Received 14 March 2021; revised 24 April 2021; accepted 17 May 2021. Date of publication 20 May 2021; date of current version 1 June 2021.

Digital Object Identifier 10.1109/OJCOMS.2021.3081996

Multi-UAV Path Planning for Wireless Data Harvesting With Deep Reinforcement Learning

HARALD BAYERLEIN [1] (Student Member, IEEE), MIRCO THEILE [2] (Student Member, IEEE), MARCO CACCAMO [2] (Fellow, IEEE), AND DAVID GESBERT [1] (Fellow, IEEE)

[1]Communication Systems Department, EURECOM, 06904 Sophia Antipolis, France

[2]TUM Department of Mechanical Engineering, Technical University of Munich, 80333 Munich, Germany

CORRESPONDING AUTHOR: H. BAYERLEIN (e-mail: harald.bayerlein@eurecom.fr)

The work of Harald Bayerlein and David Gesbert was supported in part by the French government, through the 3IA Côte d'Azur Project under Grant ANR-19-P3IA-0002, and in part by the TSN CARNOT Institute under Project Robots4IoT. The work of Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. This article was presented in part at IEEE GLOBECOM 2020 [1]. The code for this work is available under https://github.com/hbayerlein/uav_data_harvesting.

ABSTRACT Harvesting data from distributed Internet of Things (IoT) devices with multiple autonomous unmanned aerial vehicles (UAVs) is a challenging problem requiring flexible path planning methods. We propose a multi-agent reinforcement learning (MARL) approach that, in contrast to previous work, can adapt to profound changes in the scenario parameters defining the data harvesting mission, such as the number of deployed UAVs, number, position and data amount of IoT devices, or the maximum flying time, without the need to perform expensive recomputations or relearn control policies. We formulate the path planning problem for a cooperative, non-communicating, and homogeneous team of UAVs tasked with maximizing collected data from distributed IoT sensor nodes subject to flying time and collision avoidance constraints. The path planning problem is translated into a decentralized partially observable Markov decision process (Dec-POMDP), which we solve through a deep reinforcement learning (DRL) approach, approximating the optimal UAV control policy without prior knowledge of the challenging wireless channel characteristics in dense urban environments. By exploiting a combination of centered global and local map representations of the environment that are fed into convolutional layers of the agents, we show that our proposed network architecture enables the agents to cooperate effectively by carefully dividing the data collection task among themselves, adapt to large complex environments and state spaces, and make movement decisions that balance data collection goals, flight-time efficiency, and navigation constraints. Finally, learning a control policy that generalizes over the scenario parameter space enables us to analyze the influence of individual parameters on collection performance and provide some intuition about system-level benefits.

INDEX TERMS Internet of Things (IoT), map-based planning, multi-agent reinforcement learning (MARL), trajectory planning, unmanned aerial vehicle (UAV).

I. INTRODUCTION

AUTONOMOUS unmanned aerial vehicles (UAVs) are not only envisioned as passive cellular-connected users of telecommunication networks but also as active connectivity enablers [2]. Their fast and flexible deployment makes them especially useful in situations where terrestrial infrastructure is overwhelmed or destroyed, e.g., in disaster and search-and-rescue situations [3], or where fixed coverage is in any way lacking. UAVs have shown particular promise in collecting data from distributed Internet of Things (IoT) sensor nodes. For instance, IoT operators can deploy UAV data harvesters in the absence of otherwise expensive

cellular infrastructure nearby. Another reason is the throughput efficiency benefits related to having UAVs that describe a flight pattern that brings them close to the IoT devices. As an example in the context of infrastructure maintenance and preserving structural integrity, Hitachi is already commercially deploying partially autonomous UAVs that collect data from IoT sensors embedded in large structures, such as the San Juanico and Agas-Agas Bridges in the Philippines [4]. Research into UAV-aided data collection from IoT devices or wireless sensors include the works [5]–[9], with [10]–[13] concentrating on minimizing the age of information of the collected data. Additional coverage of past related work is offered in the next section.

In this work, we focus on controlling a team of UAVs, consisting of a variable number of identical drones tasked with collecting varying amounts of data from a variable number of stationary IoT sensor devices at variable locations in an urban environment. This imposes challenging constraints on the trajectory design for autonomous UAVs. In addition, the limited on-board battery energy density restricts mission duration for quadcopter drones severely. At the same time, the complex urban environment poses challenges in obstacle avoidance and adherence to regulatory no-fly zones (NFZs). Additionally, the wireless communication channel is characterized by random signal blocking events due to alternating between line-of-sight (LoS) and non-line-of-sight (NLoS) links. We believe this work is the first to address multi-UAV path planning where learned control policies are generalized over a wide scenario parameter space and can be directly applied when scenario parameters change without the need for retraining.

While some challenges to real-world deep reinforcement learning (DRL) such as limited training samples, safety and lack of explainable actions remain, DRL offers the opportunity to balance challenges and data collection goals for complex environments in a straightforward way by combining them in the reward function. Another reason for the popularity of the DRL paradigm in this context is the computational efficiency of DRL inference. DRL is also one of the few methods that allows us to tackle the complex task directly, given that UAV control and deployment in communication scenarios are generally non-convex optimization problems [2], [14]–[18], and proven to be NP-hard in many instances [2], [16], [17]. These advantages of DRL also hold for other UAV path planning instances, such as coverage path planning [19], a classical robotics problem where the UAV's goal is to cover all points inside an area of interest. The equivalence of these path planning problems and the connection between the often disjoint research areas is highlighted in [20].

A. RELATED WORK

A survey that spans the various application areas for multi-UAV systems from a cyber-physical perspective is provided in [17]. The general challenges and opportunities of UAV communications are summarized in publications by Zeng *et al.* [2] and Saad *et al.* [18], which both include data collection from IoT devices. This specific scenario is also included in [21] and [22], surveys that comprise information on the classification of UAV communication applications with a focus on DRL methods.

Path planning for UAVs providing some form of communication services or collecting data has been studied extensively, including numerous approaches based on reinforcement learning (RL). However, it is crucial to note that the majority of previous works concentrates on only finding the optimal trajectory solution for one set of scenario parameters at a time, requiring full or partial retraining if the scenario changes. In contrast, our approach aims to train and generalize over a large scenario parameter space directly, finding efficient solutions without the need for lengthy retraining, but also increasing the complexity of the path planning problem significantly.

Many existing RL approaches also only focus on single-UAV scenarios. An early proposal given in [23] to use (deep) RL in a related scenario where a single UAV base station serves ground users shows the advantages of using a deep Q-network (DQN) over table-based Q-learning, while not making any explicit assumptions about the environment at the price of long training time. The authors in [5] only investigate table-based Q-learning for UAV data collection. A particular variety of IoT data collection is the one tackled in [10], where the authors propose a DQN-based solution to minimize the age of information of data collected from sensors. In contrast to our approach, the mentioned approaches are set in much simpler environments and agents have to undergo computationally expensive retraining when scenario parameters change.

Multi-UAV path planning for serving ground users employing table-based Q-learning is investigated in [16], based on a relatively complex 3-step algorithm consisting of grouping the users with a genetic algorithm, then deployment and movement design in two separated instances of Q-learning. The investigated optimization problem is proven to be NP-hard, with Q-learning being confirmed as a useful tool to solve it. Pan *et al.* [6] investigate an instance of multi-UAV data collection from sensor nodes formulated as a classical traveling salesman problem without modeling the communication phase between UAV and node. The UAVs' trajectories are designed with a genetic algorithm that uses some aspects of DRL, namely training a deep neural network and experience replay. In contrast to the multi-stage optimization algorithms in [6] and [16], our approach consists of a more straightforward end-to-end DRL approach that scales to large and complex environments, generalizing over varying scenario parameters.

The combination of DRL and multi-UAV control has been studied previously in various scenarios. The authors in [11] focus on trajectory design for minimizing the age of information of sensing data generated by multiple UAVs themselves where the data can be either transmitted to terrestrial base stations or mobile cellular devices. Their focus lies on balancing the UAV sensing and transmission

protocol in an unobstructed environment for one set of scenario parameters at a time. Other MARL path planning approaches to minimize the age of information of collected data include [12] and [13]. In [24], a swarm of UAVs on a target detection and tracking mission in an unknown environment is controlled through a distributed DQN approach. While the authors also use convolutional processing to feed map information to the agents, the map is initially unknown and has to be explored to detect the targets. The agents' goal is to learn transferable knowledge that enables adaptation to new scenarios with fast relearning, compared to our approach to learn a control policy that generalizes over scenario parameters and requires no relearning.

Hu *et al.* [14] proposed a distributed multi-UAV meta-learning approach to control a group of drone base stations serving ground users with random uplink access demands. While meta-learning allows them to reduce the number of training episodes needed to adapt to a new unseen uplink demand scenario, several hundred are still required. Our approach focuses on training directly on random but observable scenario parameters within a given value range, therefore not requiring retraining to adapt. Due to the small and obstruction-less environment, no maps are required in [14] and navigation constraints are omitted by keeping the UAVs at dedicated altitudes. In [7], multi-agent deep Q-learning is used to optimize trajectories and resource assignment of UAVs that collect data from pre-defined clusters of IoT devices and provide power wirelessly to them. The focus here is on maximizing minimum throughput in a wirelessly powered network without a complex environment and navigation constraints, only for a single scenario at a time. Similarly, in [8] there is also a strong focus on the energy supply of IoT devices through backscatter communications when a team of UAVs collects their data. The authors propose a multi-agent approach that relies on the definition of ambiguous boundaries between clusters of sensors. The scenario is set in a simple, unobstructed environment, not requiring maps or adherence to multiple navigation constraints, but requiring retraining when scenario parameters change.

In [25], a group of interconnected UAVs is tasked with providing long-term communication coverage to ground users cooperatively. While the authors also formulate a POMDP that they solve by a DRL variant, there is no need for map information or processing. The scenario is set in a simple environment without obstacles or other navigation constraints. This work was extended under the paradigm of mobile crowdsensing, where mobile devices are leveraged to collect data of common interest in [9]. The authors proposed a heterogeneous multi-agent DRL algorithm collecting data simultaneously with ground and aerial vehicles in an environment with obstacles and charging stations. While in this work, the authors also suggest a convolutional neural network to exploit a map of the environment, the small grid world does not necessitate extensive map processing. Furthermore, they do not center the map on the agent's position, which is highly beneficial [1]. In contrast to our method, control policies have to be relearned entirely in a lengthy training process for both mentioned approaches when scenario and environmental parameters change.

B. CONTRIBUTIONS

If DRL methods are to be applied in any real-world mission, the prohibitively high training data demand poses one of the most severe challenges [26]. This is exacerbated by the fact that even minor changes in the scenario, such as in the number or location of sensor devices in data collection missions, typically requires repeating the full training procedure of the DRL agent. This is the case for existing approaches such as [7]–[9], [11]–[13], [23], [25]. Other approaches to reduce the training data demand include meta-learning [14] and transfer learning [26]. To the best of our knowledge, this is the first work that addresses this problem in path planning for multi-UAV data harvesting by proposing a DRL method that is able to generalize over a large space of scenario parameters in complex urban environments without prior knowledge of wireless channel characteristics based on centered global-local map processing.

The main contributions of this paper are the following.

- We formulate a flying time constrained multi-UAV path planning problem to maximize harvested data from IoT sensors. We consider its translation to a decentralized partially observable Markov decision process (Dec-POMDP) with full reward function description in large, complex, and realistic environments that include no-fly zones, buildings that block wireless links (some possible to be flown over, some not), and dedicated start/landing zones.
- To solve the Dec-POMDP under navigation constraints without any prior knowledge of the urban environment's challenging wireless propagation conditions, we employ deep multi-agent reinforcement learning with centralized learning and decentralized execution.
- We show the advantage in learning and adaptation efficiency to large maps and state spaces through a dual global-local map approach with map centering over more conventional scalar neural network input in a multi-UAV setting.
- As perhaps our most salient feature, our algorithm offers *parameter generalization*, which means that the learned control policy can be reused over a wide array of scenario parameters, including the number of deployed UAVs, variable start positions, maximum flying times, and number, location and data amount of IoT sensor devices, without the need to restart the training procedure as typically required by existing DRL approaches.
- Learning a generalized control policy enables us to compare performance over a large scenario parameter space directly. We analyze the influence of individual parameters on collection performance and provide some intuition about system-level benefits.

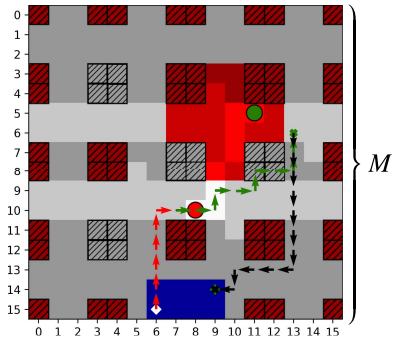


FIGURE 1. Example of a single UAV collecting data from two IoT devices in an urban environment of size $M \times M$ with NFZs, a single start/landing zone, and buildings causing shadowing. Small buildings can be flown over and tall buildings act as navigation obstacles.

TABLE 1. Legend for scenario plots.

	Symbol	Description
DQN Input		Start and landing zone
		Regulatory no-fly zone (NFZ)
		Tall buildings* (UAVs cannot fly over)
		Small buildings* (UAVs can fly over)
		IoT device
		Other agents
		*all buildings obstruct wireless links
Visualization		Summation of building shadows
		Starting and landing positions during an episode
		UAV movement while comm. with green device
		Hovering while comm. with green device
		Actions without comm. (all data collected)

C. ORGANIZATION

The paper is organized as follows: Section II introduces the multi-UAV mobility and communication channel model, which is translated to an MDP in Section III and followed by a description of the proposed map preprocessing in Section IV and multi-agent DRL learning approach in Section V. Simulation results and their discussion follow in Section VI, and we conclude the paper with a summary and outlook to future work in Section VII.

II. SYSTEM MODEL

In the following, we present the key models for the multi-UAV path planning problem. Note that some level of simplification is needed when modeling the robots' dynamics in order to enable the implementation of the RL approach. Our assumptions are explicit whenever suitable.

We consider a square grid world of size $M \times M \in \mathbb{N}^2$ with cell size c and the set of all possible positions $\mathcal{M}$. Discretization of the environment is a necessary condition for our map-processing approach, however note that our method can be applied to any rectangular grid world. The environment contains L designated start/landing positions given by the set

$$\mathcal{L} = \left\{ \left[x_i^l, y_i^l\right]^{\mathrm{T}},\ i = 1, \ldots, L,\ : \left[x_i^l, y_i^l\right]^{\mathrm{T}} \in \mathcal{M} \right\}$$

and the combination of the Z positions the UAVs cannot occupy is given by the set

$$\mathcal{Z} = \left\{ \left[x_i^z, y_i^z\right]^{\mathrm{T}},\ i = 1, \ldots, Z,\ : \left[x_i^z, y_i^z\right]^{\mathrm{T}} \in \mathcal{M} \right\}.$$

This includes tall buildings which the UAVs can not fly over and regulatory no-fly zones (NFZ). The number of B obstacles blocking wireless links are given by the set

$$\mathcal{B} = \left\{ \left[x_i^b, y_i^b\right]^{\mathrm{T}},\ i = 1, \ldots, B,\ : \left[x_i^b, y_i^b\right]^{\mathrm{T}} \in \mathcal{M} \right\},$$

representing all buildings, also smaller ones that can be flown over. The lowercase letters l, z, b indicate the coordinates of the respective set of environmental features $\mathcal{L}, \mathcal{Z}, \mathcal{B}$. An example of a grid world is depicted in Fig. 1, where obstacles, NFZs, start/landing zone, and an example of a single UAV trajectory are marked as described in the attached legend in Tab. 1.

A. UAV MODEL

The set $\mathcal{I}$ of I deployed UAVs moves within the limits of the grid world $\mathcal{M}$. The state of the i-th UAV is described through its:

- position $\mathbf{p}_i(t) = [x_i(t), y_i(t), z_i(t)]^{\mathrm{T}} \in \mathbb{R}^3$ with altitude $z_i(t) \in \{0, h\}$, either at ground level or in constant altitude h;
- operational status $\phi_i(t) \in \{0, 1\}$, either inactive or active;
- battery energy level $b_i(t) \in \mathbb{N}$.

Note that the assumption of all UAVs sharing the same flying altitude is not too restrictive and that our method allows each UAV to fly at a different altitude as long as it remains constant throughout the mission. The UAV agent's altitude can be made observable by simply adding it to the observation space along the flying time. This work only tackles 2D trajectory optimization, as the environment is dominated by high-rise buildings that would require long climbing phases to be overflown. The mission time limited by the UAVs' on-board batteries restricts the effectiveness of 3D control for the data collection performance given that climbing flight consumes more energy [27] and that the UAVs needs to land at ground level at the end of the mission. The data collection mission is over after $T \in \mathbb{N}$ mission time steps for all UAVs, where the time horizon is discretized into equal mission time slots $t \in [0, T]$ of length δ_t seconds.

The action space of each UAV is defined as

$$\mathcal{A} = \left\{ \underbrace{\begin{bmatrix}0\\0\\0\end{bmatrix}}_{\text{hover}}, \underbrace{\begin{bmatrix}c\\0\\0\end{bmatrix}}_{\text{east}}, \underbrace{\begin{bmatrix}0\\c\\0\end{bmatrix}}_{\text{north}}, \underbrace{\begin{bmatrix}-c\\0\\0\end{bmatrix}}_{\text{west}}, \underbrace{\begin{bmatrix}0\\-c\\0\end{bmatrix}}_{\text{south}}, \underbrace{\begin{bmatrix}0\\0\\-h\end{bmatrix}}_{\text{land}} \right\}. \quad (1)$$

Each UAV's movement actions $\mathbf{a}_i(t) \in \tilde{\mathcal{A}}(\mathbf{p}_i(t))$ are limited to

$$\tilde{\mathcal{A}}(\mathbf{p}_i(t)) = \begin{cases} \mathcal{A}, & \mathbf{p}_i(t) \in \mathcal{L} \\ \mathcal{A} \setminus [0, 0, -h]^{\mathrm{T}}, & \text{otherwise}, \end{cases} \quad (2)$$

where $\tilde{\mathcal{A}}$ defines the set of feasible actions depending on the respective UAV's position, specifically that the landing action is only allowed if the UAV is in the landing zone.

The distance the UAV travels within one time slot is equivalent to the cell size c. Mission time slots are chosen sufficiently small so that each UAV's velocity $v_i(t)$ can be considered to remain constant in one time slot. The UAVs are limited to moving with horizontal velocity $V = c/\delta_t$ or standing still, i.e., $v_i(t) \in \{0, V\}$ for all $t \in [0, T]$. Each UAV's position evolves according to the motion model given by

$$\mathbf{p}_i(t+1) = \begin{cases} \mathbf{p}_i(t) + \mathbf{a}_i(t), & \phi_i(t) = 1 \\ \mathbf{p}_i(t), & \text{otherwise}, \end{cases} \tag{3}$$

keeping the UAV stationary if inactive. The evolution of the operational status $\phi_i(t)$ of each UAV is given by

$$\phi_i(t+1) = \begin{cases} 0, & \mathbf{a}_i(t) = [0, 0, -h]^{\mathrm{T}} \\ & \vee \ \phi_i(t) = 0 \\ 1, & \text{otherwise}, \end{cases} \tag{4}$$

where the operational status becomes inactive when the UAV has safely landed. The end of the data harvesting mission T is defined as the time slot when all UAVs have reached their terminal state and are not actively operating anymore, i.e., the operational state is $\phi_i(t) = 0$ for all UAVs.

The i-th UAV's battery content evolves according to

$$b_i(t+1) = \begin{cases} b_i(t) - 1, & \phi_i(t) = 1 \\ b_i(t), & \text{otherwise}, \end{cases} \tag{5}$$

assuming a constant energy consumption while the UAV is operating and zero energy consumption when operation has terminated. This is a simplification justified by the fact that power consumption for small quadcopter UAVs is dominated by the hovering component. Using the model from [27], the ratio between the additional power necessary for horizontal flight at 10m/s and just hovering could be roughly estimated as 30W/310W $\approx$ 10%, which is negligible. Considering power consumption of on-board computation and communication hardware which does not differ between flight and hovering, the overall difference becomes even smaller. In the following, we will refer to the battery content as remaining flying time, as it is directly equivalent.

The overall multi-UAV mobility model is restricted by the following constraints:

$$\mathbf{p}_i(t) \neq \mathbf{p}_j(t) \vee \phi_j(t) = 0, \quad \forall i, j \in \mathcal{I}, i \neq j, \forall t \tag{6a}$$

$$\mathbf{p}_i(t) \notin \mathcal{Z}, \quad \forall i \in \mathcal{I}, \forall t \tag{6b}$$

$$b_i(t) \geq 0, \quad \forall i \in \mathcal{I}, \forall t \tag{6c}$$

$$\mathbf{p}_i(0) \in \mathcal{L} \wedge z_i(0) = h, \quad \forall i \in \mathcal{I} \tag{6d}$$

$$\phi_i(0) = 1, \quad \forall i \in \mathcal{I} \tag{6e}$$

The constraint (6a) describes collision avoidance among active UAVs with the exception that UAVs can land at the same location. Equation (6b) forces the UAVs to avoid collisions with tall obstacles and prevents them from entering NFZs. The constraint (6c) limits operation time of the drones, forcing UAVs to end their mission before their battery has run out. Since operation can only be concluded with the landing action as described in (4) and the landing action is only available in the landing zone as defined in (2), the constraint (6c) ensures that each UAV safely lands in the landing zone before their batteries are empty. The starting constraint (6d) defines that the UAV start positions are in the start/landing zones and that their starting altitude is h, while (6e) ensures that the UAVs start in the active operational state.

B. COMMUNICATION CHANNEL MODEL

1) LINK PERFORMANCE MODEL

As communication systems typically operate on a smaller timescale than the UAVs' mission planning system, we introduce the notion of communication time slots in addition to mission time slots. We partition each mission time slot $t \in [0, T]$ into a number of $\lambda \in \mathbb{N}$ communication time slots. The communication time index is then $n \in [0, N]$ with $N = \lambda T$. One communication time slot n is of length $\delta_n = \delta_t/\lambda$ seconds. The number of communication time slots per mission time slot λ is chosen sufficiently large so that the i-th UAV's position, which is interpolated linearly between $\mathbf{p}_i(t)$ and $\mathbf{p}_i(t+1)$, and the channel gain can be considered to stay constant within one communication time slot.

The k-th IoT device is located on ground level at $\mathbf{u}_k = [x_k, y_k, 0]^{\mathrm{T}} \in \mathbb{R}^3$ with $k \in \mathcal{K}$ where $|\mathcal{K}| = K$. Each IoT sensor has a finite amount of data $D_k(t) \in \mathbb{R}^+$ that needs to be picked up over the whole mission time $t \in [0, T]$. The device data volume is set to an initial value at the start of the mission $D_k(t = 0) = D_{k,init}$. The data volume of each IoT node evolves depending on the communication time index n over the whole mission time, given by $D_k(n)$ with $n \in [0, N]$, $N = \lambda T$.

We follow the same UAV-to-ground channel model as used in [1]. The communication links between UAVs and the K IoT devices are modeled as LoS/NLoS point-to-point channels with log-distance path loss and shadow fading. The maximum achievable information rate at time n for the k-th device is given by

$$R_{i,k}^{\max}(n) = \log_2\big(1 + \mathrm{SNR}_{i,k}(n)\big). \tag{7}$$

Considering the amount of data available at the k-th device $D_k(n)$, the effective information rate is given as

$$R_{i,k}(n) = \begin{cases} R_{i,k}^{\max}(n), & D_k(n) \geq \delta_n R_{i,k}^{\max}(n) \\ D_k(n)/\delta_n, & \text{otherwise}. \end{cases} \tag{8}$$

The SNR with transmit power $P_{i,k}$, white Gaussian noise power at the receiver σ^2, UAV-device distance $d_{i,k}$, path loss exponent α_e and $\eta_e \sim \mathcal{N}(0, \sigma_e^2)$ modeled as a Gaussian random variable, is defined as

$$\mathrm{SNR}_{i,k}(n) = \frac{P_{i,k}}{\sigma^2} \cdot d_{i,k}(n)^{-\alpha_e} \cdot 10^{\eta_e/10}. \tag{9}$$

Note that the urban environment with the set of obstacles $\mathcal{B}$ hindering free propagation causes a strong dependence of the propagation parameters on the $e \in \{\mathrm{LoS}, \mathrm{NLoS}\}$ condition and that (9) is the SNR averaged over small scale fading. We

would also like to point out that our DQN-based trajectory planning approach is model-free and does therefore not rely on any specific channel model. While a more accurate and complex model could be directly used with our approach, the most important features for data collection missions of the urban channel, the dependence of SNR on $d_{i,k}$ and the $e \in \{\text{LoS, NLoS}\}$ condition, are already captured in (9).

2) MULTIPLE ACCESS PROTOCOL

The multiple access protocol is assumed to follow the standard time-division multiple access (TDMA) model when it comes to the communication between one single UAV and the various ground nodes. We further assume that the communication channel between the ground nodes and a given UAV operates on resource blocks (time-frequency slots) that are orthogonal to the channels linking the ground nodes and other UAVs, so that no inter-UAV interference exists in our model and inter-UAV synchronization is not necessary. Hence, the UAVs are similar to base stations that would be assigned orthogonal spectral resources. We also assume that IoT devices are operating in multi-band mode, hence are capable of simultaneously communicating with all UAVs on the set of all orthogonal frequencies. As a consequence, scheduling decisions are not part of the action space. The number of available orthogonal subchannels for UAV-to-ground communication is one of the variable scenario parameters and equivalent to the number of deployed UAVs.

Designing multiple access protocols for UAV networks is in itself a challenging research problem [28] due to high mobility of the nodes and fast changing link performance and is out of scope for this work. However, our proposed algorithm can in principle be integrated with existing solutions and does not rely on any specific channel model or multiple access protocol. While our model avoids and does not consider inter-UAV interference, we would like to point out that the behavior of the UAV agents that emerges naturally during the learning process of dividing the data collection task geographically, as illustrated in Section VI-D, would mitigate the influence of interference on the trajectory planning decisions to some extent.

Our scheduling protocol is assumed to follow the max-rate rule: in each communication time slot $n \in [0, N]$, the sensor node $k \in [1, K]$ with the highest $\text{SNR}_{i,k}(n)$ with remaining data to be uploaded is picked by the scheduling algorithm. The TDMA constraint for the scheduling variable $q_{i,k}(n) \in \{0, 1\}$ is given by

$$\sum_{k=1}^{K} q_{i,k}(n) \leq 1, \;\; n \in [0, N], \forall i \in \mathcal{I}. \tag{10}$$

It follows that the k-th device's data volume evolves within one communication time slot according to

$$D_k(n+1) = D_k(n) - \sum_{i=1}^{I} q_{i,k}(n) R_{i,k}(n) \delta_n. \tag{11}$$

The achievable throughput for the i-th UAV for one mission time slot $t \in [0, T]$, comprised of λ communication time slots, is the sum of rates achieved in the communication time slots $n \in [\lambda t, \lambda(t+1) - 1]$ over K sensor nodes. It depends on the UAV's operational status $\phi_i(t)$ and is given by

$$C_i(t) = \phi_i(t) \sum_{n=\lambda t}^{\lambda(t+1)-1} \sum_{k=1}^{K} q_{i,k}(n) R_{i,k}(n) \delta_n. \tag{12}$$

C. OPTIMIZATION PROBLEM

Using the described UAV model in II-A and communication model in II-B, the central goal of the multi-UAV path planning problem is the maximization of throughput over the whole mission time and over all I deployed UAVs while adhering to mobility constraints (6a)-(6e) and the scheduling constraint (10). The maximization problem is given by

$$\begin{aligned} \max_{\times_i \mathbf{a}_i(t)} \; & \sum_{t=0}^{T} \sum_{i=1}^{I} C_i(t). \\ \text{s.t.} \; & \text{(6a), (6b), (6c), (6d), (6e), (10)} \end{aligned} \tag{13}$$

optimizing over joint actions $\times_i \mathbf{a}_i(t)$.

III. MARKOV DECISION PROCESS (DEC-POMDP)

To address the aforementioned optimization problem, we translate it to a decentralized partially observable Markov decision process (Dec-POMDP) [29], which is defined through the tuple $(\mathcal{S}, \mathcal{A}_\times, P, R, \Omega_\times, \mathcal{O}, \gamma)$. In the Dec-POMDP, $\mathcal{S}$ describes the state space, $\mathcal{A}_\times = \mathcal{A}^I$ the joint action space, and $P : \mathcal{S} \times \mathcal{A}_\times \times \mathcal{S} \mapsto \mathbb{R}$ the transition probability function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function mapping state, individual action, and next state to a real valued reward. The joint observation space is defined through $\Omega_\times = \Omega^I$ and $\mathcal{O} : \mathcal{S} \times \mathcal{I} \mapsto \Omega$ is the observation function mapping state and agents to one agent's individual observation. The discount factor $\gamma \in [0, 1]$ controls the importance of long vs. short term rewards.

A. STATE SPACE

The state space of the multi-agent data collection problem consists of the environment information, the state of the agents, and the state of the devices. It is given as

$$\begin{aligned} \mathcal{S} = \Big\{ & \underbrace{\mathcal{L}}_{\substack{\text{Landing}\\\text{Zones}}} \times \underbrace{\mathcal{Z}}_{\text{NFZs}} \times \underbrace{\mathcal{B}}_{\text{Obstacles}} \Big\} \text{ Environment} \\ \times \Big\{ & \underbrace{\mathbb{R}^{I \times 3}}_{\substack{\text{UAV}\\\text{Positions}}} \times \underbrace{\mathbb{N}^I}_{\substack{\text{Flying}\\\text{Times}}} \times \underbrace{\mathbb{B}^I}_{\substack{\text{Operational}\\\text{Status}}} \Big\} \text{ Agents} \\ \times \Big\{ & \underbrace{\mathbb{R}^{K \times 3}}_{\substack{\text{Device}\\\text{Positions}}} \times \underbrace{\mathbb{R}^K}_{\substack{\text{Device}\\\text{Data}}} \Big\} \text{ Devices} \end{aligned} \tag{14}$$

in which the elements $s(t) \in \mathcal{S}$ are

$$s(t) = (\mathbf{M}, \{\mathbf{p}_i(t)\}, \{b_i(t)\}, \{\phi_i(t)\}, \{\mathbf{u}_k\}, \{D_k(t)\}), \quad (15)$$

$\forall i \in \mathcal{I}$ and $\forall k \in \mathcal{K}$, in which $\mathbf{M} \in \mathbb{B}^{M \times M \times 3}$ is the tensor representation of the set of start/landing zones $\mathcal{L}$, obstacles and NFZs $\mathcal{Z}$, and obstacles only $\mathcal{B}$. The other elements of the tuple define positions, remaining flying times, and operational status of all agents, as well as positions and available data volume of all IoT devices.

B. SAFETY CONTROLLER

To enforce the collision avoidance constraint (6a) and the NFZ and obstacle avoidance constraint (6b), a safety controller is introduced into the system. Additionally, the safety controller enforces the limited action space excluding the *landing* action when the respective agent is not in the landing zone as defined in (2). The safety controller evaluates the action $\mathbf{a}_i(t)$ of agent i and determines if it should be accepted or rejected. If rejected, the resulting safe action is the *hovering* action. The safe action $\mathbf{a}_{s,i}(t)$ is thus defined as

$$\mathbf{a}_{s,i}(t) = \begin{cases} [0,0,0]^{\mathrm{T}}, & \mathbf{p}_i(t) + \mathbf{a}_i(t) \in \mathcal{Z} \\ & \vee\ \mathbf{p}_i(t) + \mathbf{a}_i(t) = \mathbf{p}_j(t) \wedge \phi_j(t) = 1, \\ & \forall j, j \neq i \\ & \vee\ \mathbf{a}_i(t) = [0,0,-h]^{\mathrm{T}} \ \wedge\ \mathbf{p}_i(t) \notin \mathcal{L} \\ \mathbf{a}_i(t), & \text{otherwise.} \end{cases} \quad (16)$$

Without path planning capabilities, the safety controller cannot enforce the flying time and safe landing constraint in (6c). Therefore, we relax the hard constraint on flight time by adding a high penalty on not landing in time instead. In the simulation, a crashed agent, i.e., an agent with $b_i(t) < 0$, is defined as not operational.

C. REWARD FUNCTION

The reward function $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ of the Dec-POMDP is comprised of the following elements:

$$r_i(t) = \alpha \sum_{k \in \mathcal{K}} (D_k(t+1) - D_k(t)) + \beta_i(t) + \gamma_i(t) + \epsilon. \quad (17)$$

The first term of the sum is a collective reward for the collected data from all devices by all agents within mission time slot t. It is parameterized through the data collection multiplier α. This is the only part of the reward function that is shared among all agents. The second addend is an individual penalty when the safety controller rejects an action and given through

$$\beta_i(t) = \begin{cases} \beta, & \mathbf{a}_i(t) \neq \mathbf{a}_{i,s}(t) \\ 0, & \text{otherwise.} \end{cases} \quad (18)$$

It is parameterized through the safety penalty β. The third term is the individual penalty for not landing in time given by

$$\gamma_i(t) = \begin{cases} \gamma, & b_i(t+1) = 0 \wedge \mathbf{p}_i(t+1) = [\cdot, \cdot, h]^{\mathrm{T}} \\ 0, & \text{otherwise.} \end{cases} \quad (19)$$

TABLE 2. Million floating point operations (MFLOPs) needed for inference of the networks based on map-processing.

Map	No Processing	Centering	Centering + Global-Local
Manhattan32	15	80	7.7
Urban50	45	217	6.5

and parameterized through the crashing penalty γ. The last term is a constant movement penalty parameterized through ϵ, which is supposed to incentivize the agents to reduce their flying time and prioritize efficient trajectories.

IV. MAP-PROCESSING AND OBSERVATION SPACE

To aid the agents in interpreting the large state space given in (14), we implement two map processing steps. The first is centering the map around the agent's position, shown in [1] to significantly improve the agent's learning performance. This benefit is a consequence of neurons in the layer after the convolutional layers (compare Fig. 3) corresponding to features *relative* to the agent's position, rather than to *absolute* positions if the map is not centered. This is advantageous as one agent's actions are solely based on its relative position to features, e.g., its distance to sensor devices. The downside of this approach is that it increases the size of the maps and the observation space even further, therefore requiring larger networks with more trainable parameters.

The second map processing step is to present the centered map as a compressed global and uncompressed but cropped local map as previously evaluated in [20]. In path planning, as distant features lead to general direction decisions while close features lead to immediate actions such as collision avoidance, the level of detail passed to the agent for distant objects can be less than for close objects. The advantage is that the compression of the global map reduces the necessary neural network size considerably.

This reduction in network size directly translates to a reduction in computational load. Table 2 shows the number of floating point operations needed for each of the two maps under different map processing regimes as given by the TensorFlow graph profiler. Only centering increases the computational load considerably, as explained in [1], while global-local map processing offsets the increase and reduces floating point operations considerably. Considering that modern embedded processors operate in the region of giga floating point operations, it seems realistic that the required processing can be carried out even on small and energy-limited UAVs. The mathematical descriptions of the map processing functions and the observation space are detailed in the following.

A. MAP-PROCESSING

For ease of exposition, we introduce the 2D projections of the UAV and IoT device positions on the ground, $\tilde{\mathbf{u}}_k \in \mathbb{N}^2$ and $\tilde{\mathbf{p}}_k \in \mathbb{N}^2$ respectively, given by

$$\tilde{\mathbf{u}}_k = \left\lfloor \begin{pmatrix} \frac{1}{c} & 0 & 0 \\ 0 & \frac{1}{c} & 0 \end{pmatrix} \mathbf{u}_k \right\rfloor, \quad \tilde{\mathbf{p}}_i = \left\lfloor \begin{pmatrix} \frac{1}{c} & 0 & 0 \\ 0 & \frac{1}{c} & 0 \end{pmatrix} \mathbf{p}_i \right\rfloor \quad (20)$$

rounded to integer grid coordinates.

1) MAPPING

The centering and global-local mapping algorithms are based on map-layer representations of the state space. To represent any state with a spatial aspect given by a position and a corresponding value as a map-layer, we define a general mapping function

$$f_{\text{mapping}} : \mathbb{N}^{Q\times 2} \times \mathbb{R}^{Q} \mapsto \mathbb{R}^{M\times M}. \tag{21}$$

In this function, a map layer $\mathbf{A} \in \mathbb{R}^{M\times M}$ is defined as

$$\mathbf{A} = f_{\text{mapping}}\big(\{\tilde{\mathbf{p}}_q\}, \{v_q\}\big), \tag{22}$$

with a set of grid coordinates $\{\tilde{\mathbf{p}}_q\}$ and a set of corresponding values $\{v_q\}$. The elements of $\mathbf{A}$ are given through

$$a_{\tilde{p}_{q,0},\tilde{p}_{q,1}} = v_q, \quad \forall q \in [0, \ldots, Q-1] \tag{23}$$

or 0 if the index is not in the grid coordinates. With this general function, we define the map-layers

$$\mathbf{D}(t) = f_{\text{mapping}}(\{\tilde{\mathbf{u}}_k\}, \{D_k(t)\}) \tag{24a}$$
$$\mathbf{B}(t) = f_{\text{mapping}}(\{\tilde{\mathbf{p}}_i(t)\}, \{b_i(t)\}) \tag{24b}$$
$$\Phi(t) = f_{\text{mapping}}(\{\tilde{\mathbf{p}}_i(t)\}, \{\phi_i(t)\}) \tag{24c}$$

for device data, UAV flying times, and UAV operational status respectively. If the map-layers are of same type they can be stacked to form a tensor of $\mathbb{R}^{M\times M\times n}$ for ease of representation.

2) MAP CENTERING

Given a tensor $\mathbf{A} \in \mathbb{R}^{M\times M\times n}$ describing the map-layers, a centered tensor $\mathbf{B} \in \mathbb{R}^{M_c\times M_c\times n}$ with $M_c = 2M-1$ is defined through

$$\mathbf{B} = f_{\text{center}}\big(\mathbf{A}, \tilde{\mathbf{p}}, \mathbf{x}_{\text{pad}}\big), \tag{25}$$

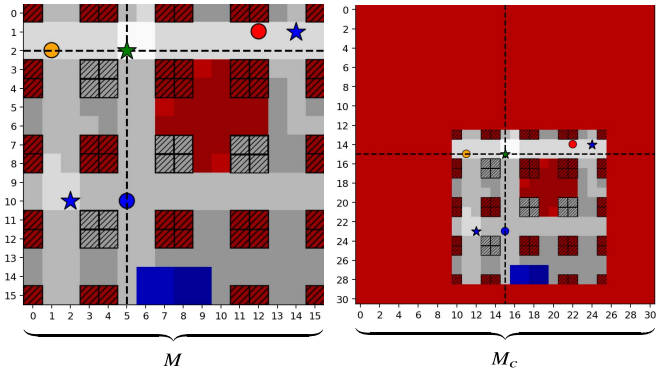
with the centering function defined as

$$f_{\text{center}} : \mathbb{R}^{M\times M\times n} \times \mathbb{N}^2 \times \mathbb{R}^n \mapsto \mathbb{R}^{M_c\times M_c\times n}. \tag{26}$$

The elements of $\mathbf{B}$ with respect to the elements of $\mathbf{A}$ are defined as

$$\mathbf{b}_{i,j} = \begin{cases} \mathbf{a}_{i+\tilde{p}_0-M+1, j+\tilde{p}_1-M+1}, & M \le i+\tilde{p}_0+1 < 2M \\ & \wedge\ M \le j+\tilde{p}_1+1 < 2M \\ \mathbf{x}_{\text{pad}}, & \text{otherwise}, \end{cases} \tag{27}$$

effectively padding the map layers of $\mathbf{A}$ with the padding value $\mathbf{x}_{\text{pad}}$. Note that $\mathbf{a}_{i,j}$, $\mathbf{b}_{i,j}$, and $\mathbf{x}_{\text{pad}}$ are vector valued of dimension $\mathbb{R}^n$. An illustration of the centering on a 16×16 map ($M = 16$, $M_c = 31$) can be seen in Figure 2 with the legend in Table 1.



(a) Non-centered input map (b) Centered input map

FIGURE 2. Comparison of non-centered and centered input maps, with UAV position represented by the green star and the intersection of the dashed lines.

3) GLOBAL-LOCAL MAP

The tensor $\mathbf{B} \in \mathbb{R}^{M_c\times M_c\times n}$ resulting from the map centering function is processed in two ways. The first is creating a local map according to

$$\mathbf{X} = f_{\text{local}}(\mathbf{B}, l) \tag{28}$$

with the local map function defined by

$$f_{\text{local}} : \mathbb{R}^{M_c\times M_c\times n} \times \mathbb{N} \mapsto \mathbb{R}^{l\times l\times n}. \tag{29}$$

The elements of $\mathbf{X}$ with respect to the elements of $\mathbf{B}$ are defined as

$$\mathbf{x}_{i,j} = \mathbf{b}_{i+M-\left\lceil \frac{l}{2} \right\rceil,\ j+M-\left\lceil \frac{l}{2} \right\rceil} \tag{30}$$

This operation is effectively a central crop of size $l \times l$.

The second processing creates a global map according to

$$\mathbf{Y} = f_{\text{global}}(\mathbf{B}, g) \tag{31}$$

with the global map function

$$f_{\text{global}} : \mathbb{R}^{M_c\times M_c\times n} \times \mathbb{N} \mapsto \mathbb{R}^{\left\lfloor \frac{M_c}{g} \right\rfloor \times \left\lfloor \frac{M_c}{g} \right\rfloor \times n} \tag{32}$$

The elements of $\mathbf{Y}$ with respect to the elements of $\mathbf{B}$ are defined as

$$\mathbf{y}_{i,j} = \frac{1}{g^2} \sum_{u=0}^{g-1} \sum_{v=0}^{g-1} \mathbf{b}_{gi+u, gj+v} \tag{33}$$

This operation is equal to an average pooling operation with pooling cell size g.

The functions f_{local} and f_{global} are parameterized through l and g, respectively. Increasing l increases the size of the local map, whereas increasing g increases the size of the average pooling cells, therefore decreasing the size of the global map.

B. OBSERVATION SPACE

Using the map processing functions, the observation space can be defined. The observation space Ω, which is the input space to the agent, is given as

$$\Omega = \underbrace{\Omega_l}_{\substack{\text{Local}\\\text{Map}}} \times \underbrace{\Omega_g}_{\substack{\text{Global}\\\text{Map}}} \times \underbrace{\mathbb{N}}_{\substack{\text{Flying}\\\text{Time}}}$$

containing the local map

$$\Omega_l = \mathbb{B}^{l\times l\times 3} \times \mathbb{R}^{l\times l} \times \mathbb{N}^{l\times l} \times \mathbb{B}^{l\times l}$$

and the global map

$$\Omega_g = \mathbb{R}^{\bar{g}\times\bar{g}\times 3} \times \mathbb{R}^{\bar{g}\times\bar{g}} \times \mathbb{R}^{\bar{g}\times\bar{g}} \times \mathbb{R}^{\bar{g}\times\bar{g}}.$$

with $\bar{g} = \lfloor \frac{M_c}{g} \rfloor$. Note that the compression of the global map through average pooling transforms all map layers into $\mathbb{R}$. Observations $o_i(t) \in \Omega$ are defined through the tuple

$$\begin{aligned} o_i(t) = \big(&\mathbf{M}_{l,i}(t), \mathbf{D}_{l,i}(t), \mathbf{B}_{l,i}(t), \Phi_{l,i}(t),\\ &\mathbf{M}_{g,i}(t), \mathbf{D}_{g,i}(t), \mathbf{B}_{g,i}(t), \Phi_{g,i}(t), b_i(t)\big). \end{aligned} \tag{34}$$

In one observation tuple, $\mathbf{M}_{l,i}(t)$ is the local observation of agent i of the environment, $\mathbf{D}_{l,i}(t)$ is the local observation of the data to be collected, $\mathbf{B}_{l,i}(t)$ is the local observation of the remaining flying time of all agents, and $\Phi_{l,i}(t)$ is the local observation of the operational status of the agents. $\mathbf{M}_{g,i}(t)$, $\mathbf{D}_{g,i}(t)$, $\mathbf{B}_{g,i}(t)$, and $\Phi_{g,i}(t)$ are the respective global observations. $b_i(t)$ is the remaining flying time of agent i, which is equal to the one in the state space. Note that the environment map's local and global observations are dependent on time, as they are centered around the UAV's time-dependent position. Additionally, it should be noted that the remaining flying time of agent i is given in the center of $\mathbf{B}_{l,i}(t)$ and additionally as a scalar $b_i(t)$. This redundancy in representation helps the agent to interpret the remaining flying time.

Consequently, the complete mapping from state to observation space is given by

$$O : \mathcal{S} \times \mathcal{I} \mapsto \Omega \tag{35}$$

in which the elements of $o_i(t)$ are defined as follows:

$$\mathbf{M}_{l,i}(t) = f_{\text{local}}\big(f_{\text{center}}\big(\mathbf{M}, \mathbf{p}_i(t), [0,1,1]^{\mathrm{T}}\big), l\big) \tag{36a}$$
$$\mathbf{D}_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}_i(t), 0), l) \tag{36b}$$
$$\mathbf{B}_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{B}(t), \mathbf{p}_i(t), 0), l) \tag{36c}$$
$$\Phi_{l,i}(t) = f_{\text{local}}(f_{\text{center}}(\Phi(t), \mathbf{p}_i(t), 0), l) \tag{36d}$$
$$\mathbf{M}_{g,i}(t) = f_{\text{global}}\big(f_{\text{center}}\big(\mathbf{M}, \mathbf{p}_i(t), [0,1,1]^{\mathrm{T}}\big), g\big) \tag{36e}$$
$$\mathbf{D}_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}_i(t), 0), g) \tag{36f}$$
$$\mathbf{B}_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{B}(t), \mathbf{p}_i(t), 0), g) \tag{36g}$$
$$\Phi_{g,i}(t) = f_{\text{global}}(f_{\text{center}}(\Phi(t), \mathbf{p}_i(t), 0), g) \tag{36h}$$

By passing the observation space Ω into the agent instead of the state space $\mathcal{S}$ as done in the previous approaches [1] and [19], the presented path planning problem is artificially converted into a partially observable MDP. Partial observability is a consequence of the restricted size of the local map and the compression of the global map. However, as shown in [20], partial observability does not render the problem infeasible, even for a memory-less agent. Instead, the compression greatly reduces the neural network's size, leading to a significant reduction in training time.

V. MULTI-AGENT REINFORCEMENT LEARNING (MARL)

A. Q-LEARNING

Q-learning is a model-free RL method [30] where a cycle of interaction between one or multiple agents and the environment enables the agents to learn and optimize a behavior, i.e., the agents observe state $s_t \in \mathcal{S}$ and each performs an action $a_t \in \mathcal{A}$ at time t and the environment subsequently assigns a reward $r(s_t, a_t) \in \mathbb{R}$ to the agents. The cycle restarts with the propagation of the agents to the next state s_{t+1}. The agents' goal is to learn a behavior rule, referred to as a policy that maximizes their reward. A probabilistic policy $\pi(a|s)$ is a distribution over actions given the state such that $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. In the deterministic case, it reduces to $\pi(s)$ such that $\pi : \mathcal{S} \to \mathcal{A}$.

Q-learning is based on iteratively improving the state-action value function or Q-function to guide and evaluate the process of learning a policy π. It is given as

$$Q^{\pi}(s, a) = \mathbb{E}_{\pi}[G_t | s_t = s, a_t = a] \tag{37}$$

and represents an expectation of the discounted cumulative return G_t from the current state s_t up to a terminal state at time T given by

$$G_t = \sum_{k=t}^{T} \gamma^{k-t} r(s_k, a_k) \tag{38}$$

with $\gamma \in [0, 1]$ being the discount factor, balancing the importance of immediate and future rewards. For the ease of exposition, s_t and a_t are abbreviated to s and a, while s_{t+1} and a_{t+1} are abbreviated to s' and a' in the following.

B. DOUBLE DEEP Q-LEARNING AND COMBINED EXPERIENCE REPLAY

As demonstrated in [23], representing the Q-function (37) as a table of values is not efficient in the large state and action spaces of UAV trajectory planning. Instead, a deep Q-network (DQN) parameterizing the Q-function with the parameter vector θ can be trained to minimize the expected temporal difference (TD) error. While a neural network is significantly more data efficient compared to a Q-table due to its ability to generalize, the *deadly triad* [30] of function approximation, bootstrapping and off-policy training can make its training unstable and cause divergence.

Mnih *et al.* [31] applied stabilizing techniques to the DQN training process, such as experience replay, reducing correlations in the sequence of training data. New experiences made by the agent, represented by quadruples of (s, a, r, s'), are stored in the replay memory $\mathcal{D}$. During training, a minibatch

of size m is sampled uniformly from $\mathcal{D}$ and used to compute the loss. The size of the replay memory $|\mathcal{D}|$ was shown to be an essential hyperparameter for the agent's learning performance and typically must be carefully tuned for different tasks or scenarios. Zhang and Sutton [32] proposed combined experience replay as a remedy for this sensitivity with very low computational complexity $O(1)$. In this extension to the replay memory method, only $m-1$ samples of the minibatch are sampled from memory, and the latest experience the agent made is always added. This corrected minibatch is then used to train the agent. Therefore, all new transitions influence the agent immediately, making the agent less sensitive to the selection of the replay buffer size in our approach.

In addition to experience replay, Mnih *et al.* used a separate target network for the estimation of the next maximum Q-value, giving the loss as

$$L^{\mathrm{DQN}}(\theta) = \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - Y^{\mathrm{DQN}}\left(s,a,s'\right)\right)^2\right] \quad (39)$$

with target value

$$Y^{\mathrm{DQN}}\left(s,a,s'\right) = r(s,a) + \gamma \max_{a'} Q_{\bar{\theta}}\left(s',a'\right). \quad (40)$$

$\bar{\theta}$ represents the parameters of the target network. The parameters of the target network $\bar{\theta}$ can either be updated as a periodic hard copy of θ or as in our approach with a soft update

$$\bar{\theta} \leftarrow (1-\tau)\bar{\theta} + \tau\theta \quad (41)$$

after each update of θ. $\tau \in [0,1]$ is the update factor determining the adaptation pace.

Further improvements to the training process were suggested in [33], resulting in the inception of double deep Q-networks (DDQNs). With the application of this extension, we avoid the overestimation of action values under certain conditions in standard DQN and arrive at the loss function for our network given by

$$L^{\mathrm{DDQN}}(\theta) = \mathbb{E}_{s,a,s'\sim\mathcal{D}}\left[\left(Q_\theta(s,a) - Y\left(s,a,s'\right)\right)^2\right] \quad (42)$$

where the target value is given by

$$Y^{\mathrm{DDQN}}\left(s,a,s'\right) = r(s,a) + \gamma Q_{\bar{\theta}}\left(s', \operatorname*{argmax}_{a'} Q_\theta\left(s',a'\right)\right). \quad (43)$$

C. MULTI-AGENT Q-LEARNING

The original table-based Q-learning algorithm was extended to the cooperative multi-agent setting by Claus and Boutilier in 1998 [34]. Without changing the underlying principle, it can also be applied to DDQN-based multi-agent cooperation. With the taxonomy from [35], our agents can be classified as homogeneous and non-communicating. Homogeneity is a consequence of deploying a team of identical UAVs with the same internal structure, domain knowledge, and identical action spaces. Non-communication is to be interpreted in a multi-agent system sense, i.e., that the agents can not coordinate their actions or choose what to communicate. However, as they all perceive state information that includes other UAVs' positions, in a practical sense, position information would most likely be communicated via the command and control links of the UAVs, that especially autonomous UAVs would have to maintain for regulatory purposes in any case.

The best way to describe our learning approach is by decentralized deployment or execution with centralized training. As DDQN learning requires an extensive experience database to train the neural networks on, it is reasonable to assume that the experiences made by independently acting agents can be centrally pooled throughout the training phase. After training has concluded, the control systems are individually deployed to the distributed drone agents. The rationale behind this concept is that we investigate a team of homogeneous UAVs with identical capabilities and tasks, therefore all experiences are useful for the training of all agents. In a real-world deployment of a team of quadcopter UAVs, all UAVs would be required to regularly return to a charging station, as flying time remains strongly limited by available on-board battery capacity. While being recharged, the UAVs would upload their experience data to a central server with larger memory and computation resources.

Our setting can not be characterized as fully cooperative as our agents do not share a common reward [36]. Instead, each agent has an individual but identical reward function. As the main component of the reward function is based on the jointly collected data from the IoT devices described in Section III-C, they do share a common goal, leading to the classification of our setting as a simple cooperative one.

D. NEURAL NETWORK MODEL

We use a neural network model very similar to the one presented in [20]. Fig. 3 shows the DQN structure and the map centering and global-local map processing. The map information of the environment, NFZs, obstacles, and start/landing area is stacked with the IoT device map and the map with the other UAVs' flying times and operational status. According to Section IV-A, the map is centered on the UAV's position and split into a global and local map. The global and local maps are fed through convolutional layers with ReLU activation and then flattened and concatenated with the scalar input indicating battery content or remaining flight time. After passing through fully connected layers with ReLU activation, the data reaches the last fully-connected layer of size $|\mathcal{A}|$ without activation function, directly representing the Q-values for each action given the input observation. The argmax of the Q-values, the greedy policy is given by

$$\pi(s) = \operatorname*{argmax}_{a\in\mathcal{A}} Q_\theta(s,a). \quad (44)$$

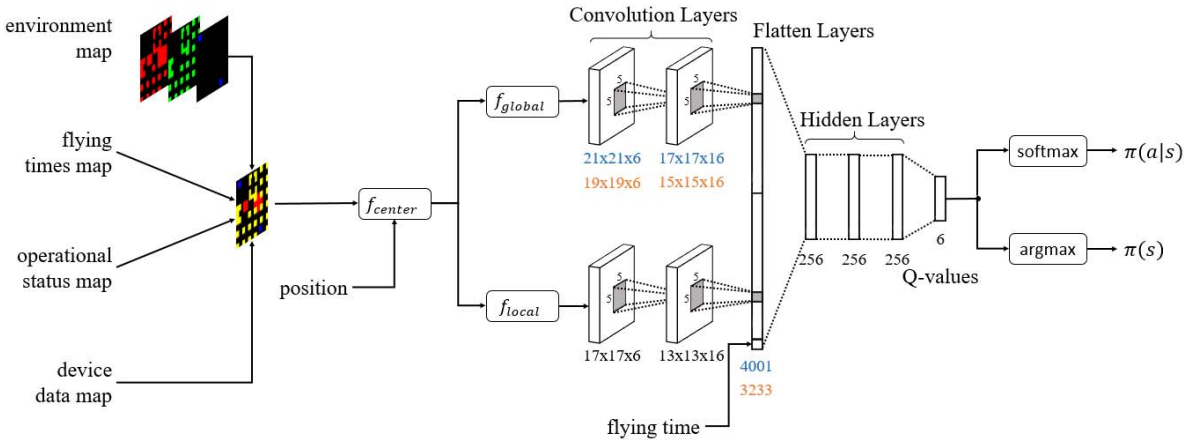


FIGURE 3. DQN architecture with map centering and global and local map processing. Layer sizes are shown in in blue for the smaller 'Manhattan32' scenario and orange for the larger 'Urban50' scenario.

TABLE 3. DDQN hyperparameters for 32 × 32 and 50 × 50 maps.

Parameter	32×32	50×50	Description
$\lvert\theta\rvert$	1,175,302	978,694	trainable parameters
$N_{\max}$	3,000,000	4,000,000	maximum training steps
l	17	17	local map scaling
g	3	5	global map scaling
$\lvert\mathcal{D}\rvert$	50,000		replay memory buffer size
m	128		minibatch size
τ	0.005		soft update factor in (41)
γ	0.95		discount factor in (43)
β	0.1		temperature parameter (45)

It is deterministic and used when evaluating the agent. During training, the soft-max policy

$$\pi(a_i|s) = \frac{e^{Q_\theta(s,a_i)/\beta}}{\sum_{\forall a_j \in \mathcal{A}} e^{Q_\theta(s,a_j)/\beta}} \tag{45}$$

is used. The temperature parameter $\beta \in \mathbb{R}$ scales the balance of exploration versus exploitation. Hyperparameters are listed in Tab. 3.

VI. SIMULATIONS

A. SIMULATION SETUP

In this work, we aim to provide an algorithm[1] that is able to generalize the learned UAV control policy over a large parameter space that defines the specific data collection scenario. That means that at the start of a new training episode, a set of scenario parameters is sampled randomly from a given range of possible values defining the mission. Then the mission starts and the agents are deployed to collect as much data as possible in the given circumstances. Specifically, we define a new mission through the following randomly varying scenario parameters:

- Number of UAVs deployed;
- Number and position of IoT sensor nodes;
- Amount of data to be collected from IoT nodes;
- Flying time available for UAVs at mission start;
- UAV start positions.

The exact value ranges from which these parameters are sampled are given in the following Sections VI-C and VI-D depending on the map. We deploy our system on two different maps. In 'Manhattan32', the UAVs fly inside 'urban canyons' through a dense city environment discretized into 32×32 cells, whereas 'Urban50' is an example of a less dense but larger 50×50 urban area. Note that we only trained a single agent on each of these maps, meaning that all results discussed in the following are a result of only two trained agents. Generalization over this large parameter space is possible in part due to the learning efficiency benefits from feeding map information centered on the agents' respective positions into the network, as we have described previously in [1].

We use the following metrics to evaluate the agents' performance on different maps and under different scenario instances.

- *Successful landing*: records whether all agents have landed in time at the end of an episode;
- *Collection ratio*: the ratio of total collected data at the end of the mission to the total device data that was available at the beginning of the mission;
- *Collection ratio and landed*: the product of *successful landing* and *collection ratio* per episode.

Evaluation is challenging as we train a single control policy to generalize over a large scenario parameter space. During training, we evaluate the agents' training progress in a randomly selected scenario every five episodes and form an average over multiple evaluations. A single evaluation could be tainted by unusually easy conditions, e.g., when all devices are placed very close to each other by chance. Therefore, only an average over multiple evaluations can be indicative of the agents' learning progress. As it is computationally

1. The Python code for this work is available under https://github.com/hbayerlein/uav_data_harvesting.

infeasible to evaluate the trained system on all possible scenario variations, we perform Monte Carlo analysis on a large number of randomly selected scenario parameter combinations.

Irrespective of the map, the grid cell size is $c = 10$m and the UAVs fly at a constant altitude of $h = 10$m over city streets. The UAVs are not allowed to fly over tall buildings, enter NFZs, or leave the respective grid worlds. Each mission time slot $t \in [0, T]$ contains $\lambda = 4$ scheduled communication time slots $n \in [0, N]$. Propagation parameters (see Section II-B) are chosen in-line with [37] according to the urban micro scenario with $\alpha_{\text{LoS}} = 2.27$, $\alpha_{\text{NLoS}} = 3.64$, $\sigma^2_{\text{LoS}} = 2$ and $\sigma^2_{\text{NLoS}} = 5$.

Due to the drones flying below or slightly above building height, the wireless channel is characterized by strong LoS/NLoS dependency and shadowing. The shadowing maps used for simulation of the environment were computed using ray tracing from and to the center points of cells based on a variation of Bresenham's line algorithm. Transmission and noise powers are normalized by defining a cell-edge SNR for each map, which describes the SNR between one drone on ground level at the center of the map and an unobstructed IoT device maximally far apart at one of the grid corners. The agents have absolutely no prior knowledge of the shadowing maps or wireless channel characteristics.

B. TRAINING WITH MAP-BASED VS. SCALAR INPUTS

In this section, we show that our map-based approach has a good complexity-performance trade-off in comparison to classical scalar input neural network approaches from the literature despite the added complexity through map-processing. To illustrate that it is in fact imperative for training success to feed map information instead of concatenated scalar values as state input to the agent, we extend our previous analysis from [1] and [20] by comparing our proposed centered global-local map approach to agents trained only on scalar inputs. This is not an entirely fair comparison as the location of NFZs, buildings, and start/landing zones can not be efficiently represented by scalar inputs and must be therefore learned by the scalar agents through trial and error. However, the comparison illustrates the need for state space representations that are different from the traditional scalar inputs and confirms that scalar agents are not able to solve the multi-UAV path planning problem over the large scenario parameter space presented. Conversely, the alternative comparison of map-based and scalar agents trained on a *single* data harvesting scenario would not yield meaningful results as our method is specifically designed to generalize over a large variety of scenarios and would require tweaking in exploration behavior and reward balance to find the optimal solution to a single scenario. Note that most of the previous work discussed in Section I-A is precisely focused on finding optimal DRL solutions to single scenario instances.

The observation space of the agents trained with concatenated scalar inputs is described by

$$\left.\begin{aligned} O_{\text{scalar}} = \underbrace{\mathbb{N}^2}_{\substack{\text{Ego}\\ \text{Position}}} \times \underbrace{\mathbb{N}}_{\substack{\text{Ego Flying}\\ \text{Time}}} \end{aligned}\right\} \text{Ego agent}$$

$$\left.\begin{aligned} \times \underbrace{\mathbb{N}^{I\times 2}}_{\substack{\text{UAV}\\ \text{Positions}}} \times \underbrace{\mathbb{N}^{I}}_{\substack{\text{Flying}\\ \text{Times}}} \times \underbrace{\mathbb{B}^{I}}_{\substack{\text{Operational}\\ \text{Status}}} \end{aligned}\right\} \text{Other agents}$$

$$\left.\begin{aligned} \times \underbrace{\mathbb{N}^{K\times 2}}_{\substack{\text{Device}\\ \text{Positions}}} \times \underbrace{\mathbb{R}^{K}}_{\substack{\text{Device}\\ \text{Data}}} \end{aligned}\right\} \text{Devices} \tag{46}$$

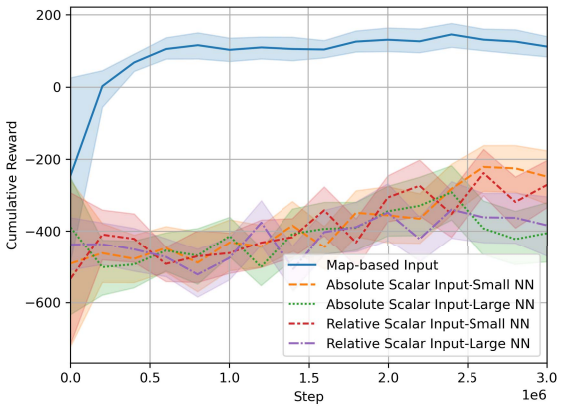
forming the input of the neural network as concatenated scalar values. Since the number of agents and devices is variable, the scalar input size is fixed to the maximum number of agents and devices. The agent and device positions are either represented as *absolute* values in the grid coordinate frame or *relative* as distances from the ego agent. The neural network is either *small*, containing the same number of hidden layers as in Fig. 3, or *large*, for which the number and size of hidden layers is adapted such that the network has as many trainable parameters as the map-based 32×32 agent in Tab. 3.

Fig. 4 shows the cumulative reward and the collection ratio with successful landing metric over training time on the 'Manhattan32' map for the five different network architectures. It is clear that the scalar agents are not able to effectively adapt to the changing scenario conditions. The *small* neural network agents seem to have a slight edge over the *large* agents, but representing the positions as *absolute* or *relative* does not influence the results.

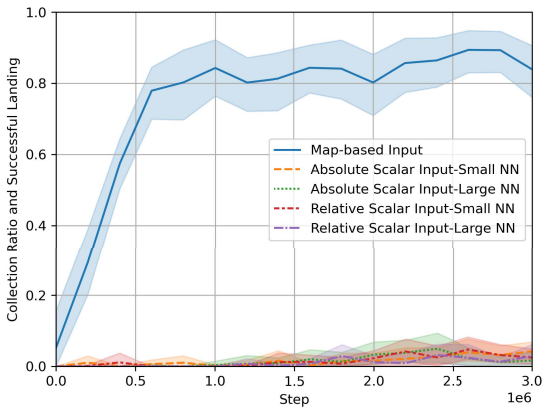
Referring further to Fig. 4, the map-based agent converges to final performance metric levels after the first 20% of the training steps. However we observed that additional training is needed after that to optimize the trajectories in a more subtle way for flight time efficiency and multi-UAV coordination. The overall training time for the full 3 million training steps was around 40 hours on a 2017 Nvidia Titan Xp GPU.

C. "MANHATTAN32" SCENARIO

The scenario, as shown in Fig. 5 is defined by a Manhattan-like city structure containing mostly regularly distributed city blocks with streets in between, as well as two NFZ districts and an open space in the upper left corner, divided into $M = 32$ cells in each grid direction. This is double the size of the otherwise similarly designed single UAV scenario in [1]. We are able to solve the larger scenario without increasing network size, thanks to the global-local map approach. The value ranges from which the randomized scenario parameters are chosen as follows: number of deployed UAVs $I \in \{1, 2, 3\}$, number of IoT sensors $K \in [3, 10]$, data volume to be collected $D_{k,init} \in [5.0, 20.0]$ data units



(a) Cumulative reward per episode



(b) Data collection ratio with successful landing per episode

FIGURE 4. Training process comparison between *map-based* DRL path planning and *scalar* input DRL path planning. Scalar inputs to the neural networks (NNs) are either encoded as *absolute* coordinate values or *relative* distances from the respective agent. We compare two different scalar input network architectures with *large* and *small* numbers of trainable parameters. The average and 99% quantiles are shown with metrics per training episode grouped in bins of 2e5 step width. Note that the metrics are plotted over training steps as training episode length is variable.

TABLE 4. Performance metrics averaged over 1000 random scenario Monte Carlo iterations.

Metric	Manhattan32	Urban50
Successful Landing	99.4%	98.8%
Collection Ratio	88.0%	82.1%
Collection Ratio and Landed	87.5%	81.1%

per device, maximum flying time $b_0 \in [50, 150]$ steps, and 18 possible starting positions. The IoT device positions are randomized throughout the unoccupied map space.

The performance on both maps is evaluated using Monte Carlo simulations on their respective full range of scenario parameters with overall average performance metrics shown in Table 4. Both agents show a similarly high successful landing performance. It is expected that the collection ratio cannot reach 100% in some scenario instances depending on the randomly assigned maximum flying time, number of deployed UAVs, and IoT device parameters.

In Fig. 5, three scenario instances chosen from the random Monte Carlo evaluation for number of deployed UAVs $I \in \{1, 2, 3\}$ for 5(a) through 5(c) illustrate how the path planning adapts to the increasing number of deployed UAVs. All other scenario parameters are kept fixed. It is a fairly complicated scenario with a large number of IoT devices spread out over the whole map, including the brown and purple device inside an NFZ. The agents have no access to the shadowing map and have to deduce shadowing effects from building and device positions.

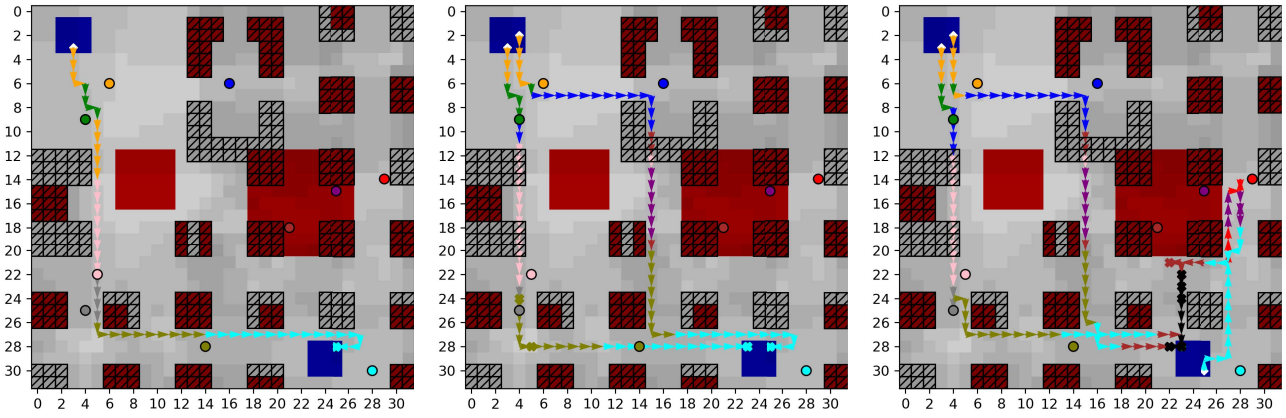
In Fig. 5(a), only one UAV starting in the upper left corner is deployed. Due to its flight time constraint, the agent ignores the blue, red, purple, and brown IoT devices while collecting all data from the other devices on an efficient trajectory to the landing zone in the lower right corner. When a second UAV is deployed in Fig. 5(b), the data collection ratio increases to 76.5%. While the first UAV's behavior is almost unchanged compared to the single UAV deployment, the second UAV flies to the landing zone in the lower right corner via an alternative trajectory collecting data from the devices the first UAV ignores. With the number of deployed UAVs increased to three (two starting from the upper left and one from the lower right zone) in Fig. 5(c), all data can be collected. The second UAV modifies its behavior slightly, accounting for the fact that the third UAV can collect the cyan device's data now. The three UAVs divide the data harvesting task fairly among themselves, leading to full data collection with in-time landing on efficient trajectories while avoiding the NFZs.

D. "URBAN50" SCENARIO

Fig. 6 shows three example trajectories for UAV counts of $I \in \{1, 2, 3\}$ for 6(a) through 6(c) in the large 50×50 urban map. The scenario is defined by an urban structure containing irregularly shaped large buildings, city blocks and an NFZ, with the start/landing zone surrounding a building in the center, divided into $M = 50$ cells in each grid direction. The map has an order of magnitude more cells than the scenarios in [1]. The ranges for randomized scenario parameters are chosen as follows: number of deployed UAVs $I \in \{1, 2, 3\}$, number of IoT sensors $K \in [5, 10]$, data volume to be collected $D_{k,init} \in [5.0, 20.0]$ data units, maximum flying time $b_0 \in [100, 200]$ steps, and 40 possible starting positions. The IoT device positions are randomized throughout the unoccupied map space.

Fig. 6(a) shows a single agent trying to collect as much data as possible during the allocated maximum flying time. The agent focuses on collecting the data from the relatively easily reachable device clusters on the right and lower half before safely landing. With a second UAV assigned to the mission as shown in Fig. 6(b), one UAV services the devices on the lower left of the map, while the other one collects data from the devices on the lower right, ignoring the more isolated blue and orange device in the top half of the map. A third UAV makes it possible to divide the map into three sectors and collect all IoT device data, as shown in Fig. 6(c).

This map's primary purpose is to showcase the significant advantages in terms of training time efficiency and the

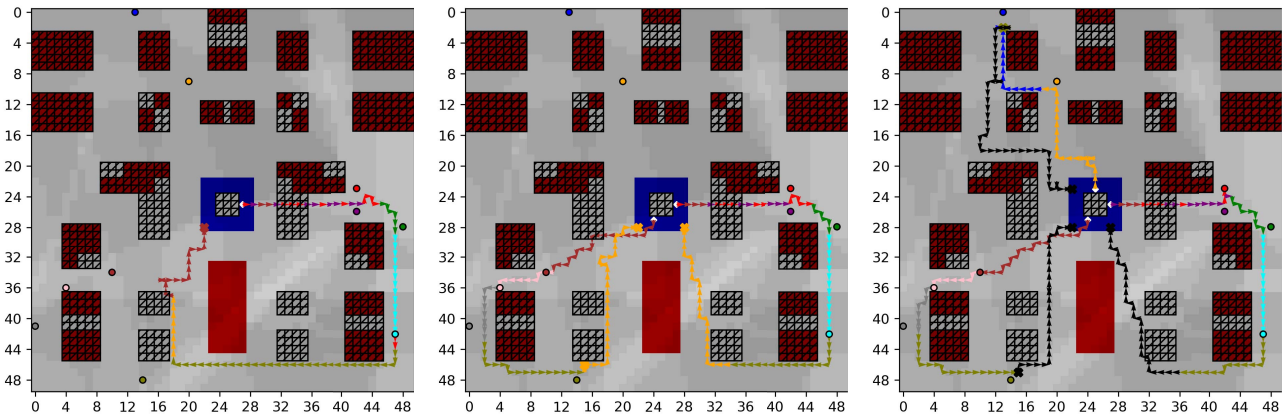


(a) $I = 1$ agent, data collection ratio 56.2%

(b) $I = 2$ agents, data collection ratio 76.5%

(c) $I = 3$ agents, data collection ratio 100%

FIGURE 5. Example trajectories for 'Manhattan32' map with $K = 10$ IoT devices, all with $D_{k,init} = 15$ data units to be picked up and a maximum flying time of $b_0 = 60$ steps. The color of the UAV movement arrows shows with which device the drone is communicating at the time (see legend in Table 1).



(a) $I = 1$ agent, data collection ratio 41.8%

(b) $I = 2$ agents, data collection ratio 80.2%

(c) $I = 3$ agents, data collection ratio 100.0%

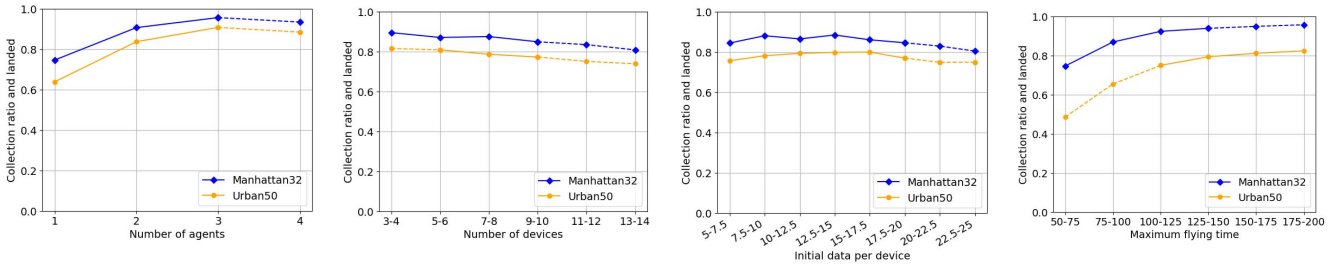
FIGURE 6. Example trajectories for 'Urban50' map with $K = 10$ IoT devices, all with $D_{k,init} = 15$ data units to be picked up and a maximum flying time of $b_0 = 100$ steps for all UAVs (legend in Table 1).

required network size from the global-local map approach. Thanks to a higher global map scaling or compression factor g (see Table 3), the number of trainable parameters of the network employed in this scenario is even smaller compared to the network used for 'Manhattan32'. A network without a map scaling approach would need to be of size 34,061,446, hence a size that is infeasible to train using reasonable resources.

E. INFLUENCE OF SCENARIO PARAMETERS ON PERFORMANCE AND SYSTEM-LEVEL BENEFITS

An advantage of our approach to learn a generalized path planning policy over various scenario parameters is the possibility to analyze how performance indicators change over a variable parameter space. This makes it possible for an operator to decide on system-level trade-offs, e.g., how many drones to deploy vs. collected data volume. An excellent example that we found for the 'Manhattan32' map was that deploying multiple coordinating drones can trade-off the cost of extra equipment (i.e., the extra drones) for substantially reduced mission time. For instance, it takes twice the flying time ($b_0 = 150$) for a single UAV to complete the data collection mission that two coordinating UAVs will require ($b_0 = 75$) to conclude successfully. Specifically, that means that for both scenarios the average data collection ratio with in-time successful landing stays at the same performance level of around 88%.

We first analyze the performance of the agent in Fig. 7 within the training range of the scenario parameters (solid lines), then extend the analysis to out-of-distribution scenarios (dashed lines) in the last paragraph of this section. Fig. 7 shows the influence of single scenario parameters on the average data collection ratio with successful landing of all agents. As already evident from the example trajectories shown previously, Fig. 7(a) indicates the increase in collection performance when more UAVs are deployed. At the same time, more UAVs lead to increased collision avoidance requirements, as we observed through more safety



(a) Number of agents $I \in \{1, 2, 3, 4\}$

(b) Number of devices $K \in [3, 14]$ sorted into bins of two.

(c) Data to be collected from devices $D_{k,init} \in [5, 25]$ sorted into eight bins.

(d) Maximum flying time sorted into six bins in $b_0 \in [50, 200]$

FIGURE 7. **Influence of specific scenario parameters on the data collection ratio with successful landing of all agents. Each data point is an average of 500 Monte Carlo iterations over the respective parameter spaces for the 'Manhattan32' and 'Urban50' map. The parameters within the training range are rendered in solid lines and the out-of-distribution parameter evaluation in dashed lines.**

controller activations in the early training phases. As IoT devices are positioned randomly throughout the unoccupied map space, an increase in devices leads to more complex trajectory requirements and a drop in performance, as depicted in Fig. 7(b).

Fig. 7(c) shows the influence of increasing initial data volume per device on the overall collection performance. It appears that higher initial data volumes per device are beneficial roughly up to the point of $D_{k,init} \in [10, 12.5]$ data units, after which flying time constraints force the UAVs to abandon some of the data, and the collection ratio shows a slightly negative trend. An increase in available flying time is clearly beneficial to the collection performance, as indicated in Fig. 7(d). However, the effect becomes smaller when most of the data is collected, and the UAVs start to prioritize minimizing overall flight time and safe landing over the collection of the last bits of data.

It is further shown in Fig. 7 how the agents react to scenario parameters which were not encountered during training. The corresponding values are highlighted with dashed lines. It can be seen that the performance of the agents follows the same trend as in the rest of the data, when increasing the number of devices (Fig. 7(b)) or initial data per device (Fig. 7(c)) out of the trained region. When increasing the maximum flying time (Fig. 7(d)) for the Manhattan32 agents, or decreasing it for Urban50 agents, the collection ratio with successful landing performance, increases or decreases accordingly. Incrementing the number of agents to four (Fig. 7(a)) reduces the performance slightly. The reason is the decrease in landing performance. However, this is to be expected since the probability of all agents landing decreases with the number of agents. Since the collection ratio is nearly saturated for the scenarios with three agents, the drop in overall landing performance decreases the collection ratio and landed performance. In general, it is evident that the proposed approach cannot only generalize over the whole range of scenario parameters encountered during training but can also extrapolate successfully to out-of-distribution parameters.

VII. CONCLUSION

We have introduced a multi-agent reinforcement learning approach that allows us to control a team of cooperative UAVs on a data harvesting mission in a large variety of scenarios without the need for recomputation or retraining when the scenario changes. By leveraging a DDQN with combined experience replay and convolutionally processing dual global-local map information centered on the agents' respective positions, the UAVs are able to find efficient trajectories that balance data collection with safety and navigation constraints without any prior knowledge of the challenging wireless channel characteristics in the urban environments. We have also presented a detailed description of the underlying path planning problem and its translation to a decentralized partially observable Markov decision process. In future work, we will extend the UAVs' action space to altitude and continuous control, requiring an RL algorithm different from Q-learning with a continuous action space and adding height information to the agents' observations space. Moreover, we will investigate if attention-based mechanisms can be used for map processing, assessing their viability with respect to performance and computational requirements in this context. Further improvements in learning efficiency could be achieved when combining our approach with multi-task reinforcement learning or transfer learning [26], a step that would bring RL-based autonomous UAV control in the real-world even closer to realization.

REFERENCES

[1] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "UAV path planning for wireless data harvesting: A deep reinforcement learning approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2020, pp. 1–6.

[2] Y. Zeng, Q. Wu, and R. Zhang, "Accessing from the sky: A tutorial on UAV communications for 5G and beyond," *Proc. IEEE*, vol. 107, no. 12, pp. 2327–2375, Dec. 2019.

[3] K. Namuduri, "Flying cell towers to the rescue," *IEEE Spectr.*, vol. 54, no. 9, pp. 38–43, Sep. 2017.

[4] M. Minevich, *How Japan Is Tackling the National & Global Infrastructure Crisis & Pioneering Social Impact—[News]*, Forbes, Jersey City, NJ, USA, Apr. 2020.

[5] J. Cui, Z. Ding, Y. Deng, and A. Nallanathan, "Model-free based automated trajectory optimization for UAVs toward data transmission," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[6] Y. Pan, Y. Yang, and W. Li, "A deep learning trained by genetic algorithm to improve the efficiency of path planning for data collection with multi-UAV," *IEEE Access*, vol. 9, pp. 7994–8005, 2021.

[7] J. Tang *et al.*, "Minimum throughput maximization for multi-UAV enabled WPCN: A deep reinforcement learning method," *IEEE Access*, vol. 8, pp. 9124–9132, 2020.

[8] Y. Zhang, Z. Mou, F. Gao, L. Xing, J. Jiang, and Z. Han, "Hierarchical deep reinforcement learning for backscattering data collection with multiple UAVs," *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3786–3800, Mar. 2021.

[9] C. H. Liu, Z. Dai, Y. Zhao, J. Crowcroft, D. O. Wu, and K. Leung, "Distributed and energy-efficient mobile crowdsensing with charging stations by deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 20, no. 1, pp. 130–146, Jan. 2021.

[10] M. Yi, X. Wang, J. Liu, Y. Zhang, and B. Bai, "Deep reinforcement learning for fresh data collection in UAV-assisted IoT networks," in *Proc. IEEE Conf. Comput. Commun. Workshops (IEEE INFOCOM WKSHPS)*, 2020, pp. 716–721.

[11] F. Wu, H. Zhang, J. Wu, L. Song, Z. Han, and H. V. Poor, "UAV-to-device underlay communications: Age of information minimization by multi-agent deep reinforcement learning," *IEEE Trans. Commun.*, early access, Mar. 10, 2021, doi: 10.1109/TCOMM.2021.3065135.

[12] J. Hu, H. Zhang, L. Song, R. Schober, and H. V. Poor, "Cooperative Internet of UAVs: Distributed trajectory design by multi-agent deep reinforcement learning," *IEEE Trans. Commun.*, vol. 68, no. 11, pp. 6807–6821, Nov. 2020.

[13] M. A. Abd-Elmagid, A. Ferdowsi, H. S. Dhillon, and W. Saad, "Deep reinforcement learning for minimizing age-of-information in UAV-assisted networks," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, 2019, pp. 1–6.

[14] Y. Hu, M. Chen, W. Saad, H. V. Poor, and S. Cui, "Distributed multi-agent meta learning for trajectory design in wireless drone networks," 2020. [Online]. Available: arXiv:2012.03158.

[15] X. Li, H. Yao, J. Wang, S. Wu, C. Jiang, and Y. Qian, "Rechargeable multi-UAV aided seamless coverage for QoS-guaranteed IoT networks," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10902–10914, Dec. 2019.

[16] X. Liu, Y. Liu, and Y. Chen, "Reinforcement learning in multiple-UAV networks: Deployment and movement design," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8036–8049, Aug. 2019.

[17] R. Shakeri *et al.*, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3340–3385, 4th Quart., 2019.

[18] W. Saad, M. Bennis, M. Mozaffari, and X. Lin, *Wireless Communications and Networking for Unmanned Aerial Vehicles*. Cambridge, U.K.: Cambridge Univ. Press, 2020.

[19] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV coverage path planning under varying power constraints using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, 2020, pp. 1444–1449.

[20] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV path planning using global and local map information with deep reinforcement learning," 2020. [Online]. Available: arXiv:2010.06917.

[21] Z. Ullah, F. Al-Turjman, and L. Mostarda, "Cognition in UAV-aided 5G and beyond communications: A survey," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 3, pp. 872–891, Sep. 2020.

[22] M.-A. Lahmeri, M. A. Kishk, and M.-S. Alouini, "Artificial intelligence for UAV-enabled wireless networks: A survey," *IEEE Open J. Commun. Soc.*, vol. 2, pp. 1015–1040, 2020.

[23] H. Bayerlein, P. De Kerret, and D. Gesbert, "Trajectory optimization for autonomous flying base station via reinforcement learning," in *Proc. IEEE 19th Int. Workshop Signal Process. Adv. Wireless Commun. (SPAWC)*, 2018, pp. 1–5.

[24] F. Venturini *et al.*, "Distributed reinforcement learning for flexible UAV swarm control with transfer learning capabilities," in *Proc. 6th ACM Workshop Micro Aerial Veh. Netw. Syst. Appl.*, 2020, pp. 1–6.

[25] C. H. Liu, X. Ma, X. Gao, and J. Tang, "Distributed energy-efficient multi-UAV navigation for long-term communication coverage by deep reinforcement learning," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1274–1285, 2020.

[26] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," 2019. [Online]. Available: arXiv:1904.12901.

[27] Z. Liu, R. Sengupta, and A. Kurzhanskiy, "A power consumption model for multi-rotor small unmanned aircraft systems," in *Proc. Int. Conf. Unmanned Aircraft Syst. (ICUAS)*, 2017, pp. 310–315.

[28] A. I. Hentati and L. C. Fourati, "Comprehensive survey of uavs communication networks," *Comput. Stand. Interfaces*, vol. 72, Oct.2020, Art. no. 103451.

[29] F. A. Oliehoek and C. Amato, *A Concise Introduction to Decentralized POMDPs*. Heidelberg, Germany: Springer, 2016.

[30] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

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

[32] S. Zhang and R. S. Sutton, "A deeper look at experience replay," 2017. [Online]. Available: arXiv:1712.01275.

[33] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double *Q*-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[34] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," in *Proc. AAAI/IAAI*, 1998, p. 2.

[35] P. Stone and M. Veloso, "Multiagent systems: A survey from a machine learning perspective," *Auton. Robots*, vol. 8, no. 3, pp. 345–383, 2000.

[36] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," 2019. [Online]. Available: arXiv:1911.10635.

[37] "Study on channel model for frequencies from 0.5 to 100 GHz version 14.0.0 Release 14," 3GPP, Sophia Antipolis, France, Rep. TR 38.901, May 2017.

HARALD BAYERLEIN (Student Member, IEEE) received the M.Sc. degree in electrical engineering and information technology from the Technical University of Munich, Germany, in 2018. He is currently pursuing the Ph.D. degree with EURECOM and Sorbonne University, France. He has also studied electrical engineering with the University of Tokyo, University College London, and the University of Erlangen–Nuremberg. His current research interests extend to machine learning in applications of communications engineering and autonomous systems, specifically UAV-aided networks and path planning.

MIRCO THEILE (Student Member, IEEE) received the M.Sc. degree in electrical engineering and information technology from Technical University of Munich, Germany, in 2018, where he is currently pursuing the Ph.D. degree. During his M.Sc., he also studied electrical engineering with the Royal Institute of Technology (KTH), Stockholm, Sweden, and was a Visiting Scholar with the University of Illinois at Urbana–Champaign, USA. His current research interests extend to reinforcement learning in applications of robotics, focussing mainly on UAV path planning with single and multiagent systems.

MARCO CACCAMO (Fellow, IEEE) studied Computer Engineering at University of Pisa, Italy. He received the degree in computer engineering in July 1997, and the Ph.D. degree in computer engineering from Scuola Superiore Sant'Anna, Italy, in 2002. Shortly after graduation, he joined the University of Illinois at Urbana–Champaign as an Assistant Professor of Computer Science and was promoted to a Full Professor in 2014. Since 2018, he has been appointed to the Chair of Cyber-Physical Systems in Production Engineering with TUM. He received visiting professorships with ETH Zurich, and TUM, Munich, as a TÜV Süd Stiftung Visiting Professor and an August-Wilhelm Scheer Guest Professor. He was awarded an NSF CAREER Award in 2003. He is a recipient of the Alexander von Humboldt Professorship. He has chaired Real-Time Systems Symposium and Real-Time and Embedded Technology and Applications Symposium, the two IEEE flagship conferences on Real-Time Systems. He also served as the General Chair of Cyber Physical Systems Week.

DAVID GESBERT (Fellow, IEEE) received the Ph.D. degree from Ecole Nationale Superieure des Telecommunications, France, in 1997. He is a Professor and the Head of the Communication Systems Department, EURECOM. From 1997 to 1999, he has been with the Information Systems Laboratory, Stanford University. He was then a Founding Engineer of Iospan Wireless, Inc., a Stanford spin off pioneering MIMO-OFDM (currently, Intel). Before joining EURECOM in 2004, he has been with the Department of Informatics, University of Oslo as an Adjunct Professor. He has published about 340 papers and 25 patents, some of them winning 2019 ICC Best Paper Award, the 2015 IEEE Best Tutorial Paper Award (Communications Society), the 2012 SPS Signal Processing Magazine Best Paper Award, the 2004 IEEE Best Tutorial Paper Award (Communications Society), the 2005 Young Author Best Paper Award for Signal Processing Society journals, and paper awards at conferences 2011 IEEE SPAWC and 2004 ACM MSWiM. He has been a Technical Program Co-Chair for ICC2017. He was named a Thomson-Reuters Highly Cited Researchers in Computer Science. In 2015, he was awarded the ERC Advanced Grant "PERFUME" on the topic of smart device Communications in future wireless networks. Since early 2019, he heads the Huawei-funded Chair on Adwanced Wireless Systems Towards 6G Networks. He sits on the Advisory Board of HUAWEI European Research Institute. In 2020, he was awarded funding by the French Interdisciplinary Institute on Artificial Intelligence for a Chair in the area of AI for the future IoT. He is a Board member for the OpenAirInterface Software Alliance.