Learning Cost-to-go Functions to Guide RRT* Algorithm for Manipulator

Pranav Patil (34790252), Rucha Deshpande (34780087)

Abstract—**Motion Planning Algorithms need to be adaptive to complex environments for practical applications. RRT* is a conventional sampling-based algorithm that attempts to find an optimal obstacle-free path. However, it faces challenges as the complexity of the planning problem rises. The quality of returned solutions is affected in more complex environments. We aim to reduce the overall cost of the path by considering both the cost to goal along with the cost from the start while adding a given point to the tree. In this project, we attempt to guide RRT* algorithm by using a neural network to predict the cost to goal from a point. We use this cost to add points to the tree that reduce the cost-to-go compared to its near neighbors.**

Index Terms—**Path planning, Robots, Motion planning, RRT*, Sampling-based planning**

I. Introduction

Motion planning is a crucial aspect of practical robotics applications. Conventional algorithms like Rapidly Exploring Random Trees achieve good results under specified conditions. However, these algorithms tend to become less efficient in terms of convergence time and quality of the solution with increasing environment's complexity. In RRT* algorithm, the role of rewire function is to check if the newly added node to the path serves as a better parent to already existing near nodes. Since RRT* is a random sampling-based approach, it explores the space till it finds the goal. However, this random exploration can be costly in large and complex environments, as the new nodes can be sampled in a direction away from the goal. RRT* does not take into account whether the sampled node is leading the tree toward the objective or not; instead, it optimizes the path based on the random samples it generates. Hence, the path obtained might not be the best or most optimal path. In this project, we address this issue by building a modified RRT* motion planner that aims to optimize the cost of the paths using a directed sampling approach. Instead of using the 'rewire' function that only optimizes the cost from the start node to an intermediate node, we also utilize the cost from the intermediate node to the goal node. We use a neural network to predict this cost to the goal and guide the RRT* planner towards the goal.

II. Background

A. RRT* - Rapidly Exploring Random Trees

Rapidly Exploring Random Trees [6] are a sampling-based method to compute efficient motion plans. An RRT grows a tree rooted at the starting configuration by using random samples from the search space [2]. A connection is attempted between each drawn sample it and the near nodes in the tree. If a collision-free path connecting the near nodes and the sampled node is feasible, the sampled node is added to the tree. For each connected node, the 'rewire' function is performed which checks if the cost to the nodes in the nearest neighbors is less through the newly added node as compared to their older costs. If less cost path is found, the near neighbor's parent is changed to new node.

P. Patil and R. Deshpande were with the Department of Computer Science, Purdue University, West Lafayette, IN, 47906 USA e-mail: patil127@purdue.edu, deshpa37@purdue.edu.

B. PRM - Probabilistic Road Maps

The probabilistic roadmap, PRM [4] is another sampling-based motion planning algorithm, which solves the problem of determining a path between a starting configuration of the robot and a goal configuration while avoiding collisions. The algorithm runs in two phases, a learning phase and a query phase. In the learning phase, a probabilistic roadmap is constructed and stored as a graph. For creating this graph, random samples are taken from the environment and tested if they lie in the free space. If they do, near nodes within a fixed radius of the sampled nodes are found. The sampled nodes are then connected to the graph if there exists a collision-free path joining them with the near nodes. In the query phase, any given start and goal configurations of the robot are connected to two nodes of the roadmap; the roadmap is then searched for a path joining these two nodes. In our implementation, we use a modified version of PRM i.e PRM* which uses a variable radius for finding nearest neighbors. The radius is determined based on the number of nodes and the number of dimensions.

III. Proposed framework

In traditional RRT*, we take into consideration only the cost from the start to the current node. In order to do so, we pick the best parent for a node that minimizes the distance from the start. Following the parent assignment, we also 'rewire' the nodes that are near the random node by checking if the new node can be a better parent, and changing the parent (the wiring) as needed. While this is an effective way to find optimal paths, it doesn't consider the cost that the newly added node will need to reach the goal. The path obtained by considering both the cost from start-to-node and the cost of node-to-goal is expected to give more optimal paths. We aim to reduce the overall costs by considering both these costs.

We divide our project into three phases: Environment Setup and Dataset Generation, Learning Component - Neural Network to predict Cost-to-go, Modification of RRT*- Guiding

RRT* using ‘Cost-to-go’. In the following sections, we describe each of these phases.

A. Environment Set up and Dataset Generation

We set up 2D and 3D environments for motion planning with 7 obstacles in each environment. More information about the environments is given in section IV. For training the model, we use a dataset generated using PRM*. We extract waypoints and the cost from the waypoint to the goal from the shortest path between queried start and goal pairs. We create our dataset with all such points and their costs in different environments.

1) PRM Implementation:* We generate a network graph of n nodes in 2D and 3D environments using PRM*. The PRM* algorithm is as follows:

Algorithm 1 PRM*

```
Initialize $G = (V, E)$
for i=0 to N do
    $q_{rand} \leftarrow sample()$
    $N_q \leftarrow select\_nearest(q_{rand}, G)$
    for all $q' \in N_q$ do
        if $(q_{rand}, q') \in E$ && $steerA(q_{rand}, q')$ then
            $E \leftarrow E \cup \{q_{rand}, q'\}$
        end if
    end for
end for
```

The radius used for selecting the nearest neighbors is given by

$$\gamma = \left(\frac{log(n)}{n}\right)^{\frac{1}{d}}$$

where, γ is a constant, n is the number of nodes currently added to the graph and d is the number of dimensions.

2) Shortest Path Algorithm: We sample obstacle-free start and goal nodes and compute the shortest path using Dijkstra’s Shortest Path Algorithm [1]. The sampled nodes might not always be the nodes in the graph, so we find the shortest paths between all combinations between k nearest nodes for the start and goal and choose the pair giving the shortest obstacle-free path. We then add the sampled start and goal node to the path.

B. Learning Component - Neural Network to predict Cost-to-go

We train a neural network to predict the cost to goal for a given node in the given environment. The network comprises an encoder and a MLP. The environment information is given in the form of a point cloud representation of obstacles. Since the point-cloud representation is large, the current node and goal node get lost in the environment. Thus, we reduce the size of the point-cloud representation to a smaller vector using an encoder model. This vector is further processed in addition to the current node and the node goal using a deep MLP with cost-to-go as the output target. We train the model using MSE Loss.

C. Modification of RRT- Guiding RRT* using ‘Cost-to-go’*

In this phase, we modify the RRT* algorithm. The original RRT* algorithm contains a ‘rewire’ function to check if the generated sample node can be a better parent of its near nodes and reduce the overall cost. In the modified RRT* algorithm, we use a guided sampling technique. We utilize cost-to-go as a heuristic to guide RRT* algorithm. When we sample a node, we add it to tree the only if it is nearer to the goal as compared to its nearer neighbors. For every sampled node, we obtain the predicted cost-to-go from the trained model. This cost is compared to the cost-to-go from the near nodes for this node. If the cost-to-go of the sampled node is less than the minimum of that from the near nodes, it means that we have taken a step toward our goal. We then add this point to the tree, else we discard the point and sample a new point. Doing this for every sampled point, we make sure that while we are randomly exploring the space, we add another node only if we are moving towards the goal from the already added nodes in the tree. Algorithm 2 describes the guided sampling approach.

Algorithm 2 Guided Sampling

```
$cost_{near} \leftarrow \infty$
while $iter \leftarrow 0 \cdots MaxIter$ do
    $q_{rand} = sample()$
    $N_q \leftarrow select\_nearest(q_{rand}, G)$
    for all $q' \in N_q$ do
        $cost_{near} = min(cost_{near}, cost\_to\_go(q'))$
    end for
    if $cost\_to\_go(q_{rand}) < cost_{near}$ then
        return $q_{rand}$
    end if
end while
return $q_{rand}$
```

We use the guided sampling approach in the modified RRT* planner. We also remove the rewire function since we already take care that the newly added node is closer to the goal node than its parent (through guided sampling).

Algorithm 3 Modified RRT* planning

```
$rnd \leftarrow generateDirectedSample()$
$nind \leftarrow getNearestListIndex(nodeList, rnd)$
$valid, cost \leftarrow steerTo(rnd, nodeList[nind])$
if valid then
    $newNode \leftarrow rnd$
    $newNode.parent \leftarrow nind$
    $newNode.cost \leftarrow rnd_{cost} + nodeList[nind].cost$
    $nearinds \leftarrow find\_near\_nodes(newNode)$
    if $isNearGoal(newNode)$ then
        $solutionSet \leftarrow newNode$
        $getMinPath(solutionSet)$
```

IV. Implementation Details

This section presents the details of the environment setup and model parameters.

A. Environment Setup

1) 2D Environments (Point Robot):

- Number of obstacles = 7
- Sizes of obstacles = Variable

Figure 1 shows PRM* graph with 100 nodes for 2D environment.

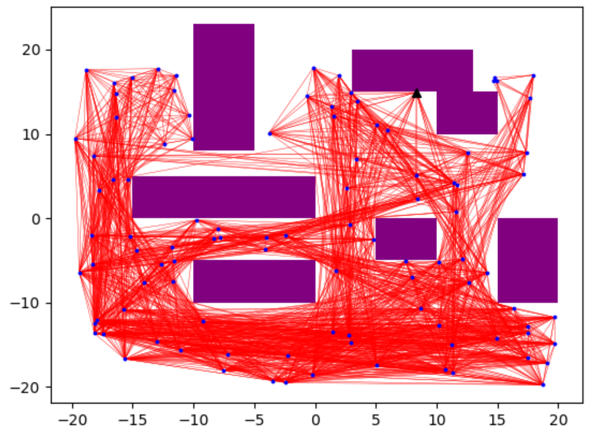


Fig. 1: PRM* Graph for 2d Environment (100 Nodes)

2) 3D Environments:
The 3D environment is setup in pybullet with a 3-DOF UR5 manipulator.

- Number of obstacles = 7
- Sizes of obstacles = $0.2 \times 0.2 \times 0.2$

Figure 2 shows PRM* graph for 3D environment.

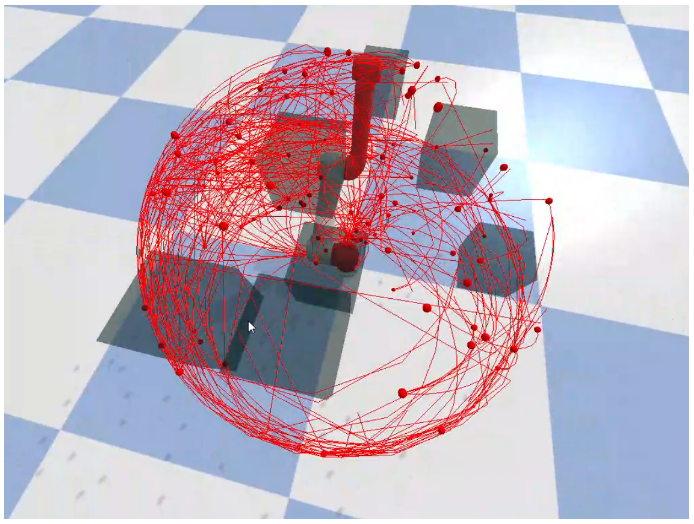


Fig. 2: PRM* Nodes for 3d Environment

B. Dataset Generation

Fig. 3 shows a shortest path for randomly chosen start-goal pair in the PRM*-graph. When we query random start and goal pairs in PRM*-generated graphs, the points might lie in free space (not exactly on the graph nodes). To find the shortest path between these nodes, we select k nearest nodes for the start and the goal. We find the shortest paths between all start and goal combinations from the k nearest nodes and choose the pair that gives the least cost path. Then we connect the start and goal with near-start and near-goal nodes respectively from the chosen pair. This is illustrated in Fig. 4. Size of dataset = 200,000+ for both 2D and 3D

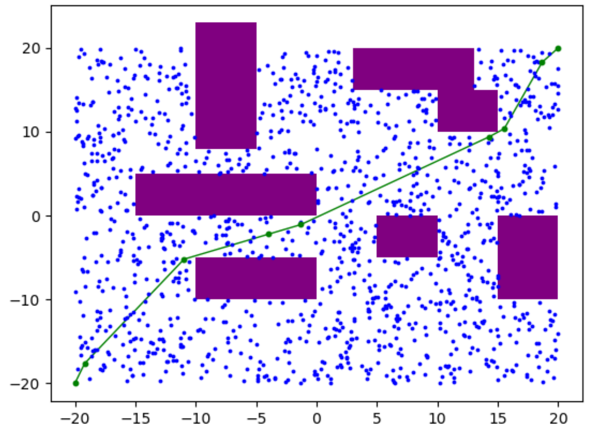


Fig. 3: Shortest path from a start and goal configuration (two corners of environment in this case, cost = 58.96)

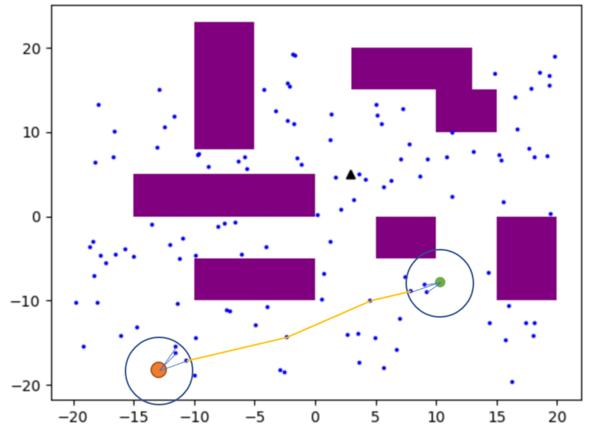


Fig. 4: Finding shortest path from k neighbors near start and goal in the graph

C. Model Parameters

The following are the model parameters used:

- Input size
  - 2D: 2800 + 4, Encoder reduction: $2800 \rightarrow 28$
  - 3D: 4200 + 6, Encoder reduction: $4200 \rightarrow 21$
- Output Size = 1 (Cost-To-Go)
- Loss = MSE
- Activation = ReLU

Figure 5 shows the model structure.

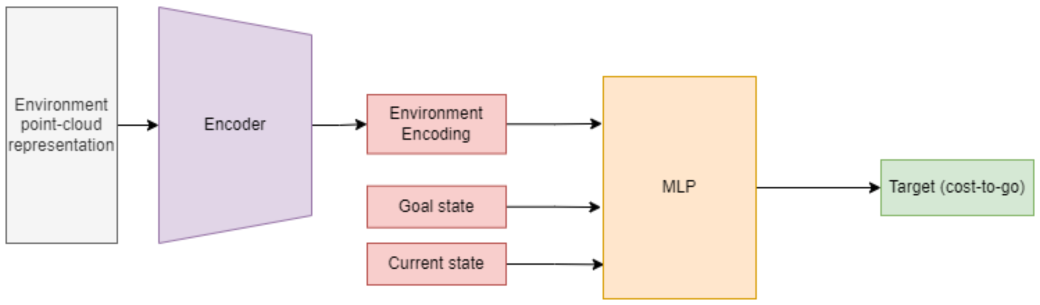


Fig. 5: Model Structure

V. Results

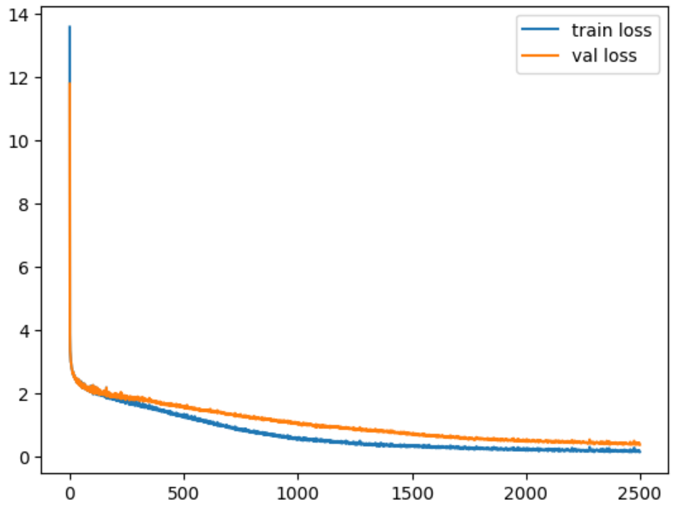


Fig. 6: Training curve for 2D

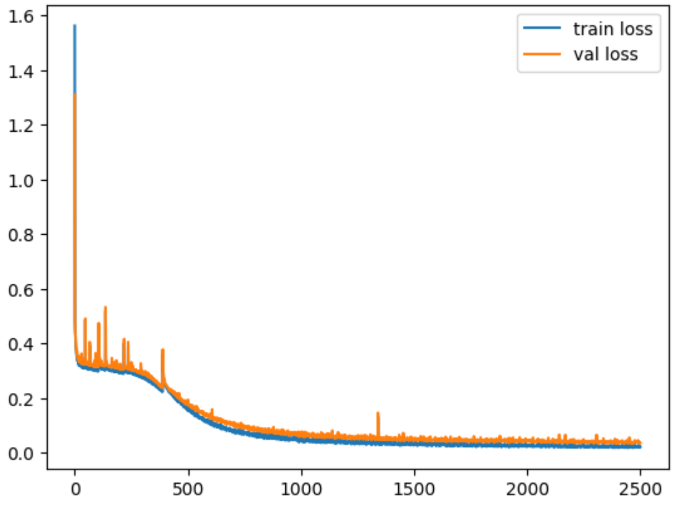


Fig. 7: Training curve for 3D

To evaluate the guided RRT* method, we consider the following parameters:

1) Cost
2) Time to find a path
3) Success Rate

We compare our approach of RRT* with directed sampling with the original RRT* algorithm. For a fair comparison of cost and time, we only consider the cases where both methods succeed in finding the path. Table I shows the results for 2D environment.

We can see from the results that the average cost of the original RRT* algorithm is slightly better than the average cost of directed RRT*. However, the success rate of guided RRT* is better than the success rate of conventional RRT*. This is predominantly observed in more complicated environments where there are many obstacles in the path. The conventional RRT* randomly explores the space, hence the probability of reaching the goal is very low for given number of iterations. In the case of guided RRT*, the new nodes advance in the direction of the goal thus increasing the probability of finding a path. This is demonstrated in the Figures. 8 and 9 (and additional cases Figure 10 to 14). We can see that the tree is dense in the direction of the goal and there is a fan-out structure created by children nodes from a parent node in the direction of the goal. Whereas, the tree in the original RRT* is randomly growing over the search space. The average time required for directed RRT* to find the path is also less compared to the time required for original RRT*. Since the guided RRT* only advances in the direction of the goal and does not involve additional computations like rewiring, the time for finding the path is reduced.

Table II shows results for 3D. In 3D environments, the model performs well and succeeds in giving a better average cost. The success rate is slightly greater than the original RRT* but the time needed is reduced considerably. The success rates are low because 3D environments are more complex with more obstacles in a small space and computational limitations while generating results due to which we had to limit the number of iterations. However, the cost-to-go model has converged well and is able to find a better path successfully in lesser time than the original RRT*.

The visualizations for 3D environment are not included in the report because the cost is determined by the robot joint angles unlike the actual distance between points in the case of 2D environment. So the join angles move towards goal configuration and showing a trend in the workspace is not feasible.

Parameter	Original RRT*	Guided RRT*
Cost	25.57	26.50
Time	15.08	13.08
Success Rate	0.92	0.95

TABLE I: Results for 2D

Parameter	Original RRT*	Guided RRT*
Cost	5.864	5.612
Time	4.575	2.971
Success Rate	0.40	0.45

TABLE II: Results for 3D

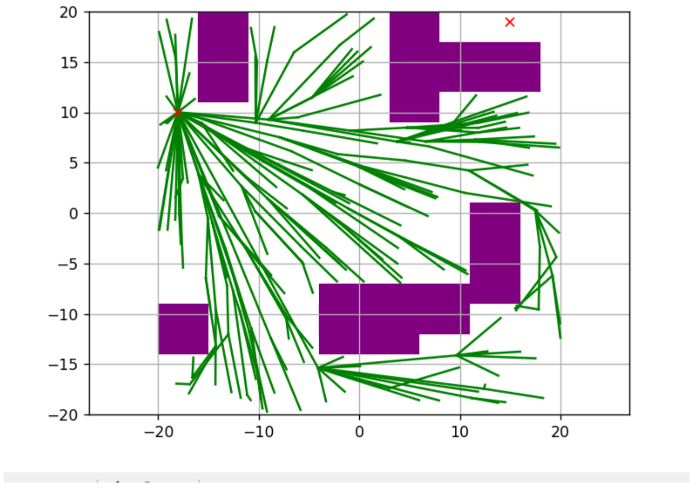


Fig. 8: OriginalRRT*

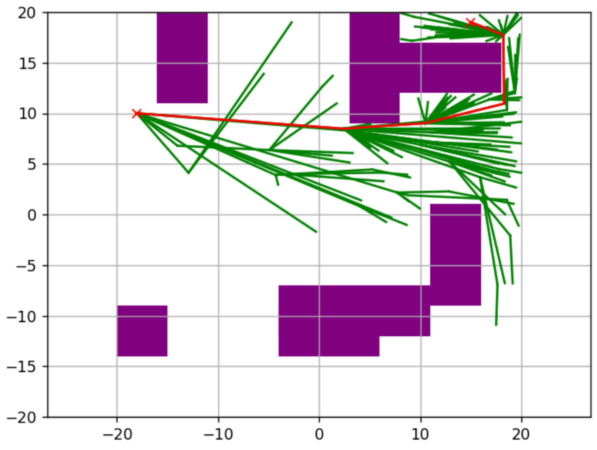


Fig. 9: GuidedRRT*

VI. Discussions and Conclusion

In this project, we built a modified RRT* motion planner that utilizes the cost-to-go heuristic to guide the robot toward the goal. We built the planner for both 2D and 3D environments to better visualize the results. We modified the RRT* algorithm using a guided sampling method where the new sampled node is added to the tree only if it is in the direction of the goal. We check this by using the predicted cost-to-go from our neural network which takes as input the environment details, the robot's current state and the goal state.

From the results, we can see that the cost-to-go function is able to direct the robot toward the goal. As we can see in the figures, the density of nodes sampled is more as the points go towards the goal i.e. the guided sampling is able to nudge the robot toward the goal.

To further build upon this we can use encoding techniques to find a good encoding for point cloud. Currently, we are using the dataset for learning the encoding. However, we would like further to experiment using variational autoencoders[5] and GAN[3]t to encode the point cloud and train the cost-to-go function using the learned encoding. We can also use path smoothening to get a better path.

VII. Statement of contribution

1) Pranav Patil
   - PRM* Implementation
   - Environment Generation
   - Experiments with model and tuning
   - Modified RRT*
   - Visualization and Results Generation
2) Rucha Deshpande
   - Environment Setup
   - Dataset Generation
   - Experiments with model and tuning
   - Modified RRT*
   - Visualization and Results Generation

References

[1] Dijkstra's algorithm. https://en.wikipedia.org/wiki/Dijkstra

[2] Rapidly-exploring random tree. https://en.wikipedia.org/wiki/Rapidly-exploring$_random_tree$.

[3] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

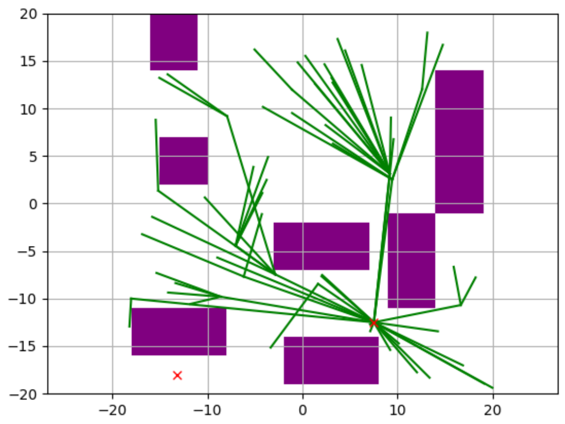
[4] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[5] Diederik P Kingma, Max Welling, et al. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019.

[6] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

Appendix

The following images show comparison between original and directed RRT* on 2D environments.


(a) $Original RRT*$

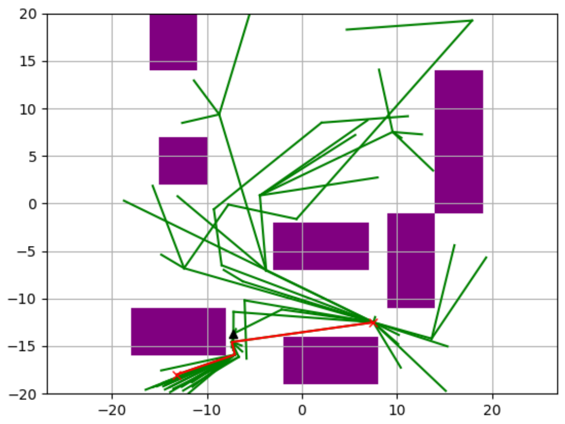
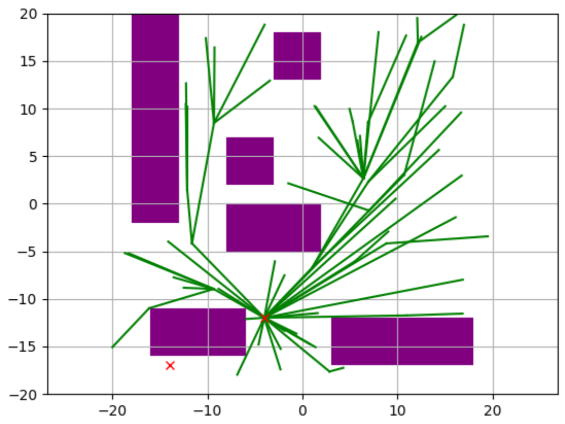

(b) $Guided RRT*$

Fig. 10: Case 1


(a) $Original RRT*$

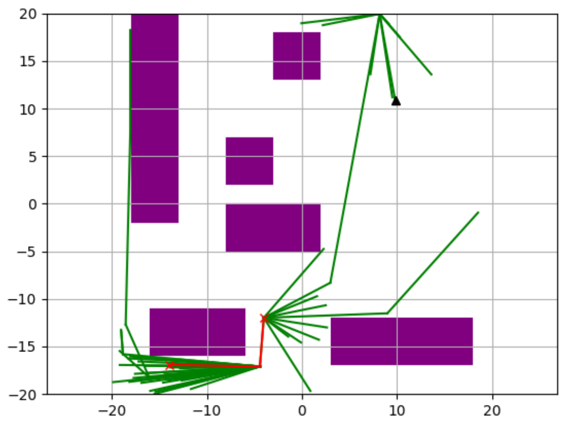
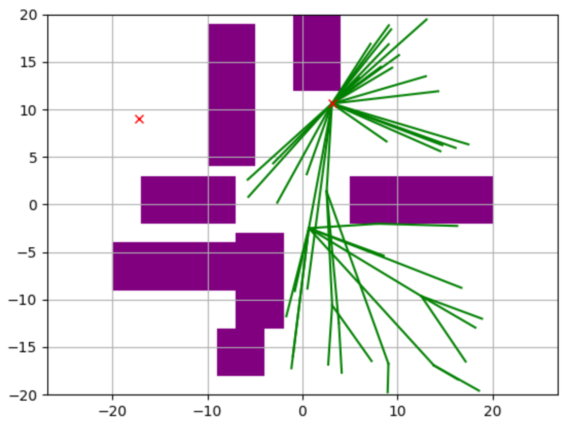

(b) $Guided RRT*$

Fig. 11: Case 2


(a) $Original RRT*$

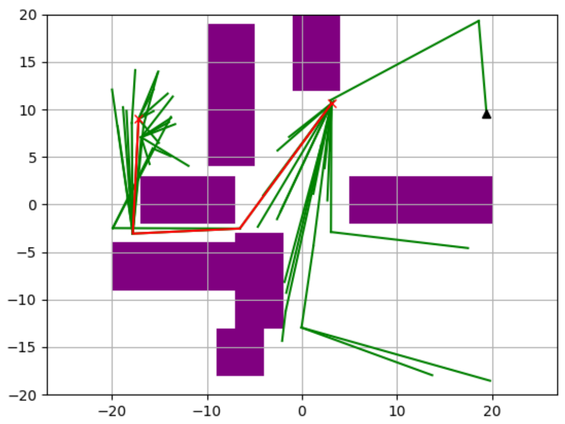
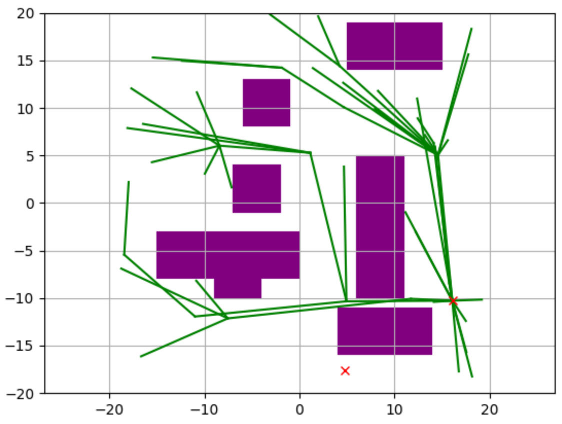
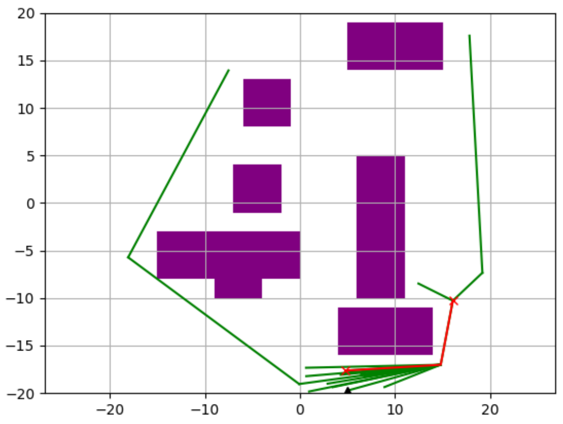

(b) $Guided RRT*$

Fig. 12: Case 3

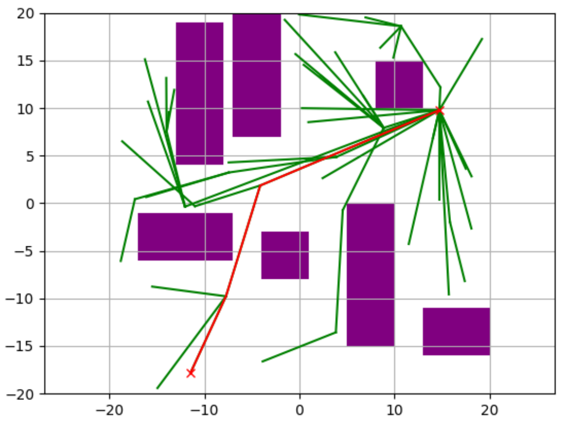


(a) $Original RRT*$

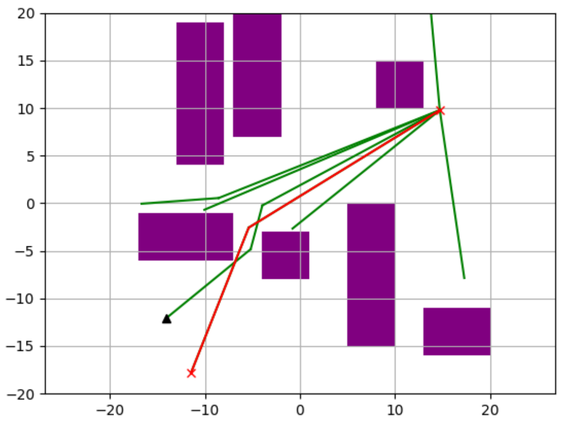


(b) $Guided RRT*$

Fig. 13: Case 4



(a) $Original RRT*$



(b) $Guided RRT*$

Fig. 14: Case 5