Learning cost-to-go functions to guide RRT* algorithm for manipulator

Team: Pranav Patil (034790252),
Rucha Deshpande (034780087)

Problem Statement

- Motion Planning Algorithms need to be adaptive to complex environments for practical applications.
- Conventional algorithms like Rapidly Exploring Random Trees achieve good results, however become less efficient with increasing number of nodes in the tree.
- In RRT* algorithm, the role of rewire function is to improve the optimality and efficiency of the tree by adjusting the connections between nodes in the tree such that they represent the shortest paths possible.
- However, while adjusting the connections in RRT*, we only consider the cost from start-to-node.
- In this project, we aim to optimize RRT* further by guiding RRT* with a '**Cost-to-go**' function that considers the cost from **start-to-node + node-to-goal** while optimizing the path.
- The 'Cost-to-go' is obtained as the output of a neural network given the intermediate node, goal node and obstacles as input.

Pipeline

Environment Setup and Dataset Generation	Learning Component - Neural Network to predict Cost-to-go	Modification of RRT*- Guiding using ‘Cost-to-go’

1. We create different environments using point cloud representation of obstacles.

2. We use PRM* to generate the roadmaps that can be used to query start-goal pairs and obtain shortest paths. These start-to-goal paths can then be used to extract node-to-goal pairs and their costs for training the model.

We train a neural network by giving inputs the obstacles represented by point clouds and the node-goal pairs. The labels will be the minimum costs corresponding to the node-goal pairs.

The trained model is used to modify RRT*. We only add a sampled node to the tree if its cost-to-go is less than the minimum cost-to-go of it’s near nodes. This approach will ensure that we add only those nodes to the tree that are in the direction of the goal.

Timeline

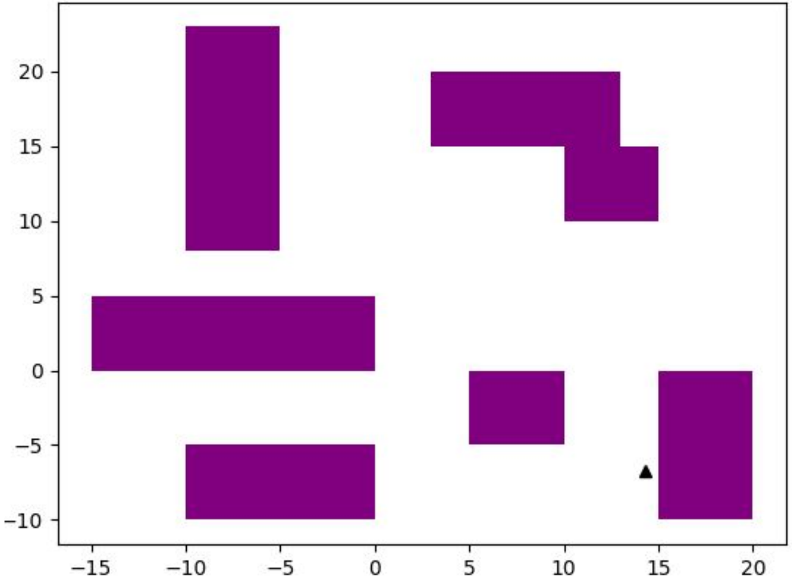
Milestone 1	Milestone 2	Milestone 3
Setup of UR5 robot with obstacles in Pybullet	Running PRM* to generate a roadmap and query different start and goal to extract waypoints and cost-to-go from each waypoint	Training a neural network to predict cost-to-go and use it to guide RRT* rewiring function

Milestone 1

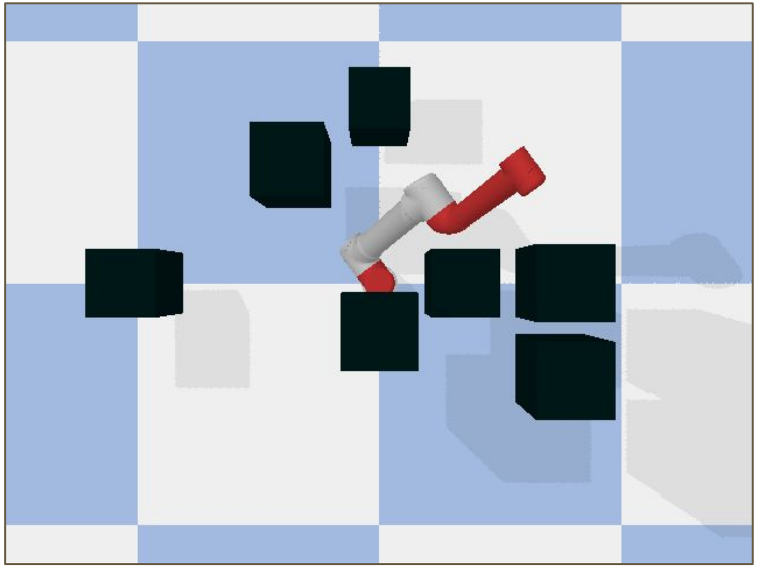
- Environment Setup
- PRM* Implementation
- Framework for Dataset Generation

Milestone 1 - Environment Setup

1. 2D Environments (Point Robot):
   a. Number of obstacles = 7
   b. Sizes of obstacles = Variable



2. 3D Environments (3DOF UR5 Manipulator):
   a. Number of obstacles = 7
   b. Sizes of obstacles = $0.2 \times 0.2 \times 0.2$



Milestone 1 - PRM*

PRM* Implementation - Created network graph of 4000 nodes in 2D and 3D environment.

PRM* algorithm

```
Initialize G = (V, E)
for i=0 to N do
     qrand ← sample(i)
     Nq ← select_nearest(qrand, G)
     for all q′ ∈ Nq do
          if (qrand, q′)∈E && steerA(qrand,q′) then
               E ← E ∪ {qrand, q′}
          end if
     end for
end for
```

Radius for near nodes- $r = \gamma * (\frac{log(n)}{n})^{\frac{1}{d}}$

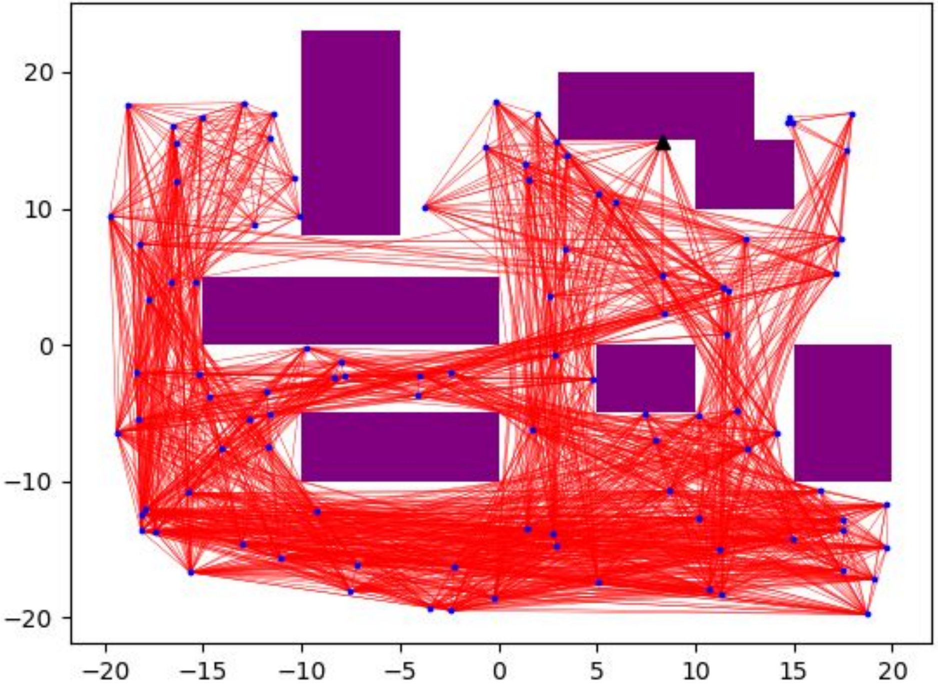


Fig. PRM* in 2D environment

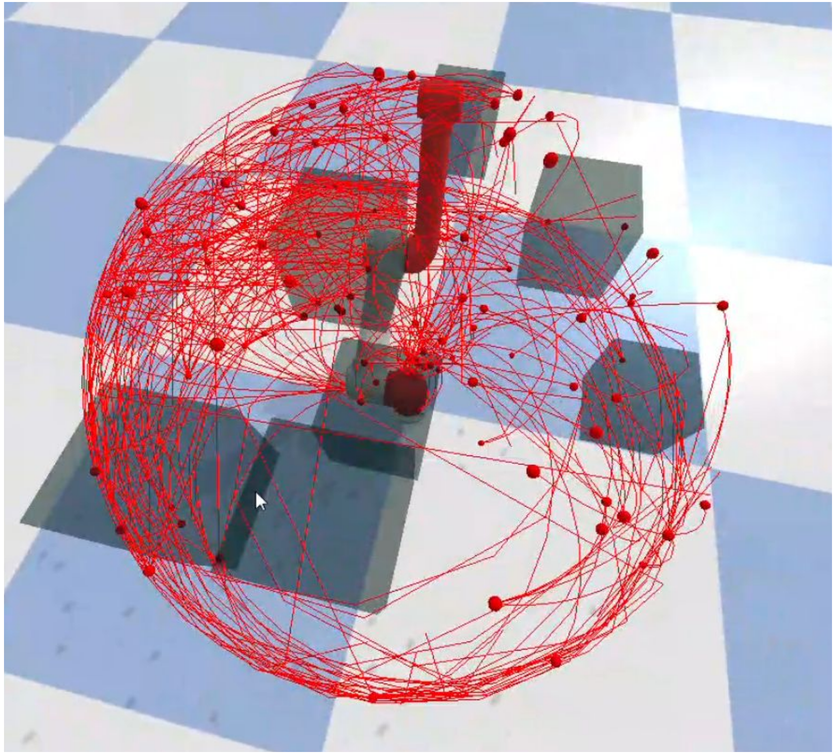


Fig. PRM* in 3D environment

Milestone 1 - Framework for Dataset Generation

We use Dijkstra’s Algorithm to obtain shortest path between queried start and goal. We then use these paths to extract node-to-goal pairs for creating dataset.

Dijkstra’s Shortest Path Algorithm

```
function Dijkstra(Graph, source):

for each vertex v in Graph.Vertices:
     dist[v] ← INFINITY
     prev[v] ← UNDEFINED
     add v to Q
dist[source] ← 0
while Q is not empty:
     u ← vertex in Q with min dist[u]
     remove u from Q
     for each neighbor v of u still in Q:
          alt ← dist[u] + Graph.Edges(u, v)
          if alt < dist[v]:
               dist[v] ← alt
               prev[v] ← u
return dist[], prev[]
```

Milestone 1 - Framework for Dataset Generation

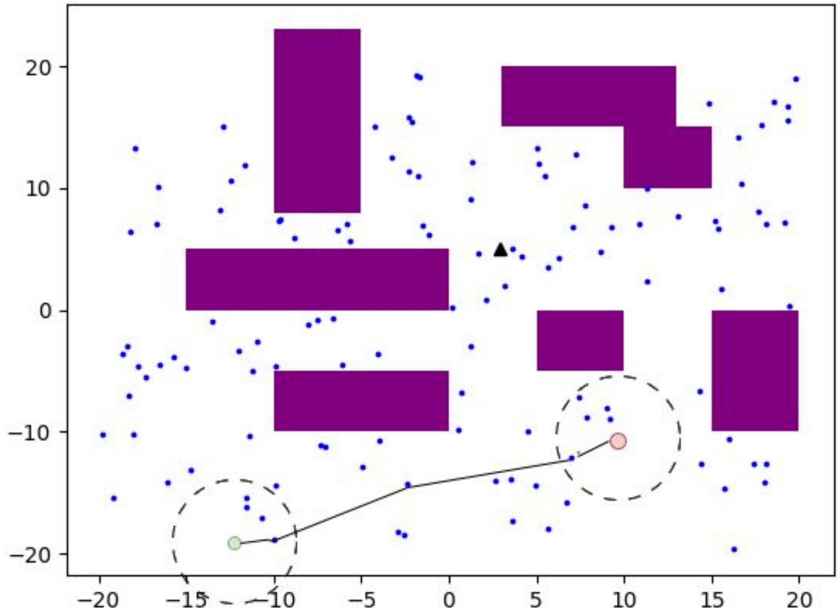


Fig. Obtaining path between a start-goal pair

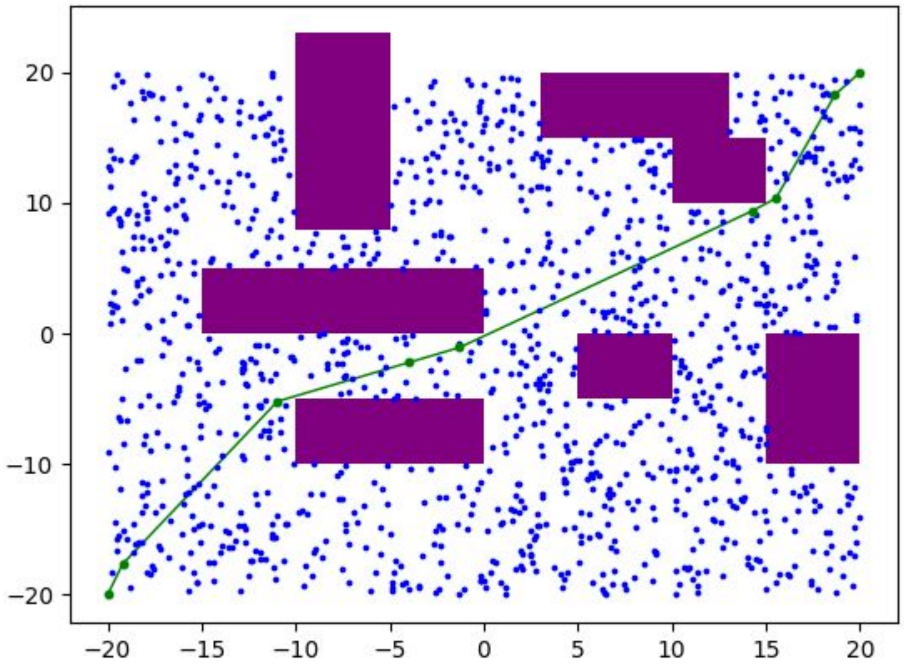


Fig. Shortest Path in 2D environment

Milestone 2

- Dataset Generation
- Model Training and Experiments

Milestone 2 - Dataset Generation

- Created 10 environments each for 2D and 3D cases.
- Converted obstacles to point clouds with 200 uniformly sampled points in each obstacle.
- Randomly sampled start and goal nodes to obtain shortest paths.
- Extracted costs from node to goal for every node in the shortest path.

Size of Dataset -

- 2D: 200000+ samples.
- 3D: 200000+ samples.

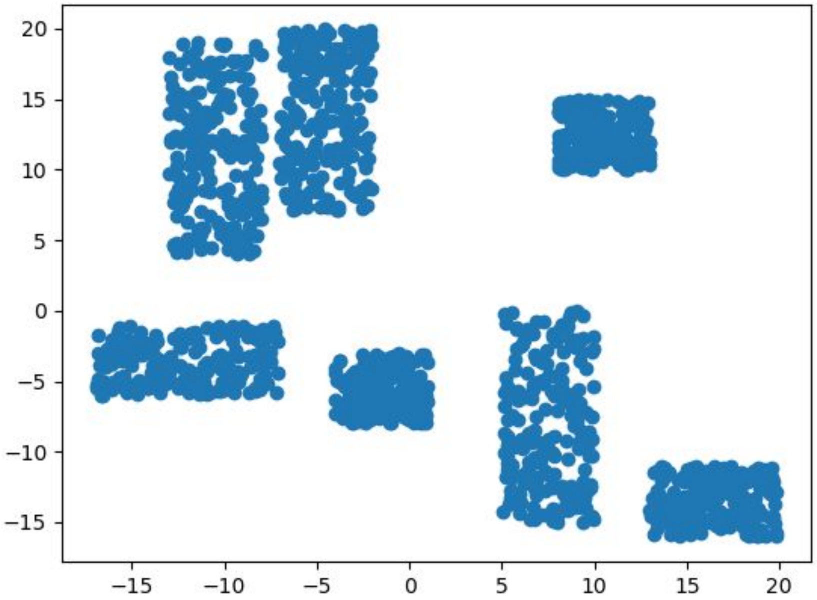


Fig. Point Cloud Visualization in 2D environment

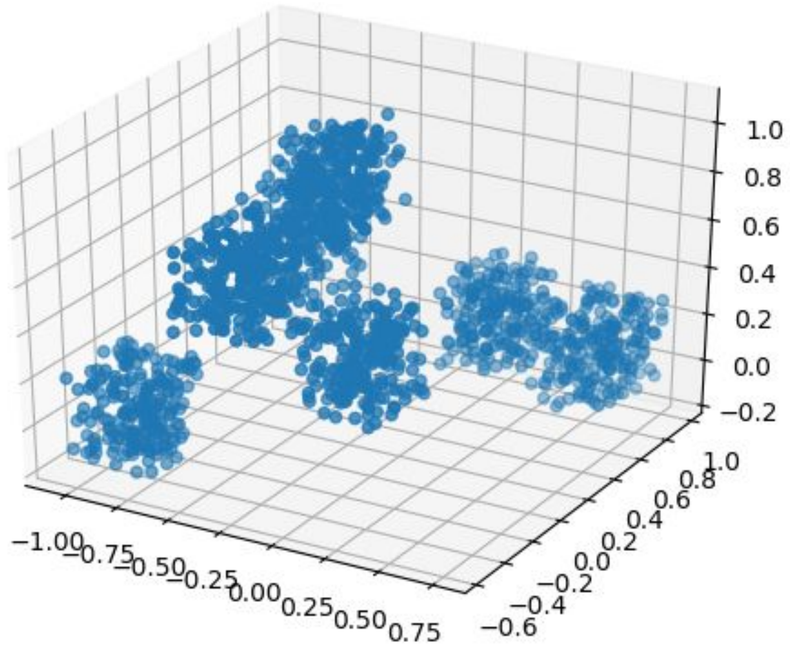


Fig. Point Cloud Visualization in 3D environment

Milestone 2 - Model

Input size = 2800 + x (x = 4 for 2D, x = 6 for 3D)

Output Size = 1 (Cost-To-Go)

Loss = MSE

Activation = ReLU

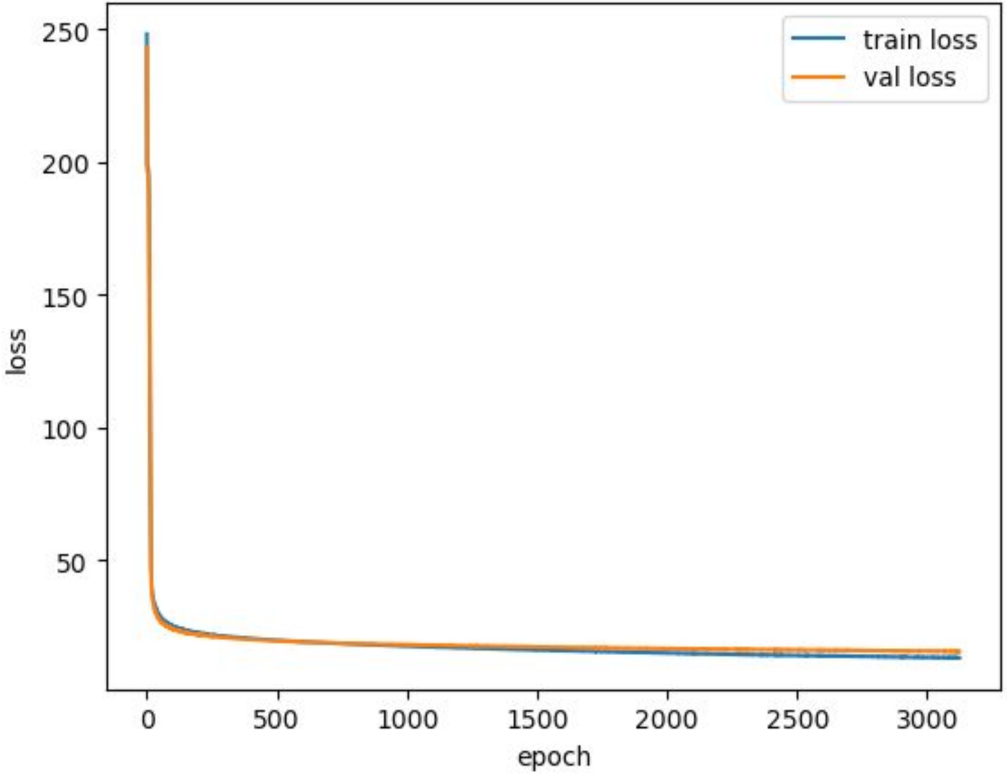
Batch Size = 1024, tuning on-going with different batch sizes.

Milestone 2 - Model

Experiments on Model:

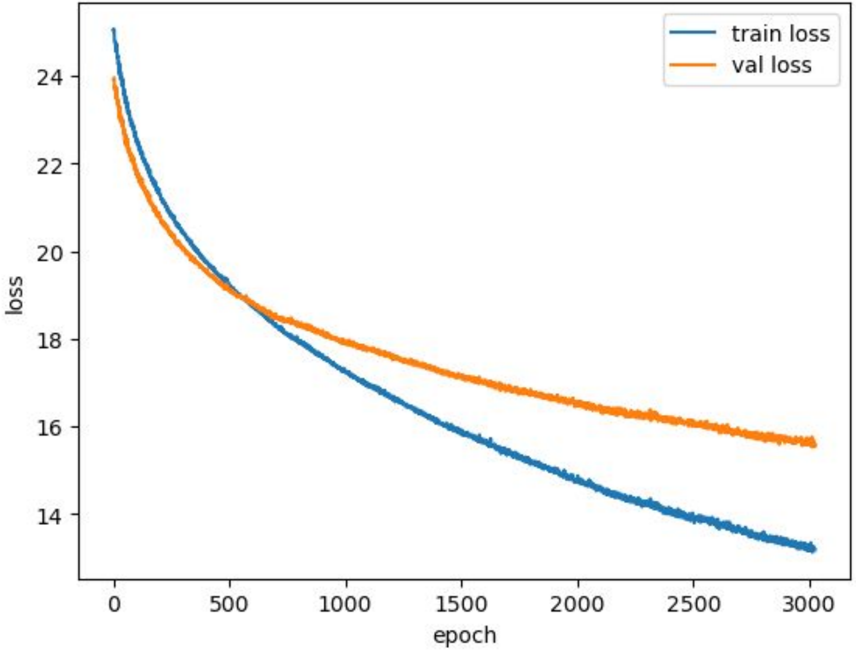
- Batch Sizes: 32, 64, 128, 256, 512, 1024, 2048
- Learning Rates: 0.1, 0.01, 0.001, decrease on plateau, steady decay
- Different Hidden Layers, with and w/o dropout.
- Using Encoder and MLP, one complete model.
- With and Without Starting Point.

Milestone 2 - Model for 2D: Results for Current Model



Milestone 2- Current Challenges

- Although the loss seems to be plateaued, it is constantly decreasing.



Milestone 2- Current Challenges (Cntd.)

- Tuning Hyperparameters:
    - Batch Size
    - Learning Rate
    - Model Architecture (hidden layers)
    - Testing with different optimizers to avoid local minima
    - Dataset Size.
    - Regularization on overfitting by introducing dropouts.

Summary - M1 + M2

We have achieved the following so far

1. Setup 2D and 3D environment with different obstacles for getting diverse training data.
2. Generated PRMs for in environments of 4000 nodes.
3. Obtained minimum costs by randomly sampling 5000 pairs of start and goal points in each environment.
4. Generated dataset from point clouds of obstacles and above obtained start-goal pairs.
5. Trained model and performed experiments .

Milestone 3

We plan to cover the following in Milestone 3

1. Overcoming current challenges (tuning models for 2D and 3D)
2. Complete the remaining pipeline with modified RRT* function that considers **Cost-to-arrive + Cost-to-go,** where cost-to-go is obtained from trained models.
3. Final results

Contribution

Pranav Patil

- PRM* Implementation
- Environment Generation
- Experiments with model and tuning

Rucha Deshpande

- Environment Setup
- Dataset Generation - Shortest Path algorithm, querying data points
- Experiments with model and tuning

Conclusion

We generated PRMs and sampled dataset from randomly generated start-goal pairs. We trained the model to obtained cost-to-go and have been able to reduce the MSE loss to 12.88. We still need to tune the models to achieve a lower value of loss in order to get optimal performance with RRT*.