Path Planning of TurtleBot in Robot Operating System (ROS)

Rishabh Mukund
Engineering, Professional Masters
University of Maryland
College-park, Maryland
rmukund@umd.edu

Mukundhan Rajendiran
Engineering, Professional Masters
University of Maryland
College-park, Maryland
mrajendi@umd.edu

***Abstract*— Path planning is an essential toolset required for mobile robots. Choosing an appropriate algorithm ensures safe, cost-effective, and collision-free point-to-point navigation for the robot. This project focuses on implementing the Rapidly Exploring Random Trees star (RRT*) algorithm to move a turtle bot in a simulated environment using Gazebo to find the optimal path from the start node to the goal node. This is a simple yet powerful algorithm and we aim to enhance the convergence of the final solution by utilizing the Informed RRT* variant.**

Keywords— Informed RRT*, RRT*, Path Planning, Motion Planning, Robot Operating System (ROS), Gazebo, TurtleBot2

I. INTRODUCTION

Path planning is a major field of research in mobile autonomous robotics. The key element for successful path planning is to maximize safety and minimize collisions and costs in the path between the starting point and the target point. The applicability of a technique will generally depend on the system's kinematics, the environment's nuances, and also the capabilities of the robot being used.

Research into the field has given us numerous algorithms to work with. However, these algorithms can be broadly classified into two: Action-based and Sampling-based planning. Action-based planning approach is through converting the environment into discrete approximations with chunks of equal size like a grid map. However, an issue with Action Based arises with its scalability. The bigger the environment becomes, the more resource-intensive and complex the solution becomes.

Alternatively, the Sampling-based algorithms avoid the need for the discretization of state space. This supplements the scalability of these solutions with the environment size and directly considers other dynamic constraints; however, the result has a significant probability of being incomplete.

In our project, we explore the functionality of one such Rapidly-exploring Random Trees (RRT) and its variants. We end with implementing one of its optimized variants, Informed RTT*. RTT algorithm is quick to spread in the environment and look for a path but lacks the ability to specifically look for optimized solutions. A variant of the RRT is RRT* motion planning algorithm which quantifies the cost associated with each path and attempts to ensure that the cost is minimized.

While RTT* provides a better solution than RTT, it in itself still iterates through each point in the environment. This can eventually tend to be resource-intensive and inefficient. To further counteract this shortcoming, the Informed RRT* motion planning algorithm reduces the overheads by reducing the scope where the RTT* algorithm should work. This reduces frivolous cycles spent exploring areas providing little to no value to our original intent of finding an optimized path.

In this paper, The Informed RRT* motion planning algorithm was used on ROS Turtlebot 3 to navigate in a configuration space consisting of static obstacles.

II. LITERATURE SURVEY

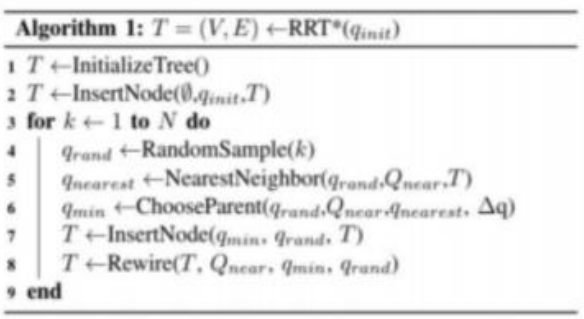
In [1], Gammell et al. discuss the application of RRT* towards path planning problems. It is demonstrated with the use of L2 norm how existing approaches tackling minimum-path-length problems give exponentially poor results as the state dimension increases. It also demonstrates how Informed RTT* can be utilized instead.

In [2], Gammell et al. demonstrate the advantages of the informed RRT* algorithm. This paper discusses how the method retains the same probabilistic features and optimality as RRT* at the same time increasing the convergence rate and quality of the final solution.

III. PROCEDURE

As for most sampling-based algorithms, a rapidly exploring random tree (RRT) efficiently searches non-convex, highly complex environments. An RRT works by, beginning from the "start point", gradually branching out to all nearby nodes as a tree. The nodes for the tree are selected randomly from the free available nodes available in the environment. The tree from this algorithm inherently selects nodes where more nodes are available and thus more towards large unsearched areas of the environment.

RRT* motion planning algorithm on the other hand although optimization of RRT differs from it by attempting to find the most optimal solution i.e. the shortest/least-cost path between the "starting" and "goal node". This is made possible by maintaining a "cost to starting node" value which is the cumulative sum of costs in the current path. RRT* algorithm then attempts to minimize the value. Figure 1 describes the pseudo-code 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

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

```
q_min ← q_nearest
c_min ← Cost(q_nearest) + c(q_rand)
for q_near ∈ Q_near do
    q_path ← Steer(q_near, q_rand, Δq)
    if ObstacleFree(q_path) then
        c_new ← Cost(q_near) + c(q_path)
        if c_new < c_min then
            c_min ← c_new
            q_min ← q_near
        end
    end
end
return q_min
```

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

```
for q_near ∈ Q_near do
    q_path ← Steer(q_rand, q_near)
    if ObstacleFree(q_path) and Cost(q_rand) + c(q_path) <
     Cost(q_near) then
        T ← ReConnect(q_rand, q_near, T)
    end
end
return T
```

Informed RRT* Motion Planning Pseudo Code:

We define the variables as:

- X_{free}: Set all available vertices/nodes in the environment
- V_{start} and V_{goal} : Start and Goal vertices/nodes
- V: Set of vertices/nodes in the solution path where $V \subseteq X_{free}$
- E: Set of edges/costs in the solution path where $E \subseteq X^2_{free}$

Using the above variables, we define the steps as:

1. Similar to RTT*(as seen in Algorithm 1 from Figure 1), the algorithm finds the optimal path between our extreme nodes V_{start} and V_{goal} and builds a tree T in the environment where $T = (V, E)$. However, unlike RRT* the nodes selected are not optimized for the minimum cost.
2. Once a tree T is formed i.e. a suboptimal path from V_{start} to V_{goal} , the algorithm searches for an optimal solution using an ellipsoidal heuristic.
3. Using V_{start} to V_{goal} as the focal points of a hyperellipsoid we create a rotation matrix and align it with our environment.
4. Using the above hyperellipsoid we look for the lowest cost nodes within the aligned regions and ignore the nodes outside.
5. Once a consequent optimized solution tree T_{new} is formed within the aligned region, we check if T_{new} equals T. If FALSE, we go to step 6 else go to step 7
6. We now have one optimal solution but to be sure it is the most optimized solution; we reduce the area of the hyperellipsoid and check for another solution path again. Then repeat step 5.
7. If reducing the area doesn't change the cost associated with the path implies that we have found the most optimal path in the environment.

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

```
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;
```

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

```
if c_max < ∞ then
    c_min ← ||x_goal − x_start||_2;
    x_centre ← (x_start + x_goal) /2;
    C ← RotationToWorldFrame(x_start, x_goal);
    r_1 ← c_max/2;
    {r_i}_{i=2,...,n} ← (√(c²_max − c²_min)) /2;
    L ← diag{r_1, r_2, ..., r_n};
    x_ball ← SampleUnitNBall;
    x_rand ← (CLx_ball + x_centre) ∩ X;
else
    x_rand ∼ U(X);
return x_rand;
```

IV. Implementation

A virtual environment with static obstacles was created using Gazebo. The environment had the following specifications:

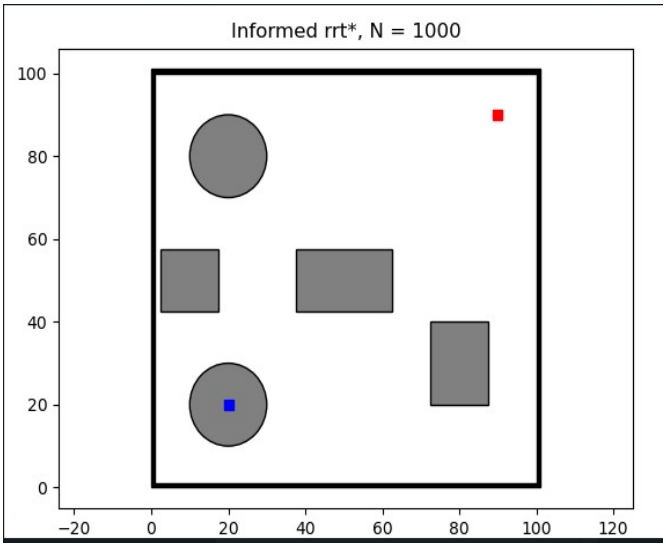
Using the environment set up we applied the Informed RRT* motion planning algorithm on ROS Turtlebot 3 to navigate through the environment to find the optimal path between the start point and the goal while avoiding all obstacles along the way.

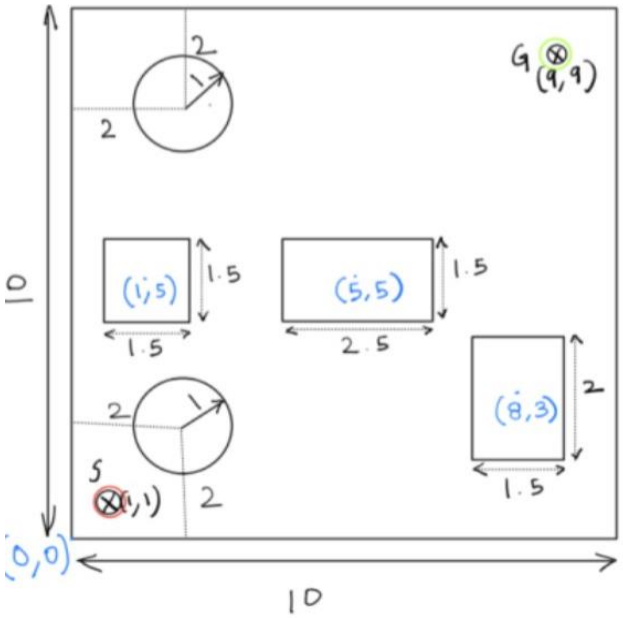
The angle between 2 vectors:

$$\theta = \sin^{-1}[|a \times b|] / (|a|\,|b|)].$$

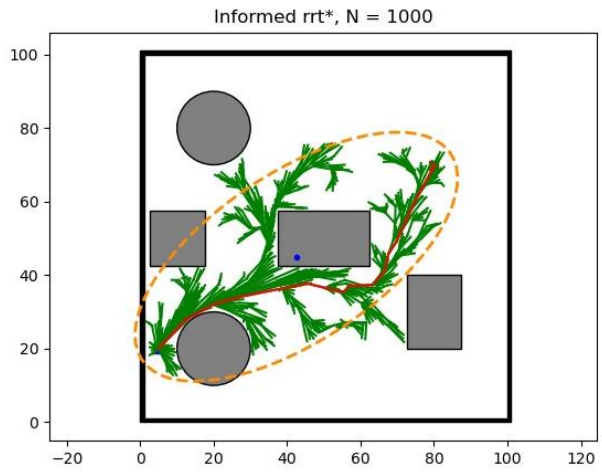
The distance between two points is also calculated in order to decide the direction of the robot.

The environment was reconstructed on Gazebo as follows:



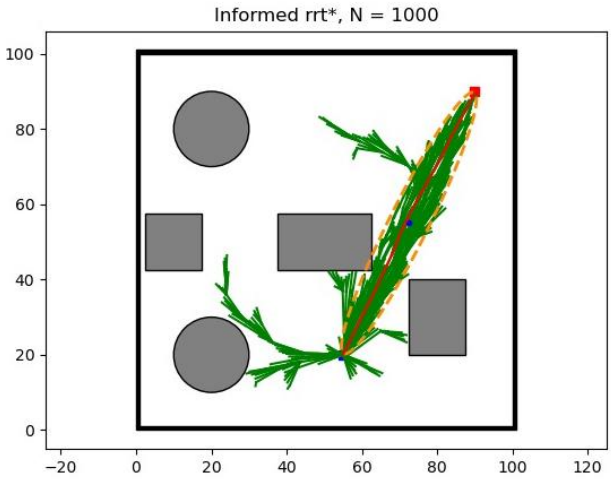


Starting point (5.20) and Goal point (80,70).

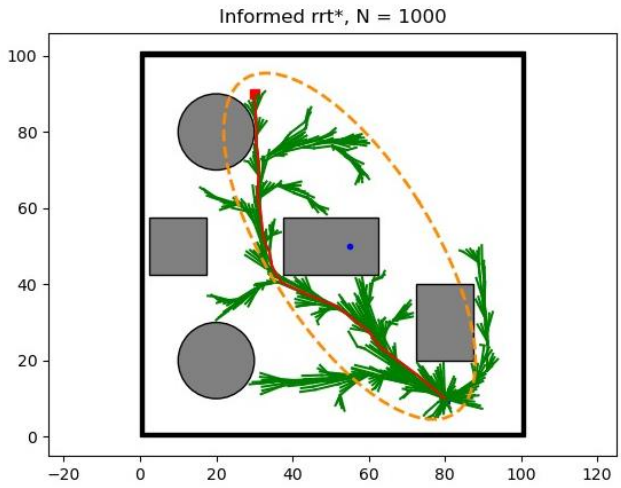


Based on the following Test Cases we go the respective solutions paths:

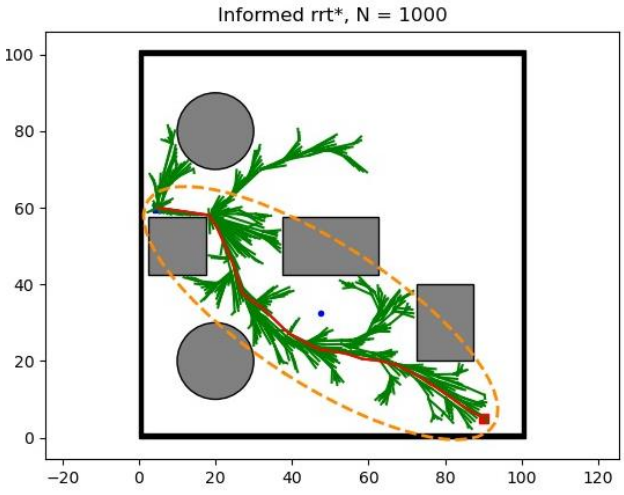
Starting point (50,20) and Goal point (90,90):



Starting point (80,10) and Goal point (30,90):

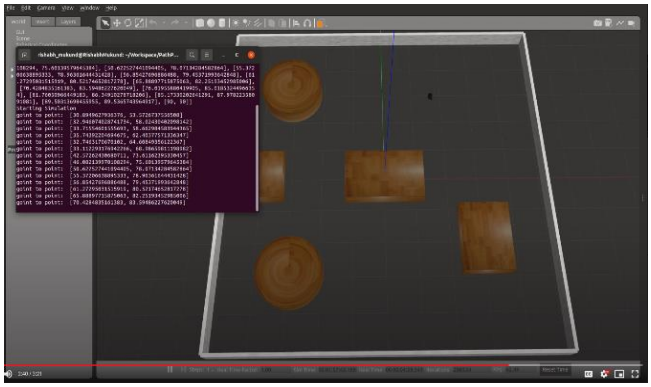


Starting point (5,60) and Goal point (90,5):



V. RESULTS

The Informed RRT* motion planning algorithm was used on ROS Turtle Bot 3 to navigate in a configuration space consisting of static obstacles. The map is loaded onto the gazebo. The turtle bot moves from the initial node to the goal node by avoiding the obstacles using the proposed algorithm.



VI. CONCLUSION

In this paper, we implemented Informed RRT* sampling-based motion planning algorithm. The algorithm was first tested with Python and then they are deployed on Turtlebot3 in a simulated environment. From the results of Python and simulation, we can tell that Informed RRT* is highly efficient compared to other algorithms like RRT* and RRT algorithms.

VII. REFERENCES

[1] Jonathan D. Gammell, Member, IEEE, Timothy D. Barfoot, Senior Member, IEEE, and Siddhartha S. Srinivasa, Fellow, IEEE "Informed Sampling for Asymptotically Optimal Path Planning"

[2] Jonathan D. Gammell1, Siddhartha S. Srinivasa2, and Timothy D. Barfoot1 "Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct A sampling of an Admissible Ellipsoidal Heuristic"

[3] S. M. LaValle and J. J. Kuffner Jr., "Randomized kinodynamic planning,"IJRR, 20(5): 378–400, 2001.

[4] L. E. Kavraki, P. ˇSvestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," TRA, 12(4): 566–580, 1996.

[5] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," IJRR, 21(3): 233–255, 2002.

[6] C. Urmson and R. Simmons, "Approaches for heuristically biasing RRT growth," IROS, 2: 1178–1183, 2003.

[7] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," IJRR, 30(7): 846–894, 2011.