2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
Macau, China, November 4-8, 2019

Fast Motion Planning via Free C-space Estimation Based on Deep Neural Network

Xiang Li[1], Qixin Cao[2], Mingjing Sun[3], Ganggang Yang[4]

***Abstract*— This paper presents a novel learning-based method for fast motion planning in high-dimensional spaces. A deep neural network is designed to predict the free configuration space rapidly given the environment point cloud. With a generated roadmap as an approximate view of the free C-space, LazyPRM is applied to find and check the path with A* search. Due to the application of LazyPRM, the presented method can preserve probabilistic completeness and asymptotic optimality. The new algorithm is tested on a 3-DOF robot arm and a 6-DOF UR3 robot to plan in randomly generated obstacle environments. Results indicate that compared to planners including PRM, RRT*, RRT-connect and the original LazyPRM, our method is of the lowest time consumption and relatively short path length, showing good performance on both planning speed and path quality.**

I. INTRODUCTION

Motion planning algorithms aim to generate collision-free motions for complex bodies from a start to a goal position among a collection of static obstacles [1]. These algorithms are widely used in robot navigation, robotic surgery and robotic manipulation.

Sampling-based planners are very popular methods for solving motion planning problems and can guarantee probabilistic completeness and asymptotic optimality. The acceleration of sampling-based motion planning is one optimization direction that researchers focus on [2] [3] [4] . Faster planning speed can be beneficial for variable environments, specifically for robots working in changing environments or robot arms on a moving platform that can change their positions. Recent work shows that using environment data to set up a heuristic single-query planner can increase planning speed [5].

The key idea of sampling-based planners is to exploit the free C-space by sampling and collision checking between the robot and the environment. This is because an explicit representation of the obstacles in the configuration space may result in an excessive computational burden in high dimensions, and in environments described by a large number of obstacles [6]. Neural networks can perform well when fitting complex functions and might be helpful to predict the free C-space with the information of the obstacles.

This paper presents a novel learning-based method to accelerate sampling-based motion planning. A deep neural network is designed to estimate the free C-space from the point cloud of an environment. Figure 1 shows a prediction of the free C-space of a 3-DOF robot arm in an obstacle environment. PointNet [7] is adopted in our network for point cloud processing. A roadmap vertex set is generated directly by the DNN with Earth Moving Distance (EMD) as the loss of the network to maintain the unordered feature of the graph vertices. Then the output vertices are rewired, and paths are found by A* search algorithm. LazyPRM [8] is applied to reduce the number of collision checking and to ensure probabilistic completeness and asymptotic optimality. The roadmap need only to be generated once in each different environment, and can be used for multiple planning problems in the same environment, making our multi-query planner suitable for mobile manipulators where the robot arm often faces different environments and may need to perform multiple tasks in the same environment.

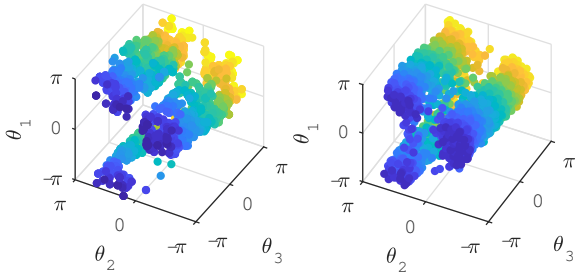

Fig. 1. Free C-space predicted by deep neural network for a 3-DOF robot arm in an obstacle environment. Left is the ground truth roadmap vertices and on the right is the prediction. The input of the network is the point cloud of the environment and the output is a set of robot configurations indicating the C-free.

The algorithm is tested on a 3-DOF robot arm and a 6-DOF UR3 robot to plan in randomly generated obstacle environments. Results indicate that compared to planners including PRM, RRT*, RRT-connect and original LazyPRM from OMPL [9], averagely, with a well-generated roadmap from the network, our algorithm can find a relatively low-cost path for the planning problem in the least time.

This paper begins with the background of sampling-based planning. Section II then shows the classical sampling-based planners and previous work for sampling-based fast planning and learning-based planning. In Section IV we describe our method in detail. Experiments are shown in Section V. Finally Section VI concludes this paper.

[1,3,4] Master students of School of Mechanical Engineering, Shanghai Jiao Tong University. lixiang3.1416@qq.com, {smj_1024,yanggang}@sjtu.edu.cn

[2] Professor of School of Mechanical Engineering, Shanghai Jiao Tong University. qxcao@sjtu.edu.cn

II. RELATED WORK

A. Sampling-based motion planning

Sampling-based planners are one of the main categories of robot motion planning algorithms. Instead of providing the exact geometric model of the C-space, sampling-based planners try to describe the C-space by sampling and checking whether these sampled robot configurations collide with the obstacles. In this way, sampling-based planners can solve the planning problem in a more computationally efficient way. The exchange of the computational efficiency is that sampling-based planners provide a weaker, but still interesting, form of completeness: probabilistic completeness, which means if a solution path exists, the planner will eventually find it [1].

Sampling-based planners can be mainly divided into two classes: single-query planners and multi-query planners. Rapidly-exploring random tree (RRT) algorithm [10] is the main representative of single-query planners. This algorithm solves a planning problem by trying to grow a tree from start configuration to the goal. Its bidirectional derivative RRT-connect [11] is famous for its fast speed and another derivative RRT* [12] can solve motion planning problems with asymptotic optimality. Multi-query planners can plan faster than single-query planners with a precise pre-generated roadmap, but lose this advantage in new environments. The probabilistic roadmap (PRM) method [13] is a representative of the multi-query planners. Its asymptotic optimal derivative is PRM* [12]. Another derivative of PRM, LazyPRM [8], can reduce the planning time using lazy collision checking.

B. Fast motion planning

Various research has been done to accelerate motion planning, mainly for robots working in unstable environments, specifically, a robot in an environment with moving obstacles or the robot arm is being moved, for example, a mobile manipulator. One main approach was to modify the algorithms to re-use information to efficiently update motion paths when the environment changes [14]. Paper [2] presented Dynamic Roadmap (DRM) algorithm, which accelerates planning speed by workspace discretization and using pre-generated collision checking data, and was able to do real-time planning in changing environments [3]. However, this algorithm cannot guarantee completeness because the workspace needs to be discretized into cells for fast collision checking, and will face difficulty while approving accuracy since the number of cells to check increases with the reciprocal of the third power of cell size. The method in [4] proposed another solution that can guarantee probabilistic completeness. LazyPRM was adopted as the planner of this method planning with a pre-generated roadmap and online estimation of vertex collide possibility. But collision checking results in the same environment are needed to set up a precise model, and the prediction model can lose effectiveness in a new environment.

Recent years, machine learning technologies have been widely concerned in robotics and were adopted to accelerate robot motion planning. Paper [15] tried to optimize the sampler by clustering-based unsupervised region identification with a small roadmap and was able to apply different sampling strategies according to the feature of the region. Another method, Lightning Framework [16], tried to reduce computation time by learning from experience. This framework consists of two main modules running in parallel, a planning-from-scratch module, and a module retrieves and repairs paths stored in a path library. However, the path library may not perform well for new environments where obstacles change a lot. Recent work in paper [17] used conditional variational autoencoder to accelerate the planning process by non-uniform sampling strategies that favor sampling in regions where an optimal solution might lie. The network is trained for a specific environment so that this method is not able to address motion planning problems in different environments. Another recent work, [5], presented a deeply informed approach for the sampler to predict the well-optimized results generated by RRT* with the data of the environment given in the form of a point cloud. However, the deep sampler can only produce one sample under each condition, leading to running neural network forward propagation multiple times in solving one problem. Its performance is highly related to the training quality of the sampler; therefore, the performance of the whole algorithm is sensitive to the quality of the dataset. Since RRT* is an asymptotic optimal planner and may cost much time to get well-optimized and each environment in the dataset need several generated paths, this algorithm might suffer from the difficulty of dataset generation.

III. PROBLEM FORMULATION

The goal of this work is to find a relatively low cost solution for motion planning problems in a short time in different environments. The formulation of the motion-planning problem follows as presented by Karaman and Frazzoli [6]. Let $\mathcal{X} = (0,1)^d$ be the configuration space, where $d \in N$, $d \geq 2$. Let $\mathcal{X}_{obs}$ be the obstacle region, such that $\mathcal{X} \backslash \mathcal{X}_{obs}$ is an open set, and denote the obstacle-free space as $\mathcal{X}_{free} = cl(\mathcal{X} \backslash \mathcal{X}_{obs})$, where $cl(\cdot)$ denotes the closure of a set. The initial condition x_{init} is an element of $\mathcal{X}_{free}$, and the goal region $\mathcal{X}_{goal}$ is an open subset of $\mathcal{X}_{free}$. A path planning problem is defined by a triplet $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$.

Problem 1 (Feasible path planning) Given a path planning problem $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$, find a feasible path $\sigma : [0,1] \rightarrow \mathcal{X}_{free}$ such that $\sigma(0) = x_{init}$ and $\sigma(1) \in cl(\mathcal{X}_{goal})$, if one exists. If no such path exists, report failure.

Let a set of all feasible paths be denoted as Σ. Let a cost function $c(\cdot)$ computes a summation of Euclidean distances between all the consecutive states in a path Σ.

Problem 2 (Optimal path planning) Given a path planning problem $(\mathcal{X}_{free}, x_{init}, \mathcal{X}_{goal})$ and a cost function $c : \Sigma \rightarrow \mathbb{R}_{\geq 0}$, find a feasible path σ^* such that $c(\sigma^*) = min\{c(\sigma) : \sigma \ is \ feasible\}$. If no such path exists, report failure.

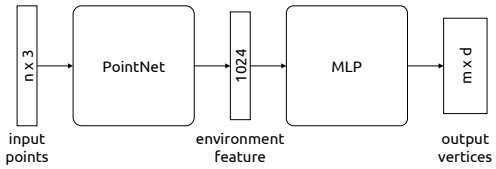

Fig. 2. Architecture of the roadmap generation network.

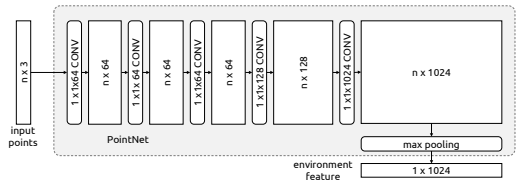

Fig. 3. Architecture of PointNet (vanilla) [7].

IV. ALGORITHM IMPLEMENTATION

In this section, we present our algorithm, including a deep neural network that can generate C-space roadmap vertices for different environments to accelerate motion planning and a planner that can utilize the results of the roadmap generation network (RGN). This section also includes specific information about the generation of training dataset.

A. Network architecture

Our network consists of two parts: the first part to extract features of the environment point cloud and the second to generate vertices from the features to build a C-space roadmap (Figure 2).

In the first part of our network, PointNet [7] is used to extract features of the environment point cloud that can be obtained from 3-D sensors (Figure 3). PointNet is an architecture that can effectively process unordered point sets by extracting point features through convolutional layers and aggregating information from all the points through a max-pooling layer. It is a simple but highly efficient and effective architecture originally created for object classification, part segmentation and scene semantic parsing. In order to use this architecture to solve motion planning problems, the simplest version of PointNet, the vanilla version (without point cloud transformation layers) is used for higher performance. This part of network consumes a $n \times 3$ data matrix where n is the number of input points and outputs a 1×1024 vector as the feature of the environment point cloud.

The second part of the network aims to reconstruct the C-space roadmap from the feature of the environment point cloud. A multilayer perceptron (MLP) is applied in the structure with ReLU activation function. The layer number of the MLP will be further discussed in later sections. Each fully-connected layer is followed by a dropout layer (Figure 4). The input of this part of the network is the 1×1024 feature vector generated by PointNet. The output is a $m \times d$ matrix where d is the dimension of the C-space and m is the number of output roadmap vertices.

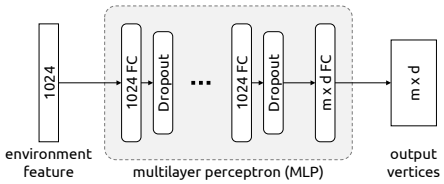

Fig. 4. Architecture of the multilayer perceptron module.

The Earth Mover's Distance (EMD) [18] between the vertices of dataset roadmap and the reconstructed roadmap is adopted as the loss of the network. EMD is the solution of a transportation problem and is defined as the least cost that one set transforms to the other. For two equally sized subsets $S_1 \subseteq R^d$; $S_2 \subseteq R^d$, their EMD is defined by

$$d_{EMD}(S_1, S_2) = \min_{\phi: S_1 \to S_2} \sum_{x \in S_1} \|x - \phi(x)\|_2$$

where ϕ is a bijection [19]. The EMD distance is symmetrical among all points in the set and can be easily adopted to high-dimensional spaces, which is suitable for our application. In our method, the distance between two configurations is set a weighted Euclidean distance with coefficients differ among joints.

B. Dataset generation

In order to train the proposed network, the needed dataset includes different environment point clouds and the corresponding roadmap vertices. The obstacle environment is generated randomly from four kinds of basic 3D objects (sphere, cube, ring and cylinder) (Figure 6). Points are only placed on the surface of the objects to simulate real data collected from 3-D cameras. This can also be more computational efficient while enhancing accuracy because in this case number of voxels to check increases with the reciprocal of only the second power of voxel size. The number of each kind of object in one environment is random and each object is transformed by a random homogeneous transformation matrix. Points too far away that will definitely not collide with the robot are removed. Then, for computation efficiency, the point cloud is processed by a PCL VoxelGrid filter. After all the process, the number of points varies from approximately 5000 to 40000. Considering the balance of a smaller neural network for higher performance and enough points for roadmap prediction accuracy, 16384 (=2^{14}) points are selected randomly in each piece of data. Points may be selected repeatedly when there are not enough in one point cloud; our network can handle repeated points well since the same points will not affect the feature that came out from a max-pooling layer.

After creating the obstacle point cloud, a roadmap in C-space is generated for this environment. As planning in open C-space regions is not the bottleneck of motion planning, a novel algorithm is presented for roadmap dataset generation (Algorithm 1) to bias samples to complex regions. Our roadmap generator starts with an empty vertex set V and

Algorithm 1: Roadmap dataset generator

$V \leftarrow \emptyset; E \leftarrow \emptyset; t \leftarrow 0;$
Start timing on t;
while $t < t_{max}$ **do**
  $x_{rand} \leftarrow RandomValidSample()$
  $X_{near} \leftarrow Neighbor(V, x_{rand}, N_{neighbor});$
  $X_{connectable} \leftarrow CheckMotion(x_{rand}, X_{near})$
  if $size(X_{connectable}) > size_{min}$ **then**
    $continue;$
  else if $SameComponent(X_{connectable})$ **then**
    $continue;$
  else if $InterConnectable(X_{connectable})$ **then**
    $continue;$
  else
    $AddVertex(x_{rand}, V, E)$
return V;

an empty edge set E, and generate time is limited for each environment. While not running out of time, a sampler first tries to sample a valid robot configuration x_{rand}. Then a neighbor finding function $Neighbor(\cdot)$ would give the $N_{neighbor}$ nearest vertices of x_{rand} in V, this result is stored in X_{near}. Then, the direct path from x_{rand} to every vertex in X_{near} is tested by the collision checking function $CheckMotion(\cdot)$ and connectable vertices are stored in $X_{connectable}$. After this, three conditions are checked sequentially and x_{rand} would be added to the graph if any condition is satisfied. The order of the three judgements is determined by computational cost, from low to high. The first condition is the vertex number in $X_{connectable}$. Small vertex number in $X_{connectable}$ means that many of near neighbors cannot be reached directly by x_{rand}, the C-space near x_{rand} is complexed and new vertices are needed. Checking the size of $X_{connectable}$ is quick since it is represented in one array in our algorithm. The second condition is whether all the vertices in $X_{connectable}$ belong to the same component of the graph. Vertices that can reach each other through edges are defined in the same component. x_{rand} will be added to the graph if it can connect different components. The judgement of this condition would only need to check the existing data. The third condition to check is whether all vertices in $X_{connectable}$ can directly reach each other. This condition also indicates the region complexity near x_{rand}. The computational cost is high because collision checking is needed in this process. When the time meets the limit, the program stops adding new configurations to the roadmap, all the vertices in V are stored together with the environment point cloud as a piece of data.

C. Planner for the generated roadmap

During online planning, the roadmap generation network generates a roadmap vertex set according to the environment data, then a planner is applied to find a path for the motion planning problem.

The planner we used is based on LazyPRM [8], a PRM-based motion planner that can reduce the planning time using lazy collision checking. At the start, the roadmap vertices generated by the neural network are loaded and wired to near vertices to build an original roadmap, the graph G. Then the start configuration x_{start} and goal configuration x_{goal} are added into G for initialization. During each loop, the planner checks whether x_{start} and x_{goal} belong to the same component. If so, an A* searcher will rapidly search the shortest path for the problem. The path found is then collision checked and the invalid vertices and edges of the path are removed. If x_{start} and x_{goal} belong to different components, a new vertex will be added to help solve the planning problem.

Except for the roadmap loaded at the beginning, the algorithm is the same with the LazyPRM [8] planner and can guarantee probabilistic completeness and asymptotic optimality as LazyPRM does. The application of A* algorithm gave our planner a feature that the originally found paths are relatively low-cost paths and this can contribute to good path quality.

Algorithm 2: Roadmap planner (x_{start}, x_{goal})

$G \leftarrow LoadRoadmap(); t \leftarrow 0; path \leftarrow \emptyset$
$AddVertex(x_{start}, G)$
$AddVertex(x_{goal}, G)$
Start timing on t;
while $t < t_{max}$ **do**
  while $SameComponent(x_{start}, x_{goal})$ **do**
    $path_{tmp} = A^*(x_{start}, x_{goal}, G)$
    if $path_{tmp}$ *is collision-free* **then**
      $path = path_{tmp}$
    else
      remove invalid vertices and edges in $path_{tmp}$
  if $path \neq \emptyset$ **then**
    break;
  $x_{rand} \leftarrow RandomValidSample()$
  $AddVertex(x_{rand}, G)$
while $t < t_{max}$ **do**
  $x_{rand} \leftarrow RandomValidSample()$
  $AddVertex(x_{rand}, G)$
  if *found valid solution shorter than* $path$ **then**
    update $path$;
if $path \neq \emptyset$ **then**
  return $path$;
else
  return failure;

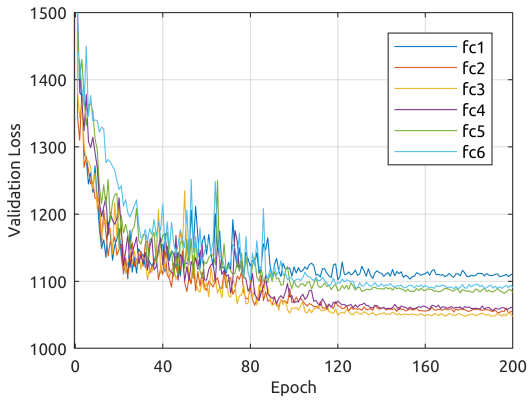

Fig. 5. Validation loss of RGN with different MLP layer numbers for the 3-DOF robot.

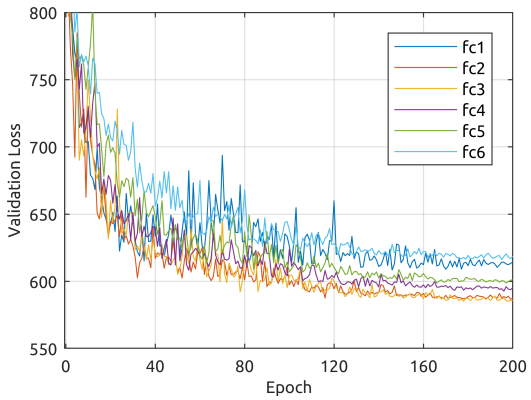

Fig. 6. Validation loss of RGN with different MLP layer numbers for UR3.

Fig. 7. The tested 3-DOF Robot Arm.

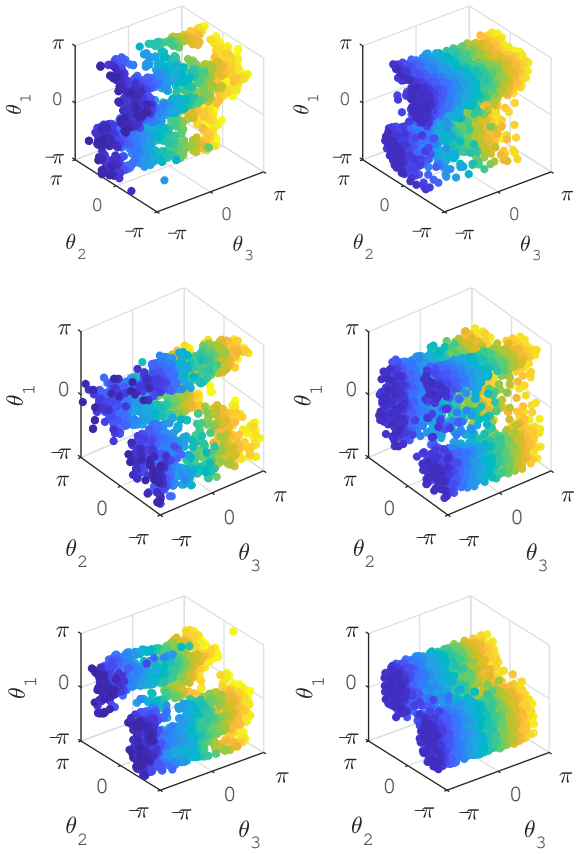

Fig. 8. Examples of C-free prediction for a 3-DOF robot in different environments, ground truth on the left.

V. EXPERIMENTAL EVALUATION

After algorithm implementation, experiments are performed on a 3-DOF robot arm and a 6-DOF UR3 robot arm.

Our network is trained with an NVIDIA GTX1080Ti GPU. The optimizer is Adam and the batch size is 16. The number of input points is 16384 and number of output vertices is 2048. In our dataset generation, t_{max} is set 3 minutes for each roadmap, $N_{neighbor}$ is 10 and $size_{min}$ is 2. Our dataset contains over 20000 environment-roadmap pairs for the 3-DOF robot and over 14000 for UR3. The ratio between training data and validation data is 3 : 1. The testing software platform is ROS kinetic with Moveit! framework on Ubuntu 16.04.

To find a better structure of the MLP module (Figure 4), validation loss of networks with different numbers of fully-connected layers are evaluated during the training process (Figure 5, Figure 6). All the networks achieve convergence after 200 epochs of training. For both robots, the final validation loss of the corresponding networks reaches the lowest with three fully-connected layers, which are the adopted network structures during the following tests.

The forward propagation time of the network is around 30ms. For performance enhancement, only a subset of the output vertices are used for roadmap generation; this number is 256. Loading and wiring these vertices takes less than 5ms. During this test, our method is compared to four typical sampling-based planners (PRM, LazyPRM, RRT* and RRT-connect) and a LazyPRM planner loading a randomly generated roadmap. All the planners above are probabilistic complete so that with enough time all the planners can find a solution if the problem is solvable. In our experiment, the planning time is set to 5 seconds and we only process data from problems that all planners can solve within the time limit. Each planner is measured by the planning speed and the cost of the found paths. For precise comparison, the planning time is measured by the time of $solution()$ function in OMPL framework to remove the time of ROS data transmission and preparation. For our method, the path length measured is the first found valid path. For RRT*,

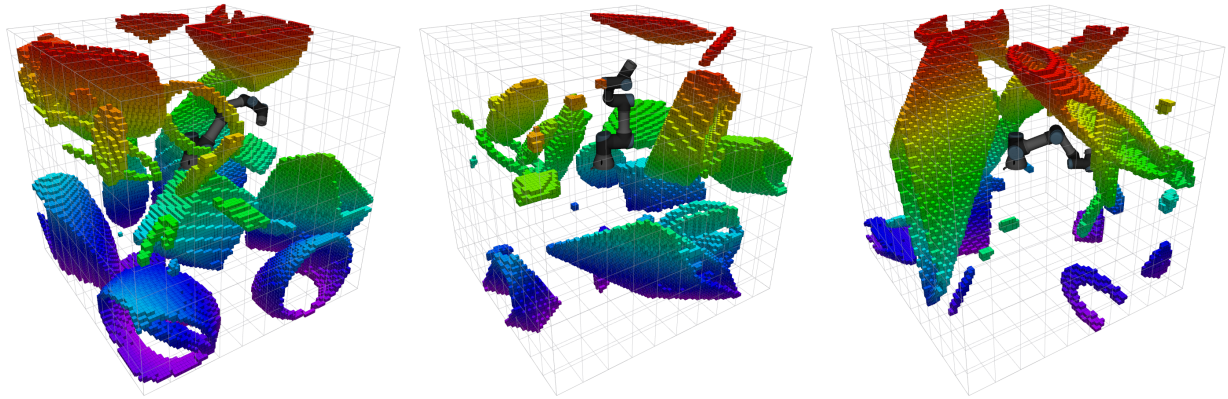

Fig. 9. UR3 robot being tested in random obstacle environments.

TABLE I

PERFORMANCE OF PLANNERS ON THE 3-DOF ROBOT

	Our Method	LazyPRM(random roadmap)	PRM	LazyPRM	RRT-connect	RRT*
Average planning time (normalized to average)	0.35	0.40	1.81	0.90	0.59	1.95
Average RSD of planning time (%)	27.75	29.50	59.15	67.49	52.35	62.33
Average path cost (normalized to lowest)	1.29	1.39	1.52	1.59	2.27	1.00
Average RSD of path cost (%)	3.31	4.62	15.08	19.28	29.09	1.22

TABLE II

PERFORMANCE OF PLANNERS ON 6-DOF UR3 ROBOT

	Our Method	LazyPRM(random roadmap)	PRM	LazyPRM	RRT-connect	RRT*
Average planning time (normalized to average)	0.34	0.52	1.54	0.84	0.71	2.05
Average RSD of planning time (%)	27.48	31.52	61.23	74.36	60.62	63.16
Average path cost (normalized to lowest)	1.27	1.55	1.42	1.70	2.07	1.00
Average RSD of path cost (%)	2.30	6.68	16.39	24.30	27.68	3.65

especially, the planning time is the time when the first path is found and the cost is from the final path as an indicator of the lowest cost, for the final result of RRT* should have been optimized for a long time. For multi-query planners, specifically PRM and LazyPRM, which can plan with pre-generated roadmaps as our method does, extra time (50ms, not included during comparison) is given for setting up a valid roadmap to make the comparison fair; the roadmaps keep the same for the same environment. All the planners above are from the Open Motion Planning Library (OMPL) [9] with default parameters. Environments and the start-goal configuration pairs are randomly generated in real-time (Figure 9). For each robot, results from over 1200 valid planning problems in over 400 environments are collected.

A. Experiment on a 3-DOF Robot Arm

The algorithm is tested on a designed 3-DOF robot arm (Figure 7) firstly as a preliminary experiment. Its C-space is of low dimensions and is easy visualized in a 3D space in the form of 3-D point clouds (Figure 8). The visualized results from the validation dataset show that our neural network can successfully predict the main structure of the free C-space. The blurry of the detailed structures indicates that there is still room for improvements. Then the vertices generated by the GPU are rewired and the planner starts planning.

The result of the 3-DOF robot experiment is shown in Table I. The performance of a planner in each problem is measured five times, then the average and the relative standard deviation (RSD) is calculated. After this, the average performance on the same problem of different planners is normalized to build an overall evaluation, planning time normalized to the average among all planners and path cost to the lowest. Compared to other planning algorithms, our method is of the fastest speed on average. The RSD of the planning time of the same problem shows the time stability of the planners, in which our method reached the lowest average value among all the planners to be of the most steady time performance. In the comparison of path cost, our method can plan relatively short paths, closer to the well-optimized paths generated by RRT* than any other algorithm compared. In conclusion, our method performs well in both planning speed and path quality with high stability in the 3-DOF arm test.

B. Experiment on UR3

UR3 is a typical 6-DOF robot arm. The experiment held over UR3 is to show the performance of our method in high-dimensional motion planning problems. Figure 6 shows the validation loss during network training. The network loss

decreases slower than in 3-DOF test, but can also converge after 200 epochs.

Comparison between our method and other planners in the UR3 test is shown in Table II. Our method showed a significant advantage over other compared planners in high-dimensional planning problems. The result is generally similar to the 3-DOF robot experiment: our algorithm can averagely find a relatively short path in the least time and have excellent stability on time cost and path length.

VI. CONCLUSIONS

This paper presents a novel approach to accelerate robot motion planning in variable environments. A neural network is designed to predict the free C-space according to the point cloud of the environment. LazyPRM is adopted as the search and check policy to ensure probabilistic completeness and asymptotic optimality. The roadmap can be used multiple times within the same environment and can be generated through GPU in a very short time, making this multi-query planner remarkably suitable for robot arms in mobile manipulator systems, where the arm often faces different environments and may need to perform multiple tasks in the same environment. In our test, compared to existing planning algorithms including PRM, LazyPRM, RRT* and RRT-connect from OMPL, our method is of the fastest speed, relatively low path cost and the most steady time performance in both low and high dimensional C-spaces.

Future work may include further optimization of the structure of the roadmap generation network. The dataset generation algorithm may also be optimized: more targeted dataset environments for specific tasks and more efficient dataset C-space roadmaps for solving motion planning problems.

REFERENCES

[1] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer, 2016.

[2] P. Leven and S. Hutchinson, "A framework for real-time path planning in changing environments," *The International Journal of Robotics Research*, vol. 21, no. 12, pp. 999–1030, 2002.

[3] T. Kunz, U. Reiser, M. Stilman, and A. Verl, "Real-time path planning for a robot arm in changing environments," in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Oct 2010, pp. 5906–5911.

[8] R. Bohlin and L. E. Kavraki, "Path planning using lazy prm," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, April 2000, pp. 521–528 vol.1.

[4] M. Pomârlan and I. A. Şucan, "Motion planning for manipulators in dynamically changing environments using real-time mapping of free workspace," in *2013 IEEE 14th International Symposium on Computational Intelligence and Informatics (CINTI)*, Nov 2013, pp. 483–487.

[5] 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)*, Oct 2018, pp. 6582–6588.

[6] 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.

[7] R. Q. Charles, H. Su, M. Kaichun, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 77–85.

[9] 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.

[10] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," Tech. Rep., 1998.

[11] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, April 2000, pp. 995–1001 vol.2.

[12] 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.

[13] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, Aug 1996.

[14] A. Short, Z. Pan, N. Larkin, and S. van Duin, "Recent progress on sampling based dynamic motion planning algorithms," in *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*. IEEE, 2016, pp. 1305–1311.

[15] L. Tapia, S. Thomas, B. Boyd, and N. M. Amato, "An unsupervised adaptive strategy for constructing probabilistic roadmaps," in *2009 IEEE International Conference on Robotics and Automation*, May 2009, pp. 4037–4044.

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

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

[18] Y. Rubner, C. Tomasi, and L. J. Guibas, "The earth mover's distance as a metric for image retrieval," *International Journal of Computer Vision*, vol. 40, no. 2, pp. 99–121, Nov 2000.

[19] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas, "Learning representations and generative models for 3d point clouds," *arXiv preprint arXiv:1707.02392*, 2017.