Date: 20th February 2017

Aditya Ghadiali - aghadial
Saurabh Nair - snnair
Flavian Pegado - ffp

16-662 Robot Autonomy:

Planning with Constraints, Kinodynamic Planning

1 Bi-Directional RRT and its Variants:

Idea: Grow trees from the start and goal towards a common sample point

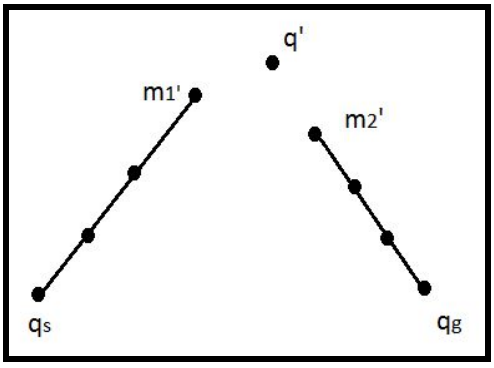

Figure 1.1 Bi-RRT Tree

Start Tree : $T_s \leftarrow \{q_s\}$; Goal Tree: $T_g \leftarrow \{q_g\}$

Pick a sample $\hat{q} \in C$ or $\hat{q} \in C_{\text{free}}$

for $T \in \{q_s, q_g\}$:

Loop N times:

1. Find nearest milestone in T : 'm'
2. Extend m towards $\hat{q}$ (take a small step and check iteratively until you fail)
3. Add m' and edge to T
4. If m' $\in$ *endgame* : return success

return failure

Note:

- *endgame* implies $d\{T_s, T_g\} <$ epsilon; where epsilon is the minimum unit distance
- The key point is, as you extend, you come close to the other tree
- Generally as the dimensionality of the space increases, likelihood of trees bumping into each other decreases.
- We make sure that the trees come close to each other. If not, we may only find solutions accidently.
- Declare success at $d(T_s, T_g) < \varepsilon$

1.1 Variant 1: Symmetric Bi-RRT

Idea: Add a milestone *at every* Δm (*smallest* Δm = *epsilon*)

Pros: There are more points to sample and you may find more solutions

Cons: Takes more time and is computationally expensive

1.2 Variant2: Asymmetric Bi-RRT

Idea: Grow the second tree towards *m1'* instead of going to the sample

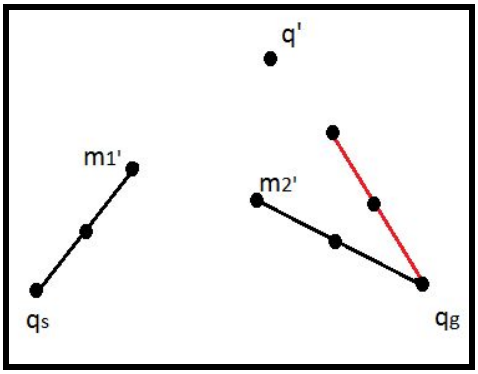

Figure 1.2 Grow the second tree towards m1' instead of the sample point (ie, follow the black colored trajectory instead of the red one)

Pros: If the forward tree can't reach a point (potentially an obstacle), we don't need to go there for the second tree.

Cons: This is a greedy approach, so we might end up with a sub-optimal solution ie there is unexplored region that might be a better solution.

Note:

- In general, asymmetric approach works better than symmetric Bi-RRT, however you need to be cautious while preferring one over the other.
- Also called Asymmetric BDRRT

1.3 Variant 3

Idea: Only extend k milestones towards the sample

Analogous to the breadth first search algorithm.

Pros: Favors rapid change in direction, with no predisposition to any particular direction.

The time spent in longer searches that return failure can be saved. Also, since there are more choices when picking the shortest path, the quality of the solution may be better.

Cons: If there exists a simple solution, we will miss that and end up with a complex solution.

Note:

- In general, smaller milestones will give better quality solutions.
- All theorems for RRT can be proved only for Symmetric variants of the algorithm.
- If the problem is small, a small value of 'k' would work. Same goes for larger problems, where a higher value of 'k' would perform better.

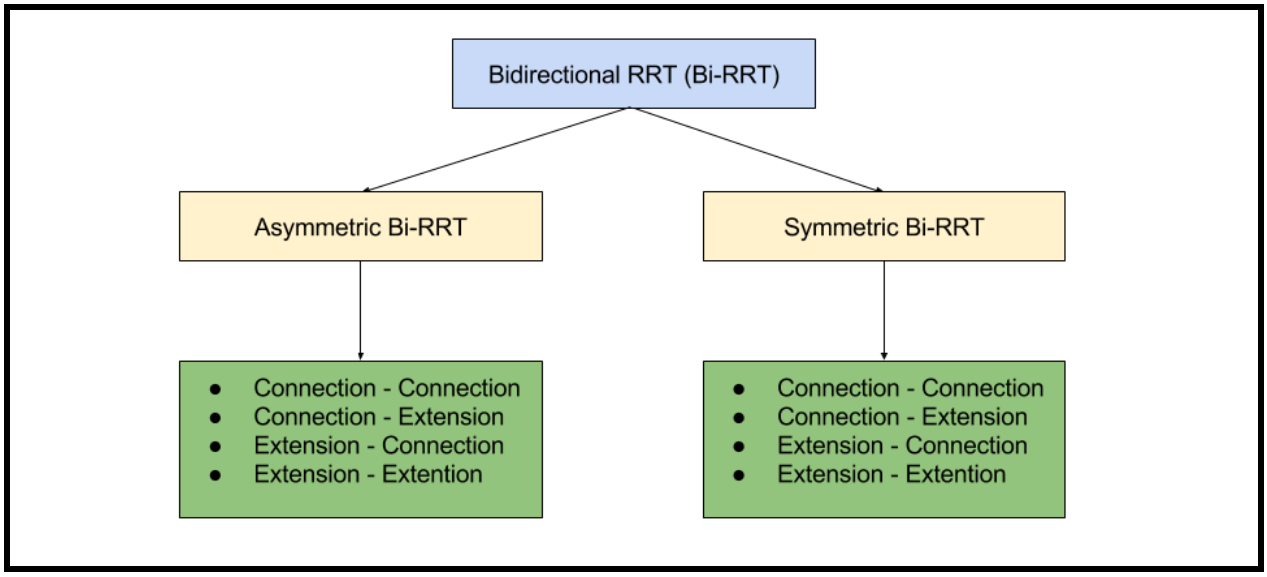

Figure 1.3: Eight variants of Bi-Directional RRT

2 Kinodynamic Motion Planning

Idea: An approximate solution is developed using a shooting method

Central to planning so far was the assumption that we have access to a simple planner B_s such that $B_s(q_1,q_2)$ is either success or failure (1 or 0).

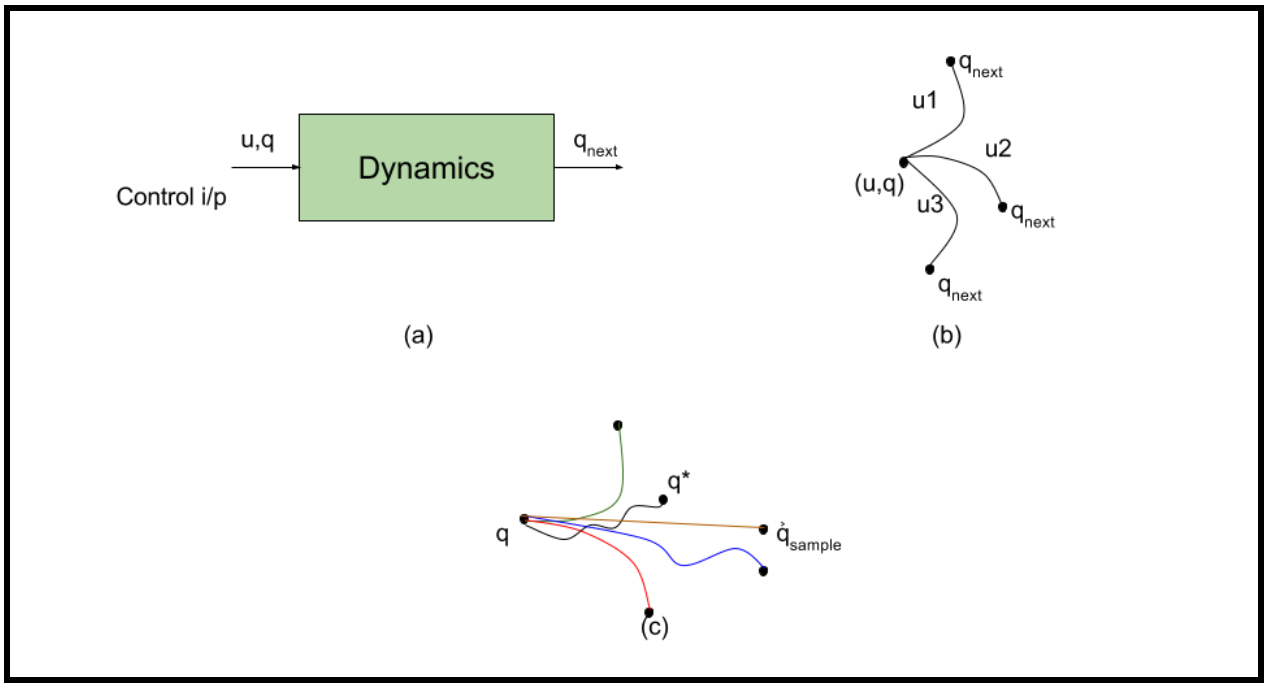

Figure 2.1: (a) Sample control system black-box; (b) Potential next states the system can be in depending on the control i/p '*u*'; (c) Shooting method

Linear Interpolation does not take into account system constraints/physical limits of the robot. So far, we have always assumed moving in a straight line between two points is possible; however, this does not take into account system constraints. Some examples of such systems are listed below:

DOFs cannot move in straight lines all the time in configuration space.

E.g:

- A car cannot move sideways, even though there are no obstacles blocking its motion. Similarly, bicycles, skateboards and other such systems are constrained by system properties rather than obstacles.
- An end effector that cannot pull an object but only push already has constraints to manipulate a mug on a table.

These constraints are called Kinodynamic constraints or nonholonomic constraints.

2.1 Shooting Method:

- Pick a time T
- Loop M times
    1. Sample ‘u’ (u = input to the black box control system)
    2. Roll out the trajectory
- Pick the control input that reached closest to the goal point.

Notes:

- There are no theoretical guarantees for Kinodynamic Motion Planning.
- Time-reversibility is important; if we make use of two trees extending towards each other, we must be able to move backwards in time. If the system is not time-reversible, use unidirectional RRT (only one tree, use the goal as a sample with some probability ‘p’).
- In a graph based method, every point must connect to every other point directly or indirectly.
- Chattering due to external factors is one problem faced with this method.
- In practice, this method delivers reasonable results.

2.2 How to sample?

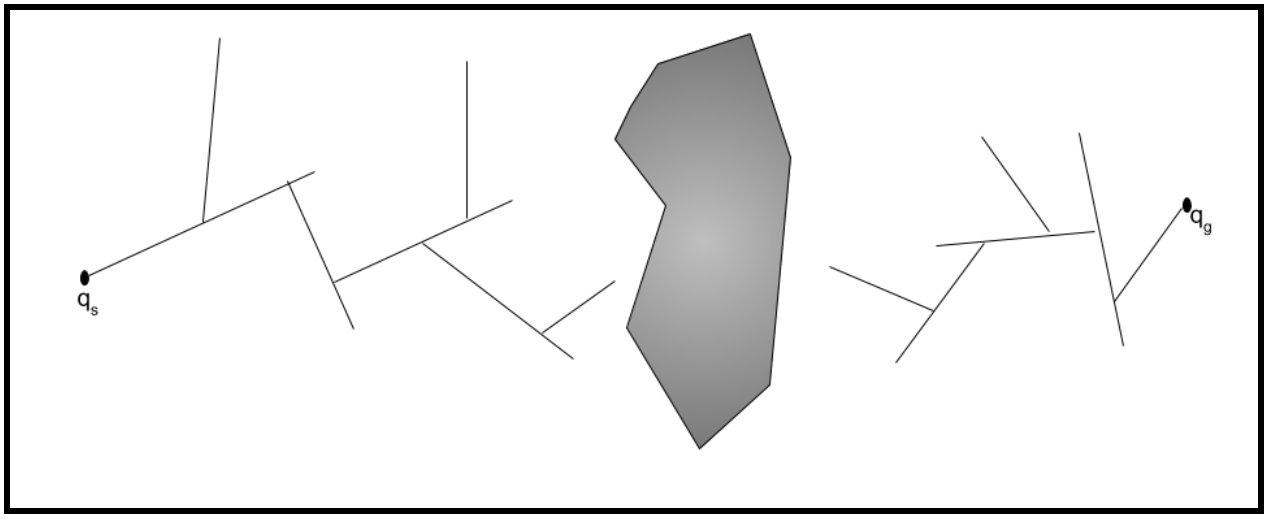

Figure 2.2: Growing a tree around an obstacle

Strategies:

1. Sample in C_{free} <- RRT
2. Expansive Space Tree <- EST
    a. Sample a milestone with P(m) α 1/(# of nearby milestones)

b. Sample a random time T & random control ‘u’
c. Extend from ‘m’ with ‘u’ from T

Note: Hence we are just sampling a milestone instead of a random point in space.

3 Planning with dynamic obstacles:

Idea: Add a dimension for time, i.e. plan in continuous time space.

When the obstacles change/move with time, we can extend planning algorithms like RRT by adding a dimension for time. The important consideration when planning in continuous time space is to maintain time consistency: only move forward in time.

4 Path Shortening:

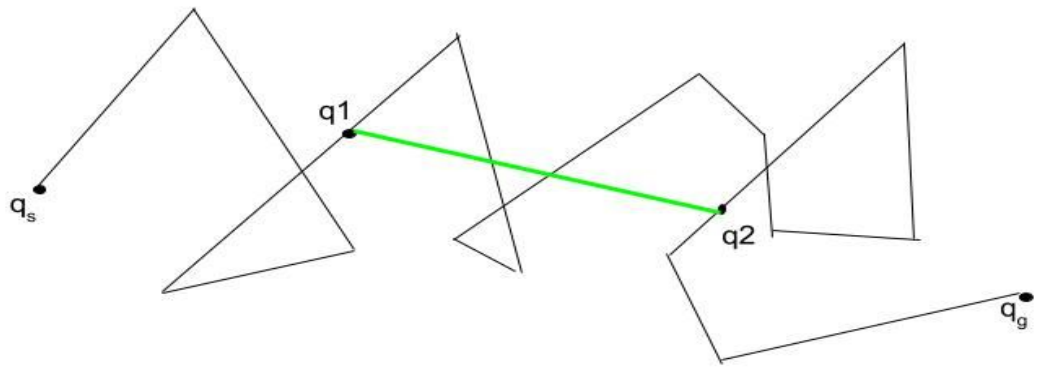

Figure 4: Path shortening between two randomly sampled points in the trajectory

Pseudo code:

- Till time elapsed < Allocated time for the process
    1. Sample two points in the trajectory & try connecting the points.
    2. If failed, sample again and try shortening.

Note:

1. We cannot shorten the path in Kinodynamic motion planning due to the two point boundary problem.
2. Path shorten in a kinodynamic system using the branch and bound method(Iterative search).