Robot Motion Planning: Algorithms for Sampling Based Motion Planning

Jin Seob Kim, Ph.D.
Senior Lecturer, ME Dept., LCSR, JHU

1 Probabilistic Road Map (PRM)

1.1 Procedure

1. Initialize the graph with an empty one: $G = (V, E)$ is empty.
2. Generate n number of collision-free random configurations.
3. For every node $q \in V$, choose a set N_q (k closest neighbors to q) by using the distance metric.
4. For each $q' \in N_q$, call the local planner Δ to generate a path (q, q'). If the path is collision-free, the edge (q, q') is added to the roadmap. This is the end of roadmap build-up.
5. After obtaining G, using a graph search algorithm, reconstruct the shortest path.

1.2 Pseudocode

This is from [1].

function G = build_PRM(n, k)
  $V \doteq \emptyset$
  $E \doteq \emptyset$
  while $|V| < n$ **do**:
    repeat
    generate $q_{rand} \in \mathcal{Q}$
    until q_{rand} is collision-free
    $V \doteq V \cup \{q_{rand}\}$
  end while
  for $\forall q \in V$ **do**:
    compute k closest neighbors of q in V, N_q using **dist**
    for $\forall q' \in N_q$ **do**:
      if $(q, q') \notin E$ and $\Delta(q, q') \neq$ NIL **then**:
        $E \doteq E \cup \{(q, q')\}$
      end if
    end for
  end for

2 Rapidly-Exploring Random Tree (RRT)

2.1 Procedure

1. Initialize the graph (or tree) G with q_{init} (one node only).
2. Pick a random node q_{rand} in C (or C_{free}).
3. Find a nearest node q_{near} in G to q_{rand}. Initially, the nearest node is q_{init}.
4. Select a new configuration q_{new} by moving from q_{near} by the step size Δq.
5. Add the node q_{new} and the edge (q_{near}, q_{new}) to the graph (needs collision checking)
6. Repeat the process until maximum number of nodes, or the goal configuration is hit.

2.2 Pseudocode

This is modified from [2].

function Build_RRT(q_{init}, NumNodes, Δq)
    G.init(q_{init})
    for $k = 1$: NumNodes **do**:
        Choose a random configuration, q_{rand}, in C
        $q_{near} \doteq$ Nearest_Vertex(q_{rand}, G)
        $q_{new} \doteq$ New_Conf$(q_{near}, \Delta q)$
        G.add_Vertex(q_{new})
        G.add_Edge($[q_{near}, q_{new}]$)
        if $q_{new} = q_{goal}$ **then**:
            Break
        end if
    end for
    Return G

function Nearest_Vertex(q, G)
    $d \doteq \infty$.
    for each vertex $v \in G$ **do**:
        if dist$(q, v) < d$ **then**:
            $v_{new} \doteq v$
            $d \doteq$ dist(q, v)
        end if
    end for
    Return v_{new}

Then by using the information of $Vertex$ and $Edge$, perform backward search to reconstruct the path.

3 RRT*

This is an extended/modified version of RRT, called RRT*. See [3] for more details.

3.1 Procedure

1. Pick a random node q_{rand}.
2. Find the closest node q_{near} from explored nodes to branch out from, towards q_{rand}.
3. Steer from q_{near} towards q_{rand}: interpolate if node is too far away, reach q_{new}. Check that obstacle is not hit.
4. Update cost of reaching q_{new} from q_{near}, treat it as C_{min}. For now, q_{near} acts as the parent node of q_{new}.
5. From the list of 'visited' nodes, check for nearest neighbors with a given radius, insert in a list $q_{nearest}$.
6. In all members of $q_{nearest}$, check if q_{new} can be reached from a different parent node with cost lower than C_{min}, and without colliding with the obstacle. Select the node that results in the least cost and update the parent of q_{new}.
7. Add q_{new} to node list.
8. Continue until maximum number of nodes is reached or goal is reached.

3.2 Pseudocode

This is modified from [3].

function Path = RRTstar(q_init, q_goal, NumNodes, Δq)
  $q\_init.cost \doteq 0$ and $q\_goal.cost \doteq 0$
  $Nodes(1) = q\_init$
  for $i = 1$: NumNodes **do**:
    generate q_rand
    if dist(q_rand, q_goal) $= 0$ **then**:
      break
    end if
    Pick the closest node q_near from the existing list of nodes.
    if dist(q_near, q_rand) $> \Delta q$ **then**:
      $q\_new \leftarrow$ moving q_near by Δq toward q_rand.
    else
      $q\_new = q\_rand$.
    end if
    if No collision **then**:
      $q\_new.cost = \text{dist}(q\_new, q\_near) + q\_near.cost$
      Select $\{q\_nearest\}$ around q_new within the radius r (check collision as well).
      $q\_min \doteq q\_near$

$C\_min \doteq q\_new.cost$
for $\forall q \in \{q\_nearest\}$ **do**:
  if no collision & $\text{dist}(q, q\_new) < C\_min$ **then**:
    $q\_min \doteq q$ and $C\_min \doteq q.cost + \text{dist}(q, q\_new)$
  end if
end for
for $\forall q \in$ Nodes **do**:
  if $q \in$ Nodes $= q\_min$ **then**:
    $q\_new.parent = q$
  end if
end for
$Nodes$.append(q_new).
end if
end for

Then by using the parent information of each point in $Nodes$, perform backward search to reconstruct the path.

References

[1] Howie Choset, Kevin Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, and Sebastian Thrun. *Princinples of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005.

[2] http://msl.cs.uiuc.edu/rrt/about.html.

[3] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. In *Robotics: Science and Systems*, Zaragoza, Spain, 2010.