**PLANNING WITH COSTS**

16-662 : Robot Autonomy
Scribe Notes : 19th March, 2014
Jiawei Liu, Sameer Ansari, Shinjini Saha, Vamshi Krishna Konduri
Andrew ID - jiawei, sameera, shinjins, kvk

**Planning with constraints**

Hard Constraint \( q^* = \text{argmin}_q (F(q)) \) such that \( g(q) = 0 \)

Soft Constraint \( q^* = \text{argmin}_q (F(q) + \lambda g(q)) \)

The trajectory is an infinite dimensional object.

\[ N_{\text{dims}} = (\text{number of dimensions in configuration space}) \times (\text{number of points} \sim \infty) = \infty \]

**Piano Movers Problem**

Choose a point \( q \in C \), where \( C \) is Configuration space, and \( C_{\text{obs}} \) is the part of \( C \) space that is obstructed, and \( C_{\text{free}} \) are the points that aren’t obstructed. Refer to Figure 1.

Given \( q_s \) and \( q_g \) which are the start and goal configurations respectively.

Find a continuous path \( \xi : [0,1] \rightarrow C_{\text{free}} \) such that \( \xi[0] = q_s \) and \( \xi[1] = q_g \).

---

**Figure 1** – Example of C space with a continuous path from the start to goal configurations.
This was a Feasible path planning problem. Now to get an optimal path, things to optimize in a trajectory are:

1. Path length given the number of waypoints.
2. Energy given by the number of joints to be moved.
3. Clearance from obstacles.
4. Planning time.

Kinetic energy can be defined by the Functional (function of a function) of the trajectory as below:

\[ F[\xi] = \frac{1}{2} \int_0^1 \dot{\xi}(t)^T M \ddot{\xi}(t) \, dt \]

**Clearance from obstacles**

We want to find a number that is high when the robot is close to the obstacle and low when away from it. So, the distance, \( d \) to the closest obstacle is determined. Figure 2 shows how the cost function, \( C \) is related to \( d \). When the robot is near the obstacle, the cost should be very high, almost close to infinity.

\[ F[q] = \int_0^1 C(\xi(t)) \, dt \]

An issue right now is our function is not invariant to re-parameterization, meaning that it will choose paths that get to the goal faster, despite any other in-efficiencies in the path. Going twice as fast results in half the cost. This butting heads of weighting needs to be addressed separately.

**Invvariance to re-parameterization**

A metaphor for invariance to re-parameterization is to imagine a field of grass. The grass grows to different heights over time. A tractor mows the grass, and no matter how fast you drive, you will mow the same amount of grass.
We need to penalize for speed to counter the reduced cost. Richard Feynman discussed (Path of a Scalar Field) this topic and defined clearance to be invariant to re-parameterization. The path integral of the scalar field is now given by,

$$ F[q] = \int_0^1 C(\xi(t)) |\xi'(t)| dt $$

**Cost-based Path Shortening**

We found a way to define the cost. Let us now write an algorithm that can optimize the cost.

Let us go through an algorithm which we are familiar with and associate cost heuristic to it.

1. **Path Shortening** – This simple method outperforms any other algorithm in high-dimensional spaces. The steps are as follows:
   i. Sample q1 and q2 as shown in Figure 3.
   ii. Connect them using the simple planner (i.e. by a straight line).
      -- If this connection succeeds (indicating this path is shorter), accept it.
      Now we incorporate a path cost, $C(q_s -> q_1 -> q_2 -> q_g)$.
      Then, the relationship between original path cost $C(\xi) = C(q_s -> q_g)$ and this new cost for this path to be acceptable is given by $C(\xi) > C(q_s -> q_1 -> q_2 -> q_g)$.

   This check is done implicitly, because this algorithm involving path length is always true. Accept the path of a new cost only if it’s lower.

2. **Metropolis Algorithm** – Every once in a while, accept an increase in the cost in order to come out of a local minima. The mathematical explanation is as follows:
   -- Accept your cost with probability, $p = 1$ if $\Delta C = C(\xi) - C(\xi') < 0$ [i.e. cost of old trajectory – cost of new trajectory < 0]. This does not prevent descent along the hill.

![Figure 3 – Path shortening.](image)
While ascending the hill i.e. $\Delta C \geq 0$, then accept it with a probability, $p \propto \exp \left[-\frac{\Delta C}{T}\right]$. The probability decays as the cost becomes larger. Also, as $T$ (temperature) goes higher, the probability of accepting the cost increase goes higher. This method is an application of simulated annealing.

The two variants of Metropolis algorithm are:

i. Monte-Carlo Method – The temperature is fixed at some point.
ii. Simulated Annealing – The temperature is modified as the optimization progresses. It should be increased when rejection of cost increase is more often and there is failure i.e. a shallow point is reached, and decreased when there is success.

Now, we enable the RRT algorithm itself to optimize cost so that we have an optimal solution as compared to general RRT. Two methods which do this:

1) Heuristically-guided RRT
2) Anytime RRT

**Heuristically-guided RRT**

See Figure 4 for an RRT with a start position and an obstacle. It includes the following:

i. $C(a,b) =$ cost of a path in the tree from $a$ to $b$.
ii. $C^*(a,b) =$ optimal cost of the path from $a$ to $b$. This path is perceived to be optimal and may not be the actual optimal path since we do not have information on that yet.
iii. $h(a,b) =$ heuristic cost of the path from $a$ to $b$.

A true optimal cost is always greater than or equal to the current cost. A straight line path or a path obtained by moving along the edges is always shorter than this optimal path. So, this is a lower bound on the cost function. This is called an **Admissible** heuristic i.e. $h(a,b) \leq C^*(a,b)$.

![Figure 4 – Various cost-functions shown to illustrate heuristically-guided RRT algorithm.](image)
So, the three things – actual cost (upper bound), heuristic cost (lower bound) and optimal cost (lying in between the two bounds) can be shown as in Figure 5.

```
   ┌──────┐
  └───────┘
    |      |
  ┌──────┐
  └───────┘

   c (a, b)
   ┌──────┐
  └───────┘
    |      |
  ┌──────┐
  └───────┘
    h (a, b)
    ┌──────┐
  └───────┘

 Figure 5 – Relationship between the cost-functions.
```

Heuristically-guided RRT defines \( m_q \) – a node quality measure i.e. how good a particular node is at being near the optimum. It defines

\[
C(q) = C(q_s, q) + h(q, q_g),
\]

where \( C(q_s, q) \) is the cost to come from the start to the current node and \( h(q, q_g) \) is the heuristic to go from the current node to the goal.

\[
C_{\text{max}} = \max_{q \in \text{nodes}} C(q) \text{ i.e. the worst possible node whose cost to come and heuristic cost to go is the highest among all of the nodes.}
\]

\[
C_{\text{opt}} = h(q_s, q_g) \text{ i.e. the lowest possible cost.}
\]

Therefore, \( m_q = 1 - \left[ \frac{C(q) - C_{\text{opt}}}{C_{\text{max}} - C_{\text{opt}}} \right] \in [0, 1] \) where, 0 indicates ‘worst’ node and 1 indicates ‘best’ node.

Now, all the nodes in RRT have unequal unimportance depending upon the above measure. If the RRT only explores the best nodes by being greedy it is trading off exploration for exploitation. The algorithm would fail in that case when an obstacle is hit.

So, there has to be a way to bias RRT tree to perform exploration rather than exploitation. The method to do that is given in the following steps :

1. Sample \( q_s \in C_{\text{free}} \)
2. Get \( q = \text{Nearest Neighbor i.e. NN}(q_s) \)
3. Compute \( m_q \). If \( m_q = 1 \), then greedy search is done otherwise, exploration.
4. \( p = \max(m_q, p_{\text{min}}) \)

<table>
<thead>
<tr>
<th>( m_q )</th>
<th>( p_{\text{min}} )</th>
<th>accept</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>0.3</td>
<td>0.8</td>
</tr>
<tr>
<td>0.1</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>
v. Accept if \( \text{rand}[0,1] \leq p \)
Larger the \( p_{\text{min}} \), more closer it is to RRT and smaller the \( p_{\text{min}} \), more closer it is to greedy search.

vi. Extend towards \( q_s \).

A disadvantage of this algorithm is that it can get stuck as illustrated in Figure 6.

![Figure 6 – Disadvantage of heuristically-guided RRT.](image)

Imagine the space which contains an obstacle and a high-cost and a low-cost region. A and B are the nodes. B has a high \( m_q \) and A has a very low \( m_q \). The \( q_s \) will then be sampled very often in the high-cost region. Its nearest neighbor is B which will then be rejected. Thus, it gets stuck as the tree stretches towards high-cost region.

For narrow passages in cost-space, the above problem becomes even harder along with having to deal with narrow passages in obstacle-space. One way to fix this is to look at K-nearest neighbors. Another method is Anytime RRT, where initially a path is found by completely ignoring the cost and only dealt with obstacles and then gradually starting to introduce cost so that we get better and better paths over time.

**Anytime RRT**

The idea behind anytime RRT is to be able to find one path as fast as possible and then iterate the RRT planner to improve the path until the time permits. This is based on the ‘Anytime Property’ which states that this algorithm will be able to provide an answer at any time before it completes and the quality of the solution provided by the algorithm will keep increasing as the algorithm is allowed to run for more and more time.

If this algorithm is allowed to run forever, it will keep generating new paths with improvements at each iteration as long as the improvement in cost is significant enough. The improvement decision is based in terms of amount of improvement in cost of the newly generated path (\( C_{s'} \)) as compared to the cost of its preceding path (\( C_s \)). Specifically, the algorithm will keep generating new paths when

\[
C_s' < (1 - \varepsilon_f) \times C_s
\]
In this equation, $\varepsilon_f$ is called as ‘improvement’ and $(1 - \varepsilon_f) \times C_s$ is called as ‘target’ ($C_{target}$). The algorithm will terminate when the cost cannot be improved within stipulated time for an iteration.

The anytime RRT algorithm can be explained in high level as follows:

Loop
- Generate a new tree from the RRT algorithm with cost improvement $C'_s$
  \[ < (1 - \varepsilon_f) \times C_s \]
- Exit the loop if the improvement in cost cannot be achieved under iteration time limit
- Erase the old tree

In order to ensure that the cost of a new solution is better than its previous path, the anytime RRT makes a few changes in the three basic tasks of the RRT, namely node sampling, node selection and extension.

1. **Node sampling** – The goal of a given iteration in anytime RRT is to generate a path with cost better than the target cost as defined previously. A tree can never achieve this target cost if the heuristic cost from start node to any given node of the tree and the cost from that node to the goal has lower bound greater than this target. This lower bound can be defined in terms of heuristic cost. Refer to Figure 7.

![Figure 7 – Disadvantage of heuristically-guided RRT.](image)

So, in the modified node sampling procedure, a randomly sampled node is rejected when

$$\text{Cost to come} + \text{Cost to go} > C_{target}$$

$$h(q_s, q_{sample}) + h(q_{sample}, q_g) > C_{target}$$

2. **Node selection** – The selection of nearest neighboring node from the tree in the normal RRT algorithm is based only the distance of the randomly sampled node from existing nodes of the RRT tree. There is no sense of cost to travel to that neighbor in this selection criteria.
In the modified node selection criteria for anytime RRT, the node selection is based on two parameters. The distance of newly sampled node to the candidate node from tree as well as the cost to come to that candidate node. In particular, node selection cost is defined as

\[ \text{SelCost}(q) = d_b \ast \text{distance}(q, q_{\text{sample}}) + C_b \ast C(q_{\text{start}}, q) \]

In the very first iteration of the algorithm, \(d_b\) is set to 1 and \(C_b\) is set to 0. This is exactly equal to the distance metric for normal RRT algorithm. As we move to further iterations, the value of \(d_b\) is gradually increased by a step of \(\delta_d\) and the value of \(C_b\) is gradually increased by a step of \(\delta_C\) with the condition that these values are bounded in the range of \((0, 1)\). Thus, we gradually increase the relative influence of the cost to come into the decision criteria with the expectation that this will result in a path with better cost.

3. **Extension** – During the extension phase, anytime RRT incorporates the heuristic cost to go to the target once again. First of all, \(k\) nearest neighbors are found from the newly sampled node \((q_{\text{sample}})\). Then, each of these neighbors are extended using the extension function to get \(T\) possible extensions \((\tau_1, \tau_2, \tau_3 \cdots \tau_T)\). Out of these extensions, the best extension is selected based on sum of cost to come to that extension and heuristic cost to go to the target from that extension. Refer to Figure 8.

![Extension method](image)

\[ q_{\text{new}} = \arg \min_{q \in \tau_{1-\tau_T}} [ C(q_s, q) + h(q, q_g) ] \]

The node \(q_{\text{new}}\) is selected only if \(C(q_s, q_{\text{new}}) + h(q_{\text{new}}, q_g) < C_{\text{target}}\). If this condition is not met, the extension function returns NULL and the anytime RRT algorithm terminates and returns last successful tree.

This method works well in practice but requires the use of many extra parameters and also does not reuse the tree.