2022 19th International Conference on Electrical Engineering, Computing Science and Automatic Control (CCE). Mexico City, Mexico. November 9-11, 2022

Multi-Robot Flocking Control Using Multi-Agent Twin Delayed Deep Deterministic Policy Gradient

Mario Salama Youssef
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
Mario.Salama@student.guc.edu.eg

Nouran Adel Hassan
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
Nouran.Adel-Hassan@guc.edu.eg

Ayman El-Badawy
Mechatronics Engineering Department
German University in Cairo
Cairo, Egypt
Ayman.Elbadawy@guc.edu.eg

***Abstract*—This paper proposes the use of Multi-Agent Twin Delayed Deep Deterministic Policy Gradient (MATD3) to resolve the issues accompanied with Multi-Agent Deep Deterministic Policy Gradient (MADDPG) such as the overestimation bias of a centralized critic when used to solve the multi-robot flocking control problem. A temporal difference error prioritized replay buffer is used alongside to achieve the most efficient learning process. In order to allow the robots to maintain a flock throughout a complex environment, an appropriate reward function was constructed, taking into consideration the following parameters: reaching the goal, maintaining a distance between the agents to ensure a stable connection while avoiding collision between one another, obstacle avoidance, and moving in a specific formation. Results of both algorithms are then compared to test the performance of the MATD3.**

I. Introduction

The popularity of Unmanned Aerial Vehicles (UAV) has been greatly increasing in the past few years due to their many applications and versatility. Some of those applications are geographical mapping, search and rescue, collaborative patrol, and target tracking. The listed applications introduce a new problem to UAV control, which is the flocking problem. In order to implement such applications, multiple UAVs have to be used simultaneously.

Due to the complexity of multi-robot flocking control, several problems arise, such as the prevention of the UAV from colliding with another UAV in the flock as well as obstacles in the environment, maintaining a communication distance with the other UAVs due to the actual limitation of the communication range of the UAV, and the complications associated with having a multi-agent environment. These problems were listed by Reynolds et al. [1] as three basic rules, namely flock centering, obstacle avoidance, and velocity matching.

Flocking is defined as a group of winged animals and is performed by birds for multiple reasons, one of which occurs as a means of defense to protect themselves from predators. Conventional techniques, such as those based on model predictive control [2], fuzzy logic control [3], and distributed control [4], were utilized in the majority of the substantial studies addressing flocking control problems. Deep reinforcement learning methods are used to tackle the flocking problem without the need for exact modeling and complicated control architecture, in contrast to conventional approaches. The reason behind the use of deep reinforcement learning is that when dealing with several agents, traditional reinforcement learning methods like Q-Learning and policy gradient are inefficient.

II. Related Work

Multi-agent reinforcement learning is a relatively new field; hence, there are few available methods. Deep Q-Learning (DQN) was introduced by Dai et al. [5] as a technique for multi-robot task assignment. According to Palmer et al. [6], the use of the experience replay buffer in single-agent deep reinforcement learning was exceptionally efficient and allowed the network to leverage sampled stored transitions. Transitions in multi-agent systems, on the other hand, may be obsolete for some agents as a result of agents updating their transitions simultaneously. To offset this effect, leniency was added to negative policy updates sampled from the Lenient-DQN's experience replay buffer. The results were then compared to a hysteretic-DQN method that demonstrated a faster convergence to the optimal policy of the lenient-DQN.

It was found that the multi-agent deep deterministic policy gradient has a slow convergence rate and limited learning efficiency. Therefore, Zhu et al. [7] introduced the prioritized experience replay multi-agent deep deterministic policy, which is an enhanced version of the MADDPG. A sample was chosen from the replay buffer based on the Temporal Difference (TD) Error. This method led to a faster convergence rate for the algorithm and a more efficient training phase.

While actor-critic approaches might yield state-of-the-art results with multi-agent deep deterministic challenges, there was a limit to the number of agents that could be added. Chu [8] focused on cooperative multi-agent problems to address this issue. As a solution, parameter sharing across agents was proposed. The three presented parameter sharing techniques were actor-critic sharing, actor sharing, and actor sharing with partial critic sharing. The outcomes demonstrated a significantly superior learning rate and memory efficiency compared to traditional multi-agent deep deterministic strategies.

Yan et al. [9] considered the limitations associated with flocking UAV swarms and flocking control issues. These limitations include communication and perception between

978-1-6654-5508-4/22/$31.00

the UAVs. Using a Partial Observable Markov Decision Process (POMDP), the flocking problem was accurately modeled while accounting for these constraints. Alongside the POMDP modeling, Deep Reinforcement Learning (DRL) and proximal policy optimization were used to execute centralized training and decentralized execution. The results illustrated the flock's ability to navigate in a densely packed environment.

This paper tackles the multi-robot flocking control problem in a 2D environment. This will be solved using DRL in conjunction with a multi-agent deep deterministic policy gradient algorithm which combines neural networks with a deterministic policy gradient algorithm. Utilizing a prioritized replay buffer in conjunction with two algorithms improves and expedites the learning process. The flocking problem will be addressed as a Markov decision process and will be initially solved using MADDPG. MATD3 will then be introduced to address the overestimation bias of a central critic and the delayed convergence time associated with the MAADPG. Finally, the various potential solutions to the flocking control problem are compared.

III. Multi-Agent Deep Deterministic Policy Gradient (MADDPG)

MADDPG is a DDPG extension for multi-agent environments. It uses decentralized execution with a centralized training setting. Centralized training planning means that agents only have access to their own observations while being driven by a centralized critic. Consider having N agents with policies $\pi = \{\pi_1, ..., \pi_N\}$ and having policy parameters $\theta = \{\theta_1, ..., \theta_N\}$, critic updates are handled using a one-step look-ahead TD-error:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{\mathbf{x},a,r,\mathbf{x}'}\left[\left(Q_i^{\mu}(\mathbf{x}, a_1, \ldots, a_N) - y\right)^2\right], \quad y = r_i + \gamma Q_i^{\mu'}(\mathbf{x}', a_1', \ldots, a_N')\Big|_{a_j' = \mu_j'(o_j)} \tag{1}$$

Here, $Q_i^{\mu} = (\mathbf{x}, a)$ is the centralized action-value function whose inputs are the states and actions of the agents and its output is the Q value of agent i. The states of all agents are represented as $\mathbf{x} = \{\mathbf{o}_1, ..., \mathbf{o}_N\}$, $\mathbf{a} = \{\mathbf{a}_1, ..., \mathbf{a}_N\}$ are the agents' actions, $\mathbf{r} = \{\mathbf{r}_1, ..., \mathbf{r}_N\}$ are their rewards, γ is a discount factor, and $\mu' = \left\{\mu_{\theta_1'}, ..., \mu_{\theta_N'}\right\}$ refers to the set of target policies that have delayed parameters θ_i'. Then, using the deterministic policy gradient, the actor of each agent is updated using the following equation.

$$\nabla_{\theta_i} J(\boldsymbol{\mu}_i) = \mathbb{E}_{\mathbf{x},a\sim\mathcal{D}}\left[\nabla_{\theta_i}\boldsymbol{\mu}_i(a_i \mid o_i)\nabla_{a_i} Q_i^{\boldsymbol{\mu}}(\mathbf{x}, a_1, \ldots, a_N)\big|_{a_i = \boldsymbol{\mu}_i(o_i)}\right] \tag{2}$$

Where D is the experience replay buffer, which holds successions of tuples $(\mathbf{x}, \mathbf{x}'a, r)$ that record the experiences of all agents. $\mathbf{x}'$ is the agent's new state following the execution of the actions. This method provides the actor with local observations while supplying the critic with the observations of the system as a whole. Such a distribution of scopes decreases the effects of non-stationary learning between agents. While this algorithm solves multiple drawbacks associated with multi-agent problems, it still has issues. MADDPG tends to overestimate the Q-values while learning the Q-function [10].

IV. Multi-Agent Twin Delayed Deep Deterministic Algorithm (MATD3)

Similar to how DDPG was extended to MADDPG, Multi-Agent TD3 (MATD3) [11] extends TD3 to the multi-agent domain. MATD3 also adopts the centralized training and decentralized execution method. It uses the observations and past experiences of all agents to set up two centralized critics $Q_{i,\theta_{1,2}}^{\pi}(\mathbf{x}, a_1, \ldots, a_N)$ for each agent i, hence the term twin. The algorithm then selects the minimum between the two Q-values to avoid the overestimation bias of the Q-function, which was a major drawback of its predecessor, the MADDPG. Choosing the minimum between the two Q-values may result in underestimation, but this outcome is still more desirable than overestimation. The reason behind this is that when overestimation occurs, the overestimated values are selected during the policy update with a high probability which leads to it being used during the next action so that the error propagates to the target update.

On the other hand, when underestimation occurs, the probability of choosing those values during policy updates is low, hence the propagation of the error does not occur. After selecting the Q-function, the policy is generated by maximizing it $\max_{\theta} \underset{x\sim\mathcal{D}}{\mathrm{E}}\left[Q_{\phi_1}(x, \mu_\theta(x))\right]$ [12]. In addition, target policy smoothing is used by clipping the target actions after adding clipped Gaussian noise ϵ to the next actions of all agents during the critic update. This acts as a regularizer for the algorithm, which also addresses the issue accompanying the MADDPG which is generating sudden peaks for some actions. With c being an adjustable parameter, the complete target actions resolve to:

$$a'(x') = \mathrm{clip}\left(\mu_{\theta_{\text{targ}}}(x') + \mathrm{clip}(\epsilon, -c, c), a_{\text{Low}}, a_{High}\right), \quad \epsilon \sim \mathcal{N}(0, \sigma) \tag{3}$$

V. Problem Formulation

A. Environment

The utilized environment enables the testing and comparison of both methods to see whether the MATD3 algorithm is superior for the multi-agent flocking problem. Various parameters in the environment are altered in order to fit the application accordingly based on real-world values. The most popular UAV control system uses 2.4GHz for UAV control and 5GHz for video transmission if needed. This configuration provides the user with around 6000m of range; hence the constructed environment has dimensions of 5500m x 3000m to replicate the UAVs' operable region.

Proceeding with the agents, the sizes of the agents are chosen to reassemble drones as efficiently as possible; therefore, a value of 0.2m is assigned based on the fact that quadcopters' frame sizes are typically between 0.18m and 0.25m. The flock's formation is then selected from among the several

shapes used by UAVs, such as the arrow, hexagonal, circular, and diamond. The diamond formation is chosen due to its many advantages, including the fact that it enables the UAVs to quickly shift direction if necessary and allows for a variable number of UAVs. In this study, the environment is tested using different numbers of agents in a diamond formation.

The agents always begin the episode at the centre of the environment in a diamond formation. The destination or objective is randomly placed on the environmental boundaries to ensure that the formation travels a distance sufficient for training in each episode. Ten obstacles are randomly placed between the agent and the goal is to ensure that the agent encounters at least one obstacle in each episode so that the policy can learn how to avoid obstacles more efficiently. In this setting, the control output is a force vector and the observations are the positions of the particles and obstacles. For a real-world application, GPS or LiDAR can be used to determine the positions of drones and obstacles. Regarding the neural network, it is a multi-layer fully-connected perceptron with three layers of 64 neurons each, activated by a rectified linear function.

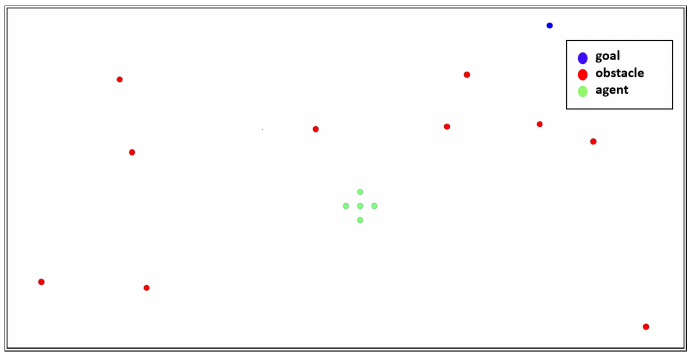

Fig. 1. Environment using five agents

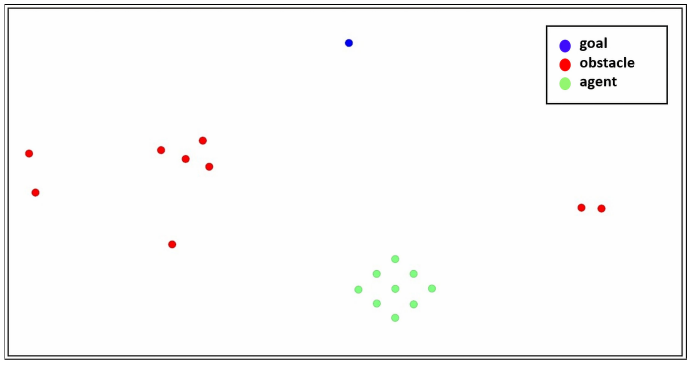

Fig. 2. Environment using nine agents

B. Reward Function

Four different reward functions are designed to satisfy the requirements of the flocking problem.

$$R_i = \omega_g R_i^g + \omega_s R_i^s + \omega_c R_i^c + \omega_o R_i^o \tag{4}$$

Where R_i^g is for reaching the target, R_i^s for maintaining a safe distance between the robots, R_i^c for maintaining communication between the robots, and R_i^o for avoiding collisions between the robots and obstacles. Moreover, weights $(\omega_g, \omega_s, \omega_c, \omega_o)$ are added to the expression to obtain a proper final reward.

The purpose of the first reward function is to instruct the agent to create a diamond shape at the destination. Each agent is assigned a precise position relative to the destination. If the agent is not within the clearance range at each step, it receives a negative reward or punishment proportional to the distance between the agent and the goal point. If the agent occupies the desired location, a reward of +4 is returned. This is a step-by-step process that occurs for each agent.

$$R_i^g = \begin{cases} r_{\text{goal}} & \text{if } \|p_i - g\|_2 \leq \rho_g \\ -\|p_i - g\|_2 & \text{otherwise} \end{cases} \tag{5}$$

To guarantee that all agents remain within the connection range, the agent is penalized with a negative reward proportional to the distance from the connection range between any two agents at any given time. The second function of rewards is to train the policy to sustain agents' connectivity. WIFI mesh networks are used to establish UAV-to-UAV communication in order to construct a wireless network architecture. Each device in a wireless mesh network is commonly referred to as a mesh node and is simultaneously connected to other numerous mesh nodes. UAV flocks typically employ 5 GHz high-frequency wireless mesh networks for automation, precision, and rapid data transfer. This network has a range of approximately 500 meters. Accordingly, the reward function is constructed. After each step, the distance between each agent and the next is measured; if it is greater than 0.5, the agent receives a negative reward of -10.

$$R_i^c = \begin{cases} r_{\text{comm}} & \text{if } d(\boldsymbol{r}_i, \boldsymbol{r}_j) \leq \rho_c \\ -(d(\boldsymbol{r}_i, \boldsymbol{r}_j) - \rho_c) & \text{otherwise} \end{cases} \tag{6}$$

The agent receives a negative reward when it is within the range of an obstacle in order to avoid it. The third reward function teaches the policy to avoid obstacles. After each step, the distance between each agent and each obstacle is measured, and a negative reward of -10 is returned if the distance is within the 40 cm critical zone.

$$R_i^o = \begin{cases} r_{\text{obstacle}} & \text{if } d(\boldsymbol{r}_i, \boldsymbol{o}_j) \geq \rho_o \\ -(\rho_o - d(\boldsymbol{r}_i, \boldsymbol{o}_j)) & \text{otherwise} \end{cases} \tag{7}$$

The fourth reward function is used to train the policy to prevent flock collisions. After each step, the distance between each agent and the other agents is computed; if they are within a specified critical distance of 40 cm, a negative reward of -10 is returned to ensure that the agents avoid colliding.

$$R_i^s = \begin{cases} r_{\text{avoid}} & \text{if } d(\boldsymbol{r}_i, \boldsymbol{r}_j) \geq \rho_n \\ -10 & \text{otherwise} \end{cases} \tag{8}$$

All positive rewards are set to zero, except r_{goal} , to encourage agents to minimise punishments while continuing to accomplish the objective. After computing the reward functions independently, they are multiplied by a weight and summed to determine the total policy reward. The weights of each reward are calculated by determining the maximum value of the first reward function and adjusting the remaining weights so that they have the same maximum value. In this manner, all reward functions would have the same range of operation, guaranteeing that they all have the same priority and are all considered while optimizing the policy. The final attained weights are as follows:

ω_g=1.5, ω_c=0.315, ω_o=2.588, ω_s=1.187

VI. Experiments and Results

Plotting data is saved for every episode throughout 300K episodes.The final reward shows minimal penalty or punishment, which also indicates a minimum of collisions and lost communications between the drones. The final reward plots show that MATD3 achieved better results in a shorter amount of time. This proves that MATD3 has a faster convergence rate and minimal overestimation bias than that accompanied by the MADDPG algorithm. Both algorithms are highly sensitive to the used hyper-parameters, so some of the values were tested. Those that gave promising results are shown in the tables below.

A. Core Training Parameters

1) MADDPG Core Training Parameters:

Parameter	Value
Learning rate	lr=1e-2
Discount factor	γ =0.95
Batch size	batch-size = 1024
Number of units	num-units =64
Episode Length	max-episode-len = 50
Number of Episodes	num-episodes = 300000

TABLE I
MADDPG Core Training Parameters

2) MATD3 Core Training Parameters:

Parameter	Value
Learning rate	lr=1e-2
Discount factor	γ =0.95
Batch size	batch-size = 1024
Number of units	num-units =64
Action noise	ϵ = 0.5
Critic noise	critic-action noise = 0.2
Episode Length	max-episode-len = 50
Number of Episodes	num-episodes = 300000

TABLE II
MATD3 Core Training Parameters

B. Results

The performance of the algorithms is compared in terms of the four rewards separately: the collision rate of agents with one another, the collision rate between agents and obstacles, the final distance between the agents and the desired destination, and finally the rate of lost communication between agents. After gathering data for 300K episodes, data sets for both algorithms are plotted in order to compare them side by side. The following results were attained while using five agents and ten obstacles.

1) Collision Avoidance Reward:
This reward function indicates the number of collisions that occur between agents every 1000 episodes; each collision returns -11.87.
MATD3 converged after 230K episodes with a final reward value of approximately -20, which indicates a collision frequency of 2 collisions every 1000 episodes. While MADDGP converged after 250K episodes with a final reward value of approximately -50, which indicates a collision frequency of 5 collisions every 1000 episodes. This comparison concludes that MATD3 converges faster to a more optimal solution with a lower collision rate.
2) Connection Reward:
This reward function indicates the number of times agents have lost connection with one another. Regarding this reward function, both algorithms behaved similarly; they both converged at about 50000 episodes with minimal loss of communication between agents. The only difference worth pointing out is that MATD3 shows less variance throughout the duration of the training session.
3) Obstacle Avoidance Reward:
This reward function indicates the number of collisions that occur between agents and obstacles every 1000 episodes; each collision returns -25.68.
MATD3 converged after 80K episodes with a final reward value of approximately -125, which indicates a collision frequency of 5 collisions every 1000 episodes. MADDPG did not converge within the 300K episodes; its final reward value is approximately -400 indicating a collision frequency of 16 collisions every 1000 episodes.
4) Destination Reward :
This reward function indicates how fast agents reach the destination while maintaining the formation.
MATD3 converged after 150K episodes with a final reward value of approximately +150. The positive value indicates that agents are rewarded for reaching the destination in formation and are not penalized much for the duration of the episodes. MADDPG converged after 20K episodes with a final reward value of approximately -200. The negative value indicates that the algorithm converges on a policy that does not reach the destination in a formation and the agents continue to get penalized for the duration of the episodes indicating a failure to complete the flocking task.

This comparison shows that MADDPG has the tendency to fall into a local minimum that is caused by overestimation bias. On the other hand, MATD3 continues to search for a more optimal solution.

5) Cumulative Reward:
Both algorithms, MADDPG and MATD3, showed convergence. Figures 3 and 4 represent the summation of the 4 previous reward functions multiplied by their specific weights to have the same range of operation. These rewards accentuate the improvement brought by MATD3 to the multi-robot flocking control problem.

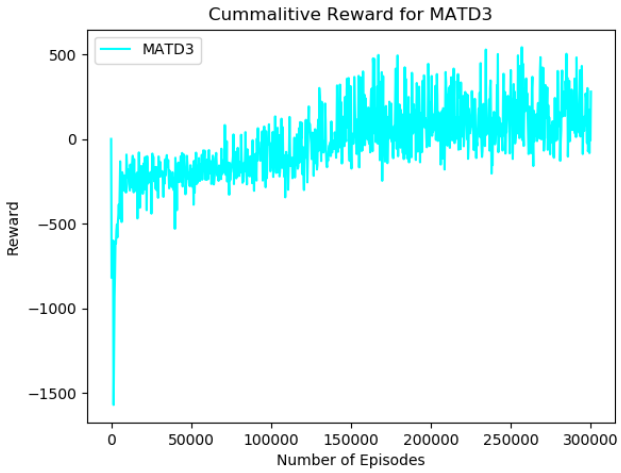

Fig. 3. Cumulative Reward For MATD3

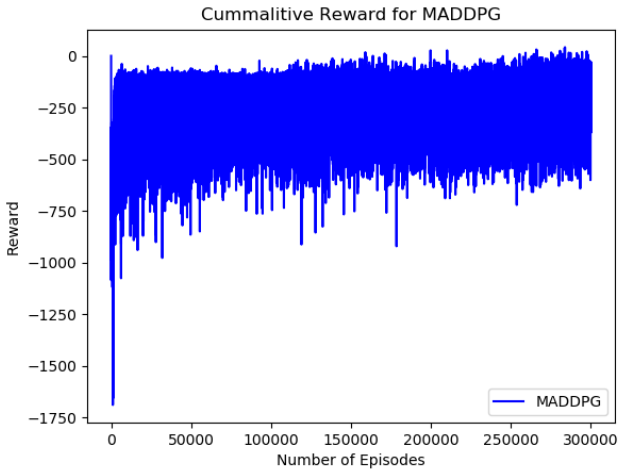

Fig. 4. Cumulative Reward For MADDPG

VII. Conclusion

This paper introduces the multi-agent twin delayed deep deterministic policy gradient algorithm to the multi-agent flocking control problem to solve drawbacks that are associated with previous solutions used alongside this problem, such as the multi-agent deep deterministic policy gradient. A particle environment is used to test both algorithms and to compare their performances. A reward function is designed to train the flock to be able to maintain communication, obstacle avoidance, collision avoidance, and reach the destination in formation. The results show that the MATD3 performs better in the 4 sub-divisions of the reward function. MATD3 does not show signs of overestimation or falling into a local minimum which occurs while using MADDPG.

References

[1] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[2] H.-T. Zhang, Z. Cheng, G. Chen, and C. Li, "Model predictive flocking control for second-order multi-agent systems with input constraints," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 6, pp. 1599–1606, 2015.

[3] B. K. Sahu and B. Subudhi, "Flocking control of multiple auvs based on fuzzy potential functions," *IEEE Transactions on Fuzzy Systems*, vol. 26, no. 5, pp. 2539–2551, 2017.

[4] Q. Luo and H. Duan, "Distributed uav flocking control based on homing pigeon hierarchical strategies," *Aerospace Science and Technology*, vol. 70, pp. 257–264, 2017.

[5] W. Dai, H. Lu, J. Xiao, Z. Zeng, and Z. Zheng, "Multi-robot dynamic task allocation for exploration and destruction," *Journal of Intelligent & Robotic Systems*, vol. 98, no. 2, pp. 455–479, 2020.

[6] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," *arXiv preprint arXiv:1707.04402*, 2017.

[7] P. Zhu, W. Dai, W. Yao, J. Ma, Z. Zeng, and H. Lu, "Multi-robot flocking control based on deep reinforcement learning," *IEEE Access*, vol. 8, pp. 150 397–150 406, 2020.

[8] X. Chu and H. Ye, "Parameter sharing deep deterministic policy gradient for cooperative multi-agent reinforcement learning," *arXiv preprint arXiv:1710.00336*, 2017.

[9] P. Yan, C. Bai, H. Zheng, and J. Guo, "Flocking control of uav swarms with deep reinforcement leaming approach," in *2020 3rd International Conference on Unmanned Systems (ICUS)*. IEEE, 2020, pp. 592–599.

[10] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, O. Pieter Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," *Advances in neural information processing systems*, vol. 30, 2017.

[11] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[12] J. Ackermann, V. Gabler, T. Osa, and M. Sugiyama, "Reducing overestimation bias in multi-agent domains using double centralized critics," *arXiv preprint arXiv:1910.01465*, 2019.

[13] A. Nguyen, "Openai's maddpg algorithm," May 2020. [Online]. Available: https://towardsdatascience.com/openais-multi-agent-deep-deterministic-policy-gradients-maddpg-9d2dad34c82

[14] "Twin delayed ddpg." [Online]. Available: https://spinningup.openai.com/en/latest/algorithms/td3.html

[15] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.