Benchmarking Motion Planning Networks

Helei Duan, Kartik Gupta, Sridhar Thiagarajan, Yi Herng Ong

***Abstract*— With the advent of several real world robotic applications like self-driving and surgical manipulation, the importance of fast and efficient motion planning algorithms is ever increasing. Existing motion planning methods, both sampling based and search based, such as RRT* and A* have computational complexity which increases exponentially with the dimensionality of the problem. Recently, Motion Planning Networks (MPNet) [1] was proposed to mitigate some of the disadvantages of these traditional techniques, and provide constant time and near-optimal solutions. Although they showed promising results on several simulated tasks, the utility of this algorithm on real-world robotic tasks is yet to be ascertained. Real world tasks present several challenges, ranging from stochasticity to uncertainty, which is seldom the case in simulators. Our main contribution is a comparison of MPNet with traditional motion planning algorithm on a real-world robotic manipulation task. Also, we propose a novel architectural extension where we discard the decoder of the autoencoder, and propose to train the system end-to-end. Our results show that this new method results in drastically faster training of MPNet.**

I. Introduction

Robotic motion planning is a field aimed at computing collision-free paths from start to goal in given configuration space. These sequential decision making tasks have a wide variety of applications, in fields ranging from robotics to surgical manipulation [2] [3]. Traditionally, sampling based motion planning algorithms like RRT* [4], as well as search based motion planning algorithms have been used for these. Although they have highly desirable properties like completeness and optimality, they lack in their ability to provide constant time solutions and scale poorly with the dimensionality of the space.

In this work, we focus on evaluating a deep learning based iterative motion planning algorithm, called MPNet (Motion Planning Network) [1], on real world manipulation tasks. MPNet can plan trajectories in a cluttered 3D environment for various start and multiple goal configurations. For each start and goal pair, MPNet can generate multiple collision-free paths in finite time. MPNet consists of two components: an obstacle-space encoder and a path generator. The first model is a Contractive AutoEncoder (CAE) [5], which embeds the point cloud of the observed obstacles in the environment into a latent space. The second neural network learns to do motion planning for the given embedded environment, start and goal configuration. We discuss this in more detail in the algorithm description section.

Although Motion Planning Networks have been tested on a variety of simulated tasks with varying utility, their performance has not been tested yet on real world tasks. Real world tasks often induce a degree of stochasticity which is seldom matched in the simulation. We begin by replicating the literature's result of this network in 2D environments. Subsequently, we test the algorithm on a real-world robotic manipulation task, evaluating its performance. In order to benchmark MPNet's performance, we also evaluate traditional sampling based planners like RRT* on the same task. We use the Open Motion Planning Library [6], a library with open source implementations of the sampling based algorithms used as a benchmark.

[1]All are with Department of Mechanical, Industrial and Manufacturing Engineering, Oregon State University, Corvallis, USA

II. Related Work

Most of the motion planning algorithms are suffering from the trade-offs between computational efficiency and optimality [7] [8] [9] [10] [11] [4]. When developing an algorithm that is computationally efficient, sampling-based motion planning algorithms such as RRT [8] [9] [10] could plan a complete yet non optimal path in real-time. An optimal version of RRT called RRT* could provide optimal path using sampling-based method, but the performance and efficiency start to scale down due to the curse of dimensionality [1] [9].

The research in using a neural network for motion planning became stagnant because of the lack of data and the undeveloped architecture in the deep neural network in early 2000s [12]. As deep learning architecture is becoming mature, several neural network frameworks are established for path generation in either static or dynamic environment. One active field in deep learning based manipulation is reinforcement learning [13] [14] [15]. Although reinforcement learning algorithms are powerful, they often take large amounts of data during the training stage, and most importantly, they need to explore the environment while learning a policy. This can be dangerous in environments where failed trajectories can be costly. An end-to-end reinforcement learning approach was proposed which uses an algorithm called Qt-opt for robot manipulation for grasping [13]. It is an instance of a general reinforcement learning algorithm called Q-learning. Although they show successful results, letting agents or robots interact with the environment during exploration is not always feasible or advisable.

The most relevant work in terms of deep learning for motion planning is Motion Planning Networks [1]. They demonstrate promising results in planning optimal paths efficiently at several different dimensions. The proposed method encodes the workspace of the robot directly from a point cloud measurement, using an autoencoder, and generates the end-to-end collision-free paths for the given start and goal

configurations. The results show that MPNet is not only consistently computationally efficient in all environments but also generalizes to completely unseen environments. Nonetheless, MPNet failed to show its performance in real-world applications. Therefore we aim to realize MPNets in real robot and benchmark the performance with state-of-the-art motion planning algorithms (RRT, RRT*, A*, D* etc) on manipulation task using Kinova Jaco arm [16].

III. Problem Statement

MPNet only shows a fast and robust motion planning in simulation, but do not provide implementation results on the hardware. Also, the authors do not provide the joint and task space trajectories of the simulated Baxter arm during the simulation test case discussed in the paper. Thus, we want to benchmark the feasibility of this algorithm both on the simulation and hardware, with a robotic manipulator platform (7DOF robotic arm) to validate this proposed method. The essential problem that we are implementing, validating, and testing is improving the solution finding time compared to classical motion planning problem.

Problem (Near-optimal Motion Planning with Time Efficiency): *Given a triplet $\{X, X_{free}, X_{obs}\}$, an initial state x_{init} and a goal region $X_{goal} \subset X_{free}$, find a path solution $\tau \in X_{free}$ such that $\tau(0) = x_{init}$ and $\tau(end) \in X_{goal}$ and also spends less time to plan before motion execution*

Specifically in our problem, the inputs/outputs, the controlled/uncontrolled parts are defined as,

$$\begin{aligned} \text{Inputs} &= X_{goal} \rightarrow \text{ Desired goal in the task space} \\ \text{Outputs} &= \tau \in X_{free} \text{ and robot joint states at each } \tau \\ \text{Controlled} &= \text{joint positions of the robotic arm} \\ \text{Uncontrolled} &= \text{uncertainties in sensors and calibrations} \end{aligned}$$

IV. Algorithm Description

We refer to the original work in [1] for the complete algorithm description. We provide a brief description of the overall method here.

The proposed method, called MPNet (Motion Planning Networks), consists of two stages. The first phase comprises of the offline training of the neural networks, while the second part comprises the online motion planning algorithm.

A. Offline Training

The offline training module consists of two neural networks. The first model is a Contractive AutoEncoder (CAE) which embeds the point cloud of the observed obstacles in the environment into a latent space. The second neural network learns to do motion planning for the given embed environment, start and goal configuration.

1) Contractive AutoEncoder (CAE): : The contractive autoencoder is used to embed the obstacles into an invariant and robust feature space $Z \in \mathbb{R}^m$, where $m \in \mathbb{N}$ is the dimensionality of the feature space. This network is trained so as to minimize the reconstruction loss.

The model architecture consists of the encoding and the decoding function, of 3 linear layers and a output layer. Each linear layer is followed by the Parametric Rectified Linear Unit. The decoder function is an inverse of the encoding unit. The model takes a $1400 \times d$ sized vector as input where 1400 are the points along each dimension, and $d \in \mathbb{N}_{\geq 2}$ is the dimension of a workspace. For 2D workspaces, the three layers have 512,256,128 hidden units respectively. We intend to experiment with changing the size of the layers as well as the effect of adding another layer the CAE so to incorporate higher dimensional environment representation for the 6D arm robot arm. This seems critical, as [14] have shown that task specific encoding of task-space gives a significant boost to the overall system performance.

2) Deep Multi-Layer Perceptron (DMLP): : This neural network is the backbone of the planning algorithm. Given the obstacle encoding Z generated by the CAE, the current state x_t and the goal state x_T, the DMLP predicts the next state x_{t+1} which would lead the robot towards the goal region.

The DMLP is trained in a supervised manner using the near-optimal, feasible trajectories generated RRT* algorithm in various environments. The training objective is to minimize the mean-squared-error (MSE) between the predicted states $\hat{x}_{t+1}$ and the actual next states given by the RRT* x_{t+1}. For transfer to real robot, we plan to experiment by adding Gaussian noise to the initial states while aiming for the same destination point. This can be visualized as creating a valley in which the robot is invariant to the small disturbances.

The model architecture of the DMLP consists of 9 layers where each layer is a sequence of a linear layer, a Parametric Linear Unit (PReLU) and Dropout. This is followed by the tenth and eleventh layer which do not use Dropout and transform the inputs to 32 units. This is taken by the final layer and transformed to the dimensions of the robot configuration space. Generally, the Dropout layers are dropped during the testing phase. However, here the Dropout layers are not dropped since it adds stochasticity to the MPNet.

B. New Offline Training

We do several attempts to make some improvements or differences with the original MPNet. One of the successful approaching is to modify the existing MPNet architecture. We keep other parts same with the original MPNet, but abandon decoder from the original architecture and proceed to end-to-end training on different environments with paths. Without tuning DMLP after finish tuning Contractive AutoEncoder in the original architecture, we combine encoder with original DMLP to form a new model of DMLP and tune both of them in the same time. The new model of DMLP still needs the obstacle encoding Z generated by encoder, the current state x_t and the goal state x_T as the input. The new model of DMLP predicts the next state x_{t+1} for motion planning. To tune the DMLP model, we still use mean-squared-error (MSE) to calculate the loss between the predicted states $\hat{x}_{t+1}$ and the actual next states given by the RRT* x_{t+1}. Meanwhile, it will calculate the loss of encoded environment, which is to tune the encoder.

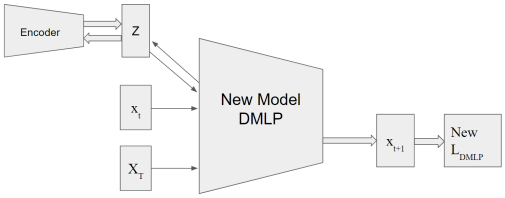

Fig. 1. Modified MPNet Architecture

C. Online Path Planning

The online path planning phase utilizes the neural models trained in the offline phase for motion planning in cluttered and complex environments. 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 the CAE, DMLP and the online motion planner is shown in 2. The overall Online Path Planning methodology is shown in 3. Essentially the path planning step consists of recursively utilizing the DMLP to output the next path step with additional checks and iterations for robustness. We do not intend to modify this section of the algorithm. The reader is encouraged to refer to [1] for a more detailed explanation.

V. Methodology

This section will explain our primary methodologies for this project. The proposed methods are mainly based on the original work from [1], however, the methods and steps state below are not meant to be the final approach to accomplish this project.

Our aim is to realize the MPNet method onto a real hardware, while performing non-trivial task, which has higher dimensionality and hardware realization. For example, the expected test case is the robot end-effector moves from one place to another and completes the motion planning without hitting the randomly placed obstacles.

A. Data generation

Two types of data will be generated in this project. We use simulation engine provided by [1] to generate 2D data with points as robot and blocks as obstacles. 3D data will be generated using Kinova Jaco arm in MoveIt and Gazebo simulation engines. We will also use rigid blocks to form 3D cluttered environment. We aim to generate 1000 3D cluttered environments and 10000 paths in both 2D and 3D for training samples.

B. Training of Autoencoder and DMLP

To validate the feasibility of MPNet, we start at low dimensional planning problem. We will use 2D data to train the autoencoder and DMLP, and test the entire motion planning pipeline with random start and goal configurations. Same process will then be repeated using 3D data generated by Kinova arm from Gazebo and MoveIt. Simulation engines with Kinova Jaco arm are shown in Figure **??** and Figure 4.

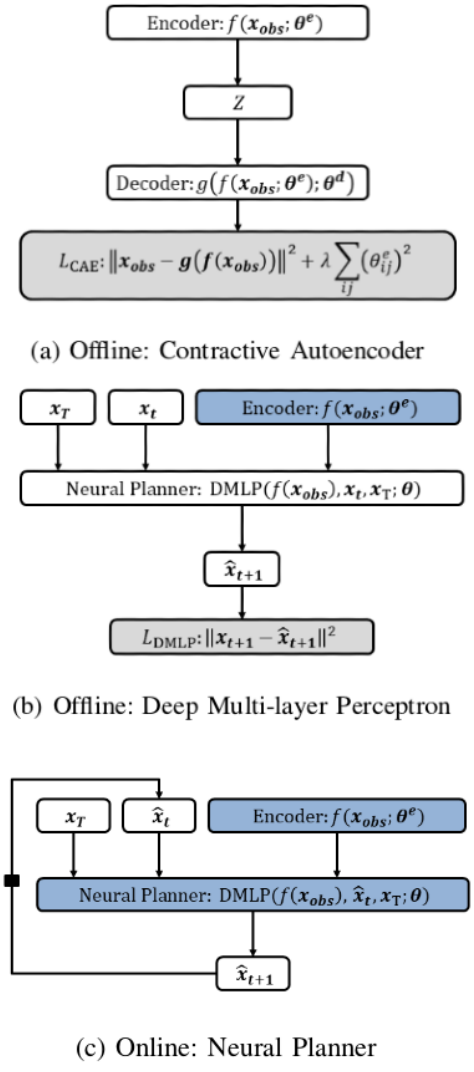

Fig. 2. The offline and online phases of MPNet. The grey shaded blocks indicate the training objectives. The sky-blue and white blocks represent frozen and non-frozen modules, respectively. The frozen modules do not undergo any training. [1]

C. Hardware Test

After the training and testing are complete in simulation, we will use the trained network and motion planner on actual Kinova Jaco arm to complete a manipulation task in cluttered space. Computational time and path cost will be recorded in the process.

D. Benchmark with conventional MP algorithms

We will use state-of-the-art motion planning algorithms to compute path for Kinova Jaco arm on the same manipulation task, in order to benchmark their computational efficiency and optimality of resulting paths with MPNet [1]. In this project we chose RRT* as the main motion planning algorithm we are going to benchmark with.

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

$\text{Z} \leftarrow f(\boldsymbol{x}_{\mathbf{obs}})$
$\tau \leftarrow \text{NeuralPlanner}(x_{\text{init}}, x_{\text{goal}}, Z)$;
if τ **then**
  $\tau \leftarrow \text{LazyStatesContraction}(\tau)$
  if IsFeasible(τ) **then**
    return τ
  else
    $\tau_{\text{new}} \leftarrow \text{Replanning}(\tau, \text{Z})$
    $\tau_{\text{new}} \leftarrow \text{LazyStatesContraction}(\tau_{\text{new}})$
    if IsFeasible(τ_{new}) **then**
      return τ_{new}
    return $\varnothing$

Algorithm 2: NeuralPlanner($x_{\text{start}}, x_{\text{end}}, Z$)

$\tau^{\text{a}} \leftarrow \{x_{\text{start}}\}; \tau^{\text{b}} \leftarrow \{x_{\text{end}}\}$;
$\tau \leftarrow \varnothing$;
Reached $\leftarrow$ False;
for $i \leftarrow 0$ ***to*** N **do**
  $x_{\text{new}} \leftarrow \text{DMLP}\left(Z, \tau^{\text{a}}(\text{end}), \tau^{\text{b}}(\text{end})\right)$
  $\tau^{\text{a}} \leftarrow \tau^{\text{a}} \cup \{x_{\text{new}}\}$
  $\text{Connect} \leftarrow \text{steerTo}\left(\tau^{\text{a}}(\text{end}), \tau^{\text{b}}(\text{end})\right)$
  if Connect **then**
    $\tau \leftarrow \text{concatenate}(\tau^{\text{a}}, \tau^{\text{b}})$
    return τ
  SWAP($\tau^{\text{a}}, \tau^{\text{b}}$)
return $\varnothing$

Fig. 3. Online Path Planning Algorithm. [1]

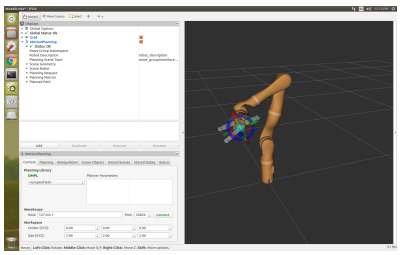

Fig. 4. Kinova Jaco arm in MoveIt environment. We use RRT* with this environment to generate data for MPNet.

VI. Results

A. Simulation

Both of the original and the modified architecture MPNet converged their loss after training for 500 epochs. In this section we show three sets of results. To evaluate the performances of both original and modified MPNet against RRT*, we randomly pick a few environments from 30000 environments to plan paths with randomly chosen start and goal configurations. We also compared path planning performance between original and modified MPnet.

1) Training of both architectures: In figure 9, we demonstrates how the losses were changing in both original and modified MPnet architectures. Note that both orignal and modified MPnet training converged to a total loss of 16 and 3.2 after 500 epochs respectively.

2) Original MPnet vs. RRT:* In one of the planning scenario, original MPnet computes an optimal path (Figure 6) in 0.26 s, compared to RRT* which took approximately 0.4 s (see Figure 5).

3) Modified MPnet vs. RRT:* Figure 8 shows that a path planned by modified MPnet architecture is surprisingly similar to RRT* in this scenario, which could not only guarantee optimality but also reduce the computational time (0.08 s), compared to RRT* (0.48 s). Note that this is only a path planning scenario.

4) Original MPnet vs. Modified MPnet: In overall path planning, original MPnet architecture can produce 90 percent feasible paths from 60 random environments, while modified MPnet architecture can generate 98 percent feasible paths from 60 random environments. Average planning time for each path is about 0.43 s for original MPnet, whereas modified MPnet takes about 0.86 s on average to plan each path.

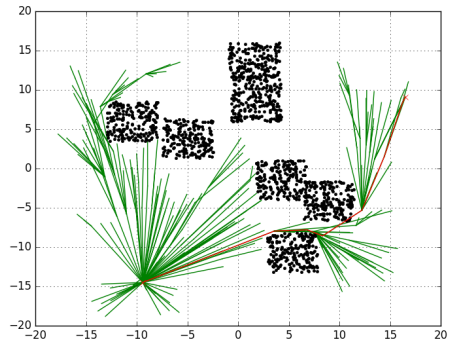

Fig. 5. A sample path computed using RRT* in 0.41 seconds.

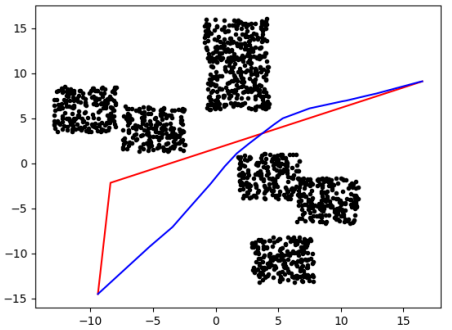

Fig. 6. MPNet path planned in 0.26s, whereas RRT* takes around 0.4 seconds. (Blue line - RRT* path as ground truth, red line - MPNet path).

B. Hardware Experiments

We used Jaco arm to validate path feasibility of MPNet output using both original and new architectures. For each

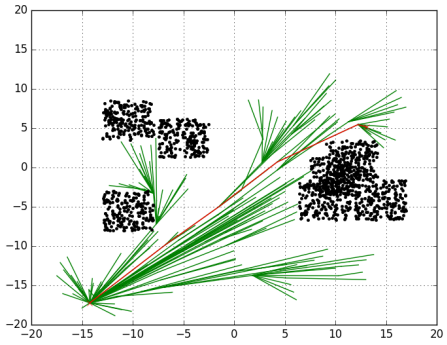

Fig. 7. RRT* path planning using 0.48s to finish.

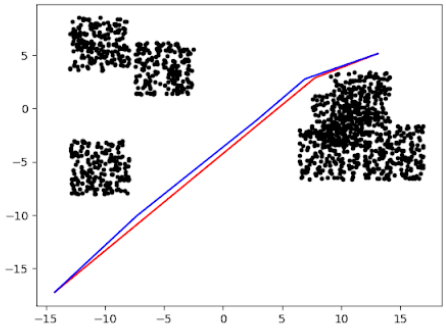

Fig. 8. Modified architecture MPNet path computed in 0.08s is indicates in red. RRT*'s path is indicated in blue.

case, MPNet generated collision free waypoints based on the input environment for high-level planning. We created a move arm node in ROS to receive the generated waypoints and move between each pair of points. During movement between each waypoints, this node calls OMPL through Moveit to use RRT to complete small distances. Figure 10 showed our experimental setup based on the 2D environment discussed in previous sections. The arm successfully followed the desired waypoints form MPNet and reached goal positions.[1]

VII. Discussions

A. Architecture Change to Original MPNet Setup

Architecture changing may lead to really bad results in many cases. However, in this project, we have the better results on modified MPNet, when comparing with the original MPNet model. Although, the average time consumption of path planning increase from 0.43s to 0.86s (original MPNet has more infeasible paths which may cause less planning time), the change of eliminating decoder part and use end-to-end training improves a lot. Baseed on our resutls, the percentage of finding feasible paths increased from 90% to 98% on our modified MPNet, and it generates more approaching optimal paths than the results of the original MPNet. The

[1]For the video for this test, click *here*.

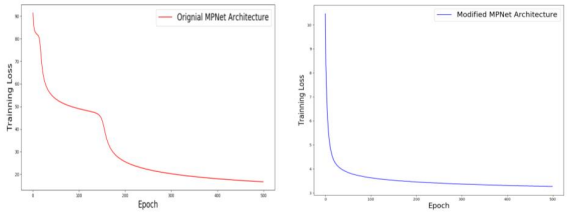

Fig. 9. Total loss of both original and modified architecture over 500 epoch. Red is original MPnet, blue is modified MPnet.

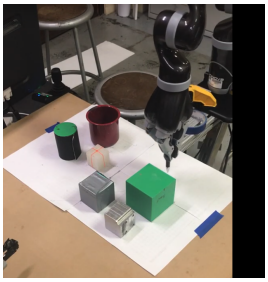

Fig. 10. Experimental Setup with Jaco Arm

trade-off between computational efficiency and optimality always exists. Based on this project, the new architecture performs much better on optimatliy with acceptable sacrifice on computational time. The reasons of improvements on new architecture comparing with the original MPNet include the limitation of Contractive AutoEncoder model and training efficiency of new model, etc. For more precise analysis, we need to do more different environments and paths training on both original and modified MPNet

B. Implementations on 7DOF Kinova Jaco Arm

We intended to implement this approach in 3D to validate whether it is able to move a 7DOF arm in space. Our proposed pipeline to accomplish this task consists of new method on data generation, training process, and also a local planner to accomplish the annotated procedures in Figure 11.

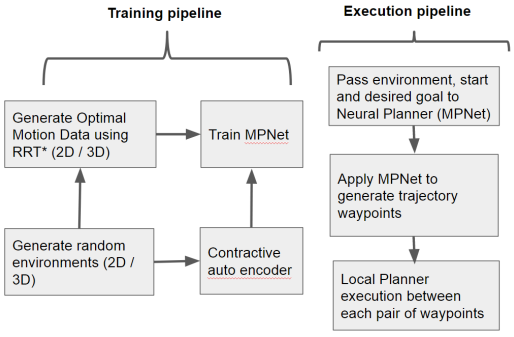

Fig. 11. Proposed System Pipeline

1) Data generation for 3D: In order to implement MPnet framework on 7 DOF arm, we need to generated significant amount of data to train the network. Refer to the sample size generated by the authors of original MPnet, we decided to randomly generated 1000 paths using RRT* in one random 3D environment, and repeated the process for a total of 50 environments. Instead of recording the end effector pose in Cartesian space, we recorded 7 joint states throughout the entire trajectory. Note that such way of data generation could work well at generalizing among environments and path planning.

Simulation in MoveIt and ROS were used for data generation of 7 DOF kinova arm. However, due to limitation of disk space and ROS, we failed to generate data automatically through MoveIt for all 50 environments. We then tried generating 10000 path data in only one environment but the entire process was unexpectedly time consuming. The reason is because RRT* that set by MoveIt is using high iterations to find an optimal path and this can take longer time for planning.

2) Training for 7DOF Arm: We trained a DMLP in order to generate waypoints for the Kinova arm. The DMLP structure in the same as discribed above, but with an additional linear layer. The inputs to this DMLP are the current joint pose and the target joint pose of the arm and it outputs the the next desired joint pose along the way to reach the goal pose. This was trained on 1800 paths generated on one environment consisting of 41000 waypoints in total. To augment our training data, we reversed all paths to double the training samples. Training was done with MSE loss with Adagrad optimisation. Test samples predicted waypoints which were collision free and ,by visual inspection, seemed to be along the path towards the goal.

3) Local Planner in 3D: When we extended the problem to 3D environment, the robot also needs local planner to know how to move between waypoints $x_1 \rightarrow x_2$, or between a pair of joint states $q_1 \rightarrow q_2$, because high-level MPNet only gives collision free path without knowing the exact configurations of the robot. Thus, in order to move between each waypoint (joint states), a local planner is needed to handle self-collision and environment collision check. The motivation of a local planner also comes from a similar re-plan function mentioned in author's original method. The re-plan function tries to re-generate waypoints (joint states) from MPNet if obstacles are between two existing points. In author's method, this is simply calling an Euclidean distance between two points and check if obstacles lie in between. However, the complexity of 3D re-plan is not simply checking obstacle. This re-plan should check self-collision, environment collision, and feasibility of the IK solutions between two waypoints (or joint states). This requires interfaces with Moveit! for all collisions check and IK feasibility.

If we developed such local planner, it will only be used when high-level planning requires queuing additional points from MPNet because of the current points fail at the local planner. However, calling a local planner in such cases would presumably have similar time to perform an near optimal path planning locally.

VIII. Conclusion and Future Work

We implemented and tested motion planning networks on a real robot and several synthetic 2D domains, benchmarking its performance with a traditional motion planning algorithm (RRT*). We consistently found MPNet to be faster, giving real-time, collision free paths for several goal and start configurations. We improved the training process of MPNet by finding that training it end-to-end, rather than training the encoder separately gives better performance and results in faster training time. Hardware experiments were run on the Kinova Jaco arm to verify MPNet's robustness for planar manipulation tasks, and we discussed the challenge with implementing a fully 3D motion planning task in hardware with our current framework.

There are several possible extensions to MPNet that we would like to explore in the future. One interesting extension we would like to pursue is the application of MPNet like algorithms for multi-agent path planning problems. Traditional algorithms like M* scale poorly with dimension, and hence there is scope for faster real-time deep learning based algorithms. Another line of future work would be to come up with an extension of MPNet for partially observable environments. Several robots, especially in marine and aerial domains have to deal with uncertainity in their environment, and coming up with a Bayesian neural network which could reason and plan around these uncertainities in real-time would be a good avenue for further research.

References

[1] A. H. Qureshi, M. J. Bency, and M. C. Yip, "Motion planning networks," *arXiv preprint arXiv:1806.05767*, 2018.

[2] M. Yip and N. Das, "Robot autonomy for surgery," *arXiv preprint arXiv:1707.03080*, p. 1, 2017.

[3] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. P. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Transactions on Control Systems Technology*, vol. 17, no. 5, pp. 1105–1118, 2009.

[4] 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.

[5] 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.

[6] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[7] M. Likhachev, G. J. Gordon, and S. Thrun, "Ara*: Anytime a* with provable bounds on sub-optimality," in *Advances in neural information processing systems*, 2004, pp. 767–774.

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

[9] I. Noreen, A. Khan, and Z. Habib, "A comparison of rrt, rrt* and rrt*-smart path planning algorithms," *International Journal of Computer Science and Network Security (IJCSNS)*, vol. 16, no. 10, p. 20, 2016.

[10] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 1478–1483.

[11] S. Koenig and M. Likhachev, "D^* lite," *Aaai/iaai*, vol. 15, 2002.

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

[13] D. Kalashnikov, A. Irpan, P. Pastor, J. Ibarz, A. Herzog, E. Jang, D. Quillen, E. Holly, M. Kalakrishnan, V. Vanhoucke *et al.*, "Qt-opt: Scalable deep reinforcement learning for vision-based robotic manipulation," *arXiv preprint arXiv:1806.10293*, 2018.

[14] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[15] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[16] V. Maheu, P. S. Archambault, J. Frappier, and F. Routhier, "Evaluation of the jaco robotic arm: Clinico-economic study for powered wheelchair users with upper-extremity disabilities," in *2011 IEEE International Conference on Rehabilitation Robotics*. IEEE, 2011, pp. 1–5.