Autonomous Single-Image Drone Exploration With Deep Reinforcement Learning and Mixed Reality

Alessandro Devo, Jeffrey Mao, *Graduate Student Member, IEEE*, Gabriele Costante, *Member, IEEE*, and Giuseppe Loianno, *Member, IEEE*

***Abstract*—Autonomous exploration is a longstanding goal of the robotics community. Aerial drone navigation has proven to be especially challenging. The stringent requirements on cost, weight, maneuverability, and power consumption do not allow exploration approaches to easily be employed or adapted to different types of environments. End-to-End Deep Reinforcement Learning (DRL) techniques based on Convolutional Networks approximators, which grant constant-time computation, predefined memory usage, and deliver high visual perception capabilities, represent a very promising alternative to current state of the art solutions relying on metric environment reconstruction. In this work, we address the autonomous exploration problem with aerial robots with a monocular camera based on DRL. Specifically, we propose a novel asymmetric actor-critic model for drone exploration that efficiently leverages ground truth information provided by the simulator environment to speed up learning and enhance final exploration performances. Furthermore, in order to reduce the sim-to-real gap for exploration, we present a novel mixed reality framework that allows an easier, smoother, and safer simulation to real-world transition. Both aspects allow to further exploit the great potential of simulation engines and contribute to reducing the risk associated with directly deploying algorithms on a physical platform with no intermediate step between the simulation and the real world. This is well-known to create several safety concerns and be dangerous when deploying aerial vehicles. Experimental results with a drone exploring multiple environments show the effectiveness of the proposed approach.**

***Index Terms*—Aerial Systems: Applications, reinforcement learning, deep learning for visual perception.**

I. Introduction

UNMANNED Aerial Vehicles (UAVs) often called drones are highly versatile robotic platforms with great potential. They can be used to address a wide range of tasks: from surveillance to monitoring of agricultural crops, from search and rescue operations to delivery of goods, from human assistance to cinematography. In many tasks, the drone can be controlled remotely or be supervised by an expert operator, able to intervene at any time. In other cases, however, it is desirable to have drones capable of completing their tasks autonomously, performing independent maneuvers.

The advances in the area of real-time end-to-end image processing, especially through the use of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have opened up new ways of addressing the autonomous decision-making process leveraging DRL [1]–[3]. There are numerous studies in which DRL techniques are being successfully applied in many different areas of robotics, such as visual navigation [4]–[6], active object tracking [7], and also robotic manipulation [8]. However, there are several challenges and safety concerns to both deploy and train DRL agents in the real world. For these reasons, prior works propose to train the model in simulation and adapt it in a second phase to real-world settings.

Despite the success that such an approach has shown, we believe that the advantages of training in a simulated environment are not yet fully exploited in the field of drone navigation. For example, most of the simulation environments [9], [10] for navigation-related tasks can provide ground truth information to the model, which is, however, not usually employed to enhance the training process, but only for reward calculation and environment management. The main advantages of this process are therefore increased training speed and easier training management (e.g., agent and scenario reset management, collision handling, reward assignment, etc..), rather than an improvement of the learning quality. Furthermore, models trained in simulation are usually directly deployed on a physical platform in a real setting, characterized by different motion dynamics and different visual appearances. Therefore, it is necessary that the control and perception components of the model are able to cover the sim-to-real gap simultaneously to perform the task successfully, avoiding possible dangerous consequences for the UAVs or the surrounding environment.

In this work, we propose a new single-image DRL method for autonomous UAV exploration, featuring an asymmetric [11] actor-critic architecture that fully exploits ground truth information from the simulation engine at training time. Considering that the critic component is only needed to evaluate the policy during training, there is no need to restrict its input to be the same as the one of the actor component. Instead, we feed it

Manuscript received September 9, 2021; accepted February 7, 2022. Date of publication February 24, 2022; date of current version March 8, 2022. This letter was recommended for publication by Associate Editor I. Sa and Editor P. Pounds upon evaluation of the reviewers' comments. This work was supported in part by the Technology Innovation Institute, in part by the NSF CPS under Grant CNS-2121391, in part by Qualcomm Research, Nokia, and in part by NYU Wireless. *(Corresponding author: Alessandro Devo.)*

Alessandro Devo is with the Department of Engineering, University of Perugia, 06125 Perugia, Italy, and also with the Tandon School of Engineering, New York University, Brooklyn, NY 11201 USA (e-mail: alessandro.devo@studenti.unipg.it).

Gabriele Costante is with the Department of Engineering, University of Perugia, 06125 Perugia, Italy (e-mail: gabriele.costante@unipg.it).

Jeffrey Mao and Giuseppe Loianno are with the Tandon School of Engineering, New York University, Brooklyn, NY 11201 USA (e-mail: jm7752@nyu.edu; loiannog@nyu.edu).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2022.3154019, provided by the authors.

Digital Object Identifier 10.1109/LRA.2022.3154019

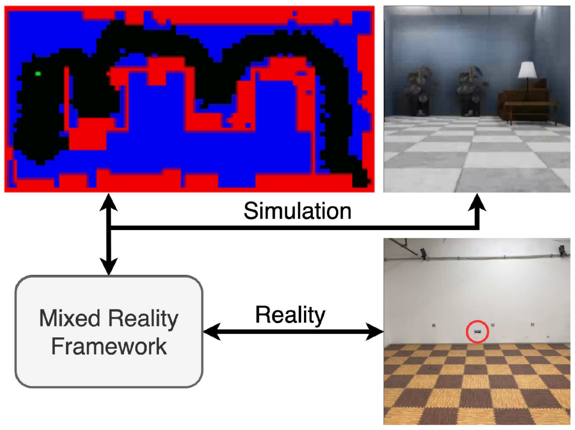


Fig. 1. In the top-left image, a grid map represents our agent (green dot) while exploring the simulated environment. The obstacles (walls and furniture) are colored in red, the area in blue indicates the undiscovered parts of the scenario, and, in black, the discovered cells. In the top-right image, a sample frame captured by the agent is shown. In the bottom picture, our laboratory setting is presented, where our model controls the physical robot (red circle) to perform exploration via our mixed reality framework, which manages the communication and coordination between the simulation and the real world.

with much more informative data, such as collision maps and position coordinates, thus providing it almost complete environment state information. In this way, the critic can obtain a much more precise and deeper understanding of the problem and can better evaluate the actor policy, which can indirectly benefit from ground truth information. Furthermore, to ensure a smoother, safer, and easier real-world deployment of our method, we introduce a novel mixed reality framework (see Fig. 1), by which the model can control a physical drone, while concurrently receiving its feedback data. Similar to [12], our framework allows to combine real-time sensors measurements generated in simulation (*e.g.*: images, laser readings, LIDAR scans, etc...) with the physical robotic platform dynamics and position measurements provided by real-world sensors (onboard or external). Differently from [12], which allows to scan real-world elements and render them in simulation (feature we plan to implement in the future), the synthetic environments provided by our framework are completely generated by a photorealistic graphic engine (as we explain in Section IV-A, that is necessary for procedural environment generation) and it allows to employ several control layers (position, velocity, attitude, and angular rate, or motor space) according to the user needs. Therefore, we believe that the framework can reduce the sim-to-real gap in two key ways. First, it represents a natural intermediate step between simulation to real environments that allows testing the control and perception components of the model separately. Second, it allows to evaluate the method performance in arbitrarily complex environments tailored for exploration tasks, not available within laboratory settings, while still flying a physical UAV. Finally, it also helps to verify the compatibility of control signal commands with the available robot framework.

This article proceeds as follows. Section II presents our literature review; Section III describes the asymmetric model architecture and training procedure; Section IV introduces the environment setup for training and the proposed mixed reality framework; Section V shows the experiments and the results; finally, Section VI draws the conclusions.

II. Related Works

Classic approaches to autonomous navigation include map-based [13], SLAM [14], [15], and SfM [16], [17] methods. While the first one assumes the availability of a map of the surroundings, hence, limiting their applicability to known environments, SLAM methods compute the map on the fly. However, many of them rely upon sensors, such as laser, LIDAR, or Kinect, that can be too expensive or heavy to be used on aerial drones. Other works [14], instead, make use of RGB cameras, like also SfM algorithms [16], [17], which, however, still need a good amount of computational resources to process the incoming data, update the map, estimate a path, and perform control maneuvers, in real-time. Other approaches include information-theoretic [18], [19] and frontier-based [20], [21] strategies, for which similar arguments apply since they typically rely upon other methods to generate maps or require extracting particular environment information while processing sensor data. These methods also need a large amount of information. These aspects contribute to increase their computational and memory requirements depending also on the scenario complexity and, hence, limiting their generalization capabilities. *Bug algorithms* are another category of exploration methods that, unlike those just described, are more computationally efficient, and therefore more suitable to be employed on aerial platforms. In [22], this technique was used to train a swarm of small drones to cover an unknown environment, by moving along the border of walls and obstacles, and then return to the starting point. This strategy is optimal in the case of environments with connected walls, but is more challenging to exploit in more complex environments that do not respect that condition. Furthermore, such algorithms also uses sensors, i.e., lasers, which are typically more expensive than a simple camera.

More recent approaches focus on end-to-end deep learning solutions based on CNNs and RNNs. In [23], the authors present a model trained by using Imitation Learning (IL) on a dataset of trajectories ended in crashes. In [24] the authors propose a modular architecture for drone racing trained to imitate an expert policy. The two main problems with supervised learning methods is that they require a substantial amount of labeled data, and they often overfit the training data distribution. As a consequence, during testing, if the model substantially deviates from the experts' policy it was trained on, it can lead to very poor performances and catastrophic failures. Conversely, DRL approaches are notoriously more challenging to train but are able to achieve higher final performances and better generalization. For these reasons, DRL techniques are becoming increasingly popular in many robotics areas. [25] presents a closed-loop predictive control for soft robotic manipulators, trained with DRL. In [8], the authors trained a DRL model to perform complex dexterous in-hand manipulation. Conversely, the work in [5] introduces a novel framework and a modular architecture for target-driven visual navigation whereas in [7], a novel continuous control approach for visual active tracking is described. [26] proposes a combination of IL and RL to train an autonomous agent for visual navigation, however, still relying upon maps and position data. Many works employ DRL algorithms specifically for autonomous drone navigation. In [27], an image-based visual servoing controller for aerial robots is presented. In [28], NAVREN-RL is introduced as a new approach for UAV navigation in real indoor settings. In [29], a vision-based DRL approach for coverage path planning with aerial robots is presented. [1] describes a DRL-based model for UAV platforms that takes into account stationary, as well as moving elements, to achieve collision-free indoor navigation. In [30], the authors address the problem of collision-free indoor flight in real scenarios, by proposing a DRL-based method trained exclusively with 3-D CAD models, but do not focus on the exploration problem. Compared to most state-of-the-art vision-based exploration approaches, we do not require an environment map or any additional localization data at testing stage. Furthermore, we better exploit the benefits of simulation environments for enhanced training speed performance quality, and efficiency and sim-to-real adaptation.

III. Approach

In this section, we first describe our task and reframe it as a classical RL problem. Subsequently, we carefully detail our model architecture and illustrate the learning procedure.

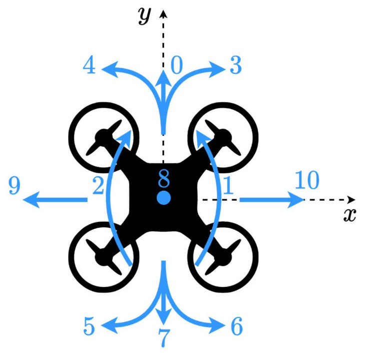
A. Problem Formulation

The goal of an exploration algorithm is to discover the surrounding environment in the most efficient way possible (in our case, in terms of time units). Such situation can be formally defined as a classical RL setting [3], in which an agent repeatedly interacts with an environment E by performing actions a_i, in order to collect observations o_i and rewards r_i. In many cases, the observation o_i coincides with the underlying state of the environment, usually referred to as s_i in the literature. In the case of an exploration problem, such a state can be fully represented, for example, by the environment map, the position and orientation of the agent, and the area explored. However, since our model observations are represented by just raw RGB images, which do not contain all the necessary information to fully describe the true hidden state, the environment is only partially observable by the agent, and the problem can be framed as a Partially Observable Markov Decision Process (POMDP). Therefore, the objective of the agent can be mathematically solved by finding a policy π^* that maximizes the sum of the discounted future reward

$$R_t = \sum_{i=t}^{T} \gamma^{i-t} r_i\left(o_i, a_i\right), \tag{1}$$

where $\gamma \in [0, 1)$ is the discount factor and $r_i(o_i, a_i)$ is the reward at time i, given the observation o_i and the action $a_i \sim \pi(\cdot|o_i)$. Such a solution can be achieved by evaluating the agent current policy π

$$V_\pi(o) = \mathbb{E}_\pi\left[\sum_{i=0}^{\infty} \gamma^i R_{t+i+1} \middle| o\right], \tag{2}$$



0 -	*move_forwards*
1 -	*turn_left*
2 -	*turn_right*
3 -	*turn_right_move_forward*
4 -	*turn_left_move_forward*
5 -	*turn_right_move_backward*
6 -	*turn_left_move_backward*
7 -	*move_backward*
8 -	*do_nothing*
9 -	*move_left*
10 -	*move_right*

Fig. 2. The UAV reference system and action space.

where $\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given π. In the following, we describe how this formulation is adapted to our particular scenario.

B. Task Details

The agent objective is to explore an unknown complex environment relying exclusively upon raw RGB frames as observations (o_t). In our settings, such an agent is represented by a quadcopter, which is free to move across the *X* and *Y* axes of a three-dimensional space. In particular, the policy (π) that it can develop comprehend 11 actions, all discrete, and within the set depicted in Fig. 2. At the start of the episode, the agent is spawned randomly in the environment and explores it until the end of the episode (*i.e.*, after a predefined number of steps).

To incentivize the agent to explore the surroundings, we design the following reward signal

$$r = \frac{\max\left(M_e\right)}{1 + \sum_{i=0}^{m} \lceil M_{e_i} \rceil}, \tag{3}$$

where m is the number of elements of M_e and M_e is a 2D matrix representing the map entropy, whose values are

$$M_{e_i} = \begin{cases} \frac{1}{\sum_{j=0}^{m} \lceil M_{e_j} \rceil}, & \text{if } M_{e_i} \text{is unexplored} \\ 0, & \text{otherwise} \end{cases}. \tag{4}$$

This reward encourages exploration and is also proportional to the number of visited cells. This is reasonable since the more cells are explored, the more complex it is to find unvisited cells, and, hence, a greater reward is earned.

C. Network Architecture

The previous section describes the classic RL framework, which, however, cannot be used, as it is, for our problem. Since the agent observations are represented by RGB images, two major issues arise: i) visual frames are only a partial observation of the hidden underlining state, and ii) the number of all the possible combinations of the image pixels forms an extremely large observation space. For these reasons, simple tables, containing the values of all states and actions, cannot be used, and more advanced approximation methods and DRL algorithms are needed. A widespread solution for visual navigation-related

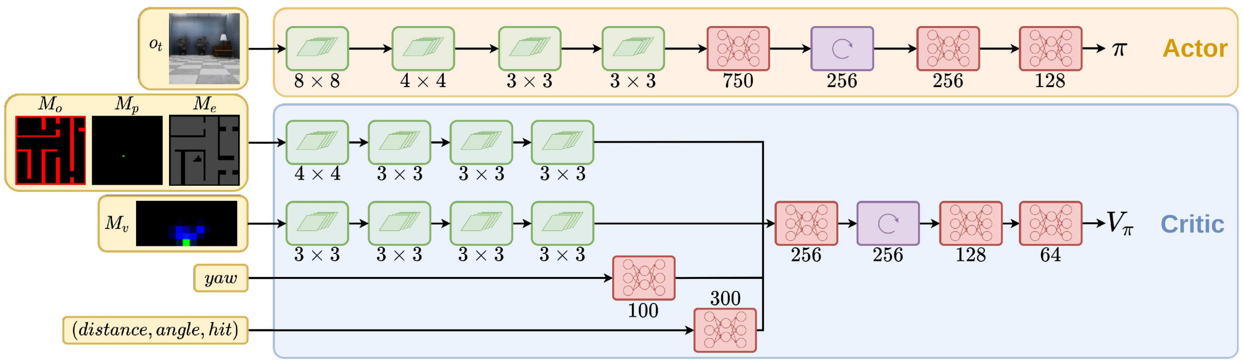


Fig. 3. The proposed architecture. The actor component (orange box) is fed with the current RGB frame. The critic component (blue box) is not used during testing and receives ground truth information that provides more insight about the underlying true system state than a monocular RGB image.

problem is to employ Deep Neural Network (DNN) approximators, specifically CNNs, for image feature extraction, and RNNs, for memory management, trained using a DRL *actor-critic* based method (such as IMPALA [31], which we discuss in Section III-D).

Fig. 3 shows the proposed model architecture for our exploration task. Many works [32], [33] implement both actor and critic in a one single network to reduce the total number of parameters and take advantage of the weight sharing between the policy and the value function. Conversely, as observed in Fig. 3, we employ two different asymmetric neural networks [11] with completely separate parameters. The main advantage of this solution is the possibility to feed the actor and critic components with completely different inputs. This is particularly valuable if using synthetic environments since only the actor needs to be used at test time (and, therefore, is required to be fed with RGB images only), the critic can make use of any ground information available from the simulation. Feeding the critic with the underlying environment state, for example, helps it to better evaluate the actor behaviour, resulting in a more robust policy and higher final performance (as we prove in Section V).

The actor component takes as input an 84×84 RGB frame that is processed by 4 convolutional layers with 16, 16, 32, 32 channels, 8, 4, 3, 3 kernel size, and 4, 2, 1, 1 stride, respectively. Each of those has a *ReLU* activation, followed by a *GroupNorm* layer. The features extracted are fed to a fully connected layer, with 750 neurons, then to a Gated Recurrent Unit (GRU) [34], with 256 hidden units, and, finally, to two additional fully connected layers, with 256 and 128 neurons, respectively. Every layer, except for the last one, is followed by a *ReLU* activation. The final output vector is passed through a softmax layer to produce 11 probabilities, *i.e.*, the policy π, or, in other words, the likelihood of choosing each one of the available actions.

Conversely, the critic is fed with different inputs so that it can leverage information not available at test time, such as:

- Obstacle Map (M_o): a 35×35 2D grid, where each cell is set to 1 if occupied by an obstacle, 0 otherwise;
- Agent Position Map (M_p): it is a 35×35 2D grid, where each cell is set to 1 if occupied by the agent, 0 otherwise;
- Entropy Map (M_e): a 35×35 2D grid that encodes the uncertainty of the environment, as described in Eq. 4;
- Visibility Map (M_v): a $21 \times 11 \times 3$ 3D grid that represents an egocentric view description of the agent field of view;
- Yaw (yaw): it is the agent heading;
- Distance, angle, and hit ($(distance, angle, hit)$): these three values represent the agent distance from the origin point, the angle between the agent and the x-axis, and if the agent hit an obstacle with its last action, respectively.

M_o, M_p, and M_e are firstly stacked to obtain a $35 \times 35 \times 3$ 3D grid, and then processed by 4 convolutional layers with 16, 16, 32, 32 channels, $4, 3, 3, 3$ kernel size, and $2, 2, 1, 1$ stride, respectively. Each layer is followed by a *ReLu* activation and by a *GroupNorm* layer. M_v is fed into 4 convolutional layers with $16, 16, 32, 32$ channels, $3, 3, 3, 3$ kernel size, and $2, 1, 1, 1$ stride, respectively. They are all followed by a *ReLu* activation and by a *GroupNorm* layer. The yaw is fed to a fully connected layer with 100 neurons and *ReLu* activation. The extracted features are concatenated together and further elaborated by a fully connected layer, a GRU network, both with 256 hidden nodes, and by two fully connected layers, with 128 and 64 neurons, all with *ReLu* activation. The final output is the value function V_π.

D. Training Algorithm

We train our agent by using the Importance Weighted Actor-Learner Architecture (IMPALA) [31]. We create several copies of our agent and place them in as many independent simulated environments. The agent copies interact with their corresponding environments, collecting different trajectories composed of observations, actions, and rewards. These are first collected in a common *replay buffer* [35], and then retrieved to train the model. The actual learning process takes place through the use of a central shared model, which computes the three standard IMPALA losses. The first one fits the V-trace targets v_t for the critic

$$l_v = \frac{1}{2}\left(v_t - V_{\theta'}\left(o_t\right)\right)^2, \tag{5}$$

where $V_{\theta'}(o_t)$ is the estimated value, parameterized by θ', for observation o_t. The second one concerns the policy π

$$l_p = \rho_t \log \pi_{\theta''}\left(a_t|o_t\right)\left(r_t + \gamma v_{t+1} - V_{\theta'}\left(o_t\right)\right), \tag{6}$$

where ρ_t is one of the truncated importance sampling weights. It should be noticed that, given the asymmetric nature of our

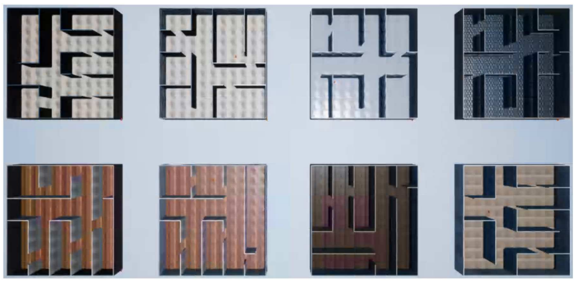


Fig. 4. Top-down view of our simulated training setting. There are 8 different environment instances where each agent independently collects trajectories.

actor-critic model, the parameters θ'' of the policy $\pi_{\theta''}$ and the parameters θ' of the value function $V_{\theta'}$ are necessarily different. The third loss promotes entropy in action selection, and it is employed to help exploration

$$l_c = -\sum_a \pi_{\theta''}(a|o_t) \log \pi_{\theta''}(a|o_t). \tag{7}$$

As it processes the data, it updates its weights to match the designed reward signal and asynchronously updates all the copies, which continue to gather new trajectories in parallel. For a more detailed description of the IMPALA losses and the V-trace algorithm, see [31].

IV. Training and Test Setup

In this section, we first describe the training scenarios, and then we detail the proposed mixed reality framework.

A. Training Scenarios

Training an RL agent directly using aerial robots is very challenging and can also be potentially dangerous. Therefore, we design a simulated environment (see Fig. 4) using the photo-realistic graphics engine Unreal Engine 4 (UE4).[1] The environment consists of a large empty floor organized in rooms and corridors, in which the agent is always spawned randomly. One of our main objectives is to achieve generalization to more complex and photo-realistic settings, thus we apply *domain randomization* [36] to our synthetic scenario, continuously varying several environmental attributes. In particular, when an episode ends, the wall structure, lightning conditions, and all texture patterns are randomly changed.

B. Mixed Reality Framework

To have an easier and smoother simulation to real-world transition, we develop a novel mixed reality framework, which we use for our model deployment into a physical quadrotor. The framework's role is to manage the communication between the model, the simulation, and the robot. In particular, it has to translate the model action for the low-level robot controller and transfer the physical robot trajectory back into the simulation. It should be also highlighted that the framework is completely platform-independent, which means that can be employed with every kind of robot, aerial or not.

[1]https://www.unrealengine.com

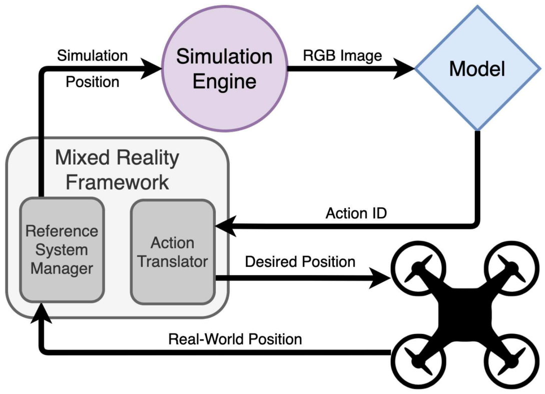


Fig. 5. Our mixed reality diagram.

In Fig. 5, we show a diagram of our mixed reality framework and how it interacts with the other involved entities. The simulation engine provides all the information the model requires, in our case, RGB images. In our setup, the model resides in the same machine that runs the simulator, so the communication is straightforward, however, this is not a necessary condition and can easily be changed according to the user needs. Once the model receives the inputs, it produces the action to be performed according to its policy. This is not send back to the simulation but it is first translated by the framework and sent to the robot. The component referred to as *Action Translator* converts the action ID (see Fig. 2) to a corresponding robot waypoint that is understandable by the robot's planner. For example, if the current drone position in (x, y, z) coordinate is $(1, 0, 1)$ (expressed in metres), the current orientation is 0, *w.r.t.* the x-axis, the predefined speed is 10 cm/step, and the action to perform is 0, then the respective waypoint generated by the *Action Translator* is $(1.1, 0, 1)$. The robot's planner is responsible to convert the waypoint into line trajectories imposing the robot a linear path from its current position to the final one. A desired velocity and acceleration is chosen by the user to determine how fast the trajectory traverses this linear path and accelerates from the starting location or decelerates toward the goal. The line trajectory is then executed through a nonlinear controller [37] responsible to send low-level control such as the robot thrust and moments based on the desired trajectory and feedback from the robot position.

Once the trajectory is executed, the physical platform processes the measurements provided by the real-world sensors (like the drone onboard sensors, or other external systems) and used them to estimate the current position, which is then sent back to the simulator. The framework cannot use the desired position directly but it always requires feedback from the real system, since, due to actuator noise, controller accuracy, measurement precision, or other disturbances, the actual robot and desired positions can differ. Finally, the real position is processed by the *Reference System Manager*, which converts it to the simulation coordinate system. For example, if the simulated environment is twice as large (in the x and y axes) as the real one,

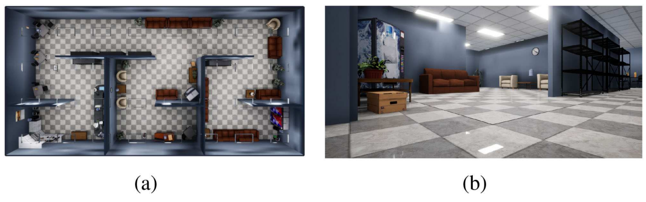


Fig. 6. Example of *Realistic Environment* from two different perspectives: (a) top-down view, (b) frontal view.

and their origin points are $(10, 10, 0)$ and $(0, 0, 0)$, respectively, then a real-world position of $(1, 1, 1)$ corresponds to $(12, 12, 1)$ in simulation. The simulated environment can be scaled to match the size of the real settings, however, its complexity (*i.e.*: environment structure, number of rooms and corridors, furniture, obstacles, etc...) must be compatible with the available physical test area, the drone size, and the measurements accuracy.

Finally, the simulated agent position is updated accordingly and the process is repeated.

V. Experiments

In this section, we describe the implementation details of our approach and we introduce the metrics and the baselines used to validate our model performances. Finally, we present the experimental setups and discuss the results.

A. Implementation details

We train our model by employing 8 IMPALA [31] agents in parallel, for 1500 episodes, each composed of 1800 steps. As optimizer, we use RMSprop with a learning rate of 0.0002, a batch size of 24, and a sequence length of 50. The discount factor γ is set to 0.99 and the shape of the matrix M_e is 35×35. When an episode starts, a new environment is randomly generated and an agent spawned in it. The goal of the agent is to explore the environment as fast as possible before the maximum number of steps per episode is reached. The training and the experiments run using a workstation with 2×NVIDIA GTX 2080Ti with 11GB of VRAM, an Intel Core processor i7-9800X (3.80 GHz×16), and 64GB of DDR4 RAM. The aerial robot is a quadrotor based on our previous work [37].

B. Test Environments, Baselines and Metrics

We evaluate our model performances, in 3 scenarios:

- *Standard*: the type used during training (see Fig. 4);
- *Large*: similar to the *Standard* environment, but it spans four times its area;
- *Realistic*: this is the environment that employs the proposed mixed reality approach. It is a photorealistic environment that resembles a real-world office, with numerous objects, details, and furniture (see Fig. 6). By using the mixed reality framework, which manages the communication between the simulator and a physical drone (see Section IV-B), it is possible to perform real-world tests in such complex scenarios in safer conditions, and without real-world space limitations.

TABLE I
Average Coverage Across Environments

	Ours	**Ours (M)**	**Ours (MV)**	**Baseline**	**Sym.**	**Random**
Standard	**58.2**%	54.8%	2.9%	27.8%	26.9%	6.4%
Large	39.3%	**42.3**%	0.8%	27%	19.3%	4.2%
Real. #1	**76.3**%	16.4%	2.8%	27.4%	9.2%	3.6%
Real. #2	**61.1**%	39%	2.2%	30%	7.8%	3%
Real. #3	**69.7**%	4.2%	2.8%	31.3%	4.5%	5.3%
Real. #4	**63**%	20.2%	2.7%	31.3%	6.6%	2.9%
Real. #5	**57.6**%	39.5%	2.8%	21.9%	8.7%	5.9%
Real. #6	**76.6**%	20.5%	2.9%	25.4%	13.1%	5.1%

We test our proposed model against 5 baselines:

- **Random**: a uniformly distributed random policy;
- **Symmetric**: symmetric variant of our approach, in which the critic component is exactly equal to the actor, and it is also fed with the same input, *i.e.*, RGB images;
- **Nav**: a state-of-the-art baseline model, whose architecture has been used in different navigation tasks [5], [6], [33] (with some small variations between implementations). Our particular settings, it features a first convolutional layer with 16 8×8 filters with stride 4, a second one with 32 4×4 filters with stride 2, a fully-connected and a GRU layer, both with 256 neurons and ReLU activation. For a fair comparison, our implementation shares the same discrete action set of the others methods (see Fig. 2);
- **Ours (MP)**: a variant of our approach, in which the critic component is not fed with the global maps and the $(distance, angle, hit)$ vector;
- **Ours (M)**: a variant of our approach, in which the critic component is not fed with the global maps.

In order to evaluate the exploration capabilities of the models, we consider the coverage percentage as a metric.

C. Standard and Large Environments Results

Both environments are procedurally generated with randomized textures, lighting conditions, and floor structure. Each run is composed of an episode with a duration of 1800 and 7200 steps, for the *Standard* and *Large* environments, respectively. All the scores are calculated by averaging 20 runs in 20 different floors for each of the two scenarios.

The first two rows of Table I summarizes the results in terms of the coverage percentage at the end of the episode. As it can be noted, the proposed approach (**Ours**) reaches the highest final coverage for both kinds of environments. Specifically, compared against **Random**, **Symmetric**, and **Nav** our method performance is dramatically superior, proving that the proposed asymmetric approach can take advantage of the ground truth information provided by the simulator to learn a much more effective exploration policy, compared to the one produced by a standard symmetric architecture. Furthermore, by looking at Fig. 7(a)-7(b), which shows the coverage percentage as a function of the number of steps elapsed, it is clear that our method performance varies greatly between different environments. Careful analysis of our model behaviour shows that this is mainly related to the walls and floor textures, which seem to affect greatly the final performance.

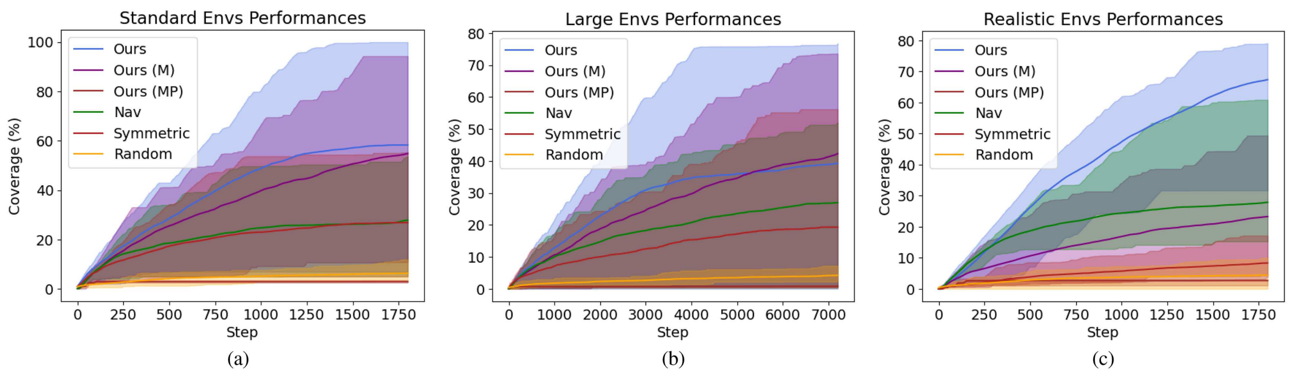


Fig. 7. Models' performance in terms of percentage of coverage as a function of the number of steps elapsed, in the 3 types of environments considered: (a) *Standard*, (b) *Large*, and (c) *Realistic*. The colored lines represent the models' average performances (**Ours** in blue, **Ours (M)** in purple, **Ours (MP)** in brown, **Nav** in green, **Symmetric** in red, and **Random** in yellow.), while the corresponding faded areas define the range between the worst and best scores across all the experiments.

Indeed, with some types of texture, the coverage can even reach a value very close to 100%. Conversely, with others, the model fails to perform the task correctly (see Fig. 7(a)-7(b)). This behaviour can also be noticed for the *Large* environment, for which a similar argument applies. The results in this second environment also demonstrate that our model can scale to much larger environments compared to those in which it is trained on.

To evaluate the role and importance of the critic inputs during training phase, we compare **Ours** also against **Ours (M)** and **Ours (MP)**. As can be observed, **Ours (M)**, whose critic has been trained without global maps, can achieve quite similar performance to **Ours**, suggesting that, for these kinds of scenarios, such inputs do not impact final performance particularly. On the contrary, **Ours (MP)** shows extremely poor performances in both settings, demonstrating that the sole Field of View (FoV) map and agent yaw are not at all sufficient to allow the critic to learn anything useful.

D. Realistic Environment Results

Compared to the other types of scenarios, the *Realistic* environment cannot be generated procedurally, since it is characterized by the presence of a large variety of furniture objects that cannot be randomly spawned. These need to be carefully placed in order for the environment to appear like a real-world office. For this reason, we manually design 6 different floors, each one with a different structure and objects disposition. For each scenario, we perform 5 different runs, with 1800 steps each for each tested approach.

The other main difference between the *Realistic* and the other kinds of environments is the use of the proposed mixed reality framework. The experiments are conducted in an indoor testbed with a flying space of $10 \times 5 \times 4\text{m}^3$ of the Agile Robotics and Perception Lab (ARPL) lab at New York University. In Fig. 8, we can see UAV flying in our flying arena. Real-world position data is collected using an 8 camera Vicon[2] motion capture system running at 100Hz. The Vicon data is used as the real-world position data feedback to our *Reference System Manager* and

[2]https://www.vicon.com/

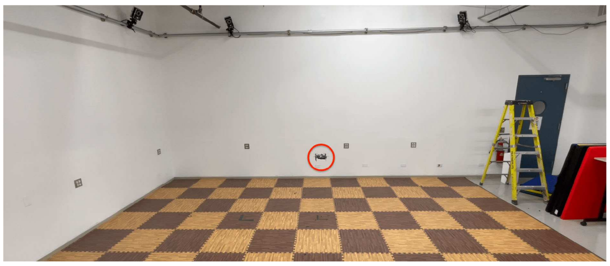


Fig. 8. Real-world setup for the mixed reality framework. The drone (red circle) is free to move over a large carpet.

controller from Section IV-B. The data can also be obtained using onboard sensing trough visual inertial odometry as shown in [37], which, however, does not offer the same level of accuracy as the Vicon system. The quadrotor system sets as 0.15m/s and 0.1m/s^2 the velocity and acceleration parameters respectively.

The results achieved by the tested approaches are reported in Fig. 7(c) and in the last six rows of Table I. Differently from the results obtained in other environments, our model does not only substantially outperform the other methods, but also shows much more consistent performances across different runs. As already discussed in Section V-C, our model fluctuating performance is mainly caused by particular textures, which, however, are not present in the *Realistic* scenario. As a consequence, the coverage percentage remains high and stable in all the considered floors, as can also be noticed in Fig. 7(c). Interestingly, even **Ours (M)** performs quite poorly, suggesting that the global maps somehow helps the development of better generalization capabilities.

These results prove that our model is capable to generalize to environments that differ significantly in terms of visual appearance (like the simulated training and realistic environments) and that the proposed mixed reality framework successfully manages the simulation and real-world synchronization. It should also be highlighted that without such a framework, it would have been very challenging to test the models on a physical platform in such complex environments without compromising user and environment safety or accuracy of the measurements.

VI. Conclusion

In this work, we presented a new End-to-End approach for autonomous exploration with aerial vehicles. Its asymmetric architecture enabled the proposed approach to exploit the benefits of training in simulation, by leveraging the highly informative ground truth data provided by the simulator itself. The results show that this feature allows our model to achieve higher performances compared to its symmetric counterpart. Finally, it manages to successfully explore larger environments compared to those used for training, and, by leveraging the proposed mixed reality framework, also to be capable to handle complex photorealistic scenarios, while controlling a physical robotic platform. This enables a smooth transition to real-world settings, which is particularly important due to safety requirements when deploying drones in the real world. Future works will consider the deployment of the algorithm on-board the robot using direct feedback from the drone camera. Finally, we aim to investigate the collaborative exploration problem with multiple robots to speed up the overall task efficiency and enhance its resilience.

References

[1] A. Singla, S. Padakandla, and S. Bhatnagar, "Memory-based deep reinforcement learning for obstacle avoidance in UAV with limited environment knowledge," *IEEE Trans. Intell. Transp. Syst.*, vol. 22, no. 1, pp. 107–118, Jan. 2021.

[2] N. Passalis and A. Tefas, "Continuous drone control using deep reinforcement learning for frontal view person shooting," *Neural Comput. Appl.*, vol. 32, no. 9, pp. 4227–4238, 2020.

[3] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT press, 2018.

[4] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017, pp. 3357–3364.

[5] A. Devo, G. Mezzetti, G. Costante, M. L. Fravolini, and P. Valigi, "Towards generalization in target-driven visual navigation by using deep reinforcement learning," *IEEE Trans. Robot.*, vol. 36, no. 5, pp. 1546–1561, Oct. 2020.

[6] A. Devo, G. Costante, and P. Valigi, "Deep reinforcement learning for instruction following visual navigation in 3D maze-like environments," *IEEE Robot. Automat. Lett.*, vol. 5, no. 2, pp. 1175–1182, Apr. 2020.

[7] A. Devo, A. Dionigi, and G. Costante, "Enhancing continuous control of mobile robots for end-to-end visual active tracking," *Robot. Auton. Syst.*, vol. 142, 2021, Art. no. 103799.

[8] O. M. Andrychowicz *et al.*, "Learning dexterous in-hand manipulation," *Int. J. Robot. Res.*, vol. 39, no. 1, pp. 3–20, 2020.

[9] A. Chang *et al.*, "Matterport3D: Learning from RGB-D data in indoor environments," in *Proc. Int. Conf. 3D Vis.*, 2017, pp. 667–676, *arXiv:1709.06158*.

[10] F. Xia, A. R. Zamir, Z. He, A. Sax, J. Malik, and S. Savarese, "Gibson Env: Real-world perception for embodied agents," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9068–9079.

[11] L. Pinto, M. Andrychowicz, P. Welinder, W. Zaremba, and P. Abbeel, "Asymmetric actor critic for image-based robot learning," 2017, *arXiv:1710.06542*.

[12] W. Guerra, E. Tal, V. Murali, G. Ryou, and S. Karaman, "FlightGoggles: Photorealistic sensor simulation for perception-driven robotics using photogrammetry and virtual reality," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2019, pp. 6941–6948.

[13] J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst., Man, Cybern.*, vol. 19, no. 5, pp. 1179–1187, Sep./Oct. 1989.

[14] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, Oct. 2017.

[15] M. W. M. G. Dissanayake, P. Newman, S. Clark, H. F. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (SLAM) problem," *IEEE Trans. Robot. Automat.*, vol. 17, no. 3, pp. 229–241, Jun. 2001.

[16] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer, "A survey of structure from motion," *Acta Numerica*, vol. 26, pp. 305–364, 2017.

[17] J. L. Schonberger and J.-M. Frahm, "Structure-from-motion revisited," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 4104–4113.

[18] S. Bai, J. Wang, F. Chen, and B. Englot, "Information-theoretic exploration with Bayesian optimization," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1816–1822.

[19] K. Saulnier, N. Atanasov, G. J. Pappas, and V. Kumar, "Information theoretic active exploration in signed distance fields," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 4080–4085.

[20] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proc. IEEE Int. Symp. Comput. Intell. Robot. Automat.*, 1997, pp. 146–151.

[21] L. Wang *et al.*, "A collaborative aerial-ground robotic system for fast exploration," in *Proc. Int. Symp. Exp. Robot.*, 2018, pp. 59–71.

[22] K. McGuire, C. D. Wagter, K. Tuyls, H. Kappen, and G. C. de Croon, "Minimal navigation solution for a swarm of tiny flying robots to explore an unknown environment," *Sci. Robot.*, vol. 4, no. 35, 2019, Art. no. eaaw9710.

[23] D. Gandhi, L. Pinto, and A. Gupta, "Learning to fly by crashing," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 3948–3955.

[24] T. Wang and D. E. Chang, "Robust navigation for racing drones based on imitation learning and modularization," in *Proc. IEEE Int. Conf. Robot. Automat.* 2021, *arXiv:2105.12923*.

[25] T. G. Thuruthel, E. Falotico, F. Renda, and C. Laschi, "Model-based reinforcement learning for closed-loop dynamic control of soft robotic manipulators," *IEEE Trans. Robot.*, vol. 35, no. 1, pp. 124–134, Feb. 2019.

[26] T. Chen, S. Gupta, and A. Gupta, "Learning exploration policies for navigation," 2019, *arXiv:1903.01959*.

[27] C. Sampedro, A. Rodriguez-Ramos, I. Gil, L. Mejias, and P. Campoy, "Image-based visual servoing controller for multirotor aerial robots using deep reinforcement learning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 979–986.

[28] M. A. Anwar and A. Raychowdhury, "NavREn-Rl: Learning to fly in real environment via end-to-end deep reinforcement learning using monocular images," in *Proc. 25th Int. Conf. Mechatronics Mach. Vis. Pract.*, 2018, pp. 1–6.

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

[30] F. Sadeghi and S. Levine, "CAD2RL: Real single-image flight without a single real image," 2016, *arXiv:1611.04201*.

[31] L. Espeholt *et al.*, "IMPALA: Scalable distributed deep-RL with importance weighted actor-learner architectures," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1406–1415.

[32] M. Jaderberg *et al.*, "Reinforcement learning with unsupervised auxiliary tasks," 2016, *arXiv:1611.05397*.

[33] P. Mirowski *et al.*, "Learning to navigate in complex environments," 2016, *arXiv:1611.03673*.

[34] K. Cho, B. V. Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," 2014, *arXiv:1409.1259*.

[35] Z. Wang *et al.*, "Sample efficient actor-critic with experience replay," 2016, *arXiv:1611.01224*.

[36] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2017, pp. 23–30.

[37] G. Loianno, C. Brunner, G. McGrath, and V. Kumar, "Estimation, control, and planning for aggressive flight with a small quadrotor with a single camera and IMU," *IEEE Robot. Automat. Lett.*, vol. 2, no. 2, pp. 404–411, Apr. 2017.