PRM for Data Generation for Better Informed Learning Based RRT* Motion Planning

Yusuf Jarada, *Purdue University*

I. Introduction

SAMPLING based motion planners have proven to be a simple and effective solution to most motion planning problems. Among these methods, single query planners, such as Rapidly expanding Random Trees, (RRT) and its optimized revisions (RRT*), and multi-query planners such as Probabilistic Road Maps (PRM), have been explored thoroughly. This paper aims to utilize both algorithms via Machine Learning, to make a motion planner better than RRT* and the use of PRMs. It aims to use PRMs to generate waypoints between a given start configuration, goal configuration, and environment, and calculate the cost of each waypoint about the goal configuration. Then, we would train a model to estimate the cost of a given point about a given goal configuration and environment, After this we would hopefully be able to develop a very accurate RRT* Algorithm that rewires its path samples based on the cost to the start and the cost to the goal. So in summary, this project has 3 parts. First, we need to implement a PRM solution to the motion planning problem using the UR5 robot in Pybullet and record the waypoints and costs for each waypoint. Second, we need to train a model on the data collected from the PRM solutions and predict the cost to goal for a given waypoint. Finally, we will implement an improved RRT* motion planner that would consider the cost to goal for a given sample on the tree. This would not be possible without the model which is why we need steps 1 and 2 to complete this part accurately.

II. Methods

Here we will discuss how we set up the learning-based RRT* motion planner by explaining it in 3 parts. The PRM data collection, the heuristic model, and finally the RRT* planner that brings them all together. We implemented all code using the Pybullet and Gym Python environments. The code was set up on different environments using the UR5 robot to test the planner. All learning was done by using the Pybullet library.

To implement the classical component of this project, the PRM solution, we first need to sample a collision-free configuration to add to the graph. This was implemented in the sampling function, where a given sample was checked for collision with the geometry. Next, we need to steer between two points and return true if the steering is collision-free. this was done in the steering function. The steering function interpolates intermediate points between two distinct samples and checks if there is a collision in any of these interpolated intermediate points. After these two steps, we built the PRM by sampling a configuration and adding it as a vertex to the graph, then we would check if it could steer to its three nearest neighbors and add the edges and weights to the graph. The weights of the edge are the Euclidean distances between the two configurations. We did not check if a sample was already in the graph as the sampling function would rarely sample the same point twice. Now that we have the PRM we can use Dijkstra's Algorithm, or any shortest path algorithm to find the shortest path from the start to the goal. It is important to note that we add the start and goal configuration to the PRM after the random sampling, however, you should be able to add the start and goal to the graph before as well. Now that we have the path, a sequence of configurations that we will call waypoints, we can extract the cost of each waypoint to the goal by calculating the cost of the entire path that remains from a given waypoint. After this, we save the results to a file, remove the start and goal vertexes from the graph, and re-run the path and waypoint computation for a new given start and goal. Note that we use the same PRM for all start and goal configurations, meaning we only build a PRM once per environment. The waypoints will then be used as data for a model that can predict the cost to goal for any point given the start and goal configuration. The algorithm outline is shown in Figure 1.

Algorithm 6 Roadmap Construction Algorithm

Input:
n : number of nodes to put in the roadmap
k : number of closest neighbors to examine for each configuration
Output:
A roadmap $G = (V, E)$

$V \leftarrow \emptyset$
$E \leftarrow \emptyset$
while $|V| < n$ **do**
  repeat
    $q \leftarrow$ a random configuration in $\mathcal{Q}$
  until q is collision-free
  $V \leftarrow V \cup \{q\}$
end while
for all $q \in V$ **do**
  $N_q \leftarrow$ the k closest neighbors of q chosen from V according to *dist*
  for all $q' \in N_q$ **do**
    if $(q, q') \notin E$ **and** $\Delta(q, q') \neq$ `NIL` **then**
      $E \leftarrow E \cup \{(q, q')\}$
    end if
  end for
end for

Fig. 1: PRM Algorithm

In this phase of the project, we focused on training a neural network model to predict the cost to reach the goal configuration from a given start configuration. The training process began with data extraction from the environment records, where each data point consisted of a start configuration, a goal configuration, an intermediate waypoint, and its corresponding cost. We organized these data points into features and labels, where the features represented concatenated vectors of start, goal, and waypoint configurations, and the labels represented the associated costs. The neural network architecture chosen for this task was a feed-forward network with three hidden layers, each followed by a dropout layer to prevent overfitting. We utilized the mean squared error loss function and the Adam optimizer with weight decay to optimize the model parameters. The training loop spanned 2500 epochs, during which the model iteratively adjusted its parameters to minimize the training loss. Additionally, we monitored the test loss to evaluate the model's generalization performance on unseen data. Both the training and test losses were visualized using separate plots to track the convergence of the model during training. Throughout the training process, we continually assessed the model's performance by computing the test loss and the coefficient of determination (R^2). These metrics provided insights into how well the model predicted the costs to reach the goal configurations. By optimizing the model's parameters and monitoring its performance metrics, we aimed to develop a robust predictive model capable of accurately estimating the costs associated with reaching goal configurations from given start configurations. Finally, we saved the trained model parameters and optimizer state for future use, enabling seamless integration into the overall project workflow.

Finally, we developed a basic implementation of RRT*. Our implementation utilized the same sampling and steering functions used in PRM to generate collision-free configurations within the given workspace. Next, we were able to find the nearest node in the tree and check if they could be steered between. If they could, then we would get the Q-nearest Nodes (Qnodes) and set the parent of the sampled node to be the node within Qnodes that had the lowest cost between it and the sampled node. We then update the cost of the sampled point to be the sum of the cost of its parent and the Euclidean distance to its parent. Finally, we check to see if any of the Qnodes can be rewired in order to reduce their cost. This results in checking if the cost from the newly sampled point to the goal plus the cost to a given Qnode is less than its original cost. If so, then the addition of the sampled node makes it possible to reach the goal for cheaper from that Qnode, and as such it should be rewired meaning we set the sampled node as the new parent of that specific Qnode and update its cost and children cost. We recursively change the cost of the child nodes since they may also have child nodes attached to them. An outline for the algorithm is shown in Figure 2 where Qnodes is the same as znear nodes.

Finally, we implemented a version of RRT* that takes into consideration the cost to start as well as the cost to the goal, this is done by modeling the heuristic cost of every node once and adding it to the total cost of the node. So the cost of a given node would be represented by the cost to start as well as the cost to goal. This is done by initializing the cost of a node using the model.

Algorithm 2.
T = (V, E) ← RRT*(z_{ini})

1 $T \leftarrow$ InitializeTree();
2 $T \leftarrow$ InsertNode(Ø, z_{init}, T);
3 ***for*** i=0 to i=N ***do***
4 $z_{rand} \leftarrow$ Sample(i);
5 $z_{nearest} \leftarrow$ Nearest(T, z_{rand});
6 (z_{new}, U_{new}) $\leftarrow$ Steer ($z_{nearest}$, z_{rand});
7 ***if*** Obstaclefree(z_{new}) ***then***
8 $z_{near} \leftarrow$ Near(T, z_{new}, $|V|$);
9 $z_{min} \leftarrow$ Chooseparent (z_{near}, $z_{nearest}$, z_{new});
10 $T \leftarrow$ InsertNode(z_{min}, z_{new}, T);
11 $T \leftarrow$ Rewire (T, z_{near}, z_{min}, z_{new});
12 ***return*** T

Fig. 2: RRT* Algorithm

III. Results

In our study, we conducted comprehensive experiments to evaluate the effectiveness of integrating machine learning with motion planning algorithms. Firstly, we employed the PRM motion planner to construct a collision-free roadmap in various environments using the UR5 robot. The PRM graph was generated with 1000 collision-free samples, from which we derived 10000 paths. While not all attempted paths successfully connected due to obstacles, the resulting dataset provided valuable training examples for our predictive model. Subsequently, we trained a neural network model on this dataset to predict the cost associated with reaching the goal configuration from a given start configuration. The model exhibited commendable performance, achieving a final test loss of 2.7153 and an R^2 value of 0.78. These metrics signify the model's ability to accurately estimate the costs involved in reaching goal configurations, thereby facilitating more informed decision-making in motion planning.

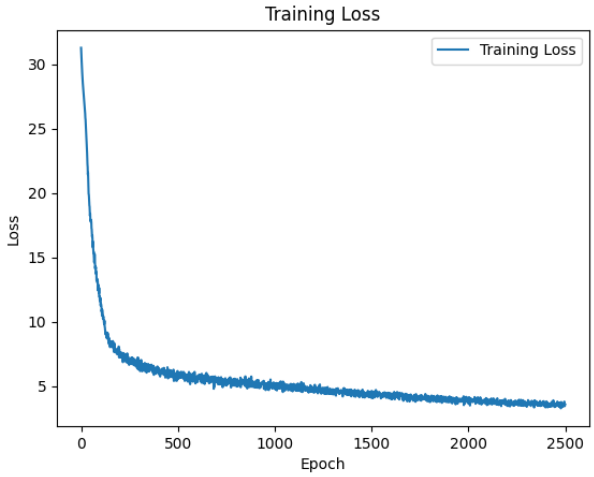

Fig. 3: Training Loss

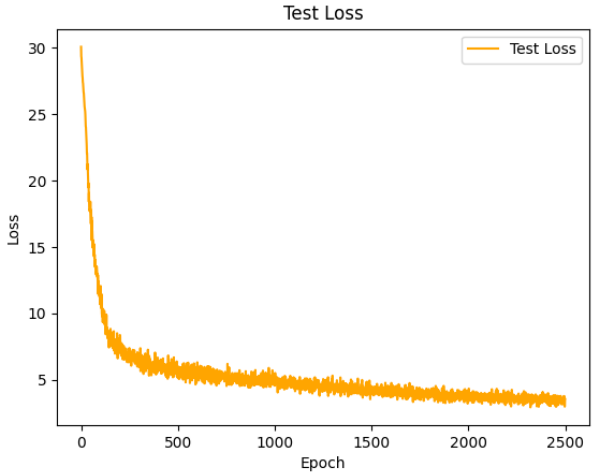

Fig. 4: Test Loss

Furthermore, we implemented an improved version of the RRT* algorithm that leveraged the predictive capabilities of our trained model. Our findings revealed that while the addition of heuristic modeling in the RRT* algorithm led to a modest reduction in path cost, the benefits were not always substantial, especially in scenarios where the start and goal configurations were relatively close together. Moreover, the effectiveness of heuristic modeling varied depending on the complexity of the environment and the characteristics of the robot. For instance, in environments with limited obstacles and simple configurations, the marginal improvement in path cost achieved by our integrated RRT* algorithm may not justify the computational overhead associated with PRM setup and model training. But, in more complex environments with larger configuration spaces or robots with greater degrees of freedom, the benefits of heuristic modeling are likely to be more pronounced.

TABLE I: RRT* Performance Metrics

Environment	Simple	Complex
Best Cost	3.92	5.22
Worst Cost	6.15	6.59
Average Cost	5.04	5.89

TABLE II: RRT* + Learning Performance Metrics

Environment	Simple	Complex
Best Cost	4.17	5.64
Worst Cost	5.47	6.33
Average Cost	4.82	5.89

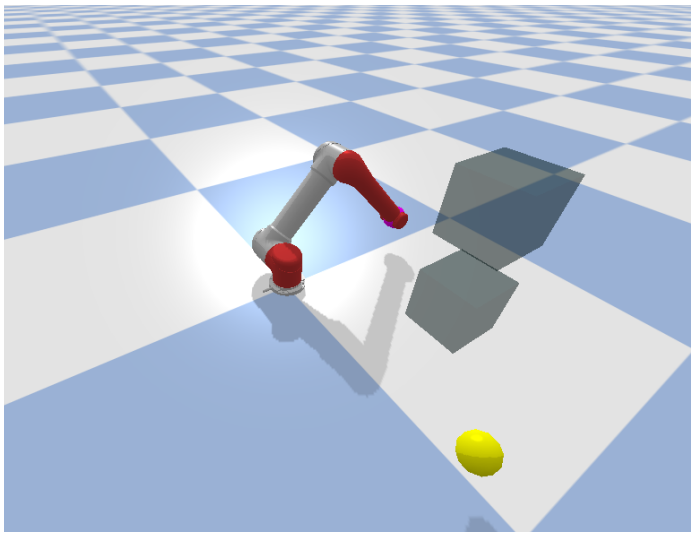

Fig. 5: Simple Environment

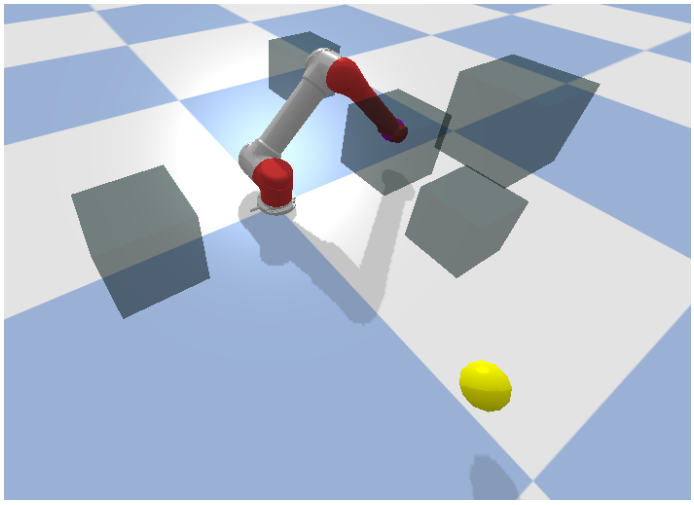

Fig. 6: Complex Environment

IV. Conclusions

While our study demonstrated promising results in integrating machine learning with motion planning algorithms, particularly in the context of the UR5 robot and relatively simple environments, there are several noteworthy points to consider. Firstly, the observed reduction in path cost achieved by incorporating heuristic modeling into the RRT* algorithm

was marginal, especially considering the overhead involved in setting up the PRM solution and training the predictive model. This suggests that for scenarios where the start and goal configurations are relatively close together, the benefits of incorporating such heuristics may not outweigh the associated costs. Moreover, the effectiveness of heuristic modeling may vary significantly depending on the complexity of the environment and the specific characteristics of the robot being used. In our experiments, the UR5 robot operated within environments with limited obstacles, potentially limiting the opportunities for significant heuristic-driven optimizations. Therefore, future research should explore more diverse and challenging environments, including those with larger configurations spaces or robots with more degrees of freedom. Such scenarios are more likely to exhibit areas where heuristic guidance can significantly improve path planning efficiency. Additionally, exploring more sophisticated machine learning architectures for estimating heuristic costs could lead to better performance, enhancing the overall effectiveness of the approach. In summary, while our study provides valuable insights into the integration of machine learning with motion planning algorithms, further research is needed to fully understand the potential benefits and limitations of this approach across a broader range of scenarios and robot configurations.

V. Future Works

Moving forward, there are several areas where we can focus our efforts to improve upon our current work. Firstly, we need to conduct more tests in different types of environments to better understand how well our approach works in various situations. This means testing it in environments with more obstacles, larger spaces, and different types of robots. Additionally, we can work on improving the way our predictive model works. This could involve trying out different methods to see which ones give us the most accurate predictions, as well as finding ways to make the model more efficient. We also want to explore ways to adjust how we use the predictive model based on the specific needs of the robot and its environment. By being smarter about when and how we use the model, we can make our planning process more efficient. Furthermore, we can look into combining our approach with other methods to see if we can get even better results. This might involve mixing our method with other types of planning techniques to take advantage of their strengths. Lastly, it's important to make sure that our approach can handle larger and more complex tasks. This means finding ways to make it faster and more scalable so that it can be used in real-time situations. Overall, by focusing on these areas, we can continue to improve our approach and make it more effective for a wide range of applications.