Expert Systems With Applications 198 (2022) 116742

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

Towards monocular vision-based autonomous flight through deep reinforcement learning

Minwoo Kim [a], Jongyun Kim [b], Minjae Jung [a], Hyondong Oh [a,*]

[a] Ulsan National Institute of Science and Technology (UNIST), Republic of Korea
[b] Cranfield University, United Kingdom

ARTICLE INFO

Keywords:
Obstacle avoidance
Depth estimation
Vision-based
Deep reinforcement learning
Q-learning
Navigation decision making

ABSTRACT

This paper proposes an obstacle avoidance strategy for small multi-rotor drones with a monocular camera using deep reinforcement learning. The proposed method is composed of two steps: depth estimation and navigation decision making. For the depth estimation step, a pre-trained depth estimation algorithm based on the convolutional neural network is used. On the navigation decision making step, a dueling double deep Q-network is employed with a well-designed reward function. The network is trained using the robot operating system and Gazebo simulation environment. To validate the performance and robustness of the proposed approach, simulations and real experiments have been carried out using a Parrot Bebop2 drone in various complex indoor environments. We demonstrate that the proposed algorithm successfully travels along the narrow corridors with the texture free walls, people, and boxes.

1. Introduction

Recently, there is an increase in demand for small drones in complex indoor environments. The employment of small drones enables various tasks such as search and rescue, environmental mapping, and exploration of unknown areas owing to their agility and small size. In order to complete the missions, drones should remain intact for all situations. For this reason, drones should be equipped with an obstacle avoidance (OA) algorithm that functions well in dangerous circumstances such as sudden emergence of moving obstacles, corners, and narrow corridors. However, there are various challenges for development of OA algorithms. One of the biggest problems is that small drones cannot load heavy sensors due to their small size and low battery power, which could degrade the performance of OA algorithm. Thus, selecting a proper sensor that best suits to small drones is a crucial task.

Commonly used sensors for OA tasks are ultrasonic sensors, LiDAR (Light detection and ranging) sensors, depth cameras, and monocular cameras. These sensors have their own strengths and weaknesses. Ultrasonic sensors measure close obstacles with relatively good performance; however, their accuracy decreases as the distance to the obstacle increases. LiDAR sensors provide rich information from the surroundings with high accuracy, long detection range, and wide field of view (FOV), but the cost is exceptionally expensive compared with other sensors. Besides, the weight of LiDAR sensors is quite heavy, which makes it difficult to be used on small drones. Depth cameras can provide the distance information of the close obstacles with geometric properties, but the weight and power consumption might be ineffective. Monocular cameras provide less information compared with LiDAR sensors and depth cameras, but they have many advantages such as light weight, small size, low energy consumption, and wide FOV with reasonable amount of information from surrounding environments. Therefore, among various sensors, monocular cameras are the most suitable candidate to be applied in small drones.

In order to fully utilize the aforementioned advantages of monocular cameras, various OA algorithms have been developed. However, directly using the RGB information from a monocular camera is challenging as monocular cameras reduce 3D real-world data into 2D image pixels, which makes it difficult to get the accurate distance information from obstacles. To deal with this problem, various studies utilize the features from RGB images to their algorithms. Alvarez, Paz, Sturm, and Cremers (2016) builds dense depth maps using the RGB images from a monocular camera and based on constructed depth maps, waypoints without obstacles are generated. The drone successfully avoids obstacles in most of the cases; however, it needs at least 1 s to compute waypoints, which could lead to failure for dynamic environments. Cho, Kim, and Oh (2019) and Green and Oh (2008) utilized an optical flow technique for obstacle avoidance. The optical flow-based methods mimic the behavior of the insects. As it requires low computational load, it fits well for small drones where high performance computing boards are unavailable. However, applying the optical flow to texture-free environments or huge obstacles is inappropriate as feature points

* Correspondence to: Department of Mechanical Engineering, Ulsan National Institute of Science and Technology (UNIST), Republic of Korea.
E-mail addresses: red9395@unist.ac.kr (M. Kim), jongyun.kim@cranfield.ac.uk (J. Kim), starshirts@unist.ac.kr (M. Jung), h.oh@unist.ac.kr (H. Oh).

https://doi.org/10.1016/j.eswa.2022.116742
Received 27 April 2021; Received in revised form 10 November 2021; Accepted 22 February 2022
Available online 9 March 2022

are not easily extracted under these situations (Eresen, İmamoğlu, & Efe, 2012). There is another widely-used method called visual simultaneous localization and mapping (SLAM) based on a feature extracting method. The visual SLAM estimates the current position and orientation of the unmanned aerial vehicle (UAV) using 3D map information. In Mur-Artal, Montiel, and Tardos (2015), the OA algorithm computes waypoints using the visual SLAM and the drone navigates through environments with obstacles. However, the visual SLAM approach requires high computational load and memory since it uses pose information of UAVs and simultaneously stores the map information (Aulinas, Petillot, Salvi, & Lladó, 2008). Besides, both optical flow and visual SLAM methods need appropriate parameter settings for effective performance depending on situations and thus applying them in various scenarios could be problematic.

Deep neural networks, on the other hand, effectively reduce the burden of making sophisticated rules and can thus be applied in various situations with little computation time in the test phase using pre-trained neural networks. Deep learning methods enhanced the performance of algorithms in various fields including robotics, autonomous systems, and computer vision (Ahmed et al., 2019; Back, Cho, Oh, Tran, & Oh, 2020; Pozna, Troester, Precup, Tar, & Preitl, 2009). Especially in the computer vision field, the level of situation awareness based on visual information has been dramatically improved. For example, AlexNet (Krizhevsky, Sutskever, & Hinton, 2012) has significantly improved the object classification performance by introducing a convolutional filter and it becomes a necessity in the computer vision area.

Supervised learning, as a typical method of deep neural networks, uses a pair of inputs and output data and figures out the relationship between them. Park and Oh (2020) trained neural networks for the OA decision making using sets of RGB images and corresponding heading commands. The images are obtained from the monocular camera attached on the drone in the virtual simulator, following the behavior of human. Loquercio, Maqueda, Del-Blanco, and Scaramuzza (2018) trained the network that determines the heading and velocity commands based on the RGB images from KITTI data sets (Geiger, Lenz, Stiller, & Urtasun, 2013). Chakravarty et al. (2017) and Yang et al. (2017) trained the neural network that can estimate the depth images from RGB images. Then, the estimated depth images are used to determine the heading direction of the UAV. However, training a depth estimation network requires long training time and it sometimes underperforms in unseen environments. Above studies reduce the efforts of designing rules or constraints for OA task; however, they require a lot of time on collecting training data and may cause mission failure when drones encounter unseen environments. Besides, the trained networks might not be able to produce the best command as the trained networks are trained with data labeled manually.

As another big branch of machine learning methods, reinforcement learning is widely used in OA tasks. Reinforcement learning produces actions based on the states so as to maximize rewards. In reinforcement learning, the agent learns a specific policy (e.g., tracking a target, grasping an object, or avoiding obstacles) by trial-and-error, while automatically collecting training data. As a result, the agent can select the best actions in various states. As the reinforcement learning is based on trial-and-error search methods, training the agent from the beginning of the training in real world is ineffective and risky. However, it might be difficult to be applied directly in real environments due to the discrepancy between simulation and real-world images, which is normally termed as the reality-gap (Jakobi, Husbands, & Harvey, 1995).

In order to relieve the reality-gap, some studies transfer the learned model from simulation environments to real environments (i.e., transfer learning) (Ahn & Song, 2020; Kang, Belkhale, Kahn, Abbeel, & Levine, 2019), use images with common features between virtual and real environments (e.g., depth images) during the training (Loquercio et al., 2021; Ramezani Dooraki & Lee, 2018; Wu, Abolfazli Esfahani, Yuan, & Wang, 2018), or use virtual simulators that are similar to real environments (He, Aouf, Whidborne, & Song, 2020; Roghair, Ko, Asli, & Jannesari, 2021; Sadeghi & Levine, 2016). Ahn and Song (2020) trained the robot arm grasping policy in the simulation using a vision sensor and deploy the learned policy in the physical worlds with additional training in real-world environments. Kang et al. (2019) trained an obstacle avoiding policy in the simulation and successfully verified the performance of the algorithm in a real world with a little fine-tuning. However, both studies (Ahn & Song, 2020; Kang et al., 2019) needs further real-world training and there still remains the potential dangers of mission failures. Loquercio et al. (2021), Ramezani Dooraki and Lee (2018) and Wu et al. (2018) trained the OA algorithm using depth images from a stereo camera. The algorithms are verified using an unmanned vehicle with a depth camera. Despite the advantage of a stereo camera, sensory inputs of stereo cameras often contain a large noise, and this could weaken the performance of the algorithm in real-world environments. Sadeghi and Levine (2016) trained the agent in real world-like environments entirely made by CAD models. By using highly randomized rendering settings in CAD models, the agent learns the OA policy without using real data and verified its performance in physical worlds. Although the algorithm has good performance, constructing numerous real world-like environments (Sadeghi & Levine, 2016) requires significant efforts and time. He et al. (2020) and Roghair et al. (2021) use real-like virtual simulators to train the obstacle avoidance algorithms but they are not evaluated in the real environments.

As an alternative way, some studies adopt image preprocessing methods that estimate depth images from RGB images. One of the frequently used methods is to use estimate depth images through deep neural networks. The estimated depth images may show less performance compared with using real depth images. In reinforcement learning, however, different from supervised learning methods, the agent can learn a better policy through continuous trial-and-error training. Singla, Padakandla, and Bhatnagar (2019) used generative adversarial networks (GANs) for depth estimation. For the OA algorithm, it adopts deep recurrent neural networks (DRQNs) which solves the partial observability problem in deep Q-networks (DQNs) using sequential time series data as the network inputs. In Xie, Wang, Markham, and Trigoni (2017), the CNN estimates depth images and then, the dueling double deep Q-networks (D3QNs) use estimated depth images to select the best guidance commands. Above two methods generally show good performance, but they are prone to produce a zigzag motion (Singla et al., 2019) or a spinning motion (Xie et al., 2017) due to their reward functions. These motions might require excessive control efforts, which leads to mission failure especially for small drones with less battery capacity. Therefore, a new reward function needs to be designed that fits to complex environments minimizing the battery loss.

With these backgrounds in mind, in this paper, we consider three conditions for the development of the efficient OA algorithm. First, the algorithm should be based on a monocular camera so that it can be applied to a small drone. Second, the algorithm should show robust performance in both simulation and real environments without further real-world training. Third, the algorithm should minimize the unnecessary motions considering the low battery capacity of small drones. From these conditions, this paper proposes the deep reinforcement learning algorithm that performs the OA missions for small drones in various indoor situations only through a monocular camera. For more stable and better performance of the algorithm, D3QN is used to complement existing problems of DQN methods (e.g., overestimation of Q-values). The main contributions of this paper are described as below:

- We implement a deep reinforcement learning-based OA algorithm in real complex indoor environments using the estimated depth information with only a monocular camera while only a few studies reported real flight experiments of a drone using deep reinforcement learning;

- We design an appropriate action space and reward function, and thus the proposed D3QN model successfully performs safe OA tasks and shows better performance compared with the existing approaches; and
- We deploy the learned policy (trained only in simulations) in the physical drone platform without any adaptation or fine-tuning.

The rest of the paper is organized as follows. Section 2 gives the concept of the deep reinforcement learning and OA algorithm. Section 3 describes the training and simulation environment for the deep reinforcement learning. Section 4 shows the simulation and real experiment results. Finally, Section 5 concludes the paper with the future work.

2. Reinforcement learning-based obstacle avoidance

In this section, we introduce the deep reinforcement learning approach for the OA algorithm. Most of reinforcement learning algorithms are suffering from reality-gap problems due to the discrepancy between simulation and real-world environments. Our framework aims to develop an algorithm trained only in simulation environments and directly apply the trained policy to a real world without any further fine-tuning. For this, a depth image is used as a sensory input in the simulation as it has a strong similarity with the physical world. In our framework, simulated depth images with suitable noises are used as inputs to the artificial neural networks during training and estimated depth images from a monocular camera are used for the real world testing. The OA algorithm is composed of a depth estimation step, followed by a decision making step.

2.1. Depth estimation from RGB images

As the proposed OA algorithm is based on only a monocular camera in real-time, a fast and accurate estimation of depth information is needed. Fully connected residual network (FCRN) (Laina, Rupprecht, Belagiannis, Tombari, & Navab, 2016), inspired by ResNet-50 (He, Zhang, Ren, & Sun, 2016), takes an RGB image as an input and produces a depth image within 50 ms on our setup. Fig. 1 shows an RGB image and corresponding various depth images in the Gazebo simulator. In Gazebo environment, the agent could obtain both RGB images and true depth images. Although the true depth images can be directly used in training, the OA algorithm trained from these depth images could not be transferable to various real-world situations. Besides, the trained policy should be robust to disturbances such as sensor noise and illumination conditions. By adding Gaussian noise and blur effects to the true depth images (obtained in the Gazebo simulator), the network could be well generalized to most environments, as the network is less-overfitted to training environments due to the noise in the depth images. In physical world implementation, a pretrained depth estimation algorithm generates an estimated depth image by using an RGB image obtained from a monocular camera. Fig. 1(c) shows the noisy depth and Fig. 1(d) shows the estimated depth image through FCRN using RGB images obtained from the monocular camera; we can see that they look quite alike. Note that the noisy depth images are used only for training and estimated depth images are used for testing.

2.2. Decision making through D3QN

2.2.1. Q learning algorithm

Over the last decade, Q-learning based reinforcement learning algorithms, especially deep Q-network (DQN) (Mnih et al., 2015), show promising results in various fields. DQN is a type of multi-layered neural networks which produce the action value a in the form of $Q(s, a, \theta)$ while taking the given state s as an input. The variable θ represents a neural network weight parameter. DQN requires two value functions for the update process: one is for the online network and the other is for the target network. The online network is updated at every iteration while the target network is fixed for a certain amount of iterations and then updated. Since the target network is fixed, the online network could undergo stable training. For the training of neural networks, the following loss function is used:

$$L_t(\theta_t) = E_{s,a,r,s'}[(y_t^{DQN} - Q(s_t, a_t; \theta_t))^2]. \tag{1}$$

By taking the derivative of the loss function, the learning parameters are updated. Since the target network is fixed for a certain period, the optimization process becomes more stable. The gradient updates equation is expressed as:

$$\nabla_{\theta_t} L_t(\theta_t) = E_{s,a,r,s'}[(y_t^{DQN} - Q(s, a; \theta_t)\nabla_{\theta_t} Q(s, a; \theta_t))], \tag{2}$$

where ∇_{θ_t} represents the gradient of the given loss function with respect to the learning parameter θ_t. The corresponding target network is expressed as:

$$y_t^{DQN} = r_{t+1} + \gamma \max_{a'} Q(s_{t+1}, a; \theta_t^-), \tag{3}$$

where θ_t^- is the target network weight parameter and γ is the discount factor. By using Eq. (1), DQN performs the following update process.

$$\theta_{t+1} = \theta_t + \alpha(y_t^{DQN} - Q(s_t, a; \theta_t))\nabla_{\theta_t} Q(s_t, a; \theta_t), \tag{4}$$

where α is a learning rate. The update of the online network is composed of two steps: (i) selecting an action on each state and then (ii) evaluating the action on the online network.

In the DQN algorithm, both the action selection and action estimation are done using the same estimator, but updating the network with only one estimator results in overestimation of an action value function, which degrades the performance of DQN algorithms. Another Q learning based method, double deep Q-network (DDQN) (Van Hasselt, Guez, & Silver, 2016) is introduced to solve such overestimation problems. DDQN has the same network architecture with DQN; however, there is a difference in the updating process. The following equation shows the DDQN target network.

$$y_t^{DDQN} = r_{t+1} + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a; \theta_t); \theta_t^-), \tag{5}$$

where DDQN takes an online network parameter θ for an action selection phase. On the contrary, instead of using the online network during the action estimation phase, the target network parameter θ^- is used as an estimator. Note that the equation for update is exactly the same except for the change in the target network equation from y_t^{DQN} to y_t^{DDQN}.

Unlike DQN and DDQN algorithms, the dueling double deep Q-network (D3QN) (Wang et al., 2015) has a different network architecture at the end of the convolutional layer. After passing through the several convolutional layers, D3QN splits the network into two forms: state value function $V(S)$ and advantage function $A(s, a)$. Fig. 2 shows the structure of DQN and D3QN algorithms. The final form of optimal Q function is described as:

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a), \tag{6}$$

with

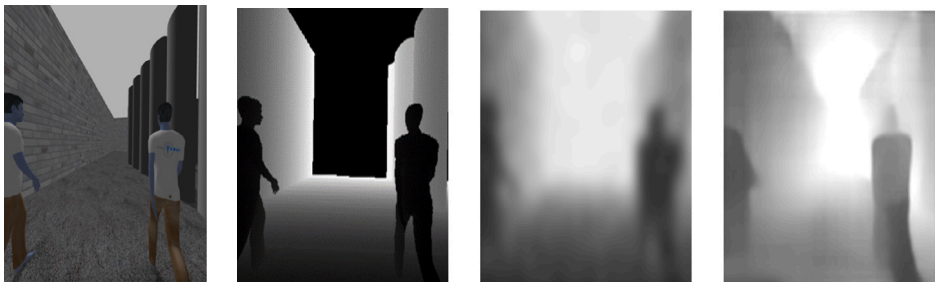
$$V^\pi(s) = E_{a\sim\pi(s)}[Q^\pi(s, a)], \tag{7}$$

$$A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s), \tag{8}$$

and the advantage function satisfies the following condition:

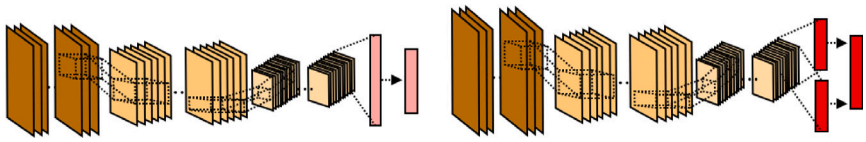
$$E_{a\sim\pi(s)}[A^\pi(s, a)] = 0. \tag{9}$$

D3QN evaluates states twice: one with the advantage function and the other with the state value function. Both state value and advantage functions take part in evaluating the given states, and they ensure better objective evaluation for given states compared with DQN and DDQN. D3QN shows the better performance in 38 problems out of 55 problems (about 70%) (Hessel et al., 2018; Mnih et al., 2016; Wang et al., 2015), compared with DDQN. Thus in this paper, we adopted the D3QN for the OA algorithm.



(a) RGB image (b) True depth (c) Noisy depth (training) (d) Estimated depth (testing)

Fig. 1. RGB image and corresponding various depth images.



(a) DQN (b) D3QN

Fig. 2. Q learning architecture. Left figure shows DQN and right figure shows D3QN. DQN has a single stream, whereas, the D3QN has two streams.

2.2.2. Training network configuration

In this section, the overall concept of the RL-based OA algorithm will be delivered. Fig. 3 shows the block diagram of the proposed OA algorithm. The OA algorithm is composed of two phases: (i) estimation of depth image through FCRN and (ii) decision making for control. As the first step, the obtained RGB images from the monocular camera go into the depth estimation network. The RGB images passes through sequential convolutional layers and finally are changed into the estimated depth image while converting its size from $[304, 228, 3]$ to $[160, 128]$. In the following step, consecutive estimated depth images are then passed to D3QN which determines the control command. Applying successive depth images to the neural network enables the networks to consider time series problems more effectively. Ramezani Dooraki and Lee (2018) analyzed the performance of the OA algorithm depending on the number of used successive images for the neural network, and it turns out that adopting eight sequential depth images into the neural network produces the best performance. Based on this, eight depth images are used for our proposed OA system. The D3QN network has several convolutional layers, and, at the end, it splits into two steams: advantage function and state value function. Adding these two functions generates Q function and this determines the angular and linear velocity compared for the agent. Dividing the action space into linear and angular velocity increases the possible action candidates, e.g., slow turn, fast turn, stop, and thus results in better performance. The linear velocity has three candidates: $[0.8, 0.4, 0]$ m/s. For the angular velocity, the agent can choose five actions: $[\pm 0.5, \pm 0.25, 0]$ rad/s. Thus, the drone can make overall fifteen combination of guided control actions. The zero linear velocity in the action candidates reduces the collision probability while decreasing its speed in complex environments.

2.2.3. Reward function

In designing the RL algorithm, we need to properly define Markov decision process (MDP) to obtain better performance. This requires several components, and among them, a well-designed reward function is the most critical one, as it directly affects the intended actions, namely, obstacle avoidance. In this paper, the reward function is designed to satisfy mainly two conditions. First, our proposed reward function should reduce the inefficient movements of the agent to increase the endurance. Next, the reward function should consider the distance to the closest obstacle from the agent. To this end, three reward functions are devised: velocity reward, depth reward, and collision reward function. The three reward functions are combined for calculation of the total reward function r_{total}.

- Velocity reward function

The ideal movement of the drone should consider the battery efficiency (i.e., endurance), by simply moving straight as much as possible. Besides, the drone should avoid the obstacle with a minimum change in heading angle. To this end, the reward function is designed as:

$$r_{velocity} = v\cos(\omega)dt - c, \tag{10}$$

where v is the linear velocity and ω is the angular velocity of the drone. dt is the frequency of the control command on the virtual simulator and its value is fixed at 5 Hz. c is the offset constant that makes the value of reward function stay in the same range with other reward functions. The drone could maximize its linear velocity and minimize angular velocity by devising the velocity reward function as in Eq. (10). The cosine function in the above equation sets the angular velocity value smaller than one and produces smaller values as angular velocity increases.

- Depth reward function

If there only exists the velocity reward function, the drone tends to flight with high linear velocity for every action. It may increase the velocity reward value, but at the same time, this may also increase the collision probability. Thus, we should prevent the drone from moving too fast near the obstacle. To overcome this side effect, another reward function is devised. The drone receives the distance information from a LiDAR during the training and the information is changed into the relative distance. The definition of the relative distance to the closest obstacle is described as:

$$d_{depth} = \frac{\min(d_1, d_2, \ldots, d_n)}{\sigma}, \tag{11}$$

where d_i is the absolute distance from drone to the ith obstacle. σ is a positive constant for normalization and set as $\sigma = 2.0$ m. The proposed method suggests two separate reward functions depending on the relative distance defined in Eq. (11). The area with a long distance

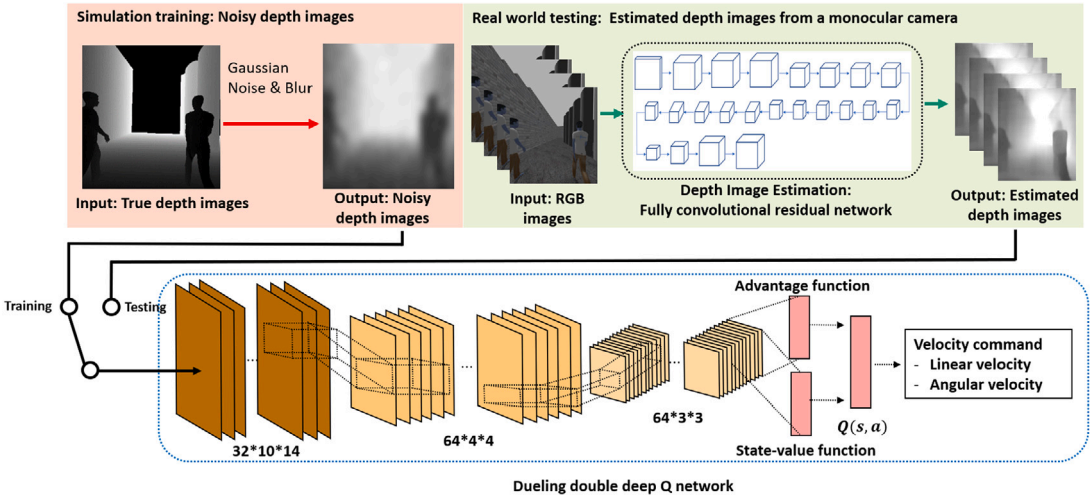


Fig. 3. The proposed reinforcement learning framework. In the training phase, noisy depth images are used as an input of D3QN. In the testing phase, the estimated depth images through depth estimation algorithm are used as an input of D3QN. Best view in color.

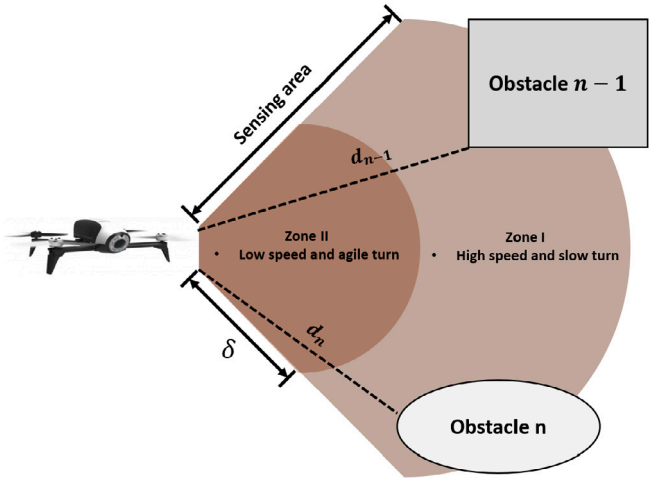


Fig. 4. Illustration of Zone I and II.

from the drone is defined as Zone I and the other with the short distance is defined as Zone II. Fig. 4 illustrates Zone I and II where the colored fan shaped sectors represent the sensing area. Note that δ is used as a constant that distinguishes Zone I and Zone II area.

Zone I: During the training, in case where the relative distance d_{depth} is bigger than the collision threshold δ, the agent receives the reward $r_{depth} = 0.4$. We could give a larger value to r_{depth}, but considering the range of r_{total} value, giving a larger value would reduce the stability of training. δ is used as a constant that divides sensing range of depth camera.

Zone II: To cope with imminent collision, we design the Zone II reward. The Zone II reward directly affects the movement of the drone especially when the drone is about to avoid the obstacle. The Zone II reward is defined as:

$$r_{depth} = \min(0.4,\ d_{depth}\cos(v) + \Delta d_{depth}), \tag{12}$$

$$\Delta d_{depth} = \epsilon\,\mathrm{sgn}(d_{depth}(t) - d_{depth}(t-1)), \tag{13}$$

where ϵ is the offset constant set as 0.35. The depth reward r_{depth} is composed of two terms. One is the velocity control term. It is expressed with multiplication of the relative depth value d_{depth} and cosine of linear velocity v. The other term is the change in the relative depth value. It represents the change in the depth of the closest obstacle from the drone compared to previous time step. The velocity control term functions as a break, decreasing the speed of the drone, as it approaches the obstacle. The change in the relative depth term lets the drone move away from the closest obstacle. The maximum value of depth reward r_{depth} is set as 0.4, which makes the agent receives the reward below than minimum Zone I reward value. As a result, the drone learns to stay in the Zone I area while receiving less reward value than that of Zone II.

In order for the drone to move slowly in the Zone II area to avoid the imminent collision, the velocity reward function value in Eq. (10) is set zero. The drone should have a proper value of δ. If the constant δ is large, there is an increase in unnecessary movements (i.e., jerky motions) of the drone. In contrast, if the value of δ is too small, the drone tends to start avoiding the obstacles only if it reaches the obstacles closely. Due to the effect of velocity reward function, the drone moves as fast as possible until it approaches to the obstacles without experiencing collision. Through various experiences on simulations, we used the value of $\delta = 0.5$ in this study.

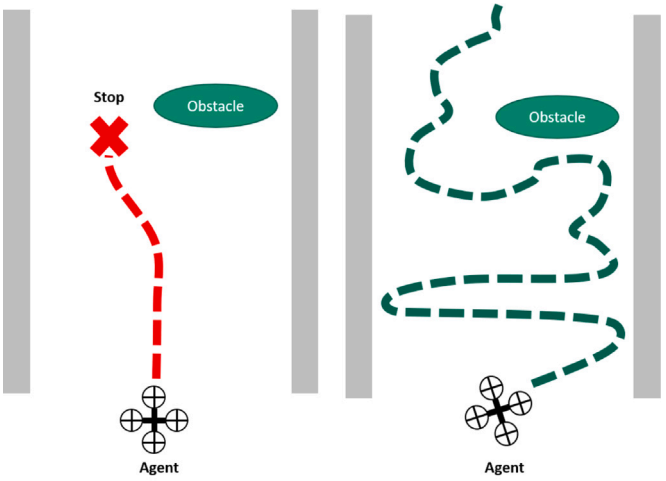
- Collision reward function

Above reward functions may reduce the collision probability in a large scale, however, after all the drone experiences collision during the training. When a collision happens, a proper negative reward must be given to the drone. The collision reward is set as -1, if the relative depth to the closest obstacle d_{depth} is smaller than a certain threshold value. Setting a proper collision threshold value is important as it affects the motion of the UAV. Fig. 5 shows the behavior of UAV with different collision threshold value. If the collision threshold value is too big, the drone cannot pass through the obstacles even though it has enough space (See Fig. 5(a)). On the other hand, if the collision threshold value is too small, the drone tends to move from side to side in a narrow passage, and approaches the obstacle too close (See Fig. 5(b)). For our simulation, the collision threshold value is set to 0.6. This value may be changed in other simulation environments since vehicles and sensors applied may have different specification.

To sum up, the drone learns to move with large linear speed, minimizing angular velocity through the Zone I reward function while avoiding obstacles. If there is no obstacle in the Zone II, the drone will move similarly as it is in Zone I case. If there is any obstacle in the Zone II area, the drone will move away from the obstacle, decelerating the linear speed. The total reward function r_{total} is expressed as:

$$r_{total} = r_{velocity} + r_{depth} + r_{crash}. \tag{14}$$

For the stable training, several factors are considered. First, the reward function r_{total} is ranged in $[-1, 1]$ for the fast convergence of the



(a) Large collision threshold value (b) Small collision threshold value

Fig. 5. Behavior characteristics of the UAV depending on the collision threshold value.

loss function during training. Second, the huber loss function (Huber, 2004) is used, which quickly decreases the loss within $[-1, 1]$, while treating values located at the outside this range as an outliers. The huber loss is optimized with the RMSprop (Ruder, 2016) optimizer. The RMSprop optimizer calculates the gradients using moving exponential average and stabilizes the training. The algorithm 1 explains how to compute total reward function.

Algorithm 1 Reward function during the training process

Initializing the position of the drone
Choose action $\mathbf{a_t} \in \mathbf{A}$ based on given state $\mathbf{s_t} \in \mathbf{S}$
while No Crash **do**
  $r_{velocity} = v\cos(\omega)dt - c$ Eq. (10)
  $d_{depth} = \frac{\min(d_1, d_2, \ldots, d_n)}{\sigma}$ Eq. (11)
  if $d_{depth} \geq \delta$ **then**
    Zone I:
    $r_{depth} = 0.4$
  else
    Zone II:
    $r_{depth} = \min(0.4,\ d_{depth}\cos(v) + \Delta d_{depth})$ Eq. (12)
    $r_{velocity} = 0$
  end if
  Check the collision
  if $\min(d_{depth}) \leq 0.6$ **then**
    $r_{crash} = -1$
    $r_{depth} = 0, r_{velocity} = 0$
  else
    $r_{crash} = 0$
  end if
  $r_{total} = r_{velocity} + r_{depth} + r_{crash}$
end while

3. Designing training and simulation environment

In this section, training environments for simulations and corresponding simulation setup will be described. More detailed information can be found in the following github link: https://github.com/mw9385/Collision-avoidance.

Table 1
Training PC specification.

Training PC environment	
RAM	62.8 GB
Processor	Intel Xeon cpu E5-1650 v4 @ 3.60 Hz * 12
GPU	Geforce RTX 2080Ti * 2ea
OS	Ubuntu 16.04
Workspace	Jupyter notebook
Python library	Tensorflow, keras

3.1. Simulation environment

D3QN is trained using the ROS (Robot operating system) and Gazebo simulator. For the fast training of D3QN, the network is trained under the Gazebo simulation three times faster than the real time. The additional acceleration was not possible since holding the control command frequency at 5 Hz is difficult in a faster situation. Furthermore, a proper batch size setting is also needed to hold the constant command frequency. Table 1 shows training environment where multi-GPU is used and tensorflow and keras are used as the python library for the good performance in Gazebo environments.

3.2. State observation through various sensors

In the simulation for the training, the hector quadrotor (Meyer, Sendobry, Kohlbrecher, Klingauf, & von Stryk, 2012) is used as a training agent. It uses various sensors such as hokuyo LiDAR, Inertial measurement unit (IMU), and ASUS depth camera. The LiDAR and IMU sensors are used for reward function calculation. The LiDAR has the maximum FOV of 270° with a 40 Hz measurement frequency and maximum detection distance of 30 m. It is also used to check the collision status of the drone using the depth information and gives successive rewards for every action. The IMU sensor measures the linear and angular velocity used to compute the reward. The depth camera is used to obtain input states for D3QN in the form of depth images. Note that the LiDAR sensor and the depth camera are only used during the training process and only RGB images from the monocular camera are used for the real flight tests.

3.3. Gazebo training environment

A well-constructed training environment plays an important role in RL training. For that reason, the environment for RL training is composed of complex elements such as texture-free walls, white corridor, boxes, gray cylinders, and a group of people, considering the diversity of obstacle's geometry. DQN, DDQN, and D3QN are trained in the identical training environment depicted in Fig. 6. Once the drone collides with an obstacle, it restarts a new episode at the designated position shown as red dots. As an initial condition, the drone has zero velocity and heading angle, and altitude of 1.5 m. Each episode is automatically terminated, if the drone travels in the training map without collision more than 2000 steps (about 400 s). This termination condition prevents the drone from traveling the same path during the training process ensuring training data are independent and identically distributed (i.e., i.i.d). As a result, the diversity of the training data set increases as training goes on where the overall performance of the algorithm becomes better. After the algorithm fully learns the obstacle avoidance rules, it is tested in new tests maps (i.e., not used in the training). As shown in Fig. 7, test maps have four different environments with numerous obstacles: map (a) contains many obstacles with different sizes; map (b) has a large obstacle right in front of the initial spawning position; map (c) has wide free space but contains long horizontal obstacles and people; and map (d) has comparatively simple obstacles such as a wide texture wall. Among the four maps, map (d) is the most difficult map. By placing a dense group of people all around the map, there is not enough space for OA. To be applied to real experiments, the RL algorithm is trained on additional new training maps as shown in Fig. 8.

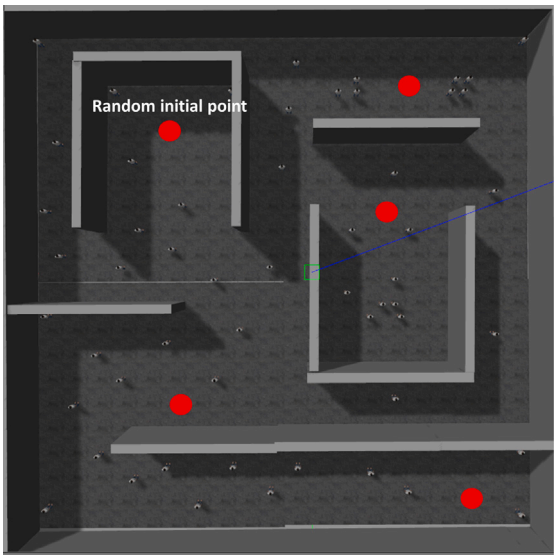


Fig. 6. Designed Gazebo indoor training environment.

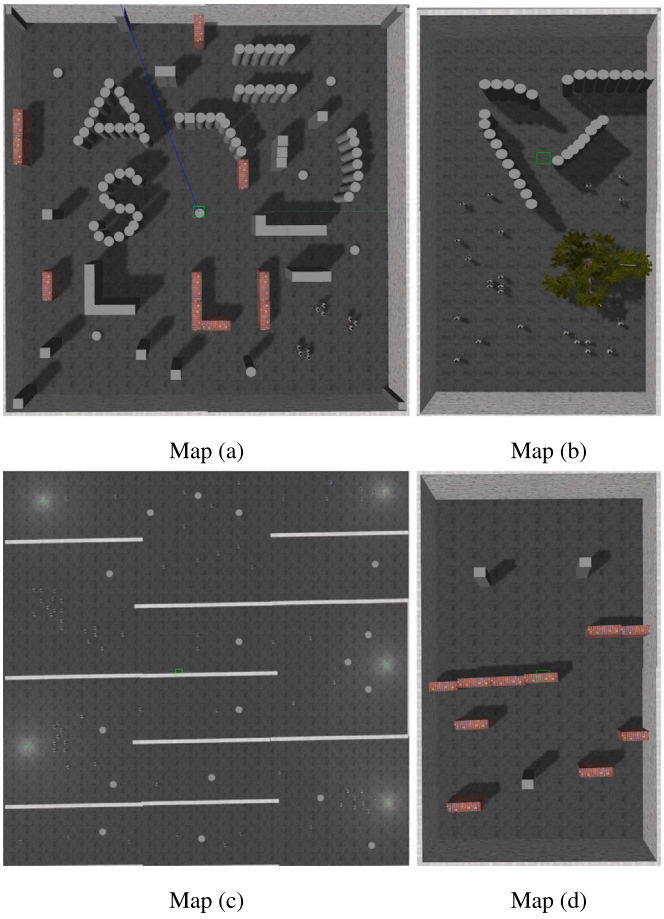


Fig. 7. Test maps for the obstacle avoidance algorithm.

3.4. Network hyperparameter configuration

For a stable training of the RL, a proper hyperparameter setting is of primary importance. Table 2 shows the specific information for hyperparameter values. A minibatch size is designed to prevent the network from overfitting. It extracts a fixed number of data from the total data batch. The total data batch size is called the replay memory size (Mnih et al., 2015). It stores the whole rollouts during the training process. If the overall number of data exceeds the replay batch size, then the network removes the data that come first and stores the latest information at the end of the memory. The agent history length determines the number of successive images for the network input. The target network update frequency means the number of iterations where the target network fixes its weight and bias until the end of the online network update frequency. The epsilon decaying value represents the ratio of exploration to exploitation of the agent during the fixed number of iteration steps. Finally, the number of no action steps represents the number of collected episodes before starting the training for the sake of data diversity.

Table 2
Hyperparameters.

Hyperparameters	Value
Loss function	Huber loss
Minibatch size	16
Replay memory size	50,000
Agent history length	8
Target network update frequency	10,000
Learning rate	0.00025
Epsilon decaying	10,000
Initial epsilon	1.0
Final epsilon	0.01
Optimizer	RMSprop (Ruder, 2016)
Number of no action steps	5

Table 3
Performance comparison with different δ values.

Map	a		b		c	
δ value	TD* (m)	T** (s)	TD (m)	T (s)	TD (m)	T (s)
0 Xie et al. (2017)	136.62	1206.2	59.86	400.91	84.74	729.03
0.5	**150.74**	1387.45	**136.38**	919.57	**140.18**	1259.51
1.5	17.08	1153.0	36.76	1506.74	17.97	960.63

TD*: Traveled distance.
T**: Time.

4. Simulation and experiment results

This section presents the simulation and experiment results of the proposed OA algorithm. For the comparison of the OA algorithms, several scenarios are considered during the simulation. Real flight experiments using D3QN are performed in complex indoor environments.

4.1. Simulation results

4.1.1. Result analysis of the depth reward function on different zones

From now on, we investigate the effect of using different values of δ which separates the sensing area into Zone I and Zone II. For a fair comparison, D3QN is applied for all cases and trained in the environment shown in Fig. 6. As for test environments, the maps in Fig. 7 are used. As the aim of the proposed algorithm is fast and safe obstacle avoidance in complex indoor environments, we set the total flight time and traveled distance as a performance measure. Table 3 shows the performance of OA algorithms with different values of δ. The tests have been conducted 35 times for all cases with random initial positions. For the performance comparison, the algorithm in Xie et al. (2017) which uses $\delta = 0$ is selected as the baseline algorithm since it is similar to our approach in that (i) it uses estimated depth images and the direction command from D3QN and (ii) it uses the same velocity reward function as in our approach; however, it does not use the separate two zones as opposed to our approach. The baseline algorithm shows worse performance than the proposed method. The algorithm with $\delta = 0.5$ shows the best performance among the three

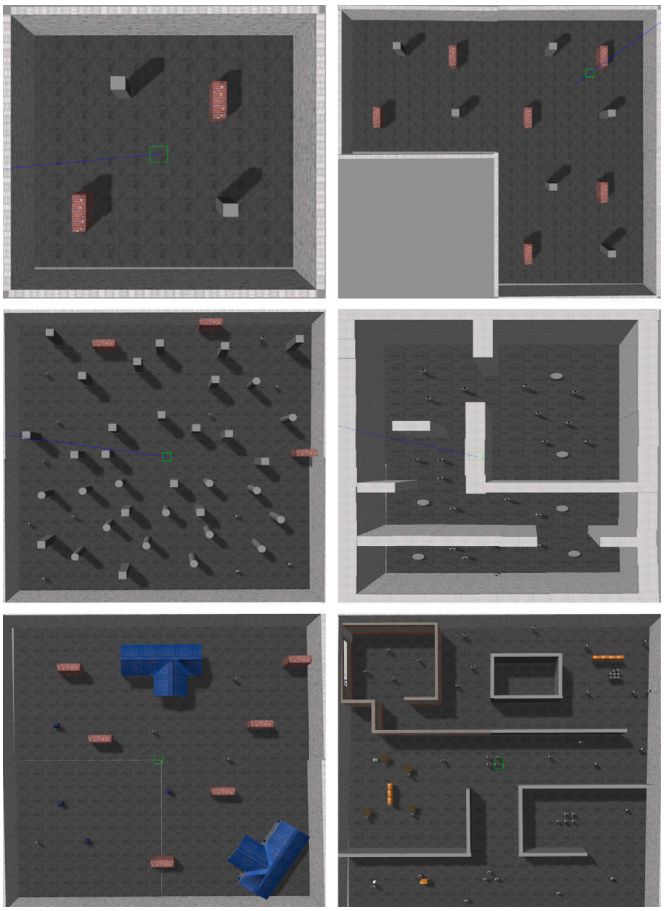


Fig. 8. Additional training environments for real-world implementation.

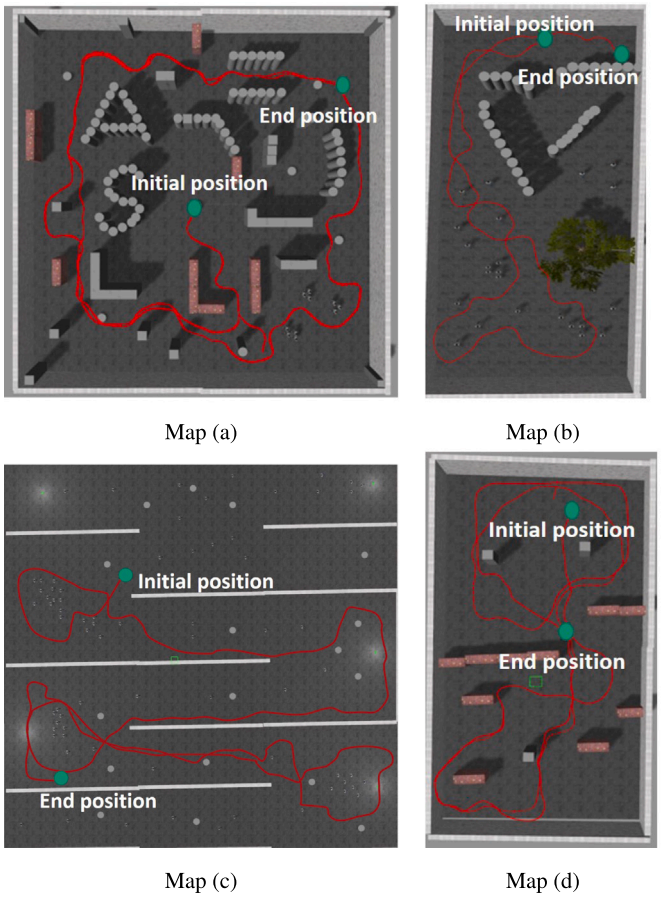


Fig. 9. Sample position history of D3QN after training in the test environments.

cases. The baseline algorithm has no big difference with the proposed method, but shows worse performance than the proposed method in all cases. When the Zone II reward function is used only (i.e., $\delta = 1.5$), the drone moves slowly since it acts as if there is an imminent collision threat, which gives a poor performance. Based on this result, if the zone reward function is used properly, then the proposed algorithm shows the better performance in various environments. The sample simulation result using the proposed D3QN algorithm is shown in Fig. 9. The red paths represent the position of the drone for the each environment during 2000 steps.

4.1.2. Comparison of OA algorithms

The trained RL algorithm is compared with the existing OA algorithm. The existing baseline algorithm is Xie et al. (2017) as mentioned above. For the fair evaluation of the algorithm, four maps in Fig. 7 are used with the same initial position, velocity and orientation with five methods: straight flight, baseline, DQN, DDQN, and D3QN. The simple straight flight is used to represent how complex the environments are. Table 4 shows the averaged traveled distance and flight time in different maps over 35 tests. As expected, the drone with the straight flight method collides with an obstacle within few time steps. The baseline algorithm (Xie et al., 2017) shows the second best performance in the test maps. The DQN algorithm shows long flight time over the four maps, but its travel distance is relatively short, which means the agent travels at a low speed. The D3QN algorithm shows the best performance in most cases. In summary, we verify that adopting the depth reward function to D3QN algorithm enhances the overall performance of the OA algorithm while maintaining its speed fast in complex environments.

4.2. Experimental results

For experiments, the Parrot Bebop 2.0 with the ROS bebop autonomy package was used. The FOV of the Parrot is 90° which is smaller than that of the simulation environment 110°. Although the drone with more limited specification is employed for the real experiment, it successfully navigates through complex environments, which shows the robustness of the proposed algorithm. The communication between a laptop as an off-board computing device and a drone is connected through Wifi and the control command frequency is fixed at 5 Hz. The drone controls its position and velocity through the estimated navigation information using the downward camera and IMU sensor. In order to operate the drone in a safe way, smaller values of velocity actions than those of the simulation are used for the real experiment. Corresponding linear velocity values are $[0.4, 0.2, 0]$ m/s and angular velocity values are $[\pm 0.3, \pm 0.15, 0]$ rad/s. The off-board computing device has the following specification: Intel Core i7-7700 CPU, NVIDIA GeForce GTX 1050 GPU, and 8 GB RAM. On this computing environment, our algorithm could operate at maximum 20 Hz.

The location of experiments is corridors of Ulsan National University of Science and Technology (UNIST) campus buildings. To verify the proposed algorithm on various real environments, we use three scenarios with different obstacles and geometric properties:

- A mid-sized corridor with texture and windows;
- A wide corridor without texture; and
- A narrow corridor without texture and low light intensity.

Fig. 10(a) shows the scenario with a wide corridor with windows and texture free walls. The width of the corridor is about 2.5 m. Fig. 10(b)

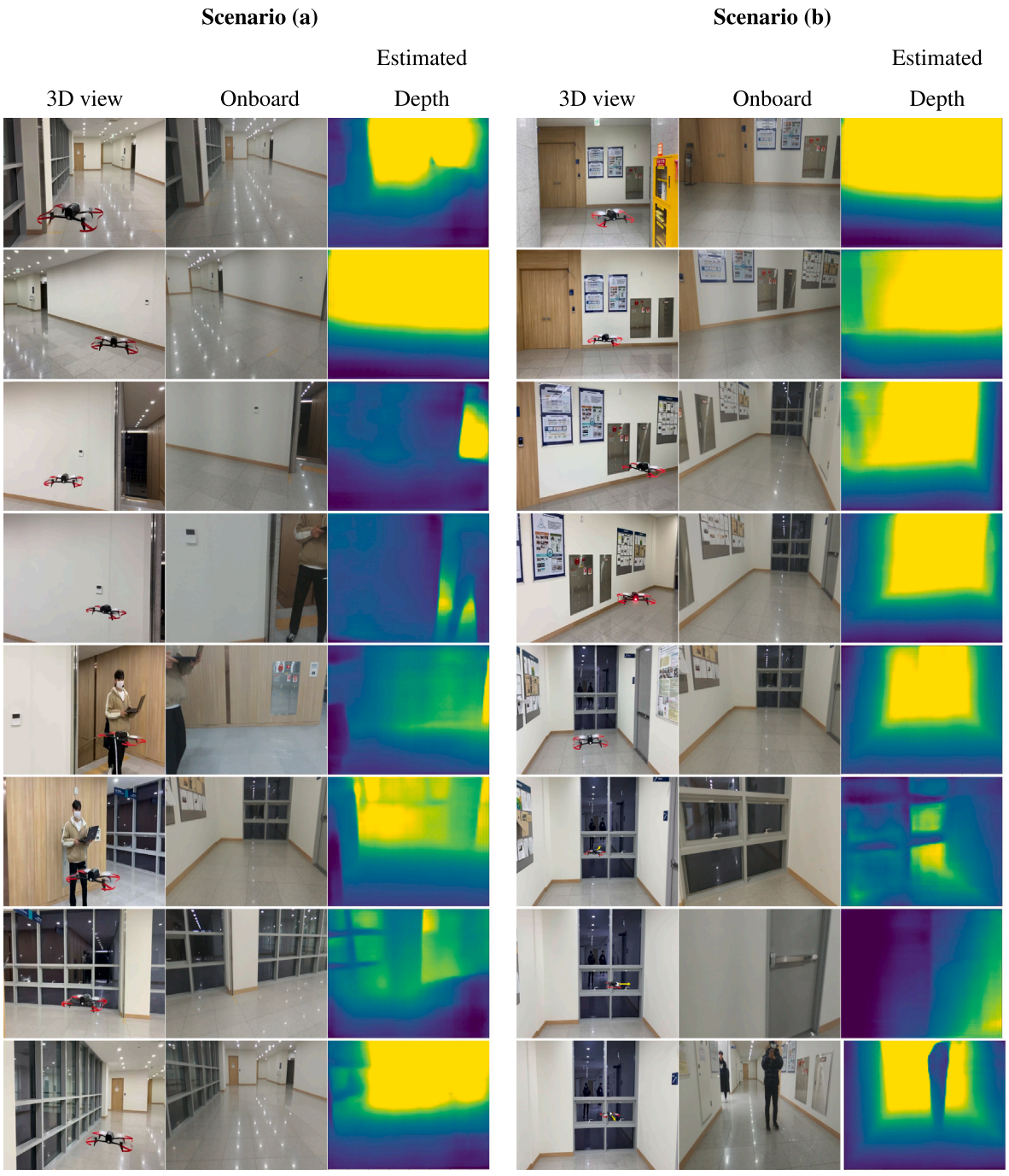


Fig. 10. Images of obstacle avoidance experiment result for two scenarios

Table 4
Obstacle avoidance performance analysis in various test environments.

Map	a		b		c		d	
Algorithm	TD* (m)	T** (s)	TD (m)	T (s)	TD (m)	T (s)	TD (m)	T (s)
Straight	5.18	51.0	9.14	79.45	5.86	51.65	15.52	122.02
Xie et al. (2017)	136.62	1206.2	94.07	715.54	84.74	729.03	59.86	400.91
DQN	28.34	1801.4	23.79	1140.46	18.53	1486.23	29.97	2000.0
DDQN	130.55	1457.57	43.02	455.94	108.69	1482.03	55.78	610.89
D3QN (Proposed)	**150.74**	1387.45	**136.38**	919.57	**140.18**	1259.51	**76.44**	591.85

TD*: Traveled distance.
T**: Time.

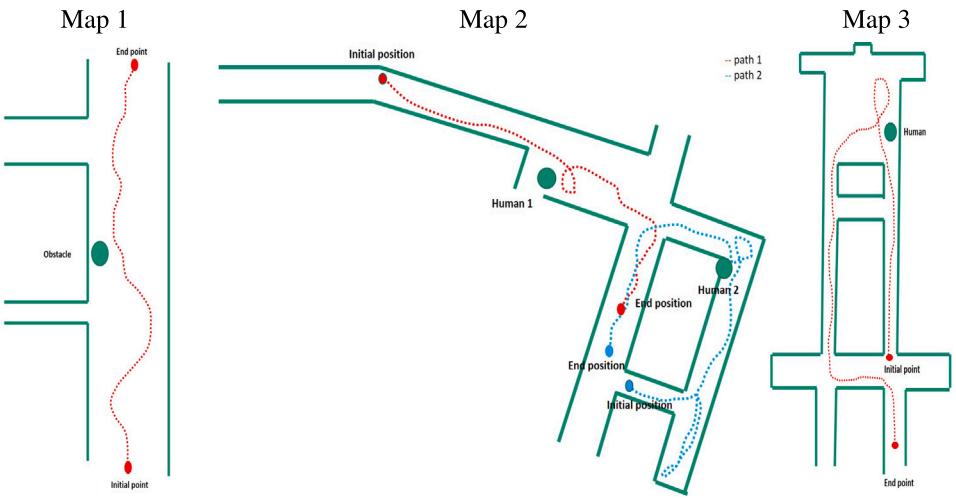


Fig. 11. Indoor experiment environments and UAV trajectories

has more limited space (i.e., width of 1.5 m) and the drone has to face with dead end during the flight. At the beginning of the test in the scenario (a), the drone moves along a narrow corridor and then encounters the wide space. In the middle of the wide space, there exist static and dynamic obstacles (i.e., slowly moving human). Even though the drone is not trained with a dynamic obstacle, the drone successfully avoids it. In the scenario (b), the drone avoids the dead end and corners in more limited space. The supplementary movie clip of the experiments can be found at https://youtu.be/oSQHCsvuE-8.

Fig. 11 shows the recorded position history estimated by optical flow information obtained from the downward camera and IMU sensor. Since this value is the estimated value, there could be some errors; nevertheless, we provide overall trajectories by using the estimated value for the visualization purpose. As shown in Fig. 11, the drone could travel through wide or narrow indoor environments successfully.

5. Conclusions and future work

This paper has proposed the OA algorithm for a small drone using only a monocular camera through deep reinforcement learning (DRL). The estimated depth images for the DRL inputs are obtained by the convolutional neural network. Adopting estimated depth images for the neural network input rather than raw RGB images, we overcame the drawbacks of DRL by reducing the dissimilarity between virtual simulation and real environments. Furthermore, in order to increase the performance, we proposed specific reward functions using the relative depth to the closest obstacle, which results in better performance than that of the existing DRL algorithm, especially in narrow indoor environments. We also show that applying the D3QN algorithm is better than the existing Q functions: DQN and DDQN. As for the future work, the development of OA algorithms combined with navigation decision making (i.e., path planning with a goal point) will be considered. Besides, we plan to develop an image-based collision probability computation algorithm for better OA performance extend OA algorithm to more general path/trajectory planning such as going towards the goal position while avoiding obstacles. More comparison studies with the state-of-the-art algorithms in various real-world scenarios will also be followed.

CRediT authorship contribution statement

Minwoo Kim: Conceptualization, Methodology, Investigation, Writing - original draft. **Jongyun Kim:** Writing - review & editing. **Minjae Jung:** Writing - review & editing, Software. **Hyondong Oh:** Project administration, Supervision, Investigation, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was supported by Theater Defense Research Center funded by Defense Acquisition Program Administration under Grant UD200043CD, Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (2020R1A6A1A03040570), Unmanned Vehicles Core Technology Research and Development Program through the National Research Foundation of Korea (NRF), Unmanned Vehicle Advanced Research Center (UVARC) funded by the Ministry of Science and ICT, the Republic of Korea (2020M3C1C1A01082375) and the Institute of Information and Communications Technology Planning and Evaluation (IITP) Grant (No. 2017-0-00067, Development of ICT Core Technologies for Safe Unmanned Vehicles) funded by the Ministry of Science and ICT (MSIT), Republic of Korea.

References

Ahmed, M. U., Brickman, S., Dengg, A., Fasth, N., Mihajlovic, M., & Norman, J. (2019). A machine learning approach to classify pedestrians' events based on IMU and GPS. *International Journal for Artificial Intelligence*, *17*, 154–167.

Ahn, K. -H., & Song, J. -B. (2020). Image preprocessing-based generalization and transfer of learning for grasping in cluttered environments. *International Journal of Control, Automation and Systems*, *18*, 2306–2314.

Alvarez, H., Paz, L. M., Sturm, J., & Cremers, D. (2016). Collision avoidance for quadrotors with a monocular camera. In *Experimental robotics* (pp. 195–209). Springer.

Aulinas, J., Petillot, Y. R., Salvi, J., & Lladó, X. (2008). The SLAM problem: A survey. *CCIA*, *184*, 363–371.

Back, S., Cho, G., Oh, J., Tran, X. -T., & Oh, H. (2020). Autonomous UAV trail navigation with obstacle avoidance using deep neural networks. *Journal of Intelligent and Robotic Systems*, *100*, 1195–1211.

Chakravarty, P., Kelchtermans, K., Roussel, T., Wellens, S., Tuytelaars, T., & Van Eycken, L. (2017). CNN-Based single image obstacle avoidance on a quadrotor. In *International conference on robotics and automation* (pp. 6369–6374). IEEE.

Cho, G., Kim, J., & Oh, H. (2019). Vision-based obstacle avoidance strategies for MAVs using optical flows in 3-D textured environments. *Sensors*, *19*, 2523.

Eresen, A., İmamoğlu, N., & Efe, M. Ö. (2012). Autonomous quadrotor flight with vision-based obstacle avoidance in virtual environment. *Expert Systems with Applications*, *39*, 894–905.

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, *32*, 1231–1237.

Green, W. E., & Oh, P. Y. (2008). Optic flow-based collision avoidance. *Robotics & Automation Magazine*, *15*, 96–103.

He, L., Aouf, N., Whidborne, J. F., & Song, B. (2020). Integrated moment-based LGMD and deep reinforcement learning for UAV obstacle avoidance. In *2020 IEEE international conference on robotics and automation* (pp. 7491–7497). IEEE.

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Computer vision and pattern recognition* (pp. 770–778).

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., et al. (2018). Rainbow: combining improvements in deep reinforcement learning. In *Conference on artificial intelligence*.

Huber, P. J. (2004). *Robust statistics, volume 523*. John Wiley & Sons.

Jakobi, N., Husbands, P., & Harvey, I. (1995). Noise and the reality gap: The use of simulation in evolutionary robotics. In *European conference on artificial life* (pp. 704–720). Springer.

Kang, K., Belkhale, S., Kahn, G., Abbeel, P., & Levine, S. (2019). Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *International conference on robotics and automation* (pp. 6008–6014). IEEE.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, *25*, 1097–1105.

Laina, I., Rupprecht, C., Belagiannis, V., Tombari, F., & Navab, N. (2016). Deeper depth prediction with fully convolutional residual networks. In *International conference on 3D vision* (pp. 239–248). IEEE.

Loquercio, A., Kaufmann, E., Ranftl, R., Müller, M., Koltun, V., & Scaramuzza, D. (2021). Learning high-ppeed flight in the wild. *Science Robotics*, *6*, eabg5810.

Loquercio, A., Maqueda, A. I., Del-Blanco, C. R., & Scaramuzza, D. (2018). Dronet: Learning to fly by driving. *Robotics and Automation Letters*, *3*, 1088–1095.

Meyer, J., Sendobry, A., Kohlbrecher, S., Klingauf, U., & von Stryk, O. (2012). Comprehensive simulation of quadrotor UAVs using ROS and gazebo. In *Simulation, modeling and programming for autonomous robots (SIMPAR)*. URL: https://github.com/tu-darmstadt-ros-pkg/hector_quadrotor.git.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., et al. (2016). Asynchronous methods for deep reinforcement learning. In *International conference on machine learning* (pp. 1928–1937).

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Mur-Artal, R., Montiel, J. M. M., & Tardos, J. D. (2015). ORB-SLAM: A versatile and accurate monocular SLAM system. *Transactions on Robotics*, *31*, 1147–1163.

Park, B., & Oh, H. (2020). Vision-based obstacle avoidance for UAVs via imitation learning with sequential neural networks. *International Journal of Aeronautical and Space Sciences*, 1–12.

Pozna, C., Troester, F., Precup, R. -E., Tar, J. K., & Preitl, S. (2009). On the design of an obstacle avoiding trajectory: Method and simulation. *Mathematics and Computers in Simulation*, *79*, 2211–2226.

Ramezani Dooraki, A., & Lee, D. -J. (2018). An end-to-end deep reinforcement learning-based intelligent agent capable of autonomous exploration in unknown environments. *Sensors*, *18*, 3575.

Roghair, J., Ko, K., Asli, A. E. N., & Jannesari, A. (2021). A vision based deep reinforcement learning algorithm for UAV obstacle avoidance. arXiv preprint arXiv:2103.06403.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747.

Sadeghi, F., & Levine, S. (2016). Cad2rl: Real single-image flight without a single real image. arXiv preprint arXiv:1611.04201.

Singla, A., Padakandla, S., & Bhatnagar, S. (2019). Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge. *Transactions on Intelligent Transportation Systems*.

Van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In *AAAI conference on artificial intelligence*.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., & De Freitas, N. (2015). Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.

Wu, K., Abolfazli Esfahani, M., Yuan, S., & Wang, H. (2018). Learn to steer through deep reinforcement learning. *Sensors*, *18*, 3650.

Xie, L., Wang, S., Markham, A., & Trigoni, N. (2017). Towards monocular vision-based obstacle avoidance through deep reinforcement learning. arXiv preprint arXiv:1706.09829.

Yang, S., Konam, S., Ma, C., Rosenthal, S., Veloso, M., & Scherer, S. (2017). Obstacle avoidance through deep networks based intermediate perception. arXiv preprint arXiv:1704.08759.