Decentralized Multi-Agent Deep Reinforcement Learning in Swarms of Drones for Flood Monitoring

1st David Baldazo
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
d.baldazo@alumnos.upm.es

2nd Juan Parras
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
j.parras@upm.es

3rd Santiago Zazo
Information Processing and Telecommunications Center
Universidad Politécnica de Madrid
Madrid, Spain
santiago.zazo@upm.es

***Abstract*—Multi-Agent Deep Reinforcement Learning is becoming a promising approach to the problem of coordination of swarms of drones in dynamic systems. In particular, the use of autonomous aircraft for flood monitoring is now regarded as an economically viable option and it can benefit from this kind of automation: swarms of unmanned aerial vehicles could autonomously generate nearly real-time inundation maps that could improve relief work planning. In this work, we study the use of Deep Q-Networks (DQN) as the optimization strategy for the trajectory planning that is required for monitoring floods, we train agents over simulated floods in procedurally generated terrain and demonstrate good performance with two different reward schemes.**

***Index Terms*—navigation, reinforcement learning, swarms, decentralized control, floods**

I. Introduction

Floods caused an estimated 367 US$ billion of economic losses from 2007 to 2016 and led to more deaths than any other type of natural disaster in 2017 [1].

Nearly real-time inundation maps are a valuable tool when planning relief work. To provide them, both satellite and aerial, radar and optical images have been employed. In particular, the performance of aerial optical sensing has been proven by works such as [2], which shows accurate detection of floods from images captured by Unmanned Aerial Vehicles (UAVs).

Some approaches to navigation such as Simultaneous Localization and Mapping (SLAM) [3] require an explicit representation of the map. This, however, imposes some constraints. An alternative approach can be found in model-free Reinforcement Learning (RL) methods, which make use of implicit representations but rely on hand-crafted features and are limited to low dimensionality problems. The solution to these limitations came in the form of Deep Reinforcement Learning (DRL), which relies on deep neural networks to automatically extract features from raw input and provide the implicit representation [4].

This work was supported by a Ph.D. grant given to the second author by Universidad Politécnica de Madrid, as well as by the Spanish Ministry of Science and Innovation under the grant TEC2016-76038-C3-1-R (HERAKLES). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan V GPU used for this research.

DRL has been previously used for automatic wildfire monitoring from UAVs in [5] to reduce operational costs by eliminating the need for remote control by a human pilot. With the same objective in mind, in this article we follow a similar approach to solve the problem of automatic flood monitoring. The use of DRL allows for the development of an end-to-end controller that maps raw input data to aircraft commands in the form of bank angle corrections.

The development of a controller requires a flood simulation model. Landlab [6] provides an implementation of a stable and simple formulation of the shallow water equations for 2-D flood modeling by Almeida et al. [7]. This tool also provides us with a quick mechanism of generating the topography of the terrain by means of erosion of a random initial terrain.

Regarding solution methods, inspired by [5], we use Deep Q-Networks (DQN) and, on top of that, we explore the effect of reward sharing, which is an alternative reward scheme in swarm applications [8].

II. Problem Description

A. Environment

The environment consists of a 2D map representing a 1 km x 1 km area. It is discretized into 100 x 100 cells, each with an elevation and a water level.

1) Terrain generation: The terrain is randomly generated at the beginning of each episode of training. The elevation map is created in a two step process: first, a 100 x 100 Brownian surface is generated. This initial map is then eroded for 50 years using the FlowAccumulator and FastscapeEroder (Braun-Willett Fastscape) modules in Landlab while the terrain is uplifted. The resulting terrain typically has a maximum relief of 10-30 m.

2) Flood simulation: We focus on those floods which are the product of dam collapse. This allows for a controlled environment in which the optimal solution is particularly intuitive: the flooded area will move downhill from the dam and the aircraft should fly through this valley. Episodes are initialized by placing a 20 m x 20 m, 5 meter high body of water in the center of the map before letting the flood simulation run. During the episode, the terrain is fixed. The Almeida et al. model (OverlandFlow Landlab component) is run with Mannings roughness coefficient n = 0.01 s m$^{-1/3}$, a relatively low friction configuration, in order to simulate fast evolving floods.

B. Agents

The agents are fixed-wing aircraft. Aircraft velocity v is constant and equal to 20 m/s, each agent decides whether to increase or decrease its bank angle ϕ by 5 degrees at a frequency of 10 Hz and position is updated with the kinematic model described in the following expressions:

$$\dot{x} = v\cos\psi,\ \dot{y} = v\sin\psi,\ \dot{\psi} = \frac{g\ \tan\phi}{v}, \tag{1}$$

where g is standard gravity, 9.8 m s^{-2}. The maximum bank angle is set to 50 degrees and actions exceeding this limit have no effect. This establishes a realistic limit to angular velocity.

III. SWARMDP IMPLEMENTATION

In contrast to [5], in which a Partially Observable Markov Decision Process (POMDP) [9] is used as a model, we frame the problem as a SwarMDP [10], which is a particularization of the Decentralized POMDP (Dec-POMDP) model. In this way, we explicitly model the multi-agent setting and the fact that all agents are equal.

Definition 1. *A swarMDP is defined in two steps. First, we define a prototype $\mathbb{A} = \langle S, A, Z, \pi\rangle$, where $\mathbb{A}$ is an instance of each agent of the swarm and where:*

- *S is the set of local states.*
- *A is the set of local actions.*
- *Z is the set of local observations.*
- *$\pi : Z \to A$ is the local policy.*

A swarMDP is a 7-tuple $\langle N, \mathbb{A}, P, R, O, \gamma\rangle$ where:

- *N is the index set of the agents, where $i = 1, 2, ..., N$ indexes each of the N agents.*
- *$\mathbb{A}$ is the agent prototype defined before.*
- *$P : S \times S \times A \to [0, 1]$ is the transition probability function, where $P(s'|s, a)$ denotes the probability of transitioning to state s' given that the agent is in state s and plays action a. Note that P depends on a, the joint action vector.*
- *$R : S \times A \to \mathbb{R}$ is the reward function, where $R(s, a)$ denotes the reward that the agent receives when it is in state s and plays action a. Note that R depends on the joint action vector a.*
- *$O : S \times Z \times A \to [0, 1]$ is the observation model, where $O(z'|s, a)$ is the probability of observing z' given that the agent is in state s and plays action a. Note that O depends on the joint action and observation vectors a and z respectively.*
- *$\gamma \in [0, 1]$ is a discount factor, used to obtain the total reward for the agent.*

A. State

The local state of each agent has several dimensions:

- The surface water depth map $D_t(x, y)$, with 100 x 100 dimensions, updated every 10 seconds by the flood simulator.
- The position of the aircraft (x_i, y_i).
- The heading angle of the aircraft ψ_i.
- The bank angle of the aircraft ϕ_i.

B. Actions

The set of local actions is $\langle$ increase bank angle by 5 degrees, decrease bank angle by 5 degrees $\rangle$. Each agent has to choose one of them every 0.1 s.

C. Observations

Each agent collects two kinds of observations: a vector of features representing some information of the local state of the own aircraft and the other aircraft, and an image representing a partial observation of the flood from the perspective of the aircraft.

The vector of features of each agent i is itself the concatenation of four vectors of continuous variables: all the bank angles $\{\phi_j \mid j \in 1, 2, ..., N\} = \phi$, the range to other aircraft $\{\rho_{ij} \mid j \in 1, 2, ..., N \wedge\ j \neq i\} = \rho_i$, the relative heading angle to other aircraft $\{\theta_{ij} \mid j \in 1, 2, ..., N \wedge\ j \neq i\} = \theta_i$ and the relative heading angles of other aircraft $\{\psi_{ij} \mid j \in 1, 2, ..., N \wedge\ j \neq i\} = \psi_i$.

The images are assumed to be the processed output of an optical hemispherical camera with a view angle of 160 degrees always pointing downwards, resulting in a maximum range of 500 m. The final image is a 30 x 40 pixel representation of the flood.

To generate the image, the surface water depth map $D_t(x, y)$ is compared with a fixed water level threshold, here set to 5 cm, to simulate the inability of the image processing algorithm to detect very shallow waters. The resulting boolean map of detectable flooded areas $F_t(x, y)$ is then used to compute the observation of the aircraft by sampling at 40 azimut angles uniformly and each of these angles at 30 elevation angles uniformly from nadir to 10 degrees below the horizon. This results in a non-uniform sampling in range with more resolution right below the aircraft than at 500 m.

D. Rewards

The reward corresponding to each agent i consists of the sum of four distinct components:

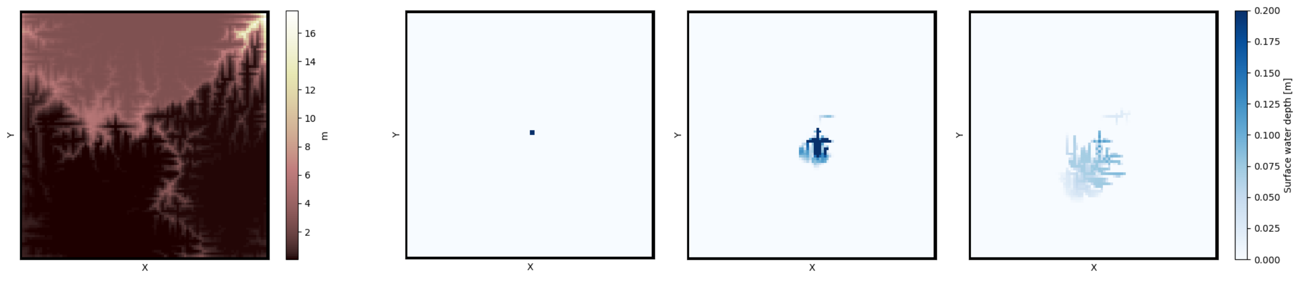
$$r_1 = -\ \lambda_1 \min_{\{s\in S|F_t(s)\}} d_s, \tag{2}$$

$$r_2 = -\ \lambda_2 \sum_{\{s\in S|d_s<r_0\}} 1 - F_t(s), \tag{3}$$

$$r_3 = -\ \lambda_3\phi_0^2, \tag{4}$$

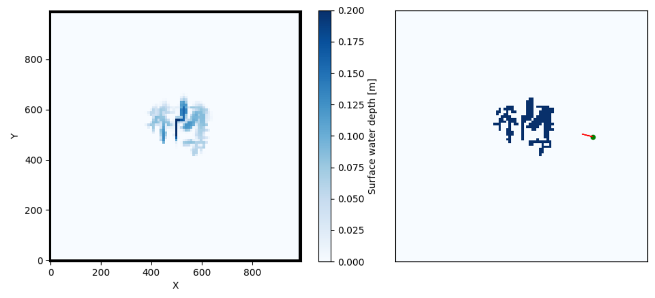
$$r_4 = -\sum_{\{j\in 1,2,...,N\wedge\ j\neq i\}} \lambda_4 \exp\Big(-\frac{\rho_{ij}}{c}\Big), \tag{5}$$

where d_s is the distance from the aircraft to cell s. These are penalties for:

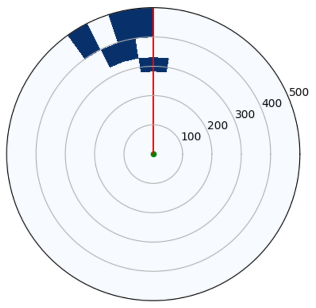


(a) Terrain. (b) Flood at t = 0 s. (c) Flood at t = 100 s. (d) Flood at t = 500 s.

Fig. 1. An example of random terrain generation and flood evolution over time.



(a) True flood. (b) Visible flood.



(c) Observation by agent.

Fig. 2. An example of the generation of observation images.

- Distance from flood (r_1): proportional to the distance to the closest flooded cell.
- Dry cells nearby (r_2): proportional to the number of dry cells in a radius r_0.
- High bank angles (r_3): proportional to the square of the bank angle.
- Closeness to other aircraft (r_4): sum of the contributions of each one of the other aircraft, which saturate to λ_4.

The tuning parameters have been set to the following values:

$$\lambda_1 = 10,\ \lambda_2 = 1,\ \lambda_3 = 100,\ \lambda_4 = 1000, \quad (6)$$

$$r_0 = 10,\ c = 100. \quad (7)$$

The reasoning behind the selection is that flying far away from the flood should be heavily penalized and, once over the flood, the penalizations from bank angle and closeness to other aircraft should become important.

IV. SOLUTION METHOD

A. Deep Q-Networks

The objective of the solution method is to find the policy π which maps observations to actions optimally. The optimality criterion is determined by the reward function R that we have defined. From it, a value $Q(s, a)$ can be deduced for each state-action pair. This value function represents the expected accumulated reward by an agent that takes action a in state s. The Q function for the optimal policy takes the values given by the Bellman equation:

$$Q(s, a) = r(s, a) + \gamma \sum_{s' \in S} p(s'|s, a) \max_{a' \in A} Q(s', a'). \quad (8)$$

For a given Q function, the optimal policy is the following:

$$\pi(s) = \operatorname*{argmax}_{a \in A} Q(s, a). \quad (9)$$

DQN [4], [11] estimates Q by training a deep neural network to approximate the function. Training is performed with the following loss function:

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}\left[e^2\right], \quad (10)$$

where the Bellman error e is:

$$e = r(s, a) + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a, \theta). \quad (11)$$

During training, the agent faces a trade-off between exploration of the action-state space through a random policy and exploitation of the knowledge that has already been acquired through the current estimation of the optimal policy. In practice this is tackled by using an exploratory policy for most of the decisions at the beginning of training and gradually increasing the fraction of exploitation actions.

DQN also tackles the problem of instability inherent to DRL, firstly by randomizing the samples used in training through a mechanism known as experience replay, in order to reduce the effect of the correlation of the sequence. This is implemented through a replay memory $\mathbb{E}$. Secondly, by training at the same time but at a lower update rate than the first network (with parameters θ) a different network, known as target network (with parameters θ^-), to estimate $Q(s', a')$, so that the first network is moving towards a fixed target. The complete algorithm is detailed in Algorithm 1.

Algorithm 1 Deep Q-Networks (DQN)

```
Input: Parameters: $\epsilon$, $L$, $N$
Output: $\pi$, the approximate optimal policy.
 1: Initialize replay memory $\mathbb{E}$ to capacity $L$
 2: Initialize $\theta = \theta^-$ randomly
 3: for each episode do
 4:     Initialize $s_0$
 5:     for each step $t$ in the episode do
 6:         Take action $a_t \sim \pi_\epsilon(\cdot|s)$ and observe $r_t$, $s_{t+1}$
 7:         Update memory replay $\mathbb{E}$ with current sample $e_t = (s_t, a_t, s_{t+1}, r_t)$
 8:         Sample randomly a set $\mathbb{N}$ of $N$ indexes from $\mathbb{E}$
 9:         for each experience sampled $e_t$ do
10:             Obtain $B_t$
11:         Update: $\theta \leftarrow \text{Adam}((B_t - Q(s_t, a_t; \theta))^2)$
12:     Update: $\theta^- \leftarrow \theta$
13: for all $s \in \mathcal{S}$ do
14:     $\pi(s) \leftarrow \arg\max_{a' \in \mathcal{A}} \text{Predict}((s, a'), \theta^-)$
15: return $\pi$, $\theta$
```

where $B_t = r_t + \gamma \max_{a'} Q(s_{t+1}, a_{t+1}; \theta^-)$ for non-terminal s_{t+1} and $B_t = r_t$ for terminal s_{t+1}.

B. Network architecture

The neural network treats the two types of observations separately. The features go through a series of dense layers with ReLU as the activation function. The image is instead processed by a series of convolutional layers and max pooling layers which reduce the dimensionality of the image in an efficient way before entering its own dense layers. The outputs of both networks are then concatenated and go through two additional dense layers. The complete architecture is shown in Fig. 3.

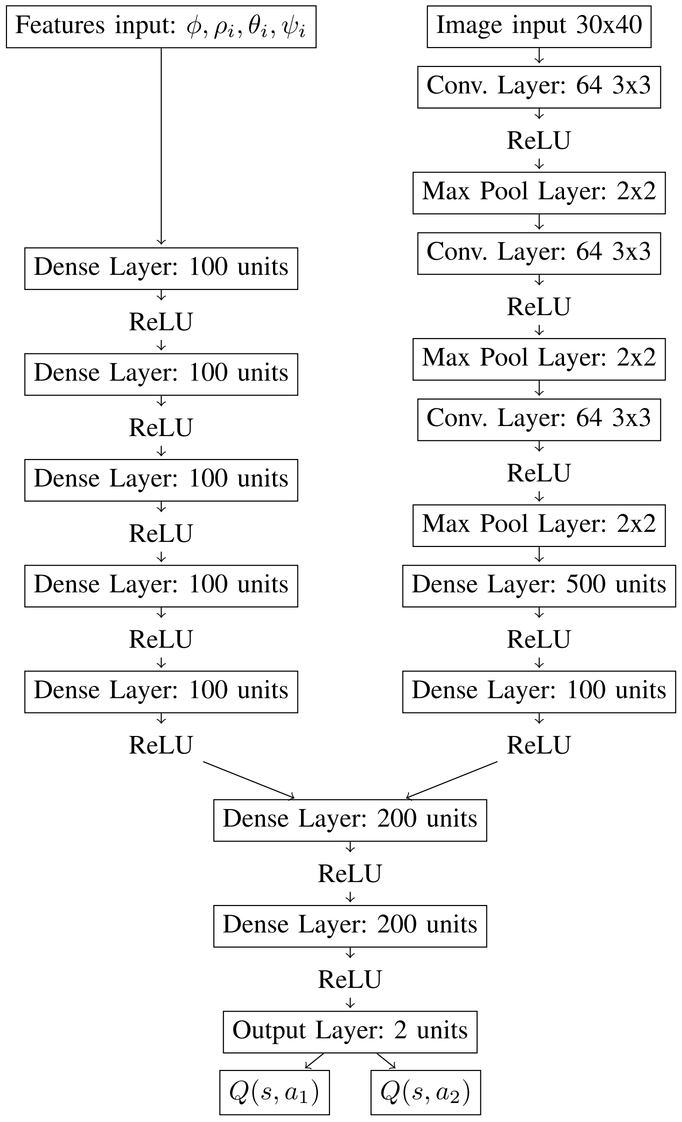


Fig. 3. Network architecture.

C. Reward configuration

We compare two reward schemes, which differ in the level of decentralization during training. Once trained, both approaches allow for decentralized control.

1) Independent rewards: Each agent receives its own reward as previously defined.

2) Shared reward: All agents share the same reward, consisting of the mean of the independent rewards.

V. SIMULATION SETUP

We test both independent and shared reward configurations with two agents. Training is performed every 100 steps, once 100 new samples have been collected. Each of the training steps uses a batch size of 2000 samples, randomly taken from a buffer of the last 100000 samples. The target network is updated every 1000 steps, that is, once for every 10 action-values update. Learning only starts after the first 2000 steps, so that the network does not overfit to the first samples. The fraction of exploratory actions starts at 1, decreases linearly during the first 70% of training time and stays at 0.1 until the end of training. The simulated episodes have a duration of 10000 steps (17 minutes). We use Adam [12] as the method for stochastic optimization and we set the learning rate to a value of $5 \cdot 10^{-4}$.

VI. SIMULATION RESULTS

The main performance metric is the episode accumulated reward, which is the sum over an episode of the instantaneous rewards of all agents. As depicted in Fig. 4, both reward configurations allow the agents to learn and reduce their penalties on average.

Fig. 5 shows a simulated episode as an example of the behaviour that is learnt by the agents. Aircraft learn to head towards flooded areas once they are detected and stay close to them, but are forced to avoid staying in the same area making small circles because high bank angles are penalized. Instead, the agents go through the flood while keeping a distance from other aircraft. When the area of interest is too small, the penalty to the proximity between aircraft outweighs the rest and one of the aircraft stays where it is making small circles.

Regarding the reward scheme, we expected a similar behaviour in both configurations for this particular multi-agent problem because three of the four penalty components (r_1, r_2 and r_3) are independent of the behaviour of the other aircraft, meaning that the margin of improvement is only related to the optimization of the relative positions of aircraft with respect to each other.

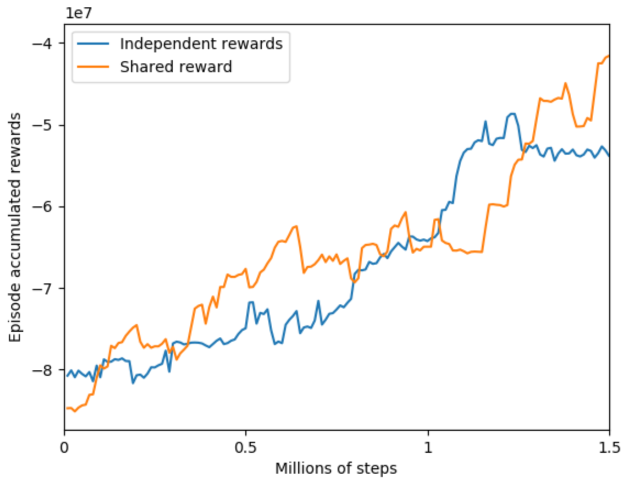


Fig. 4. Comparison of reward during training.

VII. CONCLUSIONS

In this paper we present a technique for training through DRL a deep neural network capable of guiding multiple fixed-wing aircraft to monitor floods in a decentralized fashion. Agents are able to take decisions from raw input data, consisting of a processed optical image and some information of the local state of the swarm. We tested two reward approaches, which differ in the degree of decentralization in training. Simulation results show that the aircraft are able to efficiently monitor floods in a coordinated fashion with both reward schemes. These controllers could be installed in real UAVs in order to reduce operational costs. We trained on simulated floods from dam collapses but the training approach should perform well on all kinds of floods as long as the training data is general enough.

REFERENCES

[1] CRED, “Natural disasters 2017. Brussels: CRED,” 2018 EM-DAT file dated 02/07/2018.

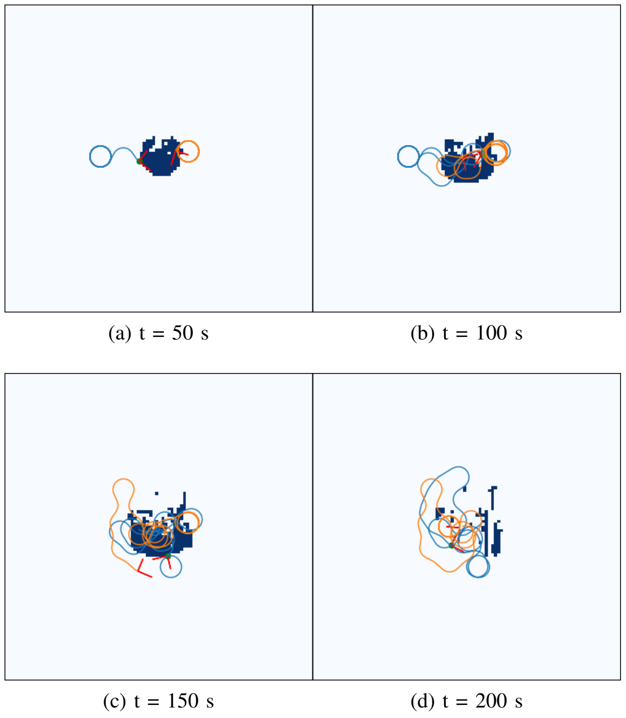


Fig. 5. An example of an episode with two aircraft.

[2] Q. Feng, J. Liu, and J. Gong, “Urban flood mapping based on unmanned aerial vehicle remote sensing and random forest classifierA case of Yuyao, China,” *Water*, vol. 7, no. 4, pp. 1437–1455, 2015.

[3] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, “Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age,” *IEEE Transactions on robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[4] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with deep reinforcement learning,” *arXiv preprint arXiv:1312.5602*, 2013.

[5] K. D. Julian and M. J. Kochenderfer, “Distributed wildfire surveillance with autonomous aircraft using deep reinforcement learning,” *Journal of Guidance, Control, and Dynamics*, pp. 1–11, 2019.

[6] D. E. J. Hobley, J. M. Adams, S. S. Nudurupati, E. W. H. Hutton, N. M. Gasparini, E. Istanbulluoglu, and G. E. Tucker, “Creative computing with Landlab: an open-source toolkit for building, coupling, and exploring two-dimensional numerical models of earth-surface dynamics,” *Earth Surface Dynamics*, vol. 5, no. 5, pp. 21–46, 2017.

[7] G. A. Almeida, P. Bates, J. E. Freer, and M. Souvignet, “Improving the stability of a simple formulation of the shallow water equations for 2-D flood modeling,” *Water Resources Research*, vol. 48, no. 5, 2012.

[8] M. Hüttenrauch, A. Šošić, and G. Neumann, “Deep reinforcement learning for swarm systems,” *Journal of Machine Learning Research*, vol. 20, no. 54, pp. 1–31, 2019. [Online]. Available: http://jmlr.org/papers/v20/18-476.html

[9] S. Thrun, W. Burgard, and D. Fox, *Probabilistic robotics*. MIT press, 2005.

[10] A. Šošić, W. R. KhudaBukhsh, A. M. Zoubir, and H. Koeppl, “Inverse reinforcement learning in swarm systems,” in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2017, pp. 1413–1421.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, “Human-level control through deep reinforcement learning,” *Nature*, vol. 518, no. 7540, p. 529, 2015.

[12] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.