# UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

Mirco Theile, Harald Bayerlein, Richard Nai, David Gesbert, Marco Caccamo

# UAV Path Planning using Global and Local Map Information with Deep Reinforcement Learning

Mirco Theile[1], Harald Bayerlein[2], Richard Nai[1], David Gesbert[2], and Marco Caccamo[1]

***Abstract*— Path planning methods for autonomous unmanned aerial vehicles (UAVs) are typically designed for one specific type of mission. This work presents a method for autonomous UAV path planning based on deep reinforcement learning (DRL) that can be applied to a wide range of mission scenarios. Specifically, we compare coverage path planning (CPP), where the UAV's goal is to survey an area of interest to data harvesting (DH), where the UAV collects data from distributed Internet of Things (IoT) sensor devices. By exploiting structured map information of the environment, we train double deep Q-networks (DDQNs) with identical architectures on both distinctly different mission scenarios to make movement decisions that balance the respective mission goal with navigation constraints. By introducing a novel approach exploiting a compressed global map of the environment combined with a cropped but uncompressed local map showing the vicinity of the UAV agent, we demonstrate that the proposed method can efficiently scale to large environments. We also extend previous results for generalizing control policies that require no retraining when scenario parameters change and offer a detailed analysis of crucial map processing parameters' effects on path planning performance.**

## I. Introduction

Autonomous unmanned aerial vehicles (UAVs) are envisioned for a multitude of applications that all require efficient and safe path planning methods, which necessitate the combination of a mission goal with navigation constraints, e.g., flying time and obstacle avoidance. Examples for these applications are area coverage path planning (CPP) [1], and data harvesting (DH) from Internet of Things (IoT) sensor nodes [2]. As its name suggests, covering all points inside an area of interest with CPP is related to conventional path planning, where the goal is to find a path between start and goal positions. In general, CPP aims to cover as much of the target area as possible within given energy or path-length constraints while avoiding obstacles or no-fly zones.

In the DH scenario, the UAV's goal is to collect data from IoT devices distributed in an urban environment, which implies challenging radio channel conditions through alternating line-of-sight (LoS) and non-line-of-sight (NLoS) links between UAV and IoT devices through building obstruction. DH and CPP are very similar when described as an RL problem since the path planning problem's constraints are mostly identical, and only the goal function changes. In previous work, we have looked at CPP [1] and DH [2] separately. We show that both problems can be solved using the same deep reinforcement learning (DRL) approach based on feeding spatial map information directly to the DRL agent via convolutional network layers. This work's focus lies in proposing improvements to existing DRL approaches to generalized, large-scale UAV path planning problems with CPP and DH as examples.

Using maps as a direct input becomes problematic for large map sizes, as the network's size, trainable parameters, and training time increase equivalently. We introduce a global-local map scheme to address the scalability problems of the standard map-based input. In path planning, the intuition is that distant features lead to general direction decisions, while close features lead to immediate actions such as collision avoidance. Thus, the level of detail passed to the agent for distant objects can be less than for close objects. With the global map, a compressed version of the complete environment map centered on the agent's position, general information of all objects on the map is provided to the agent. In contrast, the local map, uncompressed but cropped to show only the UAV agent's immediate surroundings, provides detailed local information.

While numerous path planning algorithms for both problems exist, DRL offers the possibility to solve both distinctly different problems with the same approach. For each problem, DRL agents can learn control policies that generalize over a large scenario parameter space requiring no expensive retraining or recomputation when the scenario changes. However, previous work usually only focuses on finding optimal paths for one single scenario at a time. The DRL paradigm is popular in this context because of its flexibility regarding prior knowledge and assumptions about the environment, the computational efficiency of DRL inference, as well as the complexity of autonomous UAV control tasks, which are usually non-convex optimization problems and proven to be NP-hard in many instances [3], [4]. A general summary of issues in using UAVs as part of communication networks, including IoT data harvesting, can be found in [3]. A survey of various applications for UAV systems from a cyber-physical perspective is offered in [4]. Cabreira *et al.* [5] provide a survey of UAV coverage path planning.

Previous works in UAV path planning have already made use of convolutional map processing for DRL agents. In the drone patrolling problem presented in [6], a local relevance map of the patrolling area showing the agent's vicinity cropped to a fixed size is fed into a DDQN agent. No information of the physical environment is included, and there

[1]Mirco Theile, Richard Nai, and Marco Caccamo are with the TUM School of Engineering and Design, Technical University of Munich, Germany {mirco.theile, richard.nai, mcaccamo}@tum.de

[2]Harald Bayerlein and David Gesbert are with the Communication Systems Department, EURECOM, Sophia Antipolis, France {harald.bayerlein, david.gesbert}@eurecom.fr

is no consideration for navigation constraints like obstacle avoidance or flying time. In [7], fixed-wing UAVs are tasked with monitoring a wildfire propagating stochastically over time. Control decisions are based on either direct observations or belief maps fed into the DRL agents. The focus here is the inherent uncertainty of the problem, not balancing a mission goal and navigation constraints in large complex environments. Wildfire surveillance is also the mission of the quadcopter UAVs in [8], which is set in a similar scenario without navigation constraints and makes use of uncertainty maps to guide path planning. Their approach is based on an extended Kalman filter and not the reinforcement learning (RL) paradigm. To monitor another natural disaster situation, Baldazo *et al.* [9] present a multi-agent DRL method for flood surveillance using the UAVs' local observations of the inundation map to make control decisions. All mentioned approaches focus on solving a single class of UAV missions in simple physical environments and do not consider combining local and global map information.

Missions, where UAVs provide communication services to ground users or devices, include the work in [10] set in a complex urban environment where the UAV path planning is based on exploiting map information with a method combining dynamic and sequential convex programming. In [11], data is collected simultaneously with ground and aerial vehicles on a small map with obstacles. Due to the small map size, the full global map information can be fed into the DRL agents. Another scenario is investigated in [12], where a cellular-connected UAV has to navigate from a start to an end position maintaining connectivity with a ground network exploiting a radio map. The approach includes global radio map compression to reduce computational complexity but is not based on RL and includes no higher precision local map or hard navigation constraints. To the best of our knowledge, no dual global-local map processing method applicable to multiple mission types for autonomous UAVs has been suggested previously.

The main contributions of this paper are the following:

- Establishing the presented DRL approach as a general method for UAV path planning by demonstrating its applicability to two distinctly different mission scenarios: coverage path planning and path planning for wireless data harvesting;
- Introducing a novel approach[1] to exploit global-local map information that allows DRL for path planning to scale to large, realistic scenario environments efficiently, with an order of magnitude more grid cells compared to earlier works [1], [2];
- Overcoming the limitation of fixed target zones in previous DRL CPP approaches [1] by extending control policy generalization over scenario parameters to randomly generated target zones;
- Analyzing and discussing the effects of key map processing parameters on the path learning performance.

[1]https://www.github.com/theilem/uavSim.git

## II. Problem Formulation

In the following, we show that a universal problem description of coverage path planning and path planning for data harvesting can be established through separation into two parts: the environment and the target.

### A. Environment and UAV Model

We consider a square grid world of size $M \times M \in \mathbb{N}^2$ with cell size $c$, where $\mathbb{N}$ is the set of natural numbers. The environment contains designated start/landing positions, regulatory no-fly zones (NFZs), and obstacles. The map can be described through a tensor $\mathbf{M} \in \mathbb{B}^{M\times M\times 3}$, where $\mathbb{B} = \{0, 1\}$ and with the start/landing zones in map-layer 1, the union of NFZs and obstacles in map-layer 2, and the obstacles alone in map-layer 3.

The UAV moves through this environment at a constant altitude $h$ occupying one cell of the environment. Its position can thus be defined through $\mathbf{p}(t) \in \mathbb{N}^2$. The movement of the UAV is constrained through collision avoidance with obstacles and not entering NFZs. Additionally, the UAV must start and end its mission in any cell belonging to the start and landing zones while staying within its maximum flying time determined by its initial battery level. The battery level of the UAV $b(t)$ is set to $b_0 \in \mathbb{N}$ at time $t = 0$ and is decremented by 1 per action step.

### B. Target and Mission Definitions

*1) Coverage Path Planning:* In coverage path planning, the mission is to cover a designated target area by flying above or near it, such that it is in the field of view of a camera-like sensor mounted underneath the UAV. The target area can be described through $\mathbf{T}(t) \in \mathbb{B}^{M\times M}$, in which each element describes whether a cell has to be covered or not. The current field of view of the camera can be described with $\mathbf{V}(t) \in \mathbb{B}^{M\times M}$ indicating for each cell whether it is in the current field of view or not. In this work, the field of view is a square of $5 \times 5$ surrounding the current UAV position. Additionally, buildings can block line-of-sight, which is also incorporated in calculating $\mathbf{V}(t)$. This prohibits the UAV from *seeing* around the corner.

Consequently, the target area evolves according to

$$\mathbf{T}(t+1) = \mathbf{T}(t) \wedge \neg\mathbf{V}(t), \tag{1}$$

in which $\wedge$ and $\neg$ are the cell-wise logical *and* and *negation* operators, respectively. In our mission definition, obstacle cells in the environment cannot be a coverage target, while start and landing zones and no-fly-zones can be. The goal is to cover as much of the target area as possible within the maximum flying time constraint.

*2) Data Harvesting:* Conversely, the mission in path planning for wireless data harvesting is to collect data from $K \in \mathbb{N}$ stationary IoT devices spread throughout the environment at ground-level, with the position of device $k \in [1, K]$ given through $\mathbf{u}_k \in \mathbb{N}^2$. Each device has an amount of data $D_k(t) \in \mathbb{R}$ to be collected by the UAV. The data throughput $C_k(t)$ between the selected device $k$ and the UAV is based on the standard log-distance path loss model with Gaussian

shadow fading and whether they can establish a line-of-sight connection or are obstructed by obstacles. The UAV is communicating with one device at a time and selects the device with remaining data and the highest possible data rate. A detailed description of the link performance and multiple access protocol can be found in [2]. The data at each device evolves according to

$$D_k(t+1) = D_k(t) - C_k(t) \tag{2}$$

Devices can be located in every cell except for the starting and landing zones or inside obstacles. The goal of the data harvesting problem is to collect as much of the devices' data as possible within the maximum flying time.

*3) Unifying Map-Layer Description:* Both problems can be described through a single target map-layer $\mathbf{D}(t) \in \mathbb{R}^{M\times M}$. In CPP, the target map-layer is given through $\mathbf{T}(t)$ evolving according to (1). In DH, the target map-layer shows the amount of available data in each cell that one of the devices is occupying, i.e. the cell at position $\mathbf{u}_k$ has value $D_k(t)$ and is evolving according to (2). If a cell does not contain a device or the device data has been collected fully, the cell's value is 0. Since the two problems can be described with similar state representations, both can be solved through deep reinforcement learning with a neural network having the same structure.

## III. Methodology

While a variety of methods exist to solve the CPP and DH problems separately, the approach presented in the following can be directly applied to both distinct path planning problems. In most classical CPP approaches, individual target zones are extracted through segmentation and then connected with distance costs into a graph, while each segment is covered with a boustrophedon path. This reduces the CPP problem to an instance of the travelling salesman problem (TSP), which is NP-hard and can be solved by classical methods, e.g. as demonstrated in [13] at the price of an exponential increase in time complexity with the number of target zones.

In principle, the DH problem can also be converted into a TSP with the IoT devices as nodes in the graph and the distances between the devices as edge costs. However, the conversion neglects that communication with the device happens while traveling to and from it. In general, the optimal behavior in DH problems is not a sequential visit of all devices, as data can already be efficiently collected by establishing a LoS link from farther away, or a large amount of data waiting to be collected might require the drone to hover for an extended period of time near the device. These constraints in conjunction with stochastic communication channel models and the various possibilities for the choice of multiple access protocol are non-trivial to model and solve with classical approaches. For both problems, the UAV battery constraint adds another complication for classical approaches, as full coverage or full collection are not always feasible. The following DRL methodology allows us to combine all goals and constraints of the respective path planning problems directly without the need for additional approximations.

### A. Partially Observable Markov Decision Process

To address the aforementioned problems we formulate them as a partially observable Markov decision process (POMDP) [14] which is defined through the tuple $(\mathcal{S}, \mathcal{A}, P, R, \Omega, \mathcal{O}, \gamma)$. In the POMDP, $\mathcal{S}$ describes the state space, $\mathcal{A}$ the action space, and $P : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ the transition probability function. $R : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto \mathbb{R}$ is the reward function mapping state, action, and next state to a real valued reward. The observation space is defined through $\Omega$ and $\mathcal{O} : \mathcal{S} \mapsto \Omega$ is the observation function. The discount factor $\gamma \in [0, 1]$ varies the importance of long and short term rewards.

We unify the UAV path planning problems by describing their state space with

$$\mathcal{S} = \underbrace{\mathbb{B}^{M\times M\times 3}}_{\substack{\text{Environment}\\ \text{Map}}} \times \underbrace{\mathbb{R}^{M\times M}}_{\substack{\text{Target}\\ \text{Map}}} \times \underbrace{\mathbb{N}^2}_{\text{Position}} \times \underbrace{\mathbb{N}}_{\substack{\text{Flying}\\ \text{Time}}}, \tag{3}$$

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

$$s(t) = (\mathbf{M}, \mathbf{D}(t), \mathbf{p}(t), b(t)). \tag{4}$$

The four components of the tuple are

- $\mathbf{M}$ the environment map containing start and landing zones, no-fly zones, and obstacles;
- $\mathbf{D}(t)$ the target map indicating remaining data at device locations or remaining cells to be uncovered at time $t$;
- $\mathbf{p}(t)$ the UAV's position at time $t$;
- $b(t)$ the UAV's remaining movement budget at time $t$;

Action $a(t) \in \mathcal{A}$ of the UAV at time $t$ is given as one of the possible actions

$$\mathcal{A} = \{\text{north}, \text{east}, \text{south}, \text{west}, \text{hover}, \text{land}\}.$$

The generalized reward function $R(s(t), a(t), s(t+1))$ consists of the following elements:

- $r_c$ *(positive)* the data collection or cell covering reward given by the collected data or the amount of newly covered target cells, comparing $s(t+1)$ and $s(t)$;
- $r_{sc}$ *(negative)* safety controller (SC) penalty in case the drone has to be prevented from colliding with a building or entering an NFZ;
- $r_{mov}$ *(negative)* constant movement penalty that is applied for every action the UAV takes without completing the mission;
- $r_{crash}$ *(negative)* penalty in case the drone's remaining flying time reaches zero without having landed safely in a landing zone.

### B. Map Processing

To aid an agent in interpreting the large state space given in (3), two map processing steps are used. The first is centering the map around the agent's position, shown in [2] to improve the agent's performance significantly. The downside of this approach is that it increases the representation size of the state space even further. Thus, the second map processing

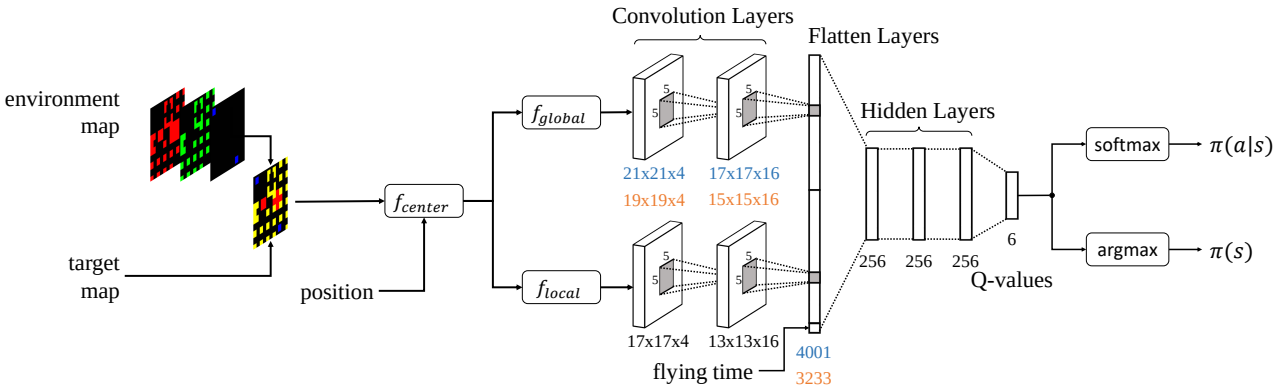



Fig. 1: DQN architecture with map centering and global and local mapping, showing differences of layer size in blue for the 'Manhattan32' and orange for the 'Urban50' scenario.

step, the main contribution of this work, is to present the centered map as two inputs: a full-detail local map showing the agent's immediate surroundings and a compressed global map showing the entire environment with less detail. The mathematical description of the three functions is presented in the following. Fig. 1 indicates where the functions are used within the data pipeline.

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

$$\mathbf{B} = f_{\text{center}}(\mathbf{A}, \mathbf{p}, \mathbf{x}_{\text{pad}}), \tag{5}$$

with the centering function defined as

$$f_{\text{center}} : \mathbb{R}^{M\times M\times n} \times \mathbb{N}^2 \times \mathbb{R}^n \mapsto \mathbb{R}^{M_c\times M_c\times n}. \tag{6}$$

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

$$\mathbf{b}_{i,j} = \begin{cases} \mathbf{a}_{i+p_0-M+1,j+p_1-M+1}, & M \le i+p_0+1 < 2M \\ & \wedge\ M \le j+p_1+1 < 2M \\ \mathbf{x}_{\text{pad}}, & \text{otherwise}, \end{cases} \tag{7}$$

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 $\mathbb{R}^n$. For both problems, the map layers are padded with $[0,1,1,0]^{\mathrm{T}}$, i.e. NFZs and obstacles. A qualitative description of centering with an example can be found in [2].

*2) Global-Local Mapping:* 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

$$\mathbf{X} = f_{\text{local}}(\mathbf{B}, l) \tag{8}$$

with the local map function defined by

$$f_{\text{local}} : \mathbb{R}^{M_c\times M_c\times n} \times \mathbb{N} \mapsto \mathbb{R}^{l\times l\times n}. \tag{9}$$

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

$$\mathbf{x}_{i,j} = \mathbf{b}_{i+M-\lceil \frac{l}{2} \rceil, j+M-\lceil \frac{l}{2} \rceil}. \tag{10}$$

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

The global map is created according to

$$\mathbf{Y} = f_{\text{global}}(\mathbf{B}, g) \tag{11}$$

with the global map function defined by

$$f_{\text{global}} : \mathbb{R}^{M_c\times M_c\times n} \times \mathbb{N} \mapsto \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor \times n}. \tag{12}$$

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

$$\mathbf{y}_{i,j} = \frac{1}{g^2} \sum_{u=0}^{g-1} \sum_{v=0}^{g-1} \mathbf{b}_{gi+u,gj+v}, \tag{13}$$

which is an operation equal to average pooling.

The functions $f_{\text{local}}$ and $f_{\text{global}}$ 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, decreasing the size of the global map.

### C. Observation Space

The observation space $\Omega$, which is the input to the agent, is given as

$$\Omega = \Omega_l \times \Omega_g \times \mathbb{N}$$

containing the local map $\Omega_l = \mathbb{B}^{l\times l\times 3} \times \mathbb{R}^{l\times l}$ and the global map $\Omega_g = \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor \times 3} \times \mathbb{R}^{\lfloor \frac{M_c}{g} \rfloor \times \lfloor \frac{M_c}{g} \rfloor}$. Note that the compression of the map layers through average pooling transforms the environment layers from boolean to real. The observations $o(t) \in \Omega$ are defined through the tuple

$$o(t) = (\mathbf{M}_l(t), \mathbf{D}_l(t), \mathbf{M}_g(t), \mathbf{D}_g(t), b(t)). \tag{14}$$

In the observation, $\mathbf{M}_l(t)$ and $\mathbf{M}_g(t)$ are the local and global observations of the environment, and $\mathbf{D}_l(t)$ and $\mathbf{D}_g(t)$ are the local and global observations of the target, respectively. $b(t)$ is the remaining flying time of the UAV and is equal to the one in the state space. Note that the local and global observations of the environment are time-dependent, as they are centered around the time-dependent position of the UAV.

The mapping from state to observation space is given by $\mathcal{O}: \mathcal{S} \mapsto \Omega$, with the elements $o(t) \in \mathcal{O}$ defined as:

$$\mathbf{M}_l(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}(t), [0,1,1]^{\mathrm{T}}), l) \quad (15a)$$

$$\mathbf{D}_l(t) = f_{\text{local}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}(t), 0), l) \quad (15b)$$

$$\mathbf{M}_g(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{M}, \mathbf{p}(t), [0,1,1]^{\mathrm{T}}), g) \quad (15c)$$

$$\mathbf{D}_g(t) = f_{\text{global}}(f_{\text{center}}(\mathbf{D}(t), \mathbf{p}(t), 0), g) \quad (15d)$$

By feeding the observation space $\Omega$ into the agent instead of the state space $\mathcal{S}$, the problem is artificially converted into a partially observable MDP. The partial observability results from the restricted size of the local map and the averaging in the global map. With the following results, we show that partial observability does not make the problem infeasible for memory-less agents and that the compression greatly reduces the size of the neural network, yielding significantly less training time.

### D. Double Deep Reinforcement Learning - Neural Network

To solve the aforementioned POMDP, we use reinforcement learning, specifically double deep Q-networks (DDQNs) proposed by Van Hasselt *et al.* [15]. DDQNs approximate the Q-value of each state-action pair given as

$$Q^{\pi}(s(t), a(t)) = \mathbb{E}_{\pi}\left[\sum_{k=t}^{T} \gamma^{k-t} R(s(k), a(k), s(k+1))\right], \quad (16)$$

describing the discounted cumulative reward of an agent following policy $\pi$. To converge to the optimal Q-value the agent explores the environment, collecting experiences $(s(t), a(t), r(t), s(t+1))$ and storing them as $(s, a, r, s')$ in a replay memory $\mathcal{D}$, omitting temporal information. Two Q-networks parameterized through $\theta$ and $\bar{\theta}$ are used, in which the first Q-network is updated by minimizing the loss

$$L(\theta) = \mathbb{E}_{s,a,s' \sim \mathcal{D}}[(Q_{\theta}(s,a) - Y(s,a,s'))^2] \quad (17)$$

given by experiences in the replay memory. The target value is given by

$$Y(s,a,s') = r(s,a) + \gamma Q_{\bar{\theta}}(s', \underset{a'}{\operatorname{argmax}}\, Q_{\theta}(s', a')). \quad (18)$$

The parameters of the second Q-network are updated as $\bar{\theta} \leftarrow (1-\tau)\bar{\theta} + \tau\theta$ with the soft update parameter $\tau \in (0, 1]$. To address training sensitivity to the size of the replay memory we make use of combined experience replay proposed by Zhang and Sutton [16].

The neural network architecture used for both Q-networks is shown in Fig. 1. The environment map and target map are stacked and centered around the UAV position and then converted into global and local observation components. After being fed through two convolutional layers each, the resulting tensors are flattened and concatenated with the remaining flying time input and passed through three hidden layers with ReLU activation functions. The output layer with no activation function represents the Q-values directly, passed through a softmax function to create an action distribution for exploration or an argmax function for exploitation.

| Parameter | $32 \times 32$ | $50 \times 50$ | Description |
|---|---|---|---|
| $\lvert\theta\rvert$ | 1,175,302 | 978,694 | trainable parameters |
| $l$ | 17 | 17 | local map size |
| $g$ | 3 | 5 | global map scaling |
| $n_c$ | 2 | | number of conv. layers |
| $n_k$ | 16 | | number of kernels |
| $s_k$ | 5 | | conv. kernel width |

TABLE I: Hyperparameters for $32 \times 32$ and $50 \times 50$ maps.

The relevant parameter for scalability is the size of the flatten layer. It can be calculated through

$$N = n_k \left( \left( l - n_c \lfloor \frac{s_k}{2} \rfloor \right)^2 + \left( \lfloor \frac{M_c}{g} \rfloor - n_c \lfloor \frac{s_k}{2} \rfloor \right)^2 \right) + 1 \quad (19)$$

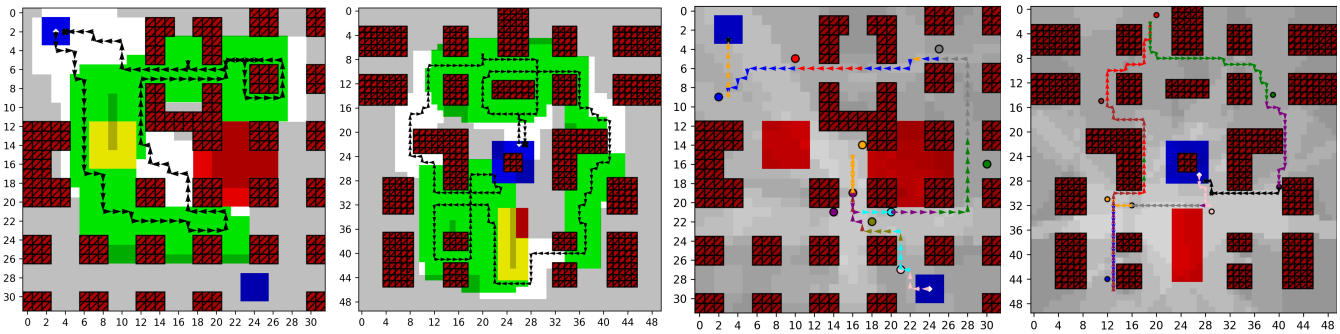
with $n_k$ being the number of kernels, $n_c$ the number of convolutional layers, and $s_k$ being the kernel size. Setting the global map scaling parameter to $g = 1$ and the local map size to $l = 0$ deactivates global-local map processing, i.e., no downsampling and no extra local map. The parameters used in evaluation are listed in Table I.

## IV. Simulations

### A. Simulation Setup

The UAV is flying in two different grid worlds. The 'Manhattan32' scenario (Fig. 2a and 2c) with $32 \times 32$ cells with two starting and landing zones in the top left and bottom right corners. Besides regular building patterns, some irregularly shaped buildings and additional NFZs are present. The 'Urban50' scenario (Fig. 2b and 2d) contains $50 \times 50$ cells and one starting and landing area around the center building. Buildings are generally larger and spaced out, and an additional large NFZ is present on the bottom of the map. Note that the number of cells in the 'Urban50' map is roughly one magnitude larger than in the previous works [1] and [2]. The cell size for the scenarios is $10\text{m} \times 10\text{m}$ with Table II providing a legend for the plots.

*1) Coverage Path Planning:* For the CPP problem, the UAV is flying at a constant altitude of $25\text{m}$ with a camera mounted underneath that has a field of view angle of $90°$. Consequently, the UAV can cover an area of $5 \times 5$ cells simultaneously, as long as obstacles do not block line of sight. The target areas are generated by randomly sampling geometric shapes of different sizes and types and overlaying them, creating partially connected target zones. For evaluation, a traditional metric for the CPP problem is the path length. However, this metric only offers meaningful comparison when full coverage is possible. In this work, we investigate flight time constrained CPP, in which full coverage is often impossible. Therefore, the evaluation metrics used are the coverage ratio (CR), i.e. the ratio of covered target cells to the initial target cells at the end of the episode, and coverage ratio and landed (CRAL), which is zero if the UAV did not land successfully and equal to CR if it did. The benefits of the CRAL metric are that it combines the two goals, achieving high coverage and returning to the landing zone



(a) Movement 124/140, CR=0.94 (b) Movement 234/250, CR=0.94 (c) Movement 92/150, CR=0.99 (d) Movement 191/200, CR=1.0

Fig. 2: Example trajectories from the Monte Carlo simulations for CPP (a)+(b) and DH (c)+(d) on $32 \times 32$ Manhattan map (a)+(c) and $50 \times 50$ Urban map (b)+(d).

| | Symbol | Description |
|---|---|---|
| DQN Input | | Start and landing zone |
| | | Regulatory no-fly zone (NFZ) |
| | | Buildings blocking wireless links and FoV |
| | | **DH:** IoT device |
| | | **CPP:** Remaining target zone (yellow also NFZ) |
| Visualization | | **DH:** Summation of building shadows |
| | | **DH:** Movement while comm. with green device |
| | | **DH:** Hovering while comm. with green device |
| | | **CPP:** Not covered and covered |
| | | Starting and landing positions during an episode |
| | | Actions without comm. |

TABLE II: Legend for scenario plots, **DH** and **CPP** are only applicable in data harvesting and coverage path planning scenarios, respectively.

within the flight time constraint. By normalizing performance to a value in $[0, 1]$, it enables performance comparisons over the changing scenarios with randomly generated target zones.

*2) Data Harvesting:* In the DH problem, the UAV is flying at a constant altitude of $10\text{m}$ communicating with devices on ground level. The achievable data rate is calculated based on distance, random shadow fading, and line-of-sight condition with the same communication channel parameters used in [2]. As in CPP, the path length is not an applicable metric. It is impossible to collect all data in all scenarios depending on the randomly changing locations of IoT devices, data amount, and maximum flying time. Therefore, the evaluation metric used is the collection ratio (CR), describing the ratio of collected data from all devices to the initially available data summed over all devices. Like in CPP, we also use collection ratio and landed (CRAL) in this context, showing the full data collection and landing performance in one normalized metric.

### B. General Evaluation

The CPP agents were trained on target zones containing 3-8 shapes covering 20-50% of the available area. The movement range was set to 50-150 steps for the 'Manhattan32' scenario and 150-250 for the 'Urban50' scenario. For the DH scenarios, 3-10 devices are placed randomly in free cells and contain 5.0-20.0 data units. The movement range was set to 50-150 steps for the 'Manhattan32' scenario and 100-200 for the 'Urban50' scenario. Four scenarios are evaluated in detail.

| Metric | Manhattan32 CPP | Manhattan32 DH | Urban50 CPP | Urban50 DH |
|---|---|---|---|---|
| Landed | 98.5% | 98.2% | 98.1% | 99.5% |
| CR | 71.0% | 83.6% | 81.5% | 74.5% |
| CRAL | 70.3% | 82.5% | 80.1% | 74.2% |

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

In the CPP scenarios, the agents in Fig. 2a and 2b show that they can find trajectories to cover most of the target area. Even the area in the NFZs is mostly covered. It can be seen that small areas that would require a detour are ignored, leading to incomplete coverage. However, most of the target area is covered efficiently.

The agents in the DH scenarios in Fig. 2c and 2d perform very well. In the 'Manhattan32' scenario, the agent leaves small amounts of data at the orange and purple devices totaling a collection ratio of 99.1%. However, the agent finds a concise path, using only 92 of the allowed 150 movement steps. In the 'Urban50' scenario, the agent manages to collect all the data and return with some movement steps in spare.

All four agents were trained for 2 million steps. When analyzing their performance in all four missions using 1000 Monte Carlo generated scenarios (see Table III), it can be seen that all agents' landing performances are good, with the 'Urban50' DH agent being slightly better.

### C. Global-Local Parameter Evaluation

To establish the performance sensitivity to the new hyper-parameters, global map scaling $g$, and local map size $l$, we trained multiple agents with different parameters on the CPP and DH problems. We chose four values for $l$ and four for $g$ and trained three agents for each possible combination. Additionally, we trained three agents without the usage of global and local map processing, which is equivalent to setting $g = 1$ and $l = 0$. The resulting 51 agents for the

| Global map scaling $g$ | Local map scaling $l$ | | | |
|---|---|---|---|---|
| | 9 | 17 | 25 | 33 |
| 2 | 8,481 | 9,761 | 13,089 | 18,465 |
| 3 | 2,721 | 4,001 | 7,329 | 12,705 |
| 5 | 273 | 1,553 | 4,881 | 10,257 |
| 7 | 33 | 1,313 | 4,641 | 10,017 |

TABLE IV: Flatten layer size for 'Manhattan32' with different global map scaling and local map sizes; Without global-local map processing the size is 48,401 neurons.

| Global map scaling $g$ | Local map scaling $l$ | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 9 | | 17 | | 25 | | 33 | |
| | CPP | DH | CPP | DH | CPP | DH | CPP | DH |
| 2 | 2.7 | 2.2 | 2.3 | 2.0 | 1.8 | 1.6 | 1.3 | 1.1 |
| 3 | 3.5 | 3.0 | 3.0 | 2.5 | 2.2 | 1.9 | 1.6 | 1.4 |
| 5 | 4.2 | 3.6 | 3.4 | 3.0 | 2.5 | 2.2 | 1.9 | 1.6 |
| 7 | 4.7 | 3.8 | 3.6 | 3.0 | 2.5 | 2.2 | 2.5 | 2.1 |

TABLE V: Training time speedup for the CPP and DH problem relative to without global-local map processing.

CPP and DH problems were trained for 500k steps each and evaluated on 200 Monte Carlo generated scenarios. The difference to the previous evaluation is that the movement budget range was set to $150 - 300$.

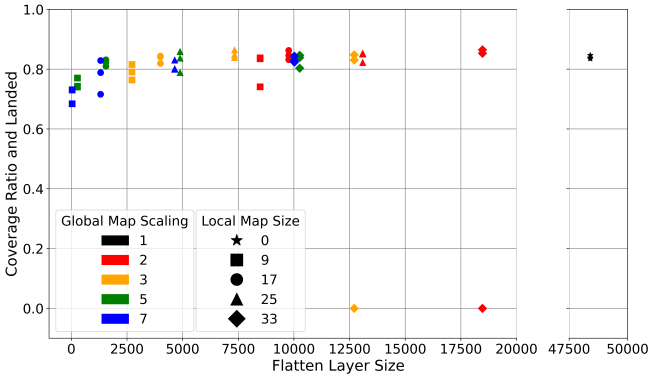
Table IV shows the selected parameters and the resulting flatten layer size according to (19). A significant speedup of the training process compared to agents without global and local map processing can be observed in Table V.

The resulting CRAL values from the Monte Carlo simulations for each agent with respect to the agent's flatten layer size are shown in Fig. 3a and 3b for the CPP and DH problem, respectively. It can be seen that the DH problem is more sensitive to the parameters than the CPP problem. Generally, a larger flatten layer yields better performance up to a point. For both problems, it can be seen that a large flatten layer can cause the learning to get unstable, resulting in a CRAL of zero for some runs. This is caused by the agent's failure to learn how to land. The DH agents, which are not using the global-local map approach, never learn how to land reliably and thus have a CRAL score near zero.

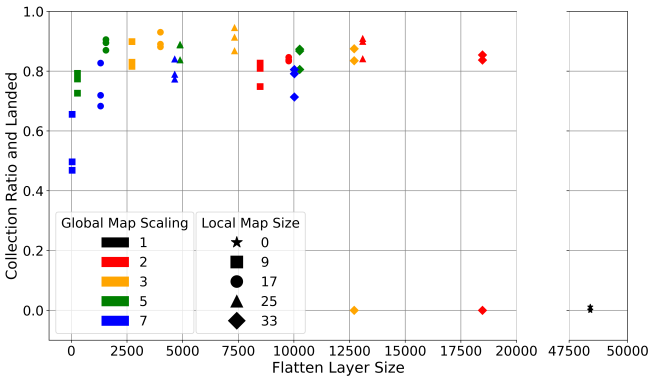
In both cases, the agents with $l = 17$ and $g = 3$ or $g = 5$ show the best performance with respect to their flatten layer size, justifying the selection in Table I. Besides these two parameter combinations, it is noteworthy that the agents with $l = 9$ and $g = 7$ also perform well in both scenarios, despite their small flatten layer size of only 33 neurons.

## V. Conclusion

We have presented a method for generalizing autonomous UAV path planning over two distinctly different mission types, coverage path planning and data harvesting. Through the flexibility afforded by combining specific mission goals and navigation constraints in the reward function, we trained DDQNs with identical structures in both scenarios to make




(a) Grid search for CPP




(b) Grid search for DH

Fig. 3: Parameter grid search for CPP and DH with parameters from Table IV; the black stars correspond to agents without global-local map processing.

efficient movement decisions. We have introduced a novel global-local map processing scheme that allows to feed large maps directly into convolutional layers of the DRL agent and analyzed the effects of map processing parameters on learning performance. In future work, we will investigate still existing hindrances for applying our method to even larger maps, namely avoiding small-scale decision alternation through the use of macro-actions or options [17]. Combining the presented high-level path planning approach with a low-level flight dynamics controller will also make it possible to conduct experiments with realistic open-source UAV simulators in the future. Additionally, we will investigate the effect of irregularly shaped, non-convex obstacles on the path planning performance.

## Acknowledgments

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research. Harald Bayerlein and David Gesbert were partially supported by the French government, through the 3IA Côte d'Azur project number ANR-19-P3IA-0002, as well as by the TSN CARNOT Institute under project Robots4IoT.

## REFERENCES


[1] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "UAV coverage path planning under varying power constraints using deep reinforcement learning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020.

[2] H. Bayerlein, M. Theile, M. Caccamo, and D. Gesbert, "UAV path planning for wireless data harvesting: A deep reinforcement learning approach," in *IEEE Global Communications Conference (GLOBECOM)*, 2020.

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

[4] R. Shakeri, M. A. Al-Garadi, A. Badawy, A. Mohamed, T. Khattab, A. K. Al-Ali, K. A. Harras, and M. Guizani, "Design challenges of multi-UAV systems in cyber-physical applications: A comprehensive survey and future directions," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3340–3385, 2019.

[5] T. Cabreira, L. Brisolara, and P. R Ferreira, "Survey on coverage path planning with unmanned aerial vehicles," *Drones*, vol. 3, no. 1, 2019.

[6] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proceedings of the 13th International Conference on Distributed Smart Cameras*, ACM, 2019.

[7] K. D. Julian and M. J. Kochenderfer, "Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 8, pp. 1768–1778, 2019.

[8] E. Seraj and M. Gombolay, "Coordinated control of UAVs for human-centered active sensing of wildfires," in *American Control Conference (ACC)*, pp. 1845–1852, IEEE, 2020.

[9] D. Baldazo, J. Parras, and S. Zazo, "Decentralized multi-agent deep reinforcement learning in swarms of drones for flood monitoring," in *27th European Signal Processing Conference (EUSIPCO)*, IEEE, 2019.

[10] O. Esrafilian, R. Gangula, and D. Gesbert, "Learning to communicate in UAV-aided wireless networks: Map-based approaches," *IEEE Internet of Things Journal*, vol. 6, no. 2, pp. 1791–1802, 2018.

[11] 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 Transactions on Mobile Computing*, vol. 20, no. 1, pp. 130–146, 2021.

[12] S. Zhang and R. Zhang, "Radio map based path planning for cellular-connected UAV," in *IEEE Global Communications Conference (GLOBECOM)*, 2019.

[13] J. Xie, L. R. G. Carrillo, and L. Jin, "An integrated traveling salesman and coverage path planning problem for unmanned aircraft systems," *IEEE Control Systems Letters*, vol. 3, no. 1, pp. 67–72, 2019.

[14] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra, "Planning and acting in partially observable stochastic domains," *Artificial intelligence*, vol. 101, no. 1-2, pp. 99–134, 1998.

[15] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Thirtieth AAAI conference on artificial intelligence*, pp. 2094–2100, 2016.

[16] S. Zhang and R. S. Sutton, "A deeper look at experience replay," *arXiv:1712.01275 [cs.LG]*, 2017.

[17] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999.