Reinforcement Learning-Based Trajectory Planning For UAV-aided Vehicular Communications

Riccardo Marini, Leonardo Spampinato, Silvia Mignardi, Roberto Verdone, Chiara Buratti
WiLab, CNIT / DEI, University of Bologna, Bologna, Italy
{r.marini, leonardo.spampinato, silvia.mignardi, roberto.verdone, c.buratti}@unibo.it

***Abstract*—It is widely expected that 6G networks will rely on Unmanned Aerial Vehicles (UAVs) acting as flying Base Stations (BSs) to provide a wide range of services that current networks cannot handle. One of the major trends deals with Vehicle-To-Everything (V2X) communications, where vehicles must be connected to the network to offer applications such as advanced driving and extended sensing. In this context, vehicles could deeply count on flying BS to increase the throughput or reduce the experienced latency, thus satisfying such services constraints. Consequently, path planning must be designed so that UAVs can keep stable links with moving vehicles. In this sense, Reinforcement Learning (RL) techniques are becoming the main enabler for solving such problem, since they offer the possibility to learn how to act in an environment with little prior information, given that full knowledge of the scenario is usually not available. In this paper, we present a RL-based approach to solve the path planning problem in a vehicular scenario, where UAVs, exploiting beamforming, are required to follow as long as possible moving cars. Different RL architectures, as well as a benchmark solution not using RL, are compared to select the best strategy maximising the sum throughput.**

I. Introduction

With the advent of 5-th Generation (5G) and beyond networks, novel advanced paradigms target network scalability and seamless communications. Therefore, the use of UAVs as mobile BSs (i.e., Unmanned Aerial Base Stations (UABSs)), that may fly on-demand exactly when and where service is needed [1], [2], arises high interest and expectations. Among others, the use of UABSs is gaining attention in the context of future Vehicle-To-Everything (V2X) applications [**?**], [3], [4]. In fact, thanks to the mobility degree of freedom, a UABS may improve link robustness and network adaptability to the dynamics of a vehicular scenario. In this context, the design of the trajectory assumes a fundamental role. Since the continuous variation in time of the environment constitutes a critical challenge, recent trends for UAV trajectory design show that the use of Reinforcement Learning (RL) is particularly helpful [5]. Indeed, solving mathematical optimization models is not possible when a-priori input data is unavailable or requires too high complexity and computation time. To solve such problems, RL allows instead to learn in an environment with little prior information available. As it will be discussed later, RL balances the environment exploration done by an agent with the exploitation of acquired knowledge through time, aspects that allow to learn the dynamics of a vehicular scenario.

During the latest years, the navigation problem has become increasingly interesting and it has reached a new level of complexity in case the target of UAV service is moving itself. Authors in [6] adopt Q-learning for the trajectory control part, but the movement of the UAV is limited to the selection of a lane and the movement forward or backward on it. In [7], Authors employ Q-learning for 3D trajectories in a multi-UAV environment where pedestrian users are moving. In this case, the main goal of the agent is to keep as many users as possible connected to the network during roaming occasions. Authors in [8] study the optimal positions of UAV BSs in a sparse highway with a multi agent Q-learning algorithm. However, this work focuses on the positioning of the UAVs rather than the trajectory design. Moving vehicles are considered in [9], where Authors propose a RL approach for the minimization of the number of UAVs accounting for their energy consumption. In this case, the trajectory planning turns out to be very simple since an highway scenario is considered. In [10], authors present a path planning strategy based on users' initial position as well as next location prediction. To do so, they exploit Q-Learning for the trajectory design, which is fed with the prediction of users' movement generated according to Global Positioning System coordinates supposed to be available by mining data from social networks.

Differently from the works cited above, we consider a urban scenario with a city map, where vehicular users drive only on available roads, exploiting more complex Q-Learning based algorithms. Here, the UAV learns how to track the moving users to offer them service during their entire traveled path, rather than finding an optimal placement. A dynamic trajectory indeed allows the UAV to provide vehicles seamless access to the network. In addition, we are able to show that introducing beamforming and feeding beams-related information as a state input to the RL problem can be beneficial for the training performance. In fact, this provides the UAV with better knowledge regarding the vehicles location and the direction they are pursuing, improving the overall performance of the algorithm.

The remainder of the paper is organized as follows; the system model and the Machine Learning (ML) problem definition are presented in Sections II and III, respectively, whereas Section IV presents the obtained results. Finally, Section V concludes the paper.

II. System Model

A. Reference Scenario

We assume one UABS, u, and a set of vehicles, denoted as Ground User Equipments (GUEs), g, belonging to the set $\mathcal{G}$,

are present in the scenario of interest. The UABS is equipped with a radio antenna system enabling beamforming for the mmWave frequency bands; its position is given by $[x_t, y_t, h]$, with constant altitude h, at time instant t. We assume GUEs want to implement an *extended sensing* application, according to which, each vehicle wants to exchange data gathered through local sensors or video with other vehicles nearby [11]. To do this each vehicle sends V2X messages (e.g., Cooperative Awareness Message (CAM) and Collective Perception Message (CPM)), to the network, every t_{msg} seconds, with $t_{msg} \in [0.1, 1]$ seconds. In this paper we set $t_{msg} = 1$ s. Cars are moving in a urban environment, that in our case is a district of the city of Bologna (Italy). This city map is characterized by many possible paths each vehicle can take; their generation is based on Simulation of Urban MObility (SUMO), an open source traffic simulator [12] (an example is provided in Fig. 1). In order to improve the system performance, the UABS is expected to identify the most suitable trajectory to maximize network service.

Since the RL algorithm used for this work is based on a discrete set of actions, as it will be discussed in Section III-A, the UABS can move only in predetermined directions. To model this behaviour, the map (1460 x 760 m^2) is divided into a squared grid-world with vertices corresponding to the possible positions. The sides length of such square depends on the UABS speed, so the slower is the UABS, the more granular the grid will be (in our simulations, we obtain 2628, 1176 and 666 possible positions for the slowest, the medium and the fastest settings, respectively).

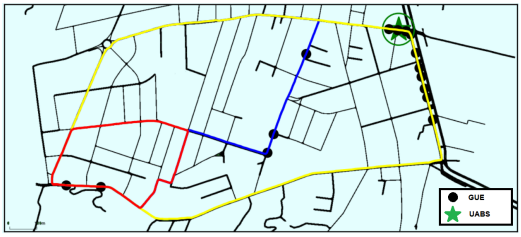

Fig. 1: Example: three possible paths (yellow, blue, red) taken by GUEs.

B. Channel Model

We assume a mmWave communication system working at a carrier frequency $f_c = 30$ GHz. Accordingly, the channel model adopted is provided by the 3rd Generation Partnership Project (3GPP) in [13], considering, specifically, the Urban Macro (UMa) scenario (more details in Chapter 7.4 of [13]). The model introduces probability for a link to be in Line-of-Sight (LoS) conditions (see Table 7.4.2-1 of [13]), p_{L}, that depends on the projected 2D terminal distance from the UABS, $d_{2\mathrm{D}}$ and user terminal height, h_{UT}. Therefore, the channel losses, l_{L} and l_{N} for LoS and Non Line-of-Sight (NLoS), respectively, depend also on other parameters characterizing the propagation link, such as the terminal height, h_{UT}, the UABS height, h, and the 3D distance between the UABS and the terminal. Slow fluctuations due to shadowing are considered with parameters $\sigma_{\mathrm{L}} = 4$ and $\sigma_{\mathrm{N}} = 6$ (dB), respectively. Consequently, each link in the scenario has a channel loss (in dB scale) $PL = l_{\mathrm{L}} + \sigma^*_{\mathrm{L}}$ with probability p_{L} or $PL = l_{\mathrm{N}} + \sigma^*_{\mathrm{N}}$ with a probability $1 - p_{\mathrm{L}}$, where σ^*_{L} and σ^*_{N} are the shadowing samples taken from the Gaussian distributions $\sigma^*_{\mathrm{L}} \sim \mathcal{N}\{0, \sigma_{\mathrm{L}}\}$ and $\sigma^*_{\mathrm{N}} \sim \mathcal{N}\{0, \sigma_{\mathrm{N}}\}$, respectively. where σ^*_{L} and σ^*_{N} are the samples taken from the Gaussian distributions $\mathcal{N} \sim \{0, \sigma_{\mathrm{L}}\}$ and $\mathcal{N} \sim \{0, \sigma_{\mathrm{N}}\}$, respectively.

Finally, we define the Signal-to-Noise Ratio (SNR) in dB as:

$$\mathrm{SNR} = [P_{\mathrm{tx}} + G_{\mathrm{tx}} + G_{\mathrm{rx}} - PL] - P_{\mathrm{noise}} \tag{1}$$

where P_{tx} is the transmitted power in dBm, G_{tx} and G_{rx} represent the gain in transmission and reception, respectively, in dB and P_{noise} is the noise power.

C. Beamforming

Tackling a 6G network scenario, we assume that transmissions take advantage from beamforming techniques. In particular, we assume the UABS is using a fixed Grid of Beams, operating on a set of $N_{\mathrm{beam}} = 9$ resulting on the ground as circular spots and arranged in a 3x3 grid. We define $\mathbf{\Phi}$ the solid angle deriving from the radiation pattern and $\alpha_{beam} \approx \mathbf{\Phi}/N_{\mathrm{beam}}$ as the solid angle of a beam. Consequently, the maximum gain G can be expressed as [14]:

$$G = \frac{41000}{(\alpha_{beam} \frac{360}{2\pi})^2} \tag{2}$$

For the sake of simplicity, we assume the radiation pattern is ideal, with gain G inside the angle α_{beam}, and zero outside. We also refer to ϕ as the 2D angle of the UABS's vertical plane, from which we infer the field of view of the UABS projected on the ground.

III. Machine Learning Model

A. Reinforcement Learning Problem Definition

We assume time t is discretized into steps, of duration t_{msg}. According to standard RL problems, an agent interacts with an environment during an episode, lasting T steps. At each step, starting from state s_t, belonging to the state space, $\mathcal{S}$, the agent chooses an action, a_t, from a set of possible actions, $\mathcal{A}$, according to its policy π. After selecting such action, the agent moves to state $s_{t+1} \in \mathcal{S}$ and receives a scalar reward $r_t = r(s_t, a_t)$ based on a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$.

To this end, we define:

- an *agent* as the UABS, u, whose target is to design the trajectory which maximizes a reward function;
- a *state* in the state space that consists of the agent location, the time instant and number of vehicles under each beam. The agent position is expressed in (x, y) coordinates at the relative time instant t. We assume that the UABS is able to get information related to the number of GUEs under its field of view, therefore we also define a vector $\mathbf{b}_t$, in which each element $b_{t,i}$ is equal to the number of GUEs under the i-th beam at time instant t. Therefore, a state is defined as $s_t = \{x_t, y_t, t, \mathbf{b}_t\}$;

- an *action* of the agent as a movement on the map in one of 8 possible directions or hovering. Therefore, the Action Space is defined as $\mathcal{A} = \{0, \leftarrow, \uparrow, \rightarrow, \downarrow, \nwarrow, \nearrow, \searrow, \swarrow\}$, where 0 represents the decision to stay still;
- the *reward* measures the benefit of selecting a specific action a while being in a state s. Since the agent aims at maximizing the number of served GUEs, it is computed as the sum throughput of the vehicles seen by the UABS at time instant t. Therefore, it is defined as

$$r_t = \sum_{g \in \mathcal{G}} r_t^{(g)} := \sum_{g \in \mathcal{G}} B_{\text{ch}} \log(1 + \text{SNR}_t^{(g)}), \quad (3)$$

where $r_t^{(g)}$ indicates the capacity of the $g-th$ GUE at time t for the UABS in state s_t and action a_t, and where B_{ch} is the channel bandwidth.

B. Q-Learning

Q-Learning is a model free RL algorithm which consists of a agent interacting with an environment in order to learn and optimize a behavior. In particular, it aims at iteratively improving the state-action value function, or Q-function, which represents an expectation of the discounted cumulative future reward R_t from the current state s_t up to the last step T:

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[R_t | s_t = s, a_t = a, \pi] \quad (4)$$

where $\mathbb{E}_{\pi}$ is the expected value under policy π and with R_t given by:

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r(s_t, a_t) \quad (5)$$

where $\gamma \in [0, 1]$ represents the discount factor, which balances the importance of the immediate and the future reward.

Given a transition $< s_t, a_t, r_t, s_{t+1} >$, $Q_{(s,a)}$ can be expressed by the Bellman equation in terms of Q-value of the next state s_{t+1}:

$$Q(s_t, a_t) = r_t + \gamma \max_a Q(s_{t+1}, a) \quad (6)$$

In other words, it is possible to express the Q-value as the sum of the immediate reward and the discounted future reward of the state that follows, without the need of calculating each value as the sum of the expected cumulative reward.

In its simplest form, Q-Learning exploits the Q-Table, a look up table in which the Q-value for each state-action pair is stored and regularly updated. The agent chooses an action based on an epsilon greedy policy [15], thus the chosen action may be random with probability ϵ or it can be the action with the highest Q-Value with probability 1-ϵ. Each time an action is chosen, the updated Q-Value is computed as:

$$\begin{aligned} Q^{new}(s_t, a_t) &= Q(s_t, a_t) + \\ &+ \alpha(r_t + \gamma \max_a Q(s_{(t+1)}, a) - Q(s_t, a_t)) \end{aligned} \quad (7)$$

where α represents the learning rate, which determines how new information overrides old information.

C. Deep Q-Learning

The main drawback of Q-Learning is that in case of high dimensional state space, the Q-Table would require too much time to be created, as well as too much space to be stored. For these reasons, Deep Q-Learning (DQN) has been introduced in order to represent the policy π or other learned functions as a deep neural network which, taking the state as input, estimate the Q-values for all the different actions an agent may take.

Q-values can be any real values, which makes the problem a regression task, thus optimized with a function of the error loss between the predicted and the true values, estimated using Eq. 6. Both true and predicted target values are used to calculate the loss and consequently to update the neural network weights, leading to a huge correlation between target values and network weights. To avoid convergence issues and make the training more stable, two different networks are used:

- the *action network*, with parameters θ, represents the predicted value and it is used to select the action the agent takes and it is updated every n steps;
- the *target network*, with parameters θ^-, is a clone of the *action network* used only to compute the true value and it is updated every $m >> n$ steps by copying the action network's weights.

Therefore, the loss is computed as:

$$L = f(r_t + \gamma \max_a Q(s_{t+1}, a, \theta^-) - Q(s_t, a_t, \theta)) \quad (8)$$

As f, we chose the Huber Loss [16].

D. Double Deep Q-Learning

In [17], authors claims that the standard Q-Learning algorithm is known to overestimate action values under certain conditions. In order to prevent such overestimation, a Double Deep Q-Learning (DDQN) algorithm is presented. The idea behind this algorithm is to decouple the selection from the evaluation, by exploiting the *action network* to select the action, whereas the corresponding Q-Value is estimated using the *target network*.

Therefore, the loss function changes as:

$$\begin{aligned} L = &f(r_t + \gamma Q(s_{t+1}, \arg\max_a Q(s_{t+1}, a, \theta), \theta^-) - \\ &- Q(s_t, a_t, \theta)) \end{aligned} \quad (9)$$

E. Dueling Deep Q-Learning

Given the policy π, it is possible to define the state-value function $V_{(s)}^{\pi} = \mathbb{E}_{a \sim \pi(s)}[Q_{(s,a)}^{\pi}]$, which represents the expected total reward following policy π starting from state s.

Consequently, it is possible to define the Advantage function as

$$A^{\pi}(s, a) = Q^{\pi}(s, a) - V^{\pi}(s) \quad (10)$$

which represents how advantageous would be an action with respect to the others at the given state following policy π.

Dueling DQN [18] introduction is motivated by the fact that it is unnecessary to know the outcome of each action at each time step, so, exploiting two separate streams (which are recombined in order to generate the Q-values) for the

estimation of the State values and the Advantages, the dueling architecture can learn separately which states are important from which action is better to select. This also provides the possibility to integrate it in the classical DQN schemes, whose functioning remains the same, since no modification to inputs and outputs are made.

IV. Numerical Results

Parameters set in simulations are reported in Table I. During training, we set the Buffer Size N $= 500000$ and the Batch Size K $= 128$.

TABLE I: Network and Channel Parameters

Parameter	Notation	Value
Number of GUEs	N_g	15
Average GUE speed	v_g	10 m/s
GUE transmit power	$P_{\text{tx},g}$	20 dBm
GUE transmission gain	$G_{\text{tx},g}$	0 dB
UABS altitude	h_u	100 m
UABS transmit power	$P_{\text{tx},u}$	23 dBm
Carrier frequency	f_c	30 GHz
Channel bandwidth	B_{ch}	1.44 MHz
Effective noise power	P_{noise}	-106.4 dBm
Episode Length	T	380 s
Learning Rate	α	0.001
Discount Factor	γ	0.8
Buffer Size	N	500k
Batch Size	K	128
Action Network Update	n	1
Target Network Update	m	500

Results are presented in terms of total reward, which is the sum of the reward obtained by the agent at each step, that is $R = \sum_{t=0}^{T} r_t$, where T is the length of one episode and r_t has been defined in Eq. (3).

Fig. 2 shows the total reward as a function of the number of episodes for different algorithms presented in Section III, when setting v_u=20 m/s and $\phi = 140°$. Note that, by changing ϕ, the antenna gain at the UABS will change according to Eq (2). It can be clearly seen that the dueling DDQN architecture offers more advantages, since the dueling DDQN and the dueling DQN perform better with respect to the other architectures. The reason behind this may be explained by the fact that the dueling architecture is able to learn which states are important regardless of the action to take, which, in our case, means that the agent is able to detect which are the most important locations (i.e., states) to reach first and then optimize how to reach them. Surprisingly, the double architecture alone does not provide good results, even worse than the standard DQN. This may happen due to the fact that the Q-value, in the double architecture, is estimated exploiting the *target network*, which is updated slower w.r.t. the *action network*, as explained in Section III-C. This penalizes the knowledge the agent may have acquired and which is not used until the next *target network* update. In addition, the figure shows how the introduction of the beam information $\mathbf{b}_t$ into the state definition

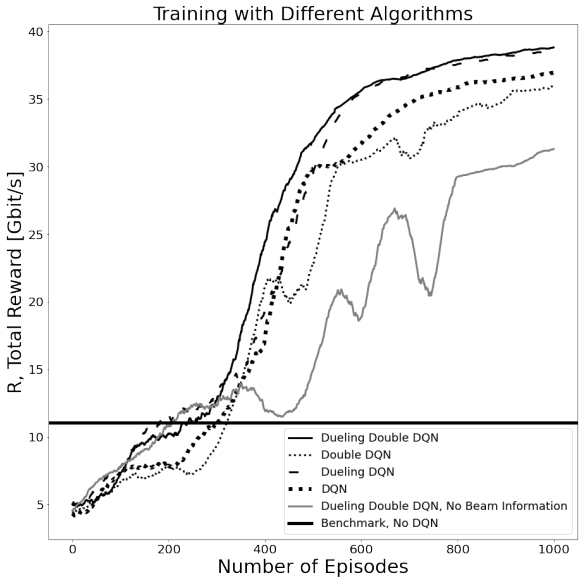

Fig. 2: Comparison between different algorithms for v_u=20 m/s and $\phi = 140°$.

gives a huge boost in terms of training time, allowing the algorithms to converge after 1000 episodes. On the other hand, the removal of such information, which changes the state definition to $s_t = \{x_t, y_t, t\}$, increases the time needed for the algorithms to converge up to 10000 episodes, since the agent has less data at its disposal. Finally, the horizontal line shows the performance of a UABS taking decisions without using any RL-based solution, but only selecting each action by going towards the direction of the beam with the highest number of vehicles inside at each time step. This allows us to compare the RL architectures with a benchmark, where the UABS is exploiting the information available but it is not actually learning how to use it properly. As it can be clearly seen, the benchmark performance is very poor w.r.t. those achieved when RL is used.

On the overall, we consider the Dueling DDQN to be the best solution, since it converges faster w.r.t. the standard Dueling architecture. Therefore, next results will be presented considering only such algorithm.

Fig. 3 shows the total reward as a function of the number of episodes for different UABS speed, v_u, having set $\phi = 140°$. Results are comparable in terms of total reward obtained at the end of the training, even though higher speeds offer faster convergence mainly because the agent is able to explore the entire map faster. In addition, it can be noticed that choosing such speeds do not worsen the performance, even if they are higher w.r.t. the average GUEs speed.

Finally, Fig. 4 depicts the total reward as a function of the number of episodes for three different overall field of view of the UABS, ϕ, when setting v_u=20 m/s. As can be seen, a larger field of view, and therefore a larger angle of aperture per beam, provides better results mainly because more GUEs

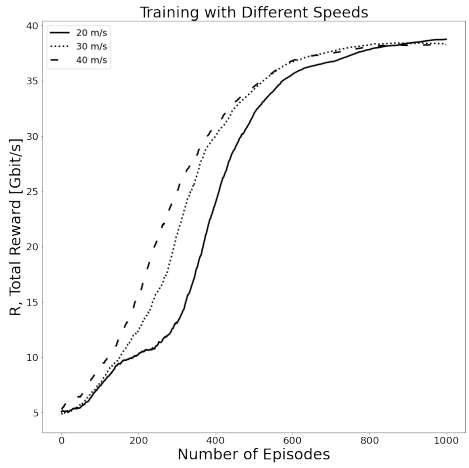

Fig. 3: Comparison between different UABS speeds with Dueling DDQN for $\phi = 140°$.

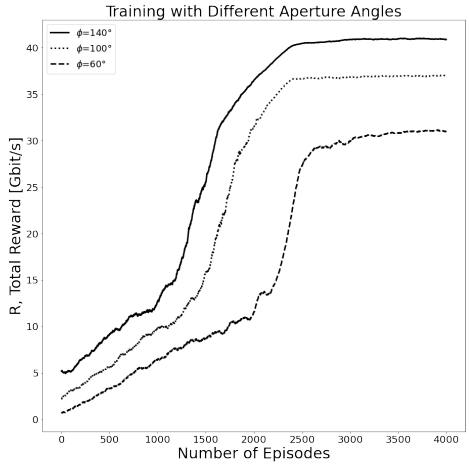

Fig. 4: Comparison between different UABS aperture angles with Dueling DDQN for v_u=20 m/s.

are seen by the UABS. In addition, the beam angle of aperture helps also in the detection of the GUE movement direction, resulting in a faster learning.

V. Conclusion

In this paper we presented different RL architectures to solve the trajectory design problem in UABS network providing services to vehicular applications. Such architectures are based on a family of Q-Learning algorithm which are able to generate a good trajectory by letting the agent explore the environment, resulting in letting the UABS track the vehicles' movement and follow them during their path. We proved that the dueling architecture is particularly suitable for the target tracking problem and that the inclusion of beamforming information in the state definition can help solve such problem. Finally, we conducted an analysis on different UABS related parameters, showing that there exists an optimal configuration which leads to better results in terms of network throughput.

VI. Acknowledgment

This work has been carried out in the framework of the CNIT National Laboratory WiLab and the WiLab-Huawei Joint Innovation Center. We would like to thank Aman Jassal of Huawei for the very fruitful discussion on this paper.

References

[1] M.-A. Lahmeri, M. A. Kishk, and M.-S. Alouini, "Artificial Intelligence for UAV-Enabled Wireless Networks: A Survey," *IEEE Open Journal of the Communications Society*, vol. 2, p. 1015–1040, 2021. [Online]. Available: http://dx.doi.org/10.1109/OJCOMS.2021.3075201

[2] A. I. Hentati and L. C. Fourati, "Comprehensive survey of UAVs communication networks," *Computer Standards and Interfaces*, vol. 72, p. 103451, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0920548919303411

[3] Houari, Abdeslam and Mazri, Tomader, "Improving V2X-6G network capacity using a new UAV-based approach in a Cloud/ICN architecture, case Study: VANET network," *E3S Web Conf.*, vol. 297, p. 01019, 2021.

[4] B. Shang, L. Liu, J. Ma, and P. Fan, "Unmanned Aerial Vehicle Meets Vehicle-to-Everything in Secure Communications," *IEEE Communications Magazine*, vol. 57, no. 10, pp. 98–103, 2019.

[5] P. S. Bithas, E. T. Michailidis, N. Nomikos, D. Vouyioukas, and A. G. Kanatas, "A Survey on Machine-Learning Techniques for UAV-Based Communications," *Sensors*, vol. 19, no. 23, 2019. [Online]. Available: https://www.mdpi.com/1424-8220/19/23/5170

[6] L. Deng, G. Wu, J. Fu, Y. Zhang, and Y. Yang, "Joint Resource Allocation and Trajectory Control for UAV-Enabled Vehicular Communications," *IEEE Access*, vol. 7, pp. 132 806–132 815, 2019.

[7] X. Liu, Y. Liu, and Y. Chen, "Reinforcement Learning in Multiple-UAV Networks: Deployment and Movement Design," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 8036–8049, 2019.

[8] B. Jiang, S. N. Givigi, and J.-A. Delamer, "A MARL Approach for Optimizing Positions of VANET Aerial Base-Stations on a Sparse Highway," *IEEE Access*, vol. 9, pp. 133 989–134 004, 2021.

[9] M. Samir, D. Ebrahimi, C. Assi, S. Sharafeddine, and A. Ghrayeb, "Trajectory Planning of Multiple Dronecells in Vehicular Networks: A Reinforcement Learning Approach," *IEEE Networking Letters*, vol. 2, no. 1, pp. 14–18, 2020.

[10] X. Liu, Y. Liu, Y. Chen, and L. Hanzo, "Trajectory Design and Power Control for Multi-UAV Assisted Wireless Networks: A Machine Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 8, pp. 7957–7969, 2019.

[11] "5G: service requirements for enhanced V2X scenarios," *ETSI*, no. ETSI TS 22.186version 16.2.0, November, 2020.

[12] P. A. L. et al., "Microscopic Traffic Simulation using SUMO," *2019 IEEE ITSC*, pp. 2575–2582, November 2018. [Online]. Available: https://elib.dlr.de/127994/

[13] 3GPP, "Technical Specification Group Radio Access Network; Study on channel model for frequencies from 0.5 to 100 GHz," *TR 38 901 version 16.1.0*, Dec. 2019.

[14] V. K. Salvia, *Antenna and wave propagation*. Laxmi, 2007.

[15] V. M. et al., "Playing Atari with Deep Reinforcement Learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[16] "Huber Loss, PyTorch implementation," https://pytorch.org/docs/stable/generated/torch.nn.HuberLoss, Accessed: 2022-01-28.

[17] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, March 2016.

[18] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," *Proceedings of the 33rd International Conference on on Machine Learning*, p. 1995–2003, 2016.