Path Planning of Robotic Arm using Neural Network

Rajath Koratagere Manjunath
Department of Electrical Engineering
New York University
Brooklyn, New York
rajath.km@nyu.edu
N19536684

***Abstract*—We present a modern motion planning method that emulates the behaviour of different Sample-based motion planning algorithms like Rapidly-exploring Random Tree(RRT), Optimal Randomly-exploring Random Tree(RRT*), Probabilistic Road Maps(PRM) which requires large computations. We discuss the issue of its performance in a known environment and extend the concept to unknown environments. We also address to the performance of the neural network based approach in terms of the time complexity and path cost.**

***Index Terms*—component, Motion Planning, Robotic arm, Robotics, ROS, Moveit.**

I. Introduction

Motion planning is defined as the sequence of states that the robot requires to transition from initial state to the final state. It is crucial for many robotic applications like self-driving cars, humanoid robots or medical robots. There are different types of motion planning algorithms. Graph-based algorithms represent the robot configuration as grids and tries to find the path from initial robot configuration to goal configuration. The complexity of the algorithm scales exponentially with the increase in dimensionality of the robot. The Reward-based motion planning algorithms tries to maximize the reward (minimize the loss), where the reward functions are defined to converge at the desired state. This algorithm requires numerous trials before learning the highest rewarding action at a particular state. Reward-based motion planning algorithms are popularly used in Markov Decision Process(MDP) mathematical framework which limits the number of actions the robot can choose. Hence the path is not smooth. Modeling of reward functions for complex problems may result is non convex reward function which might result in the robot not finding the path to desired state and getting stuck at a local minima. Artificial Potential fields are also used for motion planning. In this approach, the robots configurations are considered are points on potential field which is defined so that the decrease in gradient is in the direction of the desired states. As in the case of reward-based path planning, the robot configuration may get stuck in a local minima if the potential field is not convex and may not reach the desired states.

Sample-based motion planning algorithms are currently the state-of-the-art motion planning algorithms. They sample the configuration space and check for collision free path between the initial state and the goal state. These algorithms are said to be probabilistically complete as the probability of finding a collision free path between the initial and the goal states becomes 1 with sampling over time. Some of the popular motion planning algorithms are Probabilistic road maps(PRM), Rapidly-exploring Random Tree(RRT) and Optimal Rapidly-exploring Random trees(RRT*). Different variations of RRT* are popular to increase the efficiency of motion planning like Potentially guided-RRT* (P-RRT*) and Bidirectional RRT*. Even though the sample-based motion planning(SMP) algorithms are the state-of-the-art algorithms, they still require a lot of computation to generate path. The sample based motion planning algorithms also scale exponentially with the increase in dimensionality of the robot. Hence using these algorithms will increase the response time of the robotic applications.

To address to the above-mentioned issue, we propose using Deep Neural Networks(DNN) to emulate the behaviour of the SMP algorithms. We discuss the work on two approaches for motion planning problem using DNN.

1) *Training and testing on a single familiar environment:* In the first approach we use a deep neural network to learn the planning called the planning network(PNet). In this approach, the inputs to the neural network are the state at time step *t* and the goal state and the output to the neural network is the state at time step *t+1*. We iterate from initial state to the final state to generate a continuous path.
2) *Training and testing on different environments:* In this approach we use two neural networks, one to encode the information about the surroundings called encoder network(ENet) which takes as input the depth information of the obstacles in the surroundings and the other network for planning(PNet) which takes the encoded depth data, the states at time step *t* and the goal states and outputs the states at time step *t+1*.

The organization of the remaining of the report is as follows.

- Section II provides brief information about the relevant work in different motion planning techniques.
- Section III explains the problem definition.
- Section IV provides detailed information about the algorithm.

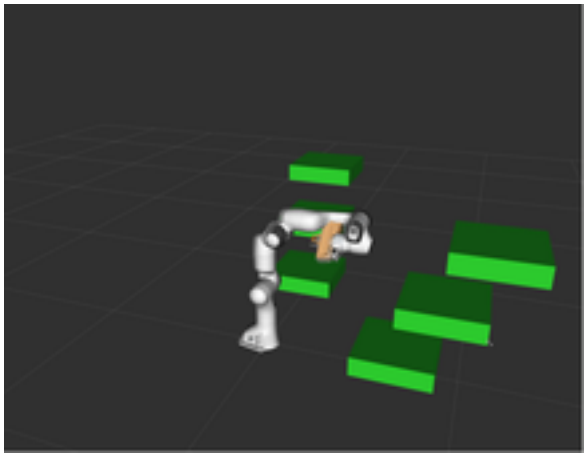

Fig. 1: Motion Planning of Panda Arm using MPNet:(Pos-1).

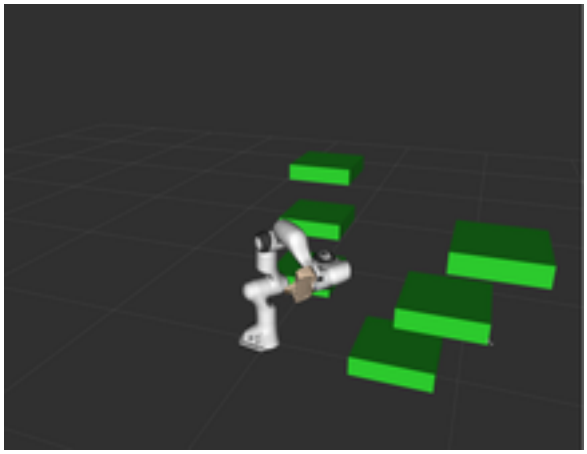

Fig. 2: Motion Planning of Panda Arm using MPNet:(Pos-2).

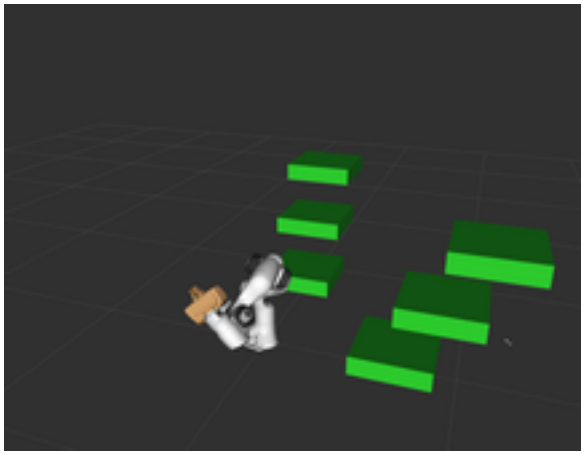

Fig. 3: Motion Planning of Panda Arm using MPNet:(Pos-3).

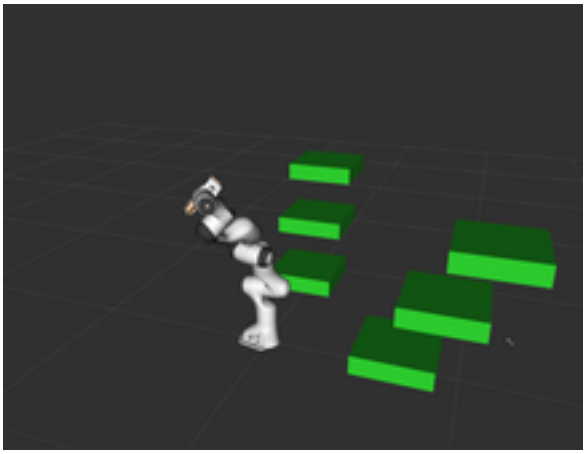

Fig. 4: Motion Planning of Panda Arm using MPNet:(Pos-4).

- Section V provides details about the implementation.
- Section VI shows the results of the simulations.
- Section VII provides conclusion of the project and talks about the future work in similar direction.

II. Related Work

The quest for solving the motion planning problem originated from development of complete algorithms that suffer computational inefficiency. This led to the development of potential based motion planning and sample-based motion planning algorithms. The development of sample-based algorithms are based on rapidly exploring trees and roadmaps. These algorithms are probabilistically complete as the probability of finding a valid path between start and the goal state approaches one as the samples in the graph approaches infinity. The popular SMP algorithms are RRT which is a single query method and PRM which is a multi-query method. Single query methods are preferred over multi-query methods as a multi-query problem can be divided into multiple single query problems. Also RRTs are minimalist designs of path planning as they try to minimize the number of paths computed. The optimal RRT(RRT*) is variant of RRT which asymptotically guarantees optimal path between start and the goal configurations. RRT* becomes computationally inefficient with increase in dimensionality. With increase in dimensions, the efficiency of RRT* is no better than grid search. To improve the efficiency further, variants of RRT* like Potentially guided RRT*(P-RRT*) are used. In this algorithm, the random exploring function incorporated artificial potential function which forces exploration in the direction of the goal. Many such heuristics are used to improve the efficiency of motion planning and to converge to the optimal path faster. In similar direction, many algorithms like informed RRT* and BIT* are also used. A similar feedback motion planning approach is using LQR trees which combines locally valid LQR into a non linear feedback to cover a reachable state space and minimize path cost between initial configuration and goal configuration.

The main goal of motion planning algorithms are to provide a valid path between start state and the goal state while minimizing the path cost. Some of the major developments in the field of path planning is in the Reinforcement learning(RL). RL considers a Markov Decision Process(MDP) mathematical framework which has finite defined actions. So the obtained

path is not smooth. MDP also considers that the system has complete knowledge of the surroundings, hence it is not very well equipped to handle new situations. RL interacts with the environment and observes the action state pair to calculate the next state. The action is chosen so as to maximize the reward. The reward function encapsulates the desired functions for the robot to follow. A lot of work is done on heuristics of cost functions. The reinforcement algorithms have proved to be promising in solving simpler and low dimensional tasks but do not perform very well in case of complicated tasks as designing cost functions for complicated tasks have proven to be difficult. The cost functions for complicated tasks may not always be convex. Hence there is a large probability of the robot configuration getting stuck in local minima and not converging to the goal. RL does not perform well in long duration tasks as the reward function decays with steps. There have been recent advancements in using RL in high dimensional problems by combining RL with functional approximators known as Neural network. Recent development in application of DNN in RL has led to development in Deep Reinforcement Learning(DRL). Application of DRL in model based and model free approaches has helped solve many robotics problems. Using Long short Term Memory(LSTM), a type of Recurrent Neural Network (RNN) which specializes in learning long and short term objectives is used to solve problems which have small rewards and long horizons.

III. Problem definition

This section first defines the notations used and explains the problem addressed in the report.

Consider the Configuration space of the robot $\mathbb{C}$, where $\mathbb{C}\epsilon\mathbb{R}^n$. The configuration space $\mathbb{C}$ can be divided into obstacle space $\mathbb{C}_{obs}$ and $\mathbb{C}_{free}$, where $\mathbb{C} = \mathbb{C}_{obs} + \mathbb{C}_{free}$.

The surrounding space of the robot is called the workspace $\mathbb{X}$. It is a subset of the configuration space, where $\mathbb{X}\epsilon\mathbb{R}^{\mathrm{m}}$ and $\mathbb{X} \subset \mathbb{C}$ and m denotes the dimensionality of the robot. Similar to the configuration space, the workspace $\mathbb{X}$ is also divided into obstacle workspace $\mathbb{X}_{obs}$, where the robot can reach but would result in collision and the obstacle free workspace $\mathbb{X}_{free}$, $\mathbb{X}_{free} = \mathbb{X}/\mathbb{X}_{obs}$.

Let $\sigma = [c_0, c_1, ..., c_m]$, where c_i is the i_{th} state of the robot. σ_t represents the states of the robot at time t, $t\epsilon[0, T]$. So σ_{init} denotes the states σ_{init} and σ_{fin} denotes the states σ_T. sigma.length() is the number of points between $[0, T]$. The motion planning problem is described as the feasible paths that connect σ_{init} to σ_{fin} and avoid the obstacles $\mathbb{C}_{obs}$. This problem can be concisely denoted as $\{\sigma_{init}, \sigma_{fin}, \mathbb{C}_{obs}\}$. Traditionally, $\mathbb{C}_{obs}$ is known and using the σ_t, the collision checker tests for collision with the surrounding obstacles $\mathbb{X}_{obs}$.

Another important problem in optimal motion planners is to minimize the path cost $J(.)$. An optimal motion planner provides weak or strong guarantee that given enough time, the motion planner can find the optimal path and minimize the path cost to the possible minimum path cost $J^*(.)$ for a given $\{\sigma_{init}, \sigma_{fin}, \mathbb{C}_{obs}\}$.

IV. MPNet: A Neural based Motion Planner

This section explains the proposed models for motion planning in a single familiar environment. It further explains the extension of the model for motion planning in different environments.

- *Planning in a single familiar environment:* To teach a neural network to predict robot configuration at time step $t + 1(\sigma_{t+1})$ given the robot configuration at time step $t(\sigma_t)$ and goal robot configuration as the input. If the neural network is trained to plan for a single environment, it can also learn the obstacle configuration space($\mathbb{C}_{obs}$). Hence the DNN model learns to solve the motion planning problem$\{\sigma_{init}, \sigma_{fin}, \mathbb{C}_{obs}\}$.
- *Planning in different environments:* The problem is similar to the previous case except the neural network cannot learn the obstacle configuration space($\mathbb{C}_{obs}$) when planning in different environments. For motion planning in different environments we define the motion planning problem as $\{\sigma_{init}, \sigma_{fin}, \mathbb{X}_{obs}\}$. We provide the obstacle workspace data as input to the planning network along with the initial robot configuration and the final robot configuration. We first encode the workspace data like point cloud data or depth data. It has been previously observed that Contractive Auto Encoder (CAE) works best for encoding workspace data. So we first encode the workspace data to latent space and feed this to the planning network. The motion planning comprises of two phases:
  1) offline training neural network model.
  2) online path generation.
- *Offline training neural network model:* The proposed method consists of two neural network models, an encoder network(ENet) and a planning network(PNet).
  1) *Encoder network:* The Encoder network is used to convert the obstacle workspace data to a latent space. A Contractive Auto Encoder is used to encode the sensor data. The CAE network can either be trained using an encoder-decoder architecture or can be trained combined with planning network. The loss function for training the Encoder network using encoder-decoder architecture is given below.
  $$L_{CAE}(\theta_e, \theta_d) = \frac{1}{N_{obs}} \sum_{x\epsilon\mathbb{D}_{obs}} ||x - \hat{x}||^2 + \lambda \sum_{ij} (\theta^e_{ij})^2 \quad (1)$$
  Where θ^e are parameters of the encoder, θ^d are the parameters of the decoder, λ is the penalizing coefficient, $\mathbb{D}_{obs}$ is the dataset of point cloud $x\epsilon\mathbb{X}_{obs}$ from $N_{obs}\epsilon\mathbb{N}$ different workspaces, $\hat{x}$ is the point cloud data reconstructed by the decoder.
  In case of using end-to-end learning to train the ENet and PNet together, the loss function is the MSE loss of the planned paths. This is explained further in the next section.
  2) *Planning Network:* We use feed-forward deep neural network to perform planning. Given the encoded ob-

stacle workspace data, the planning network learns the solution to the problem $\{\sigma_{init}, \sigma_{fin}, Z\}$, where $Z \epsilon \mathbb{R}^m$. where m is the dimensionality of the latent space of encoding. Hence given the input σ_t and σ_{des} along with the output of the E-Net (Z), the neural network predicts the configuration of the robot at time step $t+1$.

To train the network, we need to make use of an expert demonstrator. An expert demonstrator can be a human moving the robot or it can be a motion planning algorithm. In our case we make use of a motion planning algorithm to train the network. The objective is to find the intermediate robot configurations $\{\sigma_0, \sigma_1, ...\sigma_T\}$ which provide a continuous path to the desired robot configuration which we achieve by iterating the algorithm to find the next state, given the current and the desired state. The flow of the algorithm is shown in algorithms 1 and 2. The Neural network is trained to minimize the mean-squared-error(MSE) of the predicted robot configuration $\hat{\sigma_{t+1}}$ and the actual robot configuration in the expert demonstration σ_{t+1}. The loss function is formalized as:

$$L_{PNet}(\theta) = \frac{1}{N_p} \subset_{j \epsilon \mathbb{N}}^{\hat{N}} \sum_{i=0}^{T-1} ||\hat{\sigma_{i+1}} - \sigma_{i+1}||^2 \quad (2)$$

where, N_p is the number of path samples in the dataset, $\hat{N}$ is the set of path samples.

Algorithm 1 PNet$\{\sigma_{init}, \sigma_{fin}\}$

path $\leftarrow \emptyset$
$\sigma_{curr} \leftarrow \sigma_{init}$
$\sigma_{next} \leftarrow \Phi$
collision $\leftarrow$ **false**
while $\hat{\sigma_{next}} \neq \sigma_{fin}$ **do**
  $\sigma_{next} \leftarrow pnet(\sigma_{curr}, \sigma_{fin})$
  if $\sigma_{next} \epsilon \mathbb{C}_{obs}$ **then**
    collision $\leftarrow$ **true**
    break
  end if
  add σ_{next} →path
  $\sigma_{curr} \leftarrow \sigma_{next}$
end while
if collision **then**
  path $\leftarrow ExpertPlanner(\sigma_{init}, \sigma_{fin})$
end if.
return path

Algorithm 2 MPNet$(\sigma_{init}, \sigma_{fin}, \mathbb{X}_{obs})$

path $\leftarrow \emptyset$
$\sigma_{curr} \leftarrow \sigma_{init}$
$\sigma_{next} \leftarrow \Phi$
collision $\leftarrow$ **false**
while $\hat{\sigma_{next}} \neq \sigma_{fin}$ **do**
  Z $\leftarrow$ ENet$(\mathbb{X}_{obs})$
  $\sigma_{next} \leftarrow pnet(\sigma_{curr}, \sigma_{fin}, Z)$
  if $\sigma_{next} \epsilon \mathbb{C}_{obs}$ **then**
    collision $\leftarrow$ **true**
    break
  end if
  add σ_{next} →path
  $\sigma_{curr} \leftarrow \sigma_{next}$
end while
if collision **then**
  path $\leftarrow ExpertPlanner(\sigma_{init}, \sigma_{fin}, \mathbb{X}_{obs})$
end if
return path

V. Implementation details

This section gives the implementation details of Neural planning algorithms. The proposed models were implemented in PyTorch. For more details refer the project link. The Panda arm environment was implemented using MoveIt and ROS. The simulations have been run using gazebo. In case of planning in a single environment, the environment is created in RViz by creating collision objects. In case of planning in multiple environment, the environment is created in gazebo and the depth sensor data is interfaced with MoveIt.

A. Data collection

1) Contractive Auto Encoder(ENet) Training: For training of CAE network, about 50,000 random data samples are generated. The path of all the files are mentioned relative to the location `Motion-Planning-Network/src/panda_arm_motion_planning` The script for data generation can be found in the file under the name `point_cloud_datagen.py`. The script for training the Encoder network can be found under the name `enet_train.py`. The structure of ENet can be found at `utils/enet.py`.

2) Planning Network(PNet) training: For training the planning network, about 100,000 samples are generated using RRT* planner using MoveIt and the data is pre-processed to convert into a tensor of the form $\{\sigma_t, \sigma_{des}\}$ in case of planning in a single environment and in the form $\{\sigma_t, \sigma_{des}, \mathbb{X}_{obs}\}$. The structure of the planning network is given in the file `/utils/pnet.py`. The training file can be found under the `pnet_train.py`. We used Kinect depth sensor to collect the data using ROS.

B. Models Architecture

1) Encoder Network: For all the environments of panda arm we train using encoder-decoder architecture. we also train using end-to-end learning. We decoder architecture is a mirror structure of encoder. The encoder has four feed-forward layers and four parametric Rectified Linear (PReLu) layers. The input to the encoder layer is $\mathbb{N}*3$ where $\mathbb{N}$ is the number of obstacle data points and 3 is the dimension of the workspace.

TABLE I: Comparison between planning time of RRT* planner and MPNet

path index	time taken for MPNet	Time taken for RRT* planner
1	1.1363132	3.912369728088379
2	0.245835066	3.534370183944702
3	0.664265633	5.076876401901245
4	0.680688858	3.2926290035247803

2) Planning Network: The planning network is 14 layers DNN for the panda arm environment. We use Parametric Rectified Linear Layers(PReLu) for the non-linearity. We use Dropout to add stochasticity. Dropout also takes care of the batch normalization, so we don't add the batch normalization layers.

VI. Results

In this section, we compare the performance of MPNet with the RRT* planner which is considered one of the state-of-the-art motion planning methods. The MPNet was trained by fine tuning the learning rate between 10^{-3} and 10^{-6} and takes about 200 epochs to complete training. Considering the motion planning speed for multiple robot configurations, it is observed that the Neural planner performs 10 times faster than the expert demonstrator on an average. The MPNet finds path for 60% of the time. The current hybrid planning model using neural-based motion planning and sample-based motion planning performs better than simple sample-based motion planner in terms of time. The hybrid planner uses neural planner for generation of motion plans and keeps the expert demonstrator as the fails safe option. The presented algorithm is also probabilistically complete as is inherits the completeness of the expert demonstrator.

The presented algorithm performs better when performing motion planning on rectilinear joints. As the variation in joint states does not magnify with increase in distance from the end effector.

The figures 1, 2, 3 and 4 show the Motion planning of the panda arm using MPNet. Time taken by RRT* and MPNet for some paths are given in the table I. A term path cost is used to define the efficiency of the paths planned by the two planners. This is the sum of squares of distances between two consecutive configurations. The path cost for some paths is shown in figure 5.

VII. Conclusion and Future work

From the results of the experiments we can conclude that the Neural based motion planner can perform motion planning with considerable increase in speed and reasonable increase in cost compared to the expert demonstrator(RRT*). Given the initial and final configuration, MPNet is able to plan path efficiently. The algorithm presented is complete as an expert demonstrator is used for motion planning when the MPNet fails.

In the future, there is scope to extend the work on MPNet to build DNN with different architecture for motion planning.

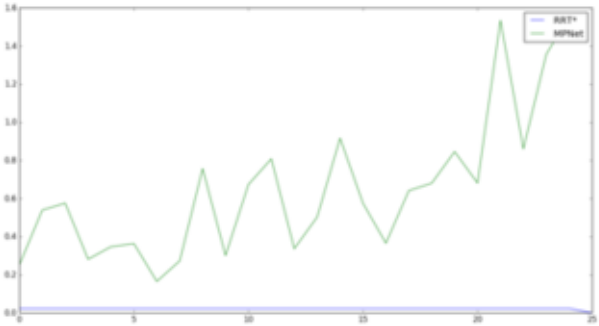

Fig. 5: Comparison of path cost

The work on MPNet opens scope for development of some of the following algorithms:

1) Using LSTM network for motion planning of self-driving cars. As the environment keeps changing, memory of the past states will help in better motion planning.
2) Changing the algorithm to explore bidirectionally rather than unidirectional exploration.
3) Building learning based actor-critic motion planning methods using collision checker.
4) Performing motion planning in operational space rather than joint space which will result in better planning precision for the end effector configuration.

References

[1] A. H. Qureshi, A. Simeonov, M. J. Bency and M. C. Yip, "Motion Planning Networks," 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 2019, pp. 2118-2124, doi: 10.1109/ICRA.2019.8793889.
[2] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.
[3] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.
[4] 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.
[5] J. Schmidhuber, "Deep learning in neural networks: An overview," Neural networks, vol. 61, pp. 85–117, 2015.
[6] 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.
[7] 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.
[8] T. Yan, Y. Zhang and B. Wang, "Path Planning for Mobile Robot's Continuous Action Space Based on Deep Reinforcement Learning," 2018 International Conference on Big Data and Artificial Intelligence (BDAI), Beijing, 2018, pp. 42-46, doi: 10.1109/BDAI.2018.8546675.
[9] B. Kiumarsi, K. G. Vamvoudakis, H. Modares and F. L. Lewis, "Optimal and Autonomous Control Using Reinforcement Learning: A Survey," in IEEE Transactions on Neural Networks and Learning Systems, vol. 29, no. 6, pp. 2042-2062, June 2018, doi: 10.1109/TNNLS.2017.2773458.
[10] J. Xin, H. Zhao, D. Liu and M. Li, "Application of deep reinforcement learning in mobile robot path planning," 2017 Chinese Automation Congress (CAC), Jinan, 2017, pp. 7112-7116, doi: 10.1109/CAC.2017.8244061.