Rapidly-Exploring Roadmaps: Weighing Exploration vs. Refinement in Optimal Motion Planning

Ron Alterovitz, Sachin Patil, and Anna Derbakova

***Abstract*— Computing globally optimal motion plans requires exploring the configuration space to identify reachable free space regions as well as refining understanding of already explored regions to find better paths. We present the rapidly-exploring roadmap (RRM), a new method for single-query optimal motion planning that allows the user to explicitly consider the trade-off between exploration and refinement. RRM initially explores the configuration space like a rapidly exploring random tree (RRT). Once a path is found, RRM uses a user-specified parameter to weigh whether to explore further or to refine the explored space by adding edges to the current roadmap to find higher quality paths in the explored space. Unlike prior methods, RRM does not focus solely on exploration or refine prematurely. We demonstrate the performance of RRM and the trade-off between exploration and refinement using two examples, a point robot moving in a plane and a concentric tube robot capable of following curved trajectories inside patient anatomy for minimally invasive medical procedures.**

I. Introduction

Motion planning requires exploring the configuration space of a robot or agent in order to find a sequence of feasible actions that maneuver the robot or agent around obstacles to a goal. However, not all motion plans are equal. In optimal motion planning, the objective is to find the best solution that optimizes relevant criteria, such as a path of minimum length, greatest clearance from obstacles, or minimum control effort. With improvements in computation speed and algorithms, we can strive to compute optimal motion plans in cases where in the past even feasible solutions were difficult to obtain.

The most successful approaches to motion planning in practice involve building discrete representations (trees or roadmaps) of the free space, the subset of the robot's configuration space where it can move without collision with obstacles. When a start and goal are specified and we are only concerned about a single query, building this discrete representation of the robot's free space introduces an inherent trade-off between exploration and refinement.

Exploration seeks to find free space where the robot can travel without colliding with obstacles. In contrast, *refinement* seeks to improve understanding of the configuration space in a local region. Refinement introduces multiple pathways to reach the same configuration, enabling selection of a path through a known free space region that optimizes some criteria. Computing a globally optimal plan in general requires both exploration and refinement.

We introduce a new motion planning method, the *rapidly-exploring roadmap (RRM)*, that allows the user to explicitly balance exploration and refinement while converging to an optimal plan. Our focus is on optimal single-query motion planning in constrained spaces. We are given start and goal configurations, obstacle locations, properties of the robot or agent, and a quality metric with which to evaluate plans. Our objective is to compute an optimal plan from the start configuration to the goal based on the given quality metric. RRM works for quality metrics that are additive over a path, such as distance. In computing a plan, RRM initially only iteratively explores the configuration space. Once exploration finds a feasible path from the start to the goal, RRM uses a user-specified parameter to weigh whether to (1) explore for new free space regions, or (2) refine the current explored space to find a solution that is closer to optimal. The method guarantees, under certain assumptions, that the computed path will converge to the globally optimal solution with probability 1 as the number of iterations increases.

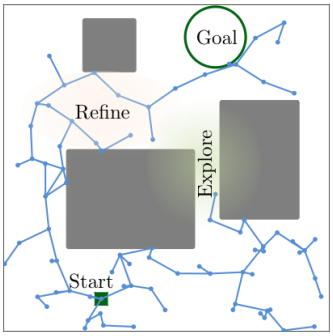

Fig. 1. *Exploration* seeks to find new regions of free space while *refinement* seeks to find better paths within the explored regions. In the RRM above, exploration could identify an entirely new region of free space, while refinement could reduce the length of the path to the goal. RRMs allow the user to explicitly specify the trade-off between exploration and refinement with the guarantee that the planner will return the optimal solution with probability 1 as computation time is allowed to increase.

RRMs combine ideas from the two most popular sampling-based motion planners: probabilistic roadmaps (PRMs) and rapidly-exploring random trees (RRTs) [5], [15]. For both PRMs and RRTs, the probability that the method fails to find a feasible solution (when one exists) decreases exponentially with the number of iterations [16]. However, Karaman and Frazzoli proved that RRTs will fail to return an optimal solution with probability 1 [11]. RRM combines the rapid exploration property of RRTs with the ability to optimize paths over a roadmap as in PRMs, allowing us to quickly find optimal solutions for single-query problems. As shown in Fig. 1, the RRM is represented using a graph in configuration space. In RRMs, exploration corresponds to sampling the free space and expanding the graph toward the

R. Alterovitz, S. Patil, and A. Derbakova are with Department of Computer Science, University of North Carolina at Chapel Hill, Chapel Hill, NC 27517, USA {ron,sachin,anya}@cs.unc.edu

sample in an RRT, tree-like manner. Refinement corresponds to connecting existing samples using additional edges to build a roadmap over which an optimal path can be selected.

Since collision detection in complex environments is a computationally expensive step in sampling-based planning, RRM strives to minimize collision checks and avoid any duplication. This is facilitated by separating exploration from refinement; there is more benefit in refining nodes along a path to the goal than nodes on explored sub-trees that cannot reach the goal. This is in contrast to prior sampling-based methods such as the original PRM [12] and to more recent optimal methods such as RRT* [11] that deterministically perform collision detections in all explored regions of the configuration space regardless of whether these regions can be part of the final optimal solution.

We first demonstrate the potential of RRM for a point robot moving on a plane. We then apply RRM to compute motion plans for a concentric tube robot, a device composed of nested nitinol tubes that can be controlled to follow curved paths through open air. These devices have the potential to assist physicians performing minimally invasive surgical procedures in constrained anatomical spaces such as the trachea and bronchi. By explicitly considering the trade-off between exploration and refinement, RRM can provide physicians with the flexibility to best utilize available computation time based on the task at hand.

II. Related Work

Randomized motion planning algorithms such as RRTs and PRMs have become increasingly popular in recent years [5], [15]. Single-query planners such as RRTs [16] focus purely on exploration and make no effort at refinement, a necessary step for computing optimal motion plans. Variants of the RRT algorithm have employed heuristics to optimize the quality of the solution during planning [25], [6] but do not provide optimality guarantees. Multi-query planners such as PRMs [12] explore (via sampling) and refine (via adding multiple edges per sample) at every iteration but result in unnecessary exploration of the entire configuration space and a large number of wasteful collision checks for single-query problems. Extensions such as Fuzzy PRM [19], Lazy PRM [2], and C-PRM [24] perform exploration and refinement as distinct phases of the planning process, but do not balance the two within a unified framework. RRM combines the benefits of RRTs and PRMs and allows balancing exploration and refinement.

Guided exploration strategies in the workspace and configuration space have been successfully used to bias future samples [7], [1], [9], [27], [4], [10]. Many of these strategies could be directly incorporated into the exploration iterations of RRMs. Rickert et al. [21] propose an exploring/exploiting tree (EET) to balance between exploration of the configuration space and exploiting the results of exploration using potential fields but do not consider path optimality. Many post-processing methods have been developed to improve the quality of jerky, unoptimized paths typically obtained from randomized planners [3], [13], [8], [20]. These approaches are tailored for a specific criterion and only refine the path within its homotopy class. In contrast, our approach computes, in the limit, an optimal path across homotopy classes without requiring a post-processing refinement step.

Karaman and Frazzoli [11] propose an extension of the RRT algorithm called RRT* to provide optimality guarantees. The RRT* algorithm constantly refines, resulting in unnecessary collision checks for pathways that have little or no chance of reaching the goal. Our approach can be thought of as a lazy variant of the RRT* algorithm that refines pathways starting from homotopic solutions that reach the goal to decrease the number of collision checks while still retaining optimality guarantees.

A different set of approaches that offer optimality guarantees apply graph search algorithms (such as A*) over a discretization of the configuration space [15], [17]. These methods only ensure optimality up to the discretization resolution, and the computational complexity grows exponentially with the dimensionality of the configuration-space.

III. Algorithm

A. Notation and Problem Definition

Let $\mathcal{C} \in \mathbb{R}^d$ be the configuration space of the robot and $\mathcal{C}_{\text{free}} \subseteq \mathcal{C}$ denote the subspace of the configuration space for which the robot is not in collision with an obstacle. Let $q \in \mathcal{C}$ denote a configuration of the robot. The RRM planner requires as input the start configuration of the robot, q_{init} and a set of goal configurations, $Q_{\text{goal}} \subseteq \mathcal{C}_{\text{free}}$.

Similar to motion planning algorithms like PRM and RRT, RRM requires as input the definition of the certain problem-specific functions. Given a set of configurations V and any two configurations $q_1, q_2 \in V$, `collision_free`(q_1, q_2) returns `false` if the path (computed by a local planner [5]) from q_1 to q_2 collides with an obstacle and `true` otherwise. The function `cost`(q_1, q_2) specifies the cost associated with moving between two configurations q_1 and q_2, which can equal control effort, Euclidean distance, or any problem-specific user-specified metric. The function `nearest_neighbor`(V, q) returns the nearest neighbor to configuration q using some distance metric and `nearest_neighbors`(V, q) returns a set of neighbors within some distance of q. We note that for computational efficiency, the distance metric used by the nearest neighbor functions does not necessarily have to equal the metric used by the `cost` function.

The objective of standard motion planning is to find a path $\Pi : (q_0, q_1, q_2, \ldots, q_{\text{end}})$ such that $q_0 = q_{\text{init}}$ and $q_{\text{end}} \in Q_{\text{goal}}$ and Π lies in $\mathcal{C}_{\text{free}}$. The objective of optimal motion planning is to find a feasible path such that the cost of the path is minimized, where the cost of a path is defined to be the summation or maximum of the costs of sequential pairs of configurations along the path Π.

The RRM algorithm also requires an additional parameter, w_{refine}, that weighs exploration versus refinement. Setting this parameter to 0 results in exploration only, which is equivalent to a standard RRT. Setting this parameter to 1 causes the RRM to refine immediately whenever possible,

Alg. 1 `RRM_plan`: Build the RRM data structure and compute a path.

Input:

q_{init}: initial configuration, Q_{goal}: goal region
d_{step}: maximum size of edge for RRT expansion
w_{refine}: weight of refinement relative to exploration, $w_{\text{refine}} \in [0, 1)$

Output:

P: Sequence of configurations defining a collision-free path from q_{init} to $q \in Q_{\text{goal}}$

counter $\leftarrow 0$
$V \leftarrow \{q_{\text{init}}\}$
prev$[q_{\text{init}}] \leftarrow$ nil
counter_add$[q_{\text{init}}] \leftarrow$ nil
counter_refine$[q_{\text{init}}] \leftarrow$ nil
$E \leftarrow \emptyset$; $U \leftarrow \emptyset$; $S \leftarrow \emptyset$
do
  $w \leftarrow$ uniform random number in range$[0, 1]$
  if $|U| = 0$ **or** $w \geq w_{\text{refine}}$
    `RRM_explore`()
  else
    `RRM_refine`()
  end if
until user stops process
$P \leftarrow$ shortest path in weighted directed graph $G = (V, E)$
return P

Alg. 2 `RRM_explore`: Expand a RRM by exploring.

Input:

Global variables $G = (V, E)$, prev, d_{step}, U, S, counter_add, counter_refine, and counter from `RRM_plan`

Output:

$G = (V, E)$, prev, U, counter_add, counter_refine, and counter are updated

$q_{\text{sample}} \leftarrow$ a randomly chosen configuration from C
$q_{\text{near}} \leftarrow$ `nearest_neighbor`(V, q_{sample})
$q_{\text{new}} \leftarrow$ the point along the straight line from q_{near} to q_{sample} that is at most distance d_{step} away from q_{near}
if `collision_free`$(q_{\text{near}}, q_{\text{new}})$
  $V \leftarrow V \cup \{q_{\text{new}}\}$
  $E \leftarrow E \cup \{(q_{\text{near}}, q_{\text{new}}, \texttt{cost}(q_{\text{near}}, q_{\text{new}})\}$
  prev$[q_{\text{new}}] \leftarrow q_{\text{near}}$
  if $q_{\text{new}} \in Q_{\text{goal}}$
    `RRM_mark`(q_{new})
    $Q_{\text{goal}} \leftarrow Q_{\text{goal}} \cup \{q_{\text{new}}\}$
  end if
  if $q_{\text{near}} \in S$
    `RRM_mark`(q_{new})
  end if
  counter_add$[q_{\text{new}}] \leftarrow$ counter $++$
  counter_refine$[q_{\text{new}}] \leftarrow \infty$
end if
return

which slows does the exploration process. As discussed in Sec. III-D, appropriately setting w_{refine} will guarantee that RRM will return the optimal plan with probability 1 as the number of iterations increases.

B. RRM Algorithm

The RRM R is defined by the tuple G, where $G = (V, E)$ is a weighted directed graph with vertices V and weighted edges E. A vertex represents a configuration as well as additional information that will be described below. The graph is directed because we do not assume that moving from q_i to q_j is equivalent to moving from q_j to q_i (e.g. moving forward may incur a different cost from moving in reverse).

We build the RRM using the algorithm `RRM_plan` defined in Alg. 1. Initially, the vertex set V contains only q_{init}. The algorithm also maintains two global lists: U and S. U is the set of vertices that should be refined but have not yet been refined. S is the set of vertices that have already been refined. The algorithm then enters a loop. If U is not empty, then the algorithm explores the configuration space with probability $1 - w_{\text{refine}}$ or refines the RRM with probability w_{refine}. The algorithm iterates until the user stops the process.

The exploration algorithm `RRM_explore`, defined in Alg. 2, proceeds similarly to an iteration of the original RRT algorithm [15]. The algorithm generates a sample configuration q_{sample}, which in our implementation is a uniform random sample from $\mathcal{C}$. We find the sample's nearest neighbor q_{near} in V. We then consider the edge that extends from q_{near} toward q_{sample} up to a distance of d_{step}, which is a user-specified constant. We define this edge as $(q_{\text{near}}, q_{\text{new}}, \texttt{cost}(q_{\text{near}}, q_{\text{new}}))$. If this edge is collision free, we add the new vertex q_{new} to V and the new edge to G.

If the newly added q_{new} is inside Q_{goal}, then we have found a new feasible path from q_{init} to the goal. This new feasible path may be in a new homotopic class, so we mark for refinement all vertices on that path by adding them to U. (We note that for computational efficiency we only need to backtrack from q_{new} until we reach the first vertex in S or U, adding vertices to U along the way.) We also add q_{new} to U if its edge connects it to a vertex that is in S since this new vertex is already adjacent to a refined vertex. This procedure is handled by Alg. 3.

The refinement algorithm `RRM_refine`, defined in Alg. 4, randomly selects a configuration q_{refine} from U to refine. The algorithm then finds the nearest neighbors of q_{refine} and saves them to the set Q_{near}. In our implementation, `nearest_neighbors` returns all vertices within distance d_{step} since we assume that d_{step} is the maximum distance edge for which we are willing to conduct a collision check.

Alg. 3 `RRM_mark`: Mark a path from a given configuration to q_{init} for refinement.

Input:
q: configuration
Global variables $G = (V, E)$, prev, U, and S from `RRM_plan`
Output:
Global variable U (the set of vertices waiting to be refined) is updated

```
1   while q ∉ S and q ∉ U and q ≠ nil
2       U ← U ∪ {q}
3       q ← prev[q]
4   end while
5   return
```

We alternatively can define a distance that decreases as the number of vertices increases, as in Karaman et al. [11]. For each configuration q_{near} in Q_{near}, the algorithm checks if the edge from q_{refine} to q_{near} is collision-free and adds the edge if possible. In doing this, we want to avoid duplicating collision checks. We only call `collision_free` if both the following conditions are met:

1) The vertex q_{near} was not already refined (i.e. $q_{\text{near}} \notin S$) or was refined before vertex q_{refine} was added. We handle the latter check in $O(1)$ computation time by introducing a global counter. The global counter is initialized at 0 and is incremented each time a vertex is added or refined. Using vectors `counter_add` and `counter_refine`, we maintain the appropriate counter values for each vertex.
2) The edge was not already checked for collisions during the exploration stage. The algorithm already performed a collision check from $\text{prev}[q]$ to q when adding q to the graph.

By only checking collisions for edges that meet both of the above criteria, we guarantee that each edge that should be checked for collision is examined exactly once. We repeat the above for the opposite direction edge from q_{near} to q_{refine}. After an edge is added, we also check if the vertices should be marked for refinement and added to set U using the same requirements as during `RRM_explore`.

The algorithm `RRM_plan` iterates and adds configurations and edges to the graph until the user stops the process. The algorithm then computes and returns the shortest path in the RRM graph using the user-specified cost metric.

C. Computational Complexity

Let k be the number of iterations that `RRM_plan` executes before computing the shortest path. We note that $|V| \leq k$ since at most k vertices can be added to the graph, corresponding to the case where we always explore and each exploration results in a collision-free edge being added to the graph. We also note $O(|E|) \leq O(k^2)$. In practice, $|E|$ will be dependent on the exact implementation of `nearest_neighbors`.

Let D be the computational complexity of detecting if a collision occurs as the robot moves between two configurations for a distance up to d_{step}. We note that D grows for environments with more complex obstacles and is dependent on the collision detection algorithm used. Although `RRM_refine` and `RRM_explore` make multiple calls to `RRM_mark`, no vertex is ever marked more than once. Thus, the total computational complexity of `RRM_mark` over the entire execution of `RRM_build` is capped at $O(|V|)$. Also, we can implement constant-time access to U and S by storing these sets as lists and including pointers to entries in those lists from each vertex in the graph. Hence, `RRM_explore` and `RRM_refine` called by `RRM_plan` have an asymptotic expected complexity of $O(|V|D)$ using brute-force nearest neighbor searching. If `nearest_neighbors` is restricted to balls of volume proportional to $\log|V|/|V|$ [11] and kd-trees or BBD-trees for nearest neighbor searching, then the

Alg. 4 `RRM_refine`: Expand a RRM by adding edges around a configuration.

Input:
Global variables $G = (V, E)$, Q_{goal}, prev, counter_refine, U, S, and counter from `RRM_plan`
Output:
Global variables $G = (V, E)$, U, S, counter_refine, and counter are updated

```
1   q_refine ← randomly chosen configuration from U
2   Q_near ← nearest_neighbors(V, q_refine)
3   for each q_near ∈ Q_near
4       if (q_near ∉ S or counter_refine[q_near]
        < counter_add[q_refine]) and
        (q_near ∉ Q_goal or q_refine ∉ Q_goal)
5           if prev[q_near] = q_refine
6               RRM_mark(q_near)
7           else if collision_free(q_refine, q_near)
8               E ← E ∪ {(q_refine, q_near,
                    cost(q_refine, q_near))}
9               RRM_mark(q_near)
10          end if
11          if prev[q_refine] = q_near
12              RRM_mark(q_near)
13          else if collision_free(q_near, q_refine)
14              E ← E ∪ {(q_near, q_refine,
                    cost(q_near, q_refine))}
15              RRM_mark(q_near)
16          end if
17      end if
18  end for
19  counter_refine[q_refine] ← counter + +
20  U ← U\{q_refine}
21  S ← S ∪ {q_refine}
22  return
```

complexity is $O(\log |V| \exp d + D \log |V|)$.

The search at the end of `RRM_plan` has computational complexity $O(|V| \log |V|)$ for Dijkstra's algorithm or other similar shortest path algorithms, although this is dominated by the prior k iterations. Hence, the algorithm `RRM_plan` has a total computational complexity of $O(k^2 D)$ for brute force nearest neighbor searching and an asymptotic expected complexity of $O(k \log k \exp d + kD \log k)$ when nearest neighbors are restricted to balls as specified above.

In comparison, the computational complexity of an RRT is $O(k^2 + kD)$ for brute-force nearest neighbor searching and $O(k \log k \exp d + kD)$ for kd-trees or BBD-trees. The only computational complexity overhead of RRM relative to RRT is the additional collision detections for the extra edges, which is required for optimal motion planning. The worst case complexity of RRM is equivalent to the worst case complexity of RRT* [11] when explicitly considering the cost of collision checks but not counting the linear complexity of propagating shortest paths during RRT* iterations that rewire the tree.

D. Discussion

RRM retains the desirable exploration rate of the RRT algorithm due to its RRT-like exploration. In addition, the cost of an RRM path converges to the optimal feasible cost with probability 1 under certain conditions as the number of iterations increases. This property is the result of RRM creating a connected graph with multiple pathways to any configuration. For $w_{\text{refine}} > 0.5$ and when searching for nearest neighbors in balls of radius d_{step}, as the number of uniform random samples approaches infinity the resulting RRM graph will be connected in the connected free space containing the start configuration. The optimal cost path will lie in this graph and will be discovered by the RRM algorithm as the number of iterations approaches infinity. A detailed analysis that considers obstacles in the configuration space as well as nearest neighbor balls that shrink with iteration count is provided by Karaman and Frazzoli [11]. Related methods have also been used for analyzing PRMs [14].

For convergence to optimality, we require that the set of vertices yet to be refined, U, does not grow at an unbounded rate. For $w_{\text{refine}} > 0.5$, the expected rate at which configurations are removed from U when $|U| > 0$ is greater than the expected rate at which configurations are added, thus guaranteeing convergence. For $w_{\text{refine}} \leq 0.5$, our results presented below look promising but more investigation is needed to assess optimality guarantees.

Just as with RRTs, the effectiveness of RRM for a particular problem is dependent on the parameter d_{step}. Low d_{step} can result in an excessive number of configurations being added to the graph, which slows the method. A large d_{step} can result in a large number of edges colliding with obstacles. As with RRTs, we can consider a variable d_{step} obtained by extending edges dynamically until collision [15].

Unlike prior methods which implicitly impose a weighting between exploration and refinement, RRM allows the user to select the weighting. For $w_{\text{refine}} = 0$, the RRM is equivalent to an RRT and only explores the configuration space. For $w_{\text{refine}} = 1$, the RRM is equivalent in the limit to RRT*, although RRM does not begin refinement until an initial solution is found in order to improve algorithm performance when little computation time is available. In future work, we will investigate automatically setting and tuning the weighting. In particular, we will investigate the impact of narrow passages, which require comparatively more exploration, on the optimal setting of w_{refine}. We will also consider approaches that compare the decrease in entropy (increase in information) [4] between recent exploration and refinement steps to determine which step is more likely to decrease entropy.

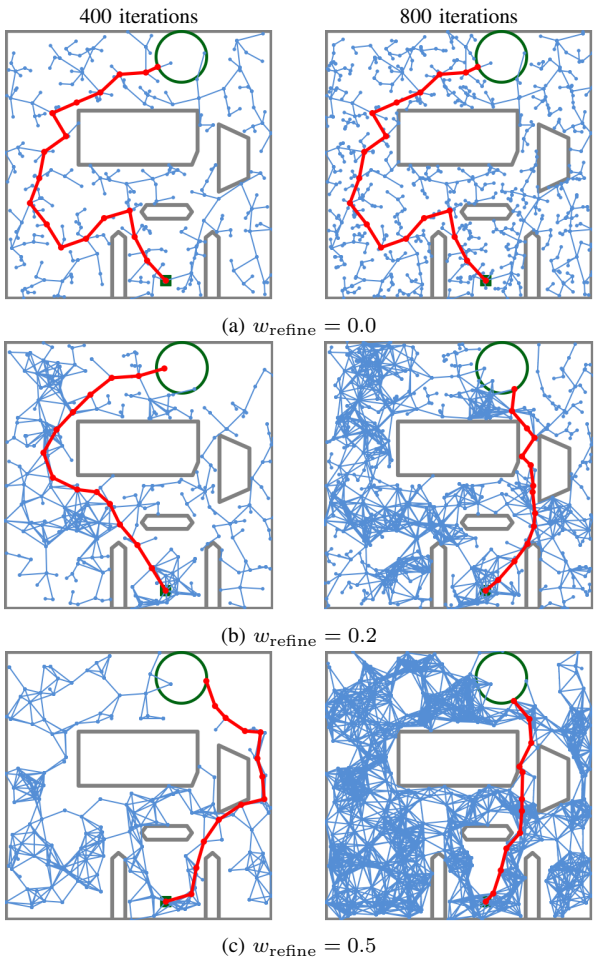

Fig. 2. (a) RRM for $w_{\text{refine}} = 0.0$, where the method is equivalent to an RRT and only explores the space. Although the space is well explored by 800 iterations, the resulting path is suboptimal due to the lack of refinement to take advantage of the exploration. (b) RRM for $w_{\text{refine}} = 0.2$. The method explores less of the space than the RRT, but the small amount of refinement finds a near-optimal solution by 800 iterations that is homotopic to the true optimal solution. (c) RRM for $w_{\text{refine}} = 0.5$, where exploration and refinement are balanced and the method converges toward the optimal solution.

IV. Implementation and Evaluation

We apply the RRM framework to two problems. The first problem, which we describe in Sec. IV-A, is to find a path for a point robot moving on a plane amongst obstacles. We then apply RRM in Sec. IV-B to compute motion plans

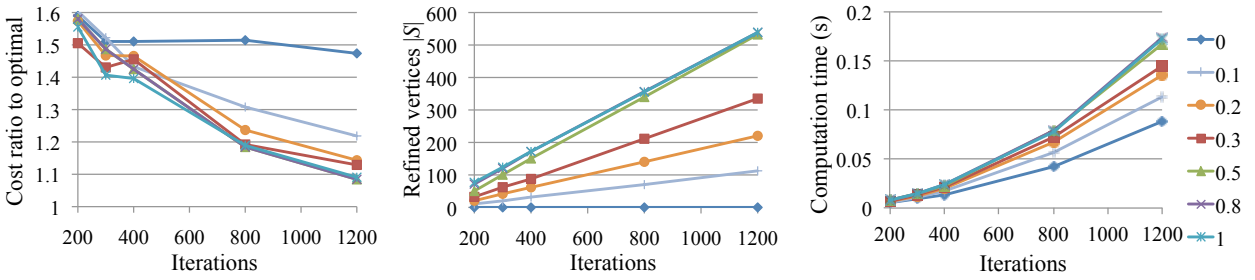

Fig. 3. For the point robot example, we show the ratio of the cost of the computed plan relative to ground-truth optimality for different values of w_{refine}, averaged over 100 runs. We also show the number of vertices refined, which is correlated to collision detection effort, and the total computation time for roadmap construction and computing shortest paths. By focusing refinement along known pathways to the goal, small amounts of refinement can result in close to optimal plans and require much less computation time.

for a concentric tube robot capable of following curved trajectories. All experiments were performed on a 2.2 GHz Intel Core 2 Duo laptop.

A. RRM for a Point Robot

We consider a holonomic point robot that moves on a plane. We define the workspace as a rectangle and define obstacles as polygons.

We execute the RRM algorithm for different weights of w_{refine} between 0 and 1 and show example runs in Fig. 2. In Fig. 3, we show the ratio of the cost of the computed plan relative to ground-truth optimality averaged over 100 runs. The results converge toward the optimal solution for low values of w_{refine}. Since RRM focuses refinement along known pathways to the goal, small amounts of refinement can result in close to optimal plans even for values of w_{refine} as low as 0.1, at the benefit of only spending time refining a small fraction of the sampled vertices.

We also show the number of vertices refined, which is correlated to collision detection effort, as well as computation times. For an equal number of iterations, increasing w_{refine} requires more computation time since the computational cost of a refinement operation is greater than an exploration operation. Combining the results in the cost ratio graph and the computation time graph suggests that refinement can be used sparingly (e.g. use a low value of w_{refine}) to obtain near optimal solutions at substantially lower computation cost than continual refinement as is done in RRT*.

B. RRM for a Concentric Tube Robot

To demonstrate the potential of RRM for higher dimensional configuration spaces, we apply RRM to a concentric tube robot. These needle-like robots are composed of nested nitinol tubes and can be controlled to follow curved paths through open air as well as soft tissues [26], [23], [18]. These devices have the potential to assist physicians in performing minimally invasive surgical procedures by maneuvering through constrained anatomical spaces to provide surgical access to clinical targets previously inaccessible using needle-like devices.

Each tube of the concentric tube robot, which consists of a straight transmission segment followed by a pre-bent segment, can be telescoped and axially rotated with respect to the other tubes. A device containing N tubes thus has $2N$ degrees of freedom. As the tubes rotate and translate within one another, the global shape of the device changes. We compute this shape (i.e. forward kinematics) using a model that considers the stiffness and torsional energy of the interacting tubes and minimizes energy [22].

Unlike bevel-tip steerable needles that have been well studied, the motion of concentric tube robots cannot be modeled with high accuracy solely using the tip pose since the entire shaft shape changes with each configuration change [22]. Prior work on motion planning for concentric tube robots has not considered these global shape changes [18].

As a proof-of-concept, we apply RRM to a $N = 3$-tube robot as shown in Fig. 4. We consider a tubular environment with protrusions, inspired by what a physician would encounter when performing targeted biopsies or localized radiation cancer treatment in the trachea or bronchi. We consider the objective of minimizing robot motion, which we quantify by the integral of the distance moved by evenly spaced points along the robot's shaft. We plan to consider other objectives, such as maximizing clearance from obstacles, in future work. We note that this single-query motion planning problem is not well suited for a PRM solution due to the highly constrained workspace; over 80% of configuration space samples are in collision due to contact with the cylinder, and hence are not feasible from the given start configuration.

We executed the RRM algorithm for 5000 iterations for $w_{\text{refine}} = 0$, 0.2, and 0.5, which required 33, 61, and 73 seconds, respectively. The objective values for $w_{\text{refine}} = 0.2$ and $w_{\text{refine}} = 0.5$ were both 32.1% better than the RRT equivalent $w_{\text{refine}} = 0$. For 10,000 iterations, the objective value for the RRT equivalent did not change and the values for $w_{\text{refine}} = 0.2$ and $w_{\text{refine}} = 0.5$ were 39.6% and 40.2% better than the RRT equivalent, respectively. Computation times in seconds increased to 86 for $w_{\text{refine}} = 0$, 166 for $w_{\text{refine}} = 0.2$, and 183 for $w_{\text{refine}} = 0.5$. We illustrate snapshots of a plan in Fig. 4.

V. Conclusion

Computing globally optimal motion plans requires exploring the configuration space to identify reachable free space regions as well as refining understanding of already explored regions to find better paths. We presented the rapidly-exploring roadmap (RRM), a new method for single-query

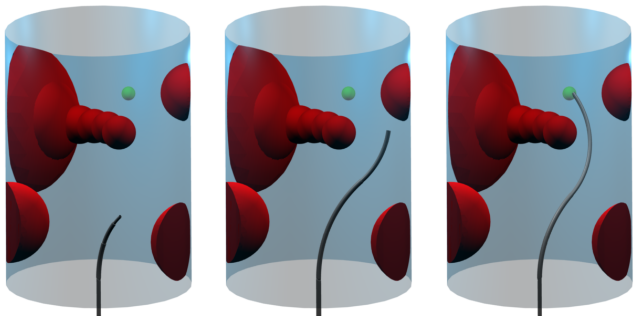

Fig. 4. Snapshots from an RRM motion plan for a concentric tube robot. The workspace is represented as a tubular environment with protrusions, as would arise in anatomical structures such as tracheae and bronchi.

optimal motion planning that allows the user to explicitly consider the trade-off between exploration and refinement. RRM initially explores the configuration space like an RRT. Once a path is found, RRM uses a user-specified parameter to weigh whether to (1) explore further to learn the free space, or (2) refine the explored space by adding edges to the current roadmap to enable finding higher quality paths. Exploration seeks to find new regions of free space while refinement seeks to find better paths within the explored regions.

We demonstrated the performance of RRM and the trade-off between exploration and refinement. In the first example, we applied RRM to plan motions for a point robot moving in a plane and show that properly weighing exploration and refinement can lead to more quickly finding optimal paths. In the second example, we applied RRM to plan motions for a concentric tube robot whose curved shape can be controlled to reach a target in a constrained space by inserting and rotating its constituent tubes.

In future work, we will investigate automatically setting and tuning w_{refine} based on workspace information and entropy reduction of exploration and refinement steps. We will also consider new sampling strategies based on ideas from the vast PRM/RRT literature. We also plan to investigate new applications of single-query optimal motion planning where RRM can help balance the trade-off between exploration and refinement.

VI. Acknowledgement

This work was supported in part by the US National Science Foundation under award IIS-0905344 and by the National Institutes of Health (NIH) under grant # R21EB011628. The authors thank Nate Dierk for creating visualizations of the concentric tube robots and Lisa Lyons for implementing concentric tube shape computation.

References

[1] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An obstacle-based PRM for 3D workspaces," in *Robotics: The Algorithmic Perspective: 1998 WAFR*, P. Agarwal *et al.*, Eds. Natick, MA: AK Peters, Ltd., 1998, pp. 156–168.

[2] R. Bohlin and L. E. Kavraki, "Path planning using lazy PRM," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2000, pp. 521–528.

[3] O. Brock and O. Khatib, "Elastic strips: A framework for motion generation in human environments," *Int. J. Robotics Research*, vol. 21, no. 2, pp. 1031–1052, 2002.

[4] B. Burns and O. Brock, "Toward optimal configuration space sampling," in *Proc. Robotics: Science and Systems*, Cambridge, MA, June 2005.

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

[6] D. Ferguson and A. Stentz, "Anytime RRTs," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006, pp. 5369–5375.

[7] R. Geraerts and M. Overmars, "Sampling and node adding in probabilistic roadmap planners," in *Journal of Robotics and Autonomous Systems (RAS)*, vol. 54, 2006, pp. 165–173.

[8] ——, "Creating high-quality paths for motion planning," in *Int. J. Robotics Research*, vol. 26, 2007, pp. 845–863.

[9] D. Hsu, T. Jiang, J. Reif, and Z. Sun, "The bridge test for sampling narrow passages with probabilistic roadmap planners," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2003, pp. 4420–4426.

[10] L. Jaillet, A. Yershova, S. M. LaValle, and T. Siméon, "Adaptive tuning of the sampling domain for dynamic-domain RRTs," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2005, pp. 2851–2856.

[11] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robotics: Science and Systems*, 2010.

[12] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high dimensional configuration spaces," *IEEE Trans. Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[13] J. Kim, R. Pearce, and N. M. Amato, "Extracting optimal paths from roadmaps for motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2003, pp. 2424–2429.

[14] A. L. Ladd and L. Kavraki, "Measure theoretic analysis of probabilistic path planning," *IEEE Trans. Robotics and Automation*, vol. 20, no. 2, pp. 229–242, 2004.

[15] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006.

[16] S. M. LaValle and J. James J. Kuffner, "Randomized kinodynamic planning," *Int. J. Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[17] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence Journal*, vol. 172, no. 14, pp. 1613–1643, 2008.

[18] L. A. Lyons, R. J. Webster III, and R. Alterovitz, "Planning active cannula configurations through tubular anatomy," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2010, pp. 2082–2087.

[19] C. L. Nielsen and L. E. Kavraki, "A two level fuzzy PRM for manipulation planning," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2000, pp. 1716–1721.

[20] B. Raveh, A. Enosh, and D. Halperin, "A little more, a lot better: Improving path quality by a simple path merging algorithm," *arxiv.org*, vol. abs/1001.2391, 2010.

[21] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, May 2008, pp. 2812–2817.

[22] D. C. Rucker and R. J. Webster III, "Parsimonious evaluation of concentric-tube continuum robot equilibrium conformation," *IEEE Trans. Biomedical Engineering*, vol. 56, no. 9, pp. 2308–2311, Sept. 2009.

[23] P. Sears and P. Dupont, "A steerable needle technology using curved concentric tubes," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2006, pp. 2850–2856.

[24] G. Song, S. Thomas, and N. M. Amato, "A general framework for PRM motion planning," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2003, pp. 4445–4450.

[25] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2003, pp. 1178–1183.

[26] R. J. Webster III, A. M. Okamura, and N. J. Cowan, "Toward active cannulas: Miniature snake-like surgical robots," in *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, 2006, pp. 2857–2863.

[27] Y. Yang and O. Brock, "Adapting the sampling distribution in PRM planners based on an approximated medial axis," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2004, pp. 4405–4410.