Game of Drones - Tier 2 and 3

Team USRG

***Abstract*— Team USRG participated in the tier 2 and 3 of Game of Drones - Competition at NeurIPS 2019. Focusing on the perception, this challenge required the drone to perceive the gate from the facing forward RGB camera. Utilizing deep-learning based algorithm to detect the gate and vision based control algorithm to control the drone, our team was able to complete the tier 2 challenge track passing the whole 21 gates in 81.191 seconds, and complete the tier 3 challenge track passing the whole 22 gates in 110.730 seconds.**

I. Introduction

Given the camera images from the RGB facing forward camera and the ground-truth pose of the drone, tier 2 challenge required the drone to complete the track using the perception related algorithm. Hence, our team divided this challenge into two parts, which were the perception part and control part. For the perception problem, we utilized the deep-learning based algorithm to detect the gates. On the control part, vision based control was used to maneuver the drone through the gate.

II. Perception Part

In the perception part, the detection of the gates in the *AirSim* environment was not a trivial task due to many reasons such as severely differing lighting condition, occlusion of the gates, nonconformity of the gates and etc. Therefore, the pixel based algorithm, such as color or shape detection, might not be suitable for this environment. Inspired by the work of Jung et al. [3],[2],[4], we considered utilizing deep-learning based algorithm to address this challenge effectively.

In this racing, *MobileNetSSD*, thanks to its fast inference capability, was employed to robustly detect the gates in real time. *MobileNetSSD* is basically a combination of neural network and object detection network. Therefore, it needed train data: the RGB images of the gates in different environment which were collected from several flights in the simulator. Assuming that the drone could see the gate in the sequential order it needed to navigate through during the race, we labeled the train data by creating the bounding box only around the nearest gate in the image in the case of the presence of multiple gates in the same image as shown in Fig. 1.

To make our detection model more robust, the collected train data were augmented by several methods such as adjusting brightness, flipping horizontally, random cropping and etc. Then, those augmented data were used to train the *MobileNetSSD*. The data augmentation and training the network were done in the *Tensorflow* framework [1].

The neural network model outputted the location of the gate by specifying the coordinates of the vertices of the bounding box as shown in Fig. 2 and 3. Ideally, the bounding box should tightly fit with the gate such that the coordinates of the vertices of the bounding box could be used to determine the center of the gate. The center of the gate, which is the point that the drone should fly through, was later fed as the reference point to the controller.

Fig. 1. Labeling the train data: in the case of multiple gates presenting in the same scene, we only labeled the nearest gate to the drone.

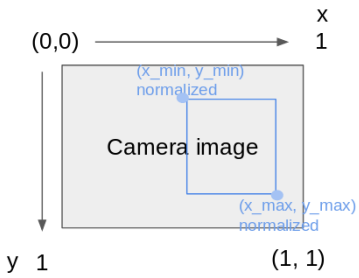

Fig. 2. Output of the *MobileNetSSD*: two coordinates of the top-left of the bottom-right vertices of the bounding box. Note that the origin of the image frame is on the top-left.

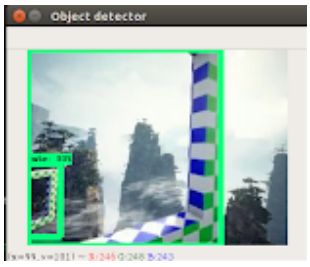

Fig. 3. Output of the *MobileNetSSD* in the real scene.

Although the employed neural network model was capable of marvelously detecting the gates throughout the race, the challenges of this method was that there can be several gates in the same scene obtained from the facing forward RGB camera. Detecting more than 1 gate at a time (false positive), Fig. 3, for instance, the drone could not be able to properly track the next target gate. Therefore, our algorithm was

designed assuming that the drone should navigate through the gates in sequence such that it always tracked the nearest gate first in the case of multiple detection of the gates.

III. Control Part

In the control part, we designed a hybrid control algorithm: position control and velocity-yaw control. Given the noisy ground truth of the gate poses, the approximate location of the gate with respect to the world frame. Therefore, in the scenario that, the drone could not detect the gate, we used position control to maneuver the drone to the approximate location of the gate. On the other hand, if the drone could detect the gate, then it followed the velocity-yaw control algorithm.

A. Position Control

In the racing, we kept track how many gates the drone has passed to determine the next target of the drone. We rely on the noisy ground truth pose to be the target point of our position control using airsimneurips API. Since the ground truth poses of the gate were unreliable, we calculated the weighted average of the current drone position and the noisy ground truth as illustrated in Fig 4. to ensure that the drone would be in the position that it could detect the next target gate using the perception algorithm described in part II.

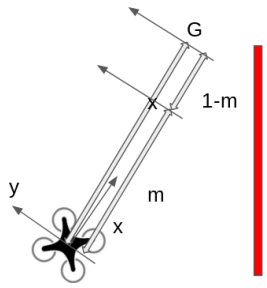

Fig. 4. Position control: the target point was calculated using the current position of the drone and the noisy ground truth pose, where G is the noisy ground truth pose of the gate and m is the weight parameter.

B. Velocity-yaw Control

Inspired by the work of Jung et al. [3], [2], [4] our team made use of the classical PD-controller to command the linear velocities in 2 axes, namely, Y, Z with respect to the drone frame, and yaw angle of the drone. Then, the velocity of X-axis is set proportionally to the distance of the drone and the target gate obtained from gate depth estimation algorithm mentioned in part III-B.1. The velocity and yaw commands from the controller were then inputted to the AirSim API to maneuver through the series of the gates.

1) Gate Depth Estimation: As we wanted to set the X-axis linear velocity to be proportional to the distance from the target gate, knowing the distance of the drone from the gate was crucial to our approach. From the output of *MobileNetSSD*, we could obtain the gate's pixel width and pixel height in the image frame. We modeled the relationship between the distance from the drone to the detected gate, called depth, and the size of the detected gate using the exponential model as shown in (1).

$$\hat{y} = Ae^{Bx}, \tag{1}$$

where x is the measured size of the gate, $\hat{y}$ is the estimated depth, A and B are the parameter to be obtained from the collected data.

To obtain the parameter in equation (1), we collected various of the gate size data with the distance by flying the drone heading straight toward the gate. Then, we used the regression technique to fit the model (1) to the collected data to obtain the parameter A and B that best fit the model with the collect data.

The estimated depth obtained from this model was then used to estimate the position of the gate and to calculate the desired control command values of X, Y, Z linear-velocity and the yaw angle of the drone.

2) Yaw Angle Control: For the drone to fly through the center of the gate, the drone heading vector needed to align with the vector from the drone to the center of the gate as shown in Fig 5. Since the ground truth pose of the drone was given, we could compute the desired yaw angle using the estimated distance from the drone to the gate and the displacement of the center of the gate and the image frame in y-axis. Both estimated distance and position errors were calculated using the information from RGB image and converted to the real distance in the simulator world. For yaw angle control, the controller tried to minimize the angle of drone heading vector and drone-gate vector as depicted on the right side of Fig 5.

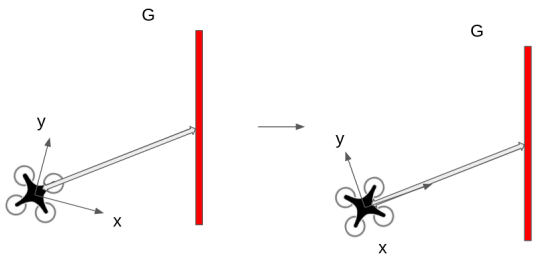

Fig. 5. Yaw Angle Control: (Top-view) X-axis of the drone was the camera direction and also drone heading vector.

3) Velocity Control: The controller obtained a reference point, which was the coordinate of the center of the detected gate, from the gate detector stated in part II. Trying to align the the center point of the gate to the center point of the image frame as shown in Fig. 6, the controller minimized the position error, calculated from the pixel distance between the two points in the image frame. Desirably, the displacement error of the two points should approach to zero to ensure that the drone flied through the center of the gate.

IV. Drone Overtaken Part

In tier 3, we also needed to consider the presence of the opponent drone and try to outrace it without crashing.

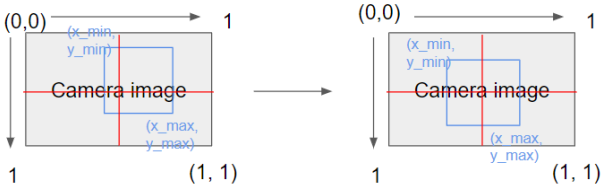

Fig. 6. Controller method: (left) the image frame when the detector detected the target gate, (right) the image frame after controller try to align the two points.

A. Drone Detection

The detection of the drone was done in the same manner with the gate detection explained in part II. However, using the same network model to detect both the opponent drone and the gates caused imbalanced label problem. Therefore, we considered using two networks for detection: one for drone detection and one for gate detection.

B. Strategy

We considered three scenarios from the RGB input image.

First, only the drone was detected in the image. In this case, we separated the image into two area (altitude separation) and set the collision risk area. Then, if the drone detected in the considered area, do the altitude separation. If the opponent drone is detected higher than the center of the image, our drone will lower the altitude as shown in Fig7. Otherwise, the drone will raise the altitude. Second,

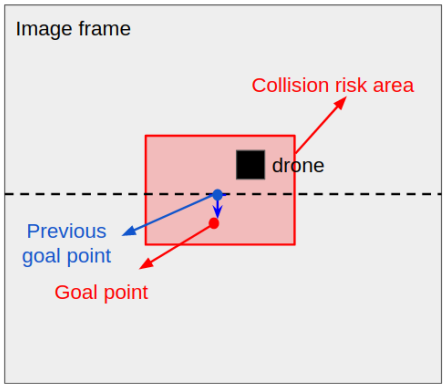

Fig. 7. Only drone was detected

only the gate was detected in the image. In this case, we could consider this challenge as tier 2 challenge, hence, the same control algorithm could be employed. Last, both the drone and the gate were detected, we re-select the area that drone would fly through and resetting the drone's goal point. Calculating the area divided by opponent drone in the gate, we selected the largest area and set the center point of the selected area as a new goal point of the drone.

V. Result

For tier 2 challenge, using the perception method explained in part II and control method mentioned in part III, we were able to complete the track in 81.191 seconds. For

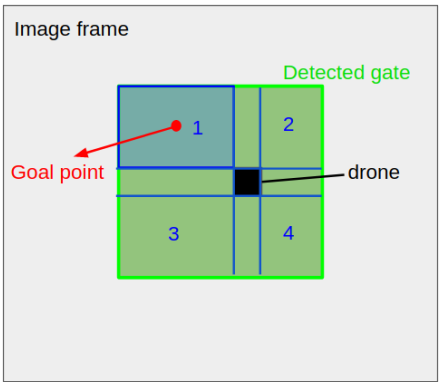

Fig. 8. Both the drone and the gate were detected

tier 3 challenge, we could complete the track in 110.730 seconds.

References

[1] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[2] Sunggoo Jung, Sungwook Cho, Dasol Lee, Hanseob Lee, and David Hyunchul Shim. A direct visual servoing-based framework for the 2016 iros autonomous drone racing challenge. *Journal of Field Robotics*, 35(1):146–166, 2018.

[3] Sunggoo Jung, Sunyou Hwang, Heemin Shin, and David Hyunchul Shim. Perception, guidance, and navigation for indoor autonomous drone racing using deep learning. *IEEE Robotics and Automation Letters*, 3(3):2539–2544, 2018.

[4] Sunggoo Jung, Hanseob Lee, Sunyou Hwang, and David Hyunchul Shim. Real time embedded system framework for autonomous drone racing using deep learning techniques. In *2018 AIAA Information Systems-AIAA Infotech@ Aerospace*, page 2138. 2018.