CS365A: Artificial Intelligence
Potential Guided RRT*
Advisor: Prof. Amitabha Mukherjee

Reid Rizvi Rahman
Roll no: 11594
Sakshi Sinha
Roll no: 11627

April 25, 2014

1 Problem Statement

We are given a workspace which contains a source S, a goal D and a set of obstacles Q. Our aim is to return a path which connects the source and the destination while avoiding the set of obstacles. At the same time, we also wish to minimize the length of this path.

In our model, we will represent this path as a sequence of nodes S, X_1, X_2, ... X_n, D such that there is an edge connecting every pair of consecutive nodes and none of these edges collide with the obstacle space Q.

For example, the following figure shows an instance of the workspace with the source shown in green, the destination in black and the path in blue.

2 Previous Work

2.1 Probabilistic Roadmap:

The PRM algorithm takes random samples from the configuration space and uses a local planner to connect these configurations to nearby configurations. Once this is done the source and the destination can be added and any shortest path algorithm can be used to determine the path.

2.2 RRT Algorithm:

The RRT algorithm aims to return a path by growing a tree rooted at the source. At each iteration of the algorithm, we generate a random node and add it to the tree by adding an edge between the newly connected node and the nearest node already present in the tree.

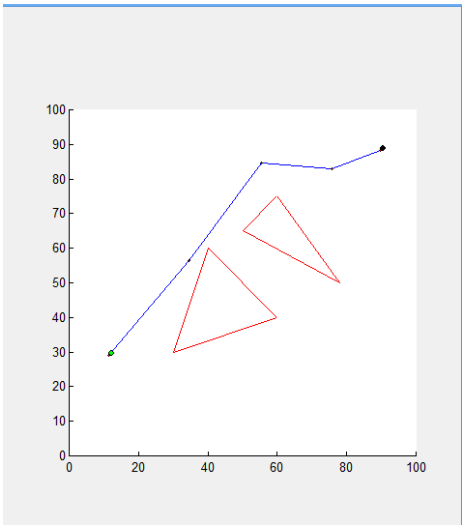

Figure 1: An instance of the robot motion planner.

2.3 RRT* Algorithm:

The main difference between the RRT* algorithm from the RRT is that the upon every insertion the algorithm updates the distance to all the existing nodes in the tree thus yielding faster convergence as compared to the RRT.

2.4 Artificial Potential Field:

This method defines an artificial potential using the obstacles and the goal. The negative gradient of the potential gives the force. The robot always follows the minimum potential path from the source to the destination.

3 Approach:

Our approach would be to combine the artificial potential model with the RRT* algorithm to give a potentialised RRT* algorithm to yield a higher convergence rate in comparison with the RRT* algorithm. At the same this algorithm avoids the local minima problem that is associated with the artificial potential field model.

4 Algorithm:

The potentialised RRT* algorithm is a single source-destination algorithm which grows a tree T rooted at the source. Initially, the source node is added to the tree. At each iteration a random node x is generated. This node x is then moved a fixed distance α along the direction of the potential gradient at this point to give a new node z. This node z is then added to the tree by adding an edge from this node to another node in the tree such that the distance from the source to z is minimised. The addition of this node may lead to a situation

wherein the distance to some other nodes that are already present need to be updated. We update these distances at each iteration to maintain the minimum distance to every node. Also this algorithm does not suffer from the drawback of local minima in potential that is exhibited by the potential field model.

5 Pseudo Code:

function POTENTIALISED-RRT*(S , D)
  $V \leftarrow \{S\}$
  $E \leftarrow \phi$
  for i = 0, 1, ... n **do**
    $z_{rand} \leftarrow SampleFree_i$
    x_{rand} *gets* $RandomisedGradientDescent(z_{rand})$
    $x_{nearest}$ *gets* $Nearest(G, x_{rand})$
    x_{new} *gets* $Steer(x_{nearest}, x_{rand})$
    if $ObstacleFree(x_{nearest}, x_{new})$ **then**
      $x_{near} \leftarrow Near(G, x_{new}, r, \eta)$
      $V \leftarrow V \cup \{x_{new}\}$
      $x_{rand} \leftarrow x_{nearest}$
      $c_{min} \leftarrow Cost(x_{nearest}) + c(Line(x_{nearest}, x_{new}))$
      for $x_{near} \in X_{near}$ **do**
        if $CollisionFree(x_{near}, x_{new})andCost(x_{near}) + c(Line(x_{near}, x_{new}))<c_{min}$ **then**
          $x_{min} \leftarrow x_{near}$
          $c_{min} \leftarrow Cost(x_{near}) + c(Line(x_{near}, x_{new}))$
          $E \leftarrow E \cup \{x_{min}, x_{new}\}$
        end if
      end for
      for $x_{near} \in X_{near}$ **do**
        if $CollisionFree(x_{near}, x_{new})andCost(x_{new}) + c(Line(x_{near}, x_{near})) < Cost(x_{near})$ **then**
          $x_{parent} \leftarrow Parent(x_{near})$
          $E \leftarrow (E \setminus \{x_{parent}, x_{near}\}) \cup \{x_{new}, x_{near}\}$
        end if
      end for
    end if
  end for
  return G = (V, E)
end function

6 Potential Field:

We define distance functions as follows:

$$\beta_i(q) = \begin{cases} -d^2(q, q_i) + r_i^2, & \text{i=0.} \\ d^2(q, q_i) - r_i^2, & \text{i¿0.} \end{cases} \tag{1}$$

$$\beta(q) = \prod_{i=0}^{n} \beta_i(q) \tag{2}$$

$$\gamma_k(q) = (d(q, goal))^{2K} \tag{3}$$

n is the number of obstacles in the workspace.
These distances are used for the calculation of artificial potential field of obstacles.

The navigation function in the sphere world is given by

$$\gamma_k(q) = \frac{d^2(q, goal)}{(d(q, goal)^{2K} + \beta(q))^{\frac{1}{K}}} \tag{4}$$

This function is the artificial potential function for the algorithm.

The gradient of this navigation potential is given by:

$$\bigtriangledown\gamma(q) = \frac{2d(q, goal) \bigtriangledown d(q, goal)(d(q, goal)^{2K} + \beta(q))^{\frac{1}{K}} - d^2(q, goal) \bigtriangledown (d(q, goal)^{2K} + \beta(q))^{\frac{1}{K}}}{(d(q, goal)^{2K} + \beta(q))^{\frac{2}{K}}} \tag{5}$$

The maximum negative gradient(5) gives the direction of movement.

7 Implementation Details:

We implemented the code for potential-guided RRT* in MATLAB. For the purpose of ease of defining potential we approximate the polygonal obstacles using a combination of circles. This approximation of the polygons using a combination of circles is done manually i.e. it is hard-coded into our program. The RRT* tree is stored as a two-column matrix with each row depicting an edge between the nodes stored in each of the columns. Corresponding to each node, we store the parent of the node in the RRT* tree and following this parent from the destination to the source allows us to find the required path.

Function Details:

1. *HW2_bozcuoglu*: It contains the main function. It generates the random points to be added to the tree.

2. $HW2\_bozcuoglu.fig$:It is the picture of the GUI.

3. randomConfigFinder: This function generates a random point.

4. checkCollision: It checks that whether the point lies inside the obstacle or not.

5. stepCollisionCheck: This function checks wheter line joining two points pass through obstacles or not.

6. circle: This function creates circles.

7. rrtConnector: This function takes those points and generates a tree rooted at the source node.This also returns the parent of each node.

8. navigatial: This function generates a potential field guided path.

9. rrtConnector1:This function takes the randomly generated points and finds new points biased according to potential field.This function returns the tree and the parent of each node of the tree.

10. PathFinder: This function formally generates the path by taking all nodes and their parent node as input returned from rrtConnector.

8 Results

Theoretically, the potential-guided RRT* is expected to give a path which is farther separated from the goal as compared to the traditional RRT* algorithm since it takes into account the repulsive potential of the obstacles. When operated on the same tree for the same source and destination, this is exactly what the results of the implementation reveals as illustrated from the following figure:

The algorithm also does not suffer from any drawback related to the minima in potential field. This is clearly exhibited by the path returned by the potentialised RRT* algorithm in the following case where a normal potential field model would cease to yield accurate results due to the local minima in potential around the destination. This is illustrated in figure 3.

Also, for the same set of points the potentialised-RRT* is biased towards the goal and away from the obstacles. This is evident from the following figure which highlights the fact that the potentialised-RRT* produces a shift in the nodes away from the obstacles and towards the goal.

9 Major Challenges:

Using MATLAB as the implementation language posed a major hurdle as it is new to both members of our group. Also, integrating the implementation of potential field with the RRT* proved a challenge as it required redefining the obstacles in terms of circles. Also, in this context, we needed to exercise caution while redefining because any random point generated outside the scope of the potential function would lead to excessive running time of the algorithm.

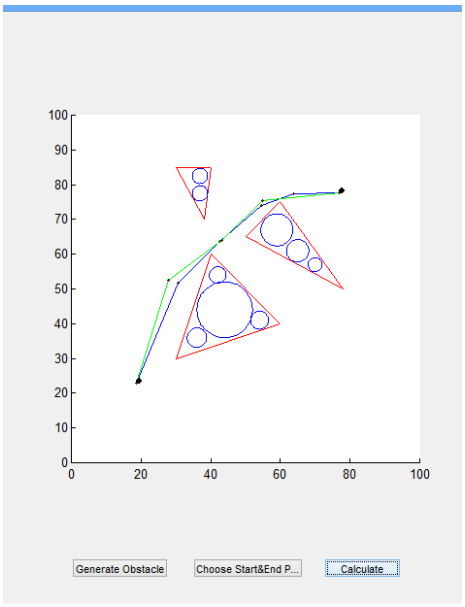

Figure 2: The path in green is the one returned by potentialised RRT*, the one is blue is the traditional RRT*.

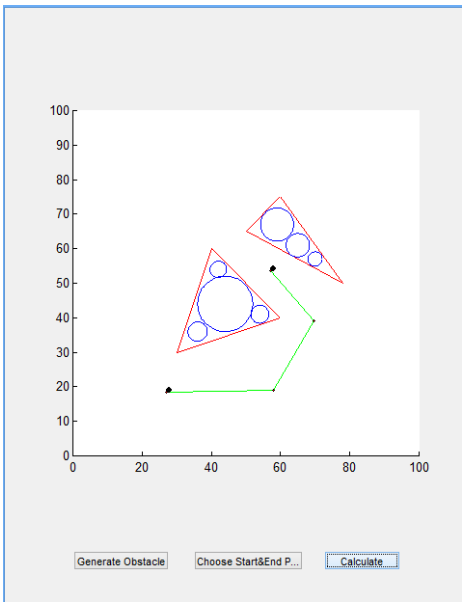

Figure 3: The path in green is the one returned by potentialised RRT* thus illustrating that it can even pass through areas of local minima in potential.

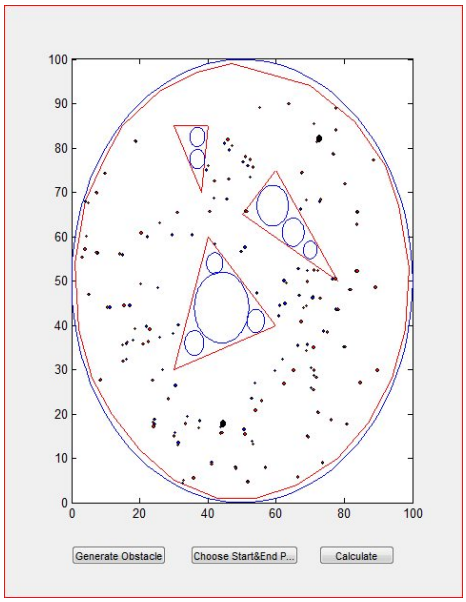

Figure 4: The red nodes are those produced during normal RRT* and the blue ones are produced during potentialised-RRT*. Clearly the blue nodes are shifted away from the obstacles and towards the goal.

10 Future Work:

The current version of motion planning is based on point robots. This could be extended to include finite-sized robots as in the case of robotic arms etc. Also the current measure of path is based entirely on the distance to the destination. Another possible measure could be the nearest distance to the obstacle and that could be treated as an extension of the algorithms used in this project.

11 References:

1. Potential Guided Directional-RRT* for Accelerated Motion Planning in Cluttered Environments by A.H. Qureshi, K. F. Iqbal, S. M. Qamar, F. Islam, Y. Ayaz and N. Muhammad

2. Sampling-based Algorithms for Optimal Motion Planning by S. Karaman and E. Frazzoli

3. Efficient Local Sampling for Motion Planning of a Robotic Manipulator by S. Byrne, W. Naeem and R. S. Ferguson

4. `http://kovan.ceng.metu.edu.tr/~asil/old/_1./hw4.html`