The 2019 International Workshop on Research, Education and Development on Unmanned Aerial Systems
(RED-UAS 2019)
Cranfield University, Cranfield, UK

Reinforcement Learning for Autonomous Aircraft Avoidance

Choo Wai Keong[1], Hyo-Sang Shin[1] and Antonios Tsourdos[1]

***Abstract*— Effective collision avoidance strategy is crucial for the operation of any unmanned aerial vehicle. In order to maximise the safety and the effectiveness of the collision avoidance strategy, the strategy needs to solve for choosing the best action by taking account of any situation. In this paper, the traditional control method is replaced by a Reinforcement Learning (RL) method called Deep-Q-Network (DQN) and investigate the performance of DQN in aerial collision avoidance. This paper formulate the collision avoidance process as a Markov Decision Process (MDP). DQN will be trained in two simulated scenarios to approximate the best policy which will give us the best action for performing the collision avoidance. First simulation is head-to-head collision simulation following with head-to-head with a crossing aircraft simulation.**

I. INTRODUCTION

Unmanned Aerial Systems (UAVs) have become major research in the aerospace industry. The difference between UAVs and manned aircraft is that the operator of the UAVs is operating on the ground, not knowing the actual situation happens at the surrounding of UAVs. As the number of UAVs increases, the chances for the collision will also increase, hence UAVs must be equipped with a collision-avoidance system in order to maintain the safety of the airspace. Traditionally, one can utilise the Traffic Collision Avoidance System (TCAS) and develop avoidance algorithm [1]. Geometric approaches is another way to fulfill the avoidance requirement where it uses the information of the line of sight and relative velocity and develop sets of logic to control the UAVs [2]. A probabilistic method allows UAVs to carry out proper actions according to the probability of collision in the future time [3].

One way to perform avoidance is to utilise Reinforcement Learning (RL) with Deep Neural Networks (DNNs). The increase of computational capability of the modern computer allows us to gain success in using DNNs [4]. Reinforcement Learning (RL) has also been proved to be a good solution to various robotics and control problems. The adoption of DNNs by RL has greatly improved the learning process of good policy and even carry out tasks like a human. Reference [5] proposed a RL technique called Deep Q Network (DQN) where two sets of DNNs are used to approximate Q-value. It is also shown that DQN can outperform human in Atari Games. Although it can out perform human in games, but the questions of whether it is feasible in carrying out complex physical task like avoiding multiple objects and how good it can perform in such tasks are still not yet answered. While collision-avoidance is different from playing Atari Games,

[1]Centre for Autonomous and Cyber-Physical Systems, School of Aerospace, Transport and Manufacturing, Cranfield University, MK430AL Cranfield, UK; wai-keong.choo@cranfield.ac.uk

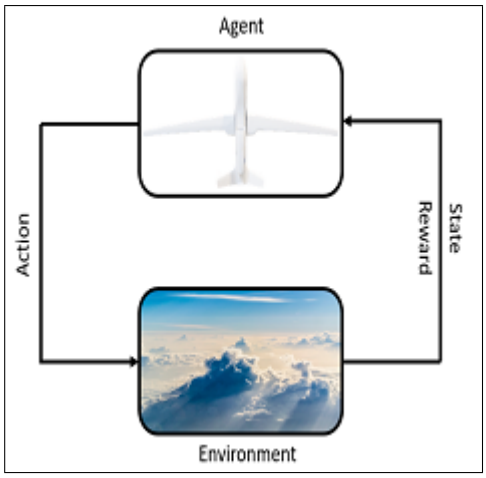


Fig. 1: Agent-Environment Interactions

value iteration of the DQN is still a feasible solution method to our problem.

In this paper, DQN will be utilised to perform collision avoidance where DQN can intelligently control the UAV from colliding with single up to multiple aircrafts in the simulation environment. As shown in figure 1, the agent (UAVs) will interact with the environment, and an optimum policy will then be learned from the interaction between the agent and the environment. Two different scenario will be simulated in this paper, the first simulation is the typical head to head collision, the second is head to head with another crossing aircraft. All the interactions are then stored as experiences to help train the agent. The following of this paper will be divided into these section 1) Formulation of Problem 2) Background Knowledge 3) Experiment 4) Results 5) Conclusion.

II. FORMULATION OF PROBLEM

Head to head collision is one of the typical presentable situations that every UAV or aircraft have to avoid. For this case, we proposed a model-free Deep Reinforcement Learning method called DQN to learn how to avoid the incoming intruding aircraft. To guarantee the collision, both UAV and the aircraft is placed in the same latitude flying towards each other, which is shown in figure 2. Both UAV and aircraft are described using the famous Constant Velocity model which is shown as below:

$$x_k = F_{cv} x_{k-1} \tag{1}$$

where:

$$x = \begin{bmatrix} x & \dot{x} & y & \dot{y} \end{bmatrix} \tag{2}$$

978-1-7281-6600-1/19/$31.00 ©2019 IEEE

$$F_{cv} = \begin{bmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3}$$

In the simulation, it is assumed that the information of the intruding aircraft is available to the UAV at all time. In each time step of the simulation, UAV will make an action according to the surrounding information that it can get. Only the UAV can carry out an action while the intruding aircraft will not make or do any action. During each beginning of a simulation, the UAV and the aircraft is located at their pre-defined initial position with their initial velocity. The relative distance in x-direction and y-direction between UAV and intruding aircraft will be measured and gave to the UAV at each instant of simulation where these measurements are referred as state or environment state (s_t) in this paper. UAV uses these pieces of information to come out with an action a_t. Possible action that UAV can choose is changing the heading angle, which can be found in Table I. A reward r_t will be given according to the result of taking that action. The state s_t, action a_t and reward r_t will be used to train the UAV to avoid the incoming aircraft. The simulation only ends when one of the following condition is meet:

- Collision: Collision occurred between the UAV and intruding aircraft.
- Safe: No collision is happened between the UAV and intruding aircraft.

III. Background Knowledge

1) Markov Process: States that contain all relevant information of the current environment but never the complete path or history is to call Markov or having Markov properties. For example, a checker only uses the current configuration to come out next move, because only the current situation or state is matter to the crucial decision. The respond of an environment at time $t+1$ to the action taken at t, in most case, depends on what happened earlier. It can be represented by a complete probability distribution:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, r_{t-1}, ..., s_{t-n}, a_{t-n}, r_{t-n}, s_0, a_0, r_0\} \tag{4}$$

The states is to be said to have Markov Properties when the environment's respond at $t+1$ is only depend on the state and action at t and this can be only represented as

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\} \tag{5}$$

In an easy way of saying a state is having a Markov property or is a Markov state happens when if and only if equation (5) is equal (4) for all s', r and histories. If the statement of Markov is true for the environment, it allows us to predict the next state and expected reward for the next states given the current state and action. Then one can predict all future state and reward from only knowledge of current state and action as well as would be given the complete history up to the current point. This will be beneficial to UAV as it can have the best policy for choosing an action as a function of Markov state which will result in a policy that just as good as the best policy for choosing an action as a function of a complete history. A reinforcement learning task that fulfills the Markov properties is called a Markov decision process (**MDP**). So, at given any state and action, s and a the probability of each possible next state s' is:

$$Pr\{s_{t+1} = s' | s_t = s, a_t = a\} \tag{6}$$

They are also call transition probability. MDP helps us to know the probability of the next state to improve value function.

2) Value Function: Most of the reinforcement learning methods are based on estimating value function. Value function allows to determine how good it is for the agent to be in a given state. Expected return reward allows us to determine the performance of the agent. Value-function is designed with respect to policy. Here, the value of taking an action a in a state s under a policy π is defined as the expected return of rewards collected by starting from s and taking action a. Hence it is denoted as $Q^\pi(s,a)$ where it is called action-value function or Q-value and it is shown at (7) where γ is the discount factor.

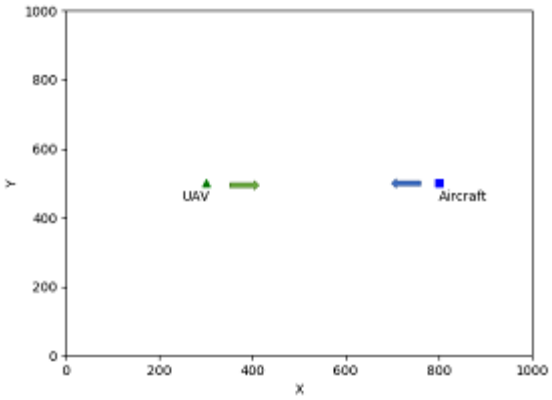
$$\begin{aligned} Q^\pi(s,a) =& E_\pi\big[R_t | s_t = s, a_t = a\big] \\ =& E_\pi\big[\sum_{t'=t}^{T} \gamma^{t'-t} r_{t'} | s_t = s, a_t = a\big] \end{aligned} \tag{7}$$

Then the optimum action-value function $Q^*(s,a)$ is defined as the maximum expected return achievable by following any policy. The optimum action-value function is defined as $Q^*(s,a) = max_\pi E\big[R_t | s_t = s, a_t = a\big]$. This optimum action-value function obeys an important condition know as **Bellman Equation**. As mentioned in [6] that if the optimal value $Q*(s',a')$ of sequence s' at the next time-step was know for all possible actions a', then the optimal strategy is to select the action a' that maximise the expected value of $r + \gamma max_{a'} Q*(s',a')$. So, optimum action-value function will be

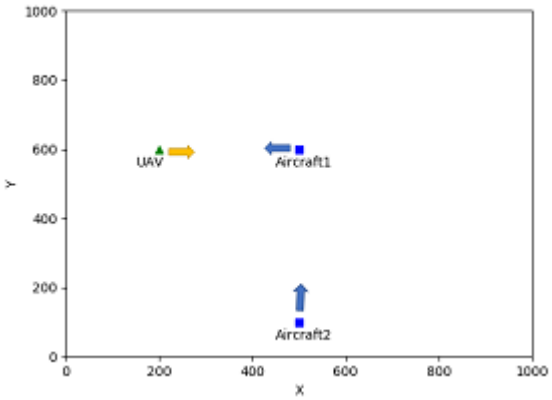
$$Q^*(s,a) = E\big[r + \gamma max_{a'} Q^*(s',a') | s,a\big] \tag{8}$$

3) Replay Memory: Memory replay has been constantly used to improve the performance of the Reinforcement Learning(RL) method. Paper like the playing Atari game [6] shows a good example of utilising the replay memory. Replay memory allows the RL to store the exploration & exploitation of the environment. A minibatch of this memory will then be randomly selected to update the parameters of the agent. The purpose here is to utilise the replay memory to break the temporal correlations and as well as to improve the efficiency of computational calculation [7]. The size of the memory also plays a vital role in RL, for it can substantially affect the learning speed of RL [8].

4) Deep Q-Network: DQN has been one of the popular milestones for the research of the RL methods. It uses iteration to search for an optimal policy to guide the agent. As what has discussed in previous section, agent will carry



(a) Geometry layout for the head-to-head collision simulation.



(b) Geometry layout for the head-to-head collision with crossing aircraft simulation.

Fig. 2: Depiction of UAV collision avoidance problem

out an action a_t given a states_t under a policy π. The ultimate goal is to maximise the reward which can be defined as $R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$. In other word, an optimum action-value function, eq (8) needs to be obtained, which is discussed in the previous section. But obtaining this optimum function poses a great difficulty, hence a common way is to use an approximator to estimate this optimum function. Here a neural network with parameter θ is defined as $Q(s, a; \theta)$ to approximate $Q^*(s, a)$, this is where the word "deep" comes from. Approximator $Q(a, s; \theta)$, called Q-Network can be trained by minimising the sequence of a loss function temporal difference error $L_j(\theta_j)$ which change at each iteration of j,

$$L_j(\theta_j) = E\big[(y_j - Q(s, a; \theta_j))^2\big] \tag{9}$$

where y_j is eq (8) but the $Q^*(s', a')$ is replaced by another neural network $Q(s', a'; \theta^-)$ called Target Network with parameter θ^-, hence $y_i = E\big[r + \gamma max_{a'} Q(s', a'; \theta^-)|s, a\big]$. Optimisation technique call stochastic gradient descent is utilise on the loss function. The gradient of the loss function with respect to the weight is shown below:

$$\nabla_{\theta_j} L_j(\theta_j) = E\big[(r + \gamma max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))\nabla_{\theta_j} Q(s, a; \theta)\big] \tag{10}$$

This approach is considered as a model-free in the sense that the agent does not have any information about the dynamic of the environment and it only obtain the states and reward from the environment. DQN is also an off-policy method as it utilises ϵ-greedy during the training to prevent over-fit.

5) Reward Function: A reward is crucial to the training and the converge of the training process. Here reward function is set such that when the relative distance between UAV and intruding aircraft is less than the safety distance, then it is considered to be a collision case where a -200 will return as the reward value. If both of the UAV and aircraft do not crash, then it is classified as a safe case and a +1 value will be returned as reward else 0 as a reward for nothing happening.

- if UAV crashed $Reward = -200$
- else if UAV not crash $Reward = +1$
- else $Reward = +0$

TABLE I: Parameters

Name	Parameters
Input Layer	States (s_t)
Layer 1	Fully Connected, 200 units
Layer 2	Fully Connected, 400 units
Layer 3	Fully Connected, 400 units
Layer 4	Fully Connected, 100 units
Layer 5	Fully Connected, 10 units
Output Layer	Action (a_t)
Learning rate	0.001
Memory Size	2500
Training Episode	200
Action Space	(a_t)$\in (-10, -5, 0, 5, 10)$
Observation Space	(s_t)$\in (x_{rel}, y_{rel})$

IV. Experiments

In this section, two experimental simulation are performed via computer simulations. In these simulation, a 2-D simulation environment is created with 1000×1000 in x and y-direction. The simulations will be ran for 200 episode, where each episode runs 20 simulations. Rewards are then stored from each simulation and appended into a database. After 20 simulations, rewards are taken from the simulation and returned to be the reward for that episode of the total episode. These episode's rewards will show how good is the agent to avoid the incoming aircraft after going through the training process. The workflow of the DQN can be found in [6].

1) Head-to-Head : The first simulation is head to head collision. The initial position for the UAV is located at $(x_{uav}, y_{uav}) = (300, 500)$ with it's velocity selected from an uniform velocity distribution $(\dot{x}_{uav}) = U(10, 15)ms^{-1}$ while the aircraft with velocity selected from it's uniform velocity distribution $(\dot{x}_{uav}) = U(-10, -16)ms^{-1}$, is located at $(x_{Air}, y_{Air}) = (800, 500)$ is flying towards the uav. This is shown in Figure 2a. As mentioned in the previous section, the simulation ends when either the "Crash" case or the "Safe" case is satisfied. Training parameter of Q-network is given

in Table I. $\epsilon-$greedy strategy is used to choose action and the value is decreased linearly from 1 to 0.01 for every 5 episode.

2) Head-to-Head with Crossing Aircraft: The second simulation is head to head with another crossing aircraft. As shown in Figure 2b, after the first simulation, additional complexity is added into the simulation where another aircraft is trying to fly across the UAV. The overall procedure for this second simulation is almost the same as the first simulation. The additional aircraft that flies vertically upward from it's initial position $(x_{Air2}, y_{Air2}) = (500, 100)$. This additional aircraft will have a velocity which selected randomly from a normal distribution $\dot{y}_{Air2} = U(9, 15)ms^{-1}$. The initial location of UAV and the first aircraft has changed to $(x_{UAV}, y_{UAV}) = (200, 600)$ and $(x_{Air1}, y_{Air1}) = (500, 600)$ while velocity is remain unchanged. Here the UAV not only needs to avoid for the first incoming aircraft but also to learn how to avoid the collision for the second crossing aircraft after passing the first aircraft.

V. Results

Two metrics that are utilised to evaluate the performance are 1) rewards, 2) action-state values (Q-values). Both of these metrics have to be calculated periodically. These values are collected during the simulations and averaged after the success of each 20 simulations. Figure 3 and 4 show the average rewards and the average Q-values collected though out the 200 episodes. As shown in figure 3a, for the first 12 episodes, the agent is experiencing the exploration stage that why the reward is always low. Then it enters the exploration and exploitation stage, where it is the battle between exploration exploitation that why the reward is sometimes high and sometimes low. Then the reward converges to the highest in episode around 30th episode. The simplicity of the simulated problem may be one of the reasons for such fast convergence in the early training episode. This fast convergence may also a result of the decaying of the epsilon (greedy-policy), for this greedy-policy forces the DQN to use the memory replay more often as the time goes. On the other hand, Q-values which approximate the policy has also shown a convergence in the values which can be seen in figure 3b. Although the values converged, the results suffer from the noise, which may be the result of overestimation (a limitation of DQN) [9]. Despite there are no theoretical guarantees of convergence, both the rewards and Q-values are still able to converge. This means that our method is able to train the neural network inside the agent by using the reinforcement signal and stochastic gradient descent in a relatively stable manner. So according to this result, the UAV is able to learn to avoid the incoming aircraft (with a distribution of velocity). Figure 3c shows the early collision avoidance from the UAV. It shows how the UAV is avoiding the incoming aircraft by performing a change in its heading. Figure 3d shows the later stage where the UAV has successfully avoid the incoming aircraft.

Figure 4a shows the averaged return rewards from the second simulation. It can be noticed that the rewards only converge after the 80th episode, which is much later compared to the previous result. Since an extra aircraft is added to the second simulation, hence there is an increase in the level of difficulty to the learning process and more time is required to reach convergence. Figure 4b shows the convergence of the action-states value. From figure 2b, it can be clearly seen that to avoid those two aircraft, flying towards the area between both aircraft is not a good option. Note that figure 4c shows that UAV choose an optimum policy that flies towards the other direction to avoid both aircraft. Figure 4d shows the later stage of simulation where UAV has successfully avoid both aircraft.

VI. Conclusions

This paper investigated the possibility of aerial collision avoidance performance by using DQN algorithm. Results are analysed from the performance of the agent in avoiding single up to multiple aircraft. Although positive results are shown in these two specified environment, more complex Monte Carlo simulations will be carried out in the future. Furthermore, discrete action space will be changed into continuous action space (DDPG) [10] for smooth manoeuvre. Extra actions like acceleration and deceleration will also be added. The reward function will be redefined for better guiding UAV to reach its destination waypoint.

Acknowledgement

This work is based on the results of an MSc project, called "Detect and Avoid for Autonomous Aircraft". The authors would like to express our great gratitude to Jonathan Read from Boeing Defence UK for his valuable advises and helps on the MSc project.

References

[1] S. Ramasamy and R. Sabatini, "A unified approach to cooperative and non-cooperative Sense-and-Avoid," 2015 Int. Conf. Unmanned Aircr. Syst. ICUAS 2015, pp. 765–773, 2015.

[2] J. Seo, Y. Kim, S. Kim, and A. Tsourdos, "Collision Avoidance Strategies for Unmanned Aerial Vehicles in Formation Flight," IEEE Trans. Aerosp. Electron. Syst., vol. 53, no. 6, pp. 2718–2734, 2017.

[3] C.-S. Yoo, A. Cho, B.-J. Park, Y. Kang, S.-W. Shim, and I.-H. Lee, "Collision Avoidance of Smart UAV in Multiple Intruders," 2012 12th Int. Conf. Control. Autom. Syst., pp. 443–447, 2012.

[4] P. H. Chen and C. Y. Lee, "UAVNet: An Efficient Obstacel Detection Model for UAV with Autonomous Flight," 2018 Int. Conf. Intell. Auton. Syst. ICoIAS 2018, pp. 217–220, 2018.

[5] V. Mnih et al., "Human-level control through deep reinforcement learning Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.

[6] V. Mnih, D. Silver, and M. Riedmiller, "Atari Deep Reinforcement learning," Nips, pp. 1–9, 2013.

[7] L.-J. Lin, "Self-improvement Based On Reinforcement Learning, Planning and Teaching," Mach. Learn. Proc. 1991, vol. 321, pp. 323–327, 2014.

[8] R. Liu and J. Zou, "The Effects of Memory Replay in Reinforcement Learning," 2018 56th Annu. Allert. Conf. Commun. Control. Comput. Allert. 2018, pp. 478–485, 2019.

[9] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," 2015.

[10] T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015.

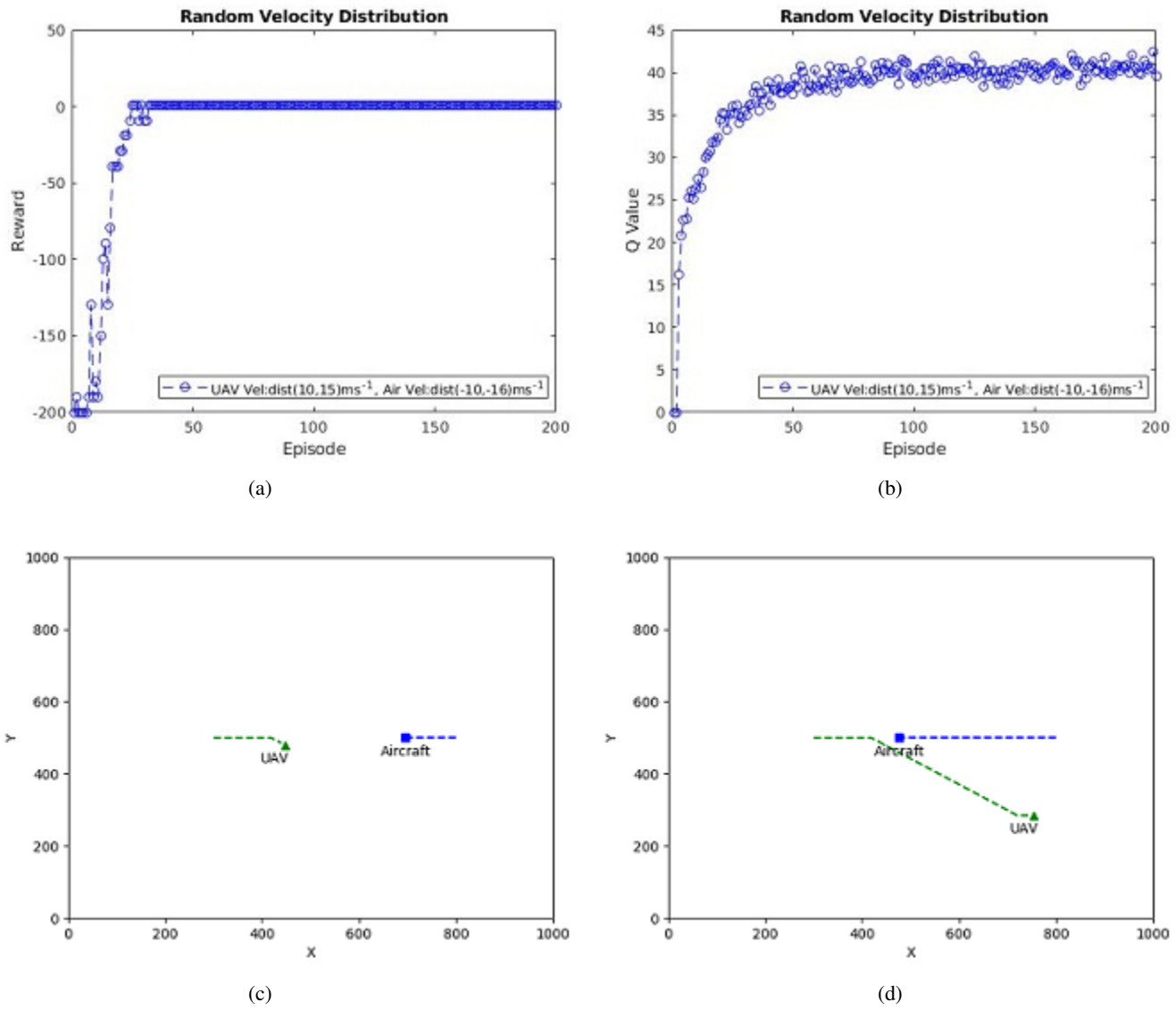


Fig. 3: Averaged return rewards and action-state values (Q-values) collected from the 200 training episode of both simulation. (a) Action-State values from first simulation. (b) Rewards from the head to head collision simulations. (c) Early stage of collision avoidance for first simulation. (d) Geometrical display of successive head to head collision avoidance.

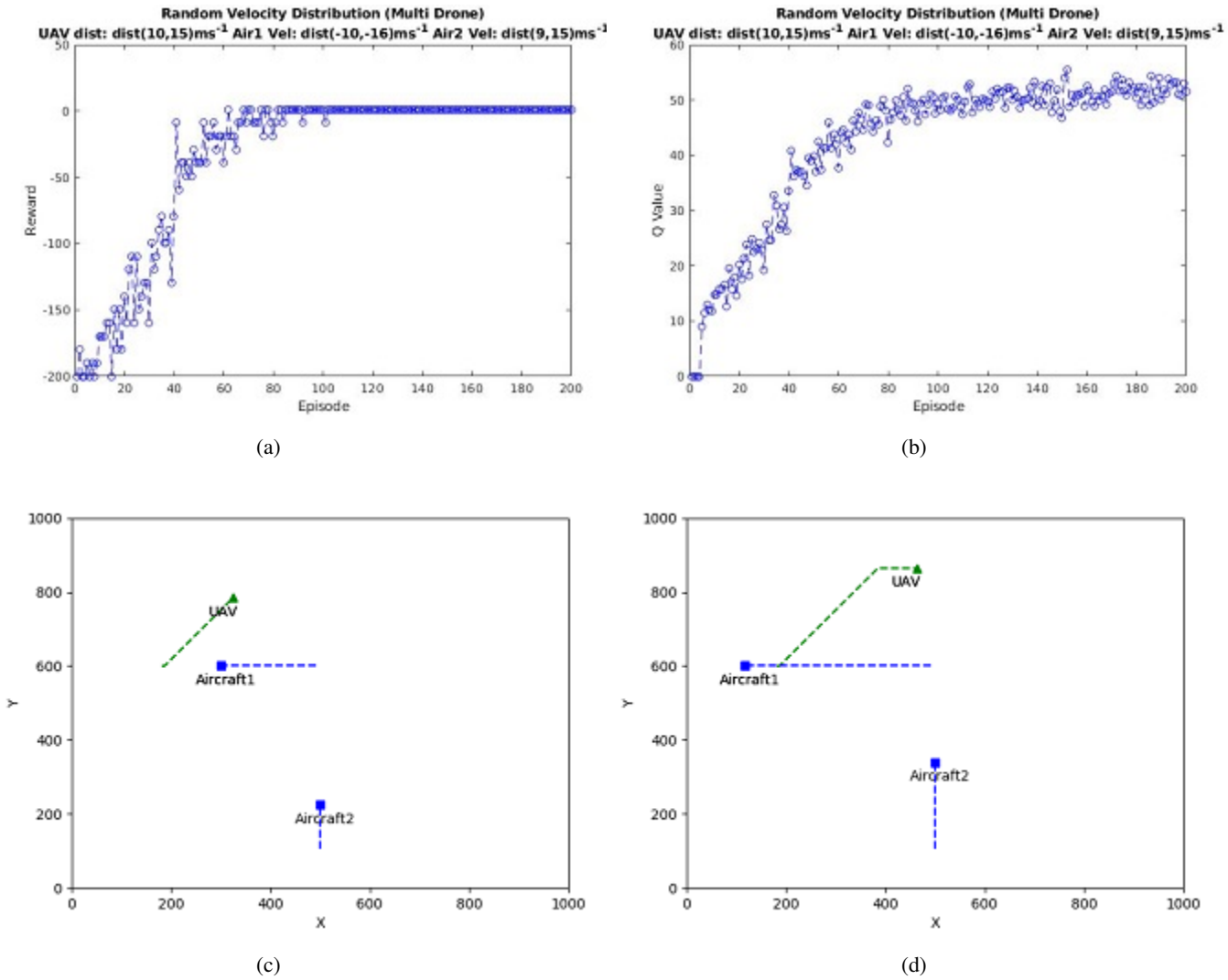


Fig. 4: Averaged return rewards and action-state values (Q-values) collected from the 200 training episode of both simulation. (a) Return episode rewards from second simulation. (b) Action-State values from second simulation. (c) Geometrical display of second simulation at early stage of avoidance. (d) Geometrical display of successive collision avoidance for the second simulation.