Single-Query Planners for Nonholonomic Systems

Scribe Notes for March 16, 2016

16-662: Robot Autonomy

Scribes:
Cole Gulino, Rohan Thakker, Feroze Naina, Bikramjob Singh Hanzra
I. Simple Planners for a Single-Query Planner

A simple planner has the form of $B_s(q_1, q_2)$.

This planner takes in two points within C-space and solves the two-point boundary value problem.

- The two point boundary problem is a differential equation with a set of additional constraints known as boundary conditions [1].
- A planner that solves a two-point boundary value problem is a solution to the differential equations that also solves the boundary constraints [1].

Figure 1 shows an example of the simple planner:

![Simple Planner](image)

Figure 1) Simple Planner

II. Kinodynamic Planning

- Holonomic constraints can be expressed in the form:
  $$f(q_1, q_2, q_3, \ldots, q_n, t) = 0$$

Where $q_1, q_2, \ldots, q_n$ are the states of the system. Solving the two-point boundary value problem is easy for holonomic systems. These are the systems that we have seen before.

- Non-holonomic constraints cannot be expressed in the above form. For example:
  $$f(q, \dot{q}, t) = 0$$

The above equation cannot be integrated to obtain the holonomic form. However, the two-point boundary value problem is much more difficult to solve for systems with non-holonomic constraints eg. cars, cycles, etc.
RRTs were invented to solve the classes of problems which do not have a close form solution to the two point boundary value problem. This problem is known as kinodynamic planning.

States: $s \in S$
Controls: $u \in U$
Model/Forward Model/Plant: $\dot{s} = f(s, u)$

- Known
- Nonlinear
- This function can be arbitrarily complex. Blimp’s could depend on air viscosity. Car’s could depend on wheel radius.

Want to find a path $s_1$ to $s_2$ s.t. $\dot{s} = f(s, u)$
- Optimization problem

Key idea that led to RRTs: What if you solve the two-point boundary value problem approximately?
- Problem: none of the edges will connect properly

Solution is the shooting method [2].
- The shooting method is a method of solving the two-point boundary value problem by reducing it to the initial-value problem
- The method involves randomly sampling in the controls space and choosing the control that brings you closest to the desired solution
- Can change shooting direction best on error from goal

![Figure 2) Shooting Method](image)

Shooting Method Algorithm:
1. Pick a time: $T$
2. Run $N$ times:
   a. Sample function: $u \in U(s)$

   ![Graph](image)

   Compute: $\tau_i \in [0, T]$ s.t.
   
   $\dot{\tau}_i(t) = f(\tau_i(t), u(t))$

   Solve: $s^* = \arg\min_{\tau \in [\tau_0...\tau_N], t \in [0,T]} f(s_2, \tau_i(t))$

   Can introduce bias in shooting method:
   
   - If you get the heuristic for the bias wrong, the algorithm will longer to run.

   Coming up a distance metric for kino-dynamics systems is really hard. For example, consider a 2 wheeled robot with differential drive. Even a point which is epsilon distance to the right of the car requires a parallel parking maneuver because of the nonholonomic constraints.

   In the above algorithm, we assuming that it is easier to solve the 2 point boundary value problem for points that are closer to each other than points that are further away. For example, consider the stabilization problem of a cart-pole in the inverted configuration. We can linearize the system about the inverted configuration and develop an LQR based controller. This controller will stabilize the system for small deviations away from the inverted configurations. Hence, the objective of our planner should be to get the pendulum close to the inverted configuration such that it lies in the region of attraction of the linear controller.

   LQR Trees algorithm is an extension of this idea, where a linear controller is developed at each node using a time varying LQR formulation. The region of attraction of this controller can be thought of as a funnel, as shown in the figure below. Our objective is to ensure that exit of each funnel lies in the entry of the next funnel. Refer to [11] for further details.
Rapidly-Exploring Random Trees

Uni-Directional RRTs

The tree data structure never needs to close loops. Compromises on optimality since multiple paths to the same points are not possible

Bi-Directional RRTs

- Would have to do this to be able to do a backward tree
- Would have to simulate dynamics backward in time
- Often impossible to invert a dynamic system backwards
  - Eg: Pushing (there is no unique solution)
You want to sample the goal with some probability and shoot towards it to increase time to find a solution

- Want to get arbitrarily close enough to goal depending on how accurate your system needs to be.

Voronoi property [3] that a tree will expand toward unexplored regions of the tree does not hold for kinodynamic systems due to the inaccuracies of the shooting method.

III. Bi-Directional Single-Query Planner

\[ T_s \leftarrow \{ q_s \} \qquad T_g \leftarrow \{ q_g \} \]

Choose a sample: \( q_s \in C_{free} \)

For: \( T \in \{ T_s, T_g \} \)

- Find nearest milestone in \( T \)
- \( m' \leftarrow \) Extend \( m \) towards \( q_s \)
- Add \( m' \) and Edge to \( T \)
- If \( m' \in \) Endgame, success

Loop \( N \) times

Return Failure
Easier to solve if they are very close together
- Good idea is to add multiple points interpolated along the edge as vertices in order to increase the ability for the trees to connect

IV. DARPA Steered Car

Example of a non-holonomic system [5].

![Figure 5) Example of a Steered Car](image)

Idea is to sample the control space of your car and then cache the trajectories. This allows you to pull feasible trajectories from your database and chain them using a lattice planner [6].
- Does not work if your system is not ego-centric (depends on where you are only)
- Works only for uni-directional planning

V. Algorithms for Picking a Sample

Rapidly-Exploring Random Trees

Pick uniformly in $C_{space}$. 
Expansive Space Tree (EST)

Algorithm:
1. Sample a milestone with probability: \( p(m) \propto 1/\text{number of nearest neighbors} \)
2. Sample a random time: \( T \in [T, T_{max}] \)
3. Sample a random control: \( u \in U \)
4. Extend from \( m \) with \( u \)

Essentially a breadth-first search
- Solution quality is better
- Much slower than the RRT

Favored much less than the RRT.
- The solution quality from the EST is still not good, and this small reward does not make up for the much slower convergence time.
VI. Forms of Bi-RRT

Extend v.s. Connect

Asymmetric v.s. Symmetric

Figure 7) Extend v.s. Connect

Figure 8) Asymmetric v.s. Symmetric
VII. Expansions on the RRT

Here are some sources on expansions of the RRT:

- Anytime RRT [4]
  - Attempts to improve solution quality by running numerous RRTs
- RRT* [7]
  - Provides provable asymptotic optimality guarantees
- Informed RRT* [8]
  - Uses an admissible ellipsoidal sampling heuristic in order to improve on the convergence rates of RRT*
- BIT* [9]
  - BIT* is an extension of Informed RRT* and LPA* [10] that uses incremental search techniques on increasingly dense random graphs
  - The algorithm works by improving the solution of a “batch” of samples by densely expanding the tree in ways that decrease cost to goal.
  - It uses LPA*’s method of updating edges after a change in graph in order to reuse parts of the tree it has already expanded.
  - It aims to balance the benefits of heuristically ordered search (A*, LPA*, etc.) with the anytime performance and asymptotic quality in a continuous space (RRT*).

Figure 9 shows performance of the methods mentioned above (taken from [9])

![Figure 9) Performance of Algorithms Expanding on the RRT](image-url)
Resources

[1] Boundary Value Problem
[2] Shooting Method
[3] Voronoi Diagrams
[4] Anytime RRTs by Dave Ferguson and Anthony Stentz
[6] Search Based Planning with Motion Primitives
[8] Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic by Gammel and Srinivasa
[10] Lifelong Planning A* by Koeing, Likhachev, and Furcy