Service Boy Robot Path Planner by Deep Q Network

Rawin Chaisittiporn

Department of Computer Science
Chandrakasem Rajabhat University
Bangokok, Thailand
rawin.ch@chandra.ac.th

บทคัดย่อ—บทความนี้ทำการศึกษาการใช้เครือข่ายคิวแบบลึกสำหรับการวางแผนเส้นทางของ ROS สำหรับการทำงานของหุ่นยนต์บริการ โดยการใช้ไลบรารี pymunk ของ Python สำหรับการฝึกฝนเครือข่ายนิวรอน เพื่อเรียนรู้การทำงานที่ดีที่สุดสำหรับสถานะต่างๆ ผู้วิจัยได้ออกแบบสถานะอินพุท การทำงาน เอาท์พุท จำนวนชั้นที่ซ่อนภายในของเครือข่ายนิวรอน หลังจากให้เครือข่ายนิวรอนเรียนรู้ ผู้วิจัยได้นำมาทำนายการเคลื่อนที่ของหุ่นยนต์ TurtleBot3 โดยใช้การทำงานของ ROS ได้แก่ slam_gmapping, amcl แต่ไม่ได้ใช้ costmap และ planner ผลการทดสอบพบว่าการใช้เครือข่ายคิวแบบลึกสามารถทำงานได้ดีกว่าการวางแผนเส้นทางของ ROS สามารถหลบหลีกสิ่งกีดขวางที่ซับซ้อนได้ดีกว่า โดยผู้วิจัยได้ปรับให้หุ่นยนต์ไปถึงเป้าหมายโดยไม่สนใจทิศทางการหันตัว และลดระยะทางระหว่างหุ่นยนต์กับเป้าหมาย ซึ่งหุ่นยนต์สามารถทำงานบริการได้เป็นอย่างดี

คำสำคัญ: การนำทาง, หุ่นยนต์บริการ

Abstract—This paper has studied how to use Deep Q Network (DQN) with ROS for path planner of service boy robot. We use pymunk library of Python for training the neuron network to learn the best action for various input states, by principle of reinforcement learning. Additionally, we use pygame library for learning simulation, observation and evaluation graphically. We have designed input states, output actions, and hidden layers of the neuron network. After training in predefined episodes we use the neuron network to predict the action of TurtleBot3 real robot. We use ROS for robot operation and use ROS slam_gmapping, amcl, except costmap and planners (both global and local planner). Instead, we use well trained Deep Q Network to predict the action of the robot. The result shows that well trained Deep Q Network has more efficient than original ROS planners. It can navigate to any place in the map and can reach the goal by obstacle avoidance. Absolutely, it can move to any destination in the map by orientation ignorance and reduce the distance accuracy between destination and the robot. Absolutely, it can suitably perform a role of service boy, like in the restaurant.

Keywords-DQN, Reinforcement Learning; ROS Navigation; Path planner; Service boy robot

***Abstract*—** This paper has studied how to use Deep Q Network (DQN) with ROS for path planner of service boy robot. We use pymunk library of Python for training the neuron network to learn the best action for various input states, by principle of reinforcement learning. Additionally, we use pygame library for learning simulation, observation and evaluation graphically. We have designed input states, output actions, and hidden layers of the neuron network. After training in predefined episodes we use the neuron network to predict the action of TurtleBot3 real robot. We use ROS for robot operation and use ROS slam_gmapping, amcl, except costmap and planners (both global and local planner). Instead, we use well trained Deep Q Network to predict the action of the robot. The result shows that well trained Deep Q Network has more efficient than original ROS planners. It can navigate to any place in the map and can reach the goal by obstacle avoidance. Absolutely, it can move to any destination in the map by orientation ignorance and reduce the distance accuracy between destination and the robot. Absolutely, it can suitably perform a role of service boy, like in the restaurant.

Keywords—DQN, Reinforcement Learning, ROS Navigation,

Path planner, Service boy robot

I. INTRODUCTION

Robot Operating System, ROS, is a notable software for robot development and has a big community of robot developers. ROS has many libraries and widely cover all of domains about robot. One of outstanding features of ROS is Navigation Stack that can make the robot navigate to any places in a map it has known. ROS Navigation Stack has many related libraries to work together, such as, slam_gmapping, amcl, move_base, cost map, global planner, base local planner.

Unfortunately, we have found that ROS global planner and local planner cannot well perform in complicated situation, like the narrow pathways, because of its algorithm limitation by ROS costmap inflation.

In this paper, we have interested in how to adopt the Deep Q Network with robot navigation on ROS. By reinforcement learning, robot can learn how to interact with any states and then take action. It learns by adapting its neuron network's weights to get the most rewards, which is Q value. The pymunk and pygame Python libraries have been used to simulate the learning. We can visually observe the behavior of the robot machine learning, testing, and evaluation.

We have designed and tested many factors of reinforcement learning, such as, states formats for neuron network input, hidden layers of neuron network, reward criteria, learning process, number of learning episodes.

After many experiments we have found the convergence of the neuron network, the virtual robot can avoid the obstacles and reach the goal altogether. Then we have taken this neuron network to predict the action of the TurtleBot3 real robot in the room with various obstacles patterns.

We have used some ROS Navigation libraries, they are slam_gmapping, odometry, amcl but we have not used costmap, move_base navigation. So we have replaced the global planner and local planner of ROS with our neuron network and have used it to predict the action of TurtleBot3. The TurtleBot3 robot can move to any goal in the map and can avoid the obstacles.

From our experiment of original ROS Navigation, trajectory planner and navfn library can not find the path when the environment has many narrow path obstacles then stuck and stop the robot movement (Figure 1). While the TurtleBot3 with DQN could avoid those obstacles (Figure 2).

Figure 1. The original Turtlebot3 is stuck by the obstacles

The algorithm has been improved for a service boy job that are 1) decrease the distance betweeen goal and the robot position to 0.30 m. 2) cancel the orientation checking. Summarily, we have found that Deep Q Network can replace the ROS global and local planner. It has more performance and accuracy than the original planners. The key methods are how to design input states and reward logic of reinforcement learning.

Figure 2. Turtlebot3 with DQN can move away the obstacles and head to goal.

II. RELATED WORKS

Deep Q Network is the state-of-art methodology of reinforcement learning developed by deepmind.com [9]. It can play Atari arcade game with frame captured of the

game RGB pixels. It can play seven Atari games with no adjustment of the architecture or learning algorithm. They have found that their reinforcement learning can perform surpassingly a human expert in three of them.

Furthermore, Deep Q Network can be adopted in the robot functions, such as, path finding [3] or robot gripper [4]. Mihai Duguleana and Gheorghe Mogan have created deep neuron network to predict the robot trajectory by simulation in MATLAB [5]. M.A. Moussa use neuron network and reinforcement learning with the robot grasping behavior. It can learn to grasp arbitrarily shaped objects [4].

Yu Fan Chen, Miao Liu, Michael Everett, and Jonathan P. How have developed decentralized multi-agent collision avoidance algorithm based on deep reinforcement learning [12]. The robot can predict the action based on its own velocity and others. So this algorithm cannot be applied in the real complex situation, because we cannot know all others velocity. Deep reinforcement learning can be used asynchronously for learning of robot navigation [8]. In that research they use with supervised auxiliary tasks.

Obviously, one technique of deep reinforcement learning that mostly used is learning from the old experience, like experience replay mechanism in [9] and transferring the relevant parts of the knowledge acquired as a result of previous experiences to improve the learning rate [1].

X. Zhuang has brought the concept of entropy to the reinforcement learning [11]. They have improved the learning performance with self-adaptive learning rate that based on the local strategy entropy.

There are various tools to make a simulation for DQN, such as, Gazebo [7], MATLAB [5], OpenAI Gym [10] [6]. Interestingly, pymunk is a Python library about Physics engine that can be used for Deep Q Network and reinforcement learning simulation. Especially, with graphical visualization from pygame. These tools are easy for coding in Python and can be simply adapted to the real robot. Matt Harvey have used pymunk and pygame to make a simulation of the virtual RC car to move around his virtual apartment [2]. His simulation is clearly efficient for Deep Q Network reinforcement learning but the robot just randomly move around the room to avoid obstacles. We have extended that work to make the robot move to predefined goal in a map, meanwhile avoiding obstacles.

III. EXPERIMENT

We have created Deep Q Network to learn the optimized path finding of the robot by reinforcement learning principle. The expected environment is the room with obstacles. Deep neuron network is the main policy function, it acquires input states and predict the action from its activation function. Reward and Q value will be calculated to adapt their weights to receive the most Q value, by stochastic gradient descent. Summarily, the robot try to receive the most Q value.

A. Learning stage

We have created the neuron network from Keras library on Theano. Physics engine Python library, pymunk, was used in the pygame to set the simulation environment for our reinforcement learning (Figure 3).

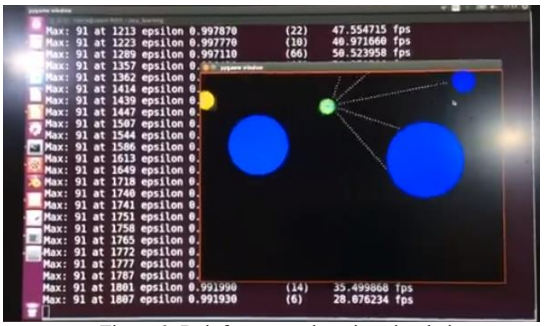

Figure 3. Reinforcement learning simulation.

We have determined experimental parameters for reinforcement learning to achieve the most Q value by:

1. Number of neuron nodes
2. Number of neuron layers
3. Number of learning episodes
4. States input design
5. Reward patterns

We had evaluation metrics to consider the convergence of learning that are:

1. Behavior of the robot
2. Number of steps moving to goal (time taken)
3. Number of obstacles hits

The process in this stage has been repetitively performed until the learning is convergent. That is the robot can avoid obstacles and move to the goal.

B. Robot testing stage

In this stage we need to test the real robot. We have taken the neuron network to predict the action of the TurtleBot3 Burger. We have used slam_gmapping library of ROS to make the map by TurtleBot3 Burger. Then we have applied ROS amcl to get location of the robot but we have not used ROS move_base library, it was replaced by the DQN instead.

For service boy concept, the robot just move near the goal in 0.30 m. range and neglect the robot orientation. Python ROS node would be created to send the cmd_vel topic to the TurtleBot3 robot to control its movement.

Then the movement of TurtleBot3 has been tested, observed, evaluated (Figure 4). Sensors data have been acquired from Lidar (Laser Scan) of TurtleBot3. The adjustment in learning could be done to fine tune the learning for robot movement optimization.

Figure 4. TurtleBot3 movement testing.

IV. RESULT

A. *Learning stage*

In learning stage we have found that all experimental parameters of reinforcement learning are inter-dependent. Only one parameter cannot determine the learning convergence. Table I. shows details of parameters that make the best results.

TABLE I. PARAMETERS TO MAKE LEARNING CONVERGENCE

Parameters	***Best value***	***remark***
Number of neuron nodes	128,128	
Number of neuron layers	2	Layer1 = 128, layer2 =128
Number of learning episodes	100,000	
States input design	5 sensors data, distance between robot and goal, angle of the robot heading and straight line to goal	We use movable goal in learning
Reward pattern	5000 if reach goal, $d_o + d_g + 10$ if action is going straight and angle between robot and goal < 0.5 radian, $d_o + d_g$ otherwise	
Learning technique	No obstacles if 1-50,000 episodes, with obstacles in 50,001 – 100,000 episodes.	

The states of reinforcement learning are:

1) 5 sensors data that are distances of around obstacles (Figure 5)

2) distance between robot and goal (Figure 6)

3) the angle of the robot heading and straight line to goal (Figure 7)

Described by:

$S_t = (\alpha_{-2t}, \alpha_{-1t}, \alpha_{0t}, \alpha_{1t}, \alpha_{2t}, d_t, \beta_t)$

where

S_t = states input
α_{-2t} = distance data of sensor at -60 degree at time t
α_{-1t} = distance data of sensor at -30 degree at time t
α_{0t} = distance data of sensor at 0 degree at time t
α_{1t} = distance data of sensor at 30 degree at time t
α_{2t} = distance data of sensor at 60 degree at time t
d_t = distance of robot and goal at time t
β_t = angle of the robot heading and straight line to goal at time t

The rewards of actions are calculated from these criteria:

1. 5000 if reach goal
2. If not reach goal

reward = $d_o + d_g + 10$; if action is going straight and angle between robot and goal < 0.5 radian

Or:

reward = $d_o + d_g$; otherwise

where
d_o = sum of distances of around five obstacles
d_g = distance from robot to goal

The actions of neuron network are:

$\pi = f(S_t)$
$= a$; $a \in$ { turn left, go straight, turn right }

In reinforcement learning, we use pygame to simulate the movement of robot. We can observe its behavior and readjust the learning parameter until the robot make optimized score.

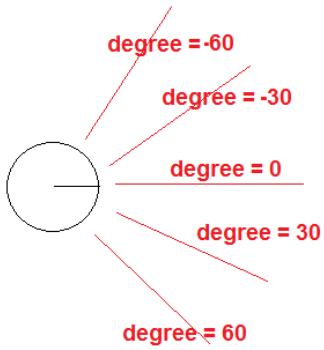

Figure 5. Five sensors data.

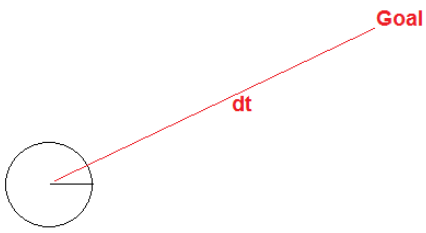

Figure 6. Distance between robot and goal.

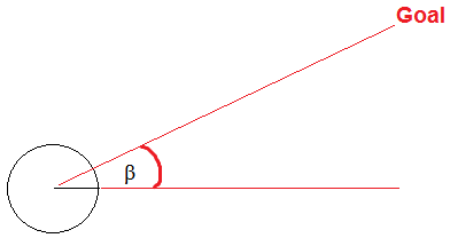

Figure 7. Angle of the robot heading and straight line to goal.

B. *Testing stage*

After learning stage we have tested the neuron network funcions by pygame and pymunk with fixed position of goal. The goal has been changed to the new position if the robot has reached the goal. The metrics of performance are number of frames (time) to reach goal and number of obstacles hits before goal reaching. Figure 8 shows result graph of the testing.

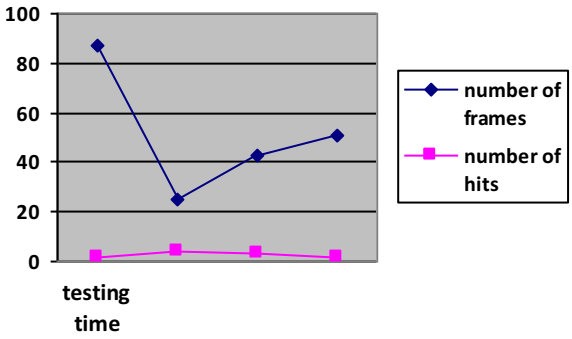

Figure 8. Result of Deep Q Network testing.

C. *TurtleBot3 testing stage*

For TurtleBot3 testing we have designed the variety of obstacles patterns that differ in position, size, and interval range between obstacles as described in figure 9. For TurtleBot3 we have modified the original algorithm for suitability of DQN. Figure 10 describes the flowchart of the new algorithm.

We use ROS rviz program to send the goal command to the TurtleBot3 and DQN nodes will trigger its activation functions. Table II shows results of the testing.

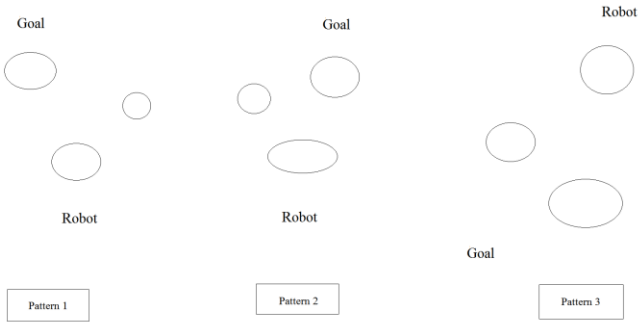

Figure 9. Various patterns for TurtleBot3 testing.

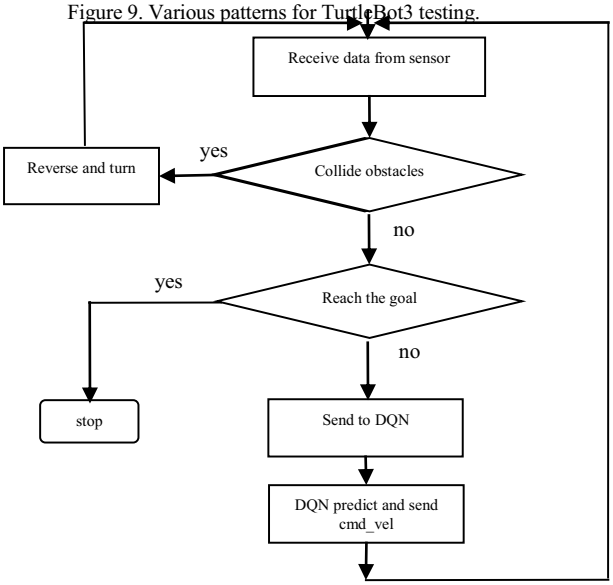

Figure 10. Flowchart of new algorithm for DQN function in TurtleBot3.

TABLE II. TURTLEBOT3 TESTING RESULTS

Obstacle pattern number	Average numbers of obstacles hit	Average time take to goal (sec.)
1	2	42
2	3	53
3	2	44

We have found that well learned Deep Q Network can absolutely be applied in robot navigation.

V. CONCLUSIONS

Robot navigation is the main composition of modern robot today. ROS Navigation Stack is now obviously impacting the robot field. But its path planners can perform only basic functions.

Deep Q Network can neatly replace the ROS path planner, by reinforcement learning simulation before adoption. In this paper has proposed how to use neuron network to learn the best actions of robot states by pymunk and pygame. Results show that by this methodology we

can take the Deep Q Network with reinforcement learning to make a noble robot navigation algorithm.

The important process is how to design the parameters that effect robot policy. We have found the useful tricks:

1) Learn with no obstacles in first half and with obstacles in second half.
2) Goal movement in learning stage can help convergence.
3) Design reward level for goal reaching, distance to goal, direction in straight direction and heading to goal, in descending order.

Finally, in adoption Deep Q Network for ROS robot, like TurtleBot3, can be done by using odometry, slam_gmapping, amcl and replace the planners with python node that implements well learned Deep Q Network predictions. The Deep Q Network can navigate the robot as it leant and performs action better than the orginal navfn and trajectory_planner.

REFERENCES

[1] B. Gökçe and H. L. Akın, "Implementation of Reinforcement Learning by transfering sub-goal policies in robot navigation," 2013 21st Signal Processing and Communications Applications Conference (SIU), Haspolat, 2013, pp. 1-4.

[2] H. Matt, "Using reinforcement learning in Python to teach a virtual car to avoid obstacles", Retrieved from http://blog.coast.ai/using-reinforcement-learning-in-python-to-teach-a-virtual-car-to-avoid-obstacles-6e782cc7d4c6, (2018, July 20).

[3] K. Nikaido and K. Kurashige, "Self-Generation of Reward by Sensor Input in Reinforcement Learning," 2013 Second International Conference on Robot, Vision and Signal Processing, Kitakyushu, 2013, pp. 270-273.

[4] M. A. Moussa, "Combining expert neural networks using reinforcement feedback for learning primitive grasping behavior," in IEEE Transactions on Neural Networks, vol. 15, no. 3, pp. 629-638, May 2004.

[5] M. Duguleana and G. Mogan, "Neural networks based reinforcement learning for mobile robots obstacle avoidance.," Expert Syst. Appl., vol. 62, pp. 104–115, 2016.

[6] R. Kaplan, C. Sauer, and A. Sosa, "Beating Atari with Natural Language Guided Reinforcement Learning.," CoRR, vol. abs/1704.05539, 2017.

[7] T. Lei and L. Ming, "A robot exploration strategy based on Q-learning network," 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR), Angkor Wat, 2016, pp. 57-62.

[8] T. Tongloy, S. Chuwongin, K. Jaksukam, C. Chousangsuntorn and S. Boonsang, "Asynchronous deep reinforcement learning for the mobile robot navigation with supervised auxiliary tasks," 2017 2nd International Conference on Robotics and Automation Engineering (ICRAE), Shanghai, 2017, pp. 68-72.

[9] V. Mnih et al., "Playing atari with deep reinforcement learning," arXiv preprint arXiv:1312.5602, 2013.

[10] W. Li, F. Huang, X. Li, G. Pan, and F. Wu, "State Distribution-aware Sampling for Deep Q-learning.," CoRR, vol. abs/1804.08619, 2018.

[11] X. Zhuang, "The Strategy Entropy of Reinforcement Learning for Mobile Robot Navigation in Complex Environments," Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain, 2005, pp. 1742-1747.

[12] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning.," in ICRA, 2017, pp. 285–292.