Drone mapping through multi-agent reinforcement learning

Riccardo Zanol, Federico Chiariotti, Andrea Zanella
Department of Information Engineering, University of Padova – Via Gradenigo, 6/b, 35131 Padova, Italy
Email: {zanolric, chiariot, zanella}@dei.unipd.it

***Abstract*—In recent years, the use of drones to map environments and survey them for items of interest such as forest fires, landslides or wild animals has gained traction in various research communities. However, the need for a human pilot or a pre-planned flight path severely limits the effectiveness of the drones, especially when a whole swarm is used. In this work, we propose a model of the drone survey problem and apply three well-known reinforcement learning strategies, showing that the performance loss due to the lack of explicit optimization and pre-programmed knowledge of the system statistics is negligible in the swarm scenario.**

I. Introduction

Over the last few years, the rapid technological improvement in the capabilities and battery life of Unmanned Aerial Vehicles (UAVs) has opened new possibilities in remote sensing. Environmental surveying and monitoring is an expensive and time-consuming task, and performing it automatically even in inaccessible areas would be a boon to several research communities and public services [1], [2].

The reconstruction of a 3D map of an environment from multiple pictures captured by UAVs [3] is a well-studied subject; examples from the mapping of natural terrain [4] to the survey of an archaeological site [1] or the detection of forest fires [2] have been studied in the literature. The standard technique is to use feature extraction tools such as Scale-Invariant Feature Transform (SIFT) [5] to match sparse features in the images, then compute an estimation of the camera and pose parameters, which is then refined using a bundle adjustment algorithm [6]. It is also possible to build a probabilistic depth map in real time using a Bayesian technique [7] if the camera pose is known, even if there is no GPS availability [8].

The use of multiple drones to reconstruct the 3D map of the environment has also been studied in the literature; in [9], two drones follow a ground vehicle and function as a stereo camera system, while [10] extends the idea to a human-controlled formation of drones building a depth map of the scene on the ground.

In all the studies listed above, the path of the drones is either pre-determined or directly controlled by a human; 3D mapping using autonomous UAVs is a more complex problem, since the drones would need to decide the most efficient way to scan the environment. This problem is known as the Next Best View (NBV) problem [11], and its solution is the position that maximizes the information gain [12], i.e., the new knowledge about the features of the observed object. The NBV problem can be reduced to planning a set of poses for a depth sensor (e.g. a stereo camera or LADAR), which is usually chosen greedily after each capture for computational reasons.

The approach described by [13], where the NBV selection algorithm was designed specifically for UAVs, consists of the generation of a set of candidate next views and searching for the candidate that maximizes a utility function. The utility function takes into account the information gain about the voxel occupancy, the cost to reach the new observation point, in terms of distance and "smoothness" of the trajectory, and the battery level. Other voxel-based NBV algorithms use a similar strategy [12], [14], maximizing a utility function over a set of candidate views.

In this work, we propose a model and a Reinforcement Learning (RL) approach to dynamically determine the trajectories of multiple drones monitoring a time-varying environment. RL is an efficient way to find optimal policies in any dynamic problem that can be represented as a Markov Decision Process (MDP); its solution is an approximation of the one found by the optimal dynamic programming approach, but it works without any *a priori* knowledge about the statistics of the environment and is far less computationally demanding [15]. In the rest of this work, we show how RL provides a good approximation of the computationally expensive optimal strategy, while being efficient enough to be performed in real time.

The rest of this work is divided as follows: Sec. II presents the model of the studied problem and the maximum achievable performance, while the analysis of the RL performance is performed in Sec. III. Finally, Sec. IV concludes the paper and lists some possible future avenues of research on the subject.

II. System model

The considered problem involves a number of drones patrolling the area surrounding a fixed staging point, reconstructing a live 3D map of the environment. The UAVs might need to return to previously surveyed areas, since the environment is assumed to be dynamic; they will also need to periodically return to the staging point in order to recharge their batteries.

In order to apply RL to the problem, we formulate it as a MDP; in the multi-agent case, it will be necessary to use multiple MDPs, since each agent will have its own state and rewards and it will choose its actions independently. The problem of multi-agent RL is complex: the actions of each independent agent can combine in unpredictable ways, with emergent effects that may go against the global objective if the reward function is not defined optimally [16]. In general, there are several ways to approach multi-agent RL [17]: one extreme is a fully centralized controller dictating the actions of all agents with a full picture of the overall state of the system, which avoids the coordination problem but significantly increases the complexity of the problem and the communication and signaling overhead, while the other is a fully distributed system in which agents only have a partial knowledge of the system and act independently to maximize their own objective

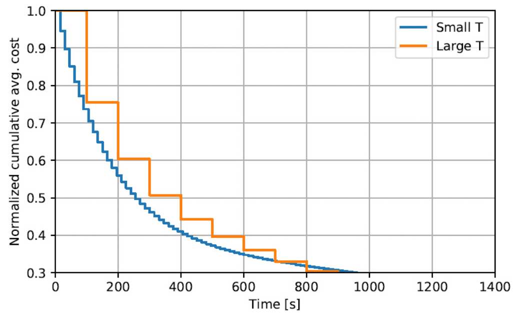

Fig. 1: Normalized cumulative average cost given by the two-states Markov chain with two sets of parameters. "Small T": $p_{\text{ON,OFF}} = \frac{1}{500}, p_{\text{OFF,ON}} = \frac{1}{12}, T = 15$ s, "Large T": $p_{\text{ON,OFF}} = \frac{1}{45}, p_{\text{OFF,ON}} = \frac{1}{2.5}, T = 100$ s.

function. Existing solutions fall on a spectrum between these two extremes, striking different balances between the benefits and drawbacks of distribution and centralization [18].

A. Environment model

In order to correctly model the environment, we need to specify a model for the image acquisition process each drone performs and the battery consumption its activities require.

1) Image acquisition process: Our model only deals with the high-level route planning for each UAV, since we assume the flight control between assigned waypoints to be performed by standard lower-level systems. The details of the 3D reconstruction process are also beyond the scope of this work; we exploit the NBV model in [13] to estimate the quality of the reconstruction.

We approximate the decrease of the normalized uncertainty about the depth map in [13]; the resulting curve is roughly exponential, and can be approximating by modeling the data acquisition process as a two-states Markov chain, where the acquisition state ON corresponds to a reward value of 1 and the inactive state OFF corresponds to a reward of 0. This kind of model can be easily inserted in a MDP.

Even if the actual reward at each time step can only be 0 or 1 with this model, the average reward decreases exponentially as in [13]. With the assumption that the chain starts in state ON, the cumulative average reward after k steps is

$$E\left[\sum_{t=1}^{k} R_t | Z_0 = \text{ON}\right] = \sum_{t=1}^{k} E\left[R_t | Z_0 = \text{ON}\right] = \sum_{t=1}^{k} p_{\text{ON,ON}}^{(t)} \tag{1}$$

where the state of the MC at time t is denoted by Z_t, the reward R_t is just $I\{Z_t = \text{ON}\}$ and $p_{s,s'}^{(t)}$ is the t-steps transition probability from s to s'. The sums in (1) start from 1 because the reward R_t is gained *after* making a state transition: the image at time t must actually be captured before knowing if it provides additional information.

By adjusting the transition probabilities of the chain and the sampling interval T it is possible to obtain a shape for the normalized cumulative average cost that is close to the desired one. Fig. 1 shows two examples: one with a small sampling interval $T = 15$ s and one with a sampling interval of $T = 100$ s. In the first case, the faster pace of the MDP requires the agent to see many more examples before converging to a stationary policy. For this reason, we will use this two-states Markov model with parameters with a long T; this is consistent with the time required to reach the chosen view point and take a high-quality picture of the environment.

2) Battery consumption: The power consumption of UAVs is often modeled with physical dynamic models and electromechanical rotor models [19], and thus need to be tailored to the specific drone. The 3DR Solo model has been studied in the literature, with power consumption models and data for different flight tasks [20], [21]. In our work, we assume the drone to move horizontally between waypoints at a speed of 8 m/s, and that the process of mapping the area around each waypoint is performed slowly enough to be equivalent, in terms of required power, to stationary hovering.

The power consumption model given by [20] focuses on the effect of the movement angle with respect to the ground. The resulting power consumption is modeled as a Gaussian variable, with parameters that depend on the flight direction and speed. Combining this result with the more detailed data reported by [21] (which includes different speeds and load weights), it is possible to adapt the Gaussian model to our scenario and specify the model parameters (μ_h, σ_h^2) when hovering and (μ_m, σ_m^2) when moving between waypoints. The value of the power consumption is also assumed to be constant during the execution of each action (i.e. during a time slot T).

The battery state can then be calculated using the nonlinear voltage-current relationship used by [22]:

$$\begin{aligned} V_b(t) &= E(t) - RI_b(t) \\ E(t) &= E_0 - K\frac{Q}{Q - \int_0^t I_b(t)dt} + Ae^{-B\int_0^t I_b(t)dt}, \end{aligned} \tag{2}$$

where I_b and V_b are, respectively, the output current and voltage of the battery, E is the no-load output voltage, Q is the battery capacity, R is the internal resistance and A, B, K, E_0 are parameters that can be derived from a constant-current discharge plot of the battery (which is typically included by the manufacturer in data sheets). The State of Charge (SOC) of this battery model can be defined as the available charge normalized with respect to the full charge:

$$\text{SOC}(t) = 1 - \frac{\int_0^t I_b(t)dt}{Q}. \tag{3}$$

Assuming that both the required power and the output current are constant during each time slot, the output voltage from (2) can be discretized, yielding the following equation:

$$P(kT) = E(kT)I_b(kT) - RI_b^2(kT) \tag{4}$$

which, given the power required in slot k and the total charge that the battery provided up to slot $k-1$, is just a quadratic equation in $I_b(kT)$. Assuming, for simplicity, a UAV with a random strategy, with the probability of moving to an adjacent waypoint defined as

$$P\left(A_k \neq (0,0)\right) = p_{\text{move}} \quad \forall k. \tag{5}$$

Simulating 100000 battery discharges using the battery and power consumption models described above, with time slots of $T = 10$ s and three different values for p_{move}, produces the SOC evolutions shown in Fig. 2. The almost linear behavior of the SOC makes it possible to approximate the full model in

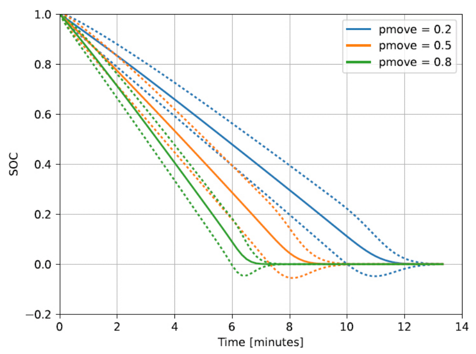

Fig. 2: Battery discharge simulation with Independent and Identically Distributed (IID) actions. The time slots are 10 s long, the SOC values are averaged over 100,000 runs and are shown with their 95% CI.

[22] by assuming the nominal battery voltage ($V_b = V_{\text{nom}} = 14.8$ V).

As Fig. 2 shows, the flight endurance that can be expected from the quadcopter and battery used in this thesis is less than 15 minutes. While this time would be more than enough to map an area of complexity and extension similar to the building in [13], it would be insufficient to operate in a larger environment and could require multiple returns to the starting point for a battery recharge.

The systems like those described by [23]–[25], which automatically replace and recharge the batteries of drones that land on them, are preferable to directly recharging the drone battery: if the number of charging stations and their capabilities of recharging the stored batteries simultaneously are sufficient it is possible to guarantee the battery availability indefinitely for a single UAV and for many hours for a fleet of UAVs [23].

For the reasons above and to avoid having to deal with a large number of (mostly idle) agents in the MDP, the battery replacement system is assumed to be fast enough to perform the swap in one time step and to guarantee an indefinite number of replacements; the battery state is represented in the model as a quantized value, using a non-uniform quantizer which represents low battery states with higher accuracy.

B. Markov Decision Process formulation

We can now formulate the drone coordination problem as an MDP, using the environment model we defined in Sec. II-A. In the multi-agent case, the model of the scenario will need to include multiple MDPs, since each of the N drones will act independently of the others. The lack of coordination means that each agent will choose its own action at each timestep, with no knowledge about the choices of other agents; reward is also given individually to each agent. Our model is fully distributed, so it requires little signaling effort; cooperation between drones is an emergent property of each individual agent's actions, as there is no explicit coordination effort.

Since all agents affect the global state, however, it is much easier to start by considering the agents' joint actions. Let $\mathcal{S}$ be the global continuous state space and $\mathcal{A}(s)$ the finite action space of every agent; the joint state and action at time k are then defined as $s_k \in \mathcal{S}$ and $\mathbf{a}_k \in \mathcal{A}^N$, respectively. The MDP will then have a state transition probability $P(s_{k+1}|s_k, \mathbf{a}_k)$, which is unknown to the agents. However, in order for the transition probability a single agent experiences to be stationary, the policies of other agents also need to be stationary. Since all agents in the system learn at the same time, this is not possible. The model then becomes a Partially Observable Markov Decision Process (POMDP), with all the negative consequences on the convergence properties of Q-Learning (QL) [26].

In order for tabular QL algorithms to work, the continuous state space will need to be quantized. The area the drones patrol can be divided into M^2 square cells, where M is the number of cells the region is quantized into in each of the two Cartesian directions. The movement between cells is deterministic and is controlled directly by each agent's action, subject to some constraints due to battery consumption, collision avoidance and the grid boundary. Each agent can usually choose between five actions in the set $\mathcal{A} = \{(0,0),(0,1),(0,-1),(1,0),(-1,0)\}$, which will make it stay in the same cell or shift to a neighboring cell along one of the two axes. The movement actions are restricted when the agent is on the grid boundary and when it is close to another agent. In this second case both agents are prevented from picking an action that could make them collide with each other.

The battery level also affects the available actions and their outcome: if the battery empties, the agent drops off the cell grid and goes into a "return chain", regardless of the action it was carrying out. While in this return chain, the agent can only move towards the staging point to swap its battery after a number of forced steps that depend on its original position (its action space is limited to the return action R, which is not available at any other time). The agents on the return chain are assumed to be flying back at a higher speed ($v_{\text{ret}} = 13$ m/s) than the one used normally, because the energy efficiency of quadcopters increases with speed [21].

Since static drones need to know the cells that need to be surveyed, the state must include the ON/OFF state $z_{x,y}$ of each cell described in Sec. II-A. Since information about empty cells cannot spontaneously increase, cells without a drone cannot go from the ON state to the OFF state, but the opposite can happen at any time, since the environment of previously surveyed cells might change.

The state of charge of the battery of each agent is modeled as mentioned in section II-A, using a Gaussian model for the battery consumption. The SOC of the i-th agent at time step k will be denoted by $b_{i,k} \in \mathcal{B} = [0,1]$; when the battery reaches 0, it is below the critical level and the drone needs to return to the staging point. As long as the drone stays in the staging point cell, its battery is 1, and in all other cases the one-step consumption is given by a Gaussian random variable with parameters (μ, σ) that depend on the action $a_{i,k}$.

The full state space of the global Markov process, shared by the N MDPs, can now be defined by combining all the variables defined above:

$$\begin{aligned} \mathcal{S} &= \mathcal{S}_i^N \times \mathcal{Z} \\ \mathcal{S}_i &= \mathcal{S}_{\text{grid}} \cup \mathcal{S}_{\text{ret}} \\ \mathcal{S}_{\text{grid}} &= \{\mathcal{X}, \mathcal{Y}, \mathcal{B}, \mathcal{A}\} \\ \mathcal{S}_{\text{ret}} &= \{0, 1, \ldots, M_{\text{ret}} - 1\} \end{aligned} \tag{6}$$

where the global state $s \in \mathcal{S}$ is the product of the state variables $s_i \in \mathcal{S}_i$ relative to each of the agents and the state variable $Z \in \mathcal{Z}$ for the reward process.

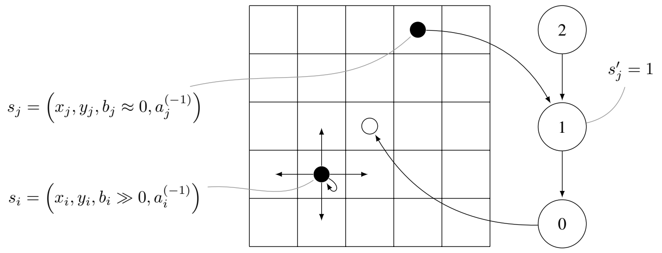
$s_j = \left(x_j, y_j, b_j \approx 0, a_j^{(-1)}\right)$
$s_i = \left(x_i, y_i, b_i \gg 0, a_i^{(-1)}\right)$
$s'_j = 1$
2
1
0

Fig. 3: Illustration of the two kinds of agent states. Agent i, with a high battery level, can successfully make transitions toward the neighboring positions and needs to keep track of its four normal state variables. Agent j, instead, has an almost-empty battery and is forced into the return chain, where it can't do anything except move toward the origin.

Every agent has four state variables: $s_{i,k} = (x_{i,k}, y_{i,k}, b_{i,k}, a_{i,k-1})$. The rift two represent the position of the agent in the cell grid, while the other two represent the SOC of its battery and the last action it took. The first three variables were defined above; the last action is added here in order to allow the agent to remember the cell it comes from. Knowing the previous cell through $a_{i,k-1}$ should help the agent to avoid going back to a location it just visited and is unlikely to have changed so soon.

The structure of $\mathcal{S}_i$ allows to distinguish the agents with a non-empty battery, which can navigate in the grid, from the agents that are returning to the origin to recharge it. When an agent is following the return chain, its normal state variables (from $\mathcal{S}_{\text{grid}}$) are no longer relevant, so they are replaced by the position of the agent in the chain (from $\mathcal{S}_{\text{ret}}$). An example of this behavior is shown in figure 3.

In order to apply tabular RL algorithms, the MDP needs to have a finite number of states. We already described the quantization of the spatial grid in Sec. II-B, so the only continuous variable is $b \in [0,1]$. The state space can be discretized by replacing b with $\tilde{b} \in \tilde{\mathcal{B}} = \left\{\tilde{B}_0, \tilde{B}_1, \dots, \tilde{B}_{|\mathcal{B}|-1}\right\}$. The state variable for the battery SOC b is continuous in the underlying environment, but the agent will only see the quantized value $\tilde{b}$; this makes the battery process lose the Markov property, so the problem becomes a POMDP even with a single agent.

In addition to the quantization of the battery state, it is useful to give the agent a reduced version of the grid state. Firstly, because it may not be realistic for an agent to know the reward process' state for the entire grid; and, secondly, because of the size of $\mathcal{S}$ with the full grid. The single-agent state space has a size

$$|\mathcal{S}| = |\mathcal{Z}| \left(|\mathcal{S}_{\text{grid}}| + |\mathcal{S}_{\text{ret}}|\right) = 2^{M_x M_y} \left(M_x M_y |\mathcal{B}||\mathcal{A}| + M_{\text{ret}}\right), \tag{7}$$

which scales exponentially in the number of cells $M_x M_y$, so a large environment becomes computationally intractable. The simplification of Z_k into $z_{x_k, y_k, k}$ avoids this problem and does not substantially reduce the information available to the agent.

C. Deriving the single-agent performance bound

A way to compute an upper bound on the average reward that the agent can collect is to consider the single-agent MDP with the following optimistic assumptions:

1) the agent can always move into an ON cell;
2) the battery can become empty only when the agent is moving out from a cell;
3) the reward process resets every cell to ON when the agents returns to the origin.

With these assumptions the single-agent MDP reduces to a simpler renewal-reward process: the agent starts with a full battery, it moves into an ON cell, it stays there for L steps, until the first OFF state, and then moves to another cell. This process of visiting a cell repeats K times, until the battery level reaches zero; at this point the agent returns to recharge in M steps and the process begins again. Let the total length of the n-th renewal cycle be T_n, which can be written as

$$T_n = \sum_{k=1}^{K_n} (2 + L_{k,n}) + M_n, \tag{8}$$

where the number of visited cells K_n is

$$K_n = \arg\min_k \left(\sum_{j=1}^{k} \Delta_{\text{SOC},j,n} \geq 1\right) \tag{9}$$

and the battery charge consumed during a cell visit $\Delta_{\text{SOC},j,n}$, due to the single move action and the repeated stay actions, is

$$\Delta_{\text{SOC},j,n} = \Delta^{(m)}_{\text{SOC},j,n} + \sum_{l=1}^{L_{j,n}+1} \Delta^{(h)}_{\text{SOC},l,j,n}, \tag{10}$$

and let the corresponding total reward be

$$R_n = \sum_{i=1}^{K_n} L_{i,n}. \tag{11}$$

Writing explicitly the event that K_n takes a certain value:

$$\{K_n = k\} = \left\{ \sum_{j=1}^{k} \left(\Delta^{(m)}_{\text{SOC},j,n} + \sum_{l=1}^{L_{j,n}+1} \Delta^{(h)}_{\text{SOC},l,j,n} \right) \geq 1, \right.$$
$$\left. \sum_{j=1}^{k-1} \left(\Delta^{(m)}_{\text{SOC},j,n} + \sum_{l=1}^{L_{j,n}+1} \Delta^{(h)}_{\text{SOC},l,j,n} \right) < 1 \right\}. \tag{12}$$

The event $\{K_n = k\}$ is independent both of $N_{j,n}$ and of $\Delta_{\text{SOC},j,n}$ $\forall j > k$. Therefore K_n is a stopping time for $L_{j,n}$ and $\Delta_{\text{SOC},j,n}$, hence Wald's equation applies to both the random sums in (8) and (11) and also to $\sum_{j=1}^{K_n} \Delta_{\text{SOC},j,n}$. This allows us to compute a bound on the average number of visited cells:

$$1 \leq \sum_{j=1}^{K_n} \Delta_{\text{SOC},j,n} < 1 + \Delta_{\text{SOC},K_n,n} \tag{13}$$

$$1 \leq E\left[\sum_{j=1}^{K_n} \Delta_{\text{SOC},j,n}\right] < 1 + E\left[\Delta_{\text{SOC},K_n,n}\right] \tag{14}$$

$$1 \leq E[K_n] E\left[\Delta_{\text{SOC}}\right] < 1 + E\left[\Delta_{\text{SOC}}\right] \tag{15}$$

$$\frac{1}{E\left[\Delta_{\text{SOC}}\right]} \leq E[K] < \frac{1}{E\left[\Delta_{\text{SOC}}\right]} + 1, \tag{16}$$

where the average battery consumption per cell visit is

$$E\left[\Delta_{\text{SOC}}\right] = E\left[\Delta^{(m)}_{\text{SOC}}\right] + E\left[\sum_{l=1}^{L_{j,n}+1} \Delta^{(h)}_{\text{SOC},l,j,n}\right] \tag{17}$$

$$= E\left[\Delta_{\text{SOC}}\right] = \mu_m + \mu_h (1 + E[N]) \tag{18}$$

since $\Delta_{\text{SOC},l,j,n}$ and $L_{j,n}$ are all independent. The average cycle length can also be computed as

$$E[T] = E[K](2 + E[L]) + E[M] \tag{19}$$

while the average reward per cycle is

$$E[R] = E[L]E[K]. \tag{20}$$

Now, using the elementary renewal-reward theorem, the long-term average reward (per slot of the MDP) is

$$\lim_{n\to\infty} \frac{E[R(n)]}{n} = \frac{E[R]}{E[T]} = \frac{E[L]}{2 + E[L] + \frac{E[M]}{E[K]}}, \tag{21}$$

and using the upper bound from (16),

$$\lim_{n\to\infty} \frac{E[R(n)]}{n} \leq \frac{E[L]}{2 + E[L] + \frac{E[\Delta_{\text{SOC}}]E[M]}{1+E[\Delta_{\text{SOC}}]}}. \tag{22}$$

An optimistic, but still realistic, assumption for the distribution of the number of return steps is $M_n = 1, \forall n$. It is realistic because the agent should learn to stay closer and closer to the origin as its battery level decreases; ideally, the battery should deplete while the agent is staying over one of the four cells adjacent to the origin.

III. Performance analysis

The simulation was performed by implementing the model in python and running it over the scenario we described. We defined values for all the parameters in the scenario, which are listed in Table I. Substituting the values in Table I in the equations derived in Sec. II-C, the upper bound on the single-client reward is 0.701.

We implemented three well-known learning algorithms, namely, standard QL, Frequency-Adjusted Q-Learning (FAQL) [27], and Repeated Update Q-Learning (RUQL) [28].

In order to provide a meaningful comparison, we also implemented a Lookahead (LA) strategy, which starts from a complete knowledge of the environment and solves the MDP explicitly using the Bellman equation [29]. Due to the large number of states that can be visited with non-zero probability, the LA strategy's computational load is daunting; even limiting the number of future steps to 3, the computational load to decide an action grows by a factor of 200 with respect to the learning agents, making it unfit for implementation in a real-time scenario. The LA strategy was only used in the single-agent scenario, since its complexity grows exponentially with the number of agents and even a scenario with just two drones makes it impossible to compute in a reasonable time. The average reward of the LA strategy over 50 episodes of 10000 steps each was 0.602, 14% less than the optimistic bound from Sec. II-C. We also implemented a Random Walk (RW) strategy as a basic benchmark, which picks an action at random from the set of available ones, as the name suggests.

TABLE I: Simulation parameters.

Parameter	Symbol	Value
Timestep	T	15 s
Grid size	M	5 cells
Number of drones	N	4
Battery quantization	$\lvert\mathcal{B}\rvert$	10 values
Hovering power consumption (mean)	μ_H	338 W
Hovering power consumption (variance)	σ_H^2	44.7 W^2
Flight power consumption (mean)	μ_F	848 W
Flight power consumption (variance)	σ_F^2	4254.7 W^2
Battery charge	Q	5.5 Ah
Internal battery resistance	R	4.75 mΩ
Nominal battery voltage	V_{nom}	14.8 V
ON state change probability	$p_{\text{ON,OFF}}$	0.002
OFF state change probability	$p_{\text{OFF,ON}}$	0.083
QL/MCTS discount rate	γ	0.6
MCTS exploration	c	$\sqrt{2}$
MCTS depth	D	6

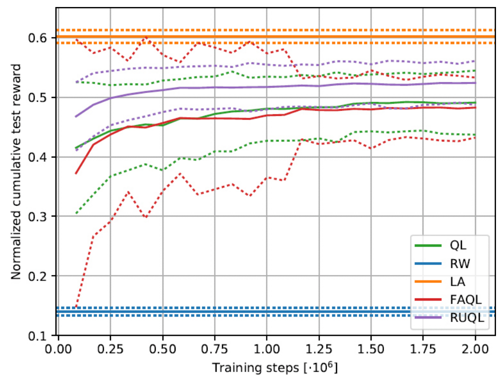

Fig. 4: Convergence of the learning algorithms over the training procedure (2 million total training steps) in the single agent scenario.

In the multi-agent case, we set the number of drones to 4 and retrained the learning agents from scratch using the expanded MDP model; in order to compare the performance of the different agents to a traditional model, we implemented a Monte Carlo Tree Search (MCTS) strategy, as the LA approach we used in the single-agent case was not computationally feasible. MCTS is a common alternative to RL [15], and works by building a tree of possible actions and states, pruning low-value paths and essentially performing a sparse lookahead. MCTS traverses the tree, selecting a leaf, then expands it and uses a default policy to simulate future possibilities. This algorithm is not entirely model-free, because it cannot learn purely from experience; it requires a generative model to simulate transitions starting from arbitrary points in the state space. This can often be much faster than explicitly computing the transition probabilities. However, its computational complexity limits the size of the scenario and the number of drones: no efficient benchmark solution exists for larger scenarios with more drones. The specific MCTS algorithm we applied is the Sparse Upper Confidence bound for Trees (UCT) [30]; it maintains an estimate of the value of each action $Q_t(a)$ by averaging the rewards it received and uses this value estimate to apply the following policy:

$$A_t = \arg\max_a \left(Q_t(a) + c \frac{\log t}{N_t(a)} \right), \tag{23}$$

where $N_t(a)$ is the number of times that action a was chosen up to time t and $c > 0$ is an exploration parameter. UCT considers each state node in the MCTS tree as a stateless decision problem, so it uses the policy in (23). When an action was never chosen ($N_t(a) = 0$), it is assumed that its value is infinite. This implies that the leaf nodes, from which

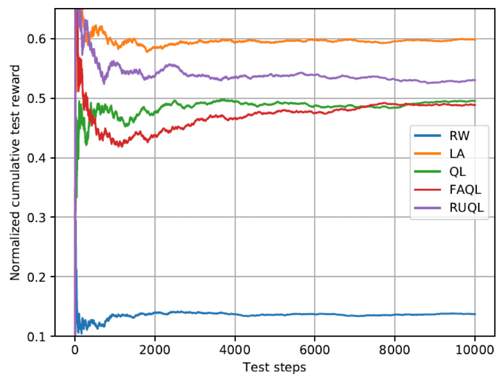

Fig. 5: Rewards obtained by the fully trained agents over a 10000-step episode in the single agent scenario.

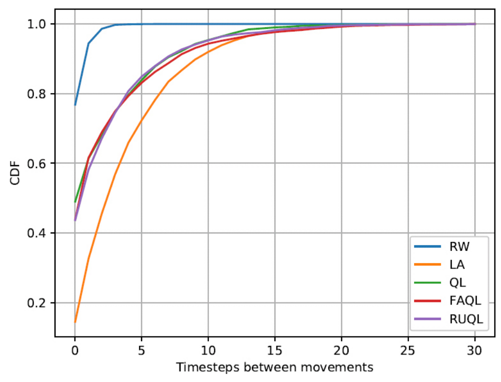

Fig. 6: CDF of the time between movements in the test phase.

the search tree is expanded, are those with actions that were never explored. MCTS is still two orders of magnitude more complex than the RL algorithms, but it is considerably faster than LA in this scenario.

A. Single-agent

In the single-agent case, we have both a theoretical upper bound and a practical one given by the performance of the LA strategy. An ideal RL algorithm would converge at least to the second one, but the effects of the partial observability of the MDP need to be taken into account. As Fig. 4 shows, RUQL converges faster than FAQL and standard QL, reaching a higher average reward.

Fig. 5 shows this difference is significant even in the test phase after the Q-values are frozen, which suggests the 2 million training steps were insufficient or the algorithms got stuck on a local maximum and learned a suboptimal strategy. However, the overall gap between their performance and LA's is relatively small; once the training phase is over, the RL approaches only require a simple table lookup instead of a computationally complex dynamic programming, speeding up the decision by a factor of 200 with respect to LA. The practical feasibility of the RL approach and the relatively small performance loss make it a very attractive solution to the drone trajectory problem, at least when only one drone is involved.

Fig. 6 shows the CDF of the number of timesteps a drone spends surveying the same point; as the figure shows, the RW

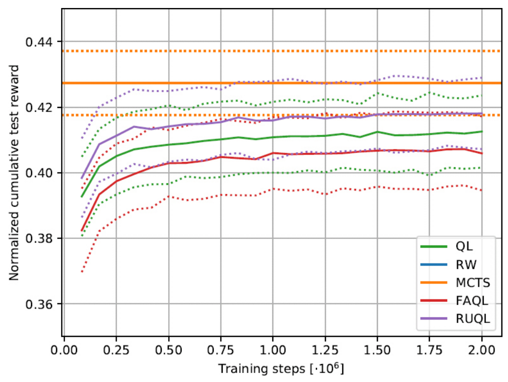

Fig. 7: Convergence of the learning algorithms over the training procedure (2 million total training steps) in the multi-agent scenario.

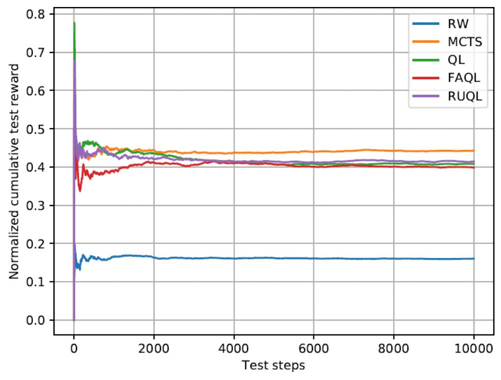

Fig. 8: Rewards obtained by the fully trained agents over a 10000-step episode in the multi-agent scenario.

policy has a very short staying time and expends its energy quickly, while the three learning algorithms have approximately the same performance. The LA algorithm exploits its knowledge of the statistics of the system and spends 4.1 slots in one spot on average, while the learning algorithms only spend 2.5; we plan to find ways to reduce this gap, and the gap in the reward, in a future work.

B. Multi-agent

In this section, we present the results in the multi-agent case; we use the average reward as a metric in order to provide comparable results. Fig. 7 shows that the learning algorithms perform almost as well as MCTS before having had much training, and that the differences between the three learning algorithms are minimal. It is interesting to note that the theoretical bound from Sec. II-C still holds, although it is far looser since the average reward is lower; even the MCTS strategy cannot reach an average reward over 0.44. However, we remark that this is the average reward for each agent, and the total reward should be multiplied by 4; the total reward is far higher than the theoretical limit for a single drone, but adding other drones gives diminishing returns. Fig. 8 shows the reward in the test phase: as above, the difference between the reward obtained by MCTS and by the learners is minimal.

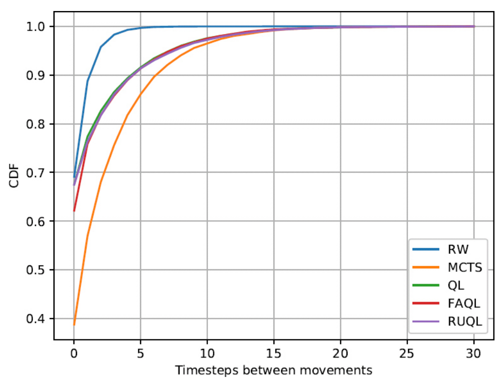

Fig. 9: CDF of the time between movements in the test phase in the multi-agent scenario.

The difference looks a bit larger in Fig. 9: while the rewards are similar, MCTS still stays on the same spot for longer times. This corresponds to longer charge cycles and drone flights, even though the performance of the system is similar in terms of the quality of the environment map.

IV. Conclusions and Future Work

In this work, we modeled a drone environmental surveying problem as an MDP, and tested three reinforcement learning strategies in both the single-agent and multi-agent scenarios. Our results show that reinforcement learning techniques still fall short of the theoretical optimum, but they provide a good approximation of explicit planning strategies while reducing computational demands by two orders of magnitude.

Interestingly, the gap between explicit planning strategies and learning decreases in the multi-agent scenario: this shows the algorithms' reliability even in complex multi-agent scenarios with minimal signaling between drones, and paves the way towards real implementation of survey swarms for several applications.

Future avenues of research include the implementation and testing of new learning strategies such as Deep Q-learning, which has already proven itself to be efficient in networking problems, and the refinement of the MDP model, as well as the implementation of more complex scenarios and tasks for the drones.

References

[1] F. Nex and F. Remondino, "UAV for 3D mapping applications: a review," *Applied geomatics*, vol. 6, no. 1, pp. 1–15, Mar. 2014.

[2] J. Martínez-de Dios, L. Merino, F. Caballero, A. Ollero, and D. Viegas, "Experimental results of automatic fire detection and monitoring with UAVs," *Forest Ecology and Management*, vol. 234, p. S232, Nov. 2006.

[3] A. Eltner and D. Schneider, "Analysis of different methods for 3D reconstruction of natural surfaces from parallel-axes UAV images," *The Photogrammetric Record*, vol. 30, no. 151, pp. 279–299, Sep. 2015.

[4] K. Douterloigne, S. Gautama, and W. Philips, "On the accuracy of 3D landscapes from UAV image data," in *International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, Jul. 2010, pp. 589–592.

[5] D. G. Lowe, "Object recognition from local scale-invariant features," in *7th International Conference on Computer Vision*, vol. 2. IEEE, Sep. 1999, pp. 1150–1157.

[6] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment–a modern synthesis," in *International Workshop on Vision Algorithms*. Springer, Sep. 1999, pp. 298–372.

[7] M. Pizzoli, C. Forster, and D. Scaramuzza, "REMODE: Probabilistic, monocular dense reconstruction in real time," in *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2014, pp. 2609–2616.

[8] C. Forster, M. Pizzoli, and D. Scaramuzza, "SVO: Fast semi-direct monocular visual odometry," in *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2014, pp. 15–22.

[9] J. H. Kim, J.-W. Kwon, and J. Seo, "Multi-UAV-based stereo vision system without GPS for ground obstacle mapping to assist path planning of UGV," *Electronics Letters*, vol. 50, no. 20, pp. 1431–1432, Sep. 2014.

[10] R. Schurig, T. Dsesquelles, A. Dumont, E. Lefranc, and A. Lux, "3D stereoscopic measurements from AR drone squadron," in *International Conference on Control Science and Systems Engineering (CCSSE)*. IEEE, Dec. 2014, pp. 23–26.

[11] J. E. Banta, Y. Zhien, X. Z. Wang, G. Zhang, M. Smith, and M. A. Abidi, "Best-next-view algorithm for three-dimensional scene reconstruction using range images," in *Intelligent Robots and Computer Vision XIV: Algorithms, Techniques, Active Vision, and Materials Handling*, vol. 2588. International Society for Optics and Photonics, Oct. 1995, pp. 418–430.

[12] S. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza, "An information gain formulation for active volumetric 3D reconstruction," in *International Conference on Robotics and Automation (ICRA)*. IEEE, May 2016, pp. 3477–3484.

[13] E. Palazzolo and C. Stachniss, "Effective exploration for MAVs based on the expected information gain," *MDPI Drones*, vol. 2, no. 1, p. 9, Mar. 2018.

[14] J. I. Vasquez-Gomez, L. E. Sucar, R. Murrieta-Cid, and E. Lopez-Damian, "Volumetric next-best-view planning for 3D object reconstruction with positioning error," *International Journal of Advanced Robotic Systems*, vol. 11, no. 10, p. 159, Oct. 2014.

[15] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998.

[16] D. H. Wolpert, K. R. Wheeler, and K. Tumer, "General principles of learning-based multi-agent systems," in *3rd Annual Conference on Autonomous Agents*. ACM, Apr. 1999, pp. 77–83.

[17] L. Panait and S. Luke, "Cooperative multi-agent learning: The state of the art," *Autonomous agents and multi-agent systems*, vol. 11, no. 3, pp. 387–434, Nov. 2005.

[18] L. Busoniu, R. Babuska, and B. De Schutter, "A comprehensive survey of multiagent reinforcement learning," *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.

[19] M. Chen and G. Rincon-Mora, "Accurate electrical battery model capable of predicting runtime and IV performance," *IEEE Transactions on Energy Conversion*, vol. 21, no. 2, pp. 504–511, June 2006. [Online]. Available: http://ieeexplore.ieee.org/document/1634598/

[20] T. Dietrich, S. Krug, and A. Zimmermann, "An empirical study on generic multicopter energy consumption profiles," in *Annual International Systems Conference (SysCon)*. IEEE, Apr. 2017, pp. 1–6.

[21] K. Goss, R. Musmeci, and S. Silvestri, "Realistic models for characterizing the performance of Unmanned Aerial Vehicles," in *26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, July 2017, pp. 1–9.

[22] O. Tremblay, L.-A. Dessaint, and A.-I. Dekkiche, "A generic battery model for the dynamic simulation of hybrid electric vehicles," in *Vehicle Power and Propulsion Conference (VPPC)*. IEEE, Sep. 2007, pp. 284–289.

[23] B. Michini, T. Toksoz, J. Redding, M. Michini, J. How, M. Vavrina, and J. Vian, "Automated battery swap and recharge to enable persistent UAV missions," in *Infotech@ Aerospace 2011*. AIAA, Mar. 2011, p. 1405.

[24] F. P. Kemper, K. A. O. Suzuki, and J. R. Morrison, "UAV consumable replenishment: Design concepts for automated service stations," *Journal of Intelligent & Robotic Systems*, vol. 61, no. 1, pp. 369–397, Jan. 2011.

[25] N. K. Ure, G. Chowdhary, T. Toksoz, J. P. How, M. A. Vavrina, and J. Vian, "An automated battery management system to enable persistent missions with multiple aerial vehicles," vol. 20, no. 1, pp. 275–286, Feb. 2015.

[26] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, May 1992.

[27] M. Kaisers and K. Tuyls, "Frequency adjusted multi-agent Q-learning," in *9th International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, May 2010, pp. 309–316.

[28] S. Abdallah and M. Kaisers, "Addressing environment non-stationarity by repeating Q-learning updates," vol. 17, no. 1, pp. 1582–1612, Jan. 2016.

[29] R. Bellman, "A Markovian decision process," *Journal of Mathematics and Mechanics*, pp. 679–684, Jan. 1957.

[30] R. Bjarnason, A. Fern, and P. Tadepalli, "Lower bounding Klondike solitaire with Monte-Carlo planning." in *19th International Conference on Automated Planning and Scheduling (ICAPS)*. AAAI, Jan. 2009.