The 2009 IEEE/RSJ International Conference on
Intelligent Robots and Systems
October 11-15, 2009 St. Louis, USA

Collision-probability Constrained PRM for a Manipulator with Base Pose Uncertainty

Yifeng Huang
RAMP, School of Engineering Science
Email: yhuangf@cs.sfu.ca

Kamal Gupta
RAMP, School of Engineering Science
Email: kamal@cs.sfu.ca

Abstract— **We address the motion planning problem for a manipulator system with base pose uncertainty, e.g., when the manipulator is mounted on a mobile base. Using a particle based representation for the uncertainty, we extend the PRM (probabilistic roadmap) approach to deal with this base uncertainty. Because of the uncertainty, a path for the manipulator is associated with a probability of being collision-free, which fundamentally changes the nature of the PRM's query phase. We plan for a shortest path such that the probability of the manipulator being collision-free is higher than a user defined threshold, were the manipulator to follow the path. The path query problem becomes a collision probability constrained shortest path problem (CP-CSPP), and is shown as NP-hard w.r.t. the number of the particles [1]. We then present a lazy query algorithm, called Lazy-CPC-PRM (collision probability constrained LazyPRM), based on a k-shortest path algorithm in conjunction with a labeling algorithm. Lazy-CPC-PRM exploits a key insight that if a portion of a path considered by the algorithm is invalid (the probability of it being collision-free is less than a threshold) or is dominated by another sub-path, then all the longer paths containing this portion can not be the solution path. This leads to significant efficiency gains in practice. Although, worst case complexity is exponential in the number of particles, we empirically show the effectiveness of our query algorithm with 30 particles for a simulated 3-dof manipulator mounted on a mobile base.**

I. Introduction

This work is motivated by planning motion for a mobile manipulator system. One of the major advantages of a mobile manipulator system is the mobility introduced by the base, which is capable of bringing the on board manipulator into a designated work area for carrying out manipulator tasks. For example, a camera mounted at manipulator's end-effector can be used to inspect areas behind obstacles. However, the mobile base's true position w.r.t the environment is not perfectly known due to localization uncertainty [2]. The key focus of this paper is to incorporate this uncertainty within a sampling based motion planning algorithm. We consider the problem where the base is kept still while the on board manipulator is required to move from a start configuration to a goal configuration in a known environment. Coupled with the assumption that manipulator motion is precise, a key aspect of this problem is that the uncertainty does not grow and remains the same as the manipulator moves.

Past works on motion planning for mobile-manipulator systems have often ignored the base uncertainty and assumed perfect base motion [3]. Motion planning with robot position uncertainty (often assuming a point robot) has been studied earlier, and has regained attention recently for mobile robots. Different formulations of the problem, different representations of the uncertainty of the sensing actions/sensor models, and different planning methods have been used. There is not enough space to review all of these here, but comprehensive treatments can be found in [4],[5]. We only mention the work most relevant to ours. The position uncertainty is represented either by geometric bounding sets [4], or by probability distributions, which can be either analytical, say gaussian [6],[7], or particle filters [2].

Earlier planning algorithms have used back projection (with bounding sets for uncertainty model) [4] and gradient descent (with probabilistic Gaussian uncertainty model) [8]. In addition, partially observable Markov decision processe (POMDP) has been proposed as a general framework for planning with uncertainty, however it is computationally intensive [9].

Many recent works in planning (with uncertainty) model a robot's localization uncertainty probabilistically and incorporate it into a motion plan. In [10][11][12] [13][14][15] (with probabilistic Gaussian uncertainty model) and [16][17] (with particle based uncertainty model), the authors extend standard motion planning approaches which search in C-space, by incorporating an extra uncertainty dimension. Among these works, sampling based planning approaches such as RRT or PRM have been extended to deal with uncertainty [15][16][17]. Besides robot's localization uncertainty, map uncertainty has also been addressed in [18][19][20], and leads to "similar" issues as for localization uncertainty, although specifics differ.

A key issue in extending PRM/RRT to deal with uncertainty is the cost of computation of probability of collision at a configuration (and along a path, which can be thought of as a sequence of configurations). [16] showed, for the mobile robot case, that representing uncertainty by a set of particles, a common representation used in SLAM (simultaneously localization and mapping) algorithms for mobile robots [2], and hence a natural choice for representing uncertainty, leads to succinct formulations for evaluation of collision probabilities, which is rather difficult with Gaussian models [19]. [18] uses a nearest point approximation technique (to compute the probabilities) which is difficult to quantify. [19] proposes an interesting approach that computes bounds on probabilities with gaussian models.

We therefore use particles to represent the base uncertainty, where each particle represents a base pose associated with a weight, which indicates the probability of this particle being the true base pose. A path for the manipulator is no longer simply either in-collision or collision-free, but is associated with a probability of being collision-free, were the manipulator to execute the path. As for the mobile robot case, the particle based uncertainty representation facilitates a concise expression for the probability of being collision-free for a manipulator path.

We then extend the probabilistic roadmap (PRM) [21] approach to plan for manipulator motions with base uncertainty. Given a roadmap, our formulation of the query phase is to search for a shortest path with the added constraint that

its probability of being collision-free is higher than a given threshold, an instance of constrained shortest path planning (CSPP) problem [22][1]. We call our path query version collision probability constrained shortest path problem (CP-CSPP), and the resulting framework, CPC-PRM (collision probability constrained PRM), and the single-query lazy version, Lazy-CPC-PRM. The latter is a key focus for us, since the mobile base may often need to be moved to other working areas, hence the roadmap needs to be reconstructed often. We show that CP-CSPP is NP-hard w.r.t number of the particles (Proof omitted here; see[1]). A labeling algorithm [22] can be used to solve CP-CSPP.

Please note that an alternative formulation, such as in [19] that combines the two independent criteria of path length and collision probability into one weighted linear objective function, is simpler to solve, however can yield short paths with unacceptably low probabilities of being collision-free, depending on the predefined weights, the trade-off between the two weights being somewhat arbitrary to start with. Another alternative [8] uses bounded sets to represent uncertainty and "enlarges" the robot by this amount. This is clearly a very conservative approach that will miss paths in narrow regions.

The path query problem in Lazy-CPC-PRM is fundamentally different from the simple shortest path in the standard Lazy-PRM [23] in that roadmap edges cannot simply be deleted since the edge status is not simple collision-free/in-collision, instead a collision-free probability is associated with it. Instead, one could use a k-shortest path algorithm [24] to generate alternative paths for verification in conjunction with a labeling algorithm. However, a naive implementation will lead to an efficiency problem, because the number of paths over the roadmap is exponential w.r.t the number of nodes in the roadmap. We present an efficient algorithm, i.e., Lazy-CPC-PRM that exploits a key insight that if a portion of a path considered by the algorithm is invalid (the probability of it being collision-free is less than a threshold) or is dominated (precisely defined later) by another sub-path, then all the longer paths containing this portion can be ignored from further consideration. This eliminates a large number of paths for subsequent verification, hence leading to significant efficiency. We show in our simulations some typical motion planning problems solved by Lazy-CPC-PRM for a simulated 3-dof manipulator on a mobile base, with 30 particles representing the robot uncertainty, a typical number used for particle based robot localization algorithms [2], and a roadmap size of up to around 1000 nodes and 10000 edges.

In summary, the key contributions of our work are: (i) a collision-probability constrained formulation of path planning problem for a manipulator with base uncertainty, (ii) a particle filter representation of base uncertainty that facilitates collision probability formulation for the manipulator, (iii) complexity results for the problem in terms of number of particles used to represent the uncertainty, and finally (iv) a key insight (mentioned above) that leads to an efficient lazy algorithm.

[1]Please note that the algorithms presented in [13][14][10][15] are also an instance of CSPP.

II. Problem Formulation

We assume the manipulator's motion can be precisely controlled. Also, the environment is assumed to be known (or acquired via previous sensing). However, the manipulator's base pose (position and orientation) is not precisely known, and is represented with a set of N weighted particles. Each particle indicates a possible base configuration $q_b = [x, y, \theta]$. Let $q_b^{[i]}$ be the i^{th} particle. The weight of the i^{th} particle, i.e., $\omega^{[i]}$, represents the probability of $q_b^{[i]}$ being the true base configuration [2], with $\sum_i^N \omega^{[i]} = 1$. Let $q = [\theta_1, \cdots, \theta_d] \in \mathcal{C}_m$, the C-space of the manipulator, be a configuration for the manipulator w.r.t the base frame, where d is the number of degrees of freedom of the manipulator. We use $q^{[i]} = [q_b^{[i]}, q]$ to represent a configuration for whole mobile manipulator, corresponding to the i^{th} particle $q_b^{[i]}$ and the manipulator configuration q. Ideally, we should have used $q_{bm}^{[i]}$ instead of $q^{[i]}$, but it becomes cumbersome. So superscript $[i]$ on q (or path π mentioned later) will always indicate mobile manipulator configuration corresponding to the i^{th} base particle, with manipulator configuration at q.

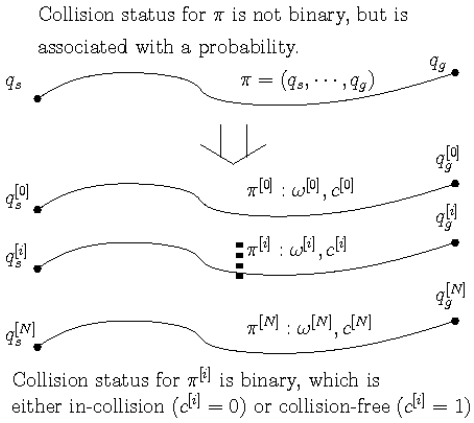


Fig. 1. For a manipulator path, there are correspondingly N possible sequences of configurations swept by the whole mobile manipulator.

A path π consists of a sequence of manipulator configurations: $\pi = (q_s, ...q_g)$, which connects the start configuration q_s and the goal configuration q_g. Were the manipulator to execute the path π (the base remains stationary), as shown in Fig. 1, with the base pose uncertainty represented by particles, there are correspondingly N different volumes in workspace swept by the manipulator corresponding to $\pi^{[i]} = (q_s^{[i]}, ...q_g^{[i]}), i = (1, \cdots, N)$.

The probability of $\pi^{[i]}$ being the true path traversed by the whole manipulator is $\omega^{[i]}$, and the collision status of $\pi^{[i]}$ is binary, denoted by $c^{[i]} = 0$ (collision) or 1 (collision-free). Let Π_π be the binary random variable that represents if the manipulator would be in-collision or free ($\Pi_\pi = 0$ or 1), were the manipulator to execute the path π. The probability of path π being collision-free, i.e., $p(\Pi_\pi = 1)$, then is:

$$p(\Pi_\pi = 1) = \sum_i^N (\omega^{[i]} * c^{[i]}) \quad (1)$$

We would like to extend PRM to plan a path for the manipulator with the base pose uncertainty. To construct a roadmap G in $\mathcal{C}_m$, we generate samples of the manipulator configuration $q \in \mathcal{C}_m$, and connect samples within a distance

threshold, as in the standard PRM construction [21]. A node n of G is a configuration for the manipulator (denoted as q_n), and edges between two nodes n_i,n_j (denoted as $e_{[n_i,n_j]}$) are line segments in $\mathcal{C}_m$. Note that the collision status for samples and edges of G are not binary, but are associated with a probability of being collision-free, which can be easily calculated in a way similar to Equ. 1.

For a roadmap G, a path π that connects the start node n_s and goal node n_g is feasible, if $p(\Pi_\pi = 1) \geq \delta$, where $\delta \in [0, 1]$ is a user defined threshold. The problem, called collision probability constrained shortest path problem (CP-CSPP), in the query phase is to return an optimized solution path π^*:

$$\begin{aligned} \pi^* &= \underset{\pi}{\operatorname{argmin}}\, C(\pi) \\ s.t.\ & p(\Pi_{\pi^*} = 1) \geq \delta \end{aligned} \tag{2}$$

where $C(\pi)$ is the length (or cost) of the path π.

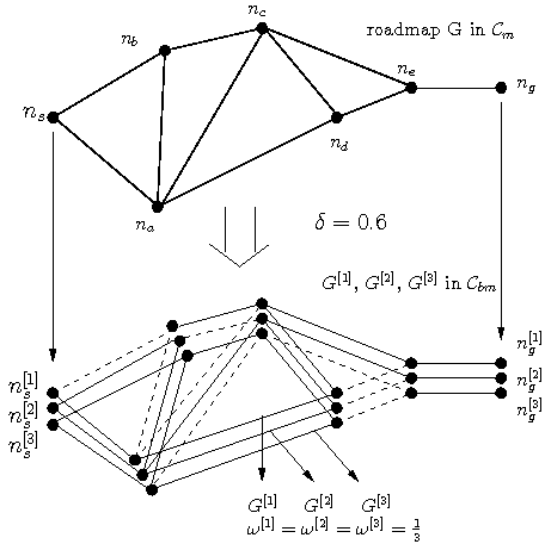


Fig. 2. Roadmap G in $\mathcal{C}_m$, and the corresponding graph set $\mathcal{G} = \{G^{[1]}, G^{[2]}, G^{[3]}\}$ constructed in C_{bm} ($N = 3$).

A. Underlying structure of PRM with base pose uncertainty

The graph G is constructed in $\mathcal{C}_m$. But it has an underlying structure, because of base uncertainty. Corresponding to each node n and edge e of G, we use $n^{[i]}$, $e^{[i]}$ ($e^{[i]}_{[n_j,n_k]}$) to denote a node or edge in the configuration space for the whole mobile manipulator (denoted as $\mathcal{C}_{bm}$), with the base pose indicated by the i^{th} particle. $G^{[i]}$ denotes the graph composed of nodes $n^{[i]}$ and edges $e^{[i]}$, and let $\mathcal{G} = \{G^{[1]}, \cdots, G^N\}$. Therefore, for a roadmap G in the $\mathcal{C}_m$, there exists correspondingly a set of N graphs in $\mathcal{C}_{bm}$. Fig. 2 shows an example, where $N = 3$, the weight of each particle is set equal to $\frac{1}{3}$. The dashed lines in $G^{[1]}$, $G^{[2]}$ and $G^{[3]}$ are edges in-collision, and the solid line are collision-free.

Let Π_e be the binary random variable that the mobile manipulator is in-collision or free ($\Pi_e = 0$ or 1), were the manipulator to travel along the edge e. Note for a path π in G consisting of a sequence of M edges $(e_1, e_2, \cdots, e_M)$, $c^{[i]}$ in Equ. 2 is given by $c^{[i]}_{e_1} \wedge c^{[i]}_{e_2} \cdots, c^{[i]}_{e_M}$, where $c^{[i]}_{e_j} = 1$(or 0) if $e^{[i]}_j$ is collision-free (or in-collision). As a consequence, we have:

$$\begin{aligned} p(\Pi_\pi = 1) &= \sum_i^N (\omega^{[i]} * [c^{[i]}_{e_1} \wedge c^{[i]}_{e_2} \cdots, c^{[i]}_{e_M}]) \\ &\leq \sum_i^N (\omega^{[i]} * c^{[i]}_{e_j}) = p(\Pi_{e_j} = 1) \end{aligned} \tag{3}$$

where $e_j \in \pi, j = 1, \cdots, M$. Therefore, if $p(\Pi_e = 1)$ is lower than the threshold δ, the edge e can not be a part of a feasible path. We call such an edge in the roadmap G, as "N-edge" ("N" stands for edges that can not be part of solution), and the rest as "S-edge" ("S" stands for edges possibly be part of the solution). For example, in Fig. 2, if $\delta = 0.6$, both $e_{[n_c,n_a]}$ and $e_{[n_d,n_e]}$ are N-edges and the rest are S-edges. In a similar way, we distinguish nodes in G as N-nodes and S-nodes.

B. Query phase: the constrained shortest path problem

CP-CSPP is a particular version of the well known constrained shortest path problem (CSPP) [22], which is finding the shortest path that satisfies a given constraint. In our case, the constraint over the solution path is the probability of being free. Another version, the weight-constrained shortest path (WCSPP) problem has been shown as NP-hard w.r.t the size of the graph [22]. In our case, the constraint have a strong underlaying structure, which we exploit to show that CP-CSPP is NP-hard as well, but in terms of the number of the particles N (see [1] for proof).

C. Using labeling algorithm to solve CP-CSPP

We now introduce the labeling algorithm, with our specific variant, to solve the general query problem over a given roadmap. The detailed algorithm below is mainly based on what is described in [22]:

Let the roadmap graph be $G = (V, E)$, which consists of a set of nodes V and edges E. The start node and goal node are n_s and n_g, respectively. Let I_i be the index set of labels on a graph node n_i. Essentially, each $k \in I_i$ corresponds to a path π^k_i from the start node n_s to n_i. Let the probability of being free for the path π^k_i, i.e. $p(\Pi_{\pi^k_{n_i}} = 1)$, be denoted as P^k_i, and the length of the path π^k_i be C^k_i. The k^{th} label at node n_i is then a pair (P^k_i, C^k_i) that is associated with path π^k_i.

Clearly, the number of paths to a node could be exponential (in terms of number of nodes in the graph) in the worst case. To help reduce the search, two key concepts are used:

a) **domination**: We say label (P^k_i, C^k_i) dominates label (P^l_i, C^l_i) if 1) $C^k_i < C^l_i$, and 2) for two paths π^l_i and π^k_i, for all $j = 1, \cdots, N$ if $\pi^{[j],l}_i$ is collision-free, $\pi^{[j],k}_i$ is collision-free as well. Clearly, if (P^k_i, C^k_i) dominates (P^l_i, C^l_i), π^l_i can not be a portion of the optimal solution path π^*, since it can be replaced by π^k_i, which has lower cost. The domination relationship between two labels can be efficiently determined in $O(N)$.

b) **efficient**: We will call a label (P^k_i, C^k_i) as efficient, if it is not dominated by any other label at node n_i and $P^k_i \geq \delta$. Otherwise we call the label as inefficient.

The general idea is to maintain all possible paths to each node, and to eliminate paths whose labels are not efficient. The set of labels is maintained as a tree structure, called

the label tree. The first label is initialized at node n_s, i.e., (P_s^1, C_s^1) with $P_s^1 = 1$ and $C_s^1 = 0$. Correspondingly, the root node of the label tree contains $[(P_s^1, C_s^1), n_s]$.

The algorithm proceeds iteratively from the start node. At each iteration, among all the labels that have not been extended before, a node whose label has minimal cost, say node n_i, with label (P_i^k, C_i^k) is chosen, and is extended to all the neighbors of n_i. This involves, for all neighbors n_j of n_i, 1) computing the label (P_j, C_j), associated with the concatenated path $[\pi_i^k, e_{[n_i,n_j]}]$, 2) discarding (P_j, C_j) if it is not efficient w.r.t existing labels at n_j, otherwise, a) storing it at node n_j and creating a new index l for it. b) updating the label tree with $[(P_j^l, C_j^l), n_j]$ added as a child of $[(P_i^k, C_i^k), n_i]$. This procedure is repeated until the label tree contains a node $[(P_g^l, C_g^l), n_g]$, such that n_g is the goal node, and C_g^l is minimal among all the labels that have not been extended. The solution path can be retrieved from the tree by tracing back from this tree node $[(P_g^l, C_g^l), n_g]$ to the root of the label tree. To check the domination between labels at a graph node n_i, a list of labels L_i is stored at n_i. Readers can refer to [22] for more detailed information about the labeling algorithm. We now adapt this labeling algorithm to Lazy-PRM.

III. Lazy-PRM with base uncertainty

For a given roadmap, the standard Lazy version first searches for a shortest path over the roadmap without considering their collision-free status Then, the path is verified (checked for collision). If it is collision-free, success is reported, otherwise, there must exist an edge along the path that is in collision. This edge is deleted from the roadmap and the shortest path algorithm is applied again over the modified roadmap to acquire an alternative path for verification. As mentioned in the introduction, in our case, we can not simply delete an edge along the path from the roadmap G. For example, in Fig. 2, in the roadmap with 3 particles each equally weighted as $\frac{1}{3}$, the threshold δ for collision-free probability for a solution path is set as 0.6. Assume that the first path being verified is n_s, n_b, n_c, n_e, n_g. It is easy to tell that the probability of being free for the path is 0, but all the four edges along the path are S-edges. If we delete the S-edge $e_{[n_c,n_e]}$, we will miss the solution path $n_s, n_a, n_d, n_c, n_e, n_g$, which is the only path that satisfies the collision probability constraint in this example. Instead of deleting an edge, we should use the next shortest path (in the same roadmap G) that satisfies the collision probability constraint. This suggests the use of a k-shortest path algorithm [24] to acquire alternative paths for verification. However, a naive version of the algorithm, i.e., to iteratively call the k-shortest path algorithm and verify the path until success is reported or all possible paths have been exhausted, is not practical, because the number of paths is exponential w.r.t the number of nodes and edges in the roadmap.

The key insight of our proposed technique is that certain path can be eliminated from consideration if a portion of the path has been identified as not feasible or is dominated by another sub-path. For instance, in Fig. 2, say the first shortest path being verified is $\pi_1 = \{n_s, n_b, n_c, n_e, n_g\}$. We can tell that the probability of being collision-free for a portion of π_1, i.e., $\{n_s, n_b, n_c\}$ is lower than the threshold. Then all the paths from n_s to n_g that contain this portion will not be the solution path. If the second shortest path being verified is $\pi_2 = \{n_s, n_a, n_b, n_c, n_e, n_g\}$, the portion of which $\{n_s, n_a, n_b\}$ is dominated by the portion $\{n_s, n_b\}$, then all paths that contain n_s, n_a, n_b will not be the solutio path either. Hence, many candidate path can be eliminated from further consideration, leading to significant efficiency in the search. We now explain the precise mechanism to do so.

A. k-shortest path algorithm

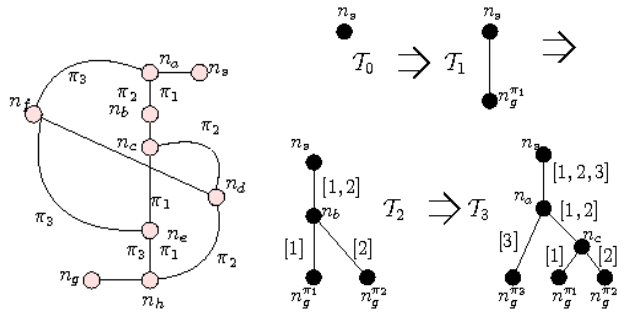
We use a recent k-shortest path algorithm [24], with run time complexity of $O(k|V|(|E|+|V|log|V|))$, where $|V|, |E|$ denote the number of graph vertices and edges. Other versions of k-shortest path algorithm with better performance might also be considered [25]. The data structure applied in [24], called the shortest path tree, allows the algorithm to be easily incorporated into our lazy path query technique discussed later.

The algorithm starting from the shortest path, iteratively returns the next shortest path. Let B_i be the set of i shortest paths returned, and $\bar{B}_i$ be the set of rest of the paths in the graph. Each time a shortest path, say the i^{th} shortest path, is returned, the algorithm divides all the paths in $\bar{B}_i$ into separate equivalent classes and computes the shortest path for each one of these classes. Clearly, the shortest among these will be the $i + 1^{th}$ shortest path.

How $\bar{B}_i$ is divided into separate classes is related to how the paths in B_i branch off from each other. From the start point, all paths in B_i share a common portion (which may be null) in the graph. Then, at least one of the paths will branch off, which divides the i paths into subsets, where paths in each subset will further share (longer) common portions in the graph. Each subset will be further divided in a similar fashion until each subset contains only one path.

The branching off process mentioned above can be topologically encoded in a tree structure, called the shortest path tree $\mathcal{T}_i$. We use an example to illustrate this. In Figure 3(a), consider three paths π_1, π_2, and π_3. All three paths share the common portion n_s, n_a. The first branch off is at node n_a in the graph, where π_3 branches off, and the three paths are divided into two subsets, $\{\pi_1, \pi_2\}$ and $\{\pi_3\}$. Correspondingly, in $\mathcal{T}_3$ (Fig. 3 (b)), branch $[n_s, n_a]$ encodes the common portion in the graph shared by all three paths. Tree node n_a, has two children, $n_g^{\pi_3}$ and n_c. The tree branch $[n_a, n_c]$ encodes the common portion further shared by $\{\pi_1, \pi_2\}$ from n_a onwards until n_c. Tree node $n_g^{\pi_3}$ is a leaf node, because the corresponding subset $\{\pi_3\}$ contains only one path. Finally, the subset $\{\pi_1, \pi_2\}$ is further divided, due to the two paths branching off at n_c in graph, into two subsets, i.e., $\{\pi_1\}$ and $\{\pi_2\}$. This leads to the two leaf nodes, $n_g^{\pi_1}$ and $n_g^{\pi_2}$ in $\mathcal{T}_i$ as children of tree node n_c.

If we use the paths in B_i, and an extra path in $\bar{B}_i$ to build a shortest path tree, denoted as $\mathcal{T}_{i+1}^{'}$, it will have an extra new branch (possibly together with a non-leaf node) and an extra leaf node as compared to $\mathcal{T}_i$. This is similar to building $\mathcal{T}_3$ from $\mathcal{T}_2$, as in Fig. 3 (b), except that the extra path is not the next shortest path. We categorize paths in $\bar{B}_i$ based on where the corresponding new branch or the new non-leaf node would be introduced in $\mathcal{T}_i$. There are two distinct cases



(a) shortest paths over the graph (b) Updating of the shortest path tree

Fig. 3. Build branching structure $\mathcal{T}_3$

here: a) a new branch occurs at node n_u; this corresponds to a set of paths in $\bar{B}_i$, denoted by $C(n_u)$; and b) a new non-leaf node occurs somewhere in branch $[n_u, n_v]$ together with a new branch emanating from it, and terminating in a new leaf node; this corresponds to a set of paths in $\bar{B}_i$, denoted by $C(n_u, n_v)$. All paths in $\bar{B}_i$ are thus divided into a set of classes (called equivalent classes) corresponding to nodes and branches of $\mathcal{T}_i$.

[24] uses an efficient algorithm to acquire the shortest path for each of the equivalent class. These shortest paths are stored in a heap. In each iteration, the shortest path in the heap is popped out, and is the next shortest path. The tree structure $\mathcal{T}_i$ is also updated. We omit these details here for lack of space (See [24] for detail).

B. Lazy-CPC-PRM

Our Lazy-CPC-PRM algorithm (Alg. 1) works as follows. We maintain a label tree, using a labeling algorithm that creates labels but only for graph nodes along the paths being verified (line-11). Given a path to verify, the labeling algorithm starts labeling nodes from the start node n_s by extending labels along the path. If the generated label at a graph node is determined as efficient (as explained in labeling algorithm in Sec. II-C), it is stored and is added as a child of the tree node, from which this new efficient label was extended. Then the next node in the path is considered, and procedure is repeated. If an extended label is determined as inefficient, the path verification step stops. The path will not be the solution path, and the next shortest path is considered for verification.

Assume that i paths have been verified and failed. To get the next path for verification, we modify the k-shortest path algorithm. Let us focus on the equivalent classes corresponding to the nodes and branches of $\mathcal{T}_i$. The key is to identify an equivalent class such that none of the paths in it can be the solution path. This entire class can be eliminated from consideration for next shortest path, hence leading to efficiency.

Given $\mathcal{T}_i$, consider an equivalent class $C(n_u, n_v)$ (or $C(n_u)$), which is a subset of all candidates paths in $\bar{B}_i$. Note that, of starting from n_s, all the path in $C(n_u, n_v)$ (or $C(n_u)$) share the common portion, that is encoded by the consecutive branches from n_s to n_u in $\mathcal{T}_i$. We denote this common portion as prefixPath(n_u). If the label that along this common portion in graph is inefficient, none of the candidate paths in the entire equivalent class can be the optimal solution path and the entire class can be eliminated from further consideration (line-17, Alg. 1). We refer to these equivalent classes as inefficient.

To tell whether an equivalent class is efficient or not (line-17), we check the label tree (created in line-11). For the equivalent class $C(n_u, n_v)$ (or $C(n_u)$) that corresponds to a tree branch (n_u, n_v) (or a node n_u) of $\mathcal{T}_i$, the prefixPath(n_u) is a portion of at least one of previous i shortest paths in G, which has been verified. The verification information has been stored as labels in the label tree. For equivalent class ($C(n_u, n_v)$ or $C(n_u)$), we track along the label tree from the root node along the prefixPath(n_u). If we encounter a label tree node (the leaf node included) that contains node n_u, the equivalent class is kept. Otherwise the equivalent class is inefficient and is discarded.

Algorithm 1: Lazy-CPC-PRM: Lazy PRM algorithm with base pose uncertainty

```
1  begin
2      R = BuildInitRoadMap()
3      Initialize T_0 to n_s; H = ∅
4      Calculate the 1^st shortest path π_1, insert π_1 into heap H.
5      InitLabeling(), i=1.
6      while TimeUp() do
7          Extract the shortest path from the heap, which will be the next
           shortest path π_i.
8          if π_i == NULL then
9              EnhanceRoadMap();
10         else
11             if verifyPath(π_i) then
12                 return π_i;
13             else
14                 Extract the next shortest path π_i from H.
15                 T_i = updateTreeStructure(π, T_{i-1})
16                 for Each new node and branch in T_i do
17                     if the corresponding equivalent class is efficient
                       then
18                         Compute the shortest path π, from n_s to n_g,
                           in the equivalent class, corresponding to the
                           new node (or branch).
19                         Insert π into heap H
20         i++
21 end
```

IV. SIMULATIONS

We have run preliminary tests of Lazy-CPC-PRM in a simulated environment in 2D[2]. We use the MobileSim[26] program as our mobile robot simulator to simulate a PowerBot™ [26], with a size of about 80cm by 65cm. A simulated range sensor, with sensing range of 4.0 meters (approximately the same range as the Hokuyo range sensor), is mounted on the front of the mobile base. The simulated manipulator on board has three degrees of freedom, and each link is 90cm long. We run our simulation under Linux on an Intel Core-2 due 3.0Ghz computer with 4GB memory.

We evaluated the Lazy-CPC-PRM algorithm in three tasks as shown in Fig. 4. We assume the robot is in the middle of an exploration task. The dark boundaries are sensed obstacles, light gray regions are unknown and white regions are free. We simulate an "inspection" task, i.e., a manually generated desired goal configuration is as if the arm (say, with a camera mounted on the end-effector) was inspecting

[2]We have also implemented the Lazy-CPC-PRM in simulated 3D environments with a six-dof manipulator mounted on a Powerbot. The initial results, although promising, are still being compiled.

an area behind the obstacles in Fig. 4. Fig. 4 (a) shows the manipulator moving from an unfolded start configuration to another unfolded goal configuration. Figs. 4 (b) and (c) show the manipulator moving from a folded start configuration to an unfolded goal configuration.

We used the true position of the base as the mean and apply a Gaussian pdf to generate 30 particles (a number that is commonly used in particle based localization algorithms [2]). The Gaussian pdf's co-variance matrix is diagonal with their values being (0.12m,0.12m,3^o) for LARGE uncertainty and (0.07m,0.07m,1^o) for SMALL uncertainty, in x,y and θ dimension, respectively.

In all our simulations, we set the threshold δ for the collision-free probability of a valid path as 0.8. We ran the planner(s) 30 times for each problem. Note that for each task, we also set a time limit, which is 1000 and 1500 seconds for SMALL and LARGE uncertainty case, respectively.

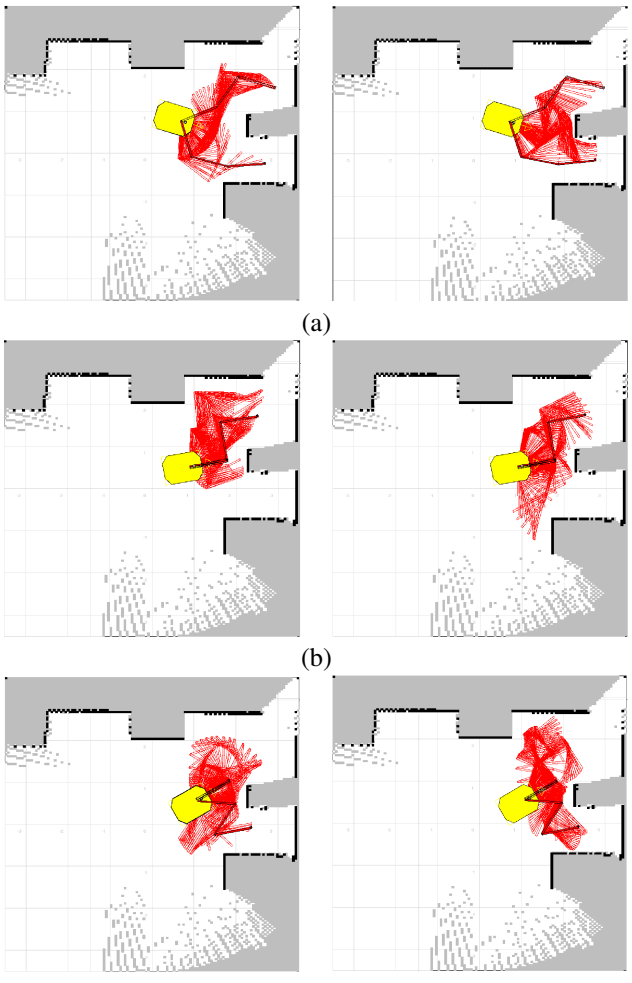


Fig. 4. Three different start/goal problems. Left: paths planned with Lazy-CPC-PRM with base uncertainty (LARGE). Right: paths planned with classic PRM with no uncertainty taken into account and with the base pose at the most weighted particle.

Our first observation is how the base uncertainty affects the planned path. In Fig. 4, the figures in left column illustrate some typical paths planned with the base uncertainty (LARGE) taken into consideration, while sub-figures in right column illustrate paths planned with the true base pose (without any uncertainty) using classic PRM. Note that paths planned with classic PRM are, at times, closer to the obstacles, as in the top right figure in Fig. 4(a). For the three start/goal problems in Fig. 4, the probability of being collision-free for the paths planned with Lazy-CPC-PRM are 0.985, 0.808 and 0.815, respectively. We also calculated this probability for the paths planned using classic PRM and they were, 0.303, 0.740 and 0.825, respectively. This clearly illustrates that the paths planned with the base uncertainty tend to be consistently safer and hence the desirability of incorporating the base uncertainty while planning.

Table-I and II show various average statistics for Lazy-CPC-PRM. In both tables, "Unc." is the size of the base uncertainty, which is marked as "S" and "L", standing for LARGE and SMALL uncertainty as mentioned before. The second column "G: Av. size" is the average graph size, including number of edges and nodes, used to find a solution. The third column, "Av. Time" records the average time spent to find the path (To.), to construct the roadmap (Cons.), to search the graph (sear.), and to check collisions (coll.). In the fourth column, i.e. "Av. Search" shows the number of shortest paths verified before the solution path is found ("paths"), and the number of the equivalent classes that are inefficient ("In.Cls."). Finally, the average number of edges being checked for collision ("♯E"), and the average total number calls of the collision checker ("♯Coll.") is listed in the last column (Av. Col.).

As shown in Table-I, Lazy-CPC-PRM is capable of solving the single query efficiently. It saves significant time on collision checking, but does spend, as expected, extra time on the k-shortest path calculation. We also observe, as expected, that increasing the base uncertainty tends to make it harder to find a path. For example, in problem shown in Fig. 4(a) the number of paths needed to be verified increases almost 10 times, as shown in Av. Search collum, in row A.

Our Lazy-CPC-PRM is efficient because it prunes inefficient equivalent classes. We also ran it without pruning the inefficient classes, for each of these three tasks, and show the results in Table-II. Clearly, the number of paths to be verified is dramatically higher (up to by an order of 10) as compared to corresponding numbers in Table-I. Table-III shows the number of times Lazy-CPC-PRM, with and without pruning inefficient classes, runs over the time limit, as well as the worst case run time. The time limit is 1000 and 1500 seconds for SMALL and LARGE uncertainty, respectively. Lazy-CPC-PRM clearly benefits from pruning those inefficient equivalent classes.

V. Discussion

We considered the problem of manipulator path planning with base pose uncertainty. We use a particle based representation for uncertainty and formulate the problem as a constrained shortest path problem, the constraint being that the probability of being collision-free for a path be greater than a user defined threshold. We show that the problem is NP-hard (in the number of particles). We propose Lazy-CPC-PRM, the core of which is a path query algorithm that smartly uses a k-shortest path algorithm in conjunction with a labeling algorithm to weed out whole classes of paths. Effectiveness of the algorithm is shown via results on a simulated 3-dof planar arm. We would like to explore the

	Unc.	G: Av. size		Av. Time				Av. Search		Av. Coll.	
		Edges	Nodes	To.	Cons.	Sear.	Col.	Paths	In.Cls.	♯ E	♯ Coll.
A	S	6838	507	13.84	2.52	1.74	9.58	97	37	79	153531
	L	7678	527	26.51	1.48	12.71	12.32	1229	1461	134	190784
B	S	6678	503	51.38	2.05	22.23	27.10	1885	2459	232	319204
	L	6801	507	88.62	2.81	55.08	31.73	4178	5816	321	445487
C	S	6838	507	97.74	2.96	49.59	45.19	3283	4603	395	533990
	L	6767	505	129.5	2.69	76.41	50.40	5478	7300	461	631613

TABLE I: RESULTS FOR LAZY-CPC-PRM WITH PRUNING.

	Unc.	G: Av. size		Av. Time				Av. Search		Av. Coll.	
		Edges	Nodes	To.	Cons.	Sear.	Col.	Paths	In.Cls.	♯ E	♯ Coll.
A	S	7186	520	49.24	2.69	38.47	8.08	984	0	98	140297
	L	7278	505	156.15	2.35	140.1	13.7	5533	0	124	184839
B	S	6630	502	95.39	3.28	65.20	26.91	2215	0	220	303790
	L	6824	507	184.99	2.33	154.64	28.02	5349	0	248	343754
C	S	6996	512	185.34	3.29	149.86	32.19	4340	0	334	455713
	L	7656	527	543.50	2.91	509.76	30.83	14586	0	313	432534

TABLE II: RESULTS FOR LAZY-CPC-PRM WITHOUT PRUNING.

		A		B		C	
		S	L	S	L	S	L
Failure times	with pruning	0	0	0	0	0	0
	without pruning	1	1	2	3	4	10
Worst case time (Sec.)	with pruning	25.06	144.58	589.13	311.24	537.81	902.94
	without pruning	> 1000	> 1500	> 1000	> 1500	> 1000	> 1500

TABLE III: FAILURE TIMES FOR LAZY-CPC-PRM (30 RUNS) WITH AND WITHOUT PRUNING.

effect of threshold δ in Lazy-CPC-PRM. Intuitively, setting a high threshold will produce safer paths, and would tend to reduce the computational cost of graph searching, since a large number of edges may be invalid right away, but at the same time, it will increase the difficulty level of finding such a path (more nodes may be required for the roadmap). On the other hand, reducing the threshold (where applications allow it, since it would mean more chance of collision) would increase the computational cost of graph searching since there would be more valid edges, but fewer roadmap nodes could be needed to find a solution path.

REFERENCES

[1] Y. Huang. Motion planning with localization and mapping uncertainties for a mobile manipulator in exploration and inspection tasks. *Ph.D thesis*, Engineering Science, Simon Fraser University, 2009.

[2] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005.

[3] Y. Yang and O. Brock. Elastic Roadmaps: Globally Task-Consistent Motion for Autonomous Mobile Manipulation in Dynamic Environments. *Robotics Science and Systems 2006 (RSS2006)*, 21:34.

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

[5] S.M. LaValle. *Planning Algorithms*. Cambridge Unversity Press, U.K., 2006.

[6] S.M. LaValle and S.A. Hutchinson. An objective-based stochastic framework for manipulation planning. In *Proceedings IEEE/RSJ/GI International Conference on Intelligent Robots and Systems (IROS94)*, pages 1772–1779, 1994.

[7] F. Bourgault, A.A. Makarenko, S.B. Williams, B. Grocholsky, and H.F Durrant-Whyte. Information based adaptive robotic exploration. In *Proceedings of IEEE International conferenece on intelligent Robots and System (IROS02)*, pages 540–545, 2002.

[8] L.A. Page and A.C. Sanderson. A path-space search algorithm for motion planning with uncertainties. In *Proceedings of IEEE International Symposium on Assembly and Task Planning*, pages 334–340, 1995.

[9] L.P. Kaelbling, M.L. Littman, and A.R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1-2):99–134, 1998.

[10] B. Bouilly, T. Simeon, and R. Alami. A numerical technique for planning motion strategies of a mobile robot in presence of uncertainty. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA95)*, pages 1327–1332, Nagoya, JP, May 1995.

[11] A. Lambert and D. Gruyer. Safe path planning in an uncertain-configuration space. In *Proceedings of the 2002 IEEE International conferenece on Robotics and Automation (ICRA02)*, pages 4185–4190, Sept 2003.

[12] J. P. Gonzalez and A. Stentz. Planning with uncertainty in position: an optimal and efficient planner. In *Proceedings of IEEE International Conference on Intelligent Robots and Systems (IROS 2005)*, pages 1327–1332, May 2005.

[13] A. Censi, D. Calisi, A.D. Luca, and G. Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA08)*, pages 1798–1805, 2008.

[14] J.P. Gonzalez and A. Stentz. Replanning with uncertainty in position: Sensor updates vs. prior map updates. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA08)*, pages 1806–1813, 2008.

[15] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a gps-denied environment. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA08)*, pages 1814–1820, 2008.

[16] Y. Huang and K.K. Gupta. RRT-slam for motion planning with motion and map uncertainty for robot exploration. In *Proceedings of IEEE/RSJ International Conference on Robotics and System (IROS08)*, 2008.

[17] R. Pepy and A. Lambert. Safe path planning in an uncertain-configuration space using RRT. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS06)*, pages 5376–5381, 2006.

[18] P. E. Missiuro and N. Roy. Adapting probabilitic roadmaps to handle uncertain maps. In *Proceedings 1995 IEEE International Conference on Robotics and Automation*, pages 1261–1267, Orlando, US, May 2006.

[19] L.J. Guibas, D. Hsu, H. Kurniawati, and E. Rehman. Bounded uncertainty roadmaps for path planning.

[20] A. Nakhaei and F. Lamiraux. A framework for planning motions in stochastic maps. In *Control, Automation, Robotics and Vision, 2008. ICARCV 2008. 10th International Conference on*, pages 1959–1964, 2008.

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

[22] I. Dumitresce and N. Boland. Algorithms for the weight constrained shortest path problem. *Intl. Trans. in Op. Res.*, 8:15–29, 2001.

[23] R.Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *Proceedings of IEEE International Conference on Robotics and Automation (ICRA00)*, pages 521–528, 2000.

[24] J. Hershberger, M. Maxel, and S.Suri. Finding the k shortest simple paths: A new algorithm and its implementation. In *ACM Transactions on Algorithmes*, volume 3, 2007.

[25] D. Eppstein. Finding the k shortest paths. *SIAM Journal on Computing*, 28(2):652–673, 1999.

[26] http://www.mobilerobots.com.