Towards Multi Robot Task Allocation and Navigation using Deep Reinforcement Learning

Towards Multi Robot Task Allocation and Navigation using Deep Reinforcement Learning

A Elfakharany[1], **R Yusof and Z Ismail**
Center for Artificial Intelligence & Robotics, Malaysia Japan International Institute of Technology, Universiti Teknologi Malaysia, Jalan Sultan Yahya Petra, 54100 Kuala Lumpur, Malaysia
E-mails: efahmed@graduate.utm.my, rubiyah.kl@utm.my and zool@utm.my

Abstract. Developing algorithms for multi robot systems to reach target positions and navigate safely in the environment is an open field of research. Most systems treat Multi Robot Task Allocation (MRTA) and Multi Robot Path Planning (MRPP) as two separate steps each with its own set of algorithms in which the MRTA algorithm assigns each robot to a task and the MRPP algorithm guides each robot through the environment towards the assigned goal position while avoiding both static and dynamic obstacles. In this paper, we present a method that combines both steps by using a deep reinforcement learning model. The model consists of a decentralized sensor level policy which outputs the robot's velocity to guide it through the environment towards the selected goal position and avoiding collisions. The model was trained in a simulation environment and all the robots are homogenous differential drive robots. The objective is to ensure that each robot reaches a unique goal position with the number of goal positions is equal to the number of robots. The results of training the policy in an environment is presented with both static and dynamic obstacles with four robots and four goal positions.

1. Introduction

Multi Robot Task Allocation (MRTA) and Multi Robot Path Planning (MRPP) are two fields of interest for robotics and artificial intelligence researchers. Both fields have applications ranging from search and rescue operations to warehouse automation. MRTA is the process of assigning robots to tasks in a multi robot system in order to decrease the probability of collisions between the robots and decrease the total cost expended by the robot [1], make it be: time, energy or distance. While MRPP is concerned with finding an optimal path between the start position and the end position that avoids collisions with other robots and obstacles [2]. Most multi robot systems treat MRTA and MRPP as two different steps [3]. However, one of the main problems that occur due to the decoupling of both MRTA and MRPP is that most MRTA algorithms is the lack of incorporation of any dynamic changes in the path's cost due to the existence of dynamic obstacles [4] or due to partial observability.

In this paper, we present a decentralized sensor level deep reinforcement learning policy that performs both tasks in a single step. The inputs to the policy are the positions of the goals, the positions of the other robots and the laser scanner readings. The output of the policy are the velocities of the robot it's running on. The output of the network should drive the robot towards a unique goal that none of the other robots are moving towards while avoiding obstacles.

[1] To whom any correspondence should be addressed.

2. Problem formulation

The problem of multi-robot task allocation and navigation is defined in the context of a differential drive mobile robot moving in a 2D plane with both static and dynamic obstacles and other moving robots (all robots are homogenous). The goal locations are laid out in the environment, the number of goal locations equals the number of robots.

Each robot *(i)* at time step *(t)* receives an observation (O_i^t) and calculates the output command (a_i^t) which drives the robot from the start position (P_i) towards the goal position (G_i). The observation for each robot is composed of four parts: $O_i^t = [O_c^t, O_o^t, O_l^t, O_t^t]$, where O_c^t is the relative positions and velocities of all the goals in the robot's local polar coordinates (the goals locations are static but the relative positions changes as the robot moves), O_o^t is the relative positions and velocities of all the goals in the other moving robots' local polar coordinates, O_l^t is the 2D laser scanner measurements and its time derivative, while O_t^t is the current time of the robot starting from the beginning of the robot's motion. The observation only provides partial information about the robot's state and the other robots' states, which leads the robot to predict the other robots' intent.

Given the observation, each robot calculates its action independently. The action is composed of two parts: $a_i^t = [v_l^t, v_r^t]$, where v_l^t is the robot's linear velocity and v_r^t is the robot's rotational velocity. The actions are sampled from two distributions produced by a policy (π) which is copied across all the robots:

$$a_i^t \sim \pi_\theta \left(a_i^t \mid O_i^t \right) \tag{1}$$

where θ denotes the parameters of the policy. The output action guides the robot towards the goal which the policy selected while avoiding obstacles on the way to the goal within the time limit between two observations. The target allocation is done in a way that the policy doesn't explicitly select one of the goals, but outputs the actions that drive the robot towards the selected goal position.

The problem is formulated as Partially Observable Markov Decision Process (POMDP) [5], in which a sequence of observations and actions made by each robot forming a trajectory (l_i) from its start position (P_i) till the goal position chosen by the policy (G_i), where (t_i^g) is the travelled time.

The main objective is to find an optimal policy that minimizes the expectation of the total arrival time of all robots to their goals successfully, defined as:

$$\underset{\pi_\theta}{argmin}\ E\left[\underset{i}{argmax}\ t_i^g \middle| \pi_\theta\right] \tag{2}$$

3. Methodology

During this section we will describe the simulation environment, then we will describe the reinforcement learning setup that we are using, finally we will describe the neural networks architecture.

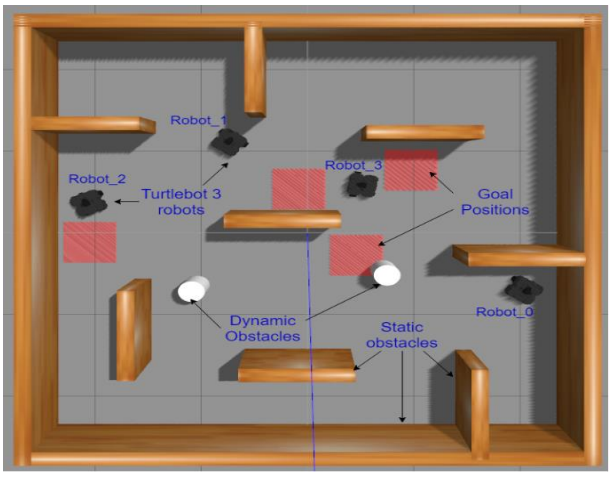

Figure 1. Simulation layout.

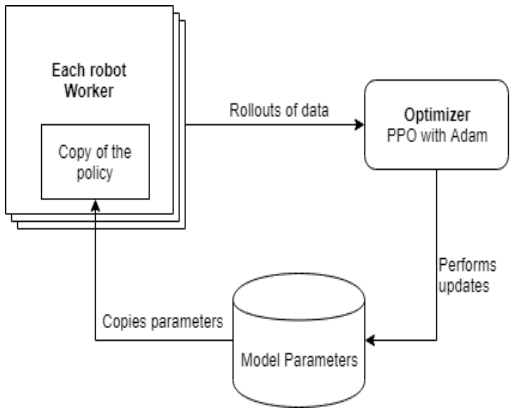

Figure 2. Model training block diagram.

3.1. Simulation environment

The simulation environment used to train the robots was built using the Gazebo simulator [6]. The area of the simulation environment is (5m x 5m) which contains four TurtleBot3 Waffle Pi robots [7], four goal positions, two dynamic obstacles and multiple static obstacles as shown in Figure 1. At the beginning of training the robots and goals are spawned at random positions, the number of goals is the same as the number of robots. During training, when all the robots reach all the goal positions the goals positions are respawned at random positions.

3.2. Deep reinforcement learning setup

In our study we focus on learning a policy that is capable of controlling a robot within a multi robot system. The policy selects one goal position within multiple ones and produces a series of actions that gets the robot to the goal position while avoiding obstacles. Inherently the policy should learn the intents of other robots and which goal position each of the other robots is going towards and aim for a different goal position.

Algorithm 1. PPO with multi-robot system

1. Initialize policy network π_θ old policy network $\pi_{\theta old}$ and value network V_φ using hyperparameters in Table 1.
2. For iteration = 1,2,… do
3. //data collection
4. For robot i = 1,2,3,…N do
5. Run Policy π_θ for T_i time steps, collecting (O_i^t, a_i^t, r_i^t) , where $a_i^t = [v_l^t, v_r^t]$
6. Calculate Advantage $A_i^t = \sum_{t=0}^{Ti} \gamma^t r_i^t - V_\varphi(O_i^t)$
7. Break if $T_i >= T_{max}$
8. End for
9. //Update Policy & value
10. For j = 1,2,…. Epochs do
11. //policy loss
12. $L^{clip} = \sum_{t=0}^{Ti} \min(\frac{\pi_\theta(a_i^t|O_i^t)}{\pi_{\theta old}(a_i^t|O_i^t)} * A_i^t, clip\left(\frac{\pi_\theta(a_i^t|O_i^t)}{\pi_{\theta old}(a_i^t|O_i^t)}, 1-\epsilon, 1+\epsilon\right) * A_i^t) /Ti$
13. //value loss
14. $L^v = -c_1 * \sum_{t=0}^{Ti}(A_i^t) /Ti$
15. //entropy loss
16. $L^e = -c_2 * \sum_{t=0}^{Ti}(a_i^t * \log(a_i^t)) /Ti$
17. //total loss
18. $L = L^{clip} + L^v + L^e$
19. Update θ, φ with Adam optimizer [10] w.r.t L
20. End for
21. $\pi_{\theta old} \leftarrow \pi_\theta$
22. End for

For our deep reinforcement learning algorithm we are using Proximal Policy Optimization (PPO) method [8] shown in Algorithm 1, we are adapting the centralized learning, decentralized execution paradigm [9] in which each robot has a copy of the policy network, each robot collects its data (O_i^t, a_i^t, r_i^t) from the environment and after each episode it sends the rollouts of data over to a centralized copy of the policy. The gradients are then calculated on the centralized policy and the centralized policy is updated. After the update, each robot receives a copy of the updated policy weights to start collecting a new batch of data as shown in Figure 2. The episode ends either by one of the robots having a collision or all the robots reaching all the goals positions successfully or the episode duration is exhausted.

Both the actor and critic networks share the parameters of their first layers as shown in Figure 3. The actions produced by 2 heads each has softmax categorical activation, the discrete output of each head is converted to a continuous value for each action in the range [-1,1].

Table 1. Hyperparameters used in Algorithm 1

Parameter	Value
γ	0.95
T_{max}	120 seconds
Epochs	5
ϵ	0.2
c_1	0.5
c_2	0.01
Batch size	4096
Learning rate	1e-5
Learning rate decay	0.96
Learning rate decay steps	1000

The observation space (O_i^t) is –as mentioned in Section 2.0- consisted of: ($O_c^t, O_o^t, O_l^t, O_t^t$). O_c^t is the relative goals positions and velocities in the current robot's polar coordinates, the O_c^t is a 3D tensor ($O_c^t \in \mathbb{R}^{75\times4\times4}$). The first dimension represents the past 75 consecutive values of this observation. The second dimension represents the goal's distance from the robot, the goal's heading from the robot, the time derivative of the goals distance and the time derivative of the goals heading. The third dimension represents each of the goals in the environment. Since O_c^t encompasses the information of all the goals relative to the current robot, it enables the robot to choose which goal position to aim towards. While O_o^t is the relative goals positions and velocities in the other robots polar coordinates, it's as 4D tensor ($O_o^t \in \mathbb{R}^{75\times3\times4\times4}$), the third dimension represents each of the other robots while the first, third and fourth are the same as O_c^t. Since O_o^t encompasses the information of all the goals relative to the other robots, it enables the current robot to predict the intents of other robots and which robot is aiming towards which goal, this aids the current robot to choose a unique goal position to aim towards. O_l^t is the 360° laser data, it's a 3D tensor ($O_l^t \in \mathbb{R}^{75\times2\times360}$), the first dimension represents the previous 75 frames of measurements, the second dimension represents the raw laser readings and the time derivative of the raw laser readings. The third dimension represents all 360 distance values. O_l^t enables the robot to detect bot static and dynamic obstacles and helps the robot to navigate without collisions. Finally O_t^t is a scalar representing the time spent since the robot started to move. Since the robots are required to train to minimize the expectation of the total arrival time of all robots to their goals successfully, each robot requires an input that informs it of the time-state O_t^t. The action space a_i^t is consisted of two parts: the robot's linear velocity v_l^t and the robot's rotational velocity v_r^t, the network outputs each of them in a 17 category discretised softmax distribution, the category with the highest probability is converted to a continuous value within the range [-1,1] for each action.

The reward function is designed to incentivise each robot to choose and move towards a unique goal position. It penalizes getting near to obstacles and colliding with them, multiple robots reaching the same goal position and consuming long amounts of time to reach the goal:

$$r_i^t = \begin{cases} (r^d)_i^t + (r^h)_i^t + (r^{ob})_i^t + (r^o)_i^t + (r^t)_i^t, \ In\ case\ of\ motion \\ 100, \ In\ case\ of\ reaching\ a\ unique\ goal \\ 0, \ In\ case\ of\ reaching\ a\ goal\ occupied\ by\ another\ robot \\ -100, \ In\ case\ of\ a\ collision \end{cases} \quad (3)$$

while the robots are moving, the reward by robot i at time step t is a sum five terms as shown in (3), $(r^d)_i^t$ is a reward that increases in value when the robot move towards the nearest goal:

$$(r^d)_i^t = 7 * 0.05^{d^t/d^0} \quad (4)$$

where d^t is the robot's current distance from the nearest goal while d^0 is the initial distance between the robot and the same goal. $(r^h)_i^t$ is a reward that increases in value when the robot is heading towards the goal:

$$(r^h)_i^t = \begin{cases} 0.5 - \frac{0.7|heading|}{\pi}, & if\ |heading| \leq \frac{\pi}{2} \\ -0.3, & otherwise \end{cases} \quad (5)$$

where heading is the bearing of the nearest goal position relative to the robot. While $(r^{ob})_i^t$ is the reward that penalizes the robot getting near to an obstacle:

$$(r^{ob})_i^t = 2.46\, log_{10}(min(O_l^t[0,0])) + 0.76 \quad (6)$$

$(r^o)_i^t$ is the reward that penalizes the robot in case of it is moving towards a goal position which another robot is moving towards:

$$(r^o)_i^t = max(0.5\, log_{10}\left(\frac{nd^t}{nd^0}\right), -10) \quad (7)$$

where nd^t is the instantaneous distance of the nearest of the other robots towards the nearest goal to the robot, while nd^0 is the initial distance. $(r^t)_i^t$ is the reward that penalizes the time consumed to reach the goal:

$$(r^t)_i^t = max(-1 * \frac{t}{120}, 0) \quad (8)$$

where t is the time in seconds since the start of the robot's motion.

3.3. Neural Network Architecture

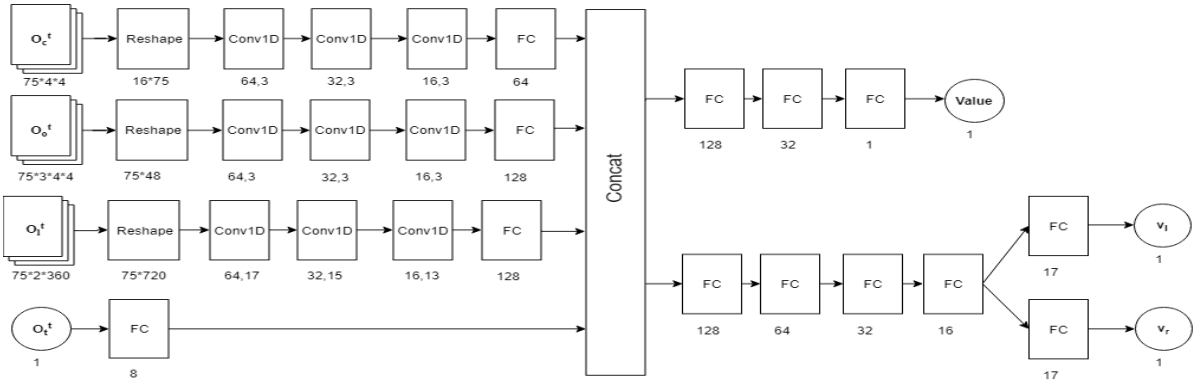

Figure 3. Network architecture

The policy network π_θ and the value network V_φ share the parameters of the first few layers, the activation function of the shared layers is ReLU [11]. For the value head, the activation used is tanh except for the final layer whose activation is linear. For the policy head, tanh activation was used for all

the layers except the two output layers whose activation is softmax. The total number of parameters is 1,179,347.

4. Simulation results and discussion

The neural network was implemented using Tensorflow [12] and the network training were run on a Nvidia DGX station. The training took about 24 hours 43 minutes which is 7331 episodes with 1,799,109 time steps. 6 of the episodes ended by all the robots reaching all the goals, 2 of the episodes ended by ending the episode duration and the rest ended by collisions. The goal positions changed randomly during training only when all the robots reach all the goals.

Table 2 shows some performance metrics of the network per robot during training, it shows the average HZ which is the average number of times a second the network runs. It shows the average linear and rotational velocities $[v_l^t, v_r^t]$. The table also includes the number of collisions each robot had during training and the average time spent from the start of the episode till the collision. The table shows the number of times each robot reached a goal position and the average time spent from the start of the episode till it reached the goal position. Figure 4 shows the mean rewards per episode per robot.

Table 2. Performance metrics of all 4 robots during training.

	Average HZ	Average v_l^t	Average v_r^t	Collisions	Reach a goal
	mean/std	mean/std	mean/std	number of occurrences/ mean time since episode start/ std time since episode start	number of occurrences/ mean time since episode start/ std time since episode start
Robot 0	13.3/6.8	0.15/0.6	0.05/0.5	2566/10.9/11.96	206/10.3/3.4
Robot 1	15.4/6.5	-0.47/0.6	0.13/0.3	1785/9.2/15.5	2882/4.7/1.7
Robot 2	13.4/6.8	-0.26/0.6	0.09/0.5	1782/11.3/14.3	308/9.3/2.5
Robot 3	13.4/6.8	0.007/0.6	0.1/0.4	1189/9.2/6.3	324/10.6/3.5

For testing, the network is evaluated on the same environment but we change the goal positions randomly every 5 episodes. The test ran for 12 hours and 50 minutes, making 7531 episodes. Figure 5 shows the mean rewards per episode per robot. Table 3 shows the testing performance metrics.

Table 3. Performance metrics of all 4 robots during testing.

	Average HZ	Average v_l^t	Average v_r^t	Collisions	Reach a goal
	mean/std	mean/std	mean/std	number of occurrences/ mean time since episode start/ std time since episode start	number of occurrences/ mean time since episode start/ std time since episode start
Robot 0	15.7/5.7	0.007/0.47	0.28/0.6	2666/5.12/4.2	8/13.27/4.97
Robot 1	15.5/5.7	-0.09/0.57	-0.29/0.57	1143/8.14/3.74	127/7.94/5.36
Robot 2	15.3/5.7	-0.03/0.56	0.08/0.45	542/6.35/5.36	124/7.46/2.38
Robot 3	15.6/5.6	0.36/0.47	0.13/0.43	3184/7.62/3.4	176/8.01/2

The results show that the policy network hasn't yet converged to an acceptable robust performance. One of the reasons might be the inherit difficulty of combining two tasks together for the network to solve. In that case using curriculum learning [13] in which the model is trained on a simpler version of the task first then its trained on the required task can aid the network to learn both tasks. A second reason might be the network requires more hyperparameters tuning.

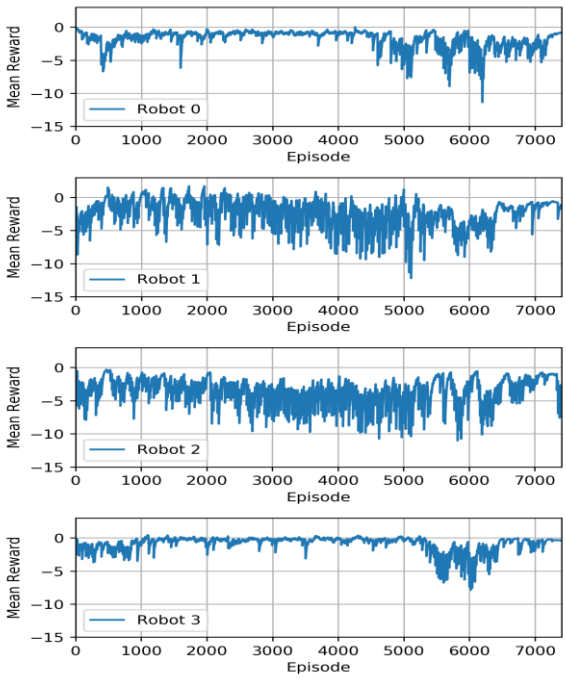

Figure 4. Mean reward per episode for each robot during training.

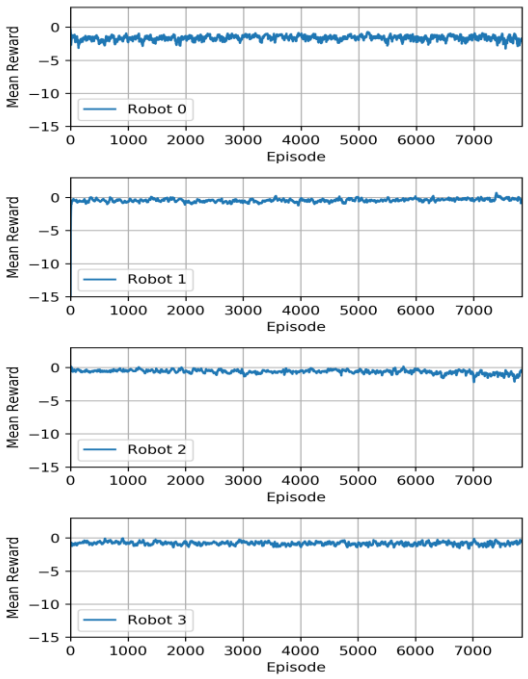

Figure 5. Mean reward per episode for each robot during testing.

Table 4 shows a comparison between our approach and work done by others, the comparison is done in terms of which parts of the multi robot system was replaced by a neural network trained by deep reinforcement learning. It shows that our approach has a deep end to end neural network that does both MRTA and MRPP, while others' related work only does one part of a multi robot system using a neural network. It also reports the computational time required for the neural network to produce a single set of actions when given a single set of states. Our computational time is slightly bigger and that is attributed to the fact that our neural network is larger than the others' related work, that is due to the incorporation of both MRTA and MRPP.

Table 4. Comparison in terms of which parts of the multi robot system were replaced by a neural network and computational time.

	MRTA	MRPP	CPU Computational time
Long et al. [9]	×	√	3 ms
Chen et al. [14]	×	√	62 ms
Zhu et al. [15]	√	×	N/A
Our approach	√	√	63.7 ms

5. Acknowledgement
The authors would like to express gratitude to Malaysia-Japan Institute of Technology, Universiti Teknologi Malaysia for funding this research under vote number R.K130000.7343.4B317.

6. Conclusion

This paper presented a deep reinforcement learning model that performs decentralized task allocation and navigation for a multi robot system. We presented the results of training the policy network in a simulation environment. The results show the network hasn't converged to an acceptable performance and it is suggested further enhancement based on hyperparameters tuning as well as using curriculum learning.

7. References

[1] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," Int. J. Rob. Res., vol. 23, no. 9, pp. 939–954, 2004.
[2] L. E. Parker, "Path Planning and Motion Coordination in Multiple Mobile Robot Teams," in Encyclopedia of Complexity and System Science, E.-C. Robert A. Meyers, Ed. Springer, 2009.
[3] A. Hussein, M. Adel, M. Bakr, O. M. Shehata, and A. Khamis, "Multi-robot Task Allocation for Search and Rescue Missions," J. Phys. Conf. Ser. OPEN ACCESS.
[4] B. Woosley and P. Dasgupta, "Multirobot Task Allocation with Real-Time Path Planning," in Proceedings of the Twenty-Sixth International Florida Artificial Intelligence Research Society Conference, 2013.
[5] M. T. J. Spaan, "Partially Observable Markov Decision Processes," Springer, Berlin, Heidelberg, 2012, pp. 387–414.
[6] N. Koenig and A. Howard, "Design and use paradigms for Gazebo, an open-source multi-robot simulator," in 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2004, vol. 3, pp. 2149–2154.
[7] "TurtleBot 3 e-manual." [Online]. Available: http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/#specifications.
[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, O. K. Openai, and O. Klimov, "Proximal Policy Optimization Algorithms," CoRR, vol. abs/1707.0, 2017.
[9] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning," in International Conference on Robotics and Automation (ICRA), 2018.
[10] D. P. Kingma and J. L. Ba, "ADAM: a method for stochastic optimization," arXiv Prepr. 1412.6980v9, pp. 1–15, Dec. 2015.
[11] V. Nair and G. E. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," Proc. 27th Int. Conf. Mach. Learn., no. 3, pp. 807–814, 2010.
[12] M. Abadi et al., "TensorFlow: A System for Large-Scale Machine Learning TensorFlow: A system for large-scale machine learning," in Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). , 2016.
[13] Y. Bengio, Jérôme Louradour, R. Collobert, and J. Weston, "Curriculum Learning," in Proceedings of the 26 th International Conference on Machine Learning, 2009.
[14] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in Proceedings - IEEE International Conference on Robotics and Automation, 2017, pp. 285–292.
[15] Q. Zhu and J. Oh, "Deep Reinforcement Learning for Fairness in Distributed Robotic Multi-type Resource Allocation," in 2018 17th IEEE International Conference on Machine Learning and Applications (ICMLA), 2018, pp. 460–466.