DeepIG: Multi-Robot Information Gathering with Deep Reinforcement Learning

Alberto Viseras* and Ricardo Garcia†

***Abstract*—State-of-the-art multi-robot information gathering (MR-IG) algorithms often rely on a model that describes the structure of the information of interest to drive the robots motion. This causes MR-IG algorithms to fail when they are applied to new IG tasks, as existing models cannot describe the information of interest. Therefore, we propose in this paper a MR-IG algorithm that can be applied to new IG tasks with little algorithmic changes. To this end, we introduce DeepIG: a MR-IG algorithm that uses Deep Reinforcement Learning to allow robots to learn how to gather information. Nevertheless, there are IG tasks for which accurate models have been derived. Therefore, we extend DeepIG to exploit existing models for such IG tasks. This algorithm we term it model-based DeepIG (MB-DeepIG). First, we evaluate DeepIG in simulations, and in an indoor experiment with three quadcopters that autonomously map an unknown terrain profile built in our lab. Results demonstrate that DeepIG can be applied to different IG tasks without algorithmic changes, and that it is robust to measurement noise. Then, we benchmark MB-DeepIG against state-of-the-art information-driven Gaussian-processes-based IG algorithms. Results demonstrate that MB-DeepIG outperforms the considered benchmarks.**

***Index Terms*—Multi-Robot Systems; Deep Learning in Robotics and Automation**

I. Introduction

INFORMATION gathering (IG) with autonomous mobile robots has emerged as a prime alternative to gather information in situations that have a high risk for humans, like e.g. in search and rescue missions, and for applications in which it is desirable to reduce required time and manpower, like e.g. in environmental analysis. In such context, IG can clearly benefit from multi-robot coordination both in terms of efficiency to gather information and robustness against robotic failures.

Information gathering with multiple robots (MR-IG) has been studied for a wide range of applications such as surveillance, tracking, monitoring, to name only a few. In particular, in this paper we concentrate on MR-IG to monitor a physical process of interest like e.g. temperature, magnetic field, terrain profile, ozone concentration, etc. Note that in the remainder of the paper we refer to "monitor a physical process of interest with multiple robots" as MR-IG.

Manuscript received: February, 22, 2019; Revised April, 18, 2019; Accepted June, 10, 2019.

This paper was recommended for publication by Editor Nak Young Chong upon evaluation of the Associate Editor and Reviewers' comments.

*A. Viseras is with the Institute of Communications and Navigation of the German Aerospace Center (DLR), 82234, Oberpfaffenhofen, Germany alberto.viserasruiz@dlr.de

†R. Garcia is with the E.T.S.I. Telecomunicación at Universidad Politécnica de Madrid (UPM), 28040, Madrid, Spain ricardo.gpinel@alumnos.upm.es

Digital Object Identifier (DOI): see top of this page.

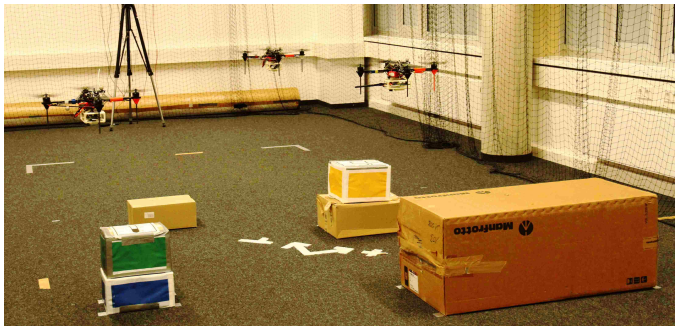


Fig. 1: Three quadcopters cooperating to map an unknown terrain profile that we built in our lab. Quadcopters run an instance of DeepIG, which uses deep reinforcement learning to teach robots how to gather information efficiently, while avoiding inter-robot collisions.

State-of-the-art MR-IG algorithms belong to the family of model-based algorithms. Most widely used classes of algorithms are Gaussian processes (GPs) [1], [2], and Partially Observable Markov Decision Processes (POMDPs) [3]. Model-based algorithms assume an underlying model that describes physical properties of the process, such as spatial and temporal correlation, states' transition function, etc. Given a model, IG algorithms exploit it to derive MR coordination strategies that allow robots to gather information by optimizing some formal IG criterion.

Model-based approaches are designed to exploit properties of a particular model. This has the advantage of achieving a high performance in applications where the model accurately describes the observed process. In contrast, model-based approaches fail to gather information of processes that cannot be accurately described by existing models.

In a future robotics society, robots will need to solve new IG tasks, and, of course, many of them will not be described by existing models. This implies that we humans will have to invest our time and efforts to develop novel models and corresponding IG algorithms. Additionally, many of the new IG tasks will be too complex to be described by traditional models like aforementioned GPs or POMDPs. Nevertheless, we would like to be able to offer, rapidly and with limited effort, adequate algorithms to solve most of the MR-IG tasks that will be demanded by a future robotics society.

As we previously stated, model-based approaches will most likely fail to offer solutions to some of the future IG tasks. In contrast, reinforcement learning (RL) [4] seems like a perfect fit to solve complex IG tasks. RL, in contrast to model-based

approaches, does not make any assumption on the process. This has the advantage that computers can derive IG strategies, regardless of their complexity, with little human effort. Recently, RL has been exploited to solve a wide spectrum of robotic tasks, including control of a quadcopter [5], robot navigation [6], [7], or multi-robot collision avoidance [8]. However, to the best of our knowledge, there is no algorithm in the literature that solves a MR-IG task using RL.

This gap in the state of the art motivates us to investigate the use of RL to monitor a physical process of interest with multiple robots. RL comprises multiple techniques to learn a mapping between robots' observations and robots' actions. In particular, here we opted for the use of Deep RL, which uses a deep neural network to implement this mapping.

Since the conception of Deep Q-Networks (DQN) algorithm [9], Deep RL has emerged as a powerful technique to handle complex sequential decision-making problems. Deep RL merges the capabilities of deep neural networks, which are able to process high-dimensional inputs and to make powerful representations, with the already successful, but limited to simpler problems, mathematical framework of RL. Deep RL has led to important breakthroughs such as learning to play Atari video games [9], or surveying wildfires with autonomous aircrafts [10].

Deep RL impressive breakthroughs motivate us to use it for MR-IG tasks. In fact, this corresponds to the essential contribution of our work: formulation of a MR-IG monitoring task as a Deep RL problem. This contribution comes together with two additional sub-contributions that we propose in this paper. These are (i) the definition of a reward function that allows robots to gather information while avoiding inter-robot collisions, and (ii) the extension of a state-of-the-art Deep RL algorithm, Asynchronous Advantage Actor-Critic (A3C) [11], which permits learning with multiple robots.

The aforementioned contributions are the pillars of the algorithm we propose to solve MR-IG tasks. This algorithm we term it DeepIG. DeepIG is designed for tasks for which a model of the information of interest is too complicated to be derived by a human. Nevertheless, it is true that there are information distributions for which very accurate models have been derived. For example GPs have shown an outstanding performance for some IG tasks [1], [2], [12]. In order to exploit existing models, we propose in this paper an extension of DeepIG that permits incorporating existing models like e.g. GPs. This extension of DeepIG we term it model-based DeepIG (MB-DeepIG).

Next we state our problem formally in Sec. II. This is followed by a detailed description of DeepIG and MB-DeepIG in Sec. IV. Then we evaluate in Sec. V DeepIG and MB-DeepIG in simulations. To finalize, we conclude this paper with the results of an experiment we carried out where three quadcopters equipped with an ultrasound sensor use DeepIG to map an unknown terrain profile.

II. Problem Statement

We define a discrete space $\mathcal{D} = \mathcal{C} \times \mathcal{R}$, with $\mathcal{C} = \{c_1, c_2, ..., c_{n_c}\} \subset \mathbb{R}^{n_c}$, $\mathcal{R} = \{r_1, r_2, ..., r_{n_r}\} \subset \mathbb{R}^{n_r}$, and $|c_{i+1} - c_i| = |r_{j+1} - r_j| = \delta$ for $i \in [1, n_c - 1]$, $j \in [1, n_r - 1]$. Let us denote a discrete time-invariant physical process $\mathcal{P} = \{p_1, p_2, ..., p_{n_c \cdot n_r}\} \subset \mathbb{R}^{n_c \cdot n_r}$ that takes place in $\mathcal{D}$, with p_k, $k = 1, ..., n_c n_r$, the magnitude of $\mathcal{P}$ at $[c_i, r_j] \in \mathcal{D}$. For simplicity, we assume that time is discrete and finite, and is denoted by $\mathbf{t} = [t_1, t_2, ..., t_{n_t}]$, with n_t the total number of IG time steps.

We consider a multi-robot system $\mathcal{M} = \{1, 2, ..., N\}$ composed by N robots. Robots motion is given by a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} \equiv \mathcal{D}$ denotes robots possible locations, and $\mathcal{E} = \{\mathcal{E}_1, \mathcal{E}_2, ..., \mathcal{E}_{n_e}\}$ denotes a set of n_e edges. Edges link two vertices iff a robot can traverse between vertices in a single time step. In this paper we limit robots movements to four possible actions $\mathcal{A}^m = \{\uparrow, \leftarrow, \downarrow, \rightarrow\}$ for $m = 1, ..., N$. We denote the joint action space of all robots as $\mathcal{A} = \mathcal{A}^1 \times \mathcal{A}^2 \times ... \times \mathcal{A}^N$. Actions considered in this paper yield edges $\mathcal{E}_l = \{\mathbf{d}_j, \mathbf{d}_k\} \in \mathcal{E}$ for $\mathbf{d}_j, \mathbf{d}_k \in \mathcal{D}$ and $|\mathbf{d}_k - \mathbf{d}_j| = \delta$.

Robot $m \in \mathcal{M}$ position at $t \in \mathbf{t}$ is denoted as $\mathbf{x}_t^m \in \mathcal{V}$. Each robot is equipped with a sensor that allows it to measure process value $p \in \mathcal{P}$ at position $\mathbf{x}_t^m$. We denote the measurement taken by robot $m \in \mathcal{M}$ at $t \in \mathbf{t}$ as z_t^m. Here we assume that uncertainty in robots positions and sensor measurements is negligible. In addition, we assume that robots can communicate all-to-all through an ideal communication channel.

Discrete space $\mathcal{D}$, process $\mathcal{P}$, robots $\mathcal{M}$, and robots positions $\mathbf{x}_t^m$ and robots measurements z_t^m, for $m = 1, ..., N$, constitute an environment. We denote the environment's state space as $\mathcal{S}$ with $s_t \in \mathcal{S}$ the environment's state at $t \in \mathbf{t}$.

Initially, we assume that all robots have an initial estimation $\tilde{\mathcal{P}} = \{\phi, \phi, ..., \phi\} \subset \mathbb{R}^{n_c \cdot n_r}$ of $\mathcal{P}$, where ϕ is a hyperparameter of the algorithm. Our goal in this paper is to devise an algorithm that allows robots to learn how to gather information. In particular, we aim robots to learn how to reduce the normalized root mean squared error (NRMSE) between $\mathcal{P}$ and $\tilde{\mathcal{P}}$ as fast as possible. Moreover, our algorithm should learn how to avoid inter-robot collisions. Here, we assume that two robots $m, n \in \mathcal{M}$ collide iff $|\mathbf{x}_t^m - \mathbf{x}_t^n| \leq \eta$, where η denotes an inter-robot safety distance.

III. Deep Reinforcement Learning

In this work, we propose the use of Deep RL to learn how to perform MR-IG tasks. A MR-IG task involves multiple robots interacting within the same environment to accomplish a common goal. This corresponds to a Multi-Agent RL (MARL) problem [13].

A MARL problem is modeled as a Markov Game (MG), and it is defined as a tuple $(N, \mathcal{S}, \mathcal{A}, T, \{R^m\}, \gamma)$. Function $T : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ defines a transition probability. Function $R^m : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ is agent's m reward, for $m = 1, .., N$. Parameter $\gamma \in (0, 1]$ is inherent to MARL, which we explain in next paragraph.

Given a formal definition of MARL, we explain next the dynamics behind it at any $t \in \mathbf{t}$. Let us imagine an environment, which has a state $s_t \in \mathcal{S}$. Each agent $m = 1, ..., N$ follows a policy $\pi^m : \mathcal{S} \times \mathcal{A}^m \rightarrow [0, 1]$. This yields a probability of choosing joint action $\mathbf{a}_t = [a_t^1, ..., a_t^N] \in \mathcal{A}$. Action $\mathbf{a}_t$ causes

a transition from s_t to s_{t+1}, with $s_t, s_{t+1} \in \mathcal{S}$. As a result of this transition agents receive a scalar reward $r^m_{t+1} \in \mathbb{R}$, for $m = 1, ..., N$. It is through the interactions between agents and environment how agents learn to accomplish a particular goal. In MARL, the goal of each agent is defined in terms of maximizing its own expected discounted return $G^m_t = \sum_{i=0}^{\infty} \gamma^i r^m_{t+i}$ from each $s_t \in \mathcal{S}$ under joint policy $\pi = [\pi^1, ..., \pi^N]$, where γ is a discount factor.

Previous definition of MARL assumes an ideal world where agents (robots) have perfect knowledge about $\mathcal{S}$. In practice, this is not true as robots perceive $\mathcal{S}$ via imperfect sensors like e.g. cameras, ultrasound sensors, lasers, etc. This implies that MARL is not a MG in practice. Instead, our MR-IG task corresponds to a Decentralized POMDP (Dec-POMDP). A Dec-POMDP is now defined by tuple $(N, \mathcal{S}, \mathcal{A}, T, \{R^m\}, O, \{\mathcal{O}^m\}, \gamma)$. Function $O : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{O}^m$ defines an observation probability, with $\mathcal{O}^m$ the set of possible observations of agent m, for $m = 1, ..., N$. Note that $\mathbf{o}^m_t \in \mathcal{O}^m$ is a partial representation of s_t, as it is perceived by sensors. Additionally, we would like to remark that, although MARL is modeled as a MG, it is possible to extend existing MARL techniques to solve a Dec-POMDP [9], [14].

There exists multiple training approaches to solve MARL problems [13]. Here we employ *centralized parameter sharing learning*. This implies that all agents use a unique common policy and, subsequently, a deep neural network with shared parameters. This approach allows agents to share experiences to more effectively train a common policy. In addition, *parameter sharing learning* permits performing the agent's control in a decentralized manner, as each agent can utilize its own instance of the policy. Besides, as we describe in Sec. IV, parameter sharing allows us to extend a state-of-the-art actor critic Deep RL algorithm – A3C [11]. A3C has been established as a successful framework for actual Deep RL applications because of its efficiency thanks to training parallelization. As all agents share the same deep neural network parameters, we can adapt A3C, which original conception was single-agent RL, to a multi-robot scenario. This extension we name it MR-A3C, and is part of DeepIG, which we explain next.

IV. DeepIG Algorithm

A. Overview

DeepIG is a MR-IG algorithm. In particular, each individual robot runs in parallel an identical instance of DeepIG, and inter-robot coordination is done by means of inter-robot communication. In Fig. 2 we depict a block diagram of DeepIG. Next let us explain DeepIG in more detail.

Robots interact with the outside world through three modules: Sensors Module, Communications Module, and Actuators Module. Actuators translate a robot's planned action into a robot's movement. Sensors and communication (S&C) modules allow robots to obtain information about the world in which robots operate. In particular, at each $t \in \mathbf{t}$, S&C modules provide each robot $m \in \mathcal{M}$ with $\mathbf{x}^m_t \in \mathcal{D}, z^m_t \in \mathcal{P}$, and with $\mathbf{x}^{\bar{m}}_t \in \mathcal{D}, z^{\bar{m}}_t \in \mathcal{P}$ from all $\bar{m} \in \mathcal{M}$ with $\bar{m} \neq m$.

Information collected with S&C modules is saved in an Information Storage module, and later processed by a Reward

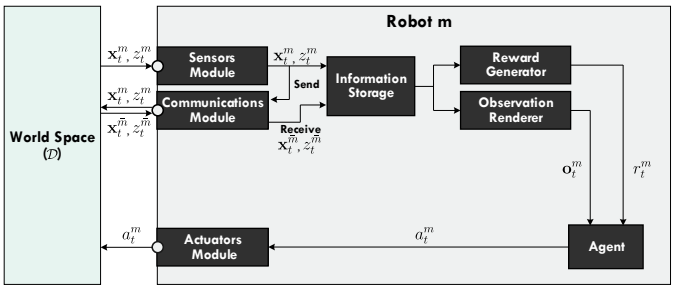


Fig. 2: Block diagram of DeepIG algorithm for an individual robot. Inter-robot cooperation is carried out through a Communications Module.

Generator and an Information Renderer. On the one hand, Reward Generator calculates a reward r^m_t, which allows the Agent to evaluate the positive/negative impact of its previous action. Our definition of reward allows us to incorporate a model of the information of interest, which is the basis of MB-DeepIG. For more details on the reward, we refer the reader to Sec. IV-B. On the other hand, Information Renderer maps the output of the Information Storage into an Observation $\mathbf{o}^m_t$. In this paper, we represent $\mathbf{o}^m_t$ as an image, which is a powerful and compact container of information. We refer the reader to Sec. IV-C for a detailed description of the Information Renderer.

Our Agent takes as inputs r^m_t, $\mathbf{o}^m_t$. The Agent is the robot's brain, and it is the module that calculates next robot's action a^m_t. Initially, we assume the Agent has no knowledge about how to cooperate with robots to gather information. Therefore we need to train the Agent. Of course, we could train robots in the real world. However, this is impractical as Deep RL algorithms typically require thousands/millions of executions to solve complex tasks like e.g. our MR-IG task. To accelerate the learning procedure, we propose MR-A3C in Sec. IV-D. Once we complete the training of the Agent, we can transfer robots to the real world to solve actual MR-IG tasks.

Given this overview of DeepIG, we explain next the essential components of DeepIG: Reward Generator, Observation Renderer, and Agent.

B. Reward Generator

We propose a reward that fulfills our two main objectives: (i) efficient MR-IG, and (ii) effective inter-robot collision avoidance. Note that each robot $m \in \mathcal{M}$ runs an instance of DeepIG. This implies that rewards are generated for a particular robot.

As we stated in Sec. II, our goal is to efficiently reduce the NRMSE between process $\mathcal{P}$ and process estimate $\tilde{\mathcal{P}}$. The NRMSE at time t is given by $e_t = \theta \sqrt{\frac{\sum_{i=1}^{n_c n_r} (\tilde{p}^i_t - p_i)^2}{n_c n_r}}$, with $\theta \in \mathbb{R}_{>0}$ a constant that normalizes e_t so that $e_t \in [0, 1]$, $p_i \in \mathcal{P}$ the physical process magnitude ground truth, and $\tilde{p}^i_t \in \tilde{P}$ the estimated physical process magnitude at time t. As our goal is to favor those actions that incur a high error reduction, we define the following IG reward:

$$r^m_t = |e_{t-1} - e_t|. \quad (1)$$

Note that $r_t^m \in [0,1],\ \forall t$. A bounded reward avoids arbitrarily sizes of error derivatives, which is crucial to guarantee convergence of the neural network parameters while sharing hyperparameters across different IG scenarios.

MB-DeepIG. Reward proposed in (1) allows us to extend DeepIG to incorporate a model of the process of interest. This extension of DeepIG we term it MB-DeepIG. MB-DeepIG exploits the process model to obtain a more accurate estimated process $\tilde{\mathcal{P}}$. In DeepIG, at time t, we only update estimated process $\tilde{\mathcal{P}}$ with $z_t^m \in \mathcal{P}$ taken at positions $\mathbf{x}_t^m \in \mathcal{D}$, for all $m = 1, ..., N$. In contrast, in MB-DeepIG, at time t, we can exploit a process model to update all $\tilde{p}_i \in \tilde{\mathcal{P}}$ for $i = 1, ..., n_c n_r$, given $z_{t_i}^m \in \mathcal{P}$ taken at positions $\mathbf{x}_{t_i}^m \in \mathcal{D}$, for all $m = 1, ..., N$ and $t_i = 1, 2, ..., t$.

MB-DeepIG exploits the model information to calculate a more informative reward. In addition, MB-DeepIG performs regression given the acquired measurements. This way robots do not need to measure at every $\mathbf{d} \in \mathcal{D}$, as the process model can fill spatial gaps between measurements with estimated values. In particular, in this paper we introduce state-of-the-art Gaussian processes in MB-DeepIG. Note that other models like e.g. a linear model or a partial differential equation could be easily introduced in MB-DeepIG.

In addition to gathering information, robots should also learn how to avoid inter-robot collisions. Therefore, we propose a simple reward that penalizes collisions. As values output from eq. (1) lie between $[0,1]$, we set $r_t^m = -1$ if robot m collides with any other robot.

C. Observation Renderer

Observation Renderer module maps information gathered by robots into an image observation. At any $t \in \mathbf{t}$, Information Storage stores two types of information: (i) robots current positions $\mathbf{x}_t^m \in \mathcal{D}$ from all $m = 1, ..., N$, and (ii) all past measurements $z_{t_i}^m \in \mathcal{P}$ from all $m = 1, ..., N$ for $t_i = 1, 2, ..., t$.

First, let us specify how we represent robots' current positions. Specifically, we represent robots as black rectangles with two possible orientations. That is, for each $m \in \mathcal{M}$, robot m DeepIG instance renders $\mathbf{x}_t^m \in \mathcal{D}$ as a horizontal rectangle. In contrast, robot m DeepIG instance renders $\mathbf{x}_t^{\bar{m}} \in \mathcal{D}$, for $\bar{m} \in \mathcal{M}$ with $\bar{m} \neq m$, as vertical rectangles. Thanks to this trick, robot m DeepIG instance is able to associate a robot m action with an observation and reward.

Additionally, we implement a coloring algorithm to render measurements values as an image. In particular, we render unmeasured $\mathbf{d} \in \mathcal{D}$ as blank pixels, and measured ones as color pixels. We use the Viridis color palette to render measurement values as color pixels. Note that we normalize measurements values so that they cover the full color palette. Viridis is perceptually uniform, and is robust to colorblindness. The former property implies that close measurements values are mapped to similar colors, which is crucial to generalize experiences between similar environment's states.

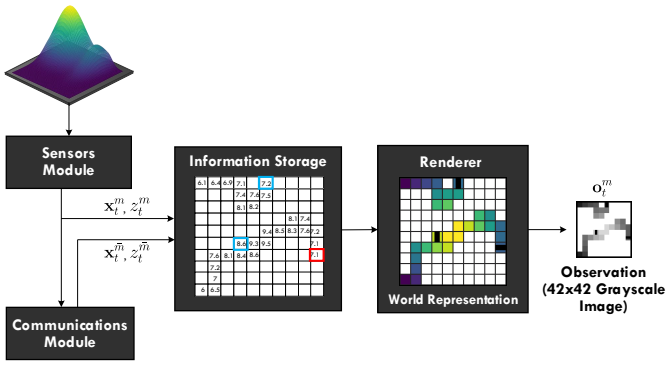


Fig. 3: Block diagram that depicts how measurements gathered by robots are mapped into an observation. Note that the cell highlighted in red in the Information Storage corresponds to robot m DeepIG.

The latter property allows us to transform a color image into a grayscale image while being able to distinguish every possible color in the palette. Thus, at $t \in \mathbf{t}$ the DeepIG instance of robot $m \in \mathcal{M}$ renders a grayscale 42×42 pixels image $\mathbf{o}_t^m \in \mathbb{R}^{42 \times 42}$ that corresponds to robot m observation of environment's state s_t. By using a one-channel grayscale image we reduce the number of parameters, and subsequent complexity, needed in the encoder layers of the MR-A3C neural's network.

To conclude this section, we depict in Fig. 3 the rendering process, which results in an observation. The observation, together with the reward, constitute the inputs to our Agent, which we describe next.

D. Agent

The Agent is the module responsible of deciding the next robot's action, which is given by π^m for $m = 1, ..., N$. In this section, we detail the Deep RL algorithm we propose to learn π that permits robots to gather information efficiently. First, we describe DeepIG's deep neural network architecture. Next, we introduce our learning algorithm: MR-A3C.

1) Deep Neural Network Architecture: DeepIG Agent utilizes a deep neural network that maps $\mathbf{o}_t^m$ to a_t^m and a value function [9]. As input the network takes $\mathbf{o}_t^m$. Observation $\mathbf{o}_t^m$ goes first through a convolutional encoder which comprises four convolutional layers. Each of these layers consists of 32 filters of 3×3 kernel with stride of 2, zero padding of 1, and an Exponential Linear Unit (ELU) activation function. The output of the convolutional encoder are feature maps. Feature maps are then fed into a recurrent layer consisting of 256 Long Short-Term Memory (LSTM) cells. The recurrent layer deals with the partial observability of the environment's state by accounting past observations. Finally, recurrent layer's output is fed into two different fully connected layers. The first one computes a linear output that results in a value function [9]. The second one utilizes a softmax activation function to compute π^m, which represents the probability with which the robot m should take an action $a \in \mathcal{A}^m$.

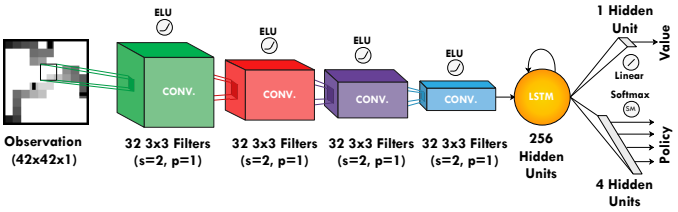


Fig. 4: DeepIG's deep neural network architecture.

We depict in Fig. 4 our deep neural network architecture. In the following, we explain MR-A3C, which allows a robot to learn DeepIG's deep neural network's parameters.

2) MR-A3C: MR-A3C is an extension of state-of-the-art A3C [11]. A3C was designed to train in parallel a single-agent system in multiple different environments. In contrast, MR-A3C allows us to train in parallel multiple multi-robot systems that gather information in multiple different environments.

The basic unit of MR-A3C consists of N robots and one environment in which robots perform an IG task. To simulate a basic unit we require two kinds of processes:

- *a robot process* that simulates a robot running DeepIG. The robot process executes an Agent instance that processes a reward and an observation to both select a new action, and to learn a policy.
- *an environment simulator process* that simulates $\mathcal{D}$, $\mathcal{P}$, and communication channel that permits inter-robot communication. The environment simulator process relies on a service that maintains a memory shared by N robot processes.

We simulate the interaction of robots with the environment by writing/reading to/from the shared memory. Besides, we simulate inter-robot communication by writing and reading from a dictionary object. That is, a robot process sends a message by writing in the dictionary, while a robot process receives messages from other robots by reading the dictionary. In the same way, every robot process takes a measurement $z_t^m \in \mathcal{P}$, for $m = 1, ..., N, t \in \mathbf{t}$, by reading the corresponding value of $\mathcal{P}$ simulated by the environment simulator process.

A basic unit of MR-A3C is sufficient to train a DeepIG Agent. Additionally, we speed up the training by running a cluster of multiple basic units in parallel. In this case, each basic unit simulates a different multi-robot system, and a different environment (see Fig. 5). Our proposed training method consists of multiple robots and world simulator processes. As in A3C, we use an architecture that comprises two classes of deep neural networks. On the one hand, we have a global deep neural network that is common to all basic units. On the other hand, each basic unit has a copy of the global network. It is the local neural network the one that is utilized by agents to select actions, generate experiences, and to calculate gradients. Then gradients are used to update, in parallel and asynchronously, the global network. In our implementation, we carry out updates every t_{max} time steps, or once a terminal state is reached. The global network is copied by all agents periodically.

To conclude, we summarize in Table I the hyperparameters used in our MR-A3C implementation.

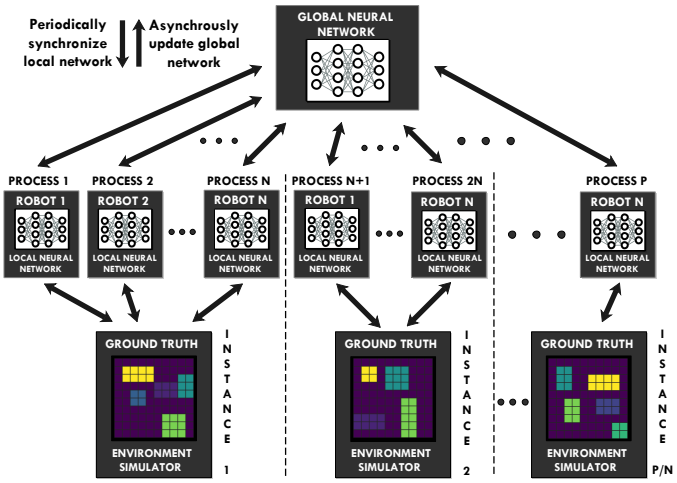


Fig. 5: MR-A3C block diagram.

Parameter	Value
Learning rate (α)	0.0001
Discount factor[1] (γ)	0.7
Value loss coefficient	0.5
GAE bias-variance trade-off (λ)	1
Entropy coefficient (β_H)	0.01
Maximum gradient norm	50
Number of forward steps (t_{max})	20
Adam optimizer decay rates (β_1, β_2)	0.9, 0.999
Adam optimizer numerical stability constant (ϵ)	10^{-8}

TABLE I: MR-A3C hyperparameters.

V. Simulations and Discussion of Results

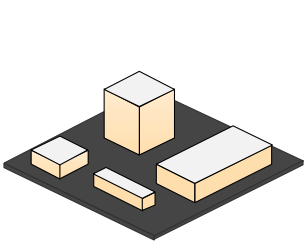
In this section we evaluate DeepIG in simulations. First, we introduce the simulations setup. Then we benchmark DeepIG against a random walk and a systematic sampling strategies. This is followed by an evaluation of MB-DeepIG and a comparison against state-of-the-art information driven GPs strategies. Finally, we illustrate DeepIG's scalability as we increase N.

A. Simulations Setup

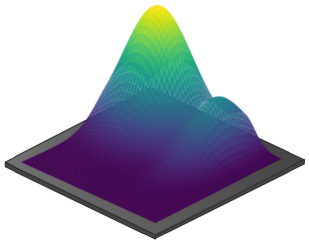
We first test DeepIG in two different IG tasks that generate $\mathcal{P}$ synthetically. First task corresponds to mapping a terrain profile $\mathcal{P}$ that consists of multiple boxes of different size, which are randomly positioned in $\mathcal{D}$ (see Figure 6a). Second task consists of mapping a process $\mathcal{P}$ that is a drawn from a Gaussian mixture model (see Figure 6b). Then we evaluate DeepIG using real data. These data corresponds to magnetic field intensity data that we gathered with a robot in an indoor environment as described in [2] (see Figure 8a). For all tasks, we assume $\mathcal{P} \subset \mathbb{R}_{\geq 0}$, $\phi = -0.5$, $n_c = n_r = 10$. We assume $\delta = 30$ cm for simulated processes, and $\delta = 10$ cm for magnetic field data.

We consider systems with $N = 1, 2, 3, 4$ robots, and assume that robots move between two cells in a single time step. We set $\eta = 0$, and fix $n_t = 140$ time steps.

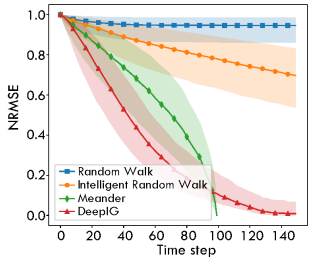
[1]We also tested values of gamma ranging between 0.1 and 0.9, and we obtained similar performance results.



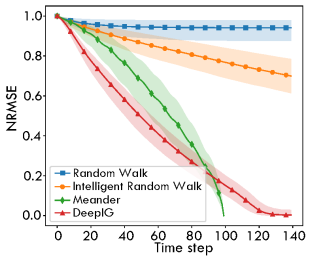
(a) Boxes-like process.



(b) Gaussian-like process.



(c) NRMSE boxes.



(d) NRMSE Gaussians.

Fig. 6: DeepIG evaluation for a single-robot system. Top: two examples of simulated processes. Bottom: comparison of DeepIG performance respect to systematic sampling strategies. DeepIG outperforms considered benchmarks.

Our implementation of DeepIG is based on PyTorch. We use a computer with 36 cores without GPU acceleration. For all simulations we use the deep neural network from Fig. 4, and hyperparameters from Table I.

Simulations go through two steps. First, we train till convergence four different DeepIG Agents end-to-end, with each Agent corresponding to a $N = 1, 2, 3, 4$ robots system. Specifically we train the single-robot Agent for 12 hours, and the $N = 2, 3, 4$ robots systems for 24 hours. In addition we train a single-robot MB-DeepIG Agent that uses GPs for 48 hours. Then we evaluate in simulations the performance of DeepIG, with a pre-trained Agent, to carry out the two aforementioned IG tasks. In particular, we evaluate the evolution with time of the NRMSE between $\mathcal{P}$ and $\tilde{\mathcal{P}}$ as robots gather information. Note that for each training and simulation run, we generate a random $\mathcal{P}$, and we initialize robots at random initial positions.

B. Comparison Against Systematic Sampling

We compare DeepIG against two strategies: a random walk, and a meander-like trajectory. For the random walk we consider two variants: one that is purely random, and one that is random but does not allow a robot to measure at an already visited cell (unless there is no other choice). We refer to this last variant as intelligent random walk.

First we evaluate DeepIG for a single-robot system to map boxes-like and Gaussian-like processes. We show results of this evaluation in Fig. 6c, 6d. In particular, curves corresponds to the mean NRMSE over 1000 simulations runs, and shaded area englobes a 1-sigma deviation from the mean.

First point that we would like to point out is that DeepIG NRMSE curves from Fig. 6c, 6d converge to zero. This allows us to conclude that DeepIG is able to learn how to gather information. Additionally, we calculated the area under the NRMSE curves for the two best algorithms (DeepIG and Meander). This yielded an area for Meander of 63.4 and 63.2 for Gaussian-like and boxes-like process, respectively; and an area for DeepIG of 53.7 and 49.4 for the same processes. Therefore, we can conclude that DeepIG outperforms the two considered benchmarks, as the area under DeepIG curves is lower than benchmarks curves.

To finalize, we would like to comment why DeepIG takes longer than a meander to bring the NRMSE to zero. A meander is designed to perform an optimal coverage. In contrast, DeepIG's goal is to reduce the NRMSE as fast as possible, which is equivalent to reducing the area under the NRMSE curve. If we observe the area under NRMSE curves, we can clearly see that DeepIG is superior to considered benchmarks.

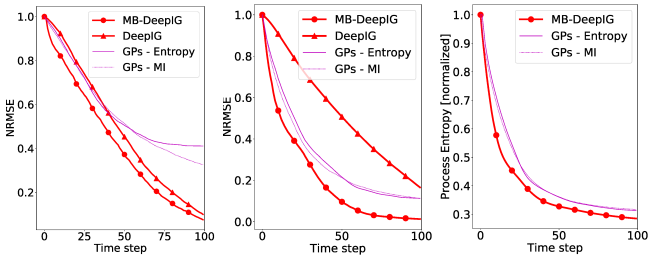
C. Comparison Against Model-Based Strategies

In this section we benchmark MB-DeepIG against two state-of-the-art model-based strategies. These use a GP to model the physical process of interest. In particular, we consider: entropy-driven [2], and mutual-information-driven algorithms [12] as benchmarks. For both algorithms, as well as for MB-DeepIG, we employ a zero mean and squared exponential covariance function, for which we learn the optimal GPs hyperparameters prior to each of the simulated IG tasks. Note that GPs hyperparameters could be also learned online from measurements, as indicated in [2].

In Figs. 7a, 7b we depict the mean NRMSE obtained for DeepIG, MB-DeepIG, entropy-driven, and MI-driven strategies. Additionally, we illustrate in Fig. 7c the posterior entropy, as calculated with the GP model, that remains about the process after the robot takes measurements. First conclusion that we can draw is that DeepIG outperforms entropy-driven, and MI-driven benchmarks for the boxes-like process, but not for the Gaussian-like one. As DeepIG does not use model information, it does not suffer from a model mismatch in the boxes-like process, and it does not benefit from a model match in the Gaussian-like one. This argument is exactly the opposite for the two considered benchmarks, which explains the curves behaviour.

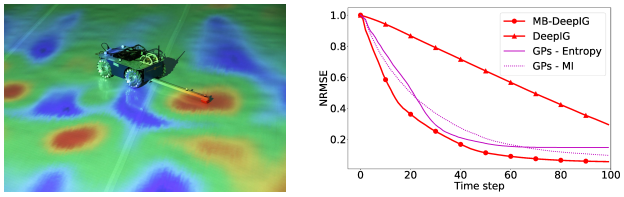
Next important fact that we can extract from Fig. 7 is that MB-DeepIG clearly outperforms GPs-Entropy and GPs-MI strategies in terms of NRMSE and posterior entropy. This demonstrates that MB-DeepIG was able to exploit the GPs model to learn a strategy that is more intelligent than the one used by the other benchmarks.

We finalize the evaluation of MB-DeepIG with an IG task that employs actual magnetic field intensity data. Magnetic field intensity is less structured than a boxes- or a Gaussian-like process. Therefore the derivation of an intelligent strategy becomes more challenging. This is exactly the result that we can observe in Fig. 8b. On the one hand, performance of DeepIG decreases. On the other hand, performance gap between MB-DeepIG and entropy- and MI-driven strategies gets reduced. Nevertheless, we can still conclude that MB-DeepIG offers the best performance with respect to the benchmarks in an actual IG task.



(a) (b) (c)

Fig. 7: MB-DeepIG evaluation for a single-robot system, and simulated IG tasks. (a) NRMSE for a boxes-like process; (b) NRMSE for a Gaussian-like process; (c) posterior entropy for a Gaussian-like process. We benchmark MB-DeepIG against DeepIG, entropy-driven [2], and MI-driven [12] strategies. MB-DeepIG offers the best performance.



(a) Magnetic field intensity. (b) Performance results.

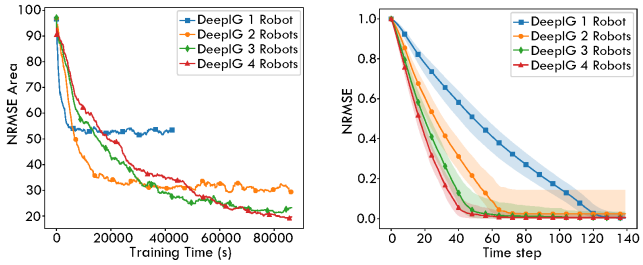
Fig. 8: MB-DeepIG evaluation for a single-robot system, and an actual process of interest $\mathcal{P}$. The process corresponds to magnetic field intensity data in an indoor environment.

D. Scalability with Number of Robots

Next we analyze DeepIG scalability as we increase N from 1 up to 4 robots. For that, there are two important metrics that need to be considered: learning rate, and NRMSE evolution with time. We use a boxes-like process for this evaluation.

First, we analyze DeepIG learning rate. Specifically, we choose as learning performance metric the area under the NRMSE-time curve. This matches our IG objective: reducing the NRMSE as fast as possible, as stated in Sec. II. We depict learning results in Fig. 9a. Curves correspond to the mean of the NRMSE area over the training time. In Fig. 9a we can observe that our learning metric decreases as training time increases. This demonstrates that agents learn how to gather information. Moreover, we can also see that our learning metric reaches a lower value as N increases, which exemplifies a proper coordination between robots.

In a second step, we use the trained agents to evaluate DeepIG performance as N increases (see Fig. 9b). According to Fig. 9b we can conclude that NRMSE decreases faster as N increases. This performance gain is particularly noticeable between systems with 1 and 2 robots. We can also see that performance gain diminishes as N increases. The explanation for this behaviour is very simple: robots must avoid inter-robot collisions, which limits robots possible actions. We would also like to point out that learned policies still cause robots to collide in 4 % of the episodes. One of our next steps is to extend DeepIG with more complex collision avoidance mechanisms like e.g. the one introduced in [8].



(a) DeepIG learning curves. (b) Performance results.

Fig. 9: DeepIG evaluation for a multi-robot system. On the left hand side we show learning curves of DeepIG. On the right hand side we present performance results for a boxes-like process and for systems with $N = 1, 2, 3, 4$ robots.

To conclude, we would like to remark that all curves in Fig. 9b converge to zero, which demonstrates that DeepIG was able to learn how to gather information with multiple robots.

VI. Experiments and Discussion of Results

In this section we present the setup and results of an experiment we carried out in our lab to validate DeepIG. In particular, we mapped an unknown terrain profile with three quadcopters, each of them equipped with an ultrasound sensor facing down.

A. Experiments Setup

We carried out experiments with systems composed by 1, 2 and 3 quadcopters. Quadcopters' task was to map a terrain profile, reducing the NRMSE between $\tilde{\mathcal{P}}$ and $\mathcal{P}$ as fast as possible. The terrain comprises an area of 3×3 m^2 in which we placed boxes of different height and width (see Fig. 1). Note that boxes distribution is completely unknown to quadcopters. Every quadcopter flies at a constant height of 1 m. For all experiments, we use same parameters as in Sec. V-A, except inter-drones safety distance that we set it to $\eta = 2\delta$ to avoid collisions. Besides, we transferred policy parameters, learned offline via simulations, to quadcopters' DeepIG instance.

We employ a commercial motion capture system (Vicon) to provide ground truth information of the quadcopters position. Quadcopters are each one equipped with a commercial ultrasound sensor from MaxBotix facing down to measure the range to the floor. Ultrasound sensors have a nominal range of approx. 7.5 m with opening angles of 45° in the near field and a cylindrical measurement profile for ranges greater than 1.5 m. Quadcopters calculate the terrain's height as the difference between their actual height, as given by Vicon, and the ultrasound sensor measured range between quadcopters and boxes. Here we use the Robot Operating System (ROS) to control quadcopters, to read ultrasound sensor measurements, and to implement inter-drones communication.

B. Experiments Results

First aspect that we can remark is that quadcopters mapped the complete space in the three experiments. In particular,

quadcopters required 143, 70 and 59, time steps with the 1, 2 and 3 robots systems, respectively.

Second relevant conclusion that we can extract is that DeepIG was able to deal with measurement noise introduced by the ultrasound sensor. In fact, these results confirmed that DeepIG deep neural network could account for noisy measurements, although we did not include measurement noise during DeepIG training phase. Note that measurement noise could be also introduced in the training. Additionally, signal processing techniques could be used in conjunction with DeepIG to reduce the impact of ultrasound sensor's noise. Both aspects are left for future work.

In addition, experiments demonstrated that the policy that we learned offline in simulations can be translated to a real MR-IG task. Although we considered in this paper a toy environment and process, experiments results allow us to conclude that DeepIG was able to learn how to gather information with multiple robots in a real environment.

Attached to this paper we include a video that illustrates the 3 drones experiment. In addition, in https://youtu.be/-aUUZPGiHlI we include a video that illustrates the DeepIG concept, the training process, and the 3 drones experiment.

VII. Conclusion and Future Work

We introduced in this paper DeepIG, a multi-robot information gathering (MR-IG) algorithm. DeepIG uses deep reinforcement learning to learn how to gather information, while avoiding inter-robot collisions. DeepIG employs *parameter sharing learning*, and extends state-of-the-art A3C algorithm to account for multiple robots that carry out an IG task. This extension we named it MR-A3C.

Additionally, we proposed a reward function that favors multi-robot cooperation. The proposed reward function allows us to incorporate a model of the physical process of interest into DeepIG. This extension of the algorithm we term it MB-DeepIG. On the one hand, DeepIG permits us to learn how to solve IG tasks whose model is unknown or too complicated. On the other hand, MB-DeepIG permits us to incorporate prior information about a process of interest by means of a process model. This is particularly relevant for tasks for which very accurate models have been developed.

We demonstrated first in simulations that DeepIG is able to learn MR-IG tasks without any parameters changes and using the same neural network for two different MR-IG tasks. In particular, we considered three IG tasks: mapping a terrain profile consisting of multiple boxes, mapping a Gaussian-like process, and mapping an actual magnetic field intensity. In a second step, we tested DeepIG, using a policy learned offline in simulations, in an experiment where three quadcopters had to map several previously unknown boxes. Results of the experiment allow us to conclude that policies learned offline with DeepIG can be translated to actual MR-IG tasks. Additionally, experimental results confirmed us that DeepIG could cope with actual measurement noise, although agents where trained in an ideal environment. In a latter step, we benchmark MB-DeepIG against state-of-the-art information-driven GPs-based strategies. Simulations results demonstrate a superior performance of MB-DeepIG with respect to the considered strategies.

Our next steps are related to increasing the complexity of the MR-IG task. This includes (i) more complex action spaces that take into account robots dynamics, (ii) time-variant processes, (iii) larger spaces, and (iv) sensors that output richer information. Additionally, we would like to investigate the use of GPU-based methods to train our agents.

References

[1] A. Singh, A. Krause, C. Guestrin, and W. J. Kaiser, "Efficient informative sensing using multiple robots," *Journal of Artificial Intelligence Research*, vol. 34, pp. 707–755, 2009.

[2] A. Viseras, T. Wiedemann, C. Manss, L. Magel, J. Mueller, D. Shutin, and L. Merino, "Decentralized multi-agent exploration with online-learning of gaussian processes," in *Robotics and Automation (ICRA), 2016 IEEE International Conference on*. IEEE, 2016, pp. 4222–4229.

[3] J. Capitan, M. T. Spaan, L. Merino, and A. Ollero, "Decentralized multi-robot cooperation with auctioned pomdps," *The International Journal of Robotics Research*, vol. 32, no. 6, pp. 650–671, 2013.

[4] R. S. Sutton, A. G. Barto, *et al.*, *Reinforcement learning: An introduction*. MIT press, 1998.

[5] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, "Control of a quadrotor with reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, Oct 2017.

[6] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sept 2017, pp. 2371–2378.

[7] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *CoRR*, vol. abs/1702.01182, 2017. [Online]. Available: http://arxiv.org/abs/1702.01182

[8] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 285–292.

[9] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[10] K. D. Julian and M. J. Kochenderfer, "Distributed Wildfire Surveillance with Autonomous Aircraft using Deep Reinforcement Learning," *arXiv:1810.04244 [cs]*, Oct. 2018.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[12] A. Krause, A. Singh, and C. Guestrin, "Near-optimal sensor placements in gaussian processes: Theory, efficient algorithms and empirical studies," *Journal of Machine Learning Research*, vol. 9, no. Feb, pp. 235–284, 2008.

[13] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *International Conference on Autonomous Agents and Multiagent Systems*. Springer, 2017, pp. 66–83.

[14] B. Bakker, "Reinforcement learning with long short-term memory," in *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, ser. NIPS'01. Cambridge, MA, USA: MIT Press, 2001, pp. 1475–1482. [Online]. Available: http://dl.acm.org/citation.cfm?id=2980539.2980731