applied sciences

MDPI

Article

# Obstacle Avoidance Drone by Deep Reinforcement Learning and Its Racing with Human Pilot

**Sang-Yun Shin, Yong-Won Kang and Yong-Guk Kim ***

Department of Computer Engineering, Sejong University, 209, Neungdong-ro, Gwangjin-gu, Seoul 05006, Korea; kimshin812@sju.ac.kr (S.-Y.S.); kyw2539@sju.ac.kr (Y.-W.K.)

* Correspondence: ykim@sejong.ac.kr; Tel.: +82-02-3408-3759

Received: 28 October 2019; Accepted: 6 December 2019; Published: 17 December 2019


**Abstract:** Drones with obstacle avoidance capabilities have attracted much attention from researchers recently. They typically adopt either supervised learning or reinforcement learning (RL) for training their networks. The drawback of supervised learning is that labeling of the massive dataset is laborious and time-consuming, whereas RL aims to overcome such a problem by letting an agent learn with the data from its environment. The present study aims to utilize diverse RL within two categories: (1) discrete action space and (2) continuous action space. The former has the advantage in optimization for vision datasets, but such actions can lead to unnatural behavior. For the latter, we propose a U-net based segmentation model with an actor-critic network. Performance is compared between these RL algorithms with three different environments such as the woodland, block world, and the arena world, as well as racing with human pilots. Results suggest that our best continuous algorithm easily outperformed the discrete ones and yet was similar to an expert pilot.

**Keywords:** drone; deep Q-learning; U-net; actor-critic network; airsim; racing between human and algorithm

## 1. Introduction

The drone is an unmanned aerial vehicle that has been around for a long time, and yet it has become a major research field recently. It seems that the recent success of the drone may come from the stable control of rotors and a bird's eye view provided by a camera installed in front of it. With the growing number of possible applications for the drone, i.e., disaster management [1] and agriculture [2,3], the demand for expert drone pilots is growing up nowadays. To become an expert level drone pilot, one has to spend extended time training. Dexterity in maneuvering a fast-flying drone has become an essential asset for professional drone racers, as a survey indicates that drone racing is one of the fastest growing e-sports, among many others [4]. Indeed, one of the major research fields for the drone has been autonomous drone navigation. One of the critical requirements for the successful application of drones is the ability to navigate through buildings and trees. To achieve this, obstacle avoidance capability is essential in particular. Controlling a drone with such dexterity involves solving many challenges in perception as well as in action using lightweight and yet high performing sensors. Among many, simultaneous localization and mapping (SLAM) has been a major representative approach to solve such challenges by utilizing the stereo camera, Lidar, and other sensors. Although SLAM can cover a range of challenges with the localization problem [5], it also shows some limitations, for example, when the target area has visually changed. Recently, an alternative by utilizing deep learning has arisen, as its ability to deal with vision data is promising.

These approaches are mainly divided into two machine learning paradigms: (1) supervised/unsupervised learning and (2) reinforcement learning (RL). In supervised learning, one has to gather a large set of data for a specific environment where the drone will operate before the training. This scheme

could be effective when the size of the dataset is large enough with high-quality labels. However, collecting massive amounts of data whenever the given environment changes could become a problem. Moreover, labeling all these data not only imposes a time-consuming job but also makes it hard to guarantee whether or not hand-crafted labeling for the massive number of data is optimal for the task. Due to this issue, unsupervised learning-based methods are adopted for the automatic labeling of the dataset in several studies.

On the other hand, the RL based approach aims to deal with this issue by letting the drone-agent itself learn a simulated environment in a trial and error manner. In RL based approaches, the agent tries to learn in a virtual environment that is similar to the real environment and then the trained model is transferred to the real drone for testing.

In this study, we experiment with obstacle avoidance drones within two control spaces: one is discrete space and the other is continuous space. For the former, Deep Q-Network (DQN) [6] will be adopted for controlling the drone while utilizing two visual inputs such as RGB and depth map from the simulated environment constructed with Airsim [7]. There are four deep RL algorithms such as DQN, Double DQN [8], Dueling DQN [9], and Double Dueling DQN (DD-DQN). The present study aims to test the possibility of whether a drone can be trained using deep RL algorithms to find a 3D path through various 3D obstacles within the Airsim environment. The first one is the woodland, called the Modular package, in which trees are randomly distributed along the pathway. The second one is a custom-made block world in which differently shaped 3D objects, such as cone, cube, and sphere, are distributed, and two wall-style obstacles are installed along the potential paths as shown in Figure 1. The third is an arena world wherein the drone needs to navigate the various obstacles used for the block world. However, it requires a higher skill set than the block world, since the drone flies with a curved trajectory using a lot of yaw control.

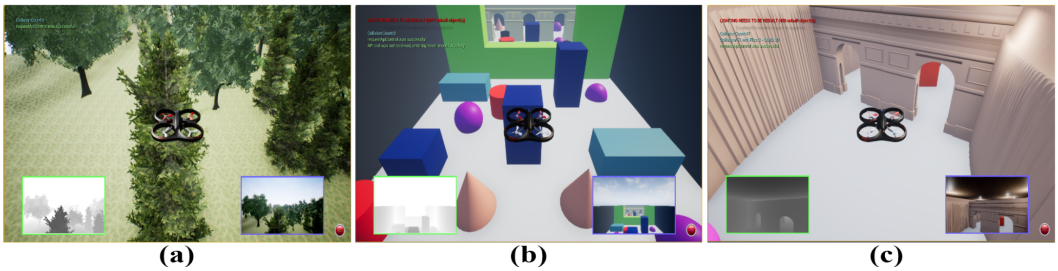


**Figure 1.** Screen shots of the woodland (**a**), the block world (**b**), and the arena world (**c**), respectively. Two insets in each shot indicate the depth map (left) and RGB input (right), respectively. As the space between blocks in the block world is narrower than that between trees in the woodland, the former is more challenging than the latter. In the arena, since the obstacles are arranged along a circular track, the drone requires a delicate roll, pitch, and yaw control.

Given that restriction of the action space into the limited number of discrete actions can lead to unnatural control of the drone, our second approach is to adopt an actor-critic network for controlling the drone in continuous space. To deal with RGB (Red Green Blue) input, U-net based segmentation model is used. This process can be understood as finding a way to go within the scene because this segmentation model is combined with the actor-critic network working in the continuous action space. The promising candidates are policy gradient algorithms such as Trust-Region Policy Optimization (TRPO) [10], Proximal Policy Optimization (PPO) [11], and Actor-Critic using the Kronecker-Factored Trust Region (ACKTR) [11] that represents the actor-critic networks. Our proposed system is to combine a U-net based segmentation model with a policy gradient algorithm with a reward-driven manner. Our contributions are:

1. It is shown that the drone agents can be trained using several deep RL algorithms in discrete and continuous spaces using three different environments such as woodland, block world, an arena world.

2. Human pilots can make a drone race against an algorithm utilizing the hardware-in-the-loop (HITL) feature, and performances between humans and algorithms are evaluated. Result suggests that our best algorithm shows similar performance to that of an expert pilot in three environments.
3. It is found that DD-DQN outperformed other deep RL algorithms for discrete action space, and the performance of the algorithm was the best when both RGB and depth maps are given to the agent, rather than when only one of the sensor signals is provided.
4. So far, U-net based segmentation model has been trained by the supervised learning paradigm where labels are provided by laborious manual labeling. In the proposed system, a label map is generated via the critic network using input provided by a simulated environment.

## 2. Related Work

Recently, deep neural network-based methods have been proposed to enhance the ability to control the drone using computer vision, such as collision avoidance, navigation, etc. [12]. Among many, supervised learning and reinforcement learning have been successfully adopted for the control of the drone, whereas unsupervised learning has been studied for the action recognition and assisting the labeling of the dataset.

### 2.1. Drone Navigation with Supervised Learning

Supervised learning can be promising with a large amount of labeled data. Almost all of the studies on applying supervised learning are based on convolutional neural network (CNN), so that their model can effectively extract features within vision inputs from various environments. It has been shown that a drone can be trained to navigate indoor, where the surrounding area is comparably small [13]. By focusing more on negative data, such as crashing and collision of a drone, it has been shown that a network can effectively avoid collision [14]. A recent study shows that a drone navigates through the corridors within a building using two well-trained CNNs: one is for depth map and the other for RGB input [15]. It is also shown that a well-trained CNN has such a small footprint that it is possible to embed on a nano quadrotor for controlling its flying trajectory [16]. The other possible way for the drone navigation is to combine a CNN with a long short-term memory(LSTM) for training within a Gazebo [17] environment, consisting of a block, a wall, and overhang in a room [18]. More recently, by utilizing a simulation environment during training, a CNN-based model for real-time navigation in a drone-racing track is proposed using micro UAV [19]. This has shown that a deep neural network not only controls a drone in real-time with vision input but also a network trained in a virtual environment can effectively maneuver a drone in the real-world. Instead of training a network in simulation first, many studies made use of data from the real-world directly to train networks. By using vision data collected from forests, a network drives a drone for forestry purposes [2]. Similarly, by using a large amount of data collected in an urban area, it has been shown that a drone can navigate robustly by avoiding obstacles appearing in a city [20]. These improvements in controlling the drone using supervised learning lead to broad applications for the society along with technologies such as the Internet of Things (IoT) [21,22]. Even though the studies mentioned above have successfully applied supervised learning to meet the needs of diverse applications, manual labeling remains a burden for researchers. Moreover, as the data is highly oriented towards a specific environment, it is necessary to go through the entire process again whenever there is a need to apply a model to a different environment. On the other hand, research based on RL try to overcome such an issue.

### 2.2. Unsupervised Learning-Based Methods for Drone Applications

Unsupervised learning-based methods have been utilized for the drone applications, mainly for the labeling. By using algorithms such as simple linear iterative clustering (SLIC), it has been shown that unsupervised-based methods can help label the data for the training of supervised-based model in an agricultural application, leading to the reduction of human effort [23]. Also, by interpreting a depth estimation problem as a reconstruction problem, a study has shown that unsupervised learning models

can successfully estimate a depth map from a monocular RGB image taken by a drone [24]. Based on a reconstruction error of an unsupervised generative model, it has been shown that a robot can decide whether to go or not [25]. Furthermore, it has been shown that a combination of unsupervised-based methods could learn features for anomaly detection in pictures taken by a drone [26].

### 2.3. Drone Navigation with Reinforcement Learning

In RL, an agent is to be trained on how to navigate through the obstacles by making trials and errors. Also, yet, this could be advantageous because once the training environment is ready, then the agent learns by itself. For example, it is shown that a drone and Radio Control(RC) car can be trained to predict uncertainty or collisions by utilizing model-based RL [27]. Here, a drone and RC car had learned by making trials with a low speed so that the hardware was not damaged. This indicates that a model can be trained directly in the real environment with a specific restriction. However, as RL usually requires a lot of trial and error for the learning, having some restrictions on the environment inevitably slows down the training speed. Such characteristics of RL approaches lead to making use of a virtual environment, where the agent can safely make trials and errors quickly without concerning the actual hardware [28,29]. By using sensory data from the accelerometer, it is shown that the RL model can plan swing-free trajectories while carrying suspended load [30]. Similarly, by applying RL, a substantial amount of practical applications have been proposed in controlling the attitude [31], navigating from an arbitrary departure place to destinations [32], enhancing the efficiency of a network of cellular-connected drones in the 5G era [33]. However, it has not been used widely with the visual input, mainly because RL usually suffers from high dimensional data, especially for the similar succeeding frames for continuous actions. One alternative solution is to restrict the continuous action space into a discrete one so that a model can be trained with RL, such as the Deep Q Network [6], known to learn well based on the image. For example, it was shown that a drone can fly to reach the goal in environments by making discrete actions [34]. However, restricting the action spaces could lead to unnatural behaviors.

### 2.4. Robot Navigation using Segmentation Map

Recently, there are several studies where the segmentation model is adopted to extract simple and handy features for robot navigation. For example, a wheeled robot navigating the outdoor street has been trained using the CNN-based segmentation model [35]. Here, the navigation direction is controlled by a fuzzy logic that receives a segmented image from the RGB input. Another work has shown that a semantic segmentation model trained in a virtual environment can minimize the gap between the real and virtual environment [36]. In this study, the segmentation model plays an essential role in visually guiding a robot where to go, and an RL agent trained in the simulation can be transferred to the real environment to control a car. It has also been shown that the segmentation model can learn from the simulated image jointly with the real image by combining a recurrent neural network (RNN) so that the robot in the real environment can use the model directly without having suffered from the gap between real and simulation [37]. Besides, a study shows that a depth map generated from an RGB image using a CNN-based model can be used for training a network controlling a drone [38] by utilizing supervised learning. With this model, it was shown that a network trained in a simulation environment could navigate in the real world during testing.

However, none of the studies have applied the model on the drone using reinforcement learning where it does not require manual labeling of the data. In this study, for the continuous control, we have adopted segmentation model to simplify the information exhibited in the raw image from the first person view (FPV) of the drone, so that an RL model can learn to navigate in the continuous action space, which has been considered very challenging due to the curse of dimension. Moreover, designing the reward function is replaced by taking advantage of mutual interaction between the segmentation network and actor-critic networks.

## 3. Deep Reinforcement Learning and U-Net Segmentation Model

In a problem of a sequential decision making, an agent interacts with an environment $\epsilon$ over discrete time steps [39]. For example, in the Atari domain, which is widely used to measure the performances of various reinforcement learning algorithms, an agent observes frames from a video at a time step $t$.

$$s_t = (x_{t-p+1,\ldots,x_t} \in S_t,\ a_t \in A = 1, \ldots, |A|) \tag{1}$$

Equation (1) shows the ingredients of a state consisting of several frames from a video at time step t: here $x$ is the frame, t is a time step, and A is the number of actions, respectively. $p$ stands for a number of previous frames to observe for making action $a_t$. Each s and a are the states that the agent observes an action that the agent takes. The agent then chooses which action to take from a set and receives a reward signal $r$. The goal of the agent is, of course, to maximize the cumulative reward $R$ for an episode,

$$R = \sum_{\tau=t}^{\infty} \gamma^{\tau-t} r_\tau. \tag{2}$$

Equation (2) shows the reward along with the time steps, where $r_\tau$ is a reward at time step t and $\gamma[0, 1]$ is the discount factor that makes the agent deal with a trade-off between immediate and future rewards [40]. In terms of action space, learning to maximize the reward in reinforcement learning mainly diverges into two schemes: (1) RL in discrete action space; (2) RL in continuous action space.

### 3.1. Learning in Discrete Action Space

In discrete action space, an agent chooses to act according to a policy $\pi$, which usually has a form of greedy learning. Here, $\pi$ is equal to a state-value function $V$, since $\pi$ can be seen to learn by greedily choosing the best action given a state. One of the popular forms for estimating the value is $Q$ function. The connection between $Q$ and $V$ can be expressed as

$$\begin{array}{l} Q^\pi(s,a) = E[R|s_t = s, a_t = a, \pi] \\ V^\pi(s) = E_{a\sim\pi(s)}[Q^\pi(s,a)] \end{array}, \tag{3}$$

where $a$ is a chosen action given a state $s$. Then, by defining the optimal $Q$ function as

$$Q^*(s_t, a_t) = max_\pi Q^\pi(s_t, a_t), \tag{4}$$

$$a = argmax_{a_{t+1}\in A} Q^*(s_{t+1}, a_{t+1}), \tag{5}$$

and

$$V^*(s_t) = max_a Q^*(s_t, a_t), \tag{6}$$

Optimal $Q$ function satisfies the Bellman equation:

$$Q^*(s_t, a_t) = E_{s_{t+1}}[r + \gamma max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})[s_t, a_t]. \tag{7}$$

For high dimensional data, i.e., image, the Deep Q Network (DQN) can effectively learn to solve a given task [6]. Loss function for optimizing the DQN:

$$Loss = ||y_t^{DQN} - Q(s_t, a_t; \theta)||, \tag{8}$$

$$Y_t^{DQN} \equiv r_{t+1} + \gamma max Q(S_{t+1}, a_{t+1}, \theta), \tag{9}$$

where $\theta$ is the parameters of a network for a current optimization step. However, it had been shown that using the on-line parameter $\theta$ for learning could cause unstable learning in vanilla DQN. To solve this issue, Double DQN [6] has been proposed. In Double DQN, a target network is copied for every

pre-defined time step and is used to estimate the target value in the optimization step. Therefore, the target value $Y$ in time step $t$ in Double DQN is as follows:

$$Y_t^{DoubleDQN} \equiv r_{t+1} + \gamma maxQ(S_{t+1}, a_{t+1}, \theta^-), \tag{10}$$

where $\theta^-$ stands for the fixed parameters of the target network,

$$e_t = (e_t, a_t, r_t, s_{t+1}), \tag{11}$$

$$D = e_1, e_2, ..., e_t, \tag{12}$$

To delete the correlation of every experience accumulated in the buffer, which causes significant deterioration of the stability of the network, DQN utilizes a technique called experience replay. Equations (11) and (12) show how experiences are accumulated in the buffer, where $s_t$ is a state, $a_t$ is an action, and $r_t$ is a reward. With experience replay, the network uses randomly sampled mini-batches of experiences from D instead of using the data that is accumulated sequentially. An improvement of DQN, called Double DQN [8] is proposed. To avoid the problem of over-optimistic estimation of $Q$ value, Double DQN uses different values to select and evaluate an action while DQN uses the same values for selecting and evaluating values:

$$Y_t^{Double} = r + \gamma Q(s_{t+1}, argmaxQ(s_{t+1}, a_{t+1}, ; \theta); \theta^-). \tag{13}$$

Instead of using different values to select and evaluate the action, Dueling Deep Q-network (Dueling DQN) [41] separates networks into a value network and an advantage network. The value network is used to estimate the quality of the state while the advantage network is used to estimate the qualities of each action,

$$\begin{aligned} Q(s_t, a_t; \theta, \alpha, \beta) = V(s_t; \theta, \beta) + A(s_t, a_t, \theta, \alpha) \\ -\frac{1}{A}\sum_{a^{t+1}} A(s_t, a_{t+1}; \theta, \alpha) \end{aligned}. \tag{14}$$

Among several equations to estimate $Q(s_t, a_t; \theta, \alpha, \beta)$ in Dueling DQN, we adopt the Equation (14) as it is known to stabilize the optimization process.

Given that there are two ways to improve the performance of DQN in previous sections, we suggest in this paper that there is a better way for an agent to learn by combining Double DQN with Dueling DQN, called it Double Dueling DQN (DD-DQN). In our experiment, we use Equation (14) to calculate $Q$ value as it is known to be robust for removing correlation. By combining Equation (14) with Double DQN, we define the target value of DD-DQN as:

$$\begin{aligned} Y_t^{DoubleDuelQ} = r_{t+1} + \gamma Q(s_{t+1}, \\ argmaxQ(s_{t+1}, a_t, ; \theta, \alpha, \beta); \theta^-, \alpha^-, \beta^-) \end{aligned}', \tag{15}$$

where $\theta$ stands for the parameter of convolutional layers, while $\alpha$ and $\beta$ are parameters of the advantage network and the value network, respectively. By following Equation (15), a loss function for optimization is defined as

$$Loss = ||Y_t^{DoubleDuelQ} - Q(s_t, a_t; \theta, \alpha, \beta)||. \tag{16}$$

The flow diagram is shown in Figure 2, where the input consists of RGB and depth map.

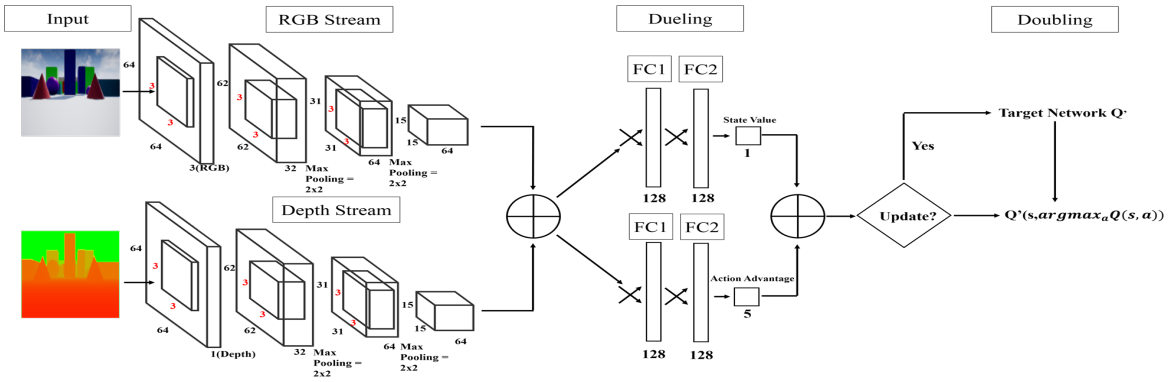


**Figure 2.** A schematic diagram of the present system with the Double Dueling Deep Q-Network (DD-DQN). Both RGB and depth maps are fed into two CNNs as input, and each convolutional neural network (CNN) has three convolutional layers. Two outputs from each CNN are concatenated and fed into fully connected layers for a dueling operation, which produces state value and action advantages. Here, state value concerns the given state itself for 2 input images, whereas action advantages consider advantages for each action. Final values for each action is then produced by merging state value and action advantages. The network is updated every predefined step, following Q doubling optimization policy.

### *3.2. Learning in Continuous Action Space*

For the learning in continuous action space, there are several model-free policy gradient algorithms available such as Trust Region Policy Optimization (TRPO) [10], Deep Deterministic Policy Gradients (DDPG) [42], Proximal Policy Optimization (PPO) [11], and Actor-Critic using Kronecker-Factored Trust Region (ACKTR) [43]. It has been shown that they are promising candidates in the continuous control MuJoCo [44] domain from the OpenAI Gym [45]. They work to obtain the maximum expected rewards by estimating the gradient:

$$g := \nabla_\theta E[\sum_{t=0}^{\infty} r_t]. \tag{17}$$

In general, estimation of policy gradient using action $a$ and state $s$ becomes

$$g = E[\sum_{t=0}^{\infty} \Psi \nabla_\theta log \pi_\theta(a_t|s_t)], \tag{18}$$

where $\pi$ is a policy(actor) network, and $\Psi$ is a critic network. $\Psi$ can be state-value, $Q$ value, or advantage, depending on the algorithm.

TRPO and PPO use constraints and advantage estimation to perform this update by reformulating the optimization problem as

$$max_\theta E_t[\frac{\pi\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} A_t(s_t, a_t)]. \tag{19}$$

Here, $A_t$ is the generalized advantage function [46]. TRPO uses conjugate gradient descent as the optimization method with a KL constraint:

$$E_t[KL[\pi_{\theta_{old}}(a|s_t), \pi_\theta(a|s_t)]] \leq \delta. \tag{20}$$

PPO reformulates the constraint as a penalty, and clip the objective to make sure that the optimization is carried out within a predefined range. DDPG and ACKTR adopt the actor-critic method, which estimates $Q(s, a)$ and optimizes a policy that maximizes the $Q$-function based on Monte-Carlo rollouts. DDPG does this using the deterministic policies, while ACKTR utilizes the Kronecker-factored trust regions to ensure stability with stochastic policies. As there is an infinite number of actions and states to estimate in the continuous action space, off-policy based approaches,

such as DQN, are computationally expensive. However, the on-policy based approach can move to the direction suggested by $g$ using the gradient ascent optimization for the best returning policy $\pi$.

### 3.3. U-Net Segmentation Model

U-Net architecture [47] stems from the so-called fully convolutional network. The main idea is to supplement a usual contracting network by successive layers, where upsampling operators replace pooling operations. Hence these layers increase the resolution of the output. A successive convolutional layer can then learn to assemble a precise output based on this information.

A critical modification in U-Net is that there are a large number of feature channels in the upsampling part, which allows the network to propagate context information to higher resolution layers. As a consequence, the expansive path is more or less symmetric to the contracting party and yields an u-shaped architecture. The network only uses the valid part of each convolution without any fully connected layers. To predict the pixels in the border region of the image, the missing context is extrapolated by mirroring the input image. Such a tiling strategy is essential to apply the network to large images as otherwise the resolution would be limited by the GPU memory.

## 4. Method

### 4.1. Methods for Discrete Action Space

Airsim is a new simulation platform from Microsoft for autonomous vehicles such as drones and autonomous cars. For the graphics, it uses the Unreal Engine that provides rich repertoires of shader and other drawing tools with which third party suppliers or research users can make the realistic landscapes. One of the early releases is called as "modular package", mainly consisting of the urban landscape and woodland areas. In particular, the present study utilizes the woodland area as it provides an ideal woodland landscape for testing and training an autonomous drone, as shown in the top of Figure 1. In this case, the drone supposes to avoid the trees and find a 3D path to the goal. It is found in the early stage that the small branches and leaves of the trees were not recognized as the obstacles, unlike the main branch, because the default option was set for the simple collision case presumably because the processing time for detail graphics is to save within Unreal Engine [48]. Therefore, it required to bond the small brunches and leaves to the main branch of a tree as one object and to work in the complex collision mode.

Though the woodland provides a stimulating environment for an agent, it does not require much moving up or down direction simply because most of the trees are planted on a plain ground. Therefore, we have designed more challenging landscapes, called the block world and the arena, consisting of diverse solid objects, such as a cube, sphere, cone and other two wall-style objects such as half-wall and arch bridge as shown in Figure 1. Given that most of the drones that use SLAM for their autonomous navigation typically adopt either stereovision or Lidar, we utilize RGB and depth sensors for our experiment. The sensed images are given to two CNNs as input to the deep RL network. The size of the memory buffer is set to 1000, which means 1000 pairs of depth and RGB images are stored and sampled for training.

DQN, Double DQN, Dueling DQN, DD-DQN algorithms are used for training and evaluating, respectively, in the environment of two different worlds: the woodland and the block world. For both cases, an agent learns to act with 5 actions, consisting of $forward$, $left$, $right$, $up$, and $down$. After choosing an action, we use Euler angle $pitch_{action}$, $roll_{action}$, and $throttle_{action}$ values to control the drone. Therefore, $Pitch_{drone}$, $Roll_{drone}$, $Throttle_{drone}$ commands sent to set the drone's attitude are

$$\begin{aligned} roll_{drone} &= roll_{action} * max_{angle} \\ pitch_{drone} &= pitch_{action} * max_{angle} \\ throttle_{drone} &= throttle_{drone} + throttle_{action} \end{aligned} . \quad (21)$$

$pitch_{action}$ is set to +1 for $forward$ and 0 for other 4 actions. $roll_{action}$ is set to +1 for $right$ and $-1$ for $left$. $throttle_{action}$ is set to 0.2 for $up$, and $-0.2$ for $down$, respectively. The above $roll_{drone}$, and $pitch_{drone}$ are converted to Quaternions and then sent to the drone to match the pose, whereas $throttle_{action}$ is sent to the drone directly. ROS was used for transmission from the RL agent to Airsim environment. Although we empirically chose $max_{angle}$ and $throttle_{action}$ for our environments, one can manually set those values depending on environments. For the woodland, reward $r$ is set to 0.1 only when the agent chooses to go forward and $-10$ when the agent collides with any obstacle recognized as an object. For example, every tree, leaves, and rocks are recognized as objects. For the block world case, $r$ is set to 0.1 when the agent chooses to go forward, $-10$ on a collision, and 0.08 when it makes an action such as left, right, up and down, respectively. Because the block world has a more complicated environment than the woodland case, the agent has to choose various actions to find a path and to prevent a deadlock: a reward is given to each action, i.e., promotion of exploration.

An RGB (and depth map) is rescaled from $256 \times 144$ to $64 \times 64$ to prevent overfitting, and certainly, it saves the training time of the network. When using both RGB and depth images for the training, two separate convolutional networks receive each image. The output of 2 CNNs is concatenated and used as an input for fully connected layers that are responsible for the dueling architecture. Finally, $Q$ value generated by the dueling operation is used in estimating the double Q as shown in Figure 2. When using only one type of input, which is either RGB or the depth map, a same CNN and fully connected layers with 64 neurons are used.

### 4.2. Method for Continuous Action Space

Given that U-Net-based segmentation model belongs to the supervised learning paradigm where labels are provided by laborious manual labeling, the proposed system is to combine a U-Net-based segmentation model with a policy gradient algorithm under the reward-driven way, wherein a label map is generated with an optical flow calculated from sequential images via the critic network in real-time while training. In other words, training in RL replaces the labeling task. By utilizing a realistic simulation environment for training Actor-Critic networks and the segmentation model, it is shown that the model successfully learns to control and navigate through obstacles. In this section, we will describe how our two learning processes mutually cooperate: (1) a model for the visual representation of the scene in the form of the segmentation map. (2) Actor-Critic RL for controlling the drone in the continuous action space. First, we will describe the representative model, followed by RL.

#### 4.2.1. Critic-Dependent Segmentation Model

Our actor-critic networks aim to assist the segmentation network by generating a label map. There are two steps for generating the training data in terms of predicted reward. The first is to recognize which direction the agent chooses to go within an image. An optical flow algorithm [49] is adopted in calculating these vectors. The second is to generate a label map for the segmentation model based on the predicted reward using the optical flow vectors.

To measure the direction where the RL agent chooses to go, we calculate optical flow vectors using two sequential images from the environment, $S_t$ and $S_{t-1}$. Here, $S_t$ stands for the raw RGB FPV on the time step $t$. Then the equation we use to calculate x movement $V_x^{'}$ and y movement $V_y^{'}$ is given as follow:

$$\begin{bmatrix} V_x^{'} \\ V_y^{'} \end{bmatrix} = \begin{bmatrix} \sum_i I_x(q_i)^2 & \sum_i I_X(q_i)^2 I_y(q_i) \\ \sum_i I_y(q_i) I_x(q_i) & \sum_i I_y(q_i)^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i I_x(q_i) I_t(q_i) \\ -\sum_i I_y(q_i) I_(q_i) \end{bmatrix}, \tag{22}$$

where $q_i$ denotes a pixel value inside a window, and $I$ stands for the partial derivative of $S_t$ and $S_{t-1}$ with respect to the position $(x, y)$. The size of the window for calculating optical flow is set to 10 in this study.

To get the vectors indicating where the drone moves, instead of the movement itself, we rotate $V^{'}$ using a rotate matrix as follow:

$$\begin{bmatrix} V_x \\ V_y \end{bmatrix} = \begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix} \begin{bmatrix} V_x' \\ V_y' \end{bmatrix}, \tag{23}$$

where $\theta$ is set by 180 to obtain aforementioned drone's movement.

Using $V$ calculated above, the second process is to make the label map. Using parallel displacement of vectors, we first move $V$ to the center of the image, so that the calculation of the direction and speed for generating the label map could be more straightforward. After this displacement, as shown in Figure 3, we separate an array into zones with directions that $V$ can have. Then, each direction of the vector and scaled speed are superposed on the zones. We use coefficient $\eta$, which is set to 20 in this study, for the speed value to calculate how many zones are superposed under each vector's direction. For every vector, an array filled with zero is initialized, and zones under a vector are filled with value one. Each of these arrays is flattened into 1-Dimension and fed into the critic network for estimating the direction. Let $W_{ij}$ denotes a set of values exhibited in a zone at $i_{th}$ row and $j_{th}$ column, and $V_n$ denotes $n_{th}$ optical flow vector, then each sample $C_n$ used as input for a critic network to make a label map are given as

$$C_n = \{\bar{W}_{i,j}, ..., \bar{W}_{h,w}\}, \tag{24}$$

where $\bar{W}_{i,j}$ means an average value of $W_{i,j}$. $h$ and $w$ are the number of zones for height and width, respectively. Then, each $C_n$ is fed into the critic network $\Psi$ to receive its predicted reward. The label map can be produced by filling a zeros array with the output values of $\Psi$ corresponding to zones indicated by $V_n$ of every sample $C_n$. Note that filling the windows with the values from $\Psi$ suggests each window in a label map will have the predicted reward since the critic network learns to predict the reward (Section 3.2). Figure 3 shows the label map generated with this sequential process. Input dimension for actor and critic networks is 25 with $h$ and $w$ both set to 5, as shown in Table 1. Given $S_{t-1}$ as an input and the corresponding label map described above as a target, our segmentation model learns simultaneously with actor-critic networks. As the RL model performs well, the segmentation model improves as well. During testing, the segmentation model receives RGB input and then produces a predicted path in terms of reward without further processing indicated as optical flow and feedback from $\Psi$ as shown in Figure 4.

**Table 1.** Detailed specification of our Actor and Critic networks. During training, all 3 networks such as U-Net, Actor, and Critic networks are required, whereas only the U-Net and Actor network are utilized during testing. Note that all 3 networks, including the U-Net in Figure 5, use identical optimizer, and weight initialization method. Note that * indicates types of networks.

| Layer | Layer Output Dimension | Activation |
|---|---|---|
| * Actor network | | |
| Input | $25 \times 1$ | |
| Fully connected | $64 \times 25$ | Tanh |
| Fully connected | 3 | Tanh |
| * Critic network | | |
| Input | $25 \times 1$ | |
| Fully connected | $64 \times 25$ | Tanh |
| Fully connected | 1 | Tanh |
| Optimizer | Adam((betas = 0.9,0.99), weight decay=0, lr = $1e^{-4}$) | |
| Weight, Bias unit | Kaiming He, Constant(0) | |

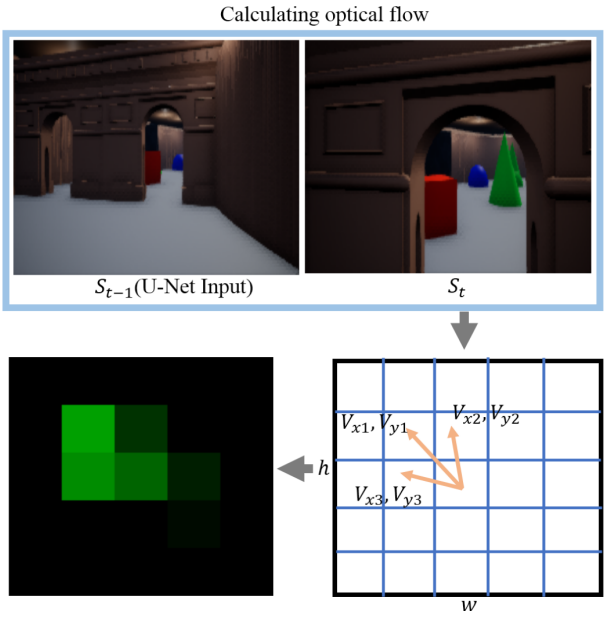


**Figure 3.** Process of generating a label map for the segmentation model. Optical flow vectors are calculated using two frames. Directions and speeds indicated by each vector are used to put value 1 on windows under the vector. Then $h \times w$ number of windows are flattened to 1-D and fed into the critic network for estimating the predicted reward in the direction. The final label map is generated by superposing the estimated rewards from all vectors. $h$ and $w$ were both set to 5 in this study.

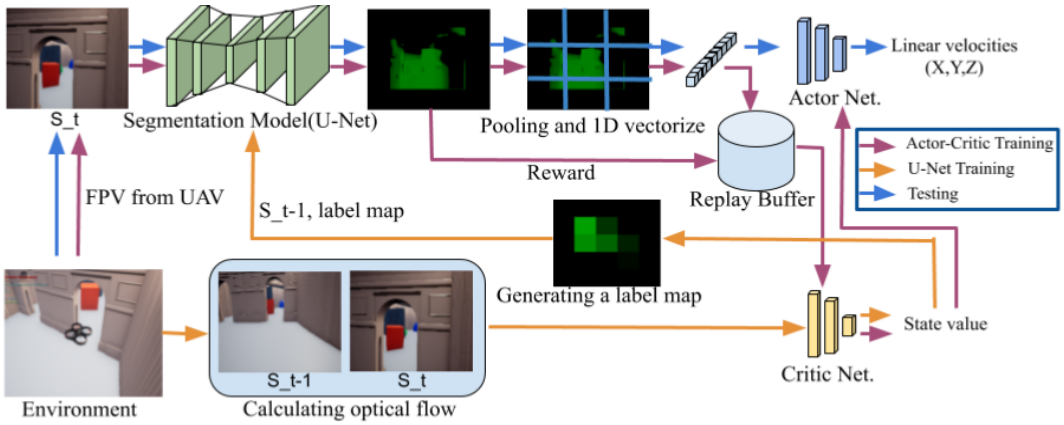


**Figure 4.** Flow diagram of training/testing for our drone navigation system. For training the U-net-based segmentation network, an optical flow calculated with two sequential images from the environment is used to generate a label map (U-net training). The output of U-net is used to calculate reward for the learning of the Critic network, as well as used as an input to Actor network through the pooling and vectorization process (Actor-Critic training). Then, the reward is calculated by comparing movement of the drone using the optical flow with the output pf U-net. The average pixel value of the segmentation map becomes a reward by comparing movements indicated by flow vectors. During testing, only a trained segmentation network and actor network are used (Testing).

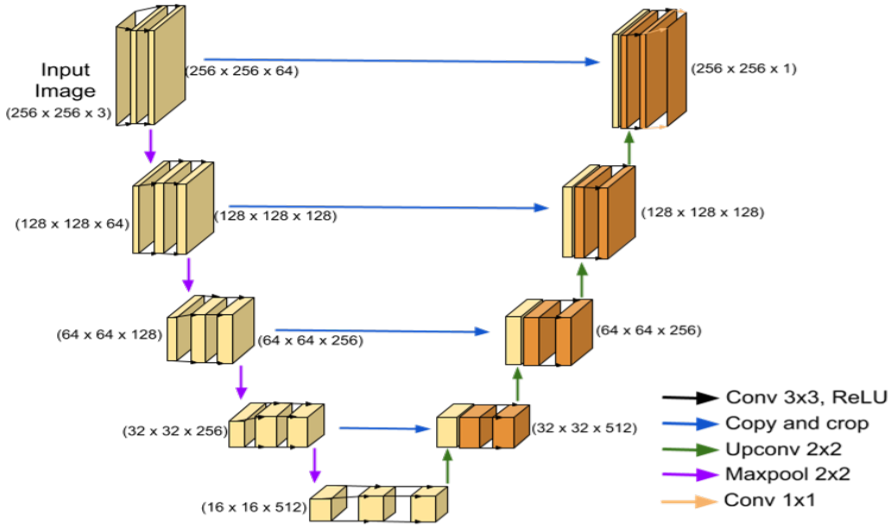


**Figure 5.** Specification of our U-Net architecture. Note that it receives $256 \times 256 \times 3$ by rescaling from $256 \times 144 \times 3$ raw image and outputs $256 \times 256 \times 1$ with Sigmoid function at the last layer, containing the segmentation of the input image in term of reward.

Although using policy gradient can be optimal in a certain environment, the problem in using vision input in the continuous action space arises when all the state has high dimensions with a similar appearance. In other words, $\pi$ and $\Psi$ cannot effectively learn from succeeding similar frames that require different actions. To overcome such an issue with high dimensional data, we adopt a segmentation model that compresses the information while preserving the useful features as mentioned earlier. Let $f_{seg}$ and $\theta_{seg}$ denote the segmentation model and its parameter, then $s_t$ for training our actor $\pi$ and critic network $\Psi$ are given as

$$s_t = \{\bar{W}_{i,j}, ..., \bar{W}_{h,w}\}, \tag{25}$$

where $W$ is the set of the values in the zones separating the output of $f_{seg}$, as Equation (24), and $\bar{W}$ means an averaged value of $W$. Our actor and critic networks accept an input that has $h \times w$ dimension for training and testing. The reward is calculated using the optical flow vector $V$ and the segmentation model's output. Given $S_t$ and $S_{t+1}$, $V$ can be calculated and indicates zones with the direction and speed. Then, the corresponding zones in a segmentation map $f_{seg}(S_t|\theta_{seg})$ are selected and averaged to become a reward for a given action $a_t$. Apart from U-net generated reward, a reward is given as $-1$ whenever the drone collides so that the critic can give both $\pi$ and $f_{seg}$ negative signals to learn. So that, the reward at time step $t$ using the segmented output $f_{seg}(S_t|\theta_{seg})$ is computed as follow:

$$r_t = \begin{cases} \frac{1}{m}\sum_{i=1}^{m} \bar{W}_i|V_t & if \text{ not collides} \\ -1 & \text{otherwise} \end{cases}, \tag{26}$$

where $\bar{W}_i|V_t$ stands for an averaged value of a zone $W_i$ indicated by $t_{th}$ optical flow vector $V_t$. $m$ is the number of zones where $V_t$ is superposed. Note that the reward value ranges from $-1$ to 1, because of Sigmoid function at the last layer of U-Net as shown in Figure 5. Action $a_t$ produced by the actor network $\pi$ using $s_t$ in Equation (25) is as follows:

$$a_t = \pi(s_t|\theta_\pi), \tag{27}$$

where $\theta_\pi$ stands for the parameter of $\pi$. With these $s_t, a_t, r_t, s_{t+1}$ defined above, our actor and critic networks make a typical optimization step in RL by utilizing an experience replay buffer. During

the testing, only the segmentation model $f_{seg}$ and the actor network $\pi$ are required as indicated by Equation (27) and the procedure, indicated as blue lines, in Figure 4.

In the end, our actor-critic networks learn to follow the segmentation model, while the segmentation model learns to generate a segmentation map as AC networks work well.

#### 4.2.2. Control Commands for Drone with the Actor Network

Our control command is conveyed as linear velocities within $X$, $Y$, and $Z$-axis. In other words, the actor network produces three continuous velocity values for a state, and this command is used to control the drone for both training and testing (evaluation) as described in Section 5.

Let $\omega_0, \omega_1, \omega_2$ denote an output action $a$ from $\pi$ given a state $s$, our control commands sent to the drone to control linear velocities are given as

$$\begin{aligned} \omega_{x\_drone} &= \omega_0 * \phi_x \\ \omega_{y\_drone} &= \omega_1 * \phi_y \\ \omega_{z\_drone} &= \omega_2 * \phi_z \end{aligned} \quad , \tag{28}$$

where $\omega_{x\_drone}$, $\omega_{y\_drone}$, and $\omega_{z\_drone}$ stand for the linear velocities for $x$, $y$, $z$ axes being sent to the drone, respectively. $\phi_x, \phi_y, \phi_z$ are the pre-defined parameters that one can set to prevent drone's attitude from moving too rapidly or either slowly. In this study, $\phi_x$ and $\phi_y$ were set to 1, whereas $\phi_y$ was set to 0.5 for the stable control of $\omega_{z\_drone}$.

## 5. Experiment

Experiments were designed to see training processes of algorithms for discrete and continuous action spaces, followed by their evaluation in 3 virtual environments made using Airsim, as shown in Figure 1. For the training in discrete action space, we have trained DQN, Double-DQN, Dueling-DQN, Double Dueling DQN(DD-DQN). For the continuous action space, TRPO, PPO, and ACKTR have been trained with U-Net for the assistant segmentation as described in Section 4. Evaluation of the networks has been carried out by comparing each algorithm for both action spaces as well as with human pilots whose maneuvering skills vary, i.e., novice, intermediate, and expert.

### *5.1. Learning Environments*

A work station equipped with Intel i7 3.4 GHz CPU and an Nvidia Titan X was used for both training and testing with a Microsoft Airsim [7] simulation environment. Python 3.6, Tensorflow 1.10.0 [50], and OpenCV 3.4.1 [51] were used for experiments in Ubuntu 16.04 OS. During the training, the *simpleflight* mode within Airsim was used. Hardware in the Loop (HITL) was used in measuring the performance of human pilot where a Graupner mz-12 Radio Control(RC) transmitter and a Pixhawk PX4 are connected to Airsim as shown in Figure 6. As PX4 is connected to Airsim, the pilot can use an RC directly to control a drone within Airsim. The signal from the RC is calculated and sent to Airsim in the same way as other commercially available drones whose flight controller is PX4. In our experiments, 3 human pilots calibrate their RC using the QGroundControl. The memory usage of the program was in a total of 280 MiB.

### *5.2. Network Training for Discrete and Continuous Action Spaces*

The experiment was designed to see if the networks were able to maximize the reward by making trials and errors. If that is the case, such rewards indicate that a drone has an obstacle avoidance capability.

#### 5.2.1. Training for Discrete Action Space

Training an agent in discrete action space has been made using algorithms, such as DQN, Double DQN, Dueling DQN, Double Dueling DQN. With such algorithms, experiments were designed to

examine the effect of varying two conditions. The first one is to see if the algorithms were able to maximize the reward by learning. The second was designed to see the impact of input data type, i.e., RGB or Depth, on the learning.

Throughout the experiments in the woodland, block world, and arena, it was found that Double Dueling DQN (DD-DQN) showed the best performance in learning to maximize the reward among four algorithms as shown in Figure 7. This was because DD-DQN is an algorithm that combines Dueling DQN and Double DQN, which differently contributed to improving vanilla DQN. Second-best was Dueling DQN, followed by Double DQN and vanilla DQN. Also, in every case, using both RGB and depth maps for the training was performing the best, as shown in Figure 7. It was also found to be better to use a depth map than using only RGB. This indicates that the depth map contains a more informative feature than RGB for the learning.

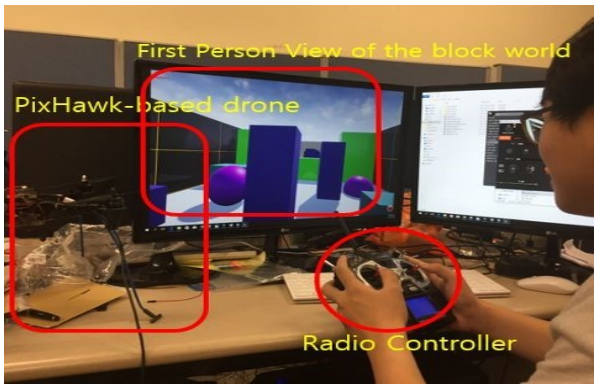


**Figure 6.** A drone pilot navigates through 3D objects within the block world where an Radio Controller(RC) and a PX4 are connected to Airsim, playing a Hardware-In-The-Loop (HITL) mode. Note that the subject sees the first person view of the block world and performance of human is measured during the navigation task.

#### 5.2.2. Training for Continuous Action Space

This experiment was made to see if actor-critic networks described in Section 4 for continuous action space can maximize the reward. As the reward is accumulated larger as an agent avoids obstacles, such a reward can indicate that a drone has an obstacle avoidance capability. For this purpose, starting points for the training were set as identical locations as training for discrete action spaces in 3 environments, respectively. After the setting, processes for training actor-critic networks and U-Net followed the typical RL process. Since U-Net already plays a role in semantic segmentation, and its role is in a way similar to convert RGB to the depth map, only RGB image was used for the learning in continuous action spaces. With the same specification of U-Net, 3 actor-critic algorithms, such as TRPO, PPO, ACKTR, were adopted to compare their performance for obstacle avoidance drones.

Figure 8 shows how our U-Net segmentation model produces the output as the training proceeds. Decrement of U-Net loss, as well as increment of reward, indicates that the agent's performance improves. As a result, it can make a successful path even in a newly configured environment. Note that when the drone saw a different scene for the first time, the segmentation map produced by U-Net was blurry on the given scene. However, by repeating trial and error made by actor-critic networks, the U-Net loss decreases as the networks learn the surrounding area. After the loss became stable by learning many scenes indicated by training steps around 55 and 78 in Figure 8 (left), the agent obtained a high reward score. It shows that our agent became familiar with these obstacles through trials and errors. The segmentation model learned by knowing where the agent had to go with the given optical flow and the feedback from the critic net, whose role was to evaluate how good the action was given the state. The obtained rewards by the actor network also increased as the U-Net loss decreased, indicating that the segmentation model and actor-critic networks cooperated to avoid

obstacles and made a successful path. Figure 9 shows how the actor network made flight trajectory given segmentation outputs.

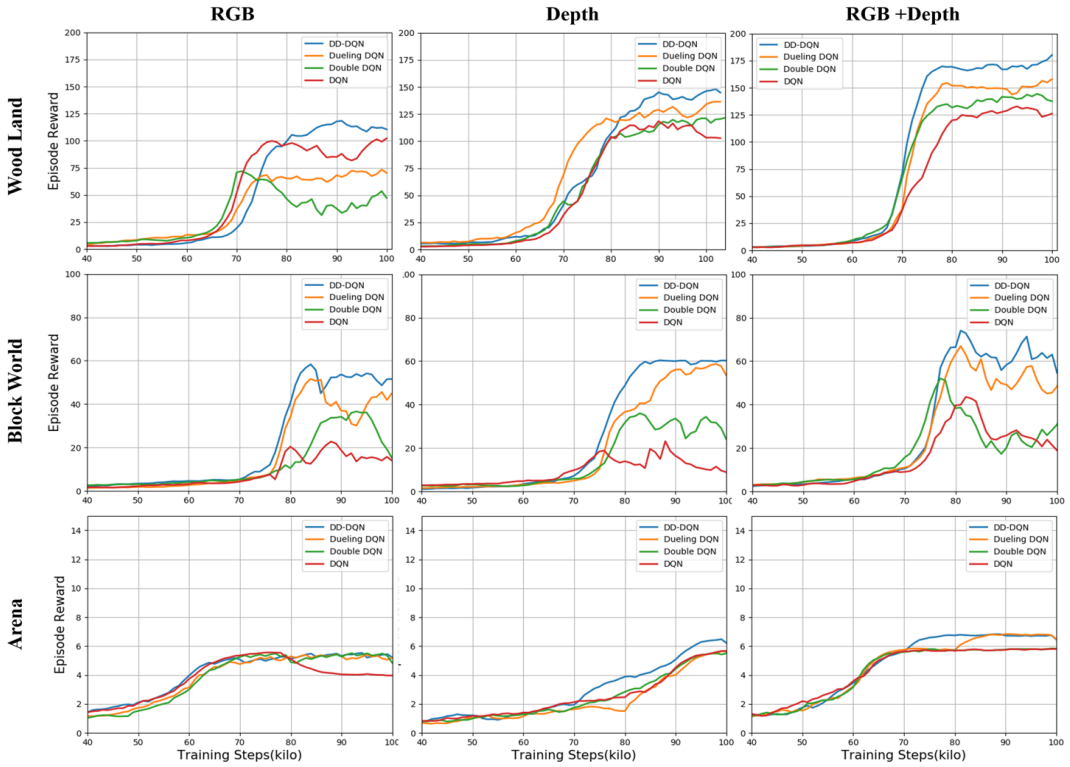


**Figure 7.** Illustration of episode rewards for four deep RL algorithms with RGB (**left**), depth (**middle**), and RGB + depth (**right**), respectively, in 3 environments such as the woodland (**top**), the block world (**middle**), and the area world (**bottom**). The reward was the biggest when both RGB and depth input were combined with DD-DQN, as shown in (c) of the woodland and the block world. However, since these algorithms could not complete their races in the arena world, the differences between them were not significant. Implementations for these 4 algorithms were based on OpenAI [45].

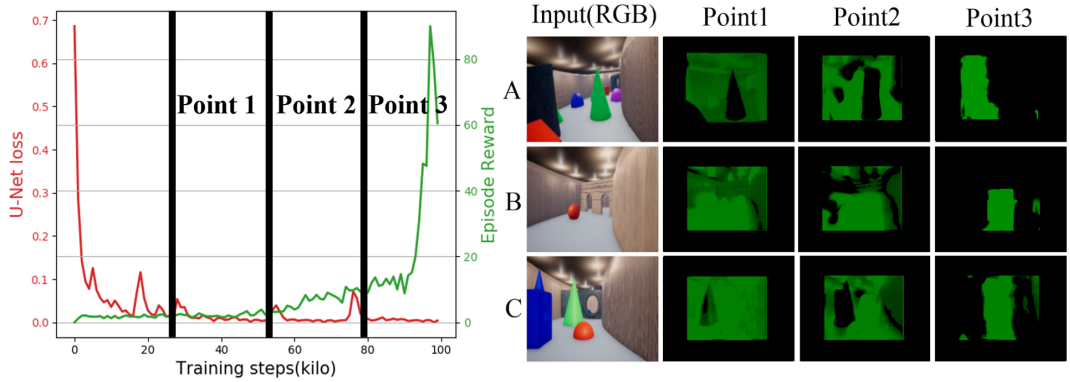


**Figure 8.** U-net training and its corresponding outputs. As the training makes a progress, the U-net loss decreases and the reward obtained by an RL (Kronecker-Factored Trust Region (ACKTR)) agent increases (**Left**). Three different points during the training period are selected to illustrate how the segmented outputs from U-Net become clearer as training makes a progress (**Right**). The three locations, A, B, and C, selected along with a drone flight within the maze, are also shown in Figure 12a.

The second experiment was to determine which actor-critic based algorithm performed better on the given task. It was conducted by replacing the actor-critic algorithm one by one, whereas the U-net and reward scheme was identical. As shown in Figure 10, 3 recent and high-performing

actor-critic algorithms, TRPO [10], PPO [11], and ACKTR [43] were evaluated. It is found that ACKTR outperformed two other algorithms for the given task, as shown in Figure 10.

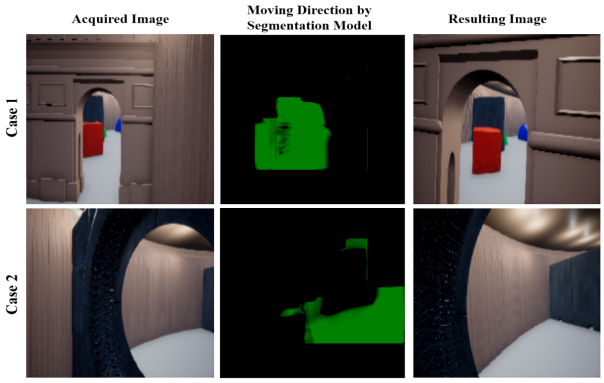


**Figure 9.** Determination of moving direction according to the segmentation model's output. Case 1: As the acquired image contains an open space in the left, the model produces a cluster of green areas in the same direction. Accordingly, the drone drifts to the left, as shown in the resulting image. Case 2: When open space is located in the right side of the acquired image, the drone drifts to right, as shown in the resulting image. See also Figure 12.

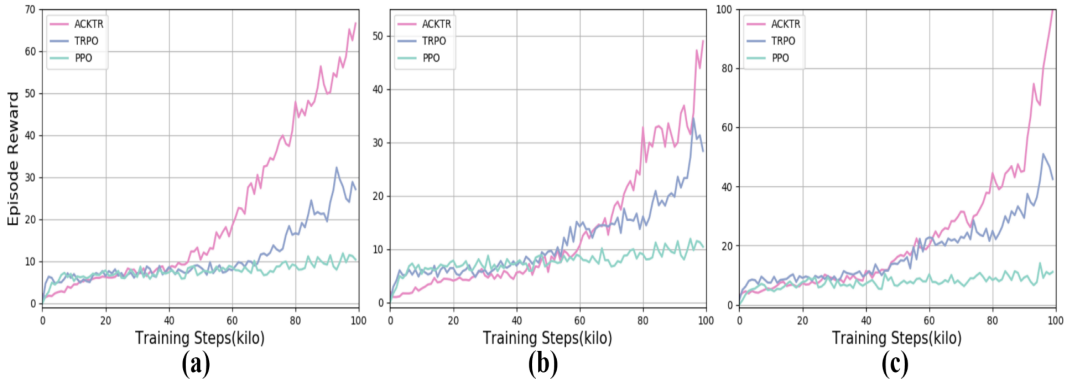


**Figure 10.** Average performances of three policy gradient algorithms used for our Actor-Critic networks within three environments: (**a**) the woodland, (**b**) the block world, (**c**) the arena world, respectively. Implementations for these algorithms were based on OpenAI [45] and their default hyper-parameters for a fair comparison. Note that ACKTR outperformed others.

### *5.3. Evaluation of Networks*

For the evaluation of algorithms for both discrete and continuous action spaces, the trained models were used for testing in each of the 3 environments.

#### 5.3.1. Comparison between Algorithms

To better understand how these algorithms work in three environments, the trained network models ran in three environments such as the woodland, the block world, and the arena world, respectively. It is found that DD-DQN in discrete action space outperformed others, whereas ACKTR in continuous action space excelled others. DD-DQN made discrete trajectories as shown in Figure 11a,c, mainly because it had only 5 possible actions, i.e., forward, left, right, up, and down. On the other hand, ACKTR made smooth and continuous trajectories as shown in Figures 9 and 11b,d, because it could produce actions in continuous space with three linear velocities. Note that such differences in trajectories were more evident in the block world where there were densely located obstacles that each agent had to avoid during its journey to the goal.

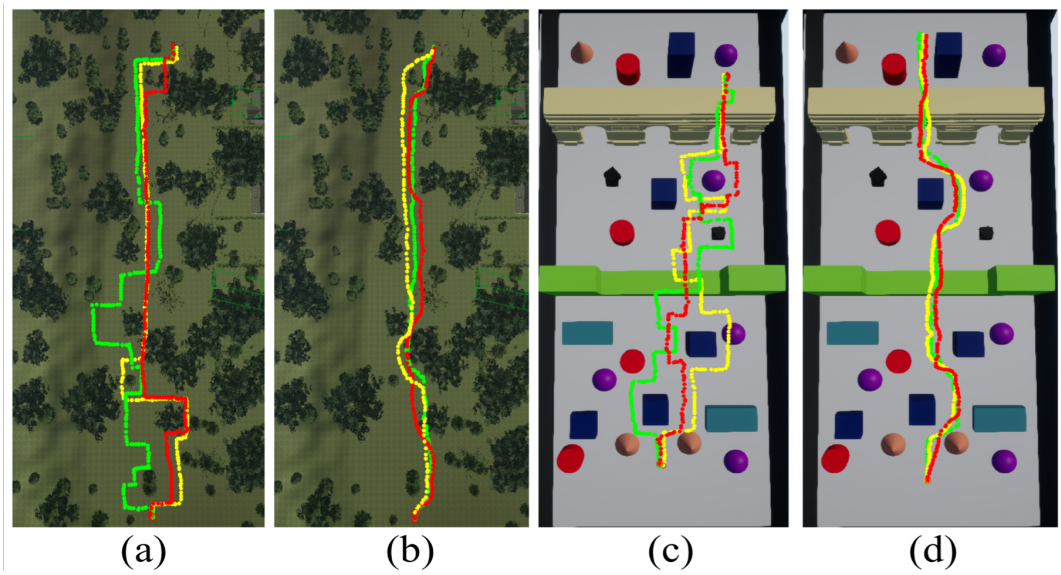


**Figure 11.** Illustration of drone flight trajectories by two deep reinforced learning (RL) algorithms such as DD-DQN (**a**,**c**) and ACKTR (**b**,**d**). (**a**,**b**) were collected from the woodland and (**c**,**d**) from the block world, respectively. Here, training models at step 90 k (green), 95 k (yellow), and 100 k (red) were deployed for each algorithm. Note that DD-DQN made discrete trajectories, whereas ACKTR generated continuous trajectories. Note that, in each panel, only 3 trials among many are drawn for clarity.

Our arena world consists of two kinds of obstacles: the first is wall-type large obstacles, consisting of two double arch bridges with sand color and two hoop-embedded black walls; the second is the small obstacles such as cone, half sphere, cuboid with different colors, as shown in Figures 8 and 12. Since all obstacles were densely located along with a circular track, algorithms with discrete action space failed to reach the goal mainly because they did not have delicate yaw control. However, it was inevitable to restrict the action space because adding yaw could be having too many actions, which is known to be leading to unstable learning. Note that DD-DQN could not reach the goal as shown in Figure 12a, whereas ACKTR reached the goal successfully by producing different linear velocities.

#### 5.3.2. Experiment on Robustness of Actor Network

As previous studies suggested that the segmentation model could offer robustness against a certain environmental change, experiments were designed to see how an actor network and the corresponding U-Net segmentation model would perform when there was a change in the environment. Since our actor network could learn to follow the maximum reward zone suggested by U-Net, the actor network could perform well if the segmentation would be successful with given obstacles.

Our segmentation network combined with the actor-critic network was trained using the arena track with Figure 12a. Three more tracks were reconfigured from the arena (a) by calculating how much each track was changed in ratio $\sigma$. The arena track (b) consists of shuffled obstacles ($\sigma = 65\%$), having the same 4 large obstacles, whereas the arena track (c) was made by shuffling large and small obstacles as well as by adding new obstacles with different colors ($\sigma = 75\%$). The arena track (d) was made from (c) with more obstacles ($\sigma = 81\%$). Results suggest that our network cooperating with U-Net could reach the goal in (b) and (c), where there were changes about 65% and 75%, respectively, from (a). However, when the environment was too different from what it had been, such as 81% in (d), it failed to reach the goal probably because of a severely reconfigured environment. Note that $\sigma$ has been calculated as follows:

$$\sigma = \frac{\text{Number of obstacles added or changed from track (a)}}{\text{Number of obstacles in a target track}} \times 100 \tag{29}$$

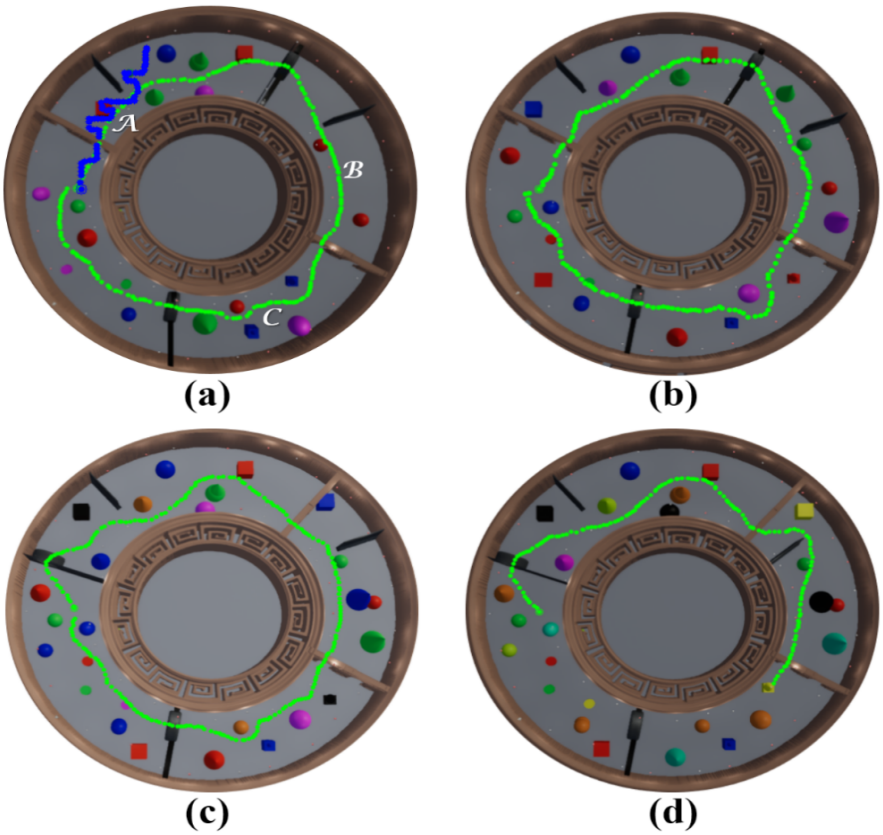


**Figure 12.** Drone flight trajectories collected from the trained arena track (**a**) as well as three untrained arena tracks (**b**–**d**), where the sequence of obstacles were shuffled. Note that A, B, C in (**a**) indicate the locations used for visualization of our segmentation model during its training, as shown in Figure 8.

#### 5.3.3. Racing between Human Pilot and Algorithm

This experiment was prepared to examine the performance between the RL algorithms and drone pilots. For this purpose, a few drone pilots were recruited based on their skills level of drone control, i.e., novice, intermediate, and expert. The novice had about two weeks of experience of drone piloting, the intermediate pilot about six months, and the expert pilot had more than two years of experience. Test environments were the woodland, the block world, and the arena (a) in Figure 12. To inspect how fast the pilots or algorithms reached the goal, 100 lines equally dividing the tracks until the goal was installed and gave 1 point when the drone passed through each of them. After the setting described in Section 5.1, all pilots did practice ten times before recording the data for comparison against the algorithms.

In the woodland, the fastest was expert pilot (47 s), followed by ACKTR (51), intermediate pilot (78), DD-DQN (84), and novice pilot (102) as shown in Figure 13. When there were more obstacles densely located in the block world, the gap between the expert (37) and ACKTR (39) agent was shortened. Interestingly, the gap between the intermediate pilot (56) and DD-DQN (70) was increased, indicating that the DD-DQN agent had difficulty in avoiding obstacles with discrete action space. In the arena, ACKTR reached the goal fastest (51) followed by the expert (61), and all others could not complete the tasks, suggesting that this track was more difficult than two other tracks.

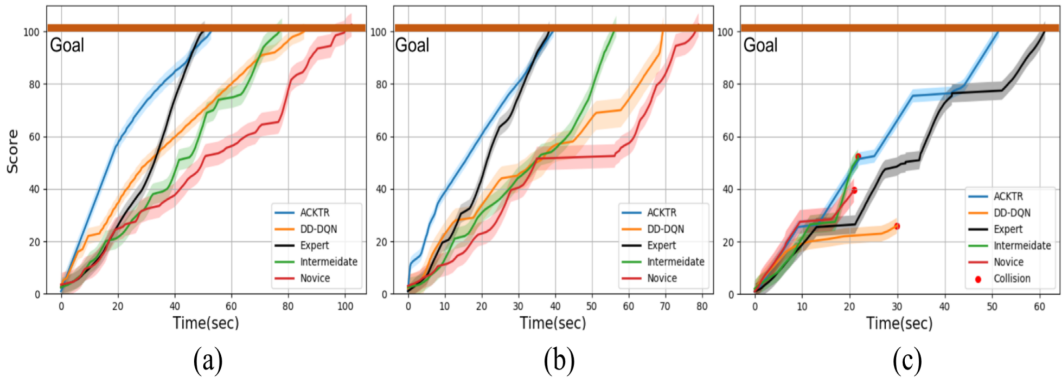


**Figure 13.** Performance comparison in terms of time taken from the start (score = 0) to the goal (score = 100) between human drone pilots and algorithms. For the woodland (**a**) and the block world (**b**), both the expert and ACKTR were the winners with similar speeds, whereas ACKTR excelled others in the arena world (**c**), wherein DD-DQN, the intermediate and novice drone pilots could not complete their races because of collisions, indicated as red dots, made during navigation. Each algorithm (as well as each drone pilot subject) made 3 trials within each environment. The average score for each case is plotted as a color line with shade for indicating its variation. Note that the deployed models were trained with 90 k (**a**), 95 k (**b**), and 100 k (**c**) steps, respectively. Our demo video can be found at: https://youtu.be/en6Xwht8ZSA.

## 6. Discussion

The drone can fly outdoor as well as indoors depending on its usage. Training a drone using the machine learning algorithm requires a well-prepared dataset. In particular, collecting a high-quality outdoor dataset is not easy. Gazebo has been a favored drone simulation tool by which one can collect data and fly his drone after training, although so far most cases were for indoor navigations. Airsim provides a new opportunity of collecting realistic outdoor datasets. In this study, we use one of the packages from Airsim, i.e., the woodland, and design two environments, i.e., the block world and the arena world, for training deep RL algorithms. We also plan to release these for public use.

The present study utilizes both RGB and depth map images as input for 4 RL algorithms with discrete action space. Given that RGB image is the FPV of the environment from the drone and depth map has the same view except for the fact that it is extracted from stereo images, these inputs contain the partial information of the environment, whereas the input image typically has the whole information at a given moment in playing any Atari game with DQN, that could make the present pathfinding task harder. As we want to know how individual sensor contributes to the performance, an experiment was carried out using 3 different cases such as RGB, depth image and depth + RGB, separately. Result suggests that the depth + RGB case outperformed the other two cases. For the comparison between 4 deep RL algorithms with such input types, we found that all deep RL algorithms succeed in finding the goal for the woodland experiment, whereas only Dueling DQN and DD-DQN were able to arrive at the goal for the block world case. This confirms that pathfinding in the block world was relatively harder than in the woodland for the deep RL algorithms. However, these algorithms failed to reach the goal in the arena track, confirming that they had a limitation in yaw control.

It is well known that gaining some skill for maneuvering the drone using an RC, especially for navigating through a group of 3D obstacles, often needs a certain period of training for a drone pilot. In this study, deep RL algorithms have been used to train a drone agent that supposes to find a path through obstacles and eventually to arrive at the goal. By using the HITL mode of Airsim, it was possible to measure ther performance of human pilots and to compare it with the performance of the algorithms.

## 7. Conclusions

In this study, we compared RL for discrete and continuous action spaces in avoiding obstacles by drone. A new method using a segmentation network for continuous action space is trained using an actor-critic network from the RL paradigm. The significant advantage of this is, of course, that manual labeling is not necessary, saving labor and time. The performance of U-net-based segmentation model is also improved very much. The question for RL in terms of autonomous navigation problems was how one could minimize the gap between real and training environments. Through a series of experiments, we demonstrate that our trained model made successful flying journeys not only in the trained environment but also in some re-configured environments. As far as we know the literature, the present study could be the first attempt where human pilot made a drone racing with algorithm and performance between them were evaluated. We plan to make a real arena shaped environment, that is often used for the drone racing championship, for testing obstacle avoidance drones.

**Author Contributions:** S.-Y.S.: Methodology, Software, Validation, Writing; Y.-W.K.: Software, Validation, Formal analysis, Y.-G.K.: Conceptualization, Project administration, Supervision,Writing

**Acknowledgments:** This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2016-0-00312) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation).

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Restas, A. Drone applications for supporting disaster management. *World J. Eng. Technol.* **2015**, *3*, 316. [CrossRef]
2. Tang, L.; Shao, G. Drone remote sensing for forestry research and practices. *J. For. Res.* **2015**, *26*, 791–797. [CrossRef]
3. Tripicchio, P.; Satler, M.; Dabisias, G.; Ruffaldi, E.; Avizzano, C.A. Towards smart farming and sustainable agriculture with drones. In Proceedings of the IEEE 2015 International Conference on Intelligent Environments, Prague, Czech Republic, 15–17 July 2015; pp. 140–143.
4. Prosser, M. Why the Fast-Paced World of Drone Sports Is Getting So Popular. 2017. Available online: https://singularityhub.com/2017/05/05/why-the-fast-paced-world-of-drone-sports-is-getting-so-popular (accessed on 9 December 2019).
5. Lynen, S.; Sattler, T.; Bosse, M.; Hesch, J.A.; Pollefeys, M.; Siegwart, R. Get Out of My Lab: Large-scale, Real-Time Visual-Inertial Localization. In *Robotics: Science and Systems*; RssPublisher: Rome, Italy, 2015; Volume 1.
6. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; et al. Human-level control through deep reinforcement learning. *Nature* **2015**, *518*, 529. [CrossRef] [PubMed]
7. Shah, S.; Dey, D.; Lovett, C.; Kapoor, A. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In *Field and Service Robotics*; Springer: Berlin, Germany, 2018; pp. 621–635.
8. Van Hasselt, H.; Guez, A.; Silver, D. Deep reinforcement learning with double q-learning. In Proceedings of the Thirtieth AAAI conference on artificial intelligence, Phoenix, AZ, USA, 12–17 February 2016.
9. Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; De Freitas, N. Dueling Network Architectures for Deep Reinforcement Learning. In Proceedings of the 33rd International Conference on International Conference on Machine Learning—Volume 48. JMLR.org (ICML'16), New York, NY, USA, 19–24 June 2016; pp. 1995–2003.
10. Schulman, J.; Levine, S.; Abbeel, P.; Jordan, M.; Moritz, P. Trust region policy optimization. In Proceedings of the International Conference on Machine Learning, Lille, France, 6–11 July 2015; pp. 1889–1897.
11. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal policy optimization algorithms. *arXiv* **2017**, arXiv:1707.06347.
12. Carrio, A.; Sampedro, C.; Rodriguez-Ramos, A.; Campoy, P. A review of deep learning methods and applications for unmanned aerial vehicles. *J. Sens.* **2017**, *2017*, 3296874. [CrossRef]

13. Kim, D.K.; Chen, T. Deep Neural Network for Real-Time Autonomous Indoor Navigation. *arXiv* **2015**, arXiv:1511.04668.
14. Gandhi, D.; Pinto, L.; Gupta, A. Learning to fly by crashing. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 3948–3955.
15. Yang, S.; Konam, S.; Ma, C.; Rosenthal, S.; Veloso, M.; Scherer, S. Obstacle avoidance through deep networks based intermediate perception. *arXiv* **2017**, arXiv:1704.08759.
16. Andersson, O.; Wzorek, M.; Doherty, P. Deep Learning Quadcopter Control via Risk-Aware Active Learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, San Francisco, CA, USA, 4–9 February 2017; pp. 3812–3818.
17. Koenig, N.; Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. In Proceedings of the 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No. 04CH37566), Sendai, Japan, 28 September–2 October 2004; Volume 3, pp. 2149–2154. [CrossRef]
18. Kelchtermans, K.; Tuytelaars, T. How hard is it to cross the room?—Training (Recurrent) Neural Networks to steer a UAV. *arXiv* **2017**, arXiv:1702.07600.
19. Kaufmann, E.; Loquercio, A.; Ranftl, R.; Dosovitskiy, A.; Koltun, V.; Scaramuzza, D. Deep Drone Racing: Learning Agile Flight in Dynamic Environments. *arXiv* **2018**, arXiv:1806.08548.
20. Loquercio, A.; Maqueda, A.I.; del Blanco, C.R.; Scaramuzza, D. Dronet: Learning to fly by driving. *IEEE Robot. Autom. Lett.* **2018**, *3*, 1088–1095. [CrossRef]
21. Alsamhi, S.H.; Ma, O.; Ansari, M.S.; Almalki, F.A. Survey on Collaborative Smart Drones and Internet of Things for Improving Smartness of Smart Cities. *IEEE Access* **2019**, *7*, 128125–128152. [CrossRef]
22. Alsamhi, S.H.; Ma, O.; Ansari, M.S.; Gupta, S.K. Collaboration of Drone and Internet of Public Safety Things in Smart Cities: An Overview of QoS and Network Performance Optimization. *Drones* **2019**, *3*. [CrossRef]
23. Bah, M.D.; Hafiane, A.; Canals, R. Deep Learning with Unsupervised Data Labeling for Weed Detection in Line Crops in UAV Images. *Remote. Sens.* **2018**, *10*, 1690. [CrossRef]
24. Huang, T.; Zhao, S.; Geng, L.; Xu, Q. Unsupervised Monocular Depth Estimation Based on Residual Neural Network of Coarse–Refined Feature Extractions for Drone. *Electronics* **2019**, *8*, 1179. [CrossRef]
25. Hirose, N.; Sadeghian, A.; Goebel, P.; Savarese, S. To go or not to go? A near unsupervised learning approach for robot navigation. *arXiv* **2017**, arXiv:1709.05439.
26. Wang, Y.; Yoshihashi, R.; Kawakami, R.; You, S.; Harano, T.; Ito, M.; Komagome, K.; Iida, M.; Naemura, T. Unsupervised anomaly detection with compact deep features for wind turbine blade images taken by a drone. *Ipsj Trans. Comput. Vis. Appl.* **2019**, *11*, 3. [CrossRef]
27. Kahn, G.; Villaflor, A.; Pong, V.H.; Abbeel, P.; Levine, S. Uncertainty-Aware Reinforcement Learning for Collision Avoidance. *arXiv* **2017**, arXiv:1702.01182.
28. Imanberdiyev, N.; Fu, C.; Kayacan, E.; Chen, I.M. Autonomous navigation of uav by using real-time model-based reinforcement learning. In Proceedings of the IEEE 2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV), Phuket, Thailand, 13–15 November 2016; pp. 1–6.
29. Xie, L.; Wang, S.; Markham, A.; Trigoni, N. Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning. In Proceedings of the RSS 2017 workshop on New Frontiers for Deep Learning in Robotics, Boston, MA, USA, 15 July 2017.
30. Faust, A.; Palunko, I.; Cruz, P.; Fierro, R.; Tapia, L. Automated aerial suspended cargo delivery through reinforcement learning. *Artif. Intell.* **2017**, *247*, 381–398. [CrossRef]
31. Koch, W.; Mancuso, R.; West, R.; Bestavros, A. Reinforcement learning for UAV attitude control. *Acm Trans. -Cyber-Phys. Syst.* **2019**, *3*, 22. [CrossRef]
32. Wang, C.; Wang, J.; Zhang, X.; Zhang, X. Autonomous navigation of UAV in large-scale unknown complex environment with deep reinforcement learning. In Proceedings of the 2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Montreal, QC, Canada, 14–16 November 2017; pp. 858–862.
33. Challita, U.; Saad, W.; Bettstetter, C. Cellular-Connected UAVs over 5G: Deep Reinforcement Learning for Interference Management. *arXiv* **2018**, arXiv:1801.05500.
34. Shin, S.; Kang, Y.; Kim, Y. Automatic Drone Navigation in Realistic 3D Landscapes using Deep Reinforcement Learning. In Proceedings of the 2019 6th International Conference on Control, Decision and Information Technologies (CoDIT), Paris, France, 23–26 April 2019; pp. 1072–1077. [CrossRef]

35. Lin, J.; Wang, W.J.; Huang, S.K.; Chen, H.C. Learning based semantic segmentation for robot navigation in outdoor environment. In Proceedings of the IEEE 2017 Joint 17th World Congress of International Fuzzy Systems Association and 9th International Conference on Soft Computing and Intelligent Systems (IFSA-SCIS), Otsu, Japan, 27–30 June 2017; pp. 1–5.
36. Hong, Z.W.; Chen, Y.M.; Yang, H.K.; Su, S.Y.; Shann, T.Y.; Chang, Y.H.; Ho, B.H.L.; Tu, C.C.; Hsiao, T.C.; et al. Virtual-to-Real: Learning to Control in Visual Semantic Segmentation. In Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18, International Joint Conferences on Artificial Intelligence Organization, Stockholm, Sweden, 9–19 July 2018; pp. 4912–4920. [CrossRef]
37. Mousavian, A.; Toshev, A.; Fišer, M.; Košecká, J.; Wahid, A.; Davidson, J. Visual representations for semantic target driven navigation. In Proceedings of the IEEE 2019 International Conference on Robotics and Automation (ICRA), Montreal, QC, Canada, 20–24 May 2019; pp. 8846–8852.
38. Chakravarty, P.; Kelchtermans, K.; Roussel, T.; Wellens, S.; Tuytelaars, T.; Van Eycken, L. CNN-based single image obstacle avoidance on a quadrotor. In Proceedings of the 2017 IEEE International Conference on Robotics and Automation (ICRA), Singapore, 29 May–3 June 2017; pp. 6369–6374.
39. Sutton, R.S.; Barto, A.G. *Introduction to Reinforcement Learning*; MIT Press: Cambridge, UK, 1998; Volume 135.
40. Watkins, C.J.C.H. Learning from Delayed Rewards. Ph.D. Thesis, King's College, Cambridge, UK, 1989.
41. Wang, Z.; Schaul, T.; Hessel, M.; Van Hasselt, H.; Lanctot, M.; De Freitas, N. Dueling network architectures for deep reinforcement learning. *arXiv* **2015**, arXiv:1511.06581.
42. Silver, D.; Lever, G.; Heess, N.; Degris, T.; Wierstra, D.; Riedmiller, M. Deterministic Policy Gradient Algorithms. In Proceedings of the 31st International Conference on Machine Learning, ICML 2014, Beijing, China, 21–26 June 2014.
43. Wu, Y.; Mansimov, E.; Grosse, R.B.; Liao, S.; Ba, J. Scalable trust-region method for deep reinforcement learning using kronecker-factored approximation. In *Advances in Neural Information Processing Systems*; The MIT Press: Cambridge, MA, USA, 2017; pp. 5279–5288.
44. Todorov, E.; Erez, T.; Tassa, Y. MuJoCo: A physics engine for model-based control. In Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, Vilamoura, Portugal, 7–12 October 2012; pp. 5026–5033.
45. Dhariwal, P.; Hesse, C.; Klimov, O.; Nichol, A.; Plappert, M.; Radford, A.; Schulman, J.; Sidor, S.; Wu, Y.; Zhokhov, P. OpenAI Baselines. 2017. Available online: https://github.com/openai/baselines (accessed on 9 December 2019).
46. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; Abbeel, P. High-Dimensional Continuous Control Using Generalized Advantage Estimation. In Proceedings of the International Conference on Learning Representations (ICLR), San Juan, Puerto Rico, 2–4 May 2016.
47. Ronneberger, O.; Fischer, P.; Brox, T. U-net: Convolutional networks for biomedical image segmentation. In Proceedings of the International Conference on Medical image computing and computer-assisted intervention, Shenzhen, China, 13–17 October 2019; Springer: Berlin, Germany, 2015; pp. 234–241.
48. Karis, B.; Games, E. Real shading in unreal engine 4. *Proc. Phys. Based Shading Theory Pract.* **2013**, *4*, 621–635.
49. Lucas, B.D.; Kanade, T. An Iterative Image Registration Technique with an Application to Stereo Vision. In Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI'81), Vancouver, BC, Canada, 24–28 August 1981; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 1981; Volume 2, pp. 674–679.
50. Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G.S.; Davis, A; Dean, J; Devin, M; et al. TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, 2015. Software. Available online: tensorflow.org (accessed on 9 December 2019).
51. Bradski, G. *The OpenCV Library*; *Dr. Dobb's Journal of Software Tools*. Software. 2000. Available online: https://opencv.org/ (accessed on 12 December 2019).