2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Motion Planning Networks

Ahmed H. Qureshi, Anthony Simeonov, Mayur J. Bency and Michael C. Yip

Abstract— **Fast and efficient motion planning algorithms are crucial for many state-of-the-art robotics applications such as self-driving cars. Existing motion planning methods become ineffective as their computational complexity increases exponentially with the dimensionality of the motion planning problem. To address this issue, we present Motion Planning Networks (MPNet), a neural network-based novel planning algorithm. The proposed method encodes the given workspaces directly from a point cloud measurement and generates the end-to-end collision-free paths for the given start and goal configurations. We evaluate MPNet on various 2D and 3D environments including the planning of a 7 DOF Baxter robot manipulator. The results show that MPNet is not only consistently computationally efficient in all environments but also generalizes to completely unseen environments. The results also show that the computation time of MPNet consistently remains less than 1 second in all presented experiments, which is significantly lower than existing state-of-the-art motion planning algorithms.**

I. Introduction

Robotic motion planning aims to compute a collision-free path for the given start and goal configurations [1]. As motion planning algorithms are necessary for solving a variety of complicated, high-dimensional problems ranging from autonomous driving [2] to space exploration [3], there arises a critical, unmet need for computationally tractable, real-time algorithms. The quest for developing computationally efficient motion planning methods has led to the development of various sampling-based motion planning (SMP) algorithms such as Rapidly-exploring Random Trees (RRT) [4], optimal Rapidly-exploring Random Trees (RRT*) [5], Potentially guided-RRT* (P-RRT*) [6] and their bi-directional variants [7], [8]. Despite previous efforts to design fast, efficient planning algorithms, the current state-of-the-art struggles to offer methods which scale to the high-dimensional setting that is common in many real-world applications.

To address the above-mentioned challenges, we propose a Deep Neural Network (DNN) based iterative motion planning algorithm, called MPNet (Motion Planning Networks) that efficiently scales to high-dimensional problems. MPNet consists of two components: an encoder network and a planning network. The encoder network learns to encode a point cloud of the obstacles into a latent space. The planning network learns to predict the robot configuration at time step $t + 1$ given the robot configuration at time t, goal configuration, and the latent-space encoding of the obstacle space. Once trained, MPNet can be used in conjunction with our novel bi-directional iterative algorithm to generate feasible trajectories. We evaluate MPNet on a large test dataset including multiple planning problems such as the planning of a point-mass robot, rigid-body, and 7 DOF Baxter robot manipulator in various 2D and 3D environments. As neural networks do not provide theoretical guarantees on their performance, we also propose a hybrid algorithm which combines MPNet with any existing classical planning algorithm, in our case RRT*. The hybrid planning technique demonstrates a 100% success rate consistently over all tested environments while retaining the computational gains. Our results indicate that MPNet generalizes very well, not only to unseen start and goal configurations within workspaces which were used in training, but also to new workspaces which the algorithm has never seen.

A. H. Qureshi[1], A. Simeonov[2], M. J. Bency[1], and M. C. Yip[1,2] are with (1) Department of Electrical and Computer Engineering; (2) Department of Mechanical and Aerospace Engineering; University of California San Diego, La Jolla, CA 92093 USA. {a1qureshi, asimeono, mbency, yip}@ucsd.edu

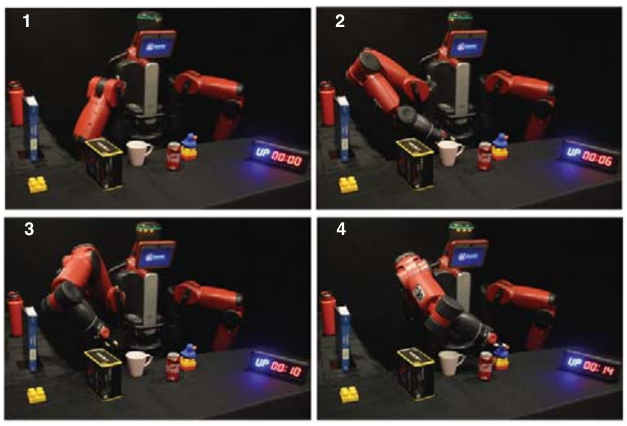


Fig. 1: MPNet planned motion for a 7 DOF Baxter robot manipulator. The path profile followed by the robot from initial to target configuration is shown through frames 1-4. The stopwatch in the images show the execution time. In this particular case, MPNet took less than 1 second whereas BIT* [9] took 3.1 minutes on average to find a feasible path solution of comparable euclidean cost.

II. Related Work

Research into developing neural network-based motion planners first gained traction in the early 1990s but faded away due to computational complexity of training deep neural networks [10]. However, recent developments in Deep Learning (DL) have allowed researchers to apply various DL architectures to robotic control and planning.

An active area of research within robotic control and planning is Deep Reinforcement Learning (DRL). For instance, [11] shows how to train a robot to learn visuomotor policies to perform various tasks such as screwing a bottle cap, or inserting a peg. Although DRL is a promising framework, it extensively relies on exploration through interaction with the environment, thus making it difficult to train for many real-world robotic applications. A recent work, Value Iteration Networks (VIN) [12] emulates value iteration by leveraging recurrent convolutional neural networks and max-pooling. However, in addition to limitations inherited from underlying DRL framework, VIN has only been evaluated on simple toy problems, and it is not clear how VIN could extend beyond such environments.

Imitation learning is another emerging field in which the models are trained from expert demonstrations. Many interesting problems have been addressed through imitation learning [13], [14], [15]. A recent method [16] uses deep neural networks trained via imitation to adaptively sample the configuration space for SMP methods. Our proposed method also learns through imitation but unlike [16], it provides a complete feasible motion plan for a robot to follow.

Another recent and relevant method is the Lightning Framework [17], which is composed of two modules. The first module performs path planning using any traditional motion planner. The second module maintains a lookup table which caches old paths generated by the first module. For new planning problems, the Lightning Framework retrieves the closest path from a lookup table and repairs it using a traditional motion planner. This approach demonstrates superior performance compared to conventional planning methods. However, not only are lookup tables memory inefficient, they also are incapable of generalizing to new environments.

III. Problem Definition

This section describes the notations used in this paper and formally defines the motion planning problem addressed by the proposed method.

Let Q be an ordered list of length $N \in \mathbb{N}$, then a sequence $\{q_i = Q(i)\}_{i \in N}$ is a mapping from $i \in \mathbb{N}$ to the i-th element of Q. Moreover, for the algorithms described in this paper, $Q(\text{end})$ and $Q.\text{length}()$ give the last element and the number of elements in a set Q, respectively. Let $X \subset \mathbb{R}^d$ be a given state space, where $d \in \mathbb{N}$ is the dimensionality of the state space. The workspace dimensionality is indicated by $d_w \in \mathbb{N}$. The obstacle and obstacle-free state spaces are defined as $X_{\text{obs}} \subset X$ and $X_{\text{free}} = X \backslash X_{\text{obs}}$, respectively. Let the initial state be $\boldsymbol{x}_{\textbf{init}} \in X_{\text{free}}$, and goal region be $X_{\text{goal}} \subset X_{\text{free}}$. Let an ordered list τ be a path having positive scalar length. A solution path τ to the motion planning problem is feasible if it connects $\boldsymbol{x}_{\textbf{init}}$ and $\boldsymbol{x} \in X_{\text{goal}}$, i.e. $\tau(0) = \boldsymbol{x}_{\textbf{init}}$ and $\tau(\text{end}) \in X_{\text{goal}}$, and lies entirely in the obstacle-free space X_{free}. The proposed work addresses the feasibility problem of motion planning.

IV. MPNet: A Neural Motion Planner

This section introduces our proposed model, MPNet[1] (see Fig. 2). MPNet is a neural network based motion planner comprised of two phases: (A) offline training of the neural models, and (B) online path generation.

[1] Supplementary material including implementation parameters and project videos are available at https://sites.google.com/view/mpnet/home.

A. Offline Training

Our proposed method uses two neural models to solve the motion planning problem. The first model is an encoder network which embeds the obstacles point cloud, corresponding to a point cloud representing X_{obs}, into a latent space (see Fig. 2(a)). The second model is a planning network (Pnet) which learns to do motion planning for the given obstacle embedding, and start and goal configurations of the robot (see Fig. 2(b)).

1) Encoder Network: The encoder network (Enet) embeds the obstacles point cloud into a feature space $\boldsymbol{Z} \in \mathbb{R}^m$ with dimensionality $m \in \mathbb{N}$. Enet can be trained either using encoder-decoder architecture with a reconstruction loss or in an end-to-end fashion with the Pnet (described below). For encoder-decoder training, we found that the contrative autoencoders (CAE) [18] learns robust and invariant feature space required for planning and genalization to unseen workspaces. The reconstruction loss of CAE is defined as:

$$L_{\text{AE}}(\boldsymbol{\theta}^e, \boldsymbol{\theta}^d) = \frac{1}{N_{\text{obs}}} \sum_{\boldsymbol{x} \in D_{\text{obs}}} ||\boldsymbol{x} - \hat{\boldsymbol{x}}||^2 + \lambda \sum_{ij} (\theta^e_{ij})^2 \quad (1)$$

where $\boldsymbol{\theta}^e$ are the parameters of encoder, $\boldsymbol{\theta}^d$ are the parameters of decoder, λ is a penalizing coefficient, D_{obs} is a dataset of point clouds $\boldsymbol{x} \in X_{\text{obs}}$ from $N_{\text{obs}} \in \mathbb{N}$ different workspaces, and $\hat{\boldsymbol{x}}$ is the point cloud reconstructed by the decoder.

2) Planning Network: We use a feed-forward deep neural network, parameterized by $\boldsymbol{\theta}$, to perform planning. Given the obstacles encoding $\boldsymbol{Z}$, current state $\boldsymbol{x}_t$ and the goal state $\boldsymbol{x}_T$, Pnet predicts the next state $\hat{\boldsymbol{x}}_{t+1} \in X_{\text{free}}$ which would lead a robot closer to the goal region, i.e., $\hat{\boldsymbol{x}}_{t+1} = \text{Pnet}((\boldsymbol{x}_t, \boldsymbol{x}_T, \boldsymbol{Z}); \boldsymbol{\theta})$

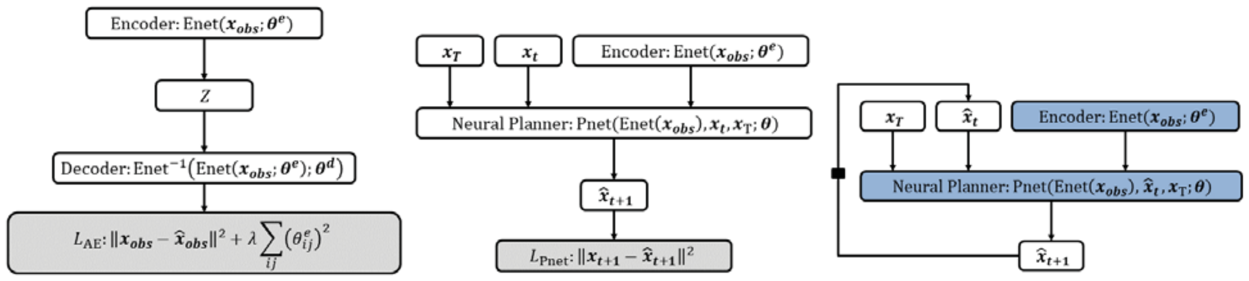
To train Pnet, any planner or human expert can provide feasible, near-optimal paths as expert demonstrations. We assume the paths given by the expert demonstrator are in a form of a tuple, $\tau^* = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$, of feasible states that connect the start and goal configurations so that the connected path lies entirely in X_{free}. The training objective for the Pnet is to minimize the mean-squared-error (MSE) loss between the predicted states $\hat{\boldsymbol{x}}_{t+1}$ and the actual states $\boldsymbol{x}_{t+1}$ given by the expert data, formalized as:

$$L_{\text{Pnet}}(\boldsymbol{\theta}) = \frac{1}{N_p} \sum_{j}^{\hat{N}} \sum_{i=0}^{T-1} ||\hat{\boldsymbol{x}}_{j,i+1} - \boldsymbol{x}_{j,i+1}||^2, \quad (2)$$

where $N_p \in \mathbb{N}$ is the averaging term corresponding to the total number of paths, $\hat{N} \in \mathbb{N}$, in the training dataset times the path lengths.

B. Online Path Planning

The online phase exploits the neural models from the offline phase to do motion planning in cluttered and complex environments. The overall flow of information between Enet and Pnet is shown in Fig. 2(c). To generate end-to-end feasible paths connecting the start and goal states, we propose a novel incremental bidirectional path generation heuristic. Algorithm 1 presents the overall path generation procedure and its constituent functions are described below.



Encoder: Enet(x_{obs}; θ^e)
Z
Decoder: Enet^{-1}(Enet(x_{obs}; θ^e); θ^d)
L_{AE}: $\|x_{obs} - \hat{x}_{obs}\|^2 + \lambda \sum_{ij} (\theta^e_{ij})^2$
x_T
x_t
Encoder: Enet(x_{obs}; θ^e)
Neural Planner: Pnet(Enet(x_{obs}), x_t, x_T; θ)
$\hat{x}_{t+1}$
L_{Pnet}: $\|x_{t+1} - \hat{x}_{t+1}\|^2$
x_T
$\hat{x}_t$
Encoder: Enet(x_{obs}; θ^e)
Neural Planner: Pnet(Enet(x_{obs}), $\hat{x}_t$, x_T; θ)
$\hat{x}_{t+1}$

(a) Offline: Encoder Network (b) Offline: Planning Network (c) Online: Neural Planner

Fig. 2: The offline and online phases of MPNet. The grey shaded blocks indicate the training objectives. The blue blocks represents frozen modules that do not undergo any training.

1) Enet: The encoder network $\mathrm{Enet}(\boldsymbol{x}_{\mathbf{obs}})$, trained during the offline phase, is used to encode the obstacles point cloud $\boldsymbol{x}_{\mathbf{obs}} \in X_{\mathrm{obs}}$ into a latent space $\boldsymbol{Z} \in \mathbb{R}^m$.

2) Pnet: Pnet is a feed-forward neural network from the offline phase which takes $\boldsymbol{Z}$, current state $\boldsymbol{x}_t$, goal state $\boldsymbol{x}_T$ and predicts the next state of the robot $\hat{\boldsymbol{x}}_{t+1}$. To inculcate stochasticity into the Pnet, some of the hidden units in each of its hidden layer were dropped out with a probability $p : [0,1] \in \mathbb{R}$. The merit of adding the stochasticity during the online path generation are presented in the discussion section.

3) Lazy States Contraction (LSC): Given a path $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$, the LSC algorithm connects the directly connectable non-consecutive states, i.e., $\boldsymbol{x}_i$ and $\boldsymbol{x}_{>i+1}$, and removes the intermediate/lazy states.

4) Steering: The steerTo function takes two states as an input and checks either a straight trajectory connecting the given two states lies entirely in collision-free space X_{free} or not. The steering is done from $\boldsymbol{x}_1$ to $\boldsymbol{x}_2$ in small, discrete steps and can be summarized as $\tau(\delta) = (1-\delta)\boldsymbol{x}_1 + \delta\boldsymbol{x}_2; \forall\delta \in [0,1]$. The discrete step size could be a fixed number or can be adapted for different parts of the algorithm.

5) isFeasible: Given a path $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$, this procedure checks either the end-to-end path, formed by connecting the consecutive states in τ, lies entirely in X_{free} or not.

6) Neural Planner: This is an incremental bidirectional path generation heuristic (see Algorithm 2 for the outline). It takes the obstacles' representation, Z, as well as the start and goal states as an input, and outputs a path connecting the two given states. The sets τ^a and τ^b correspond to the paths generated from the start and goal states, respectively. The algorithm starts with τ^a, it generates a new state $\boldsymbol{x}_{\mathbf{new}}$, using Pnet, from start towards the goal (Line 5), and checks if a path from start τ^a is connectable to the path from a goal τ^b (Line 7). If paths are connectable, an end-to-end path τ is returned by concatenating τ^a and τ^b. However, if paths are not connectable, the roles of τ^a and τ^b are swapped (Line 11) and the whole procedure is repeated again. The swap function enables the bidirectional generation of paths, i.e., if at any iteration i, path τ^a is extended then in the next iteration $i+1$, path τ^b will be extended. This way, two trajectories τ^a and τ^b march towards each other which makes this path generation heuristic greedy and fast.

7) Replanning: This procedure is outlined in the Algorithm 3. It iterates over all the consecutive states $\boldsymbol{x}_i$ and $\boldsymbol{x}_{i+1}$ in a given path $\tau = \{\boldsymbol{x}_0, \boldsymbol{x}_1, \cdots, \boldsymbol{x}_T\}$, and checks if they are connectable or not, where $i = [0, T-1] \subset \mathbb{N}$. If any consecutive states are found not connectable, a new path is generated between those states using one of the following replanning methods (Line 5).

a) Neural Replanning: Given a start and goal states

Algorithm 1: MPNet($\boldsymbol{x}_{\mathbf{init}}, \boldsymbol{x}_{\mathbf{goal}}, \boldsymbol{x}_{\mathbf{obs}}$)

```
1  $\boldsymbol{Z} \leftarrow \mathrm{Enet}(\boldsymbol{x}_{\mathbf{obs}})$
2  $\tau \leftarrow \mathrm{NeuralPlanner}(\boldsymbol{x}_{\mathbf{init}}, \boldsymbol{x}_{\mathbf{goal}}, \boldsymbol{Z})$;
3  if $\tau$ then
4      $\tau \leftarrow \mathrm{LazyStatesContraction}(\tau)$
5      if IsFeasible($\tau$) then
6          return $\tau$
7      else
8          $\tau_{\mathrm{new}} \leftarrow \mathrm{Replanning}(\tau, \boldsymbol{Z})$
9          $\tau_{\mathrm{new}} \leftarrow \mathrm{LazyStatesContraction}(\tau_{\mathrm{new}})$
10         if IsFeasible($\tau_{\mathrm{new}}$) then
11             return $\tau_{\mathrm{new}}$
12         return $\varnothing$
```

Algorithm 2: NeuralPlanner($\boldsymbol{x}_{\mathbf{start}}, \boldsymbol{x}_{\mathbf{goal}}, \boldsymbol{Z}$)

```
1  $\tau^{\mathrm{a}} \leftarrow \{\boldsymbol{x}_{\mathbf{start}}\}; \tau^{\mathrm{b}} \leftarrow \{\boldsymbol{x}_{\mathbf{goal}}\}$;
2  $\tau \leftarrow \varnothing$;
3  Reached $\leftarrow$ False;
4  for $i \leftarrow 0$ to $N$ do
5      $\boldsymbol{x}_{\mathbf{new}} \leftarrow \mathrm{Pnet}(\boldsymbol{Z}, \tau^{\mathrm{a}}(\mathrm{end}), \tau^{\mathrm{b}}(\mathrm{end}))$
6      $\tau^{\mathrm{a}} \leftarrow \tau^{\mathrm{a}} \cup \{\boldsymbol{x}_{\mathbf{new}}\}$
7      Connect $\leftarrow \mathrm{steerTo}(\tau^{\mathrm{a}}(\mathrm{end}), \tau^{\mathrm{b}}(\mathrm{end}))$
8      if Connect then
9          $\tau \leftarrow \mathrm{concatenate}(\tau^{\mathrm{a}}, \tau^{\mathrm{b}})$
10         return $\tau$
11     SWAP($\tau^{\mathrm{a}}, \tau^{\mathrm{b}}$)
12 return $\varnothing$
```

Algorithm 3: Replanning($\tau, \boldsymbol{Z}$)

```
1  $\tau_{\text{new}} \leftarrow \varnothing$;
2  for $i \leftarrow 0$ to $\tau$.length() do
3      if steerTo($\tau_i, \tau_{i+1}$) then
4          $\tau_{\text{new}} \leftarrow \tau_{\text{new}} \cup \{\tau_i, \tau_{i+1}\}$
5      else
6          $\tau_{\text{mini}} \leftarrow$ Replanner($\tau_i, \tau_{i+1}, \boldsymbol{Z}$);
7          if $\tau_{\text{mini}}$ then
8              $\tau_{\text{new}} \leftarrow \tau_{\text{new}} \cup \tau_{\text{mini}}$
9          else
10             return $\varnothing$
11 return $\tau_{\text{new}}$
```

together with obstacle space encoding $\boldsymbol{Z}$, this method recursively finds a new path between the two given states. To do so, it starts by finding a coarse path between the given states and then if required, it replans on a finer level by calling itself over the non-connectable consecutive states of the new path. This recursive neural replanning is performed for the fixed number of steps to limit the algorithm within the computational bounds.

b) Hybrid Replanning: This heuristic combines the neural replanning with the classical motion planning methods. It performs the neural replanning for the fixed number of steps. The resulting new path is tested for feasibility. If a path is not feasible, the non-connectable states in the new path are then connected using a classical motion planner.

V. Implementation details

This section gives the implementation details of MPNet, for additional details refer to supplementary material. The proposed neural models were implemented in PyTorch [19]. For environments other than Baxter, the benchmark methods, Informed-RRT* and BIT*, were implemented in Python, and their times were compared against the CPU-time of MPNet. The Baxter environments were implemented with MoveIt! [20] and ROS. In these environments, we use a C++ OMPL [21] implementation of BIT* to compare against a C++ implementation of MPNet. The system used for training and testing has 3.40GHz$\times$ 8 Intel Core i7 processor with 32 GB RAM and GeForce GTX 1080 GPU. The remaining section explains different modules that lead to MPNet.

A. Data Collection

We generate 110 different workspaces for each presented case, i.e., simple 2D (s2D), rigid-body (rigid), complex 2D (c2D) and 3D (c3D). In each of the workspaces, 5000 collision-free, near-optimal, paths were generated using RRT*. The training dataset comprised of 100 workspaces with 4000 paths in each workspace. For testing, two types of test datasets were created to evaluate the proposed and benchmark methods. The first test dataset, seen-X_{obs}, comprised of already seen 100 workspaces with 200 unseen start and goal configurations in each workspace. The second test dataset, unseen-X_{obs}, comprised of completely unseen 10 workspaces where each contained 2000 unseen start and goal configurations. In the Baxter experiments, we created a dataset comprised of ten challenging simulated environments, and we show the execution on the real robot. For each environment, we collected 900 paths for training and 100 paths for testing. The obstacle point clouds were obtained using a Kinect depth camera with the PCL [22] and pcl_ros[2] package.

B. Models Architecture

1) Encoder Network: For all environments except Baxter, we use encoder-decoder training, whereas for Baxter, we train the encoder and planning network end-to-end. Since the decoder is usually the inverse of the encoder, we only describe the encoder's structure. The encoding function $\text{Enet}(\boldsymbol{x}_{\mathbf{obs}})$ comprised of three linear layers and an output layer, where each linear layer is followed by the Parametric Rectified Linear Unit (PReLU) [23]. The input to the encoder is a vector of point clouds of size $N_{\text{pc}} \times d_w$ where N_{pc} is the number of data points, and $d_w \in \mathbb{N}$ is the dimension of a workspace.

2) Planning Network: PNet is 9-layers and 12-layers DNN for Baxter and other environments, respectively. We use Parametric Rectified Linear Unit (PReLU) [23] for non-linearity. To add stochasticity, we Dropout (p) [24] in all hidden layers except the last one. In point-mass and rigid-body cases, we fix the pretrained encoder parameters, since they were trained using the encoder-decoder method, and use them to compute environment encoding, whereas, for Baxter environments, we train the Enet and Pnet end-to-end.

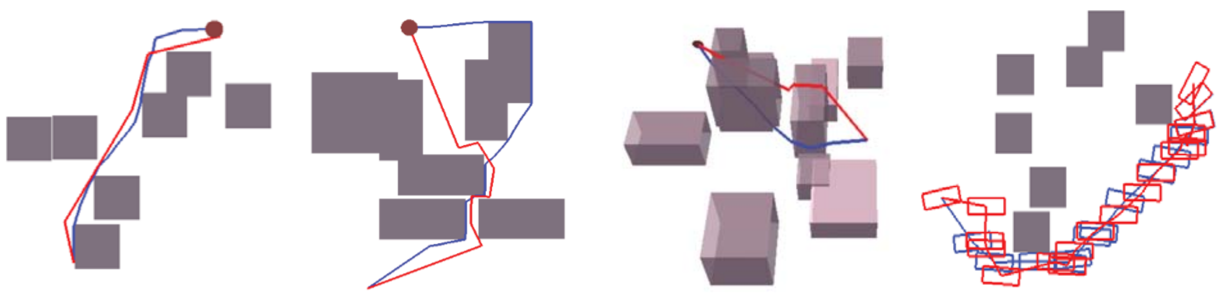
VI. Results

In this section, we compare the performance of MPNet with Neural-Replanning (MPNet: NR) and Hybrid-Replanning (MPNet: HR) against state-of-the-art motion planning methods, i.e., Informed-RRT* and BIT*, for the motion planning of the 2D/3D point-mass robots, rigid-body, and Baxter 7 DOF manipulator in the 2D and 3D environments.

Figs. 3 show different example scenarios where MPNet and expert planner, in this case RRT*, provided successful paths. The red and blue colored trajectories indicate the paths generated by MPNet and RRT*, respectively. The goal region is indicated as a brown colored disk. The mean computational time for the MPNet and RRT* is denoted as t_{MP} and t_{R}, respectively. We see that MPNet is able to compute near-optimal paths for both point-mass and rigid-body robot in considerably less time than RRT*.

Table I presents the CPU-time comparison of MPNet: NP and MPNet: HR against Informed-RRT* [25] and BIT* [9] over the two test datasets, i.e., seen-X_{obs} and unseen-X_{obs} . We report the mean times with standard deviation of all algorithms for finding the initial paths in a given problem. For initial paths, it is observed that on average the path lengths of benchmark methods were higher than

[2] http://wiki.ros.org/pcl_ros



(a) $t_{\text{R}} = 6.9s, t_{\text{MP}} = 0.50s$ (b) $t_{\text{R}} = 6.9s, t_{\text{MP}} = 0.50s$ (c) $t_{\text{R}} = 6.9s, t_{\text{MP}} = 0.50s$ (d) $t_{\text{R}} = 5.3s, t_{\text{MP}} = 0.44s$

Fig. 3: MPNet (Red) and RRT* (Blue) planning paths in complex 3D environments (c3D).

Environment	Test case	MPNet (NR)	MPNet (HR)	Informed-RRT*	BIT*	$\frac{\text{BIT}: t_{\text{mean}}}{\text{MPNet(NR)}: t_{\text{mean}}}$
Simple 2D	Seen X_{obs}	0.11 ± 0.037	0.19 ± 0.14	5.36 ± 0.34	2.71 ± 1.72	24.64
	Unseen X_{obs}	0.11 ± 0.038	0.34 ± 0.21	5.39 ± 0.18	2.63 ± 0.75	23.91
Complex 2D	Seen X_{obs}	0.17 ± 0.058	0.61 ± 0.35	6.18 ± 1.63	3.77 ± 1.62	22.17
	Unseen X_{obs}	0.18 ± 0.27	0.68 ± 0.41	6.31 ± 0.85	4.12 ± 1.99	22.89
Complex 3D	Seen X_{obs}	0.48 ± 0.10	0.34 ± 0.14	14.92 ± 5.39	8.57 ± 4.65	17.85
	Unseen X_{obs}	0.44 ± 0.107	0.55 ± 0.22	15.54 ± 2.25	8.86 ± 3.83	20.14
Rigid	Seen X_{obs}	0.32 ± 0.28	1.92 ± 1.30	30.25 ± 27.59	11.10 ± 5.59	34.69
	Unseen X_{obs}	0.33 ± 0.13	1.98 ± 1.85	30.38 ± 12.34	11.91 ± 5.34	36.09

TABLE I: Time comparison of MPNet (NR: Neural Replanning; HR: Hybrid Replanning), Informed-RRT* and BIT* on two test datasets. Note in the right most column that MPNet is at least $20\times$ faster than BIT*.

the path lengths of MPNet. It can be seen that in all test cases, the mean computation time of MPNet with neural and hybrid replanning remained around 1 second. The mean computation time of Informed-RRT* and BIT* increases significantly as the dimensionality of planning problem is increased. Note that, on average, MPNet is about 40 and 20 times faster than Informed-RRT* and BIT*, respectively, in all test cases and consistently demonstrates low computational time irrespective of the dimensionality of the planning problem. In these experiments, the mean accuracy of MPNet: HR and MPNet: NP was 100% and 97% with the standard deviation of about 0.4% over five different trials.

From experiments presented so far, it is evident that BIT* outperforms Informed-RRT*, therefore, in the following experiments only MPNet and BIT* are compared. Fig. 4 compares the mean computation time of MPNet: NP and BIT* in our two test datasets. It can be seen that the mean computation time of MPNet stays around 1 second irrespective of the planning problem dimensionality. Furthermore, the mean computational time of BIT* not only fluctuates but also increases significantly in the rigid-body planning problem. Finally, Fig. 1 shows a single 7 DOF arm of the Baxter robot executing a motion planned by MPNet for a given start and goal configuration. In Fig. 1, the robotic manipulator is at the start configuration, and the shadowed region indicates the manipulator at the target configuration. On the Baxter's test dataset, MPNet took about 1 second on average with 85% success rate. BIT* took about 9 seconds on average with a success rate of 56% to find paths within a 40% range of the path lengths found by MPNet, and was found to take up to several minutes to find paths within the 10% range of average MPNet path lengths.

VII. Discussion

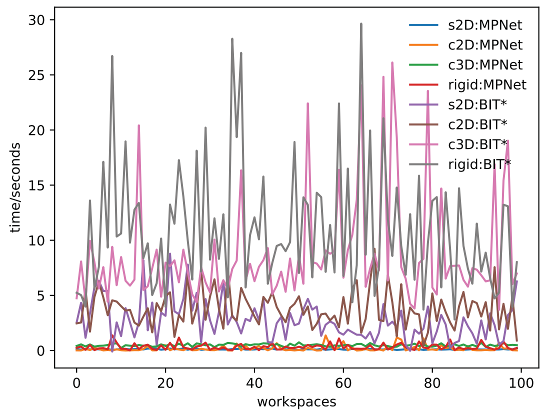
A. Stochasticity through Dropout

Our method uses Dropout [24] during both online and offline execution. Dropout is applied layer-wise to the neural network and it drops each unit in the hidden layer with a probability $p \in [0, 1]$, in our case $p = 0.5$. The resulting neural network is a thinned network and is essentially different from the actual neural model [24].

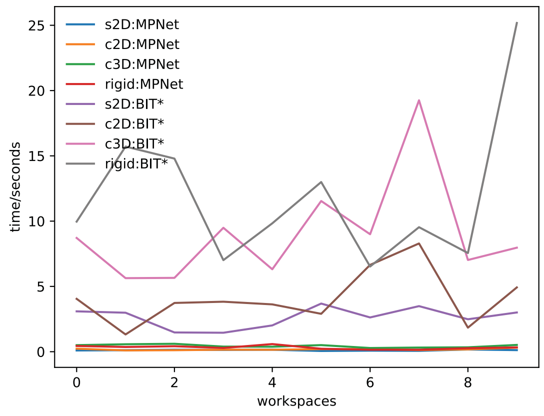
Note that, in the neural replanning phase, MPNet iterates over the non-connectable consecutive states of the coarse path to do motion planning on a finer level and thus, produces a new path. The replanning procedure is called recursively on each of its own newly generated paths until a feasible solution is found or a loop limit is reached. Dropout adds stochasticity to the Pnet which implies that on each replanning step, the Pnet would generate different paths from previous re-planning steps. This phenomenon is evident from Fig. 5 where the Pnet generated different paths for a fixed start and goal configurations. These perturbations in generated paths for fixed start and goal help in recovery from the failure. Thus, adding Dropout increases the overall performance of MPNet. Furthermore, due to stochasticity by Dropout, our method can also be used to generate adaptive samples for sampling based motion planners [26].

B. Completeness

In the proposed method, a coarse path is computed by a neural network. If a coarse path is found to be not fully connectable, a re-planning heuristic is executed to repair the non-connectable path segments to provide an end-to-end collision-free path. The completeness guarantees for the proposed method depends on the underline replanning



(a) Test-case 1: seen-X_{obs}



(b) Test-case 2: unseen-X_{obs}

Fig. 4: Computational time comparison of MPNet and RRT* on test datasets. The plots show MPNet is more consistent and faster than BIT* in all test cases.

heuristic. The classical motion planner based replanning methods are presented to guarantee the completeness of the proposed method. Since we use RRT*, our proposed method inherits the probabilistic completeness of RRTs and RRT* [5] while retaining the computational gains.

C. Computational Complexity

This section formally highlights the computational complexity of the proposed method. Neural networks are known to have online execution complexity of $O(1)$. Therefore, the execution of lines 1-2 of Algorithm 1 will have a complexity no greater than $O(1)$. The lazy state contraction (LSC) heuristic is a simple path smoothing technique which can be executed in a fixed number of iteration as a feasible trajectory must have a finite length. Also, note that the LSC is not an essential component of the proposed method. Its inclusion helps to generate near-optimal paths. The computational complexity of the replanning heuristic depends on the motion planner used for replanning. We proposed the neural replanning and hybrid replanning methods. Since the neural replanner is executed for fixed number of steps, the complexity is $O(1)$. For the classical motion planner, we use RRT* which has $O(nlogn)$ complexity, where n is the number of samples in the tree [5]. Hence, for hybrid replanning, we can conclude that the proposed method has a worst case complexity of $O(nlogn)$ and a best case complexity of $O(1)$. Note that, MPNet: NR is able to compute collision-free paths for more than 97% of the cases, presented in Table I. Therefore, it can be said that MPNet will be operating with $O(1)$ most of the time except for nearly 3% cases where the RRT* needs to be executed for a small segment of overall path given by MPNet:NR. This execution of RRT* on small segments of a global path reduces the complicated problem to a simple planning problem which makes the RRT* execution computationally acceptable and practically much less than $O(nlogn)$.

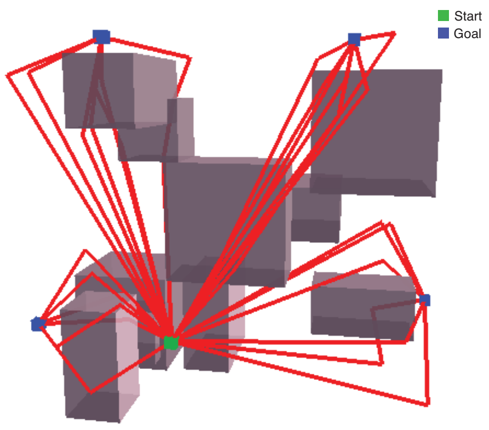


Fig. 5: MPNet generates multiple collision-free paths (red) between fixed start (green) and goal pairs (blue) in a finite-time due to its stochastic behavior.

VIII. Conclusions and Future work

In this paper, we present a fast and efficient Neural Motion Planner called MPNet. MPNet consists of an encoder network that encodes the point cloud of a robot's surroundings into a latent space,, and a planning network that takes the environment encoding, and start and goal robotic configurations to output a collision-free feasible path connecting the given configurations. The proposed method (1) plans motions irrespective of the obstacles geometry, (2) demonstrates mean execution time of about 1 second in all presented experiments, (3) generalizes to new unseen obstacle locations, and (4) has completeness guarantees.

In our future work, we plan to extend MPNet to build learning-based actor-critic motion planning methods by combining it with proxy collision checkers such as the Fastron algorithm [27], [28]. Another interesting extension would be to address the challenge of kinodynamic motion planning in dynamically changing environments.

References

[1] S. M. LaValle, *Planning algorithms*. Cambridge university press, 2006.

[2] T. Lozano-Perez, *Autonomous robot vehicles*. Springer Science & Business Media, 2012.

[3] R. Volpe, "Rover functional autonomy development for the mars mobile science laboratory," in *Proceedings of the 2003 IEEE Aerospace Conference*, vol. 2, 2003, pp. 643–652.

[4] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[5] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[6] A. H. Qureshi and Y. Ayaz, "Potential functions based sampling heuristic for optimal path planning," *Autonomous Robots*, vol. 40, no. 6, pp. 1079–1093, 2016.

[7] ——, "Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments," *Robotics and Autonomous Systems*, vol. 68, pp. 1–11, 2015.

[8] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, "Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments," *Robotics and Autonomous Systems*, vol. 108, pp. 13–27, 2018.

[9] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs," in *Robotics and Automation (ICRA), 2015 IEEE International Conference on*. IEEE, 2015, pp. 3067–3074.

[10] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural networks*, vol. 61, pp. 85–117, 2015.

[11] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *Journal of Machine Learning Research*, vol. 17, no. 39, pp. 1–40, 2016.

[12] A. Tamar, Y. Wu, G. Thomas, S. Levine, and P. Abbeel, "Value iteration networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 2154–2162.

[13] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[14] S. Calinon, F. D'halluin, E. L. Sauser, D. G. Caldwell, and A. G. Billard, "Learning and reproduction of gestures by imitation," *IEEE Robotics & Automation Magazine*, vol. 17, no. 2, pp. 44–54, 2010.

[15] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "Learning real manipulation tasks from virtual demonstrations using lstm," *arXiv preprint*, 2016.

[16] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 7087–7094.

[17] D. Berenson, P. Abbeel, and K. Goldberg, "A robot path planning framework that learns from experience," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 3671–3678.

[18] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*. Omnipress, 2011, pp. 833–840.

[19] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[20] I. A. Şucan and S. Chitta, "Moveit!", [online] available: http://moveit.ros.org."

[21] I. A. Şucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, December 2012, http://ompl.kavrakilab.org.

[22] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[23] L. Trottier, P. Giguère, and B. Chaib-draa, "Parametric exponential linear unit for deep convolutional neural networks," *arXiv preprint arXiv:1605.09332*, 2016.

[24] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*. IEEE, 2014, pp. 2997–3004.

[26] A. H. Qureshi and M. C. Yip, "Deeply informed neural sampling for robot motion planning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 6582–6588.

[27] N. Das and M. Yip, "Learning-based proxy collision detection for robot motion planning applications," *arXiv preprint arXiv:1902.08164*, 2019.

[28] N. Das, N. Gupta, and M. Yip, "Fastron: An online learning-based model and active learning strategy for proxy collision detection," in *Conference on Robot Learning*, 2017, pp. 496–504.

[29] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska *et al.*, "Overcoming catastrophic forgetting in neural networks," *Proceedings of the National Academy of Sciences*, vol. 114, no. 13, pp. 3521–3526, 2017.

[30] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.