Sensor-based Mobile Robot Navigation via Deep Reinforcement Learning

Seungho-Ho Han and Ho-Jin Choi
School of Computing
KAIST
Daejoen, Republic of Korea
{*seunghohan, hojinc*}@*kaist.ac.kr*

Philipp Benz and Jorge Loaiciga
School of Electrical Engineering
KAIST
Daejoen, Republic of Korea
{*pbenz, mario.lr*}@*kaist.ac.kr*

***Abstract*—Navigation tasks for mobile robots have been widely studied over past several years. More recently, there have been many attempts to introduce the usage of machine learning algorithms. Deep learning techniques are of special importance because they have achieved excellent performance in various fields, including robot navigation. Deep learning methods, however, require considerable amount of data for training deep learning models and their results may be difficult to interpret for researchers. To address this issue, we propose a novel model for mobile robot navigation using deep reinforcement learning. In our navigation tasks, no information about the environment is given to the robot beforehand. Additionally, the positions of obstacles and goal change in every episode. In order to succeed under these conditions, we combine several Q-learning techniques that are considered to be state-of-the-art. We first provide a description of our model and then verify it through a series of experiments.**

***Keywords*-mobile robot navigation, deep reinforcement learning, Q-learning, sensor-based navigation**

I. INTRODUCTION

Navigation tasks for mobile robots represent a broad topic covering a wide spectrum of technologies and applications, such as robotics, optimization problems, etc. For the successful navigation of mobile robots, there are several existing algorithms and other methods [1]. Recently, there have been many attempts to implement novel algorithms using sensor networks [2] and machine learning methods such as deep neural networks (DNNs) [3].

DNN-based algorithms have achieved excellent performance in various fields, including robot navigation, because of the efficient matrix multiplications provided by GPU streaming multi-processor architectures. However, these approaches require the collection of considerable amounts of data for training and it is difficult for researchers to interpret the events when a robot fails to navigate. Hence, we must adjust the hyperparameters of the DNN, retrain the model, and verify the performance of new model. As a result, these tasks must be repeated many times to determine adjusted parameters which make the model's performance well.

In order to solve the learning based problems, one of the unsupervised learning techniques, deep reinforcement learning (DRL) [4], is being studied in various fields. The DRL models perform well in cases where true labels are not practically attainable, but can be generated through the implementation of the actual dynamics of the system, because the outcomes of actions in distinct states are stochastically distributed. Besides, the DRL have recently been applied in image pixels based domains such as video games. Such domains have some rules given an environment, therefore the DRL model is learned using the pixels of a game and a result value scored based on the rules.

There are not many studies applied to the navigation tasks using DRL. But recently, [5] proposed a DRL framework for target-driven visual navigation. For the visual navigation in this paper, the authors used scene data for target and observations, and used deep residual learning method by combining their DRL model to recognize the scene data. Although this paper verified that navigation tasks using DRL works well, their model also used image data for training.

Unlike the model proposed in [5], our paper proposes a novel DRL-based model more suitable for a navigation environment. This means that our model uses only sensory data obtained from sensors mounted on a mobile robot and position data of the robot and goal. Thus, our DQN model is able to learn sensor-based mobile robot navigation. In our navigation tasks, the mobile robot is given no information regarding its environment (i.e., environment dimensions or boundaries). Additionally, the positions of robot and goal change in every episode. Under these conditions, we want to find the goal without any collisions with surrounding obstacles and navigate to the goal using the shortest path. To accomplish this, we use a specific type of network that is known as a deep Q network (DQN) [6]. Additionally, we combine several Q-learning techniques that are considered to be state-of-the-art, including the dueling DQN [7] and double DQN [8].

The contributions of this paper are the following:

- We introduce a novel DRL model for mobile robot navigation. To implement our model, we first define a problem statement about mobile robot navigation task (Sec. III-A), and design a learning setup of our model such as observations, action space, and reward (Sec. III-B). And then, we design and implement our model (Sec. III-C).
- We show how our model is performing well and

2375-9356/18/$31.00 ©2018 IEEE
DOI 10.1109/BigComp.2018.00030

suitable for the navigation problems. To do this, our navigation environment continuously change positions of robot and goal. Under this constrained environment, a mobile robot can navigate to the goal with the shortest path using trained model.

- Finally, we validate the usefulness of our model through experiments (Sec. IV). In our experiments, we perform two types of experiments that are discrete and continuous environment respectively. The details are described in next section.

II. Background

A. Reinforcement Learning

The theory of RL [9] is concerned with how an agent should choose a sequence of actions in an environment to maximize a cumulative reward. The environment is typically formulated as a Markov decision process (MDP), which consists of various states connected by actions. Rewards or penalties after performing actions are defined to provide feedback to the agent. Currently, Q-learning is perhaps the most representative set of RL algorithms. The main components of Q-learning are described below.

1) Q-function: A Q-function is related to the state-action value functions in an MDP. When state and action values are inputted into a Q-function, it estimates a Q-value according to a stochastic policy π and discounted reward R_t as follows:

$$\begin{aligned} Q^{\pi}(s,a) &= \mathbb{E}[R_t|s_t=s,a_t=a,\pi] \\ V^{\pi}(s) &= \mathbb{E}_{a\sim\pi(s)}[Q^{\pi}(s,a)] \end{aligned} \tag{1}$$

where the policy π is a rule that the agent follows when selecting actions given its current state. The discounted reward R_t is described below.

2) Exploitation and Exploration: Q-learning is performed iteratively in accordance with stochastic policies using Q-functions. In this case, the action in each step must be selected through some selection mechanism for next step. Q-learning typically uses exploitation and exploration methods [10], such as the decaying epsilon-greedy algorithm or random noise additions.

3) Discounted future reward: The discounted future reward represents the same concept as the discount factor in an MDP. For updating a Q-value in some state, a reward for the current state and selected action is required. If the number of states and actions ranges from 0 to n, the future reward at a certain step t is defined as $R_t = \sum_{i=t}^{n} r_i$. We can calculate a discounted future reward by applying a discount factor $\gamma \in [0,1]$ to the future reward as follows:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \tag{2}$$

4) Updating Q-value: For updating the Q-value, we must define an optimal Q-value (using equation 1) as $Q^*(s,a) = \max_\pi Q^\pi(s,a)$ and $V^*(s) = \max_a Q^*(s,a)$. It follows that the optimal Q-function can be represented (according to the Bellman equation) as:

$$Q^*(s,a) = \mathbb{E}_{s'}[r + \gamma \max_{a'} Q^*(s',a')|s,a] \tag{3}$$

where s and a are the current state and action, and s' and a' are the next state and action.

B. Deep Q Network

Conventional Q-learning uses a Q-table for managing Q-values during learning. However, the Q-table has a critical limit, which creates a capacity problem. To solve this problem, Q-learning using a neural network was proposed [4] and called a Q-network. However, this simple Q-network diverges because of the correlation between samples and non-stationary targets. In [6], the authors suggested the following solution to the divergence problem.

1) Huge network: First, they scaled the network to be much deeper so that it could be stabilized and become convergent. In other words, they used a huge deep neural network combined with Q-learning.

2) Experience replay: In order to solve the problem of correlation between samples, they suggested the concept of an experience replay. This term refers to storing an agents experiences during training and then sampling batches in a random manner to train the network. By keeping past experiences, one can prevent the neural network from learning based only on its latest actions and encourage it to learn from diverse and random past experiences.

3) Separate target networks: In order to solve the non-stationary targets problem, the DQN can be separated into a network for training and updating weights (primary DQN), and a target network for calculating target Q-values (target DQN). In the DQN model, the input is an observation (or state) at step t and the output is the maximum Q-value among the Q-values the input state and each action. To train the model, we must find weights to minimize the following loss function:

$$\min_\theta \sum_{t=0}^{T} [\hat{Q}(s_t,a_t;\theta) - y_t^{DQN}]^2 \tag{4}$$

with

$$y_t^{DQN} = r_t + \gamma \max_{a'} \hat{Q}(s_{t+1},a';\bar{\theta}) \tag{5}$$

where $\hat{Q}(s_t,a_t|\theta)$ represents a predicted maximum Q-value at step t from the DQN with weight θ and y_t^{DQN} represents the target Q-value from the separate target DQN with weight $\bar{\theta}$.

(4) is the loss function for the DQN. We approximate the predicted Q-value based on the optimal Q-value $\hat{Q}(s,a|\theta) \sim Q^*(s,a)$ by minimizing the loss function. As shown (5), the target Q-value is obtained from the target DQN. However, the weights of the target DQN are not initially optimal. Therefore, the weights of the original DQN are periodically copied to the target DQN during learning.

C. Double DQN

In our approach, we apply state-of-the-art DQN techniques to our DQN model. The traditional Q-learning algorithm is known to overestimate action values under certain conditions. One method to prevent this issue is known as the double DQN [8]. The double DQN concept proposes a simple modification to the traditional DQN: the procedure for computing the maximum Q-value from the target DQN is performed by two different DQNs. First, a primary DQN is used to calculate the best possible action given the current state s. This choice is then passed to the secondary DQN, which computes the target-Q value for the current training step. By decoupling the action choice from target Q-value generation, the authors of [8] achieved results where overestimations were substantially reduced, training time was shortened, and accuracy was improved. Formally, this modification alters the traditional Bellman equation used for updating the target value, as shown in (6).

$$y_t^{DDQN} = r + \gamma Q(s', \underset{a'}{\operatorname{argmax}} Q(s', a'; \theta_t); \bar{\theta}) \tag{6}$$

D. Dueling DQN

We use the concept of the dueling DQN [7] to generalize our model for changes in the environment of a given maze (such as the positions of the robot or goal). The justification for the dueling DQN requires detailed consideration of network bias issues. Using a dueling DQN, the Q-value shown in (1) can be further decomposed into two fundamental notions. The first is a value function $V(s)$, which quantifies how beneficial it is to stay in any given state. The second is an advantage function $A(a)$, which provides a measurement of how much better it is to choose a particular action over the others. Thus, the dueling DQN uses two streams to separately estimate state values and the advantages of each action, then combines them to calculate a final Q-value.

The key advantage of this technique is that an agent can obtain a significant reward not only by being active and moving from one state to the next, but also by remaining in a particular state when environmental conditions dictate this to be the best strategy. Thus, it is possible to achieve more accurate estimations as the networks are able to fine-tune the parameters linked to $V(s)$ and $A(a)$ independently. A more detailed equation for the dueling DQN will be described in the next section.

III. Sensor-based Mobile Robot Navigation

In this section, we describe our model for mobile robot navigation. First, we state our navigation problem and learning setup for training our model, such as observations, action space, reward design, etc. Next, we describe our model in detail and our algorithm for the training process.

A. Problem Statement

The objective of our DQN model is to have our mobile robot find an optimal path to the goal from an arbitrary location without colliding with any obstacles in the given map. To accomplish this, we implement a DQN model that takes sensory data provided to the robot as input (or observations). The output of the model is instructions for robot movement, such as move forward, turn left, turn right, and move backward, which are executed by modifying the positions of the robot's wheels.

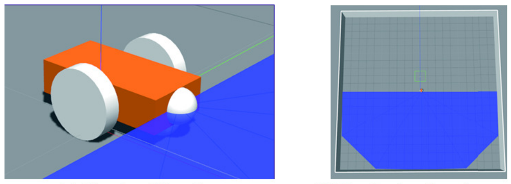

(a) Virtual mobile robot (b) Simple virtual environment

Figure 1. Virtual environment using Gazebo.

As shown in Fig. 1, we implemented a virtual mobile robot and environment for navigation tasks using Gazebo [11], which provides a 3D dynamic multi-robot environment in an open-source format. As mentioned in previous sections, the DNN-based approach has some limitations, including requiring a large amount of data to train model and producing results that are difficult to interpret. Additionally, a traditional DQN [6] must be retrained depending on changes in the maze or environment when applying it to navigation tasks because it lacks the ability to generalize. In practice, these are difficult problems to solve.

Our model is able to learn to navigate mazes without using any previously collected data and without retraining because we apply the double DRL and dueling DRL methods mentioned above. In other words, even if the obstacles and goals in given maze change, the mobile robot can still navigate to the goal because our model is generalized. In order to control the mobile robot to navigate the maze, we use robot and goal positional data, as well as the distances between obstacles and the robot in the maze. The distances are obtained using seven laser sensors mounted on the robot. Therefore, we refer to this navigation task as sensor-based mobile robot navigation.

As learning progresses, the model learns the distances between the robot and the obstacles or goals. If the robot collides with an obstacle, the model learns about that obstacle by receiving a penalty (i.e., negative reward) based on the distance between each sensor on the robot and the obstacle. Similarly, when the robot arrives at a goal, the model learns about that goal based on the distance to the goal and the goal positional data by receiving a positive reward. Therefore, no retraining for changes in obstacle or goal positions in a given maze is required.

Formally, the learning objective of our sensor-based model is to find the set of action values that maximizes the cumulative reward from the starting point to a goal based on a policy $\pi(s)$. To find this action value set, we use the concept of the dueling DQN described previously. Using this concept, the Q-values calculated in the output layer are decomposed into two layers to estimate the state-value (output value of deep feedforward network in our model) and advantages as a scalar $V(s;\theta,\beta)$ and $|\mathcal{A}|$-dimensional vector $V(s,a;\theta,\alpha)$, respectively, where θ denotes the weights of the DQN, and α and β are the parameters of two split layers. The state-value function $V(s;\theta,\beta)$ means that how good it is to be in any given state, and advantage function $A(s,a;\theta,\alpha)$ means that how much better taking a certain action would be compared to the others. Thus, the final Q-value can be calculated using the following equation:

$$Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(A(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a'} A(s,a';\theta,\alpha)\right) \tag{7}$$

B. Learning Setup

Before describing our model in detail, we first introduce the key components of the DQN learning setup below.

1) Observations: One observation is composed of four consecutive sets, each of which is composed of sensory data and positional data for the robot and goal. Our robot is equipped with seven lasers sensors $(d_0, d_1, d_2, d_3, d_4, d_5, d_6)$ to measure the distances between itself and surrounding objects, in addition to a proximity sensor (D) where proximity is calculated internally using the positional data of the goal and robot. All sensor values are normalized by their maximum variation, as shown in Table I.

Table I
SENSOR VALUES: (G_xG_y) AND (R_x, R_y) ARE THE COORDINATE POSITIONS OF THE GOAL AND ROBOT

Sensor	Normalized value
d_i	$\|\|d_i\|\|/10\sqrt{2}$
D	$\sqrt{(G_x-R_x)^2+(G_y-R_y)^2}/10\sqrt{2}$

The reason that we use four consecutive sets for observations is that the sensor data has continuous characteristics because the experimental environment is continuous. Thus, considering several sensor data in succession, rather than only a single sample of sensor data, yields significant performance improvements. We also use the positional data of the robot and goal to determine if the robot has reached the goal. In summary, our observations consist of a set of four consecutive sensory data and positional data points, where each set takes the form $\{(d_0, d_1, d_2, d_3, d_4, d_5, d_6), (G_x, G_y), (R_x, R_y)\}$.

2) Action space: For our mobile navigation tasks, we consider four actions: forward, backward, turn left, and turn right. These actions are mapped to denote the spin of the two robot wheels. For our main experiment, the mobile robot has a velocity value (0 1), because we considered a 3D continuous environment.

3) Reward design: We focus on two tasks when the robot is navigating. The first is to reach the goal and the second is to find the shortest path to goal. To accomplish this, the reward design is important because the rewards determine how the robot learns and ultimately affect the navigation performance of the robot. The scoring convention awards +10 points upon collision with the goal and -5 points upon collision with anything else. If no collisions occur, the scoring convention awards a point value between -1 and -0.5 depending on the current number of steps.

It is important to note that if the robot collides with obstacles or moves without any collisions, the model learns the maze even if it does not find the goal because it receives negative rewards based on obstacle collisions, goal proximity, and the distance between itself (specifically, the seven laser sensors equipped in the robot) and obstacles. In contrast, when the robot arrives at the goal, the model learns the maze, other than obstacle positions, by receiving positive rewards. Therefore, as learning progresses, the model learns the maze based on such rewards. If the model receives negative rewards, it means that the robot and an obstacle are getting closer to each other, but if the model receives positive rewards, it means that the robot and goal are getting closer to each other.

C. Model and Network Architecture

For the robot to navigate to the goal successfully, we implemented a novel DQN suited to our particular application. Our model is illustrated in Fig. 2. As shown in the figure, we use a deep feed-forward network (DFFN) for our Q-network. The reason for using the DFFN is that it yields better performance than other deep learning algorithms because it network is fully-connected between layers. Additionally, we use split layers for calculating Q-values using state-value and advantage functions.

The network has five fully-connected layers, an input layer (512), three hidden layers (256, 128, 64), and an output later (32). To train the model, we used the stochastic gradient descent (SGD) algorithm with a loss function and mean squared error. An observation is fed into the network and the output is the maximum Q-value among all the Q-values from the split layers. We then calculate the difference between the output Q-value from primary the DQN and Q-value of the target DQN, and update the weight of the network to minimize the loss function. The target DQN has the same network architecture as the primary DQN. However, the weights of the target DQN are not updated continuously like those of the primary DQN. Instead, it copies the updated

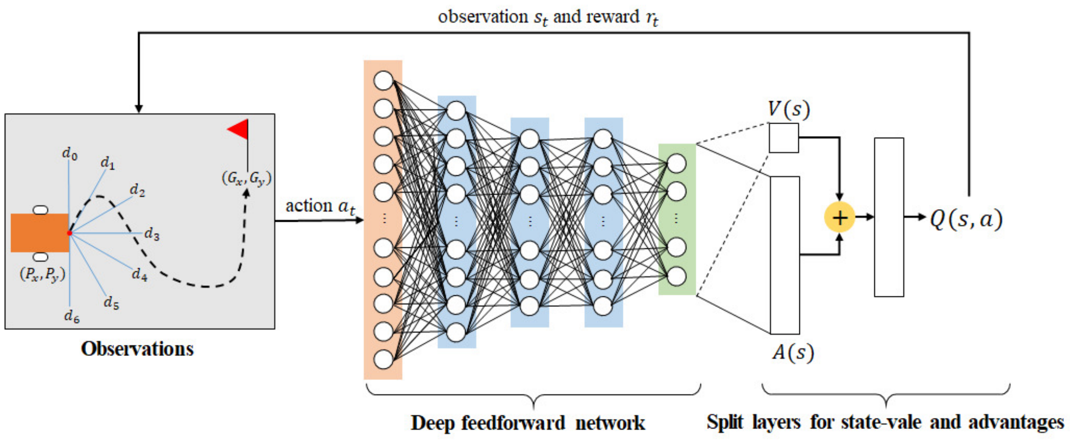


Figure 2. Network architecture for our DQN model

weights at regular interval τ from the primary DQN. The entire training process is outlined in Algorithm 1.

As shown in the algorithm, we first initialize the maze environment, data storage (i.e., experience replay memory), primary DQN, and target DQN . The training process is performed with a maximum number of episodes (EP_{max}) and the positions of the robot and obstacles are randomly initialized for each episode (reset *env* in the algorithm). In our algorithm, there is a maximum number of steps that the robot can move in one episode (*max_ep_step* in the algorithm). If the robot does not reach the goal within the maximum number of steps, the episode fails and we move to the next episode.

The action for the current state is selected through exploitation and exploration by comparing a randomly selected floating value with a predefined epsilon value. Additionally, we define the minimum number of training steps to collect initial data for mini-batches in the data storage D. In other words, when the current step value is smaller than this predefined step value, the action is always selected randomly. Then, we determine the next step using the selected action and check whether or not the next state is the goal state (if the robot arrives at the goal, the current episode is finished). Next, the current experience is stored in the data storage D and the primary DQN is trained by sampling mini-batches from the data storage. Finally, we update the weights of the target DQN by coping the weights from the primary DQN at regular interval τ.

Algorithm 1 DQN Training algorithm

```
Initialize environment $env$ $\{robot, goal, obstacles\}$
Initialize data storage $D$
Initialize primary DQN $Q_{primary}$ with random weights $\theta$
Initialize target DQN $Q_{target}$ with random weights $\bar{\theta}$
for $episode := t$ $EP_{max}$ do
    $s_t$ = reset $env$
    while $current\_step < max\_ep\_step$ do
        if $random\_floating\_value < \epsilon$ then
            select a random action $a_t$
        else
            $a_t = \text{argmax}_a Q_{primary}(s_t, a; \theta)$
        end
        $s_{t+1}$ = move to next state using $a_t$
        store current experience $\{s_t, a_t, r_t, s_{t+1}\}$ in $D$
        if $current\_step > predefined\_train\_step$ then
            train the network with mini-batches from $D$
            $y_i = \begin{cases} r_i \\ r_i + \gamma \max_{a'} Q_{target}(s_{i+1}, , a'|\theta_i); \bar{\theta}) \end{cases}$
            calculate the loss $(y_i - Q_{primary}(s_i, a_i|\theta))^2$
            update weights $\theta$ of the primary DQN
            every $\tau$ steps, copy weights $\theta$ to weights $\bar{\theta}$
        end
    end
end
```

IV. Experiment

The objective of the mobile robot navigation tasks is to navigate the goal through the shortest path without colliding with any obstacles using only seven laser sensors mounted on the robot and the position data of the robot and the goal. The major difficulty in this problem is that the positions of the goal and robot in a given environment change randomly for each episode. Our proposed DQN-model and algorithm for meeting this objective were described above. To validate our model, we simulated two experiments: 1) in a discrete 2D grid environment and 2) in a 3D continuous environment. For the experiment in the 2D environment, we performed the simulations with a simpler learning setup to verify that the mobile robot was able to find the goal using our algorithm. For the experiment in the 3D environment, we performed the simulations using the learning setup introduced in Sec. III and analyzed the performance of our model.

A. Coordinate-assisted Navigation in Discrete Environment

First, we simulated mobile robot navigation using our model in a discrete 2D grid environment. In this experiment, we verified that the mobile robot was able to reach the goal using our algorithm. For this experiment, we simplified the learning setup because the experimental maze was based on a grid environment with x and y coordinates. The environmental settings and learning setup for the 2-D grid maze were as follows:

- Maze environment: For this experiment, we tried various grid sizes ranging from very small to large. In this paper, we only present a maze with a 15 x 15 grid size because the performances for each maze was very similar.
- Observations: In this environment, the robot had no laser sensors. Therefore, we used observations of the positions of the obstacles, goal, and robot in the form $\{(R_x, R_y), (G_x, G_y), (H_{x1}, H_{y1}), \cdots, (H_{xn}, H_{yn})\}$, where (R_x, R_y) is the position of the robot, (G_x, G_y) is the position of the goal, and (H_{xm}, H_{ym}) is the position of an m^{th} obstacle.
- Action space: The action space for this experiment was the same as that described in Sec. III. The only difference is that the movement of the robot was based on coordinates of the form (x, y) because this experiment utilized a discrete environment.
- Reward design: The scoring convention awarded +4 points upon reaching the goal, -1 points for a collision, and 0 otherwise. The reason that the reward was bigger when reaching the goal +4 was to force the model to learn the direction of the goal.

Fig. 3 presents a simple example of a 2D maze environment. As shown in Fig. 3, the maze contains the robot, goal, and some obstacles (the robot and goal are labeled for identification). The blue square represents the mobile robot and the green square represents to the goal. Red squares serve as obstacles. Upon completion of a training episode, the goal and the robot are placed in random locations on the grid for the next episode.

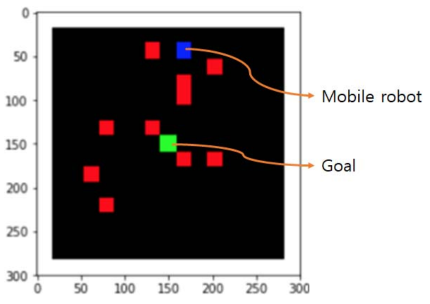


Figure 3. Discrete 2D grid environment example

Our model must be trained for successful navigation in a given environment. The training process consisted of 10,000 episodes; 50,000 pre-training steps (i.e., the minimum number of training steps without network updating); and 60 maximum steps per episode. After training the model, we tested it by comparing it to a random agent. Fig. 4 presents the results of the experiment.

Fig. 4 presents the results of training the model in terms of the amount of variation in loss values, sum of rewards, and steps. Fig. 4a presents the amount of variation in loss values during the training process. The reason that the loss values are a negative number is because we used the log arithmetic function because the deviations in the loss values were too large. In this figure, the loss values continuously decrease as learning progresses. This indicates that the Q-network's weights are being optimized according to the gradient algorithm.

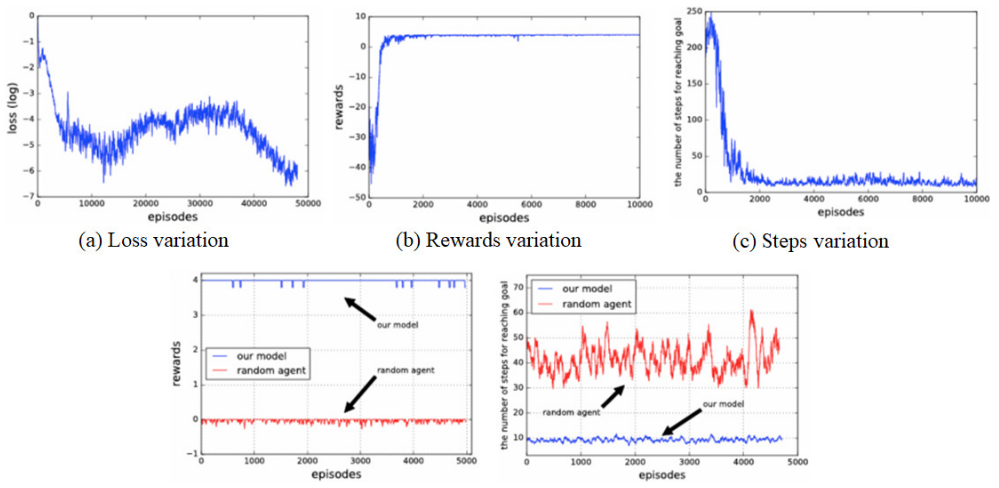
Fig. 4b and Fig. 4c presents the amount of variation in rewards and steps to reach the goal per episode. As training of the model progresses, the rewards move closer to four points and the number of steps decreases. This indicates that our model has been trained well because the sum of rewards moving toward four points means that the number of arrivals at the goal is increasing, and the reduction in the number of steps means that optimized paths are being used to find the goal. To summarize, the results verify that our algorithm using DQN is able to accomplish our navigation tasks.

Fig. 4d and Fig. 4e presents the test results by comparing our model to a random agent. Fig. 4d presents the results in terms of the sum of rewards per episode for each agent. As shown in the figure, the mobile robot in our model mostly received the +4 point reward, which indicates that the robot typically reached the goal without any collisions with obstacles. In contrast, the random agent, which randomly selects an action for each step, only received 0 or -1 point rewards, which indicates that the random agent always collided with obstacles or did not reach the goal. (The success rates for our model and the random agent were 95.6% and 2.6%, respectively. The graph was drawn using the averages of 15 rewards for all sums of rewards)

Fig. 4e presents the number of steps to reach the goal for each episode. Our model typically reached the goal in approximately 10 steps. However, the random agent had an average step count of 40. This means that the robot using our model nearly always navigated to the goal using the optimal path. In contrast, the random agent collided with obstacles during navigation or did not find the goal (exhausted all opportunities).

B. Sensor-driven Navigation in Continuous Environment

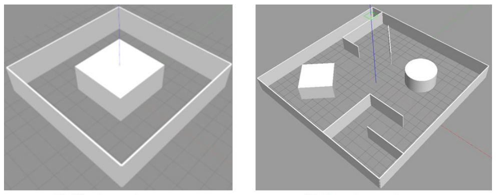
In this experiment, we simulated mobile robot navigation in a 3D continuous environment. To implement the 3D continuous environment, we used a program called Gazebo, as mentioned in Sec. III. In this experiment, we proved that



(a) Loss variation (b) Rewards variation (c) Steps variation

(d) Rewards comparison of our model and random agent (e) Steps comparison of our model and random agent

Figure 4. Training and testing results in 2D grid environment: comparison of the mobile robot using our model to a random agent in terms of sum of rewards and steps to reach the goal.



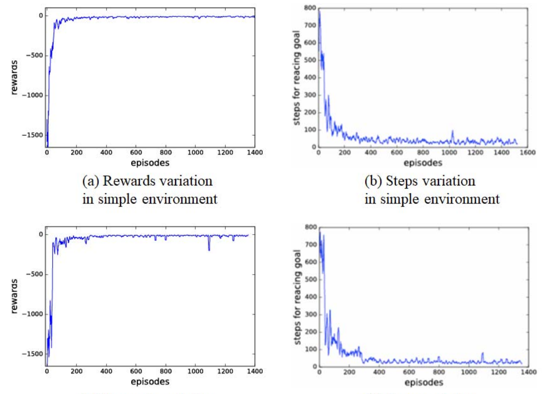
(a) Simple environment (b) Complex environment

Figure 5. Examples of 3D continuous maze used in the experiment

our model is superior to a random agent and the simple DQN model in a 3D continuous environment (the simple DQN is a basic DQN model without applying extra techniques, such as double DQN and dueling DQN).

Because the environment was continuous, the mobile robot was constrained, meaning it had to learn how to maneuver more carefully. There was also a chance that the goal would not be visible upon episode start, which forced the robot to perform more intensive exploration of the environment. Our mobile robot used seven laser sensors to measure the distances between itself and surrounding objects. Our model used the learning setup described in Sec. III. The experimental environments were the two mazes shown in Fig. 5.

As shown in Fig. 5, we experimented using two 3D continuous mazes (simple and complex maze). We placed the robot and goal randomly within these two maxes to perform training and test our model. The training process consisted of 2,000 episodes, with 250 maximum steps per episode. In a continuous environment, the length of one step depends on the velocity of the robot. In this experiment, we increased the speed value of the robot to the maximum possible value for quick learning. We fixed the maximum length of one episode to 250 steps and varied the number of episodes. The training results under these conditions are presented in Fig. 6.



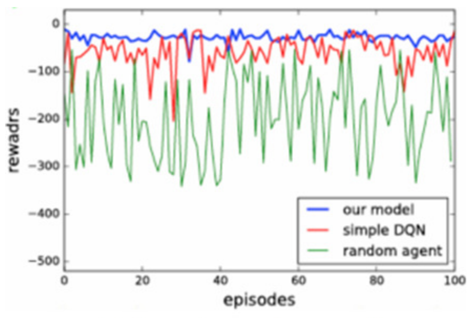
(a) Rewards variation in simple environment (b) Steps variation in simple environment

(c) Rewards variation in complex environment (d) Steps variation in complex environment

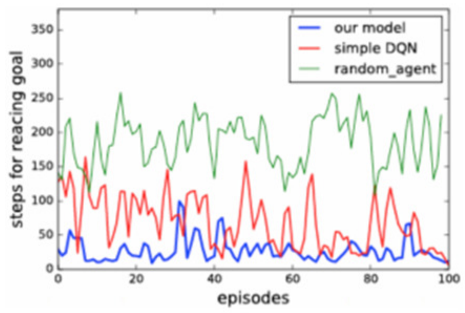
Figure 6. Training results from 3D environments

As shown in Fig. 6, the model for the 3D environment is trained following a trend similar to that in the 2D grid environment. Similar to the simple environment, as training progresses, the reward approaches zero (specifically, between -50 and -20) and the number of steps gradually decreases each time the robot reaches the goal. Thus, the results of this experiment indicated that the training of the model was successful.

Fig. 7 presents the result of testing in the 3D continuous environments by comparing our model (blue line) to a simple DQN model-based agent (red line) and random agent (green line) using 100 testing episodes. Fig. 7a presents the sum of rewards for each episode. By comparing the graphs in the figure, one can see that the sum of rewards for our model is the closest to zero, followed by simple the DQN model-based agent and the random agent. Similarly, Fig. 7b presents the number of steps required to reach to the goal. The results show that our model required the lowest number of steps. This is a reasonable result because the sum of rewards is the largest for our model.



(a) Sum of rewards comparison



(b) The number of steps comparison

Figure 7. Testing results: comparison of our model to a simple DQN and random agent in terms of sum of rewards and steps required to reach the goal per episode.

V. Conclusion

In this paper, we proposed a DQN-based model for sensor-based mobile robot navigation by applying a state-of-the-art Q-learning techniques to further refine our model and improve its performance. Consequentially, we can find the optimal path to the goal using our model, which uses laser sensory data and the positional data as observations. Our experiments demonstrated that our model is superior to a simple DQN model-based agent and random agent by comparing them through navigation simulations. Our future work will include two things. The first is that we will be conducting more experiments comparing with other DNN or DRL models. And the second is that we will improve our DRL model using other DNN models, such as RNNs or LSTM.

Acknowledgment

This work was supported by Institute for Information CommunicationsTechnology Promotion (IITP) grant funded by the Korean Government (MSIP)(No.2013-0-00131, Development of Knowledge Evolutionary WiseQA Plat-form Technology for Human Knowledge Augmented Services; and No.2017-0-00868, Development of Conversational Solution for Intelligent Chat Ser-vices Based on Pragmatic and Context Analysis of Dialogues).

References

[1] R. Bayuk and S. Ratering, "A comparison of robot navigation algorithms for an unknown goal," 2005.

[2] M. A. Batalin, G. S. Sukhatme, and M. Hattig, "Mobile robot navigation using a sensor network," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 1. IEEE, 2004, pp. 636–641.

[3] B. Ko, H.-J. Choi, C. Hong, J.-H. Kim, O. C. Kwon, and C. D. Yoo, "Neural network-based autonomous navigation for a homecare mobile robot," in *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*. IEEE, 2017, pp. 403–406.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[5] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *arXiv preprint arXiv:1609.05143*, 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[7] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling network architectures for deep reinforcement learning," *arXiv preprint arXiv:1511.06581*, 2015.

[8] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016, pp. 2094–2100.

[9] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[10] S. Whiteson, M. E. Taylor, and P. Stone, "Empirical studies in action selection with reinforcement learning," *Adaptive Behavior*, vol. 15, no. 1, pp. 33–50, 2007.

[11] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.