that is improved as more planning time is spent.

Our test bed is a simulation of the Space Station Remote Manipulator System (SSRMS) deployed on the International Space Station (ISS). The SSRMS is a 17-meter long articulated robot manipulator, having a translation joint,

* Corresponding author. E-mail: khaled.belghith@usherbrooke.ca



(a) View from one of the monitors (b) Robot control station

Fig. 1. (Colour online) ISS path planning domain and robot control station.

seven rotational joints (each with a range of 270°) and two latching end-effectors, which can be moved to various fixtures, giving it the capability to "walk" from one grappling fixture to next on the exterior of the ISS.[6] Astronauts operate SSRMS using the robot control station located inside one of the ISS compartments (Fig. 1). A robot control station has an interface with three monitors, each connected to a camera placed at a strategic location of the ISS. There are many cameras covering different parts of the ISS structure and three of them are selected and mapped to the three monitors.

Most of the SSRMS tasks on the ISS involve moving the robot from one configuration to another in order, for example, to move a payload from the shuttle or inspect a region of the ISS exterior using a camera mounted on the end effector. A judicious choice of the camera on each of the three monitors along different segments of a robot path ensures that the operator is appropriately aware of the robot motion. Computed paths must go as much as possible through camera fields of view to enable a good appreciation of the robot motion. In other words, the camera fields of view convey preferences for regions through which the robot path should remain as much as possible while avoiding collisions with the ISS structure.

In the next section we give the background and discuss some of the related works. We then present our path planning approach to handling path preferences in the robot workspace. We follow with experiments in the ISS environment and in a car repair domain, showing the capability of the new planning approach to handle path preferences and search control specifications that are expressed by assigning desirability degrees to workspace regions.

2. Background and Related Work

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 or in the space of collision-free configurations, C_{free}, if the robot does not collide with any obstacle in the workspace in any of the configurations on the path. Computing a path is seen as making a query (to the path planner) with the input of the start and goal configurations. Two very commonly used approaches to path planning are the combinatorial and randomized approaches.

Combinatorial approaches, also called decomposition or exact approaches, proceed by searching through a geometric representation of C_{free}. Given a 2D or 3D model of obstacles in the workspace, a 2D or 3D model of the robot, the configuration space is decomposed into an occupancy grid of cells, also called a roadmap. A path from a start cell to a goal cell is then found by searching a sequence of moves between adjacent free cells, connecting the start configuration to the goal.[8,13,17,21] These moves correspond to possible edges in a graph with nodes corresponding to free cells in the grid. Graph-search algorithms such as A* search[9,23] or AD*[20] can be used to compute a path between the start and goal configurations.

Randomized approaches, also known as sampling-based approaches, proceed by sampling the space of the robot configurations. Given a 2D or 3D model of obstacles in the workspace and a 2D or 3D model of the robot, a randomized planner builds a graph of nodes corresponding to configurations in C_{free} by picking configurations randomly and checking that they are collision-free. It uses a fast collision detection checker (called a local planner) to check that an edge between two adjacent nodes is also collision-free; each time the local planner succeeds, the corresponding edge (i.e., local path or path segment) is inserted into the graph. The graph built that way is called a probabilistic roadmap (PRM)[12] or a rapidly exploring random tree (RRT)[17] and is a simplified representation of C_{free}. Here too graph-search algorithms such as A* search[23] or AD*[20] can be used to explore the graph to find a collision-free path linking the start to the goal configuration.

Therefore, combinatorial as well as randomized approaches have in common the discretization problem to build an intermediate graph structure (the roadmap) and search through it. The key difference lies in what the graph represents and how it is built. With combinatorial approaches the graph is meant to be an exact representation of C_{free} and its construction takes into account the geometry of the workspace and the robot. With sampling approaches,

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

the graph represents samples of C_{free}. It is not an exact representation of C_{free}. Given that the configuration space is randomly sampled, randomized approaches do not guarantee a full coverage of free space and they are not complete and do not guarantee optimality. In fact, they are probabilistically complete, meaning that the more the samples are made, the closer the probability of guaranteeing the absence or presence of a solution gets to 1.[25] Combinatorial approaches guarantee completeness and optimality by using a sufficiently small discritization step. In practice, this results in large search spaces, making the approaches generally intractable for high-dimensional configuration spaces.[10]

A heuristic method for grid decompositions is to plan using a coarse discretization space. If no solution is found or to improve the solution found so far, a new planning iteration is made with finer discretization pace. The process can be iterated as more planning time is invested or until a satisfactory solution is found. Another exploration strategy for the occupancy grid maybe to use random search.[18] 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.[10] Sampling-based methods generally offer better performance than exact methods for domains with high-dimensional spaces.[5,10,17]

2.1. Probabilistic roadmap approach

Our randomized implementation follows a PRM approach.[25] However, it can be easily adapted to an RRT approach given an RRT approach fundamentally corresponds to a single-query PRM with on-the-fly search of the sampled roadmap combined with on-the-fly collision detection.[18] Our implementation includes various configuration modes that allow the planner to run in a single or multiple-query mode, with on-the-fly search of the roadmap or not and with collision detection on-the-fly or delayed.

A PRM is an undirected graph $G = (N, V)$ with N being the nodes of the graph and V the arcs. The nodes are sampled configurations in C_{free}, also called milestones. The arcs represent links or segments v connecting two configurations. Algorithm 1 shows a basic PRM path planning algorithm. It starts by initializing the roadmap G with the start and goal configurations n_{start} and n_{goal}. Then a new node n is sampled randomly, with a probability measure π, in C_{free} and added to the roadmap. A set of nodes in G and in the neighborhood of n called V_n is selected. Using a collision checker (local planner), we look for a node n' in V_n such that the link (n, n') is free of collisions and then add it to G. The process is repeated until a path connecting n_{start} and n_{goal} is found.

The above algorithm follows a single-query on-the-fly collision detection approach. The samples of C_{free} corresponding to the nodes in the graph G are generated while searching G and detecting collision on the fly. On each query, the graph is reconstructed. It is conceivable to generate G, store it and then search it each time we have a query. In this case, a sufficiently large G needs to be generated to cover potential queries. This is known as a multiple-query approach because several queries can be made on the same roadmap. A delayed collision-checking approach would avoid checking collisions (Step 6) until a whole path has been found. If a segment on the path turns out to be colliding, the algorithm would backtrack to search for a new path. A delayed collision-checking method can outperform a non-delayed one on some planning domains.[3,25]

Algorithm 1: Basic PRM Algorithm

01. Initialize the roadmap G with two nodes, n_{start} and n_{goal}
02. Repeat
03. Sample a configuration n from C_{free} with probability measure π
04. if ($n \in C_{\text{free}}$) Then add n as a new node of G
05. for some nodes n' in a neighborhood V_n of n in G such that $n' \neq n$
06. if collisionFree(n,n') then add $v = (n, n')$ as a new edge of G
07. until n_{start} and n_{goal} are in the same connected component of G
08. if $n_{\text{start}}$and n_{goal} are in the same connected component of G Then
09. return a path between them
10. else
11. return No Path

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.[10] A naive sampling measure will likely lose efficiency when the free space C_{free} contains narrow passages. A narrow passage is a small region in C_{free} where the sampling probability becomes very low. Some approaches exploit the geometry of obstacles in the workspace to adapt the sampling measure accordingly.[2,15,16] Other methods use machine learning techniques to adapt the sampling strategy dynamically during the construction of the probabilistic roadmap.[4,11,16]

2.2. Path planning with preferences

In addition to collision avoidance, the concept of dangerous areas that have to be avoided as much as possible has been addressed in some path planning approaches.[22,26] Herein we generalize the concept to preferences among regions in the C_{free} space. Different regions can be assigned different degrees of desirability, meaning that we would like the path planner to compute a path which not only avoids obstacles but also maximizes the degree of desirability for the path. Since path quality criterion may also depend on other metrics such as the distance along the path, we define the overall path quality as a trade-off between region desirability and distance. The trade-off is conveyed by a parameter weighing the contribution of each of these criteria to the global path-quality criterion. As a means to convey preferences among collision-free paths, region desirability provides a way to specify search control information for a path planner. It can be used to determine how the search process chooses the next node to expand.

2.3. Anytime path planning

In real-time applications involving the computation of an optimal solution, it is often desirable to have an incremental algorithm that computes its solution as a sequence of intermediate useful but suboptimal solutions, converging

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

toward an optimal solution. Dean and Bodddy[7] called these anytime algorithms. Such an algorithm guarantees a useful approximate solution anytime, which gets improved incrementally if more planning time is allowed. With a PRM planner, an anytime capability can be integrated into the search algorithm exploring the sampled roadmap. In particular, using the A* heuristic graph-search algorithm, one can implement a twofold anytime capability.

If given a transition function covering the entire search space and an admissible heuristic, A* guarantees finding an optimal solution. In our case, the search space is the sampled roadmap and a heuristic function $h(n)$ is a function taking a configuration n as input and returning the estimated distance from a configuration n to the goal configuration. It is admissible if it never overestimates the actual distance $h^*(n)$. We use the Euclidean distance as admissible heuristic. Obviously, the larger the search space, the more time it may take to find an optimal solution, even though A* does not have to exhaust the search space to guarantee optimality. To mitigate the combinatorial size of the state space, one can run A* on a smaller portion of the search space (producing an approximate path), then expand the search space and compute a new solution path for it and so on. This gives a sequence of solutions, converging to the optimal when the entire search space is covered. Given a deadline for computing a solution, the search space will be iteratively expanded accordingly; a solution for each chunk of expansion is then computed. This implements an anytime capability through state expansion. The search process can be stopped anytime 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 will be.

Another interesting property of A* is that if given an inadmissible heuristic $h(n) = h^*(n) + \epsilon$, then the cost of the path computed by A^* minus the cost of the optimal distance is less than or equal to ϵ. In other words, ϵ is an upper bound on the error for the cost of the solution compared to the optimal solution. Moreover, A* tends to return a solution, possibly suboptimal, faster with inadmissible heuristics than with admissible heuristics. Based on these two observations and given an admissible heuristic h (e.g., the Euclidean distance), another way to implement an anytime A* search would be to compute a path using $h(n) + \epsilon_1$, then another solution using $h(n) + \epsilon_2$ and so on. In other words, a sequence of solutions using a decreasing error bound ϵ_i on the admissible heuristic is computed, with $\epsilon_{i+1} < \epsilon_i$. Given a deadline for computing a solution path, the inflating factor will be decreased iteratively, computing a solution for each decrease. This is the anytime capability through heuristic improvement.

Both previous methods for implementing anytime capabilities with A* are complementary and can be combined as is the case in the Anytime Repairing A* (ARA*) algorithm.[21] We use a similar approach to explore a randomized flexible roadmap.

2.4. Dynamic path planning

In the ISS environment, most of the structure is fixed, only the robots can move. However, region desirability degrees can change as well as the goal. Regions of desirability depend on the task and the involved camera views. Depending on the orbit of the ISS, a camera may have its view toward the sun, making it undesirable. From the roadmap perspective, this means that the cost of a segment between two nodes can change dynamically. 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. Replanning is necessary in such cases.

Dynamic path planners adapt dynamically to change happening around the robot by repairing incrementally their representation of the environment. Different approaches exist that are extensions of the A* algorithm, including D* Lite,[13] Anytime Dynamic A* (AD*)[20] and Generalized Adaptive A* (GAA*).[27] These algorithms extend A* search to solve dynamic search problems faster by updating heuristics on nodes using knowledge acquired from previous searches.

3. Flexible Anytime Dynamic Probabilistic Roadmap Path Planner

Combining region preferences, anytime search and dynamic replanning, we obtain a flexible anytime dynamic probabilistic roadmap planner (FADPRM). The general idea is to keep track of milestones in an optimal solution to the goal. When changes are noted, edge 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 now only concerned with the roadmap to the current goal that the robot is trying to reach in a dynamic environment.

FADPRM uses GAA* to explore the roadmap. The cost of an edge between two configurations depends on the actual distance between the configurations and the desirability degrees of the configurations along that edge. In a preliminary version of FADPRM,[1] we have used AD* instead of GAA*. We now use GAA* instead of AD* because they have comparable performances, yet GAA* has a simpler description. Note that the contribution of FADPRM does not just amount to using GAA* to explore a sampled roadmap. The integration of preferences and their use to control both the path quality as well as the search-process for computing such a path are the key contributions.

3.1. Algorithm sketch

FADPRM works with C_{free} segmented into zones, each zone being assigned a degree of desirability (dd), that is, a real number in the interval [0 1]. The closer the dd is to 1, the more desirable is the zone. 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 as follows: a start 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, we can perform the following:

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

1. To find a path connecting the input and goal configuration, we search backward from the goal toward the start (current) robot configuration. Backward instead of forward search is done because the robot moves; 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 being updated anymore) and a list $CLOSED$ contains non-frontier nodes (i.e., nodes already expanded).
3. Search consists of repeatedly picking a node from $OPEN$, generating its predecessors and putting the new ones and the ones not yet updated 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 to be improved.
6. Changes in the environment (moving obstacles and zones or changes in dd for zones) cause updating of the roadmap and replanning.

With β equals to 0, the selection of a node to expand is totally blind to zone degrees of desirability and to edge costs (Euclidian distance). Assuming $OPEN$ is the entire roadmap, this case corresponds to a normal PRM and the algorithm probabilistically converges toward an optimal solution as is the case for a normal PRM.[25] With $\beta = 1$, the selection of a node is the 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 start 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 normal A* except that

- We do backward search, hence $g(n)$ and $h(n)$ are reversed.
- 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:

$$g(n) = pathdd(n_{\text{goal}}, n)/(1 + \gamma pathCost(n_{\text{goal}}, n))$$

  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}:

$$h(n) = pathdd(n, n_{\text{start}})/(1 + \gamma pathCost(n, n_{\text{start}})).$$

The factor γ determines the influence of dd on $g(n)$ and $h(n)$. 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 the 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 is guaranteed to converge to the optimal solution. When dds are involved and since zones can have arbitrary configurations, it is difficult to define admissible heuristics. The algorithm guarantees improvement of the solution, but it is 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 or when dynamic changes are observed in the environment.

3.2. Algorithm details

The detailed structure of the FADPRM path planner is presented in Algorithm 2. Since FADPRM proceeds backwards, it updates h-values with respect to the start configuration of all expanded nodes n after every search as follows:

$$h(n) = g(n_{\text{start}}) - g(n).$$

Following GAA*, FADPRM does not initialize all g- and h-values upfront. Instead, it uses the variables *counter*, *search(n)* and *pathcost(x)* to decide when to initialize and update them by calling *UpdateState()*:

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

Algorithm 2: FADPRM Algorithm

```
01. KEY(n)
02.   f(n) = [g(n) + h(n)]/2;
03.   return [(1 − β)/density(n) + β.f(n); h(n)]

04. UPDATESTATE(n)
05.   if search(n) ≠ 0
06.     if (g(n) + h(n) < pathcost(search(n))
07.       h(n) = pathcost(search(n)) − g(n);
08.       g(n) = 0;
09.   else if (search(n) = 0)
10.     g(n) = 0;
11.   search(n) = counter

12. COMPUTEORIMPROVEPATH()
13.   while (NoPathfound)
14.     remove n with max key from OPEN;
15.     if (Connect(n, n_start))
16.       return β-suboptimal path;
17.     else
18.       ExpandNode(n);;
19.       For all n′ ∈ Pred(n)
20.         UPDATESTATE(n′)
21.         g(n′) = g(n) + c(n, n′)
22.         insert n′ into OPEN;
23.       insert n into CLOSED;

24. MAIN()
25.     counter = 1;
26.     g(n_start) = g(n_goal) = 0;
27.       search(n_start) = search(n_goal) = 0;;
28.     β = β0;
29.     OPEN = CLOSED = ∅
30.     UPDATESTATE(n_start)
31.     UPDATESTATE(n_goal)
32.     insert n_goal into OPEN with key(n_goal);
33.     while (Not collision-free Path)
34.         Rearrange Tree;
35.         ComputeorImprovePath();
36.         counter = counter + 1;
37.         if OPEN = ∅
38.             pathcost(search(n)) = 0
39.         else
40.             pathcost(search(n)) = g(n_start)
41.     publish current β0−suboptimal solution;
42.     while (n_start not in neighborhood of n_goal)
43.         if n_start changed
44.             if addtoTree(n_start)
45.                 publish current solution;
46.         if changes in edge costs are detected
47.             for all changed edges (u, v)
48.                 Update the edge cost c(u, v);
49.                 UpdateState(u);
50.             Update the priorities for all
51.               n ∈ OPEN according to Key(n);
52.             CONSISTENCYPROCEDURE()
53.             decrease β or replan from scratch;
54.         if β < 1
55.             increase β;
56.         CLOSED = ∅;
57.         while (Not collision-free Path)
58.             Rearrange Tree;
59.             ComputeorImprovePath();
60.             counter = counter + 1;
61.             if OPEN = ∅
62.                 pathcost(search(n)) = 0
63.             else
64.                 pathcost(search(n)) = g(n_start)
65.         publish current β−suboptimal solution;
66.         if β = 1
67.             wait for changes in edges cost;
```

- The value of *counter* is x in the xth execution of $ComputeOrImprovePath$, that is, the xth call for GAA* on the roadmap.
- $search(n)$ stores the number of the last search that generated node n. FADPRM initializes these values to 0 for new nodes in the roadmap.
- $pathcost(x)$ stores the cost for the best path found on the roadmap by the xth search. More precisely, the formula for $pathcost(x)$ is

$$pathcost(x) = g(n_{\text{start}}) = pathdd(n_{\text{goal}}, n_{\text{start}})/(1 + \gamma pathCost(n_{\text{goal}}, n_{\text{start}})).$$

Nodes in $OPEN$ are expanded in decreasing priority to update their g-values and their predecessors' g- and h-values. 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:

$$key(n) = [(1 - \beta)/density(n) + \beta f(n),\ h(n)],$$

with $f(n) = [(g(n) + h(n)]/2$ and $key(n) \leq key(n')$ if $k_1(n) \leq k_1(n')$ or $(k_1(n) = k_1(n')$ and $k_2(n) \leq k_2(n'))$. During the update on nodes, FADPRM initializes the g-value of nodes not yet generated by an already performed search, nodes with $search(n) = 0$, to zero.

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 (for Single-query Bidirectional PRM planner with Lazy collision detection).[25] If it succeeds, a path is then returned (line 16 in Algorithm 2). The expansion on a node n with maximum key from the $OPEN$ (line 18 in Algorithm 2) consists of sampling a new collision-free node in the neighborhood of n[25] and then the sampled node is added in the set $Pred(n)$. After increasing the connectivity of the roadmap by adding a new node, FADPRM executes an update of the heuristics of all nodes in $Pred(n)$ in order to make them more informed and then allow for later more focused searches.

FADPRM updates the h-values of node n (line 7) if the following different conditions are satisfied:

- The node has not yet been generated by the current search $(search(n) \neq counter)$.
- The node was generated by a previous search $(search(n) \neq 0)$.
- The node was expanded by the search that generated it last $(g(n) + h(n) < pathcost(counter))$.

FADPRM sets $h(n)$ (in line 7) to the difference between $g(n_{\text{start}})$ that is the cost of the path from n_{start} to n_{goal} during the last search that expanded n and $g(n)$ that remained the same since the same search. Dynamic changes in the environment affect (increase or decrease) edge costs. Such changes are handled by a consistency procedure, adapted from GAA* and described below. This procedure invoked at line 52 of the main algorithm whenever a cost decrease is observed. When invoked, it updates the h-values with respect to the start node.

The Main procedure in FADPRM first sets the inflation factor β to a low value β_0 so that a suboptimal plan can be generated quickly (line 41). Then if no changes in edge costs are detected, β is increased to improve the quality of its solution (lines 54–55). This will continue until the maximum of optimality is reached with $\beta = 1$ (lines 66–67).

FADPRM also follows the concept of lazy collision checking. 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 34, 58). FADPRM also

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

Algorithm 3: Consistency Procedure

```
01. ConsistencyProcedure()
02.   update the increased and decreased action costs (if any);
03.   OPEN = ∅;
04.   for all edges (n, n')
05.   with (n ≠ n_start) and edge cost c(s, s') decreased
06.     UpdateState(n);
07.     UpdateState(n'));
08.     if (h(n) > c(n, n') + h(n'))
09.       (h(n) = c(n, n') + h(n'))
10.       if n ∈ OPEN
11.         delete n from OPEN ;
12.       insert n in OPEN with key-value KEY(n);
13.     while (OPEN ≠ ∅);
14.       delete n' with smallest key-value from OPEN ;
15.       for all states n ≠ n_start with succ(n) = n'
16.         UpdateState(n);
17.         if (h(n) > c(n, n') + h(n')))
18.             (h(n) = c(n, n') + h(succ(n')))
19.             if n ∈ OPEN
20.                 delete n from OPEN ;
21.             insert n in OPEN with key-value KEY(n);
```

handles the case of a floating starting configuration (lines 43–44).

4. Experimental Results

In the first set of experiments we illustrate and validate the replanning and anytime capabilities of FADPRM in dealing with highly complex environments with preferences. In the second set of experiments we illustrate the search control capability of FADPRM and show how region desirability specifications can help control the exploration of the sampled search space and make path planning more efficient.

Experiments are made in two different environments: a simulation of the SSRMS on the ISS and a Puma robot operating on a car. The SSRMS is the most complex environment: 7 DOF, 75 obstacles modeled with 85,000 triangles. The Puma robot has 6 DOF and its environment is modeled by approximately 7000 triangles.

All experiments were run on a 1.86 GHZ Core 2 Processor with 2 GB 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 pathdd as a measure for path quality. For all experiments, PRM refers to an implementation of SBL.[25]

4.1. Fast replanning capability

In the SSRMS application, 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. 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 that 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 on paper, we just picked conical zones approximating camera view regions and polygonal zones at arbitrary locations. Figure 2 illustrates a trajectory of the SSRMS carrying a load and going through three camera fields of view (purple, green and blue cones) and avoiding a non-desired zone with very limited lighting conditions (red box).

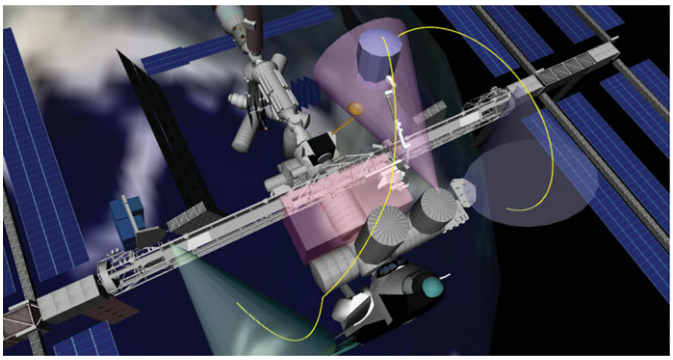


Fig. 2. (Colour online) SSRMS going through three different camera fields of view (purple, green and blue cones) and avoiding a non-desired zone (red box).

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 the FADPRM planner to calculate a path with a high dd to the goal. If no such path exists, we notify the learner that he is moving the SSRMS to a dead end. Although paths are computed to confirm the learner is on the right track, they are not displayed to him. Hence, while the learner is making suitable progress toward the goal, they are solving the problem on their own.

Figure 3 shows the time taken for replanning while the human operator is moving the robot toward a goal configuration in the scenario of Fig. 2. We conducted the experiment three times with the operator doing exactly the same manipulations to reach the goal from the start configuration and each time using FADPRM (with $\beta = 0$) and the normal PRM. Except for the first few iterations, FADPRM take less replanning time than PRM. For FADPRM and in the first few iterations, the overhead incurred by the GAA*-based exploration dominates the planning time. In later iterations, it is outweighed by the savings gained by replanning from the previous roadmap.

In Fig. 4 we compare the time needed for FADPRM and PRM to find a solution for 15 arbitrary queries in the ISS environment. Since the time (and path quality) for finding 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 is used in a mode that does not store 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 0.4. With $\beta = 0$, FADPRM behaves exactly like the normal PRM. With $\beta = 0.4$, planning takes more time for both planners. This validates our previous analysis about FADPRM: with $\beta =$

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

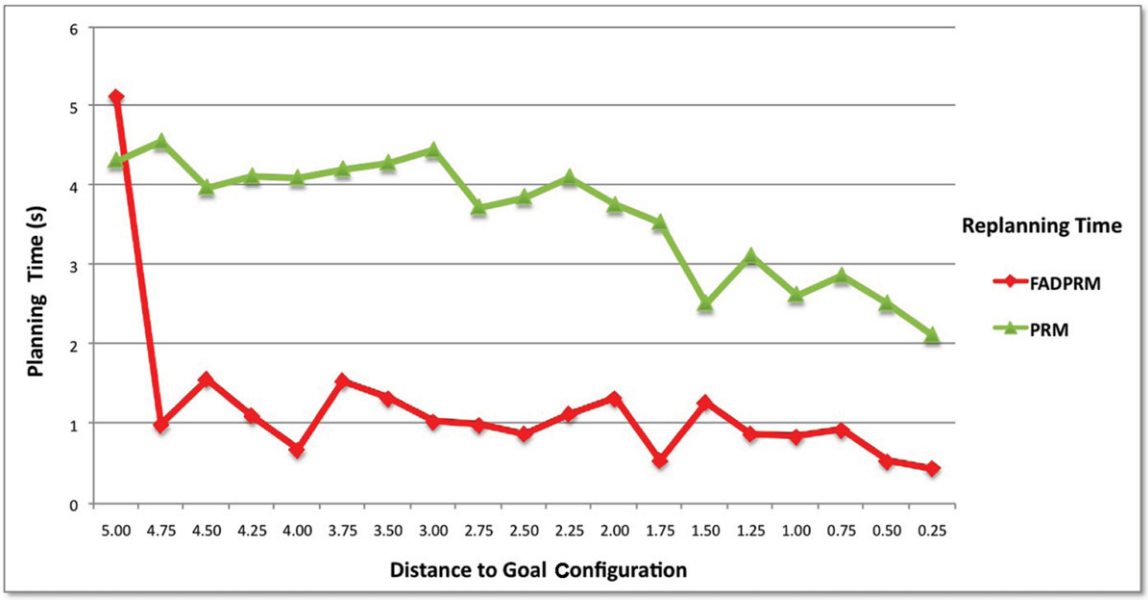


Fig. 3. (Colour online) FADPRM versus PRM in replanning.

0, an FADPRM planner behaves in way very similar to a normal PRM, but as soon as we start seeking optimality (in our case with $\beta = 0.4$), the time for planning will increase proportionally.

On an other hand, Fig. 5 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.

4.2. *Search control capability*

Specifying zone degrees of desirability provides a means to accelerate the computation of a path. Since FADPRM explores configurations in the order of specified preferences, it is possible to control the search process via the specification of regions with suitable degrees of desirability, so it reaches a solution very quickly on average.

To illustrate the search control capability in the ISS environment, we establish a planning scenario where the SSRMS has to carry a load to a space shuttle docked to the ISS. Path planning is further made more complex in this scenario with the final configuration placed in a narrow passage: near the shuttle and surrounded by a number of modules as shown on Fig. 6. Normal PRMs start losing efficiency in such areas since the sampling probability becomes very low. In the first experiment (Fig. 6(a)), we plan for a path with a wide desired zone on the left of the shuttle. In the second experiment (Fig. 6(b)), we specify a non-desired zone on the right of the shuttle and two wide desired zones on the front and on the left of the shuttle. By adding zones with appropriate degrees of desirability, we wanted to influence the sampling of the free workspace to yield better paths. Here planning is done without any call to a post-processing smoothing step as is usually done

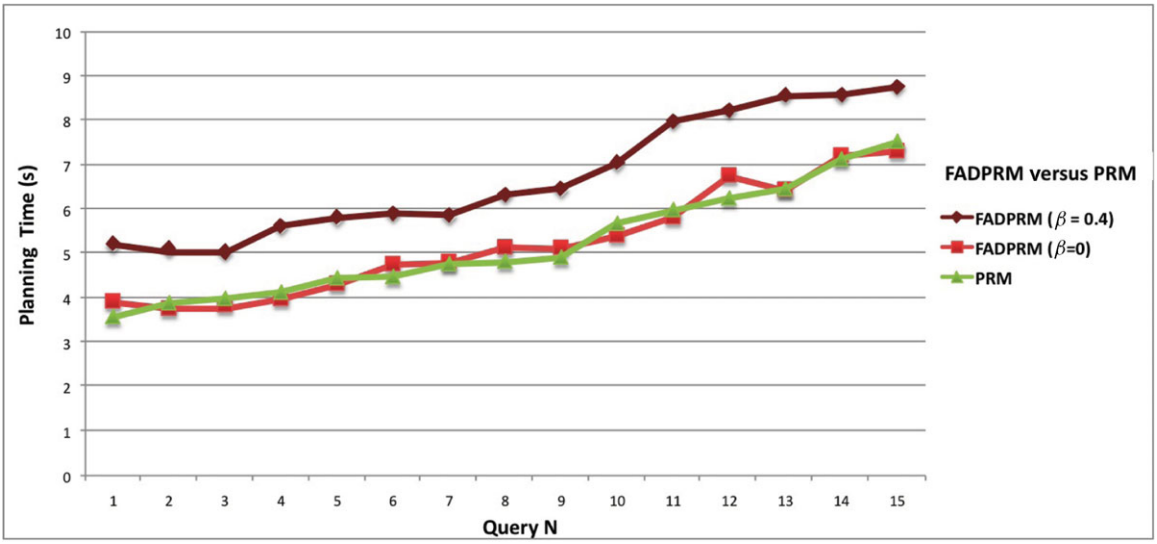


Fig. 4. (Colour online) FADPRM versus PRM in planning time.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

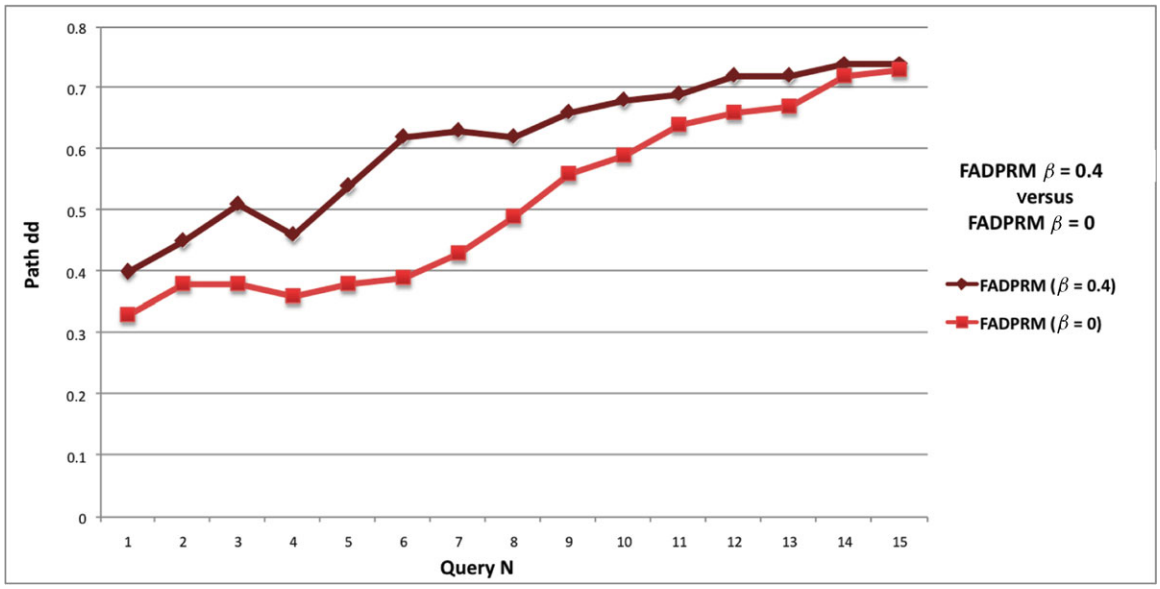


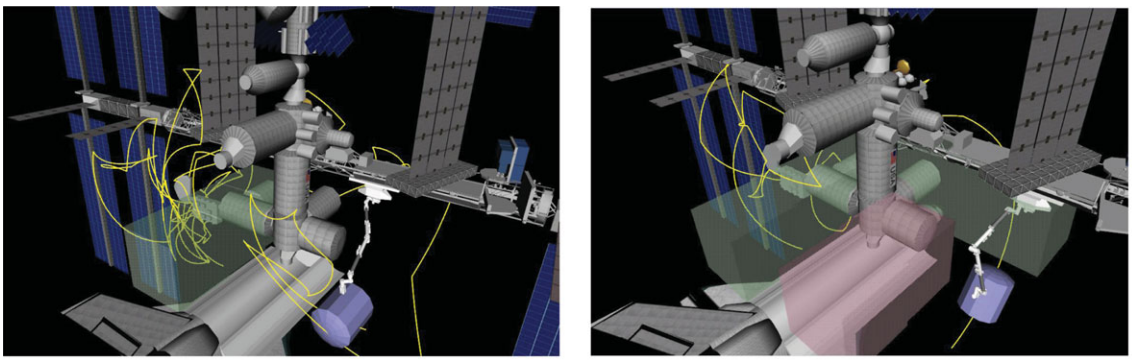
Fig. 5. (Colour online) FADPRM versus PRM in path quality (*Pathdd*).

with normal PRM planners.[25] This explains why we have awkward trajectories in Figures 6(a) and (b).

In Fig. 7 we compare the time needed for FADPRM and PRM to find a solution for 15 different queries within both scenarios of Figs. 6(a) and (b). For all queries, the goal configuration with the carried module inside the shuttle remains the same. Start configurations are picked randomly at different locations around the shuttle. For each query we ran each of the planners 10 times and reported the average planning time. 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 the results with $\beta = 0$, 0.4 and 0.7. In these experiments, the bias factor γ that determines the influence of the dd on the cost of edges within the roadmap is equal to 0.5.

With $\beta = 0$, FADPRM behaves exactly like a normal PRM in both scenarios yielding very complex awkward paths requiring an approximately equivalent time to compute. As soon as we start seeking optimality with $\beta = 0.4$ and 0.7, the time for planning in the two scenarios increases. In both scenarios, FADPRM with $\beta = 0.7$ yields better paths than FADPRM with $\beta = 0.4$ but takes more computing time. Figure 8 confirms this and shows better quality (i.e., better dds) for paths computed with FADPRM ($\beta = 0.7$) compared with paths found by FADPRM ($\beta = 0.4$).

If we compare the time taken for planning with the same version of FADPRM ($\beta = 0.4$ or 0.7) within the two different scenarios of Fig. 6, we note that more planning time is needed in the scenario of Fig. 6(a). With larger values of β, the sampling with FADPRM is pushed into areas with high values of dd. If the workspace is not covered with enough desired zones, the planner may remain stuck sampling within one desired zone of the workspace. This explains the problem of local minima that we see in Fig. 7 with FADPRM in scenario of Fig. 6(a). The more β is increased, the more the sampling is pushed within high dd zones. This explains why the local minima problem is more frequent with FADPRM with $\beta = 0.7$. By increasing the coverage of the workspace with more desired and non-desired zones such as in Fig. 6(b), we significantly improve the planning time needed for finding an optimal solution and considerably reduce the probability of having the local minima problem (Fig. 7).



(a) Scenario 1 with one desired zone

(b) Scenario 2 with two desired zones and one danger zone

Fig. 6. (Colour online) Not smoothed paths with FADPRM ($\beta = 0.7$) in the ISS environment.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

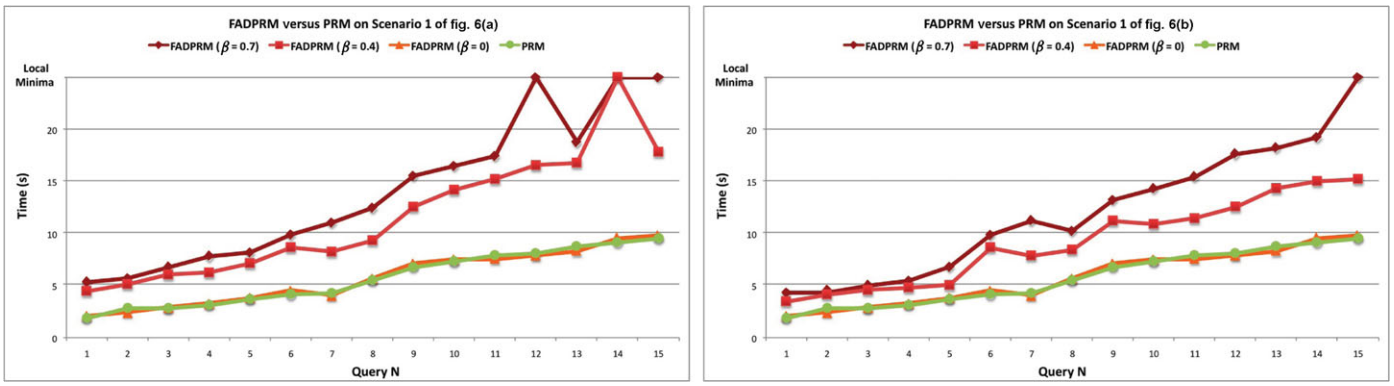


Fig. 7. (Colour online) FADPRM ($\beta = 0.4$ and 0.7) versus PRM in planning time.

The factor γ determines the influence of the dd on $g(n)$ and $h(n)$. 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 the least cost to the goal are privileged. In the following experiment, we test the influence of different values of γ on the planning time with FADPRM ($\beta = 0.7$) in the scenario of Fig. 6(b) with a well-constrained workspace.

Seeking optimality in the robot path takes more time and can lead to local minima problems. With FADPRM, the local minima problem can have the following two reasons:

1. *distance*-minima problem: The planner remains stuck sampling without success in a narrow passage around a configuration too close to the *goal*-configuration.

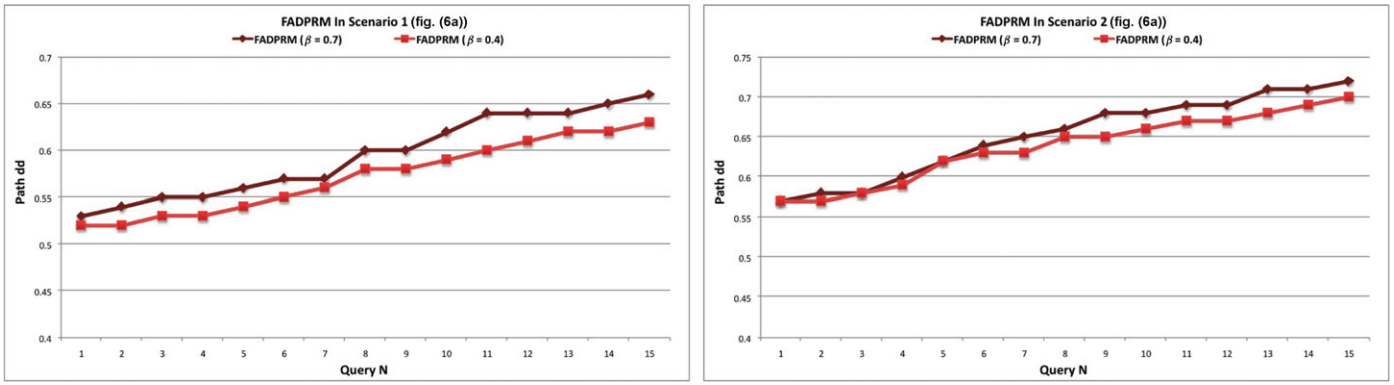


Fig. 8. (Colour online) Path quality with FADPRM ($\beta = 0.4$ and 0.7).

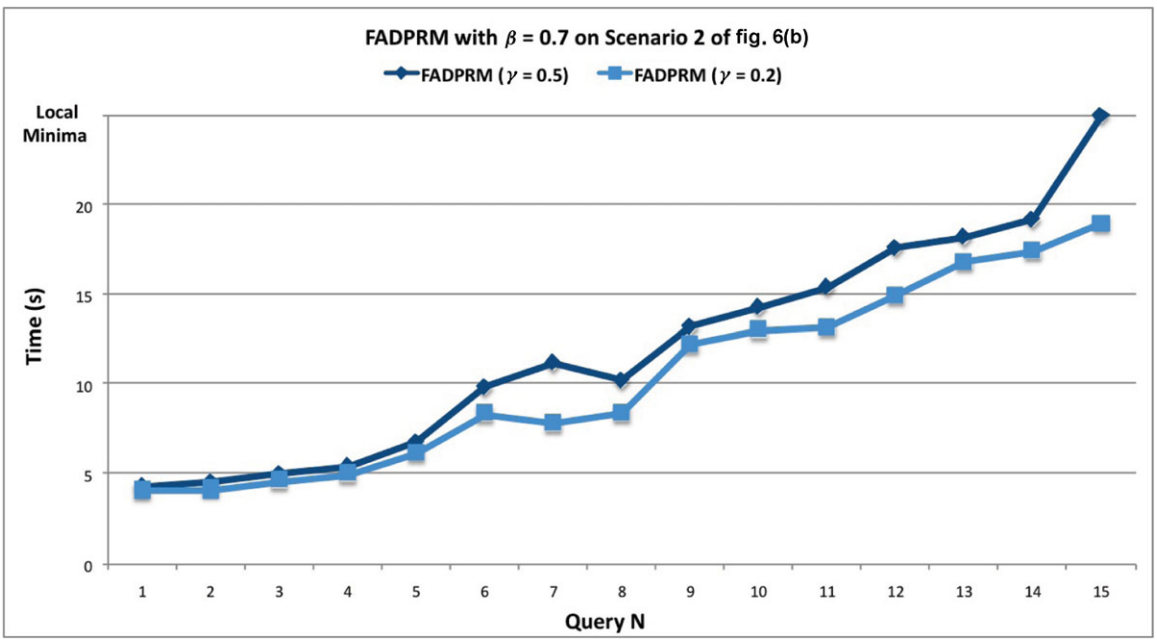
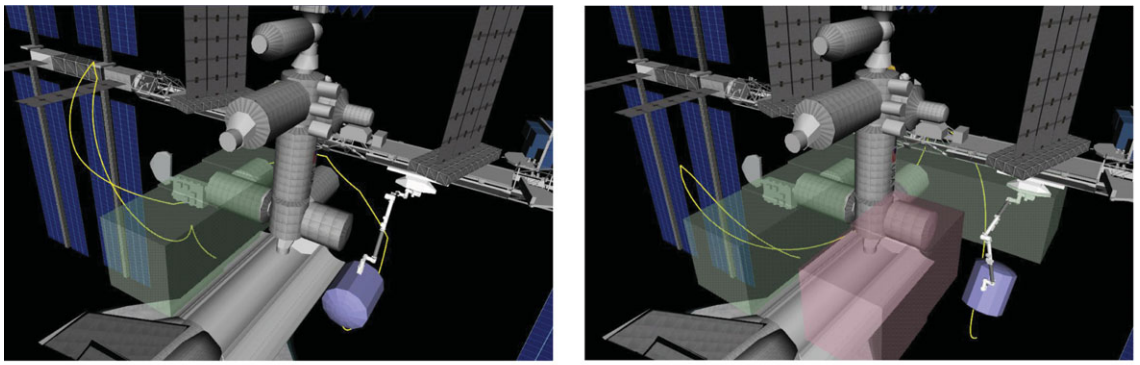


Fig. 9. (Colour online) Planning time with FADPRM ($\beta = 0.7$) with $\gamma = 0.5$ and 0.2.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press



(a) With one desired zone (b) With two desired zones and one danger zone

Fig. 10. (Colour online) Smoothed paths with FADPRM in the ISS environment ($\beta = 0.7$).

2. *dd*-minima problem: The planner remains stuck sampling without success within a tiny desirable zone of the workspace like in the scenario of Fig. 6(a).

By increasing the influence of *dd* on the cost of nodes, we reduce the probability of having the *distance*-minima problem. By increasing the coverage of the workspace with desired and non-desired zones, we reduce the probability of having the *dd*-minima problem. This explains why in Fig. 9, FADPRM with $\gamma = 0.2$ takes less time for planning compared with FADPRM with $\gamma = 0.5$. More importantly, no occurrences of the local minima problem are observed with FADPRM ($\gamma = 0.2$).

4.3. Path-quality control capability

With randomized path planners, 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. With FADPRM, it is possible to influence the sampling strategy and generate less awkward paths by specifying zones we prefer them to go through or zones we want them to avoid.

In Fig. 6, we note an improvement in the smoothness of the path generated with FADPRM on the scenario of Fig. 6(b) compared with the path in Fig. 6(a). In Fig. 11, we measure the time needed for smoothing the paths (as shown on Fig. 10) found on the 15 queries of Fig. 7.

Compared with normal PRM, FADPRM (with $\beta = 0.4$ or 0.7) always produces paths that need less post-processing smoothing time. In both scenarios, FADPRM with $\beta = 0.7$ needs less time for smoothing than FADPRM with $\beta = 0.4$. Here we note that the more we seek optimality in the robot path, the less awkward solution paths are, which explains why they require less time to smooth.

We also note that for both versions of FADPRM (with $\beta = 0.4$ or 0.7), more smoothing time is required in the first scenario in Fig. 6(a). The more the path is constrained with desired and undesired zones within the workspace, the more quality and efficiency we guarantee in the solution path. That is, the path is smoother and requires less time to smooth.

Figure 12 confirms the same results found with not smoothed solution paths in Fig. 8. Increasing β into FADPRM yields smoothed paths with better quality in terms of degrees of desirability. Also, the more we cover the workspace with desirability zones, the more this path quality is enhanced. Zones with degrees of desirability provide a means to specify a sampling strategy that controls the search process to generate better paths (better *dd*s and better smoothness) by simply annotating the 3D workspace with the regions' degrees of desirability.

4.4. Generality of the results

By running the previous experiments on several randomly chosen samples and reporting the average results, we somewhat try to verify that FADPRM features are not dependent on one specific scenario. Obviously, the ISS domain has a specific structure. Runs on different domains are necessary to increase confidence in the generality of the observed performances of FADPRM. In this regard, we did

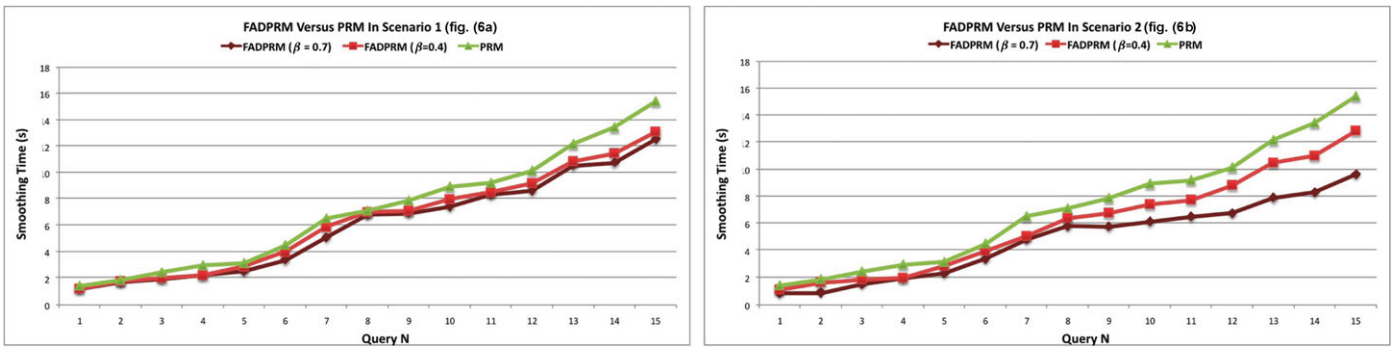


Fig. 11. (Colour online) FADPRM versus PRM in smoothing time.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

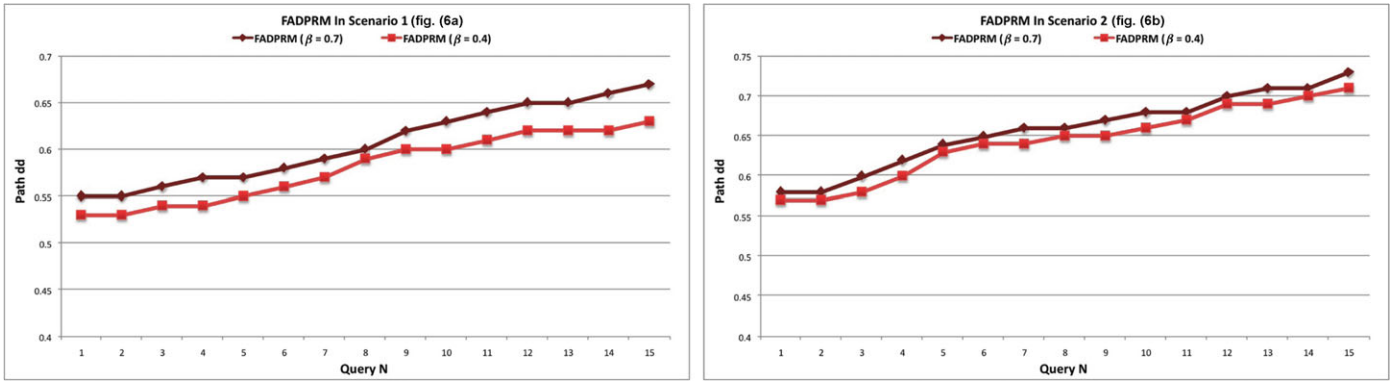


Fig. 12. (Colour online) FADPRM versus PRM in path quality of smoothed paths.

numerous experiments on a simulated Puma robot operating on a car. The obtained results confirm those in the SSRMS domain.

For instance, to evaluate the search-control capability of FADPRM, in one of the experiments we specified a small desirable zone (see Fig. 13(a)) and in another we specified a wider desirable zone (in front of the car) and a wide undesirable zone (in the back) (see Fig. 13(b)). In both sets of experiments we wanted to influence the sampling of the free workspace to yield paths that move the robot in front of the car (from the left side, to the front, then to the right side).

By specifying a desirable zone on the right front of the car as shown in Fig. 13(a) and running FADPRM planner many times on the same query (input/goal configuration), they yielded better paths, on average, than PRM. On the other hand, by enlarging the size and coverage of the desirable zone and adding an undesirable zone (right, on the back of the car), as shown in Fig. 13(b), we noted that the quality of paths increased by 50% over 100 trials. 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 makes the probability of sampling a configuration along the desirable region higher than in the first setup.

Figure 14 shows the anytime capability of FADPRM (with $\beta = 0.4$) on these two experiments. We note continuous improvement of the path quality ($pathdd$) for the two settings. The more the time it is given, the better the path provided by FADPRM. The results here confirm the observations noted in the first experiment with the SSRMS. Handling zones with degrees of desirability provides FADPRM with a powerful sampling strategy that helps generate better quality paths. And, the better the coverage of the workspace with preference zones, the more optimal (in terms of degrees of desirability) the solution path to which FADPRM converges.

5. Conclusion

In many real-world path planning applications, in addition to obstacles that must be avoided, we may have areas that the body is preferred to avoid (or, conversely, to go through) as much as possible. This is the case in the ISS domain where preferences are tied to camera views which change dynamically, in part because of varying environmental conditions throughout the orbit.

In this paper we presented a new randomized approach for robot path planning which extends the PRM framework to handle a workspace containing regions with degrees of desirability. Our approach integrates dynamic and anytime search exploration strategies to deal with problems in dynamic environments where obstacles and region desirability can change in real time. The dynamic strategy allows the planner to replan efficiently by exploiting results from previous planning iterations. The anytime strategy starts with a quickly computed path with a potentially low degree

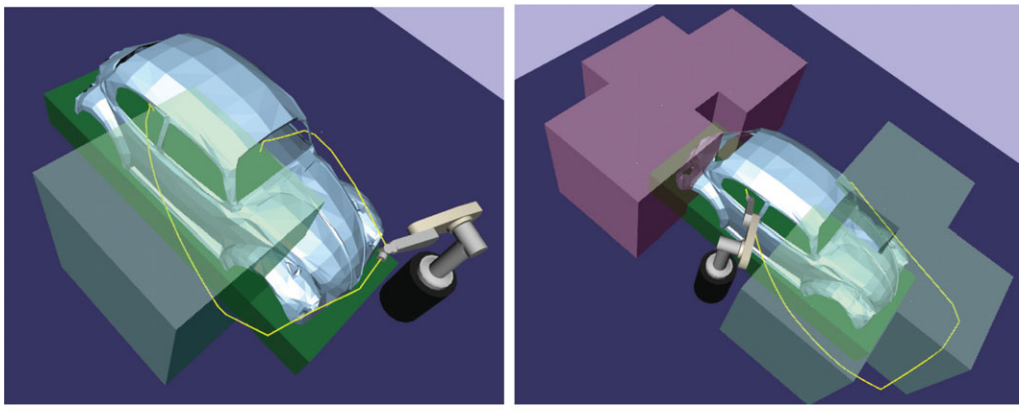


Fig. 13. (Colour online) PUMA robot around a car.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

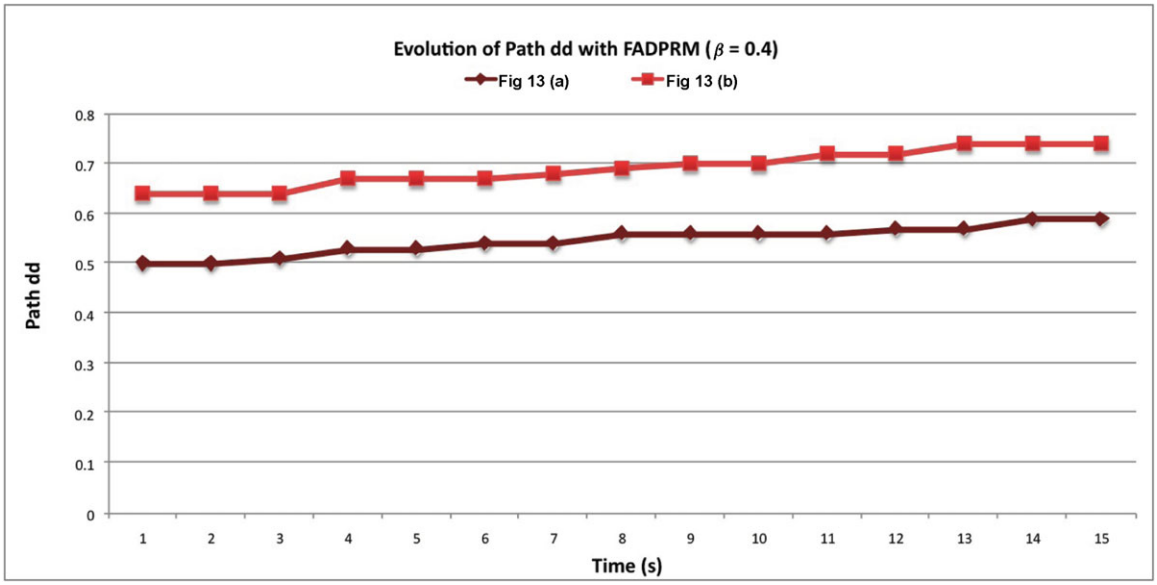


Fig. 14. (Colour online) Path quality evolution with FADPRM.

of desirability which is then incrementally improved if more planning time is allowed.

The experiments validated the different features of FADPRM on two particular path planning domains. Although we obtain good path quality and better replanning time than normal PRM approaches, there remains potential for improvement on both dimensions. Paths still need to be smoothed in post-processing, and for real-time applications we still want a planning algorithm that is as fast as possible. We will therefore continue to explore ways to improve our approach and look for alternatives.

FADPRM is a component in a large simulation prototype for training astronauts on the SSRMS. It is invoked by an intelligent tutoring system (ITS) to monitor robot operations carried out by a student and provide feedback on how to control the arm. For instance, given the task of moving the SSRMS from one configuration to another, the ITS can try computing a path from the current configuration to the goal and advise the student when the current configuration seems to be a dead-end. The student can then backtrack to a previous point from which better paths to the goal are available.

Acknowledgment

The work presented herein was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

References

1. K. Belghith, F. Kabanza, L. Hartman and R. Nkambou, "Anytime Dynamic Path-Planning with Flexible Probabilistic Roadmaps," **In**: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (Orlando, Florida, USA, 2006) pp. 2372–2377.
2. J. P. V. D. Berg and M. H. Overmars, "Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners," *Int. J. Robot. Res.* **24**(12), 1055–1071 (2005).
3. R. Bohlin and L. Kavraki, "Path Planning Using Lazy PRM," **In**: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (San Francisco, CA, USA, 2000) pp. 521–528.
4. B. Burns and O. Brock, "Sampling-Based Motion Planning Using Predictive Models," **In**: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (Barcelona, Spain, 2005) pp. 3120–3125.
5. H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations* (MIT Press, Cambridge, MA, 2005).
6. N. Currie and B. Peacock, "International Space Station Robotic Systems Operations: A Human Factors Perspective," **In**: *Proceedings of Human Factors and Ergonomics Society Annual Meeting, Aerospace Systems*, Baltimore, Maryland (Human Factors and Ergonomics Society, Santa Monica, CA, USA, 2002) pp. 26–30.
7. T. Dean and M. Boddy, "An Analysis of Time-Dependent Planning," *Proceedings of the 14th National Conference on Artificial Intelligence (AAAI)* (St. Paul, Minnesota, USA, 1988).
8. D. Ferguson, M. Likhachev and A. Stentz, "A Guide to Heuristic-Based Path-Planning," **In**: *Proceedings of ICAPS Workshop on Planning Under Uncertainty for Autonomous Systems* (Monterey, California, USA, 2005) pp. 9–18.
9. P. Hart, N. Nilsson and B. Rafael, "A formal basis for the heuristic determination of minimum cost paths," *J. IEEE Trans. Syst. Sci. Cybern.* (SSC) **4**(2), 100–107 (1968).
10. D. Hsu, J. Latombe and H. Kurniawati, "On the probabilistic foundations of probabilistic roadmap planning," *Int. J. Robot. Res.* **25**(7), 627–643 (2006).
11. D. Hsu, G. Sanchez-Ante and Z. Sun, "Hybrid PRM Sampling with a Cost-Sensitive Adaptive Strategy," **In**: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (Barcelona, Spain, 2005) pp. 3885–3891.
12. L. Kavraki, P. Svestka, J. C. Latombe and M. Overmars, "Probabilistic Roadmaps for Path Planning in High Dimensional Configuration Spaces," *IEEE Trans. Robot. Autom.* **12**(4), 566–580 (1996).
13. S. Koenig and M. Likhachev, "D*lite," **In**: *Proceedings of the 18th National Conference on Artificial Intelligence (AAAI/IAAI)* (Edmonton, Alberta, Canada, 2002) pp. 476–483.
14. S. Koenig and M. Likhachev, "Adaptive a*," **In**: *Proceedings of the 4th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Utrecht, The Netherlands, 2005) pp. 1311–1312.
15. H. Kurniawati and D. Hsu, "Workspace Importance Sampling for Probabilistic Roadmap Planning," **In**: *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* (Sendai, Japan, 2004) pp. 1618–1623.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press

16. H. Kurniawati and D. Hsu, "Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning," **In**: *Proceedings of the 7th International Workshop on the Algorithmic Foundations of Robotics (WAFR)*, New York, USA (S. Akella et al., eds.) (Springer, Berlin, Germany, 2006).
17. S. M. Lavalle, *Planning Algorithms* (Cambridge University Press, Cambridge, UK, 2006).
18. S. LaValle, M. Branicky and S. Lindemann, "On the relationship between classical grid search and probabilistic roadmaps," *Int. J. Robot. Res.* **23**(7–8), 673–692 (2004).
19. M. Likhachev, D. Ferguson, G. J. Gordon, A. Stentz and S. Thrun, "Anytime search in dynamic graphs," *Artif. Intell.* **172**(14), 1613–1643 (2008).
20. M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz and S. Thrun, "Anytime Dynamic a*: An Anytime, Replanning Algorithm," **In**: *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)* (Monterey, California, USA, 2005) pp. 262–271.
21. M. Likhachev, G. Gordon and S. Thrun, "ARA*: Anytime A* Search with Provable Bounds on Sub-Optimality," *Proceedings of the 17th Annual Conference on Neural Information Processing Systems (NIPS)* (Vancouver, British Columbia, Canada, 2003).
22. 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," *J. Signal Process.* **83**(11), 2387–2396 (2003).
23. N. Nilsson, *Principles of Artificial Intelligence* (Tioga, Wellsboro, PA, 1980).
24. M. Saha, J. Latombe, Y. Chang and F. Prinz, "Finding narrow passages with probabilistic roadmaps: The small-step retraction method," *J. Auton. Robots* **19**(3), 301–319 (2005).
25. G. Sanchez and J. Latombe, "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking," **In**: *Proceedings of the 10th International Symposium on Robotics Research (ISRR)* (Lorne, Victoria, Australia, 2001) pp. 403–417.
26. D. Sent and M. Overmars, "Motion Planning in Environments with Danger zones," **In**: *Proceedings of IEEE International Conference on Robotics and Automation (ICRA)* (Seoul, Korea, 2001) pp. 1488–1493.
27. X. Sun, S. Koenig and W. Yeoh, "Generalized Adaptive a*," **In**: *Proceedings of the 7th International Conference on Autonomous Agents and Multiagent Systems (AAMAS)* (Estoril, Portugal, 2008) pp. 469–476.

https://doi.org/10.1017/S0263574713000428 Published online by Cambridge University Press