Lecture #18: *Sampling-based Motion Planning*

Scribes: *Amy Puente, Matthew Fontaine, Nishanth Hegde, Rey Pocius*

1 Introduction

In the previous lecture, we covered roadmap algorithms. By pre-processing the environment then running standard shortest path algorithms on the resulting networks, roadmap algorithms are usable for static environments where the state is know. However, roadmap algorithms do not respond to changes in a dynamic environment. To address this limitation, sampling-based motion planning algorithm were proposed. As an example of how sampling-based motion planning can handle dynamic configuration space, this lecture introduces Rapidly-Exploring Random Trees (RRT).

2 Rapidly-Exploring Random Trees

Though a famous sampling-based motion planner, RRT's are not used much in practice due to limitations we will discuss at the end of this section. To describe the inner workings of the algorithm, consider the configuration space visualized in Figure 1. A new point is sampled q_{rand}, then the nearest neighbor $q_{nearest}$ in the RRT is found. However, a path from $q_{nearest}$ to precisely q_{rand} may not be possible. To address this problem the function *Steer* is used to construct a new point near q_{rand} reachable from $q_{nearest}$. The path between the new point, q_{new}, and $q_{nearest}$ is checked for collisions. If no collisions exist, the new vertex q_{new} and edge $(q_{nearest}, q_{new})$ are added to the RRT. Refer to Algorithm 1 for details of the implementation of RRT.

Note that the RRT algorithm makes no optimality guarantee. To demonstrate RRT is not optimal, Figure 7 illustrates an RRT run for different values of n. Notice how the RRT results in jagged paths from the start node q_s to any of the goal points. This video[1] demonstrates RRT applied to a 12 dimensional configuration space of a real robot. Notice how the robot performs a dance on the way to reaching the goal state, an artifact of the path not being optimal.

[1] https://www.youtube.com/watch?v=vW74bC-Ygb4

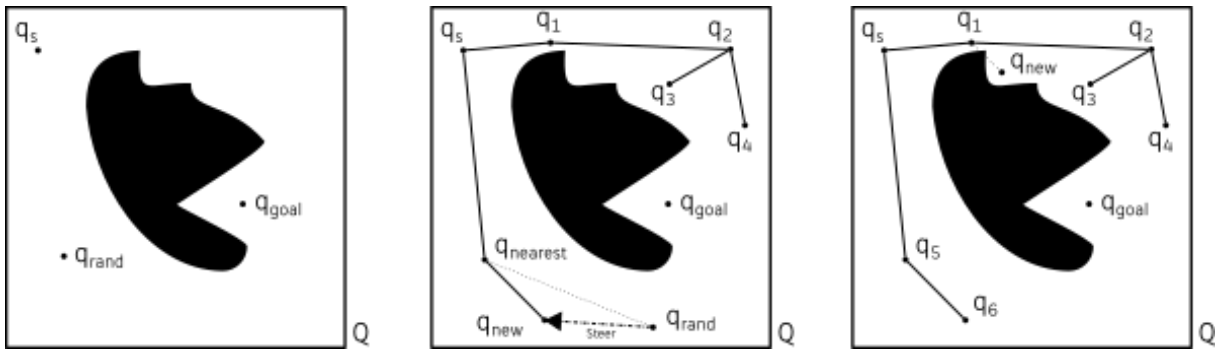
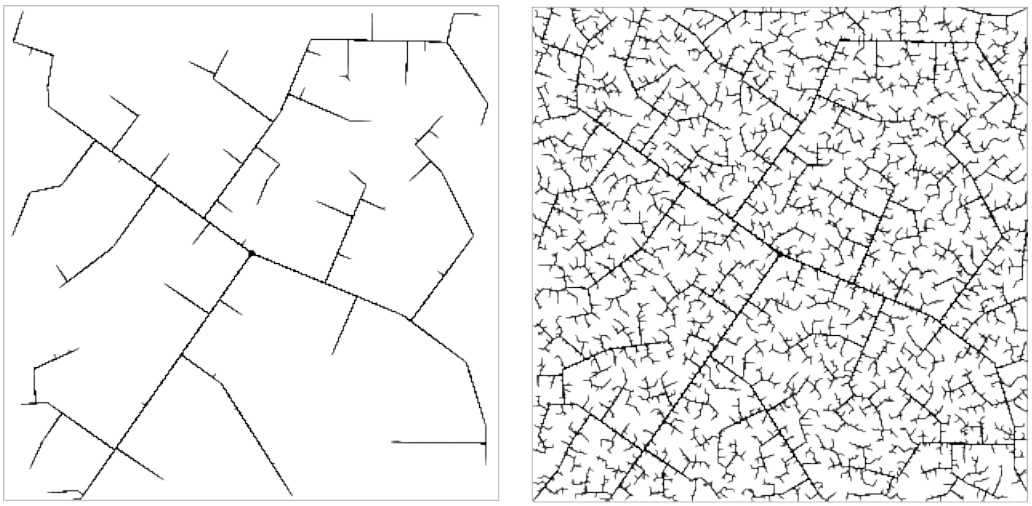


Figure 1: A demonstration of the sampling process of RRT. The left figure shows how new points are sampled from configuration space. The center figure demonstrates how a new point q_{rand} is sampled then steered to a new point using the nearest neighbor. The right figure shows how this new steered point is checked for collisions in configuration space. Samples that pass through obstacle space are thrown out.



45 iterations 2345 iterations

Figure 2: An example RRT run for differing values of n. Note the jagged paths found by the algorithm.

Algorithm 1: Compute an obstacle-free random tree in configuration space

RapidlyExploringRandomTree (q_s, n)
input : An initial configuration q_s and the number of vertices in the tree n.
output: A random tree with vertices V and edges E.
$V \leftarrow \{q_s\}$;
$E \leftarrow \{\}$;
for $i \leftarrow 1$ **to** n **do**
  $q_{rand} \leftarrow Sample()$;
  $q_{nearest} \leftarrow Nearest(V, E, q_{rand})$;
  $q_{new} \leftarrow Steer(q_{nearest}, q_{rand})$;
  if $ObstacleFree(q_{nearest}, q_{rand})$ **then**
    $V \leftarrow V \cup \{q_{new}\}$;
    $E \leftarrow E \cup \{(q_{nearest}, q_{new})\}$;
  end
end
return V, E;

3 Properties of RRT

3.1 Completeness

If a search algorithm is guaranteed to find the solution when it exists and reports failure when a solution does not exist, such an algorithm is called a complete algorithm. RRT and other similar sampling based algorithms are not complete. It is possible that there is a solution and the algorithm does not find it or that a solution does not exist and the algorithm does not terminate. Hence, we use probabilistic completeness as a relaxed notion of completeness.

3.2 Probabilistic Completeness

If an algorithm ALG is probabilistically complete, if for a robustly feasible motion planning problem defined by (c_{free}, q_S, q_G) where q_S is the start configuration, q_G is the goal configuration, and c_{free} is the free space, the probability of finding solution approaches 1 as we sample more points. That is,

$$\lim_{N \to \infty} P\left(\left\{V_n^{RRT} \cap q_G \neq \emptyset\right\}\right) = 1$$

which is essentially stating that as the number of sampled points approaches infinity, the probability that the goal configuration exists in our tree approaches 1.

We still need to define a robustly feasible planning problem. Consider the case of a planning problem where a solution exists. If we dilate the obstacles by a parameter δ and the solution still holds, such a problem is called robustly feasible. This takes care of some edge cases as shown in the figure below.

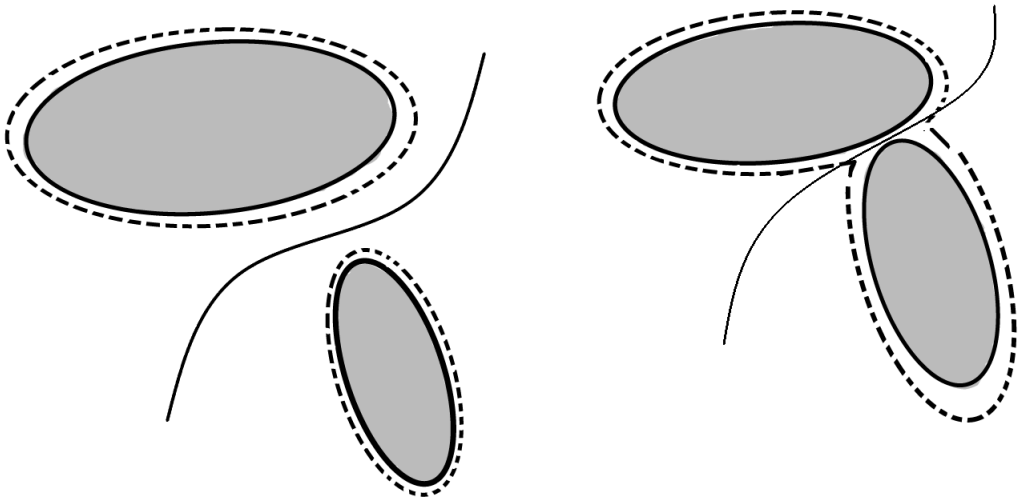


Figure 3: In this figure the grey objects are obstacles. The dotted boundaries represent the boundaries of the objects dilated by a small parameter δ. The line in between the obstacles represent the path of the robot. In the figure on the left, the path still holds when the obstacles are dilated. This is a robustly feasible problem. In the figure on the right, the dilated boundaries combine to form a single large obstacles thus invalidating the previous path. If no other path exists in this environment, this path planning problem is not robustly feasible.

RRT and its variants are probabilistically complete algorithms.

3.3 Asymptotic Optimality

RRTs are asymptotically sub-optimal. Consider a modification to the RRT algorithm. After finding a path to the destination, keep sampling more random points and improving the tree and thus finding multiple paths from source to goal state. When a new path is found, compare the cost of the old path to the cost of new path. Cost of the path could simply be the sum of lengths of the individual edges of the tree that constitute the path. Let Y_N^{RRT} be the cost of the best path from running the RRT algorithm after n iterations

of RRT. It can be shown that

$$P\left(\left\{\lim_{N\to\infty} Y_N^{RRT} > c^*\right\}\right) = 1$$

So in the general case, RRT almost surely converges to a sub-optimal path.

4 Extensions to RRT

RRT has been proven to not be asymptotically optimal. Work has been done to optimize RRT to overcome some of the shortfalls of the algorithm. Extensions to RRT such as RRT* have been proven to be asymptotically optimal. RRT* has an advantage over RRT by being more computationally efficient and guaranteeing shortest path when the number of nodes in the graph approaches infinity. RRT* keeps track of the distance between each node and its parent node and alters node connections based on cost within a child nodes neighborhood. Figure 4 shows simulation of both the RRT and RRT* algorithms[2].

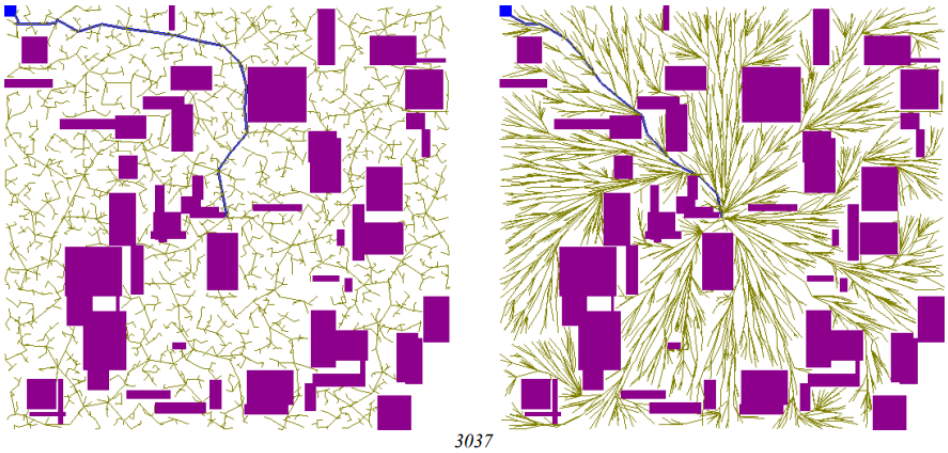


Figure 4: The graph on the left is simulation of RRT, the right graph is simulation of RRT*. Both graphs were generated after 3037 iterations of the algorithms.

[2]http://personal.cimat.mx:8181/~miguelvargas/Slides/Optimal%20Rapidly-exploring%20Random%20Trees.pdf

5 Manipulation Planning

5.1 Background

Many robotic systems are *hybrid systems* involving a combination of discrete and continuous variables. As a result, the state space of these systems is the Cartesian product of configuration space and a finite set known as the *mode space*. One application of such a hybrid system is manipulation planning. This section summarizes how the manipulation planning problem can be represented and solved. It's derived primarily from Section 7.3 of *Planning Algorithms*.

5.2 Hybrid System Model

Problems involving continuous and discrete variables can be modeled using a combination of components from discrete feasible planning and basic motion planning.

5.2.1 Components of a Hybrid System Model

1. A world, W, and a configuration space, C, are used.
2. A *mode space*, M, that is a nonempty, finite or countably infinite set of *modes*. Example modes could be, $m1$: where an object, o in the robot's world is grasped and $m2$: where o is not grasped.
3. An *obstacle region*, $O(m)$ for each $m \in M$.
4. A robot, $A(m)$, for each $m \in M$.
5. A *state space*, X, which is defined as the Cartesian product $X = C \times M$. A state is represented by $x = (q, m)$, with $q \in C$ and $m \in M$. Let $X_{obs} = \{(q, m) \in X \mid A(q, m) \cap O(m) \neq \emptyset\}$ and $X_{free} = X \backslash X_{obs}$.
6. Each state, x, has a finite *action space*, $U(x)$. U is the set of all possible actions in x.
7. A *mode transition function*, f_m which produces a mode $f(x, u) \in M$, for each $x \in X$ and $u \in U(x)$. This model assumes that the transition function does not produce any race conditions between modes. So, if q is fixed, the mode can change only once.
8. A *state transition function*, f, which is derived from f_m by changing the mode and keeping the configuration fixed. So, $f(x, u) = (q, f_m(x, u))$.
9. An initial state, $x_1 \in X_{free}$

10. A designated *goal region*, which is a set $X_G \in X_{free}$. A region is used instead of a point to allow for the specification of a goal configuration which doesn't depend on the final mode.

11. An algorithm must compute a (continuous) path $\tau : [0,1] \rightarrow X_{free}$ and an *action trajectory* $\sigma : [0,1] \rightarrow U$, such that $\tau(0) = x_1$ and $\tau(1) \in X_G$ if such a path exists. Otherwise, the algorithm must return that such a path and action sequence doesn't exist.

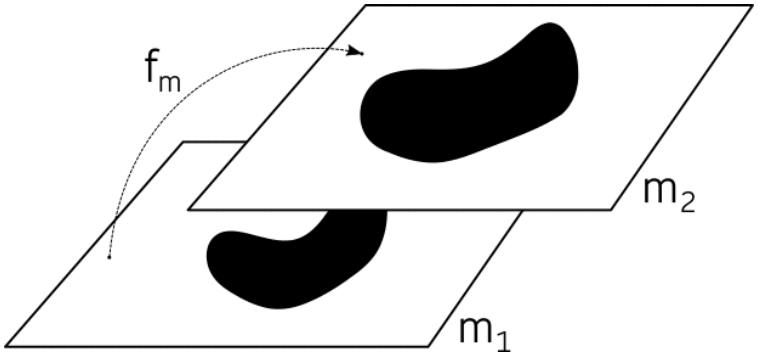


Figure 5: An example of two modes and a mode transition function.

Depending on the problem at hand, the obstacle region and robot may or may not depend on the mode. Mode changes depend on the action of the robot, and it's often assumed that a do nothing action is available. In such a case the robot would remain in its current mode. Additionally, a case could arise in which only one action is available to the robot, which would prevent the robot from having any control over the mode.

The solution to a manipulation planning problem must include both an action trajectory and a path, as without the action trajectory, the mode transitions would be unknown. Because σ indicates the mode transitions, he solution must specify both σ and τ.

5.3 Manipulation Planning

In a manipulation planning problem, a robot is considered a *manipulator* that interacts with a part. The robot can *grasp* the part and move it within the environment. Typically, a manipulation planning problem involves moving the part to a specific location in the environment, W. The following model simplifies much of the difficulties involved in manipulation planning, including grasping, stability, friction and other uncertainties. Instead, it focuses on the geometric issues involved.

5.3.1 Admissible Configurations

The robot A is the manipulator, which has C^a the *manipulator configuration space*. A part, P, will be a rigid body defined geometrically within the robot's environment. P can undergo rigid-body transformations, and thus will have its own configuration space, C^p. Then $q^p in C^p$ will denote a part configuration. The transformed part model will be denoted by $P(q^p)$.

The combined configuration space of the system, C, is the Cartesian product of C^a and C^p, $C = C^a \times C^p$. In C, each configuration $q \in C$ takes the form $q = (q^a, q^p)$. In this configuration space many configurations must be avoided, such as those in which the manipulator penetrates the part or in which the part penetrates an obstacle. Configurations in which the manipulator and obstacles collide are defined as $C^a_{obs} = \{(q^a, q^p) \in C | A(q^a) \cap O \neq \emptyset\}$.

Configurations in which the part collides with or penetrates obstacles should also be removed. However, configurations in which the part touches an obstacle, like the part sitting on a table, should be allowed. Let $C^p_{obs} = \{(q^n, q^p) \in C | int(P(q^p)) \cap O \neq \emptyset\}$ denote the set in which the interior of the part intersects with an object.

The configurations remaining in C guarantee that the robot and the part do not collide with O, but the interaction between A and P must also be regulated. Again, P and A should be allowed to touch, but one should not penetrate the other. Thus, let $C^{ap}_{obs} = \{(q^a, q^p) \in C | A(q^a) \cap int(P(q^p)) \neq \emptyset\}$. Removing these unallowable configurations leads to a set of admissible configurations, $C_{adm} = C \backslash (C^a_{obs} \cup C^p_{obs} \cup C^{ap}_{obs})$.

5.3.2 Stable and Grasped Configurations

There are two important subsets of C_{adm} used in manipulation planning problems. C^p_{sta} denotes the set of *stable part configurations*, in which P can safely rest without any forces applied by A. The configurations included in C^p_{sta} depends on the part's properties, like geometry, friction and mass distribution. For example, C^p_{sta} cannot include any configurations in which the part might fall. Then, let $C_{sta} \subseteq C_{adm}$ be *stable configurations*, defined by $C_{sta} = \{(q^a, q^p) \in C_{adm} | q^p \in C^p_{sta}\}$

The second important subset of C_{adm} are the *grasped configurations* in which the robot is grasping the part. This is denoted by $C_{gr} \subseteq C_{adm}$. IN every configuration $(q^a, q^p) \in C_g r$, the manipulator is touching the part.

In manipulation problems, $x \in C_{sta}$ or $x \in C_{gr}$ should always be true. As a result, $C_{free} = C_{sta} \cup C_{gr}$, which reflects the configurations in C_{adm} that can be used for planning.

There are two modes in M, the *transit mode* and the *transfer mode*. In the transit mode, the manipulator is not carrying the part, so q must be in C_{sta} (the part is in a stable position). In the transfer mode the manipulator is carrying the part, so q must be in C_{gr}. As a result, the mode can only change if $q \in C_{sta} \cap C_{gr}$. In every other configurations, the mode cannot change. Then the set of transition configurations, or the configurations

in which the mode can change, can be defined as $C_{tra} = C_{sta} \cap C_{gr}$.

The *state space* can then be defined as $X = C \times M$. Because there are two modes, there are two copies of C, one for each mode. X_{free}, X_{tra}, X_{sta} and X_{gr} can be derived from C_{free}, C_{tra}, C_{sta} and C_{gr} by ignoring the mode. For instance, $X_{tra} = \{(q, m) \in X | q \in C_{tra}\}$.

Finally, the manipulation planning problem can be defined. A problem specifies an *initial part configuration*, $q_{init} \in C_{sta}$ and a *goal part configuration*, $q^p_{goal} \in C_{sta}$. A path $\tau : [0, 1] \rightarrow X_{free}$ with $\tau(0) = q^p_{init}$ and $\tau(1) = q^p_{goal}$. An *action trajectory*, $\sigma : [0, 1] \rightarrow U$ must be specified to show the necessary mode changes when $\tau(s) \in X_{tra}$. A manipulation planning solution is an alternating sequence of *transit path* and *transfer paths*.

5.3.3 Manipulation Graph

The manipulation planning problem can be solved by forming a manipulation graph, G_m. Section 7. 3 of *Planning Algorithms* describes the process as follows.

1. Compute the connected components of X_{tra} to yield the vertices of G_m.
2. Compute the edges of G_m by applying ordinary planning methods to each pair of vertices in G_m.
3. Apply motion planning methods to connect the initial and goal states to every possible vertex of X_{tra} that can be reached without a mode transition.
4. Search G_m for a path connecting the initial and goal states. If one exists, then extract the corresponding solution as a sequence of transit and transfer paths, yielding σ.

5.4 Example 1

One example of a manipulation planning problem using discrete modes involves a robot with a start position to the left of a wall with a closed door and a goal position to the right of the wall with the door open.
In this example, the model has the following characteristics:

- $u = \{open, close\}$
- $x_s = \{q_s, closed\}$
- $x_g = \{q_g, open\}$
- $M = \{O(open), O(closed)\}$

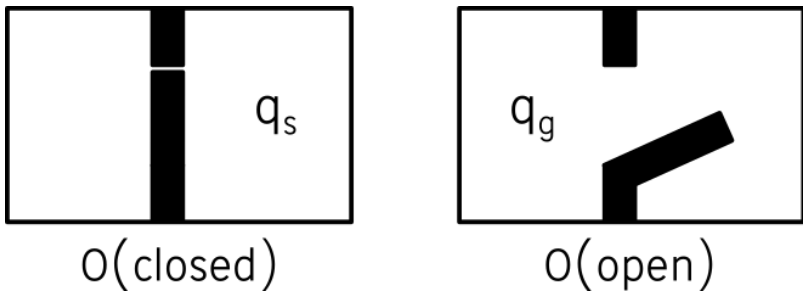


Figure 6: A system involving two modes, one with the obstacle open and another with the obstacle closed.

5.5 Example 2: Bead on a Wire

A second example involves a bead on a wire and a robot below it. The robot can move horizontally (*translate*), grasp and let go of the bead, and move the bead horizontally (*transfer*) while it's grasped.
The model has the following characteristics:

- $M = \{transit, transfer\}$
- $u(x) = \{grasp, ungrasp\}$, which can occur when $q_a = q_p$
- $f_m = ([q_a = q_p, transit], grasp)$

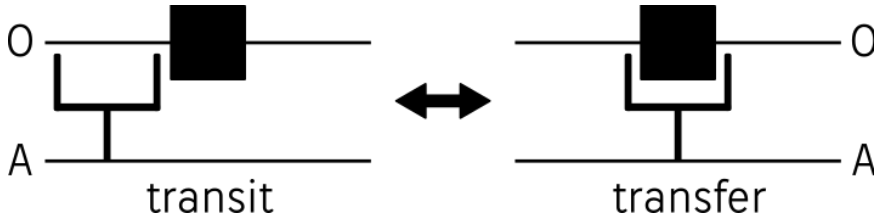


Figure 7: An example transit and transfer modes. In transit mode, the robot moves from left to right. In transfer mode, the part is grasped by the robot, and the robot can move it horizontally.

References

[1] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.