A Deep Learning Approach to Motion Planning

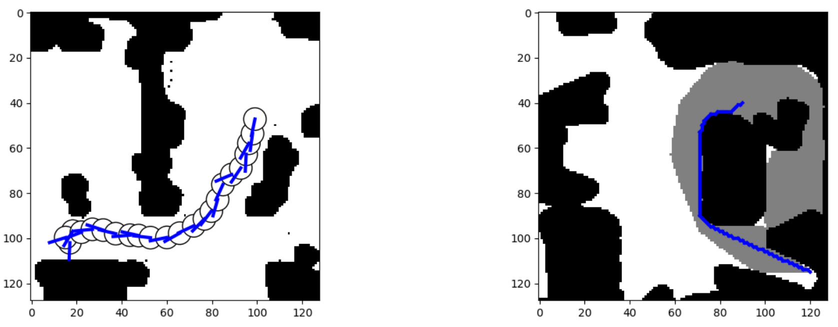
Mino Nakura, Nishant Elkunchwar, Pratik Gyawali

June 10, 2020

1 Introduction

Existing path planning methods, such as A* and Rapidly-exploring Random Trees (RRT), are known to provide collision-free paths from a start state to a goal state. However such methods become computationally expensive in high dimensional planning spaces. Recent work in this area [2] has shown neural networks to be a viable and computationally inexpensive alternative to such traditional motion planning methods. In this report, we explore neural network-based algorithms that imitate these path planning algorithms. Specifically, we present two neural network architectures and consider three neural network-based planning algorithms. We demonstrate the applications of these planning algorithms on a holonomic point robot in a grid world and on a car-like robot with non-holonomic constraints in a continuous 2D plane. We then evaluate these algorithms on a variety of maps and present the results. [1] [2]

2 Dataset



(a) Nonholonomic robot path using RRT (b) holonomic robot path using A*

Figure 1: Sample Visuals of Dataset

2.1 Holonomic robot Data

We generate 200 random 500x500 pixel croppings of the original maps and resize them to be 128x128 pixel images. For each map, we generate two types of paths by: (1) manually picking the start and goal state such that the generated path has to go around obstacles and (2) randomly generating start and goal configurations. These paths are connected by the A* algorithm. (1) ensures the dataset includes an ample amount of data that avoids obstacles, and (2) mitigates any bias that might have arise from manually picking start/goal configurations. We generate a total of 400 paths, which translate to around 30 thousand datapoints. Each data point consists of the current state of the robot (a point along the path), the goal state, the map we use to generate the path, and the action taken to get to the next state. Since we use a grid world, we encode the robot actions as a value 1 through 8, with each class representing a possible direction the robot can head in.

[1]GitHub repository
[2]Youtube Link

2.2 Nonholonomic robot Data

We use the RRT algorithm to generate paths for this dataset. Instead of picking an explicit goal configuration, we run the algorithm and terminate when either (1) 5 seconds has passed or (2) a path with at least a cost of 200 is generated. In the case of (1), we pick the path with the largest cost. In addition, we generate additional data by (3) picking an identical start/goal state only with different start/goal orientations. (3) has the added benefit of including data that could potentially teach the neural network how to generate a series of actions that could turn the robot in different directions. In total, we generate around 250 paths for each of the 200 random croppings of the original maps for a total of 48 thousand data points.

3 Dataloader

We implemented two versions of the dataloader - one for the holonomic robot and one for the non-holonomic car. The implemented dataloader was an iterator that returned the required training inputs and labels. The inputs consisted of an image of the map with the generated path from the supervisor (A* or RRT) and the start and goal positions. The labels were the encoded actions 1-8 for the holonomic robot and the linear velocity, angle pair for the car with non-holonomic constraints. This made it easier to extend `torch.utils.data.DataLoader` to use it for training our neural networks.

4 Neural Network Designs

In this section, we broadly describe the general architecture used in our algorithms. Each trained network has minor quirks (different dropout value, num. training iterations, layers etc), but the general structure of the neural network is mostly identical, so we will only describe the basic architecture here.

The input into the network consists of a start configuration, goal configuration, and an 128x128 image of the map. We pass the map image through 5 convolutional layers, flatten the outputs, and then pass it through 5 fully connected layers. In the third fully connected layer, we inject our start/goal configurations into the neural network. This allows the network to extract features from the image in the earlier layers, and then combine them with the start/goal configurations later on in the network.

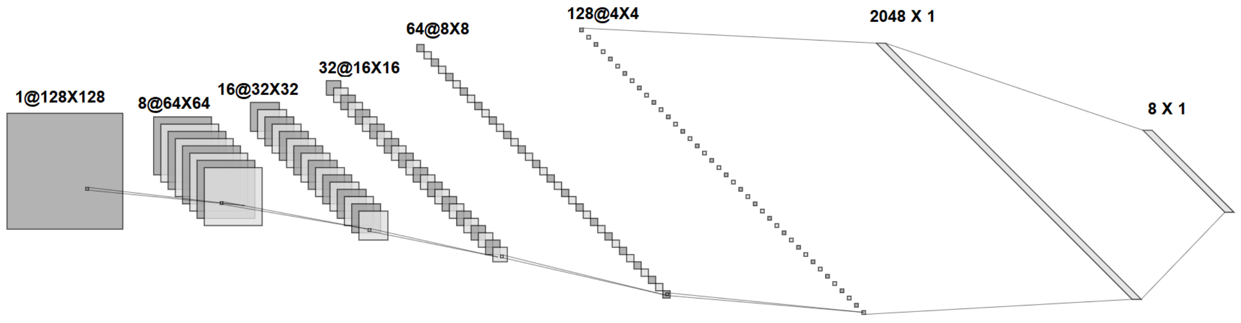


Figure 2: Neural Network Architecture for holonomic robot

In some cases, we also include two dropout layers in our neural network. The dropout layers have several benefits for our task. First, it helps avoid overfitting on the dataset and ensures that the network doesn't rely too much on certain nodes in the network. Second, Dropout can be used during the predictions phase because it adds stochasticity. If the network produces an action that is invalid, the same input can be passed through the network to obtain a different action. Thus, the randomness can help recover from failed predictions [2].

5 Algorithms

Algorithm 1 Unidirectional Planner

```
procedure Plan(start, goal, env)
    $curr = start$
    $plan = [start]$
    while curr != goal do
        $action = Net(curr, goal, env.map)$
        if action == None then
            $break$
        $curr = curr + delta$
        $plan.append(curr)$
    return $plan$
```

Algorithm 2 Bidirectional Planner

```
procedure BiDirectionalPlan(start, goal, env)
    $curr, dest = start, goal$
    $forward = True$
    $splan, gplan = [start], [goal]$
    while curr != dest do
        if forward then
            $action = net(curr, dest, env.map)$
        else
            $action = net(dest, curr, env.map)$
        if forward then
            $curr = curr + action$
            $splan.append(curr)$
        else
            $dest = dest + action$
            $gplan.append(curr)$
        $forward = \neg forward$
        $plan.append(curr)$
    return $plan$
```

The Unidirectional planner is a greedy planner that will greedily take the next action predicted by the neural network. If the prediction is invalid, it will take the next best prediction and use that as its path. Concretely, it starts from the start state and gradually tries to move towards the goal state. On the other hand, the bidirectional planner plans from both the start state and goal state and updates the start/goal destination it is trying to reach with every iteration of the algorithm. Once the two paths converge at a state, we concatenate the two paths together to form one path. In addition, we also use a lazy state contraction algorithm to remove any unnecessary states that could be cut out from the planned path.

Algorithm 3 Dropout Planner for Nonholonomic Car

```
procedure PLAN(START, GOAL, ENV)
    $dropout = 0.1$
    while dropout $\leq 0.9$ do
        $curr = start$
        $plan = [start]$
        while curr != goal and <150 iterations do
            $action = Net(curr, goal, env.map, dropout)$
            if action != None then
                $curr = curr + delta$
                $plan.append(curr)$
        if curr == goal then
            break
        $dropout+ = 0.2$
    return $plan$
```

The dropout planner is an extension of the unidirectional planner. However, instead of picking the next best option when it predicts an unfeasible action, we keep on iterating until it predicts a viable action. This is possible because of the stochastic nature of using dropout layers. If we iterate for more than 150 actions, we drop the plan and restart with a higher dropout rate. A higher dropout rate allows for more randomness in our predictions, meaning we are more likely to test out a wider variety of actions if the first action is unfeasible.

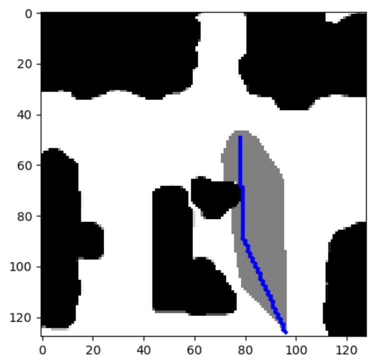
6 Results

6.1 Holonomic point robot (trained with A* as supervisor)

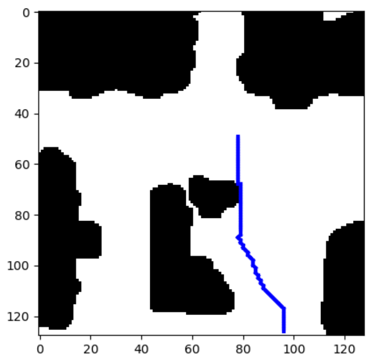
We generated a testing dataset of 25 maps with random start, goal positions and tested ADoubleStarNet on them with unidirectional and bi-directional planners. The table below summarizes the success rate of our neural network, i.e., the number of times the Neural Network was able to successfully find paths from start to goal.

DoubleShootingStarNet success rate

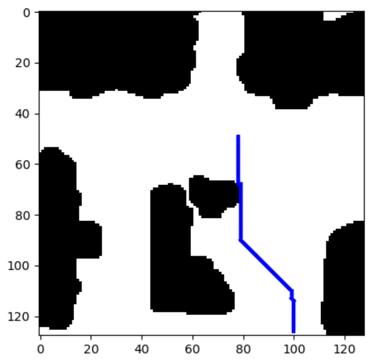
Planner	Successes	Testing dataset size	% Success
Unidirectional	7	25	28%
Unidirectional	1427	3068	46.51%
Bi-directional	13	25	52%
Bi-directional	1728	3068	56.32%



(a) Path generated by A* (cost = 84.455844)



(b) Path generated by DoubleShootingStarNet with bidirectional planner (cost = 84.870058)



(c) Path generated by DoubleShootingStarNet with unidirectional planner (cost = 86.698485)

Figure 3: Comparison of paths from A* and from our Neural Network with unidirectional and bidirectional planners on a test map

6.2 Car with non-holonomic constraints (trained with RRT as a supervisor)

We generated a testing dataset of 50 maps with random start, goal positions and tested RRTwoNet on the Dropout planner. The table below summarizes the success rate of our neural network, i.e., the number of times the Neural Network was able to successfully find paths from start to goal.

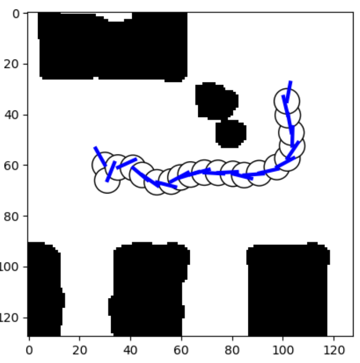
Dropout Planner Success Rate

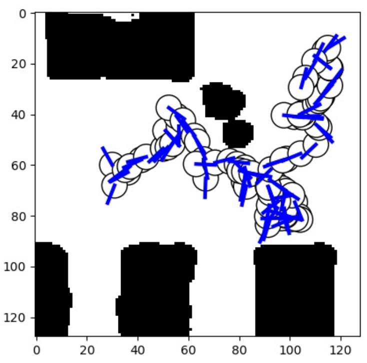
Planner	Successes	Testing dataset size	% Success	Avg. Dropout Probability
Dropout Planner	29	48	60.41%	0.52
Dropout Planner	708	937	75.56%	0.41

In addition to the percentage accuracy, the position and the orientation error for each successful plan was studied. The table below summarizes these performance metrics.

Performance Measure of Sucessful Plans

Testing Data	Mean Position Error	Var. in Position Error	Mean Orientation Error	Var. in Ori. Error
48	6.1992	2.5498	2.3706	1.7587
937	5.8354	2.5622	2.5839	1.4780





(a) Path generated by Non-holonomic RRT (Cost: 220)

(b) Path generated by Dropout Planner (Cost: 504, drop out probability: 0.7)

Figure 4: Comparison of paths from RRT non-holonomic supervisor and from our Neural Network on a test map

7 Conclusion

In this project we developed a neural network based approach to motion planning for a holonomic robot and for a car with non-holonomic constraints. Both problems presented unique challenges which were solved in different ways. For the holonomic robot, the neural network worked as a classifier that predicted the next state for the robot, given the current state and goal position. This was tested with unidirectional and bidirectional planning algorithms on several testing datapoints. For the non-holonomic car, the neural network predicted the actions, given the current state and goal position. However due to the constraints, some stochasticity had to be introduced into the network which was done by adding dropout layers.

Convolutional layers at the beginning of the network helped extract features from the images of maps with generated paths and our trained neural networks were able to successfully find paths in approximately 50% cases for the point robot and around 70% for the non-holonomic car. Moreover, instead of predicting an entire path before execution like traditional motion planning approaches, we adopted an approach presented in a recent work [1] - to predict and execute the action for the current time step and then again predict the next action, till the goal is reached. This can turn out to be helpful in situations where the map is dynamic or there are moving obstacles like people.

References

[1] Ashwini Pokle, Roberto Martín-Martín, Patrick Goebel, Vincent Chow, Hans M Ewald, Junwei Yang, Zhenkai Wang, Amir Sadeghian, Dorsa Sadigh, Silvio Savarese, et al. Deep local trajectory replanning and control for robot navigation. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 5815–5822. IEEE, 2019.

[2] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.