ATLANTIS PRESS
Advances in Engineering Research, volume 149
2nd International Conference on Mechanical, Electronic, Control and Automation Engineering (MECAE 2018)

Robot Motion Planning Under Uncertain Condition Using Deep Reinforcement Learning

Zhuang Chen[1, a], Lin Zhou[2, b] and Min Guo[2, c]

[1]School of Chongqing University of Technology, Chongqing 400054, China;

[2] School of Chongqing University of Posts and Telecommunications, Chongqing 400065, China.

[a] cqcz0852@163.com, [b] cqchowlam@163.com, [c] ls58742@163.com

Keywords: Motion planning, Reinforcement learning, Deep reinforcement learning, Uncertain condition.

Abstract. The motion planning of industrial robot plays an important role in today's production systems, such as Made in China 2025 and Industry 4.0. The motion planning under uncertain condition is an important research topic in autonomous robots. For promoting the ability of motion planning to adapt to the environment change, in this paper, we propose a deep reinforcement learning (DRL) method which combines reinforcement learning with deep learning for industrial robot motion planning. Our work shows that the DRL-agent is capable of learning how to control the robot to successfully reach robotic tasks without explicit prior Knowledge of kinematics. We conclude that DRL has great potential for industrial robots and production systems, especially in robot motion planning.

1. Introduction

One of ultimate goals in robotic industry is to create autonomous robots in the field of artificial intelligence (AI). Such robots equipped with actuators and sensors, can execute tasks with less human intervention. Motion planning [1] is an important research topic in autonomous robots. Most motion planning algorithms are based on the exact environmental model [2], which means these algorithms cannot deal with the motion planning under uncertain condition that mainly refers to unknown environment and uncertainty environment. The unknown environment is an environment that is very complex and difficult to model accurately. The uncertainty environment refers to the results of robot execution are biased due to control errors and environmental factors, Even the mission failed.

Reinforcement learning (RL) [3],which is a branch of machine learning (ML) [4],which is good at controlling autonomous agent that interacts with the environment. RL improves the behavior of A in the process of continuous interaction with the environment. For example, the environment of arcade game Pac-Man is a maze. In the Pac-Man, the actions including up, down, left, and right, the reward is to eat the bean (+) and meet the monster (-). The goal of Pac-Man is to get out of the maze and maximize the accumulated reward value. The advantages of applying RL to motion planning including without accurately model of environment, adaptability to new environment and self-learning ability. However, RL has many challenges, high-dimensional state, no suitable reward function, the reward value for action has sparse feature, training data is not easy to obtain, and so on. Deep learning (DL) has made breakthroughs in computer vision [5] and speech recognition [6] in recent years. Such as convolution neural network, multilayer perceptron, restricted Boltzmann machine and recursive neural network to solve the environment perception in different scenarios. DL provides the possibility to effectively solve the problem of high dimensional feature mapping in RL. Deep reinforcement learning (DRL) [7] is a combination of DL and RL. In DRL, the state of RL is expressed by deep neural network. The correlation between data samples generated by RL and no suitable neural network structure problems appear when DRL solves high-dimensional state.

This paper use DRL methods to solve the robot motion planning problem in uncertain conditions. The state is represented by the convolution neural network and obtaining a lot of training data through simulation environment, using Pool of experience to solve the training data correlation, analyzing the effects of different neural network structure and reward function on robot arm motion planning, taking

the number of attempts to make the right motion planning as one of the indicators of reward function to improve training speed, improving algorithm q-learning to train the network and using stochastic gradient descent algorithm to update network weights. The experimental environment is simulated by mujoco [8]. The system makes right motion planning for a given starting point to ending point. System input is the RGB image obtained by a fixed monocular camera in mujoco and output is the best action for a given state. The experiment confirmed and improved DRL's ability to solve the robot motion planning problem in uncertain conditions.

2. Background

RL technique is essentially a trial and error method which aims to replicate the human learning process. The goal of RL is to maximize the accumulated reward value $\mathrm{R_t}$ as computed by

$$\mathrm{R_t} = \sum_{\mathrm{i=t}}^{\mathrm{N}} \gamma^{i-t} r_i \tag{1}$$

Where γ is the discount, N is maximum number of tries each episode.

The theoretical basis of RL is Markov decision making process (MDP) that the next state associated with the current state and the current the action taken. Reinforcement learning algorithm include based model (states, actions, transition probability, and rewards) and based free-model (states and actions) that only knows part of the information, this article is based free-model. The RL updates the table through the value iteration and select action by looking up the table. Turning the update of the table into a function fitting problem, namely, the approximate state gives an approximate action, and finally, update θ makes $\mathrm{Q_t}$ approach Q^*. The table records the mapping from state to action, so, it is impossible to deal with such massive state issue such as the go and robot motion planning, the solution is to extract the main features $\hat{\mathrm{s}}$ of s, then we have the approximate Q-function $\mathrm{Q}(\hat{\mathrm{s}}, \mathrm{a})$. So, the form of data is critical to learning.

The problem of robot motion planning under uncertain conditions: given the unknown obstacles, the initial position and target position in Euclidean space W. With free collision, generate a sequence of continuous path from start point to ending point. This paper does not directly give a sequence of motion planning from the beginning to the end but select the best action in the current state $\mathrm{s_t}$ until the whole motion planning finished. Due to the RL doesn't need precise modeling experiment environment, so the experiment is not set obstacles. Because RL without accurately model of environment, the experiment is not set obstacles. The Euclidean space W that the robot arm can reach, robot arm action set is $\mathrm{A} = \{\mathrm{a_1}, \dots, \mathrm{a_k}\}$, the current state $\mathrm{s_t}$ is the RGB image obtained by a fixed monocular camera in mujoco, selecting the best action $\mathrm{a_t}$ for a given state $\mathrm{s_t}$ then do action $\mathrm{a_t}$ in the mujoco, getting next state $\mathrm{s_{t+1}}$, then calculating the reward value $\mathrm{r_t}$ by reward function, repeat the process until the whole motion planning is completed. We can say that the process of finding the best motion planning is a reinforcement learning process.

The specific process is as follows:

1.Observe current state $\mathrm{s_t}$

2.Select the best action $\mathrm{a_t}$ for a given state $\mathrm{s_t}$ then execute $\mathrm{a_t}$

3.Observe next state $\mathrm{s_{t+1}}$

4.Calculate the reward value $\mathrm{r_t}$ by reward function

5.Update the Q-value as follows:

$$\mathrm{Q_{t+1}}(\mathrm{s_t}, \mathrm{a_t}) = \mathrm{Q_t}(s_t, a_t) + \alpha\left\{\mathrm{r_t} + \gamma \max_{a' \in A} Q_t(s', a') - \mathrm{Q_t}(s_t, a_t)\right\} \tag{2}$$

Where $\alpha \in [0,1]$ is the learning rate, $\gamma \in [0,1]$ called the discount rate that determines the present value of future rewards. $\mathrm{Q_t} \to \mathrm{Q}^*$ as $\mathrm{t} \to \infty$, namely:

$$\mathrm{Q}^*(\mathrm{s_t}, \mathrm{a_t}) = \mathrm{E}\left[\mathrm{r_t} + \gamma \max_{a' \in A} Q^*(s', a') | \mathrm{s_t}, \mathrm{a_t}\right] \tag{3}$$

The mainly function of DL is understand state $\mathrm{s_t}$, using the neural network replaces the lookup table to solve the problem of high state dimension, extracting the main features $\hat{s}_t$ of $\mathrm{s_t}$ then similar states can map to similar action.

3. Deep Re-enforcement Learning

In 2006, Hinton proposed a back-propagation (BP) algorithm based on the chain rule. In a broad sense, the network structure of deep learning is also a kind of multi-layer neural network. The BP algorithm makes multi-layer neural networks popular, deep learning has made breakthroughs in computer vision and speech recognition, in recent years. DL has a strong environmental understanding which means small number of states. In 2013, DeepMind [7] uses the deep neural network to automatically extract complex features to solve the problem of high dimensional mapping, combining DL with RL become Deep Reinforcement Learning (DRL) to learn control strategy directly from high-dimensional raw data, putting convolutional neural network(CNN) with Q-Learning then get Deep Q-Network (DQN) algorithm that input raw image s_t and output each action q value.

RL widely used in mobile robot navigation and the control of biomimetic underwater microrobot, DL helps indoor visual navigation [9] and other applications go farther [10]. Training with DL under the simulation environment can reduce high-dimensional state and training data is not easy to obtain, DL training requires that samples have labels, good independence and determination of target distribution. However, reward always with noise, delay and sparsity in RL, current state s_t correlation with next state s_{t+1}, the determination of target distribution is unstable and different scenarios have different state distribution. Therefore, we need to solve these problems, using reward $\mathrm{r_t}$ construct sample's label and putting sample (s_t, a_t, r_t, s_{t+1}) into experience pool D until training by randomly select part of data to solve the correlation between samples. Update the network through the loss function $\mathrm{L}(\theta)$ until the whole network convergence.

Q-Learning update by:

$$\mathrm{Q_{t+1}}(\mathrm{s_t}, \mathrm{a_t}) = \mathrm{Q_t}(s_t, a_t) + \alpha\left\{\mathrm{r_t} + \gamma \max_{a' \in A} Q_t(s', a') - \mathrm{Q_t}(s_t, a_t)\right\} \tag{4}$$

The loss function:

$$\mathrm{L}(\theta) = \mathrm{E}\left[\left(\mathrm{r_t} + \gamma \max_{\mathrm{a}' \in \mathrm{A}} \mathrm{Q_t}(s', \mathrm{a}'; \theta) - \mathrm{Q_t}(\mathrm{s_t}, \mathrm{a_t}; \theta)\right)^2\right] \tag{5}$$

Where θ is parameters of network, then using random gradient descent to update θ.

3.1 Model Architecture

Lin [11] used the neural network early to enhance the generalization ability of reinforcement learning. Today we continue with the idea of using convolutional neural network (CNN), CNN needs to constantly adjust the parameters to ensure network convergence.

The local feature size of the image determines the size of kernel per layer, the number of convolutional layers and the number of kernel per layer determine the complexity of the neural network model. This paper compares three different neural network structures, all of activation is Relu. C3F2 which means three convolution layers and two Fully connected layers. Same as Mnih, V., et al. [7], C3F2 inputting single 84*84 grayscale image then through three convolution layers and two full connection, finally output q-value for each action. The more details about C3F2 see table 1. C2F2 is less a convolution layer than C3F2, the more details about C2F2 see table 2. C1F2 is less a convolution layer than C2F2, the more details about C1F2 see table 3.

Table 1. Convolutional neural network structure C3F2

Layer	Input	Kernel Size	Stride	Num Filters	Output
C1	1*84*84	8*8	4	32	20*20*32
C2	20*20*32	4*4	2	64	9*9*64
C3	9*9*64	6*6	1	64	4*4*64
F1	4*4*64	N/A	N/A	512	512
F2	512	N/A	N/A	6	6

Table 2. Convolutional neural network structure C2F2

Layer	Input	Kernel Size	Stride	Num Filters	Output
C1	1*84*84	8*8	4	32	20*20*32
C2	20*20*32	4*4	2	64	9*9*64
F1	9*9*64	N/A	N/A	512	512
F2	512	N/A	N/A	6	6

Table 3. Convolutional neural network structure C1F2

Layer	Input	Kernel Size	Stride	Num Filters	Output
C1	1*84*84	8*8	4	32	20*20*32
F1	20*20*32	N/A	N/A	512	512
F2	512	N/A	N/A	6	6

3.2 Reward Function

Reward function [3] is a key part of RL in motion planning. There is no uniform reward function, such as, Mnih, V., et al. [7] for feedback good reward is 1 and bad is -1, if no feedback reward is 0. Zhang, F., et al. [12] have the reward function $f(d) = 10^{-3} * (\delta/d - 1)$, where, $\delta = 0.05m$ is threshold for distance. James, F., et al. [13] build the reward function $f(d) = e^{-\gamma * d}$, where, $\gamma = 0.99$, d is the Euclidean distance between the end of the robot arm and the target point. So, through reading and analyzing a lot of literature, this paper proposes the convergence speed become one of the indicators of reward function II that accelerates system learning.

In the experiment, N is maximum number of tries each episode, d_i is the Euclidean distance between the end of the robot arm and the target point after execute action a_i, Δd is variation of the distance d_i. Specific reward function detailed in table 4.

Table 4. Reward function and convergence speed

I irrelevant	II related
01: **If** $\Delta d > 0$ 02: **Set** $r_i = -1$ 03: **Else** $r_i = 1$	01: **If** $\Delta d > 0$ 02: **Set** $r_i = -1 + 1/(1 + N - i)$ 03: **Else** $r_i = 1 - 1/(1 + N - i)$

3.3 Algorithm

The refresh rate is 60hz, in the simulation environment. Caffe framework only with CPU, initial random probability $\varepsilon = 0.9$ (ε decrease with the increase of episode) to select random action a_t (or with probability $1 - \varepsilon$ to select action $a_t = \max_{a' \in A} Q_t(s', a'; \theta)$ when given state s_t, then do action a_t and obtain the RGB image as x_{t+1} by a fixed monocular camera in mujoco. If $t > N$ then restart new episode, where N is maximum number of tries each episode. The s' means that the maximum action a' corresponds to state, given a state s then calculate the q value corresponding to each action $a \in A$, and select the maximum one action a'. After image preprocessing $\phi(x_{t+1})$ we have s_{t+1} and calculate the reward value r_t by reward function. The image preprocessing includes turning the RGB image x_t into a grayscale image s_t with size 84*84.

Store sample (s_t, a_t, r_t, s_{t+1}) into experience pool D, and only the size of experience pool D more than 1000, then training by randomly select part of data, here we choose batch size is 32. Update the network by random gradient descent, then repeat. The general framework is shown in figure 1 and the detailed algorithm is shown in table 5. Line 11 to 14 is update the network in table 5, the value of γ is set to 0.95 in the figure 1.

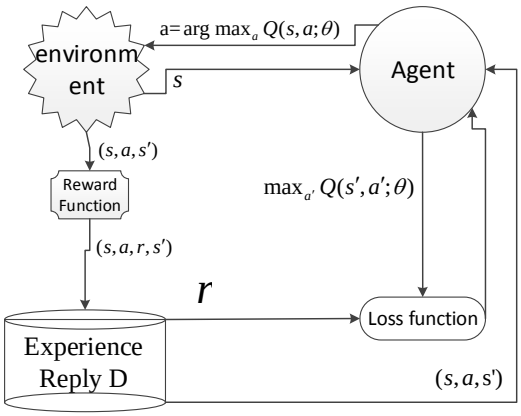

Figure 1. System model image diagram.

Table 5. Robot motion planning algorithm based on deep reinforcement learning.

Robot motion planning algorithm based on DRL
01: initial size of experience pool D is M
02: initial action-value function Q with random weights
03: **for** episode $= 1, \mathrm{M}$ **do**
04: initial x_1, d and image preprocessing $s_1 = \phi(x_1)$
05: **for** $\mathrm{t} = 1, \mathrm{T}$ **do**
06: with probability ε to select action $\mathrm{a_t}$, where, if $\Delta \mathrm{d} > 0$ then $\mathrm{a_t} \in \{A - \mathrm{a_{t-1}}\}$ else $\mathrm{a_t} = \mathrm{a_{t-1}}$
07: or with probability $1 - \varepsilon$ select action $\mathrm{a_t} = \max_{a' \in A} Q_t(s', a'; \theta)$
08: do action $\mathrm{a_t}$, obtain $x_{\mathrm{t+1}}$
09: get next state $s_{\mathrm{t+1}} = \phi(x_{\mathrm{t+1}})$, calculate the reward value $\mathrm{r_t}$ by reward function
10: store sample $(\mathrm{s_t}, a_\mathrm{t}, r_\mathrm{t}, \mathrm{s_{t+1}})$ into experience pool D
11: random small batch sample $(\mathrm{s_j}, a_\mathrm{j}, r_\mathrm{j}, \mathrm{s_{j+1}})$ from D
12: **if** $\mathrm{s_{j+1}}$ is terminal **set** $\mathrm{y_j} = \mathrm{r_j}$
13: **else set** $y_j = r_j + \gamma \max_{a' \in A} Q(s', a'; \theta)$
14: random gradient descent on $(y_j - Q(s_t, a_t; \theta))^2$
15: **end for**
16: **end for**

4. Experiments

Experiment with robot JR605 (more details see website http://www.hsr2013.com), each joint has two actions (five degrees clockwise or five degrees counterclockwise), three joints and each time has six actions to select. We use the ADADELTA [14] that one of the Stochastic Gradient Descent algorithm to update the network, experiment defines an epoch is 5000 iterations, namely $\mathrm{M} = 5000$, each iteration (update one times of the parameters of network) sets batch samples size is 32, each epoch means training network by $5000 * 32$ samples. The experiment verifies the effectiveness of deep reinforcement learning under uncertain conditions, a visual image of the motion planning of the robot arm's reaching task using deep reinforcement learning can be seen in figure 5.

Under Reward function II, experiment with three different convolutional neural network structures, from the figure 2 and figure 3 we can see that basically the same effect.

Under the convolutional neural network structure F3C2, experiment with two different reward functions, from the figure 4 we can see that reward with convergence speed is done well and more efficient.

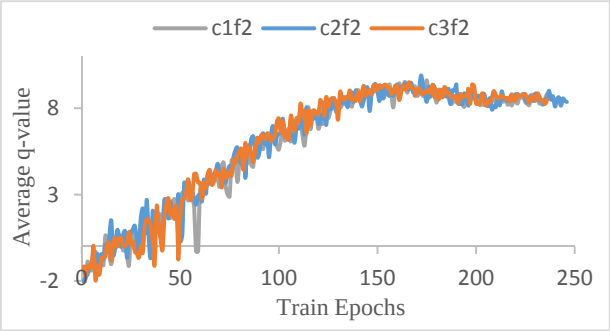

Figure 2. Average q-value with different convolutional neural network structures

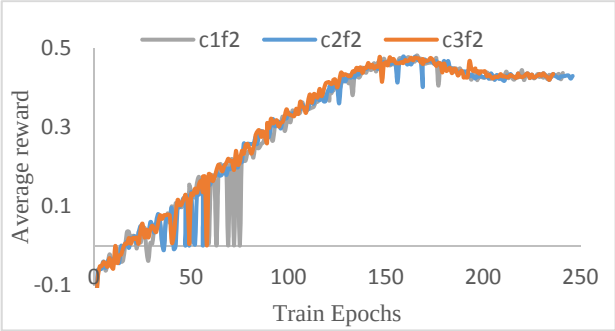

Figure 3. Average reward with different convolutional neural network structures

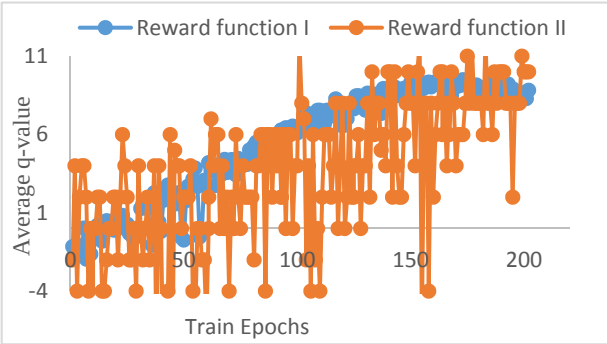

Figure 4. Average q-value with different reward functions

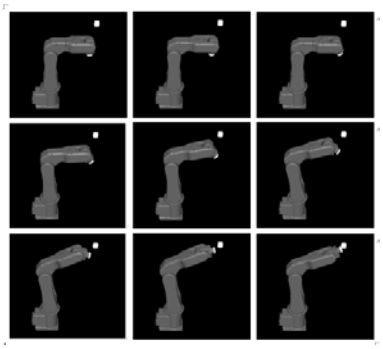

Figure 5. A visual image of the motion planning of the robot arm's reaching task using deep reinforcement learning

5. Summary

This article uses deep reinforcement learning for robot arm motion planning under uncertain conditions, which doesn't require precise environmental modeling, learning robot control strategy directly with high dimensional state input and achieving the robot arm motion planning from perception to action.

In the future, we will add some noise when training in mujoco to make the difference between the simulation and the real environment smaller, focus on prior knowledge to improve the train learning of the system, such as teaching based on the drag torque sensor.

References

[1]. Latombe, J.-C., Robot motion planning. Vol. 124. 2012: Springer Science & Business Media.

[2]. LaValle, S.M., Planning algorithms. 2006: Cambridge university press.

[3]. Sutton, R.S. and A.G. Barto, Reinforcement Learning I: Introduction. 1998.

[4]. Lecun, Y., Y. Bengio, and G. Hinton, Deep learning. Nature, 2015. 521(7553): p. 436-444.

[5]. Krizhevsky, A., I. Sutskever, and G.E. Hinton, ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, 2012. 25(2): p. 1097-1105.

[6]. Hinton, G., et al., Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. IEEE Signal Processing Magazine, 2012. 29(6): p. 82-97.

[7]. Mnih, V., et al., Playing Atari with Deep Reinforcement Learning. Computer Science, 2013.

[8]. Todorov, E., T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. in Ieee/rsj International Conference on Intelligent Robots and Systems. 2012.

[9]. Zhu, Y., et al., Target-driven Visual Navigation in Indoor Scenes using Deep Reinforcement Learning. 2016.

[10]. Tai, L. and M. Liu, Deep-learning in Mobile Robotics - from Perception to Control Systems: A Survey on Why and Why not. 2016.

[11]. Lin, L.J., Reinforcement learning for robots using neural networks. 1993.

[12]. Zhang, F., et al., Towards Vision-Based Deep Reinforcement Learning for Robotic Motion Control. Computer Science, 2015.

[13]. James, S. and E. Johns, 3D Simulation for Robot Arm Control with Deep Q-Learning. 2016.

[14]. Zeiler, M.D., ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.