Path Planning for Turtlebot using Informed RRT* algorithm

Vishwanath Datla

UID: 117705928

University of Maryland, College Park

***Abstract*—In this project, Informed RRT* algorithm is used for motion planning of turtlebot 2 in a map consisting of stationary obstacles. Informed RRT* algorithm generates a path for the turtlebot to follow in the map with obstacles. It is important that the path generated from the original position to goal position is optimal and also generated quickly. Informed RRT* was picked as the algorithm for this project because of it's efficient rate of convergence, cost of solution, and it's ability to generate a path for the robot irrespective of the dimension of the state or range of motion planning problem. The project is simulated in gazebo for path planning visualisation.**

***Index Terms*—Informed RRT*, Path Planning, Turtlebot, Gazebo**

I. Introduction

Robot path planning is done for mobile robots to be able to traverse in an environment with obstacles. The motive behind developing a path planning algorithm is to generate a collision-free and efficient path for the robot. Humans are capable of avoiding small obstacles and also take a completely different path when a huge obstacle is blocking their path. However, a robot does not possess the intelligence to do so and requires the help of path planning algorithms. Some of the most commonly used path planning algorithms are Dijkstra [1] [2], A* [3], RRT(Rapidly exploring random tree), and RRT* [4].

Since the robot does not have any nodes to traverse, a graph must be generated to help guide the robot by creating virtual nodes. A* is technically a variation of Dijkstra with the only difference being that A* looks for a better path by using a heuristic function that gives priority to nodes that lead to a more efficient path whereas Dijkstra explores all possible paths.

In RRT algorithm, points are randomly generated and the closest node is selected for traversal. It generates the optimal path by randomly building a space filling tree. It has to be made sure that the vertex is not inside an obstacle. The algorithm ends when a randomly generated vertex is within the goal region. Different variations can be used with the aforementioned algorithm as the basis. One such variation is the RRT algorithm. RRT* is an optimized version of RRT. RRT* improves on RRT in 2 ways:

- It introduces a cost function which records the distance a vertex travelled relative to the parent vertex. Once the closest node is found, the algorithm looks at nodes within a radius of the selected node. If of these nodes has a lower cost, it replaces the original node.
- It can deliver an optimal path even when the number of nodes reaches infinity.

Informed RRT* algorithm is used for navigation in this project. Informed RRT* algorithm outperforms RRT* in terms of convergence rate and cost. The algorithm will be discussed in detail in the following sections. This project is based on the paper 'Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic' [5].

II. Related Work

Gammell et al. [5] introduce the informed RRT* algorithm and show that it outperforms the RRT* algorithm. Chang et al. [6] propose an improved RRT* algorithm to navigate a mobile robot in an unknown environment using the sensors of the robot to detect obstacles. Karaman et al. [7] provide an in depth analysis of the behaviour and solution cost of sampling based algorithms like PRM*(Probabilistic roadmaps) and RRT*. Elbanhawi et al. [8] do an extensive review of state of the art sampling-based motion planning algorithms and shed light on current challenges in motion planning.

III. Algorithm

It is important to establish the working of RRT* algorithm before moving to informed RRT*. RRT* provides a significant improvement in the paths generated as compared to RRT by introducing the cost function. A significant drawback of the RRT* algorithm is that although it generates better and more efficient paths, it takes a longer time to execute. The execution time is more because it constantly tries to improve the path that is already generated which leads to more computations. When a new random vertex is selected, the RRT algorithm adds the nearest node to the tree. However RRT* checks the neighbouring nodes within a defined radius to find a node that can possible be a more cost effective path. If such a node is found, the tree is updated with the more optimal node. Connell et al. [9] provide the algorithm for RRT* algorithm.

Algorithm 1: $T = (V, E) \leftarrow$RRT*(q_{init})

$T \leftarrow$InitializeTree()
$T \leftarrow$InsertNode($\emptyset$,q_{init},T)
for $k \leftarrow 1$ **to** N **do**
  $q_{rand} \leftarrow$RandomSample(k)
  $q_{nearest} \leftarrow$NearestNeighbor(q_{rand},Q_{near},T)
  $q_{min} \leftarrow$ChooseParent(q_{rand},Q_{near},$q_{nearest}$, Δq)
  $T \leftarrow$InsertNode(q_{min}, q_{rand}, T)
  $T \leftarrow$Rewire(T, Q_{near}, q_{min}, q_{rand})
end

Fig. 1. Algorithm 1 [9]

Algorithm 2: $q_{min} \leftarrow$ChooseParent(q_{rand},Q_{near},$q_{nearest}$,Δq)

$q_{min} \leftarrow q_{nearest}$
$c_{min} \leftarrow$Cost($q_{nearest}$) + c(q_{rand})
for $q_{near} \in Q_{near}$ **do**
  $q_{path} \leftarrow$Steer(q_{near}, q_{rand}, Δq)
  if *ObstacleFree*(q_{path}) **then**
    $c_{new} \leftarrow$Cost(q_{near}) + c(q_{path})
    if $c_{new} < c_{min}$ **then**
      $c_{min} \leftarrow c_{new}$
      $q_{min} \leftarrow q_{near}$
    end
  end
end
return q_{min}

Fig. 2. Algorithm 2 [9]

Algorithms 1 and 2 are used for the selection of a node, checking the neighbouring nodes for a more cost effective solution and updating the tree if such a solution is found. Algorithm 3 is used to rewire the path if a more effective

Algorithm 3: $T \leftarrow$Rewire(T, Q_{near}, q_{min}, q_{rand})

for $q_{near} \in Q_{near}$ **do**
  $q_{path} \leftarrow$Steer(q_{rand}, q_{near})
  if *ObstacleFree*(q_{path}) *and Cost*(q_{rand}) + *c*(q_{path}) < *Cost*(q_{near}) **then**
    $T \leftarrow$ReConnect(q_{rand}, q_{near}, T)
  end
end
return T

Fig. 3. Algorithm3 [9]

node is found. These algorithms are discussed in more detail in the following subsections.

A. Initialization

Before implementing the RRT* algorithm, the start or root node must be decided. This is where the robot will start it's path. Once the starting point is intialized, the step size for the robot is specified. Step size is the minimum distance that the robot must cover between consecutive node. If a step size of 5cm is selected, the new node cannot be within 5cm radius of the previous node. If the map has a lot of obstacles which are clumped together, it is better to select a smaller step size to avoid a having to find many nodes in obstacle spaces. However, if there are a small number of obstacles in a large space, a large step size can be selected. Step size plays an important role in the execution time.

B. Vertex selection

After the starting node is initialized and step size is decided, the next node for robot traversal must be found. This node is generated randomly and there is no pre-determined algorithm influencing what the next node will be.

C. Searching the neighbouring nodes

once the random vertex is generated, the neighbourhood of this node within a radius is searched and all the nodes within this area are detected. The node which is at the lowest euclidian distance from the random vertex is selected as the new node.

D. Finding the best parent node

The neighbourhood of this node is examined with a radius which is generally a multiple of the step size. The node in the neighbouring region which is at the smallest euclidian distance of the previous node is previous node is selected. If this node provides a more efficient path for the robot it is updated as the new node.

E. Rewiring

After the neighbourhood is checked and all the possible nodes are added to the list, the neighbours are checked for optimal solution and once found, the graph is updated.

F. Termination of the algorithm

After all the above steps are implemented, it is checked if the goal node is within the threshold distance of the goal node. If the new node is close enough to the goal node the algorithm is terminated and the robot path is decided.

Algorithm 4: ExecutePath()

SetObsDestination($numObs$)
SetObsVelocities($numObs$)
SetRobotDestination()
SetRobotVelocity()
while $robotLocation! = GOAL$ **do**
  UpdateObsLocation($numObs$)
  UpdateRobotLocation()
  if $Replan$ **then**
    DoReplan()
  end
end

Fig. 4. Algorithm 4 [9]

Algorithms 4 and 5 are used for dynamic replanning. In real world, we cannot expect objects to be stationary. If there are moving obstacles that are detected using robot sensors, the robot path must be constantly updated. From the initial planning, the robot fixes a traversal path and starts to move towards the goal. If during traversal, a random moving obstacle is encountered, the robot needs to replan it's path. Sensors like LiDAR can be used to detect obstacles. Once detected, if the robot estimates that there is a possibility of the random

Algorithm 5: UpdateRobotLocation()

```
robotLocation ← robotLocation + robotVelocity
if robotLocation == robotDestination then
    robotDestination ←GetNextPathLocation()
    SetRobotVelocity()
end
while obsIndex < numObs do
    obsDistance ←GetDistance(robotLocation,...
          ObsLocation(obsIndex))
    if obsDistance < robotRange then
        obs_path ←Steer(robotLocation, obsLocation)
        if ObstacleFree(obs_path) then
            if IsPathBlocked(obsIndex) then
                Replan ←TRUE
            end
            SetObsVisible(obsIndex)
        end
    end
end
```

Fig. 5. Algorithm 5 [9]

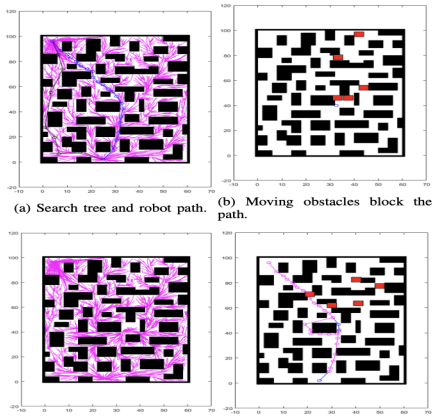
Algorithm 6: $T \leftarrow$DoReplan()

```
InvalidateNodes()
GetReplanGoalLocation()
SetReplanSamplingLimits()
Rewire(T, Q_all, NULL, q_robot)
RRT*(q_robot)
SetReplanPath()
```

Fig. 6. Algorithm 6 [9]

obstacle being in it's path, a new path is planned again and the route is updated using the algorithm below.

Following is the sample result of the algorithm rerouting on encountering a random obstacle in it's original path.



(a) Search tree and robot path. (b) Moving obstacles block the path. (c) Search tree after modification during replanning. (d) Final obstacle positions and the executed path.

Fig. 7. Sample result [9]

IV. Implementation

When used with simple configuration spaces RRT* and informed RRT* provide similar results but as the configuration space becomes more and more complex, informed RRT* generates much better results when compared with RRT*. Informed RRT* algorithm is an improvement on the RRT* algorithm in that it has a better convergence rate. The following steps are taken to implement the informed RRT* algorithm:

- Just like RRT* algorithm, informed RRT* algorithm also finds the random vertex, then the best node based on euclidian distance and builds the tree. The graph generated is then rewired in a way that the cost to the nearby vertices is minimised. All the steps for RRT* discussed in detail above are the initial steps of informed RRT* too.
- This algorithm is different from RRT* in that after the solution is found in a similar way to RRT*, it tries to improve the solution further by directly sampling the ellipsoidal heuristic. To optimise or minimise the length of the path, it is necessary that states from an ellipsoidal subset of the planning domain are added to all the iterations.

Following is the algorithm implemented for informed RRT* Sample function: Given two poses, x_{from}, x_{to} X_{free} and a

Algorithm 1: Informed RRT*($\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}$)

```
V ← {x_start};
E ← ∅;
X_soln ← ∅;
T = (V, E);
for iteration = 1 ... N do
    c_best ← min_{x_soln ∈ X_soln} {Cost(x_soln)};
    x_rand ← Sample(x_start, x_goal, c_best);
    x_nearest ← Nearest(T, x_rand);
    x_new ← Steer(x_nearest, x_rand);
    if CollisionFree(x_nearest, x_new) then
        V ← ∪{x_new};
        X_near ← Near(T, x_new, r_RRT*);
        x_min ← x_nearest;
        c_min ← Cost(x_min) + c · Line(x_nearest, x_new);
        for ∀x_near ∈ X_near do
            c_new ← Cost(x_near) + c · Line(x_near, x_new);
            if c_new < c_min then
                if CollisionFree(x_near, x_new) then
                    x_min ← x_near;
                    c_min ← c_new;
        E ← E ∪ {(x_min, x_new)};
        for ∀x_near ∈ X_near do
            c_near ← Cost(x_near);
            c_new ← Cost(x_new) + c · Line(x_new, x_near);
            if c_new < c_near then
                if CollisionFree(x_new, x_near) then
                    x_parent ← Parent(x_near);
                    E ← E \ {(x_parent, x_near)};
                    E ← E ∪ {(x_new, x_near)};
        if InGoalRegion(x_new) then
            X_soln ← X_soln ∪ {x_new};
return T;
```

Fig. 8. Informed RRT* algorithm 1 [5]

maximum heuristic value, c_{max}, the function Sample (x_{from}, x_{to}, c_{max}) returns independent and identically distributed samples from the state space, x_{new} X, such that the cost of an optimal path between x_{from} and x_{to} that is constrained to go through x_{new} The algorithm discussed above is implemented for turtlebot2 to navigate a relatively cluttered configuration space consisting of stationary obstacles. In order to determine the path for the robot without collision, the dimensions of the robot should be considered. A certain amount of clearance is also taken into consideration so the robot can stay clear of the obstacles comfortably and not be too close to them. Following are the robot parameters used in the code:

- Clearance = 25cm

Algorithm 2: $\texttt{Sample}\left(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}, c_{\max}\right)$

if $c_{\max} < \infty$ **then**
  $c_{\min} \leftarrow \left|\left|\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{start}}\right|\right|_2$;
  $\mathbf{x}_{\text{centre}} \leftarrow \left(\mathbf{x}_{\text{start}} + \mathbf{x}_{\text{goal}}\right)/2$;
  $\mathbf{C} \leftarrow \texttt{RotationToWorldFrame}\left(\mathbf{x}_{\text{start}}, \mathbf{x}_{\text{goal}}\right)$;
  $r_1 \leftarrow c_{\max}/2$;
  $\{r_i\}_{i=2,\ldots,n} \leftarrow \left(\sqrt{c_{\max}^2 - c_{\min}^2}\right)/2$;
  $\mathbf{L} \leftarrow \text{diag}\{r_1, r_2, \ldots, r_n\}$;
  $\mathbf{x}_{\text{ball}} \leftarrow \texttt{SampleUnitNBall}$;
  $\mathbf{x}_{\text{rand}} \leftarrow (\mathbf{C}\mathbf{L}\mathbf{x}_{\text{ball}} + \mathbf{x}_{\text{centre}}) \cap X$;
else
  $\mathbf{x}_{\text{rand}} \sim \mathcal{U}(X)$;
return $\mathbf{x}_{\text{rand}}$;

Fig. 9. Informed RRT* sample function [5]

- Radius = 20cm
- Wheel radius = 3.8cm
- Distance between wheels = 34cm

These values are taken from the datasheet of turtlebot2. Turtlebot2 is considered a rigid robot with the above mentioned dimensions.

Fig. 10. Turtlebot2

The configuration space is defined before the algorithm is implemented. Following is the configuration space used to ensure it is sufficiently cluttered.

Gazebo is used in combination with the Robot Operating System(ROS) for simulation of this algorithm with turtlebot2. The path generated by the algortihm is published to a ROS topic which then moves the robot along the generated path.

Prior to the package being launched, an executable must be created for the python file using the command chmod +x turtlebot_rrt.py. The launch file is then launched along with the specification of the posiiton and orientation of the robot. This opens a Gazebo window and also an additional terminal which asks for the start and goal position to be implemented. When the start and goal cooridnates are entered, the algorithm generates the path and the robot follows it.

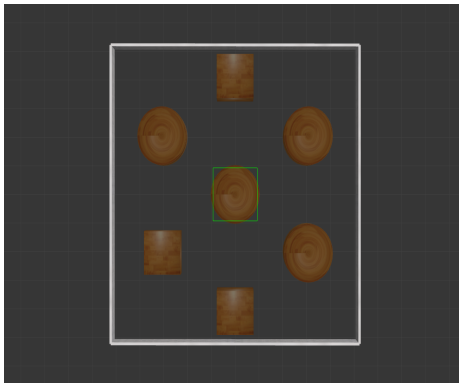


Fig. 11. Configuration space(Map)

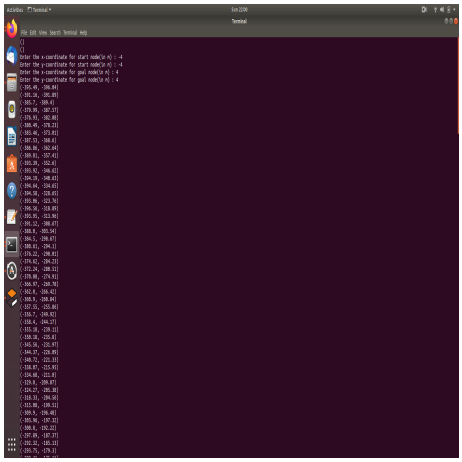


Fig. 12. Path generation

The path for the robot is generated accurately and gazebo is launched with the configuration space. However, there is a problem with spawning the robot in gazebo.

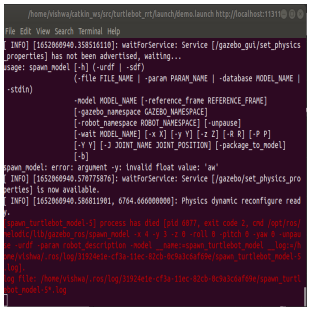


Fig. 13. Error

V. Conclusion and Future Work

Informed RRT* is an optimal algorithm for robot path planning in a configuration space such as the one used for this project. Future work involves figuring out the spawn error for the robot and working on implementing the algorithm in a configuration space with movable obstacles.

References

[1] Shu-Xi Wang, "The Improved Dijkstra's Shortest Path Algorithm and Its Application" *Procedia Engineering*. 29. 1186-1190. 10.1016/j.proeng.2012.01.110.

[2] Seifedine Kadry, Ayman Abdallah, Chibli Joumaa, "On The Optimization of Dijkstra's Algorithm ", *https://arxiv.org/pdf/1212.6055.pdf*

[3] Peter E Hart, Nils J Nilsson, Bertram Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths",*IEEE Transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100-107, July 1968

[4] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli and S. Teller,"Anytime Motion Planning using the RRT*",*IEEE International Conference on Robotics and Automation*, 2011, pp. 1478-1483.

[5] Jonathan D. Gammell, Siddhartha S. Srinivasa, Timothy D. Barfoot, "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic", *https://arxiv.org/abs/1404.2334*

[6] Liu Chang-an, Chang Jin-gang, Li Guo-dong and Liu Chun-yang, "Mobile robot path planning based on an improved rapidly-exploring random tree in unknown environment,"*EEE International Conference on Automation and Logistics*,2008, pp. 2375-2379

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning with deterministic -calculus specifications," *American Control Conference (ACC)*, 2012, pp. 735-742

[8] M. Elbanhawi and M. Simic, "Sampling-Based Robot Motion Planning: A Review," *IEEE Access*, vol. 2, pp. 56-77, 2014

[9] D. Connell and H. M. La, "Dynamic path planning and replanning for mobile robots using RRT," *2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2017, pp. 1429-1434