MRdRRT

Tuesday, March 7, 2017 6:00 AM

Algorithm 1 dRRT_PLANNER (s, t)

```
1: T.init(s)
2: loop
3:     EXPAND(T)
4:     Π ← CONNECT_TO_TARGET(T, t)
5:     if not_empty(Π) then
6:         return RETRIEVE_PATH(T, Π)
```

- T: tree
- s : start configuration (q1, q2, ..., qm)
- t : goal/target configuration (q1, q2, ..., qm)

Assumptions

- We have already constructed a PRM for an individual robot in the space.
- All of the robots are the same (or have the same size, kinematic constraints, etc.)

Tree representation

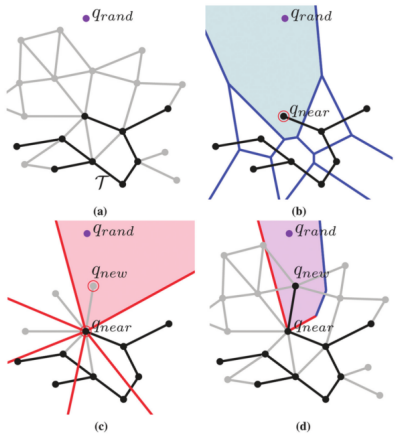
- Graph is represented implicitly, but that doesn't really matter for tree. Just needs to store configurations (returned from expand) as nodes. Most of the work/reasoning is done in EXPAND.
- Nodes will include:
  - Configuration
  - Pointer to its parent / where it came from
- Operations we need:
  - Add vertex
  - Add edge (store pointer to parent, too)
  - Find nearest neighbor

Retrieve path

- Like in normal RRT, each node in T will store a pointer to its parent / where it came from. We can just work backward from the goal configuration to reconstruct the path.

Algorithm 2 EXPAND $(\mathcal{T})$

```
1: for i = 1 → N do
2:     q_rand ← RANDOM_SAMPLE()
3:     q_near ← NEAREST_NEIGHBOR(T, q_rand)
4:     q_new ← O_D(q_near, q_rand)
5:     if q_new ∉ T then
6:         T.add_vertex(q_new)
7:         T.add_edge(q_near, q_new)
```



Oren used:

- PQP for collision detection
- FLANN (Fast Library for Approximate Nearest Neighbors) for queries
- Metrics, sampling, interpolation in 3D environments following guidelines in:
  - Kuffner J (2004) Effective sampling and distance metrics for 3D rigid body path planning.

Random sample

- Sample random points in individual configuration spaces of robots, then combine then into:
  - $q_{rand} = (q_{rand1}, q_{rand2}, \dots q_{randm})$

Nearest Neighbor

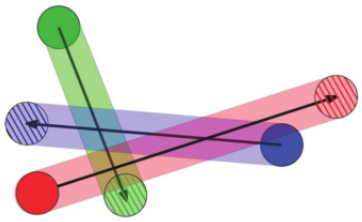
- Returns nearest neighbor to q_rand in current tree
  - Metric for "nearest" is Euclidean distance (see 3.2 of paper)
    - Calculate Euclidean distance for each robot individually, then sum?
    - How to use FLANN to efficiently do this?
      - How do we put composite configurations into k-d tree? - Store as 2N state vector

Direction Oracle O_D (q_{near}, q_{rand})

- Returns new configuration q_{new}
- Try to connect q_{near} and q_{rand}, which can be denoted C and C'
  - Find the most suitable neighbor for every individual robot and combine the m single-robot neighbors into a candidate neighbor for C
    - Find neighbors for individual robots by just traversing all neighbors in roadmap
  - If (C,C') is a valid edge, then return C'. Else, return NULL
    - (C, C') is a valid edge if no robot-robot collision occurs.
      - Collision along paths of each robot
      - We use local connector for every edge
        - Have to save ordering for each edge?

Algorithm 3 CONNECT_TO_TARGET$(\mathcal{T}, t,)$

```
1: for q ∈ NEAREST_NEIGHBORS(T, t, K) do
2:     Π ← LOCAL_CONNECTOR(q, t)
3:     if not_empty(Π) then
4:         return Π
5: return ∅
```



This step is equivalent to checking if the expanded point is within epsilon of the goal configuration, in normal RRT. In both continuous space and discrete composite space, it's highly unlikely that you'll end up sampling or expanding to the exact goal configuration. Because of this, you just want to try to expand "close enough" and then check if you can get to the goal from there.

Local connector

- Just do naïve local connector first, where you move the robots at constant "phase" along their paths and check for collisions.
- "The connector attempts to find an ordering of the robots such that robot i does not leave its start position on πi until robots with higher priority reached their target positions on their respective path, and of course that it also avoids collisions. When these robots reach their destination, robot i moves along πi from πi(0) to πi(1). During the movement of this robot the other robots stay put."
- "The priorities are assigned according to the following rule: if moving robot i along πi causes a collision with robot j that is placed in vj then robot i should move after robot j. Similarly, if i collides with robot j that is placed in vj? then robot i should move before robot j. This prioritization induces a directed graph I. In case this graph is acyclic we

Solovey, Kiril, Oren Salzman, and Dan Halperin. "Finding a needle in an exponential haystack: Discrete RRT for exploration of implicit roadmaps in multi-robot motion planning." *Algorithmic Foundations of Robotics XI*. Springer International Publishing, 2015. 591-607.

Other considerations

PRM representation

- OMPL has a PRM class
  - http://ompl.kavrakilab.org/geometricPlanningSE3.html

generate a solution according to the prioritization of the robots. Otherwise, we report failure."

- This is basically topological sort? https://en.wikipedia.org/wiki/Topological_sorting
- Or does this algorithm actually allow multiple robots to move at the same time, as long as they aren't constrained wrt each other?

Observation 4. Execution sequence S is valid if, for all $i \in [1, k]$, robot $R_i \in S$ can move from its start to its goal without collisions, even when the *goal* configurations of robots in $\{R_1, \ldots, R_{i-1}\}$ and the *start* configurations of robots in $\{R_{i+1}, \ldots, R_k\}$ have been added to the obstacles.

Algorithm 1 $\mathcal{P}(R)$

Let $s, g \in RM(R)$ be the start and goal configuration of R.
for all vertices x in $RM(R)$ **do**
  $\mathcal{P}(x) \leftarrow \bot$
$\mathcal{P}(s) \leftarrow \bigwedge_{r_i, r_j \in R} r_i \sim r_j$
$Q \leftarrow \{s\}$
while not (priority) queue Q is empty **do**
  Pop front vertex x from Q.
  if not $\mathcal{P}(x) \Rightarrow \mathcal{P}(g)$ **then**
    for all edges (x, x') in $RM(R)$ **do**
      $C \leftarrow \mathcal{P}(x)$
      for all robots $r_i \notin R$ **do**
        if robot r_i configured at s_i "blocks" edge (x, x') **then**
          $C \leftarrow C \wedge r_i \prec R$
        if robot r_i configured at g_i "blocks" edge (x, x') **then**
          $C \leftarrow C \wedge R \prec r_i$
      if not $C \Rightarrow \mathcal{P}(x')$ **then**
        $\mathcal{P}(x') \leftarrow \mathcal{P}(x') \vee C$
        if $x' \notin Q$ **then**
          $Q \leftarrow Q \cup \{x'\}$
return $\mathcal{P}(g)$

  - http://ompl.kavrakilab.org/pathVisualization.html
  - http://ompl.kavrakilab.org/stateValidation.html
  - http://ompl.kavrakilab.org/classompl_1_1geometric_1_1PRM.html
- Represent as configurations with corresponding node IDs?
- What are the advantages of node IDs?
  - Abstraction
  - Lower memory usage / compact data structure for nodes
    - one int vs. m floats
- Disadvantages of node IDs?
  - Have to repeatedly convert between node ID and configuration
    - But this is an O(1) lookup

Environment representation

Does this only matter for constructing initial PRM?

- Represent environment in octree or k-d tree
  - https://www.reddit.com/r/robotics/comments/2yge6h/collision_distance_checking_for_tion_planning/