Multi agent Exploration with Reinforcement Learning

1st Alkis Sygkounas
Author
Department of Signals and Automated Control, Patras, Greece

2nd Dimitris Tsipianitis
Co-Author
Department of Signals and Automated Control, Patras, Greece

***Abstract*—Modern mobile robots have begun to be used in many exploration and exploration and rescue applications. They are essentially coordinated by human operators and work with inspection or rescue teams. Over time, robots (agents) have become more sophisticated and we see more autonomy in more complex environments. Therefore, the purpose of this research is to present an approach for autonomous multi-agent coordination for exploring and covering unknown environments. The method we suggest is reinforcement learning in combination with the use of multiple neural networks (Deep Learning) to plan the course for each agent separately and also the effort to achieve collaborative behavior amongst them. Specifically, we have used two techniques applied in recent years, which are the target neural network and the prioritized experience replay, which have been proven to stabilize and accelerate the training process. Also, agents must avoid obstacles (walls) throughout the exploration. We have also solved the problem of the need for prior information/knowledge about the environment, thus using only the local information available at any given time to make the decision of each agent. Furthermore, two main neural networks are being used for generating actions and also a neural network with a logic of switcher that chooses between those. The exploration of the unknown environment is done in a two-dimensional model (2D) using multiple agent for a variety of different maps, ranging from small sizes to and large sizes. Finally, the efficiency of the exploration is investigated for a different number of agents and for a different type of neural network.**

I. Introduction

Autonomous flight for UAVs (unmanned aerial vehicles) has become an increasingly important research area in recent years. One of the major challenges with autonomous motion planning is ensuring that an agent can efficiently explore a space while avoiding collision with objects in complex and dynamic environments. To resolve this, recent research has turned to applying deep reinforcement learning techniques to robotics and UAVs. For the last years, the number of applications where mobile robots collaborate with humans [10] dramatically increased. Mainly, this is due to advances in batteries, powerful computation boards, and miniaturization of sensors. Unmanned aerial vehicles (UAV) have been deployed in areas that are deemed dangerous for human operation, and provide important information about the environment in applications such as search and rescue, site inspection, victim search in disaster situations and monitoring. UAV must be designed to operate autonomously with no prior information about the environment Moreover, they are used in disaster search and rescue missions [1], utility inspection [7] crop monitoring [2], etc. Depending on mission requirements, mobile robots could be equipped with different types of sensors or actuators that depend on the mission objectives. Their assistance in inspection, search, and rescue missions, and the exploration of unknown areas plays crucial role and provides benefits to human operator safety, decision making, fast exploration, three-dimensional (3D) reconstruction of the environment and providing localization.

In this research we introduce a reinforcement learning approach (RL) [4] of exploring and navigating in unknown maps with different amount of agents without any previous knowledge of the area. The RL approach is being selected due to the fact there is no prior information about the environment neither mathematical model for navigation through unknown environments [8]. Also, using deep reinforcement learning methods for exploration acquires a major progress recently since deep Q-learning has successfully applied to the continuous action domain problem [5] and understanding of a UAV agent, while exploring a partially observable environment [6]. Furthermore, the agents are getting information only from their local observations and the information they exchange/share. We tried to cover as much area as possible with multiple agents on a limited time, while simultaneously they avoided obstacles(walls). We used a variation of Q-learning [3] and also utilized the prioritized experience replay [9]. Furthermore, we used two different neural networks, one being the main net for exploration and the other being a secondary for going to a specific destination. The selection of which network will be used each time step is being solved from a third neural network called switcher. The testing maps are two dimensions with size of grid by grid. The testing (and training) were concluded in the size of 50x50 (grid-style) maps with different number of randomly generated (each time) walls. Also, different number of agents were tested for determining the coverage percentage of the maps. In addition, there is a comparison in performance (total coverage) between the main model with a shallow neural network, the same main

model with a deeper network and finally the combination of the deeper model with the usage of the second neural network alongside with the switcher. To be noted, the resolution of each grid cell is not taken in consideration in our approach not either the limited battery operating time of the agents in a real life scenario. We also assume perfect localization of the agents.

II. Methods

A. Training Algorithm

The main method being used for the training of the agents is the Reinforcement Learning. Specifically, the algorithm that is used is the Q-learning algorithm [11]. Given an action u_t that generates a reward r_t for the state s_t and observing a new state s_{t+1}. The Q-function updates according:

Equation 1 Q-learning

$$Q_{t+1}(s_t,u_t) = Q_t(s_t,u_t) + a(r_t + \gamma max_{t+1} Q_{t+1}(s_{t+1},u_{t+1}) - Q(s_t,u_t)) \quad (1)$$

where a is the learning rate with a value between $[0,1]$ and γ is the discount factor that determines the future rewards.

For most problems, it is impractical to represent the Q-function as a table containing values for each combination of s and a. Instead, we train a function approximator, such as a neural network with parameters θ, to estimate the Q-values, i.e $Q(s,a;\theta) \approx Q(s,a)$. This can be done by minimizing the following loss at each step i:

Equation 2 Temporal Difference

$$L_i(\theta_i) = E_{s,a,r,s'}[(y_i - Q(s,a;\theta_i))]^2, \text{where } y_i = r + \gamma \max_{a'} Q(s',a';\theta_{i-1}). \quad (2)$$

Here, y_i is called the TD (temporal difference) target and $y_i - Q$ is called the TD error. Note that the parameters from the previous iteration are fixed and not updated. In practice we use a snapshot of the network parameters from a few iterations ago instead of the last iteration. This copy is called the target network. This technique is called Double Deep Q-learning [3] and the main neural network is updated from the target network via:

Equation 3 Target Neural Network

$$Q(s,a,\theta_{main}) = r + \gamma max_a Q(s_{t+1},a_{t+1},\theta_{target}) \quad (3)$$

We update the weights of the target network equal to the weights of the weights of the main model every c-iterations. The θ_{main} and the θ_{target} are the weights of the main and target networks. The main and target networks consists the double Q-learning algorithm.

B. Memory model

In the classic DQN approach the agent takes experiences uniformly at random that are stored in his memory in order to train. So, all experiences are treated equally and directly being in contrast to the real life experience in which some experiences will be more beneficial than others. To accomplish that, we use the PER [9] method sample experiences to a probability p_t is proportional to the absolute difference between target and estimation values. That being said the p_t is given by:

Equation 4 Prioritized Experience Replay (PER)

$$p_t \sim (|r_t + \gamma max_{t+1} Q(s_{t+1},u_{t+1};\theta') - Q(s_t,u_t;\theta)| + \epsilon)^\omega \quad (4)$$

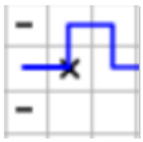
where ϵ is a small positive constant ensuring that there is no experience with zero probability of being sampled. Also, ω determines how much prioritization is being used with $\omega = 0$ being the uniform case.

C. Main model's algorithm

For the training phase for the main model, firstly we initialize a memory buffer of a standard siz, in which we store our current state, the next state, action and reward for each step in each episode per batch size. Then, each time-step the agents take an action based on some policy and observe the reward and the new state, with each agent having his own neural network. After a specific number of replay steps, we train the agents based on the store experiences using the Huber loss function. For choosing the previous experiences we used prioritized experienced replay since it performs better than uniform experience replay. Also, in all cases we used the double neural technique(DDQN).

D. Local information and input state

Due to lack of any previous knowledge of the maps, the agents don't know the size of the map. So, the only information that is available is the local observation of each agent from his current location. Specifically, each cell on the map can have 3 possible states, meaning that it is either unexplored, already explored or it has a wall. In our case the agents have 1-cell in each direction (and diagonally) in depth as sensed input. So the input state for the main model is a total number of 9 (8+information of the current position cell) plus the gps coordinates of the agents, as we want to share basic information between them. Specifically each agent has the gps-coordinates information of the one that is closest to him plus his own. So the total input size is 13. In the figure 1 we can see how the agent proceeds his local information. With -1 we represent the undiscovered squares, with 0 the already discovered squares and with 1 the occupancy of a wall. The x represents the current position of the agent and the blue line the path he has done already.



1	0	0
0	0	0
1	-1	-1

Fig. 1: Local observations

E. Reward function of main model

For the reward function we want to award the agent for discovering new spots but we don't want to punish if the agent goes to a previously discovered spot, as they may need to pass from it or come back from a dead end. So we give 2 points for each newly discovered place and we punish -10 for each movement that results to a wall. Also, we punish -1 for each time-step in order to try to maximize the newly discovered place in correlation to time. For each time step (t) the reward R_t is given in the equation 5.

Equation 5 Reward function

$$R_t = 2 * new_{discovered} - 10 * wall_{hit} - 1 \quad (5)$$

F. Main neural net structure

The main network has been tested in two sizes, a shallow and a deep. The first one has the input state with size of 13 (local observations plus gps coordinates), three hidden layers with a total of 512 nodes (neurons) and an total output of 5 for each possible actions(including the action of staying still). Moving on, the deep model has the same input and output size but it has 7 hidden layers with each layer having 256 nodes. So the total parameters of the first model is 535.045 and 333.829 for the second one (37% less total parameters for the deeper model).

G. Second model

Due to the insufficiency of the main's model results (see section results), there was a need of adding new components and exploiting more information. Specifically, when an agent is into a position he perceives the eight surrounding local areas that he can observe, in which each area can have 3 possible states unexplored, explored and obstacle. So, for each agent in each time step we store the gps coordinates for each unexplored area that he perceives but not visiting to a list of undiscovered places and we remove it (from the list) in case of visiting that position on the future. This list is being shared amongst the agents. For that purpose, a second neural network is being introduced, that its purpose is to produce actions for the agent to go to a desired destination (unvisited area based on the list above), while moving on the environment based on the local observations and on the same time avoiding obstacles. The input of this neural network is again the local observations, plus the current position and the desired position. The structure of the net has 4 hidden layers and the output has changed to four possible actions and that is removing the action of staying still. The net was trained again with the Reinforcement Learning algorithm (DDQN) with PER as memory model and with a reward of :

Equation 6 Reward function of second model

$$R = \begin{cases} -|x_{tar} - x_{curr}| - |y_{tar} - y_{curr}| - 2 * wall_{hit}, \\ \text{if target is not reached.} \\ \\ 1000, \text{otherwise.} \end{cases}$$

with $x_{tar}, y_{tar}, x_{curr}, y_{curr}$ being the gps coordinates of target and current position respectively.

When, the second neural net is enabled (via switcher neural network) it calculates the closest (undiscovered) position from its current position from the saved list of all the undiscovered places and then it produces the necessary actions until the agents reaches the target destination.

H. Final Model

Finally, the last model that is being add is a third neural net called switcher, which the main purpose is to choose which model(main or second) will be producing actions based on the input state for each agent for each time step.

The input state for that neural net is again the local observations, plus the current and target (of the second model) positions plus the total number of steps that the agent hasn't discovered a new area (for each agent) and finally the total number of new areas that the agents have found so far. Again, the net is being trained with DDQN with a reward being shown on equation 7. The C_{total_t} being the

Equation 7 Reward of switcher

$$R = C_{total_t} - sum_{total}$$

total coverage gaining each timestep and sum_{total} the total sum of steps, that the agent has not found a new area.

For the final model, for each time step and for each agent the switcher chooses which network will produce action. If it selects the main network then the algorithm works like the 1. If it selects the second network, then firstly it calculates the closest undiscovered position and then it produces actions until the agent reach its target destination.

I. Loss function and optimizer used in the proposed method

The used loss function that is being used is the Huber loss. The Huber loss is being selected because it is less sensitive to outliers in data than the classic loss functions (MSE,MAE). This function is quadratic for small values of a and linear for large values, with equal values and slopes of the different sections at the two points where $|a| = \delta$. The huber loss function is shown in the equation 8.

RMSprop is being used as an optimization algorithm. RMSprop uses an adaptive learning rate instead of treating

ALGORITHM 1

1: Initialize Q(s,a) arbitrarily
2: Initialize memory buffer
3: For each episode repeat
4: Initialize initial state
5: For each step in episode
6: For each agent
7: Observe local state
8: Produce output action from switcher network
9: if $action_{switcher} == 0$
10: Find closest undiscovered position from current position
11: Take action from target network ***a*** *and observe* ***r,s'***
12: else
13: Take action from main network ***a*** *and observe* ***r,s'***
14: Take action ***a*** *and observe* ***r,s'***
15: Store in memory buffer ***s,s',a*** *and* ***r***
16: Calculate ***Q*** *from target net*
17: Every r-replay steps train the agent
18: Every c-steps update target network equal to main network
19: Set s'=s
20: Calculate total number of new explored areas
21: Calculate total number of steps that the agent hasn't found new area

the learning rate as a hyper-parameter. The update rule is being shown on the equation 9.

Finally, as for policy, we use the e-greedy policy. The hyper parameter is updated based in the equation 10. The epsilon variable is the probability of making a random action (instead of using optimal action based on Q-learning) for each agent for the current episode and step. The min_e is the minimum (final) probability of the agent doing a random action (0%) and the max_e is the maximum (initial) probability with a starting value of 100%.

Equation 8 Hubber Loss Function

$$R = \begin{cases} L_\delta(y, f(x)) = \frac{1}{2}(y - f(x))^2, \text{if} \quad |y(x) - f(x)| \le \delta. \\ \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, \text{otherwise.} \end{cases}$$

Equation 9 RmsProp Optimizer

$$v_t = \rho\, v_{t-1} + (1-\rho) * g_t^2$$

$$\Delta\omega_t = -\frac{\eta}{\sqrt{v_t + \epsilon} g_t}$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : Initial Learning rate
v_t : Exponential Average of squares of gradients
g_t : Gradient at time t along ω^i

Equation 10 Decay of epsilon

$$epsilon = min_e + (max_e - min_e)\frac{max_{expl} - curr_{step}}{max_{expl}}$$

III. RESULTS

For the testing, we assume perfect localization, perfect communication of the agents and we have a limit of 1000 individual time steps (movements) for each agent and episode. The testing is being made on 50x50 (grid x grid) maps with the walls occupying between 15-20% of the entire map (randomly generated each time).

A. Hyper parameters and optimizing algorithms

The array I shows the hyper-parameters of the all the models. The agents will get increased number of epochs for training based on the total coverage they achieved in the end of each episode. The replay steps variable is in every how many steps the agents will get trained. The variables beta (increase) and Prioritizing scale are the (hyper) parameters of the PER memory model. The target frequency is the frequency in which the main network model updates the values from the target network based on the equation 3. Finally, the memory buffer size is the size of the array that stores the experiences per batch.

	Model-1	Model-2	Model-3
Learning rate	0.00005	0.0005	0.0005
epochs	1,5,7	1,5,10	1,3,5
Batch size	32	32	32
Prioritizing Scale	0.9	0.8	0.9
Target Network Frequency	10.000	10.000	10.000
Memory buffer size	100.000	100.000	100.000
Optimizer	RmsProp	RmsProp	RmsProp
Loss Function	Huber Loss	Huber Loss	Huber Loss
Epsilon Decay scale	Equation 10	Equation 10	Equation 10
Replay Steps	2	10	4
Beta increase	Equation 10	Equation 10	Equation 10
Maximum Exploration	50000	25000	35000

TABLE I: Hyper parameter table

B. Main model results

First we compare the total coverage and the usefull coverage (how much they actually covered for the 1000 maximum time-step) for 1,2,3 and 4 agent case. For each case the agents learned to avoid the obstacles with 100% accuracy. The table II shows the median coverage for each different number of agents. Specifically, the first row shows the agent number scenario, the first column show individual coverage for each agent, the column use full coverage shows the % of newly discovered of the respected total step (1000) and finally the total coverage shows the total coverage from each agent.

Also, the graphs 2, 3, show the total coverage for 1,2 and 3,4 agents for general testing. The median for those cases are 32.56%, 46.17%, 53.53% and 59.4% for each agent case respectively.

	single agent		two agents		three agents		four agents	
	usefull cover	total cover	usefull coverage	total cover	usefull cover	total cover	usefull cover	total cover
agent-1	70.2%	32.56%	51.1%	24.05%	39.4%	18.13%	26.6%	12.21%
agent-2	-	-	47%	22.12%	44%	20.2%	28.6%	13.18%
agent-3	-	-	-	-	33%	15.2%	36.7%	16.89%
agent-4	-	-	-	-	-	-	37%	17.12%
total coverage	-	32.56%	-	46.17%	-	53.53%	-	59.4%

TABLE II: multi agent coverage for main model

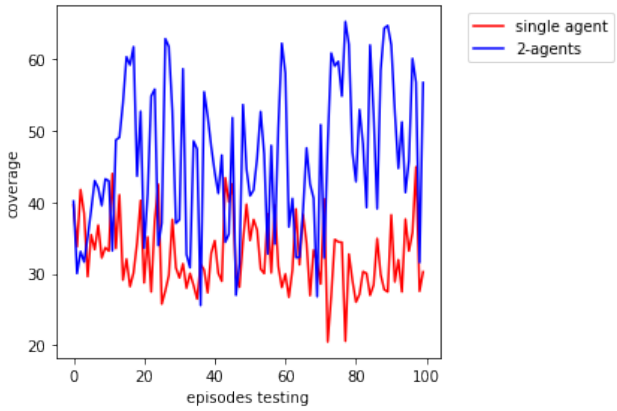


Fig. 2: 1 and 2 agents coverage

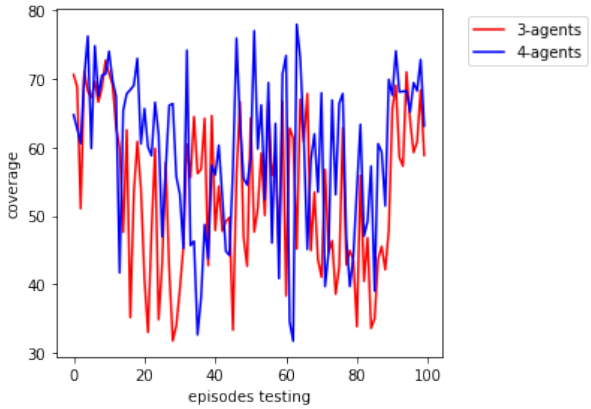


Fig. 3: 3 and 4agents coverage

C. Final Model

Firstly, the changed of the main model to the deeper one had an increase of 8.6% (median of 59.4% went to 68%) in the total coverage. Still we see episodes with low coverage. For the four agent case we see all the above in the figure 4.

Furthermore, for the second neural network (closest target position) the 5 graph shows that after the training episodes the agents learned to go to the target position, while avoiding the obstacles with 100% accuracy. Finally, the combined models with the switcher network had a raise in total median coverage of 23% (83% from 66%) for the 4 case agent. The coverage for the general testing can be seen in the graph 6. The median values (across several testing episodes) of the actions that the main and the target closest position network are taking are 2067 and 1933 respectively. Also, the table III shows the results of 4 agents for the main, the deep and the combined model and specifically the median coverage, the best and worst coverage and the peak to peak difference of them. The pathing of 4 agents can be seen in the figure 7. We observe that the combined model has increased greatly it's performance.

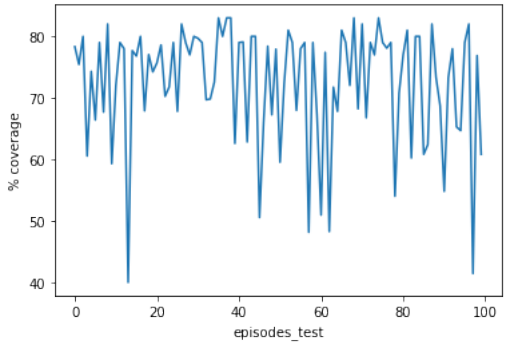


Fig. 4: Deeper model

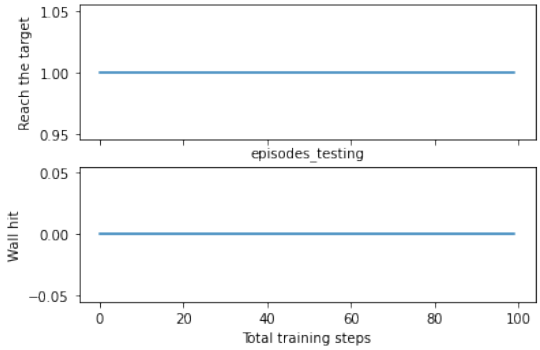


Fig. 5: Target Model

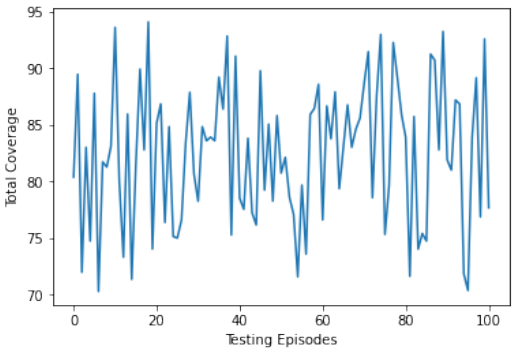


Fig. 6: Combined Model 4 agent case

4-agent case	Main Model	Deeper Model	Combined Model
Median coverage	59.4%	66%	83%
Best Coverage	78%	85%	94%
Worst Coverage	31.8	42%	72%
Peak to Peak Difference	46.2%	43%	22%

TABLE III: All model comparison

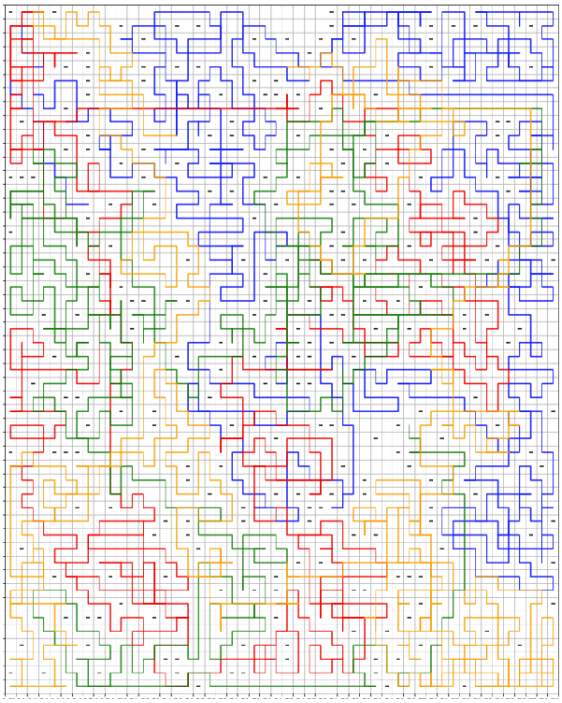


Fig. 7: Four agents path with 95% total coverage

IV. Discussions & Future Work

Summarizing, we introduced a method of solving the problem of exploration in unknown environments. Specifically, the algorithm we established is based on the Reinforcement Learning approach with the Q-learning as the main core of it. Multiple secondary algorithms were used on the main model (PER, DDQN) in order to achieve better performance and stability. The proposed method is not bound of any prior information of the environment, meaning that it does not need to know the size or the type of it, as it only needs the local information of it.

Continuing, the main model alone, it didn't offer very good results, as it achieved basic performance with the main criteria being the total coverage (avoiding obstacles wass 100% successful). Tho, with the introduction of the two new neural networks (one being responsible to generate actions to a target position and the other for choosing which network should generate actions each step) and also with the change of the architecture of the main model to a deeper (with lower total parameters nevertheless) we had a much greater performance and more stability (less variance). Specifically, we observed that the best episodes are getting a lot closer to the 100% coverage (94% across several testing episodes) and also the minimum is greatly increased (72%).

Moving on, we observe the importance of the hyperparameter of max time step. Specifically, there is no terminal case so the only parameter that ends the episode is the that. So, finding the optimal number of it requires more research, as also more research is again required for the optimal number of agents, based on the size of the environment and the variable of max steps. Finally, in order for the agents to explore all the areas and not leaving any of it unexplored, future research should be done. Also, stabilizing algorithms (like Target Neural Network) could be implemented on the main model in order to reduce the variance along the episodes.

References

[1] GD Cubber et al. "Search and rescue robotics-from theory to practice". In: *Search Rescue Robotics From Theory to Practice*. IntechOpen, 2017.

[2] Jnaneshwar Das et al. "Devices, systems, and methods for automated monitoring enabling precision agriculture". In: *2015 IEEE International Conference on Automation Science and Engineering (CASE)*. IEEE. 2015, pp. 462–469.

[3] Hado Hasselt. "Double Q-learning". In: *Advances in neural information processing systems* 23 (2010), pp. 2613–2621.

[4] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. "Reinforcement learning: A survey". In: *Journal of artificial intelligence research* 4 (1996), pp. 237–285.

[5] Bohao Li and Yunjie Wu. "Path planning for UAV ground target tracking via deep reinforcement learning". In: *IEEE Access* 8 (2020), pp. 29064–29074.

[6] Bruna G Maciel-Pearson et al. "Online deep reinforcement learning for autonomous UAV navigation and exploration of outdoor environments". In: *arXiv preprint arXiv:1912.05684* (2019).

[7] Serge Montambault and Nicolas Pouliot. "Design and validation of a mobile robot for power line inspection and maintenance". In: *Field and Service Robotics*. Springer. 2008, pp. 495–504.

[8] Jeremy Roghair et al. "A Vision Based Deep Reinforcement Learning Algorithm for UAV Obstacle Avoidance". In: *arXiv preprint arXiv:2103.06403* (2021).

[9] Tom Schaul et al. "Prioritized experience replay". In: *arXiv preprint arXiv:1511.05952* (2015).

[10] Alicja Wasik et al. "Graph-based distributed control for adaptive multi-robot patrolling through local formation transformation". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Ieee. 2016, pp. 1721–1728.

[11] Christopher JCH Watkins and Peter Dayan. "Q-learning". In: *Machine learning* 8.3-4 (1992), pp. 279–292.