Mapless Navigation for Autonomous Robots: A Deep Reinforcement Learning Approach

Pengpeng Zhang*, Changyun Wei*, Boliang Cai*
,Yongping Ouyang*
*College of Mechanical and Electrical Engineering, Hohai University
NO.200 Jinling Bei Road, Changzhou, Jiangsu Province, China
Email: pengpengzhang2019@outlook.com; changyun.wei@outlook.com;boliang_cai@outlook.com

***Abstract*—Finding a collision-free path for mobile robots is a challenging task, especially in sceneries where obstacle information is partly observed. Our work presents a decentralized collision avoidance approach based on an innovative application of deep reinforcement learning. The approach takes the spare 10-dimensional range findings and the target position in mobile robot coordinate frame as input and the continuous action commands as output. Traditional method for finding collision-free paths deeply depends on extremely precise laser sensor and the map making work of the roadblocks is inevitable. Our work shows that, using an asynchronous deep reinforcement learning method, a mapless path planer can be trained from start to finish without any manual operations. The trainer is available in other virtual environment directly. We compare a traditional method with the asynchronous method and find that our asynchronous method can decrease training time at beginning.**

***Index Terms*—Reinforcement Learning, Path Planning, Deep deterministic policy gradient methods, Mapless Navigation, Asynchronous Method**

I. Introduction

1) Deep Reinforcement Learning in mobile robots: Deep Reinforcement Learning method has been used to solve many tasks successfully, such as visual games [1] and simulations [2]. The utilization of reinforcement learning for robot navigation is mainly used in fully observed environment. For mobile robots, deep reinforcement learning methods usually select the actions from a discrete space to make the problem [3] [4] [5] easier. Continuous control is fundamental for widely used robots. Therefore, we pay attention to the navigation problem with continuous control, which is the reason of complicated computation.

2) Mapless Navigation: Action planning for navigation robots intends to reach the target position from the present position without colliding with others. For navigation robots, methods, such as simultaneous localization and mapping (SLAM), solve the problem [6] based on a map of the environment using dense laser range finding. Characteristics that are aimed to build the obstacle environment are mainly exacted manually. There are two defects in this task: (1) the time-consumption building and modernizing of the map, and (2) the extraordinarily dependance on the precise dense laser sensor for the mapping work. It is still a hard issue to select proper actions for mobile robot quickly without an environment map and using spare range findings.

3) Asynchronous Deep Reinforcement Learning: Compared with the traditional deep reinforcement learning framework, we separate the sample collection thread and the training thread as in [7], named asynchronous deep reinforcement learning. [7] demonstrates the asynchronous variant of the Normalized Advantage Function algorithm (NAF) can further reduce training time. Thus, we try to decrease the time consumption of training process utilizing the asynchronous method. The consumption of time of our asynchronous method is less than the original deep reinforcement learning methods at beginning . That means the asynchronous method is able to finish the task more quickly. It is mainly owning to the numerous sample threads.

We list our main contributions in this paper: (1) The path producer was trained from start to finish without any manual operations based on an asynchronous deep reinforcement learning. Path producer can obtain continuous angular acceleration to control angular velocity in a constant linear velocity. (2) A mapless path producer based on 10-dimensional distance information and target information was proposed. (3) We compare our asynchronous method to traditional reinforcement learning methods, finding that the time consumption of the asynchronous method is less than original methods.

II. Preliminary

A. Parametrization

This paper provides a mapless navigation method for autonomous robots. We try to receive a function:

$$\mathbf{A} = f(\mathbf{S}).$$

We use $\mathbf{S}$ to denote the set of states and $\mathbf{A}$ to denote the set of actions . We apply a neural network to predict the Q-value of the states and actions, called critic network, described as

$$Q = Q(s, a|\theta^Q),$$

where θ^Q is weights of the critic network, a is the action of agent and s is the state of agent at some point. Similarly, we apply a neural network, called action network, to produce actions and we use θ^μ to denote weights of action network, described as

$$a = \mu(s|\theta^\mu).$$

978-1-7281-4094-0/19/$31.00 ©2019 IEEE

In reinforcement learning problems, the reward function is of great significance and we use R to denote the set of rewards and r to denote the reward at some point.

B. MDP

As we all know, path planning problems can be modeled as a Markov decision processes (MDP). In MDP, an agent makes decisions every step to maximize the total reward. So we next explain MDP using notations we defined before as outlined in Algorithm 1. The agent considers actions based on current state end-to-end. The environment refreshes the state, the reward. When the agent reaches the target zones or collides an obstacle, an episode is ended. At first, we initialize the environment and other parameters. Next, the agent selects a path by making a sequence decisions to reach the target and maximize the total reward. In MDP, the agent knows the correspondence of state, action and reward. Thus, it can easily select a collision-free and time-efficient path because it will only select the biggest reward decision if the reward is set properly. But in fact, the agent does not know the reward function. It only receives a number denoting the reward after it selects an action. As a conclusion, only modelling the path planning problem as a MDP is not enough to solve path planning problem, but it is the foundation of a better algorithm.

Algorithm 1 MDP-Agent

Require: S
Ensure: A
1: Initialize environment
2: Initialize step,step = 0
3: **while** not end **do**:
4: Agent selects a_{step} based onS_{step}, R_{step}.
5: step = step + 1
6: Agent receives S_{step}, R_{step} and end
7: **end while**

C. Reinforcement Learning

Because MDP is not enough to find a collision-free path. So we next explain an algorithm which can solve this problem as shown in Algorithm.2. Similarly we first initialize the environment and other parameters, such as the weights of all neurons. We train the network for a certain number of step and the number is set before training. When the agent is learning, the agent continually selects an action,which is added a disturbance, from action space and receives rewards. Then, it will store the current state, the current action, the current reward and the next state. If the data was stored, the neural network will refresh its weights for every neuron to better forecast the function of current state,action, reward and the next state:

$$L = \left(\frac{1}{B}\right) * \sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2, \tag{1}$$

$$\nabla_{\theta\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,\ a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}. \tag{2}$$

When the network was trained well, we can only make use of the network but not train it later. At this time, we eventually know the value function and the agent can continually select the biggest increased direction to maximize the total reward using MDP model. But this algorithm needs a lot of time to train the network until the network is satisfied for the requirement. In fact, we usually drop DDPG algorithm because of too much time consumption. We need a new algorithm which is time-efficient.

Algorithm 2 DDPG

Require: S
Ensure: A
1: Randomly initialize critic network$Q(s, a|\theta^Q)$, and actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ
2: Initialize target network $Q(s, a|\theta^{Q'})$ and $\mu(s|\theta^{\mu'})$ with weights $\theta^{Q'}$,$\theta^{\mu'}$.$\theta^Q \rightarrow \theta^{Q'}$,$\theta^\mu \rightarrow \theta^{\mu'}$
3: Initialize replay buffer M
4: **for** $episode = 1$ to HE **do**:
5: Initialize a random process N for action exploration
6: Receive initial observation state s_1
7: **for** $step = 1$ to HS **do**:
8: Select action $a_{step} = \mu(s|\theta^\mu) + N$ according to the current policy and exploration noise
9: Execute action a_{step} and observe reward r_{step} and new state s_{step+1}
10: Store transition (s_{step},a_{step},r_{step},s_{step+1}) in M
11: Sample a random minibatch of B transitions ($s_i a_i r_i s_i$) from M
12: Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s|\theta^{\mu'}|\theta^{Q'}))$
13: Update critic by minimizing the loss L as in Eq.1
14: Update the actor policy using the sampled policy gradient as in Eq.2
15: Update the target networks:

$$\theta^{Q'} = \theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} = \theta^\mu + (1-\tau)\theta^{\mu'}$$

16: **end for**
17: **end for**

D. Asynchronous Reinforcement Learning

Since the time consumption for the reinforcement learning algorithm in practice is too much ,we propose a new algorithm based on asynchronous reinforcement learning algorithm to decrease the time consumption. So we next explain our algorithm which can solve this problem faster as shown in Algorithm 1(a). Similarly we first initialize the environment and other parameters, such as the weights of all neurons as same as DDPG algorithm. We train the network for a certain number of step and the number is set before training. When the agent is learning, the agent continually selects not only a action in ADDPG algorithm from action space and receives

Algorithm 3 ADDPG

Require: S

Ensure: A

1: Randomly initialize critic network $Q(s,a|\theta^Q)$, and actor network $\mu(s|\theta^\mu)$ with weights θ^Q and θ^μ

2: Initialize target network $Q(s,a|\theta^{Q'})$ and $\mu(s|\theta^{\mu'})$ with weights $\theta^{Q'}$ $\theta^{\mu'}$ $.\theta^Q \rightarrow \theta^{Q'}$, $\theta^\mu \rightarrow \theta^{\mu'}$

3: Initialize replay buffer M

4: **for** $episode = 1$ to HE **do**:

5: Initialize a random process N for action exploration

6: Receive initial observation state s^1, s^2, ..., s^n

7: **for** $step = 1$ to HS **do**:

8: Select action $a^1_{step} = \mu(s|\theta^\mu) + N_1$,$a^2_{step} = \mu(s|\theta^\mu) + N_2$, $a^n_{step} = \mu(s|\theta^\mu) + N_n$ for every agent according to s^1_{step}, s^2_{step} , ..., s^n_{step}

9: Execute action a^1_{step}, a^2_{step} , ..., a^n_{step}, and observe reward $r^1_{step}s^1_{step+1}$, r^2_{step} and new states^2_{step+1}, ..., r^n_{step},s^n_{step+1}

10: Store transition$(s^1_{step}a^1_{step}r^1_{step}s^1_{step+1})$, ..., $(s^n_{step}a^n_{step}r^n_{step}s^n_{step+1})$ in M

11: Sample a random minibatch of B transitions$(s_i a_i r_i s_i)$ from M

12: Set $y_i = r_i + \gamma Q'(s_{i+1}, \mu'(s|\theta^{\mu'}|\theta^{Q'}))$

13: Update critic by minimizing the loss L:

$$L = \left(\frac{1}{B}\right) * \sum_i \left(y_i - Q(s_i, a_i|\theta^Q)\right)^2$$

14: Update the actor policy using the sampled policy gradient:

$$\nabla_{\theta_\mu} J \approx \frac{1}{N} \sum_i \nabla_a Q(s,a|\theta^Q)|_{s=s_i, a=\mu(s_i)} \nabla_{\theta^\mu} \mu(s|\theta^\mu)|_{s_i}$$

15: Update the target networks:

$$\theta^{Q'} = \theta^Q + (1-\tau)\theta^{Q'}$$

$$\theta^{\mu'} = \theta^\mu + (1-\tau)\theta^{\mu'}$$

16: **end for**

17: **end for**

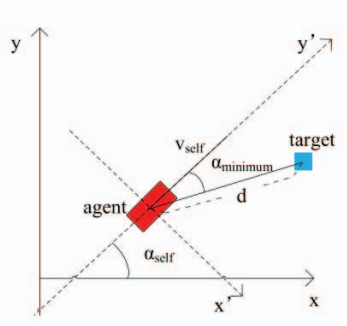
(a)

not only one reward every step and it will store n pieces of data, n is the number of agent which is learning at the same episode. In DDPG algorithm there is only one agent learning, but in our algorithm there are n agents learning together. Thus the algorithm takes less time than traditional algorithm. But the result isn't better as n is increasing. When n is too big, the application of ADDPG is too busy to run for a simple computer. It could be worse because the network may be outfitting. So n needs to set properly not too big nor too little.

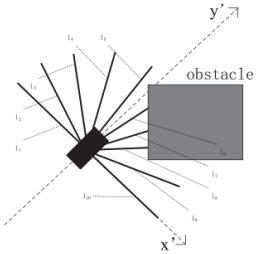
III. Path Planning Approach

A. State Explanation

There is some redundancy in the parametrization of the state S to receive a better consequence.There is a piece of 10-



(b) Target information



(c) Collision information

Fig. 1. State information explanation

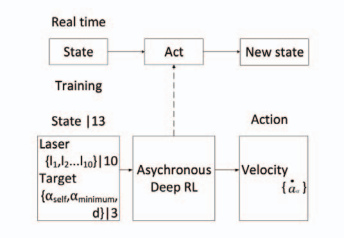


Fig. 2. Input vector

dimensional data, collision information, in state S for agent to avoid collision.That is, $\{l_1, l_2, l_3.....l_{10}\}$ where l_i is one piece of data which denotes the distance between the agent and obstacles. To reach a target zone, there is a piece of 4-dimensional data, target information, for agent in state S. That is, v_{self}, α_{self}, $\alpha_{minimum}$, d, where v_{self} is the linear velocity of the agent, α_{self} is the angel of the agent, $\alpha_{minimum}$ is the minimum angel for agent to reach the target, d is the distance between agent and the target, $\|p_t - p_a\|^2$, where p_t is the target position and p_a is the agent position, as shown in Fig.1. Because in our model the linear velocity is constant, to simplify our work we omit the linear velocity in target information. As a result, there is a piece of 3-dimensional data in target information. Furthermore our state S is composed of collision information and target information and it is 13-dimensional. As we all know, acceleration can be described by linear acceleration and the angular acceleration. $A = (a_l, a_\alpha)$ To parameterize action in our paper, we simple the problem by limiting the linger velocity at a constant value and we limit the angular acceleration under a constant value. The linger velocity is not necessary in our model, because it is constant. As a result, the action A is an 1-dimensional vector only including the angle velocity information a_α.

B. Network Structure

The problem can be converted into a reinforcement learning problem. We use our new reinforcement learning method to train the model as shown in Algorithm 1(a). As presented in Fig.2, the planner was trained in the virtual environment. The spare laser range finding are received from the raw laser finding from -90°to 90°in a average and fixed angle

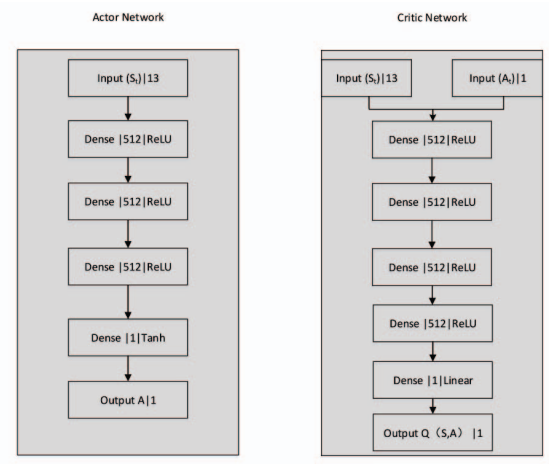


Fig. 3. Network structure

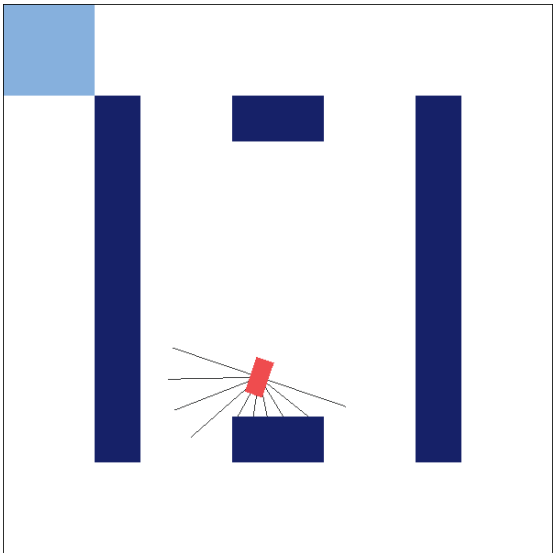


Fig. 4. Training environment

distribution. The input data is normalized to (0,1). As shown in Fig.3, the imported vector is transferred to the value of angular accelerate of the robot. To constrain the angular accelerate in (0,1), a hyperbolic tangent function(tanh) is used as the activation function. After 4 network layers with 512 nodes and 1 layer with 1 node, the input vector, the state S and the actor A , is transferred to the value of current state and actor .

C. Reward Function Definition

There are 3 kinds of reward.

$$R = \begin{cases} r_{collision} \\ r_{target} \\ r_{normal} \end{cases}$$

The first kind of reward, when the agent has collision with environment obstacle, the agent receives a negative $r_{collision}$ reward and this episode ends. The second kind of reward, when the agent arrives the target, the agent receives a positive reward r_{target}. Otherwise, the third of reward is the difference between current distance from the target and that of the last time step. Reward makes the agent get close to the target. In most time, the agent does not collide obstacles nor get the target, so the third kind of reward is used mostly.

IV. Experiences and Results

A. A path planner for autonomous robots in virtual environment based on reinforcement learning

We provide a path planner based on reinforcement learning. The path plan is trained for 4000 episodes without any manual operations. The planner learns how to reach a target position via selecting an action every step. The training environment is shown in Fig.4

We built a 600*600 indoor environment with wall. 4 disparate obstacles were setted in the environment, which are denoted by dark blue rectangles. The target position is denoted by sky-blue square. The target position changes randomly every episode. And the red rectangle represents our robot whose purpose is to reach the target position without collision with others. The robot finds a path to reach the target position in real-time. In order to simplify the path planning problem, the linear velocity is 50 constantly. In real world, control of angular velocity is continuous. Discrete control is difficult to transfer to real world. So our method selects actions in a continuous region. Trained mapless path planner reached the target without collisions, as shown in Fig.6. Our path planner is trained from start to finish without any manual operations. The success rate of our method in training every 50 episodes is shown in Fig.5. The success rate of our method in

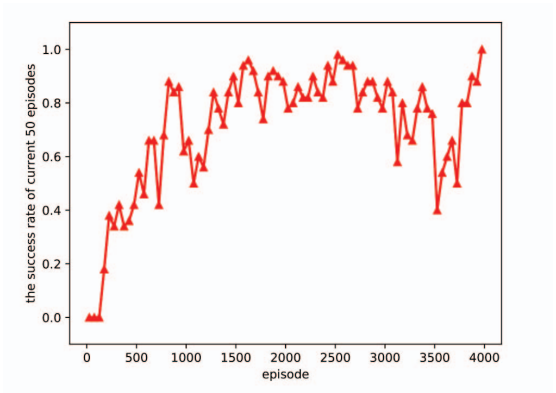
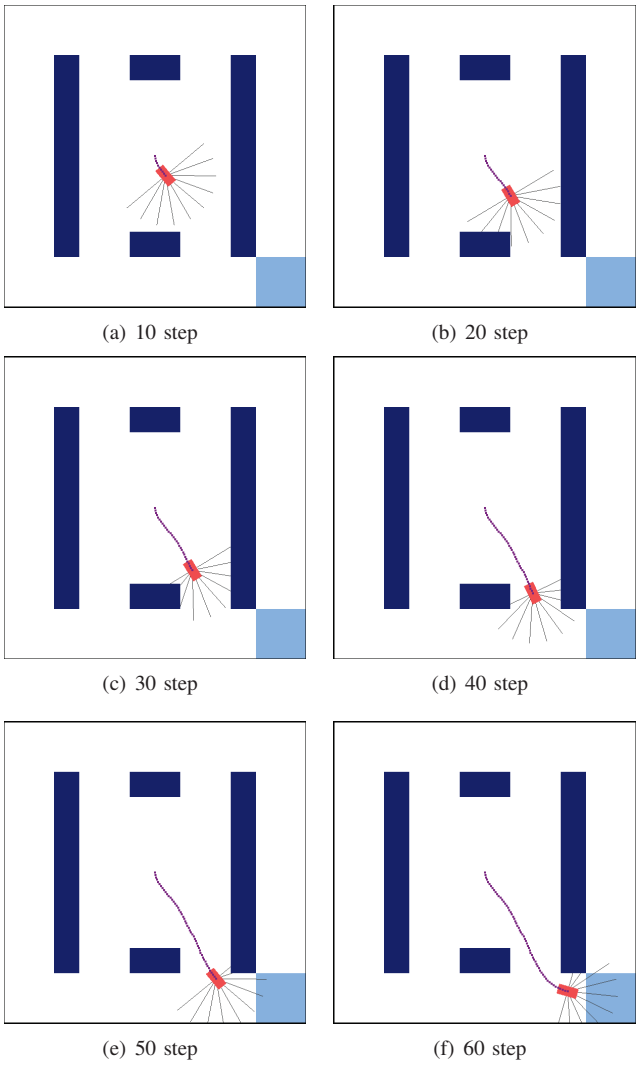


Fig. 5. Success rate in training.

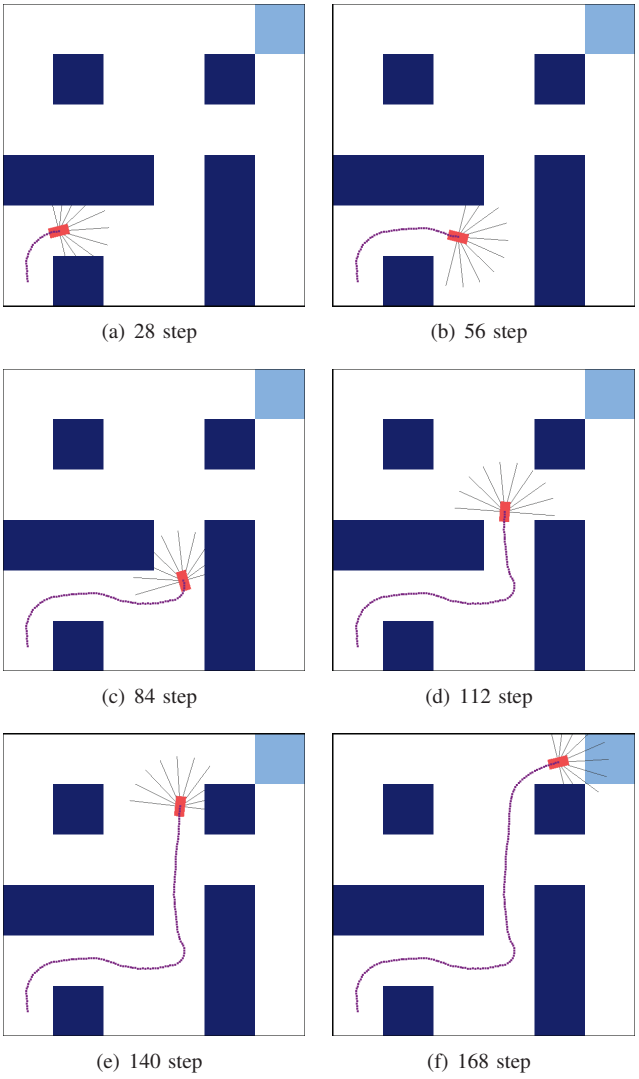
training is raising as training episode increasing. And after 1000 episode training, the success rate is over 70 percent mainly. And when it has been trained successfully, it can reach the target position nearly 100 percent. Our method can solve the path planning problem effectively. We compared our asynchronous reinforcement learning trained planner with the state-of-art Move Base motion planner. It uses the full laser range information for making decisions, while our mapless

producer only needs 10-dimensional information. And Move Base sometimes can not reach the target position when only using 10-dimensional finding [8].



(a) 10 step (b) 20 step (c) 30 step (d) 40 step (e) 50 step (f) 60 step

Fig. 6. Trajectory in virtual environment.



(a) 28 step (b) 56 step (c) 84 step (d) 112 step (e) 140 step (f) 168 step

Fig. 7. Trajectory in other virtual environment.

B. The planner can be utilized in other environment directly

We train the path planner not only for special environment but also other environments. After we finished the training process, we test it in another environment which we build in virtual, as shown in Fig.7. And the success rate of another environment is nearly 100 percent. In other word, our trained path planner is able to find a path without collision in real-time,which is basic for path planning in real world. From these figures, we can draw a conclusion, our trained path planner can be used in other environment directly.

C. Asynchronous methods decrease training time

Our asynchronous reinforcement learning method set the collection thread apart from the learner thread. The method make it straightforward to clearn from multiple robots in parallel. This is effective in real-world robot. When data collection is the limiting factor, quicker data collection may translate directly to faster learning. Asynchronous variant of the Normalized Advantage Function algorithm (NAF) can further reduce training time. Thus, we try to decrease the time consumption of training process utilizing the asynchronous methods. The consumption of time of our asynchronous method is less than the original deep reinforcement learning methods at beginning. That means the asynchronous method is able to finish the task more efficiently. This is mainly owning to the sample thread in parallel. As shown in Fig.8, the original deep reinforcement learning collects one sample every step while the asynchronous method collects five times more samples than the original deep reinforcement learning method.

In asynchronous training, the trainer thread trains the network four times per step, which each collector thread runs once per step. The main question to answer is: give these constrains, how much speedup can we gain from increasing the number

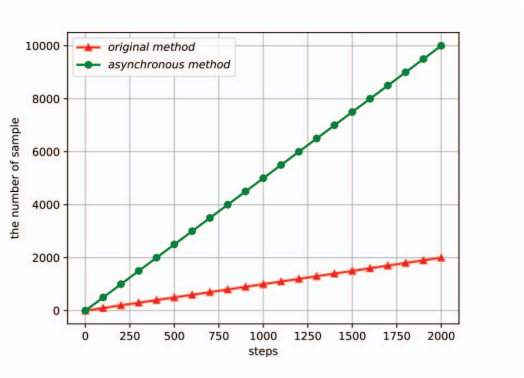


Fig. 8. The count of samples collected with the step increasing.

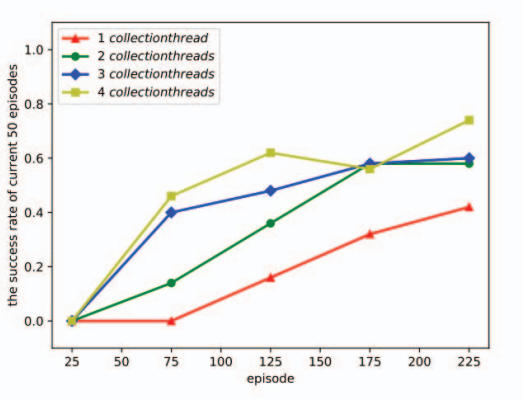


Fig. 9. Asynchronous training of reinforcement learning.

of robot. To analyse this, we set up the following experience in simulation. We locked each training thread to run at times the speed of the collector thread. Then, we varied the number of collector thread N. Thus, the overall data collection speed is approximately (1/S)*N times that of the trainer thread. We set S=4 for our training thread runs at approximately 4 times the speed of data collection. Fig.9 shows the results on path planning in virtual environment, as shown in Fig.4. The x-axis is the value of episode, which is increase as network is training. The results demonstrate three point: (1) at the beginning of training, increasing data collection makers the learning converge faster with respect to the number of training episode, (2) policy depend a lot on the ratio between collecting and training speeds, and (3) there is a limit where collect more data dose not help speed up learning. However at the beginning of training, more collection accelerates the speed of neural network training in these case. More collection thread allows our model to learn more data in the same time.

V. Conclusions and Future Work

In this paper, a mapless path producer was trained end-to-end through continual control. We improve the traditional reinforcement learning so that the training and sample collection can be existed in parallel, which decrease the training time compared with traditional method. By taking the 10-dimensional distance information and the target information as input, the trained path producer can not only be used in training environment but also other environments directly. When compared with low-dimension Move Base path planner, our approach prove to be more effective in complicated environment.

In the future, we are going to improve our method by changing the ratio between collecting and training speeds. Because the success rate of path planning is not stable as training episode increasing, we need to find a more steady method or make our asynchronous method better. As we all know, we hope that robots work in real word and robots help us finish path planning problems. So, virtual environment to real environment is considerable. Next, we will test our method in real world for path planning. We not only test it in real environment that is same as training environment but also other different environments. If it can be applied in real world directly, it can be a efficacious method to solve path planning problems. In virtual environment, our model was trained successfully. Then, we make use of our model in real world directly to find whether our model can solve the path planning problem. Besides, we also compare our asynchronous method with traditional methods in real world to select a better method. In a conclusion, we will test our asynchronous method in real world and verify whether our asynchronous method is better than traditional methods.

References

[1] Mnih, V., Kavukcuoglu, K., Silver, D.F., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G.: Human-level control through deep reinforcement learning. Nature **518**(7540) (2015) 529

[2] Tkachenko, Y.: Autonomous crm control via clv approximation with deep reinforcement learning in discrete and continuous action space. Computer Science (2015)

[3] Sadeghi, F., Levine, S., Sadeghi, F., Levine, S.: Cad2rl: Real single-image flight without a single real image. In: Robotics: Science & Systems. (2017)

[4] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Farhadi, A.: Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: IEEE International Conference on Robotics & Automation. (2017)

[5] Lei, T., Ming, L.: Towards cognitive exploration through deep reinforcement learning for mobile robots. (2016)

[6] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part i. IEEE Robotics & Automation Magazine **13**(2) (2006) 99–110

[7] Gu, S., , Lillicrap, T., Holly, E.: Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In: IEEE International Conference on Robotics & Automation. (2017)

[8] Lei, T., Paolo, G., Ming, L.: Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: IEEE/RSJ International Conference on Intelligent Robots & Systems. (2017)