Report for Game of Drones: A NeurIPS 2019 Competition

Sang-Yun Shin
Dept. of Computer Engineering
Sejong University
Seoul, Korea
kimshin812@gmail.com

Yong-Won Kang
Dept. of Computer Engineering
Sejong University
Seoul, Korea
kyw2539@gmail.com

Yong-Guk Kim*
Dept. of Computer Engineering
Sejong University
Seoul, Korea
ykim@sejong.ac.kr

***Abstract*—This document is a report for Game of Drones: A NeurIPS 2019 Competition. Several latest deep learning techniques, such as reinforcement learning and supervised learning for object detection, are combined to control the drone in navigating through the gates. First, an object detection model is trained to detect objects, such as the gates, sky, and trees, from the input. Second, an actor-critic network with an U-Net model is trained within a simulated environment. Then the trained actor model is used for controlling the drone within two competition environments. The result shows that our model successfully flies through the gates while avoiding collision to the surrounding objects.**

***Index Terms*—object detection, actor-critic network, U-net, simulation, obstacle avoidance**

I. Method

This section describes our approach for Tier 2 (perception only), and Tier 3 (perception+planning) competition wherein the drone perceives the gates from the incoming image and navigates through them without making any collision. Given this mission is given, the first task is to detect the gates as objects using a neural network. As a target gate among many is determined, the drone needs to fly through it. For this, a reinforcement learning-based pretrained actor model is used [1].

A. Training object detection model

Using the object detection API from Tensorflow [2], we have tested recent detection models such as SSD-mobilenet [3], [4], SSD-inceptionV2 [5], fastrcnn-resnet [6], [7], and rfcn-resnet [8]. To train these models, we have collected the fpv from the training maps for Tier 1,2, and 3 consisting of 346 images. They are annotated with 4 labels: gate, gate entry, sky, and tree. Then, data augmentation was carried out using a method such as cropping and morphing. The final dataset has 12,943 images in total.

To make the detection models indicate the direction, we fill the value 1 to the zero arrays where the bounding box for the gate is. Then, the array is pooled and flattened into 1D for the actor-network, as shown in Figure 2.

B. Reward-driven training for obstacle avoidance drone

In a method that has been recently published [1], two learning processes have cooperated interactively: 1) a model

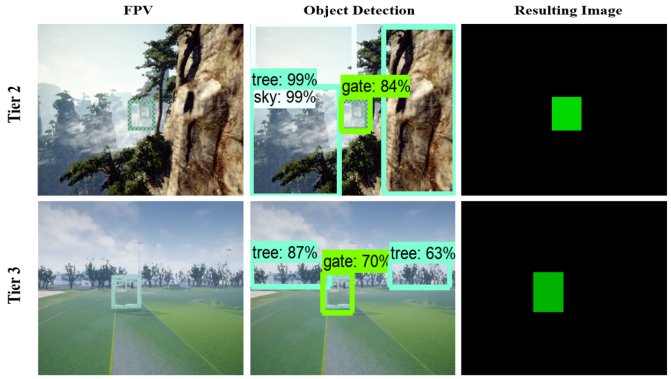

Fig. 1: Illustration of generating input for our pre-trained actor network using an object detection model. A FPV scene(left) from the drone is fed into the detection model to detect where a gate is(middle). Then, the gate's region is filled with value 1 to indicate where to go for our pre-trained actor network(right). SSD-Mobilenet trained with 346 images is used here.

for the visual representation of the scene in the form of the segmentation map. 2) an Actor-Critic for controlling the drone in the continuous action space. First, we will describe the representative model, followed by the reinforcement learning (RL) part.

1) Critic-dependent Segmentation Model: The Actor-Critic network aims 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 [9] is adopted in calculating these vectors for its simplicity and accuracy. 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 V using two sequential images from the environment, S_t and S_{t-1}. Here, S_t stands for the raw RGB FPV on time step t. Using V, 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

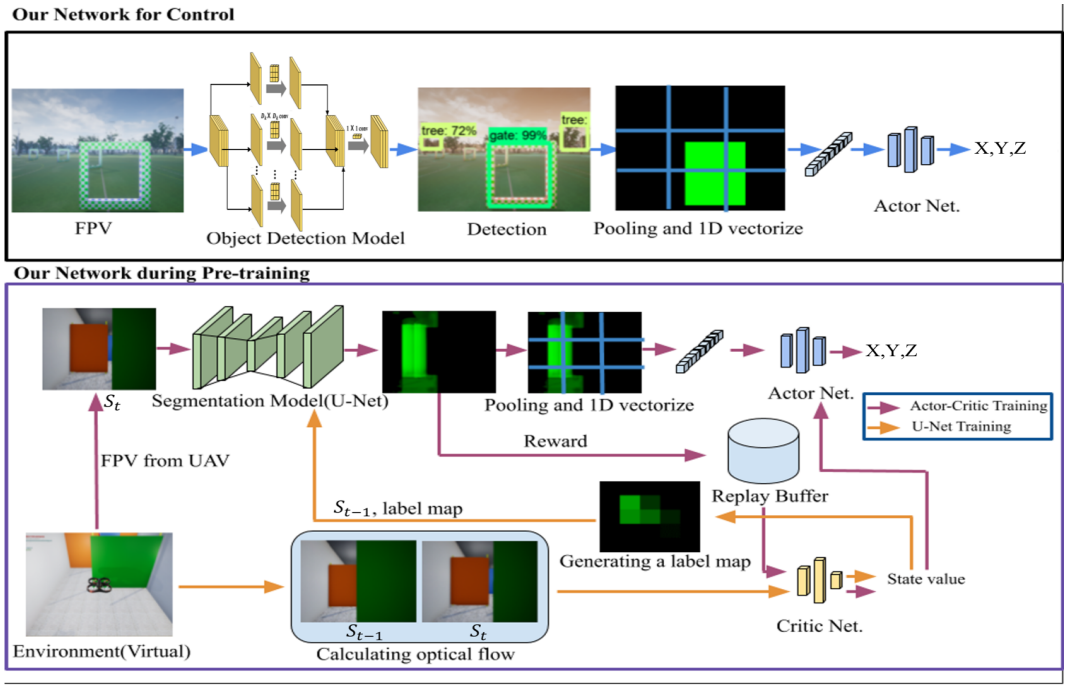

Fig. 2: Flow diagram of training/testing for our drone navigation system. For the pre-training of an actor-network in a relatively relaxed environment, we have followed the scheme in our preliminary work [1]. That is, an optical flow calculated with two sequential images from the environment was used to generate a label map (U-Net training). U-Net's output was used as a reward for the Critic network as well as input to the Actor-network through the pooling and vectorization process (Actor-Critic training). The size of the experience replay buffer was 5×10^4 for our experiments. During testing, an object detection network and the actor-network were used for controlling the drone.

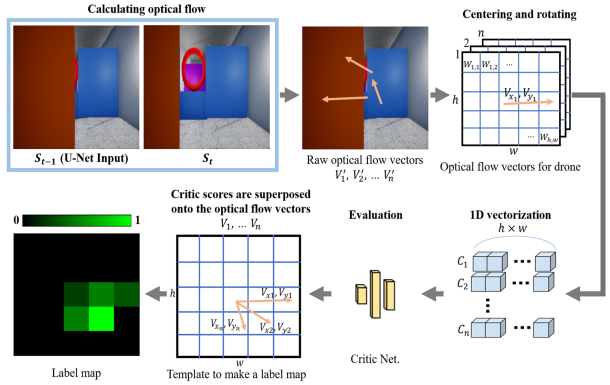

Fig. 3: Process of generating a label map for the segmentation model. The estimated raw optical vectors between two frames are used in calculating optical flow vectors for drone control using the centering and rotation operations. Then, n number of windows are flattened to 1D having to dimension $h \times w$, and the 1D vectors are fed into the critic network to estimate the predicted rewards for the given directions. The final label map is generated by superposing the critic scores onto these vectors. Note that h and w were both set to 5.

label map could be 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 a coefficient of η, 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:

$$C_n = \{\bar{W}_{i,j}, ..., \bar{W}_{h,w}\} \quad (1)$$

where $\bar{W}_{i,j}$ is 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 Ψ to receive its predicted reward. The label map can be produced by filling a zeros array with the output values of Ψ corresponding to zones indicated by V_n of every sample C_n. Note that filling the windows with the values from Ψ suggests each window in a label map will have the predicted reward since the critic network learns to predict the reward. 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 Figure 4. Given S_{t-1} as an input and the corresponding label map described above as a target, our segmentation model

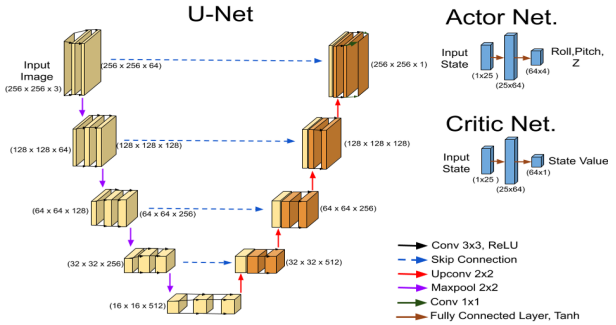

Fig. 4: Specifications of our U-Net, Actor network and Critic network structures. Note that it receives 256×256×3 by rescaling the raw image and outputs 256×256×1 with Sigmoid function at the last layer, containing the segmentation of the input image in term of reward. During training, all three networks such as U-Net, Actor, and Critic networks are required, whereas only U-Net and Actor-network are utilized during testing. All 3 networks use *Adam* optimizer with $\beta = (0.9, 0.99)$ and *learning rate* $= 1e^{-4}$. Their weights are initialized using *Kaiming He* method, and the bias units are set to 0.

learns simultaneously with actor-critic networks. As the RL model performs well, the segmentation model improves, as well.

2) Control using Reinforcement Learning: To overcome a known issue with the high dimensional data for policy gradient, we adopt a segmentation model that is to compress the information while preserving the useful features as mentioned earlier. Let f_{seg} and θ_{seg} denote the segmentation model and its parameter, then s_t for training our actor π and critic network Ψ are given as:

$$s_t = \{\bar{W}_{i,j}, ..., \bar{W}_{h,w}\} \tag{2}$$

where W is the set of the values in the zones separating the output of f_{seg}, as equation 1, 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 π 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:

$$r_t = \begin{cases} \frac{1}{m}\sum_{i=1}^{m} \bar{W}_i|V_t & if \text{ not collides} \\ -1 & \text{otherwise} \end{cases} \tag{3}$$

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 4. Action a_t produced by the actor network π using s_t in equation 2 is as follow:

$$a_t = \pi(s_t|\theta_\pi) \tag{4}$$

where θ_π stands for the parameter of π. With these s_t, a_t, r_t, s_{t+1} defined above for a transition, our actor and critic networks make a typical optimization step in RL by utilizing an experience replay buffer, that saves transitions up to the predefined size of 5×10^4 as shown in Figure 2. The role of the Critic-network is to assess an action from an Actor-network, as well as U-Net for their optimization. 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.

Initially, we used $moveByRPYZ$ for the control in **qualification round**. However, we have found that function $moveBySplineAsync$ provides a more reliable and faster control scheme. Therefore, we have changed all of the control functions into $moveBySplineAsync$ for the **final round**. The equation we used to calculate X, Y, and Z position for the function without the model's re-training is as follow:

$$\begin{aligned} q_x &= o_x + cos(Yaw) * (p_x - o_x) - sin(Yaw) * (p_y - o_y) \\ q_y &= o_y + sin(Yaw) * (p_x - o_x) + cos(Yaw) * (p_y - o_y) \end{aligned} \tag{5}$$

where p_x and p_y stand for the point that x and y vectors were initially pointing, and o_x and o_y are the origin of the vector. q_x and q_y are the rotated vector given yaw angle Yaw. For 4 directions, such as Forward, Backward, Left, and Right, q_x and q_y have been calculated for every direction each time a control command has to be made. A combination of these vectors is used to make the drone move. Since controlling roll and pitch leads to make the drone move to either forward/backward or left/right respectively, we were able to convert the pre-trained actor model's output into the command for $moveBySplineAsync$. Once the drone moves by setting a position in X, Y, and Z, we have set a threshold parameter $dist$ for the termination of the command. That is, the movement command stops when the distance between the drone's current position and destination is less than $dist$.

During our preliminary study, it was shown that our actor-network was able to navigate in a diverse environment without retraining when the segmentation model could perform well. However, for the given competition environment, there was a limitation in segmentation for tier 2 and 3, where the environments are more challenging by having various objects. Therefore, instead of using the segmentation model, we thought that the promising candidate for generating a label map for actor-network is a recent object detection model.

C. Rule-based Control scheme

In addition, we have used the function $moveOnSplineAsync$ to move the drone near to the next gate before starting vision-based control. That is, when the drone passes n_{th} gate G_n, we move the drone to a

position Pos_{n+1} between the current position Pos_n of the drone and G_{n+1} using η. This was because such a distance was enough for the drone to catch the next gate in FPV scene, though there are noises added in the ground truth. Once the drone reaches the point, the vision-based control scheme mentioned above begins until it passes the G_{n+1} appearing. Then, these processes are iteratively made until the drone reaches the goal. The formula we use to calculate the position Pos_{n+1} is as follow:

$$Pos_{n+1} = Pos_n + (G_{n+1} - Pos_n) * \eta \tag{6}$$

where η was a predefined parameter to determine how close the drone should move given the next gate's position with noise. The equation is used to calculate the position for X, Y, and Z, respectively. Also, yaw control is carried out to ensure the gate is always located around the center of the FPV. That is, when the gate is detected near the sides of the image, we stop the drone and control yaw until the gate is at the center. All of these commands for vision-based control is carried out by function $moveOnSplineAsync$. As there are noisy ground truth given for tier 2 and 3, this operation using the ground truth stops after the gate is detected in a FPV scene. That is, the vision-based control scheme depicted in previous sections starts to maneuver the drone.

D. Evolution Algorithm for Optimizing Racing Parameter

For the optimization of the parameter such as vel, acc, and $dist$ in the **final round**, we have used genetic algorithms [10] because the algorithm carries out its optimization based on a random search with a sophisticated policy. By defining $Airsim$ as a function running the race and returning the termination time, the objective function we define for the minimization using genetic algorithm is as follow:

$$Airsim(w^1_{vel}, w^1_{acc}, w^1_{dist}, .., w^n_{vel}, w^n_{acc}, w^n_{dist}) \tag{7}$$

where n is a total number of gates, and w is variable that the genetic algorithm has to optimize. Note that there are $3n$ variables in total. Using w, function $moveOnSplineAsync$ is called with its parameter set to w_{vel} and w_{acc}. For instance, if the drone is moving just passed first gate and is going toward the second gate, all of $moveOnSplineAsync$ call is carried out with the parameter w^2_{vel} and w^2_{acc} with the threshold set to w^2_{dist} until the drone passes gate 2. The process was terminated either when the drone arrives at the last gate or when the lap time becomes higher than the best lap time, which leads each generation to evolve faster.

II. Experimental result

A. Hardware, Software, and Equipment

For the **qualification round**, a workstation equipped with Intel i7 3.4 GHz CPU and an Nvidia Titan X was used for both training and testing. Python 3.6, Tensorflow 1.12.0 [2], and OpenCV 3.4.1 [11] were used for our network programming in Ubuntu 16.04 OS. For the **final round**, 2 workstations with each of them equipped with Intel Xeon E5-2600 v4 CPU and 8 Nvidia Titan X were used for the optimization of the racing parameter using the genetic algorithm.

	FPS	Complete/ Non-complete	Missing	Collision	Lag
SSD Mobilenet	27	9/1	2.3	5.4	138
SSD Inception	21	7/3	4.1	7.2	172
FRCNN Resnet	9	9/1	4.72	2.4	220
RFCN Resnet	8	5/5	8.4	2.2	254

TABLE I: Success ratio, average numbers of the missing gate, collision, and lag time when using each object detection model for tier 2 in **qualification round**. Success ratio was measured by calculating whether or not the drone could reach the goal during 10 trials. Among 4 detection models, SSD Mobilenet V1 and FRCNN Resnet 50 showed the highest ratio, whereas RFCN Resnet 50 showed the lowest ratio. Here η was set to 0.5

	FPS	Complete/ Non-complete	Missing	Collision	Lag
SSD Mobilenet	26	9/1	1.7	3	183
SSD Inception	21	8/2	3.4	5.2	214
FRCNN Resnet	9	9/1	2.2	2	275
RFCN Resnet	9	7/3	4.2	1.4	290

TABLE II: Success ratio, average numbers of the missing gate, collision, and lag time when using each object detection model for tier 3 in **qualification round**. η was set to 0.5

B. Experiment on object detection models

This experiment is to see what is the impact of object detection models for generating a label map as shown in Figure 2. Four latest models, such as SSD Mobilenet, SSD Resnet, Faster RCNN Resnet, and RFCN Resnet, were tested during the **qualification round**. The performance for each model has been measured in terms of FPS, success ratio, and average numbers of the missing gate, collision and lag time during 10 times of trials. Table I and II illustrate these performances for Tier 2 and Tier 3, respectively. It was found that the average numbers of the collision were high when the execution time was faster such as SSD models, indicating that sending command by $movebyRPY$ could become unstable when its frequency is higher than a certain number. Interestingly, when it comes to the heavier models such as FRCNN and RFCN, average numbers of the missing gate went higher, though the collision was low. This suggests that sending command too slowly could lead to an undesired trajectory.

It was shown that there are less missing gates in Tier 3 for **qualification round** in general with all detection models, because there was no variation in the shape of the gate. However, since there are diverse shapes of gates in Tier 2, all of the models show higher numbers of collisions and failures to reach the goal than Tier 3 as shown in Table I and II.

	$Population$	$N_{parents}$	$N_{generation}$	$Mutation$
Tier2	25	4	9000	2
Tier3	30	4	9000	2

TABLE III: Specification of parameters used for racing optimization using genetic algorithm in the **final round**.

	Complete/ Non-complete	Avg Num. of gate missed	Avg Num. of collision
eta=0.3	6/4	7.8	11.2
eta=0.5	9/1	2.3	5.4
eta=0.7	2/8	9.2	27.1
eta=0.9	0/10	11.7	41.8

TABLE IV: Performances measured by varying η in Tier 2 in **qualification round**. Here, SSD-Mobilenet was used in experiments.

	Complete/ Non-complete	Avg Num. of gate missed	Avg Num. of collision
eta=0.3	7/3	5.2	6.2
eta=0.5	9/1	1.7	3
eta=0.7	3/7	14.1	26.7
eta=0.9	0/10	21.7	54.4

TABLE V: Performances measured by varying η in Tier 3 in **qualification round**. SSD-Mobilenet was used in experiments.

C. Experiment on parameter η

This experiment was designed to see how different settings of η could affect the model's performance. Table IV and Table V show the experimental results of η for Tier 2 and 3 during the **qualification round**. Given that there are noises added in the ground truth, moving the drone too close to the next gate by setting η to higher than 0.5 leads to a significant failure of the control. Also, smaller success ratios for η lower than 0.5 indicates that moving the drone to a position closer to the next gate helps improve the entire system performance for vision-based control even if there are noises added in the ground truth.

D. Experiment on optimizing racing parameter

For the optimization of racing parameters such as vel and acc during the function call, as described in Equation (7), we have set an environment running multiple Airsim processes in parallel. Also, the learning parameter of the genetic algorithm, such as $population$(number of airsim processes running in parallel), $N_{parents}$, and $N_{generations}$(number of optimization step) are differently set for Tier 2 and 3, as shown in Table III. After the optimization steps, the best parameter sets for each tier have shortened arrival times from 140 to 53 sec for tier 2 and 160 to 68 sec for tier 3.

REFERENCES

[1] S.-Y. Shin, Y.-W. Kang, and Y.-G. Kim, "Reward-driven u-net training for obstacle avoidance drone," *Expert Systems with Applications*, p. 113064, 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S095741741930781X

[2] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: http://tensorflow.org/

[3] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[4] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[5] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2818–2826.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[8] J. Dai, Y. Li, K. He, and J. Sun, "R-fcn: Object detection via region-based fully convolutional networks," in *Advances in neural information processing systems*, 2016, pp. 379–387.

[9] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, ser. IJCAI'81. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[10] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms: concepts and applications [in engineering design]," *IEEE Transactions on Industrial Electronics*, vol. 43, no. 5, pp. 519–534, Oct 1996.

[11] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000.