UTCC

Proceedings

การประชุมวิชาการ

และนำเสนอผลงานทางวิชาการระดับชาติ ครั้งที่ 5

The 5th UTCC National Conference

วันอังคารที่ 8 มิถุนายน 2564

การประชุมออนไลน์ผ่านโปรแกรม Cisco Webex

www.utcc.ac.th

Development of Obstacle Avoidance Autonomous Mobile Robot Using Deep Reinforcement Learning (DQN)   th ROS

Sirinart Hansuwan[1], Kiattisak Sritrakulchai[2]

Abstract

Currently, mobile robots using wheels, and automated operating systems are widely used in many industrial applications to support human operator. However, there is still lack of flexibility when the environment changes. For example, when a robot moves forward with changing obstacles or the destination position has changed. This causing the existing automation system has to be updated to comply with its usage, that requires a lot of cost and time. Accordingly, this research aims to solve the problem of mobile robot using wheels can avoid obstacles by deep reinforcement learning. We have developed the obstacle avoidance system by using deep reinforcement learning (Deep Q-Network), it is a type of reinforcement learning. ROS (Robot Operating System) is an important component of building robotic control systems. In addition, the learning system will be created in simulation environment model. In order to find the suitable parameter for decision making of robot control system, before applying to real robots.

Keywords: Mobile robot / Obstacle avoidance / DQN / Deep reinforcement learning

Background and problem statement

Today, artificial intelligence (AI) continues to be a subject of study to provide machines with learning abilities. It is important to understand the nature of learning in order to achieve the goal of intelligent machines. Although there is a great number of algorithms that were developed as supervised and unsupervised learning methods in the

[1] Master Degree Student, Department of Industrial Engineering, Faculty of Engineering, Mahidol University

[2] Lecturer, Department of Industrial Engineering, Faculty of Engineering, Mahidol University

ML field, the fundamental idea of reinforcement learning (RL) usage is that of learning from interaction, in order to obtain interaction ability. Reinforcement learning-based mobile robot navigation.

In the recent years, research and industrial interests are focused on developing smart machines such as robots that are able to work under certain conditions for a very long time and without any human intervention. This includes doing specific tasks in hazardous and hostile environments. The mobile robot should be able to move along the path from a given starting point to the target point in a complicated dynamic environment and conduct real-time local trajectory planning to avoid obstacles when encountering dynamic obstacles. But if the robot acquires local environmental information by relying on sensors with limited perception when there is no prior information in the dynamic environment, inaccuracy will be inevitably caused to the environmental map model established. Local trajectory planning depending on uncertain environmental model will certainly result in uncertainty of transmissibility.

This research aims to solve the obstacle avoidance problem using Deep Reinforcement Learning. In previous work, various mathematical models have been developed to plan collision-free paths for such robots. In contrast, our method enables the robot to learn by itself from its experiences, and then fit a mathematical model by updating the parameters of a neural network. The derived mathematical model is capable of choosing an action directly according to the input sensor data (from LiDAR) for the mobile robot. In this research, we develop an obstacle avoidance framework based on deep reinforcement learning. The essential components of robots are programmed under a ROS system. A 3D simulator (Gazebo) is designed to provide the training and testing environments, in order to solve the problem of control for the robot's intelligent decisions in a complicated dynamic environment.

Objective of research

To develop an autonomous mobile robot obstacle avoidance system using ROS and deep reinforcement learning.

Theory and algorithms of Reinforcement Learning

Reinforcement learning (RL) is an area of machine learning that is inspired by psychological and neuroscientific perspectives on animal behavior. It is the problem of getting an agent to act in an environment in such a way as to maximize its rewards, by predicting the long-term impact of its actions.

Much of reinforcement learning research stems from Markov decision processes, which are a low-level approach to representing stochastic environments. A Markov decision process consists of states, and transitions between those states. Actions that an agent is able to take are often represented as transitions. One point of note is that an action is not guaranteed to result in a particular state: instead, there are multiple possible states that may follow an action, characterized by a probability distribution. This accounts for partial observability and stochastic factors which may be impossible to predict accurately. We will not delve on a formal definition of Markov processes, as it is not used directly in this research. But it is important to note that they serve as a foundation for much of the theory behind machine learning.

Most reinforcement learning algorithms share a basic common structure:

1. Observe the environment.
2. Perform an action based on observation.
3. Observe a scalar reward/punishment signal.
4. Modify its behavior in light of the new knowledge.

Elements of Reinforcement Learning

Reinforcement learning can be understood using the concepts of agents, environments, states, actions and rewards, all of which are explained below in figure 1 that shown a Markov decision process. Capital letters tend to denote sets of things, and lower-case letters denote a specific instance of that thing; e.g. A is all possible actions, while a is a specific action contained in the set (Abdullah, 2018).

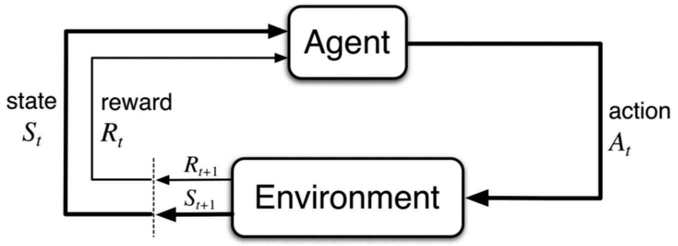


Figure 1 The agent environment interaction in a Markov decision process.
Note. From Reinforcement learning (p. 48), by Richard S. Sutton and Andrew G Barto, 2018

Agent: An agent takes actions; for example, a drone making a delivery, or Super Mario navigating a video game. The algorithm is the agent. It may be helpful to consider that in life, the agent is you.

Action (A): A is the set of all possible moves the agent can make. An action is almost self-explanatory, but it should be noted that agents usually choose from a list of discrete, possible actions.

Discount factor: The discount factor is multiplied by future rewards as discovered by the agent in order to dampen these rewards' effect on the agent's choice of action. Often expressed with the lower-case Greek letter gamma: γ. If γ is .8, and there's a reward of 10 points after 3 time steps, the present value of that reward is $0.8^3 \times 10$. A discount factor of 1 would make future rewards worth just as much as immediate rewards. We're fighting against delayed gratification here.

Environment: The world through which the agent moves, and which responds to the agent. The environment takes the agent's current state and action as input, and returns as output the agent's reward and its next state.

State (S): A state is a concrete and immediate situation in which the agent finds itself; i.e. a specific place and moment, an instantaneous configuration that puts the agent in relation to other significant things such as tools, obstacles, enemies or prizes. It can the current situation returned by the environment, or any future situation.

Reward (R): A reward is the feedback by which we measure the success or failure of an agent's actions in a given state. Rewards can be immediate or delayed. They effectively evaluate the agent's action.

Policy (π): The policy is the strategy that the agent employs to determine the next action based on the current state. It maps states to actions, the actions that promise the highest reward.

Value (V): The expected long-term return with discount, as opposed to the short-term reward R. Vπ(s) is defined as the expected long-term return of the current state under policy π. We discount rewards, or lower their estimated value, the further into the future they occur.

Q-value or action-value (Q): Q-value is similar to Value, except that it takes an extra parameter, the current action a. $Q_{\pi}(s, a)$ refers to the long-term return of an action taking action a under policy π from the current state s. Q maps state-action pairs to rewards. Note the difference between Q and policy.

The agent and environment interact at each of a sequence of discrete time steps,

t = 0, 1, 2, 3, ... t each time step t, the agent receives some representation of the environment's state,

$S_t \in$ S, and on that basis selects an action, $A_t \in$ A s). One-time step later, in part as a consequence of its action, the agent receives a numerical reward, R_t +1 ∈ R ⊂ R, and finds itself in a new state, S_t+1. The MDP and agent together thereby give rise to a sequence or trajectory that begins like this:

$$S_0 , A_0 , R_1 , S_1 , A_1 , R_2 , S_2 , A_2 , R_3 ,\dots$$

at time step t = 1, 2, 3,... ;

$$\dots, S_t , A_t , R_{t+1} , S_{t+1} , A_{t+1} , R_{t+2} , S_{t+2} , A_{t+2} , R_{t+3} ,\dots$$

Q-Learning

Q is the value of every possible state we can arrive at by performing an available action. If we know the value of the state succeeding each action, we can determine the optimal action to take by choosing the one that results in the highest value. Traditional Q-Learning is based on storing a table that maps state-action pairs to an expected value for that action. This table represents Q, and is updated iteratively using the following assignment rule that presented in figure 2, the same as equation of figure 3.

At terminal state; Q(*s*,*a*) = *r*

At non-terminal state;

$$Q(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

Figure 2 Q-Learning update rule

Note. From https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html

or

$$\underbrace{NewQ(s,a)}_{\text{New Q value for that state and that action}} = \underbrace{Q(s,a)}_{\text{Current Q value}} + \underset{\text{Learning Rate}}{\alpha}[\underbrace{R(s,a)}_{\text{Reward for taking that action at that state}} + \underset{\text{Discount rate}}{\gamma} \underbrace{max\, Q'(s',a')}_{\text{Maximum expected future reward given the new s' and all possible actions at that new state}} - Q(s,a)]$$

Figure 3 Equation of Q-Learning update rule

Note. From https://medium.com/mindboard/q-matrix-update-to-train-deep-recurrent-q-network-more-effectively-de616e7c72fa

The process of Q-Learning creates an exact matrix for the working agent which it can “refer to” to maximize its reward in the long run. Although this approach is not wrong in itself, this is only practical for very small environments and quickly loses its feasibility when the number of states and actions in the environment increases.

The solution for the above problem comes from the realization that the values in the matrix only have relative importance i.e. the values only have importance with respect to the other values. Thus, this thinking leads us to Deep Q-Learning which uses a deep neural network to approximate the values. This approximation of values does not hurt as long as the relative importance is preserved.

The basic working step for Deep Q-Learning is that the initial state is fed into the neural network and it returns the Q-value of all possible actions as on output (Gupta, 2019).

One modification to the algorithm is that a simplified update rule is used:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \gamma \max_a Q^{\pi}(s_{t+1}, a)$$

Robotic Operating System (ROS)

The Robot Operating System (ROS) is not an actual operating system, but a framework and set of tools that provide functionality of an operating system on a heterogeneous computer cluster. Its usefulness is not limited to robots, but the majority of tools provided are focused on working with peripheral hardware (Ademovic, n.d.)

For ROS's working, consists of a master section, a primary node that controls all other nodes in the system, such as registering to publisher node, subscriber node, as figure 4, which works in this way, can be said to be peer to peer, namely, each node communicates with each node, it can send message to each other using the TCP/IP protocol. The Master's function is not involved in communicating between nodes, but only as coordinators, namely the storage of communication-related information.

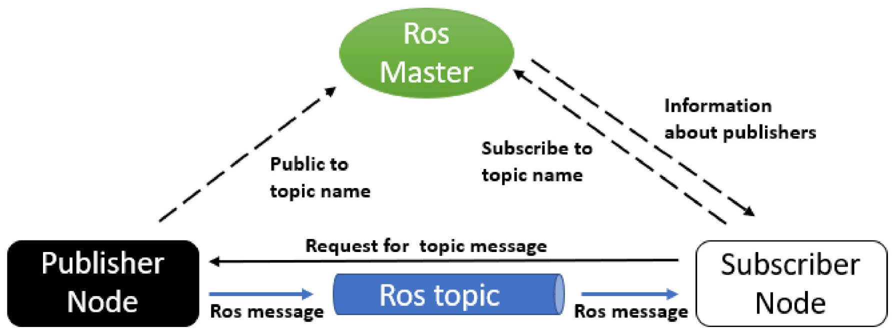


Figure 4 ROS communication structure

Note. From https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system(ros)/

Simulation: Gazebo

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. There are physics engine, high-quality graphics, and convenient programmatic and graphical interfaces (Why Gazebo?, n.d.).

TurtleBot3 supports development environment that can be programmed and developed with a virtual robot in the simulation. There are two development environments to do this, one is using fake node and 3D visualization tool RViz and the other is using the 3D robot simulator Gazebo. The fake node method is suitable for testing with the robot model and movement, SLAM and navigation, which can use sensors such as IMU, LDS, and camera in the simulation (TurtleBot3 simulation, n.d.).

Research methodology

From the evaluating of alternatives, the concept design consists of 4 functions. First, input data of robot position from LiDAR. Then, it sends the data which received through network, deep reinforcement learning for making a decision to the next position. This process worked on Robotic Operating System (ROS); the simulation of mobile robot has shown in Gazebo. The robot will also learn to avoid obstacles and reach the goal in Gazebo. Afterwards, we will get the fit parameter of robot learning for testing in real world. The conceptual design of process is shown in Figure 5.

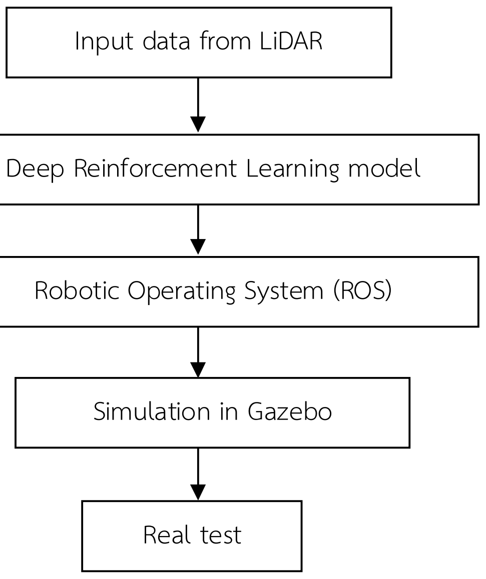

Input data from LiDAR
Deep Reinforcement Learning model
Robotic Operating System (ROS)
Simulation in Gazebo
Real test

Figure 5 Conceptual design

Setup and implementation

The equipment of mobile robot is chosen to attach with a robot, is shown in Figure 6 and the proposed trajectory planer is operating according to his flow chart presented in figure 7.

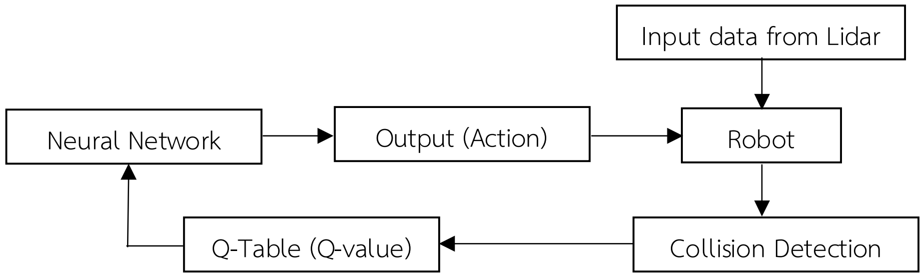


Figure 6 Trajectory planner structure

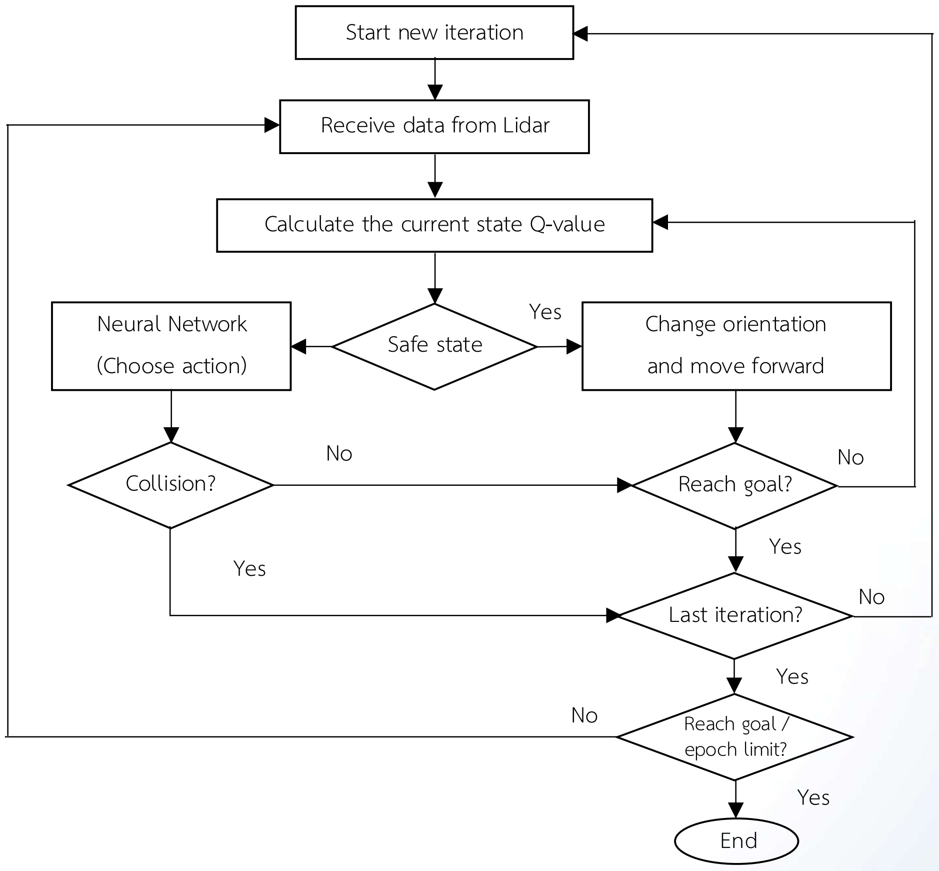


Figure 7 Trajectory planner algorithm flow chart

Mathematical models

Mathematical model of the obstacle avoidance mobile robot is design for using in making decision of the movement of robot to the desired target. Therefore, designing mathematical model for programming is separated into 2 parts, Deep Q-Network for robot's movement decision making and reward function when robot choose some action.

Deep Q-Network

This research is to train a DQN agent that learns an optimal policy to navigate the robot from point a to point b with minimum effort. Algorithm is that a simplified update rule is used:

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \gamma \max_{a} Q^{\pi}(s_{t+1}, a)$$

State

State is an observation of environment and describes the current situation. This is vital for the agent because it would calculate and act depending on the state. The state size is 26 and 24 LDS (Laser Distance Sensor) values. The other two are distances to goal, and angle to goal. A mathematical approach for this is as follow:

State = LDS (8 values) + Distance (1) + Angle (1)

LDS denotes the (24) values that the lidar sensor emits. Distance represents the distance to the goal and Angle is the angle between the robot heading and vector to the goal.

Action (Degrees of Freedom)

The robot has three actions which can act on depending on the type of state. In here, the robot has a fixed linear velocity of 0.15m/s and the angular velocity is determined by action, shown in figure 8.

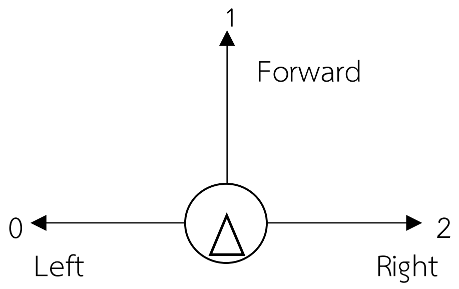


Figure 8 Three actions of robot

Table 1 Actions and angular velocity of robot

Action	Angular velocity (rad/s)
0	-1.5
1	0
2	1.5

Action: [0, 1, 2] with angular velocity of robot is shown in Table 1.

Reward Function

When turtlebot3 takes an action in a state, it receives a reward. The reward design is very important for learning. A reward can be positive or negative. When turtlebot3 gets to the goal, it gets big positive reward. When turtlebot3 collides with an obstacle, it gets big negative reward.

Before giving the reward function, one environment state is classified into four different properties, called the state property:

- Safe State (SS): a state where the robot has a low or no possibility of collision with surrounding obstacles.
- Non-Safe State (NS): a state where the robot has a high possibility of collision with some obstacles in the environment.

- Winning State (WS): one of the terminate states when the robot reaches its goal.

- Failure State (FS): one of the terminate states when the robot collides with obstacles.

Experiment Setup

Installation

Install Tensorflow, Keras and Anaconda with Ubuntu 16.04 and ROS kinetic, Anaconda 5.2 for Python 2.7 version. To use ROS and Anaconda together, we must additionally install ROS dependency packages. After that, install Keras, is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. And then, install Machine Learning packages.

Set State

sample = 360 degrees, modify it to 8 regions (G_n , n $\in [1,8]$) are shown in figure 9 and figure 10 show the distances between TurtleBot3 (robot) and obstacle d_{RO} and another one between the robot and goal d_{RG} determine the transition state of the robot.

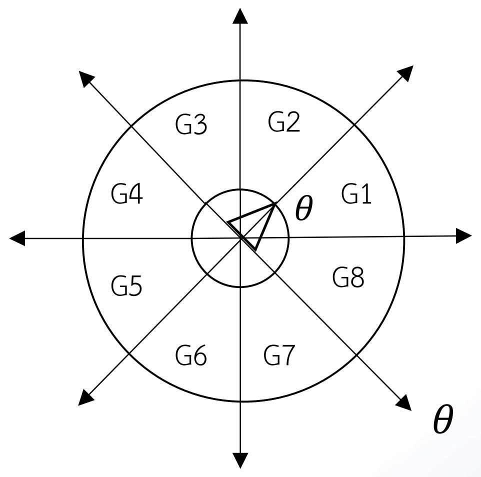


Figure 9 Eight regions for each angle

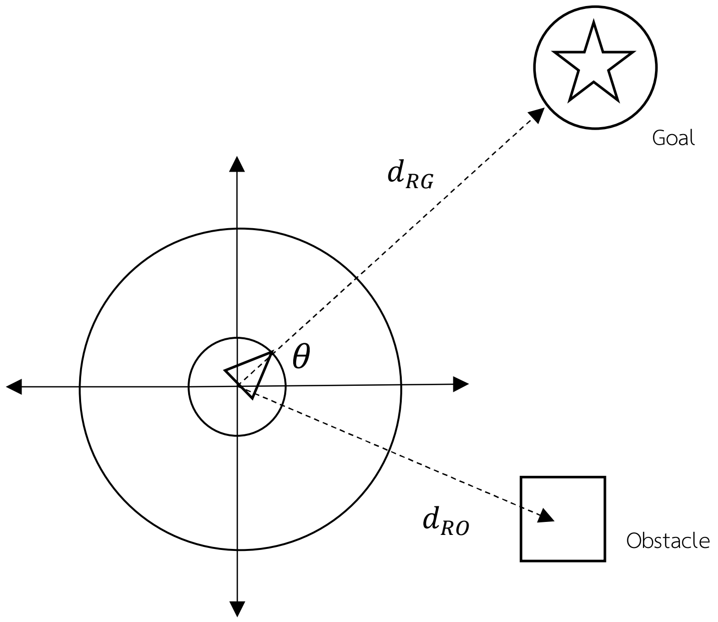


Figure 10 Main distance for determining TurtleBot3 state

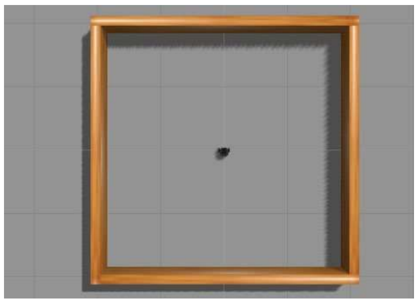
Set Reward

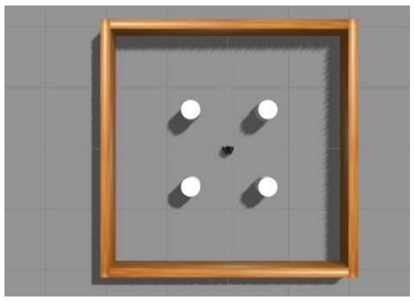
The reward function is exactly as the one proposed by Jaradat (2011):

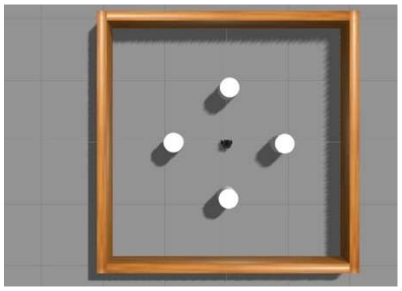
$$r = \begin{cases} 2, \mathrm{SS} \rightarrow \mathrm{WS} \\ 1, \mathrm{NS} \rightarrow \mathrm{SS} \\ 0, \mathrm{NS} \rightarrow \mathrm{NS}, \mathrm{d_{RO}}(\mathrm{n}+1) > \mathrm{d_{RO}}(\mathrm{n}) \\ -1, \mathrm{SS} \rightarrow \mathrm{NS} \\ -1, \mathrm{NS} \rightarrow \mathrm{NS}, \mathrm{d_{RO}}(\mathrm{n}+1) < \mathrm{d_{RO}}(\mathrm{n}) \\ -2, \mathrm{NS} \rightarrow \mathrm{FS} \end{cases}$$

4 Stages of environment

Figure 11 show the stages of each environment, Stage 1 is a 4x4 map with no obstacles, stage 2 is a 4x4 map with four cylinders of static obstacles, stage 3 is a 4x4 map with four cylinders of moving obstacles and stage 4 is a 5x5 map with walls and two cylinders of moving obstacles.


Stage 1 (No Obstacle)


Stage 2 (Static Obstacle)


Stage 3 (Moving Obstacle)

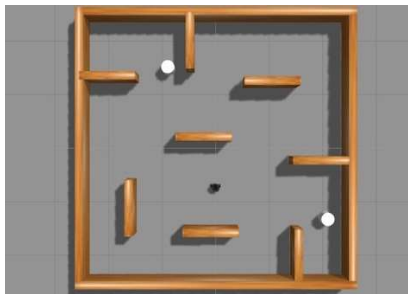

Stage 4 (Combination Obstacle)

Figure 11 Four stages of environment

Note. From https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_ learning/#set-parameters

Conclusion

This research aims to solve the problem of mobile robot using wheels can avoid obstacles by deep reinforcement learning. We have developed the obstacle avoidance system by using deep reinforcement learning (Deep Q-Network), it is a type of reinforcement learning. ROS (Robot Operating System) is an important component of building robotic control systems. In addition, the learning system will be created in simulation environment model, Gazebo. In the part of simulation, the robot is designed to provide the training and testing environments, in order to solve the problem of control for the robot's intelligent decisions in a complicated dynamic environment. We have just tested it in stage 1, the robot learn about moving when there are no obstacle. In stage

2, the robot learn about moving and trying to avoid static obstacle, we found that in this stage, the robot cannot avoid obstacle. The robot move from starting point and hit obstacles many times and then it moves repeatedly around obstacles. So we have to provide more training and testing and find out the suitable parameter for this research. After we get suitable parameter and result from simulation for decision making of robot control system, we will apply to real robots in real environment.

Reference

Abdullah, Hamza. (2018). **Machine learning: A strategy to learn and understand (Chapter 5) Part 5: Reinforcement Learning** [Online]. Available: https://medium.com/the-21st-century/machine-learning-a-strategy-to-learn-and-understandpart-5-reinforcement-learning-8191d7e0406b [2019, 20 March]

Ademovic, Adnan. (n.d.). **An Introduction to Robot Operating System: The Ultimate Robot Application Framework** [Online]. Available: https://www.toptal.com/robotics/introduction-to-robot-operating-system [2020, 5 February]

Bhatt, Shweta. (2018). **5 Things You Need to Know about Reinforcement Learning** [Online]. Available: https://www.kdnuggets.com/2018/03/5-things-reinforcement-learning.html [2019, 20 March]

Duguleana Mihai, Mogan Gheorghe. (2016, June). Neural networks based reinforcement learning for mobile robots obstacle avoidance. **Expert Systems With Applications.** (n.p.). 62 (2016), 104-115.

Gupta, Alind. (2019). **Deep Q-Learning** [Online]. Available: https://www.geeksforgeeks.org/deep-q-learning/ [2020, 5 February]

Jaradat Mohammad Abdel Kareem, Al-Rousan Mohammad, Quadan Lara. (2010, June). Reinforcement based mobile robot navigation in dynamic environment. **Robotics and Computer-Integrated Manufacturing.** (n.p.). 27 (2011), 135-149

Machine Learning [Online]. Available:
https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/#set-parameters [2019, 20 March]

Robert, John. (2020). **Hands-On Introduction to Robot Operating System(ROS)** [Online]. Available: https://trojrobert.github.io/hands-on-introdution-to-robot-operating-system(ros)/ [2021, 5 July]

S. Sutton, Richard & G. Barto, Andrew. (2018). **Reinforcement Learning: An Introduction second edition** [Online]. Available:
http://incompleteideas.net/sutton/book/RLbook2018.pdf [2019, 20 March]

TurtleBot3 Simulation [Online]. Available:
https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation [2019, 20 March]

Why Gazebo? [Online]. Available: http://gazebosim.org [2019, 20 March]