sciendo

JAISCR, 2020, Vol. 10, No. 3, pp. 189 – 207
10.2478/jaiscr-2020-0013

MULTI AGENT DEEP LEARNING WITH COOPERATIVE COMMUNICATION

David Simões[1,2,3,*], Nuno Lau[1,2], Luís Paulo Reis[3,4]

[1]*Electronics, Telecommunications and Informatics Department, University of Aveiro, Portugal*

[2]*Institute of Electronics and Informatics Engineering of Aveiro, University of Aveiro, Portugal*

[3]*Artificial Intelligence and Computer Science Lab, Oporto, Portugal*

[4]*Informatics Engineering Department, Faculty of Engineering of the University of Porto, Portugal*

[*]*E-mail: david.simoes@ua.pt*

Submitted: 1st November 2019; Accepted: 26th March 2020

Abstract

We consider the problem of multi agents cooperating in a partially-observable environment. Agents must learn to coordinate and share relevant information to solve the tasks successfully. This article describes Asynchronous Advantage Actor-Critic with Communication (A3C2), an end-to-end differentiable approach where agents learn policies and communication protocols simultaneously. A3C2 uses a centralized learning, distributed execution paradigm, supports independent agents, dynamic team sizes, partially-observable environments, and noisy communications. We compare and show that A3C2 outperforms other state-of-the-art proposals in multiple environments.

Keywords: multi-agent systems, deep reinforcement learning, centralized learning

1 Introduction

A multi-agent system (MAS) is defined by an environment containing multiple interacting entities. These systems form the basis of most complex systems, and their entities include humans, animals, robots, and software agents. This team of distributed agents can accomplish tasks that would be impossible or impractical for a single agent. Examples can be found in multiple fields, including robotics [1], both in small- and swarm-scale, distributed tracking [2] for military purposes, traffic control in smart cities [3], craft formation [4], and complex on-line video-games [5, 6]. Modeling agents by hand for environments like these may prove to be prohibitively complex. Reinforcement learning, on the other hand, has considerable potential in such cooperative multi-agent environments. Agents are often required to run independently, solely with access to local partial information [7]. Decentralized agents also alleviate complexity problems by ignoring joint action-spaces, which grow exponentially with the amount of agents [8]. However, this makes it harder for agents to learn implicit coordination, where they assume the behavior of other members of the team, and act accordingly.

A possible solution is for agents to exchange information, compensating for their local action-observation history. Language and communication between intelligent agents have long been a topic of intense debate [9]. Through *centralized learning, distributed execution* paradigm [10], the learning phase can take advantage of global information and

error derivatives can be backpropagated through communication channels in order for agents to improve their teammates' policies. In practical terms, this causes agents to learn a communication protocol that benefits the team, and doing this for all agents leads to learning coordination among them.

We focus on settings that support *centralized learning, distributed execution* and cooperative environments that require relevant information sharing between agents of a team. This paper describes Asynchronous Advantage Actor-Critic with Communication (A3C2), a distributed deep reinforcement learning algorithm for cooperative multi-agent systems. Agents learn policies and communication protocols simultaneously, and exchange information through message-passing during execution. A3C2 supports noisy communications, distributed execution, and its agents are shown to achieve state-of-the-art performance in partially-observable cooperative environments. Our main contribution is the description and thorough evaluation of A3C2 in multiple multi-agent environments. It is compared against state-of-the art approaches, achieving the best results. The repercussions of different communication channels are also studied, including the effects of increasing the amount of transmitted information and the impact of introducing noise into communications.

The remainder of this paper is structured as follows. Section 2 describes the state-of-the-art on deep reinforcement learning algorithms in multi-agent systems. Section 3 describes our proposal, including our methodology to handle noisy communications. Section 4 describes two multi-agent environment suites, and Section 5 shows results obtained in those with A3C2 and other state-of-the-art algorithms. Finally, Section 6 draws conclusions and lists future work directions.

2 Related Work

The simplest multi-agent reinforcement learning algorithms rely on applying single-agent algorithms in a multi-agent environment, having each agent learning independently, as shown in the Independent Q-Learners (IL) algorithm [11]. Theoretical convergence guarantees are lost, due to the non-stationarity of the environment, but the method is versatile and popular [10, 12, 13, 14].

Another popular approach is the Joint-Action Learners (JAL) [11], where a central entity decides the joint-action for all agents based on the joint observation of the team. This approach is highly restrictive, as it does not support decentralized execution. In the limit, the environment becomes single-agent, where all entities are monolithically controlled.

These methods do not take advantage of the multi-agent aspect of the environment. This Section now describes the most relevant multi-agent deep-learning algorithms in the literature.

2.1 Counterfactual Multi-Agent Policy Gradients

The Counterfactual Multi-Agent Policy Gradients (COMA) [10] algorithm is an actor-critic extension that supports distributed execution, but requires centralized training. This *centralized-learning, distributed-execution* framework follows the intuition that algorithms (the value network, in this case) can be augmented with extra information regarding the other agents during the learning phase, while during execution only local information is required, thus allowing agents to run in a decentralized manner. Agents use network sharing for the critic network, so COMA does not support heterogeneous reward functions.

COMA uses the same centralized value network for all agents, with the shared agent observations and their actions as input. The use of agent actions as inputs for the value networks means the environment is now stationary for the critic, even as policies change. COMA addresses the credit-assignment problem by comparing how each agent's action effectively affects the expected value (using the critic network to estimate this).

Since the critic's architecture depends on the amount of agents being trained (as it incorporates their actions and observations), then COMA does not support dynamic amounts of agents. Using the same centralized critic for all agents also means the algorithm does not support different reward functions for different agents. Finally, it is unclear how the network scales to large numbers of agents.

2.2 Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments

The Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments (MADDPG) [15] is a DDPG extension that also follows the *centralized-learning, distributed-execution* approach. Similarly to COMA, the algorithm has a critic network with the shared agent observations and their actions as input. However, MADDPG uses a value network for each agent, which allows for agents with different reward functions to learn together (any non fully cooperative environment, for example).

MADDPG can also suffer from scalability issues, and does not support dynamic amounts of agents. The approach is based on implicit coordination. The authors compare their work with an IL version of DDPG on a proposed suite of environments, known as MPE, and described below.

2.3 Value-Decomposition Networks

Sunehag et al. present Value-Decomposition Networks (VDN) [16], where agents learn a factorized joint-action value function based on their independent observations, and the sum of each agent's estimation approximates the centralized joint-action function. Agents can communicate by concatenating the output of their layers at some points, thus assuming noiseless communication without constraints once more. VDN disregards any additional information available from the environment, and limits the complexity of the centralized joint-action function to a simple sum.

Rashid et al. present QMIX [7], a VDN-extension, where each agent's value function is no longer summed to approximate the centralized joint-action function. Instead, an additional *mixing network* is used to combine each individual value function in a more complex manner, which is also able to incorporate additional environment information. QMIX does not use communication between agents, and thus relevant information in partially-observable environments is not shared.

2.4 CommNet

The CommNet algorithm [17] proposes a single network in the multi-agent setting, passing the averaged message over the agent modules between layers. It uses a fully differentiable communication channel to learn explicit continuous communication between agents, learned concurrently with the agent's policy. The communication channel at time-step t is the summed transmission of messages sent by other agents at time-step $t-1$, and each environment state undergoes multiple communication steps (a value defined a priori).

Allowing multiple cycles of communications among agents is an uncommon assumption, since in many environments, actions usually have an equal or higher rate than message transmissions. Not only that, but the amount of cycles with which to communicate with is a hyper-parameter of the algorithm with no intuitive value. The model outputs actions for all agents simultaneously, similarly to JAL, which does not support distributed execution of the policies. It also makes CommNet unable to handle dynamic numbers of agents with this shared observation, and it remains unclear how it scales to large numbers of agents. Finally, the authors assume perfect communication between agents.

2.5 Multi-Agent Bidirectionally Coordinated Network

The Multi-Agent Bidirectionally Coordinated Network (BiCNet) [18] is an actor-critic extension based on Minimax-Q [19]. Using as an input the local view of an agent, and a shared view of all agents, a policy network outputs the action for an agent, and a value network the expected Q-value for that state. Agents are organized in a hierarchical order, and communicate with their neighbors, which allows a variable number of agents to use the same policy. Through the use of the RNN structure [20], agents have a local memory, and they share information between them while calculating their actions, by sharing the RNN state with their neighbors.

It is unclear what are the constraints of this sharing methodology and its robustness when communication channels can fail. The use of a shared observation for the policy network is also reminiscent of JAL, which does not support distributed execu-

tion of the policies. It also makes it unclear how the network handles dynamic numbers of agents with this shared observation, and how it scales to large numbers of agents. Finally, requiring RNN structures in the policy and value networks is a strong requirement, since not all problems require such complex structures, notoriously hard to train [21].

2.6 Differentiable Inter-Agent Learning

The Differentiable Inter-Agent Learning (DIAL) algorithm [9] uses a Q-network and a neural network that outputs messages through an end-to-end differentiable channel. Agents send messages at each cycle, and these messages are used as inputs for other agents' next cycles, along with their state observations. This approach requires centralized learning, although authors have also proposed an experience-replay based approach that supports decentralized learning [22]. Gradients are then pushed through the communication channels in order to optimize the messages to send.

The authors discretize the sent messages during execution and assume perfect communication between agents. DIAL is also only demonstrated to work in a limited set of short-horizon environments.

2.7 Others

Another end-to-end differentiable learning communication algorithm is found in the methods of Mordatch et al. [23]. Agents learn to communicate by learning a Gumbel distribution, later used on a set of discrete symbols, while simultaneously learning to act in an fully cooperative environment, using a joint reward function. Policies are based on neural networks with recurrent modules and support different numbers of agents. The algorithm requires fully cooperative environments, and the authors also assume perfect communication.

Das et al. [24] propose an algorithm for a one-on-one cooperative game. Using Hierarchical Recurrent Encoder-Decoder neural networks to model policies, and the REINFORCE algorithm [25] for learning a communication policy, agents learn to communicate using a pre-determined vocabulary consisting of natural-language symbols. Eventually, one of the agents guesses what image the remaining agent was shown.

D'Ambrosio et al. [26] use neural networks to learn communication in a hive-mind style. Certain neurons are shared among all agents, and the network learns how to set the weights in order to achieve coordination. However, this approach does not allow agents to run in a distributed manner.

Lazaridou et al. [27] also propose a communication learning algorithm for agents to use in order to identify images. In their work, communication is simply the action space with which policies are learned to complete tasks.

Some authors also show how communication can arise in a mix of multi-agent reinforcement learning frameworks and supervised learning techniques. By training agents to maximize a goal, and interspersing the training with supervised learning, Lewis et al. [28] demonstrate agents that learn natural language protocols. Using dialogue rollouts, the models plan ahead in bargaining tasks, and fake interest to take advantage of high-value goals.

The Multi-Step, Multi-Agent Neural Network (MSMANN) algorithm [29] uses supervised learning for decentralized agents to learn to imitate a centralized strategy. Agents learn action and communication policies simultaneously during centralized training, despite requiring a JAL strategy to be learned *a priori*. Authors leave a reinforcement-based approach for future work.

A3C2 relies on continuous communication protocols, based on generic message passing, which supports both distributed execution, as well as a dynamic number of agents during execution. It does not require any additional information from the environment, other than the one individually supplied to each agent.

3 Asynchronous Advantage Actor-Critic with Communication

Our proposal is based on the Asynchronous Advantage Actor-Critic (A3C) [30] algorithm, a single-agent deep reinforcement learning algorithm. Multiple distributed workers keep local copies of both actor and critic, and asynchronously update their global versions, as shown in Figure 1.

A3C is on-policy, operates in the forward view, and uses n-step returns to update both the policy and

the value-function every t_{max} steps or until a terminal state is reached. Actor-Critic methods decouple the value and policy functions into two separate networks. The Critic network with weights θ_v approximates a value function $V(o_t, \theta_v)$ and estimates the expected return at a given state o_t. The Actor network with weights θ_a maintains a policy $\pi(a_t|o_t, \theta_a)$ from which action a_t is sampled for state o_t.

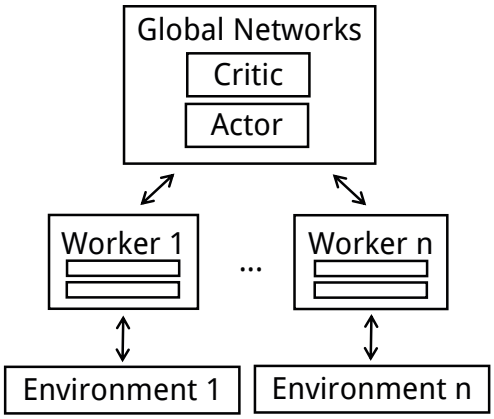


Figure 1. The framework for A3C [30], with n workers. Each worker keep a local copy of the on-line network, and interacts with its own environment. Updates are asynchronously performed on the global on-line and target networks.

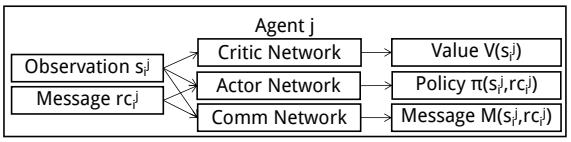


Figure 2. The architecture of an agent j at time-step t, which sampled observation o_t^j and received messages rc_t^j. Each agent has three separate networks: a policy (or actor) network, which outputs an action probability $\pi(o_t^j, rc_t^j)$ (from which a_t^j is sampled); a communication network, which outputs an outgoing message sc_t^j; and a value (or critic) network, which outputs a value estimation $V(o_t^j)$.

Our proposal, which we refer to as Asynchronous Advantage Actor-Critic with Communication (A3C2), is a multi-agent extension of A3C and allows for communication to be simultaneously learned alongside agent policies. Agents keep actor and critic networks, and an additional communication network with weights θ_c to output outgoing messages sc_t^j, as can be seen in Figure 2. These messages may be received as rc_{t+1}^j by other agents in the following time-step (depending on communication restrictions and constraints), and are used as input for their actor networks. During the execution phase, agents only require the actor and communication networks.

Each A3C2 worker keeps global networks for each agent, as can be seen in Figure 3. Workers copy the global networks into local memory and each handle an entire environment and set of agents for that environment. A3C can be framed as a specific case of A3C2, with a number of agents $J = 1$ and no communication.

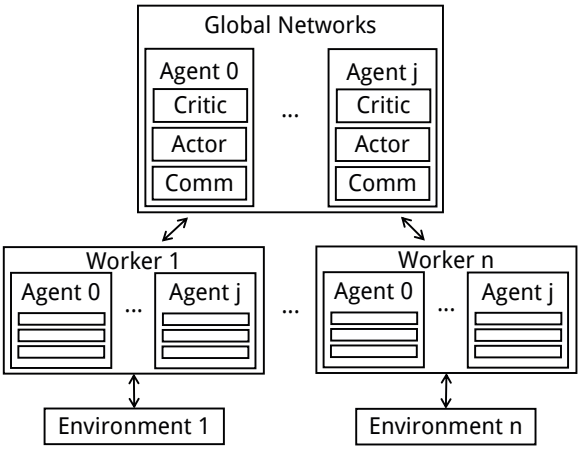


Figure 3. The algorithm architecture, using n separate workers. Each worker keeps local copies of each agent's three networks, and interacts with its own environment and its separate set of j agents. As samples are collected in mini-batches, workers calculate gradients based on their local networks, asynchronously update the global networks, and update their copies.

The pseudo-code for A3C2 is shown in Algorithm 1. A worker copies global networks into local memory (line 5) and obtains the initial observation of the environment (line 7). In a loop, the worker now computes an action and outgoing message (lines 10-11). The environment may be partially observable, and each agent's observation may be a local partial observation of the environment's current state. The worker also creates a record of the senders and recipients of each received message (line 12).

Algorithm 1 Pseudo-code for a worker thread running A3C2.

Require: Global shared learning rate η, discount factor γ, entropy weight β, number of agents J, network weights θ_a^j, network weights θ_v^j, network weights θ_c^j, batch size $t_{\max}$, maximum iterations $T_{\max}$, and default message value rc_{initial}

Require: Local network weights ϑ_a^j, network weights ϑ_v^j, network weights ϑ_c^j, and step counter t

1: $t \leftarrow 0$
2: $rc_0^j \leftarrow rc_{\text{initial}}$ for all agents j
3: **for** iteration T = $0, T_{max}$ **do**
4: Reset gradients $d\theta_a^j \leftarrow 0$, $d\theta_v^j \leftarrow 0$, and $d\theta_c^j \leftarrow 0$ for all agents j
5: Synchronize $\vartheta_a^j \leftarrow \theta_a^j$, $\vartheta_v^j \leftarrow \theta_v^j$, $\vartheta_c^j \leftarrow \theta_c^j$ for all agents j
6: $t_{start} \leftarrow t$
7: Sample observation o_t^j for all agents j
8: **repeat**
9: **for** agent $j = 1, J$ **do**
10: Calculate message sc_t^j to send with $sc_t^j \leftarrow M(o_t^j, rc_t^j, \vartheta_c^j)$
11: Sample action a_t^j according to policy $\pi(a_t^j | o_t^j, rc_t^j, \vartheta_a^j)$
12: Map sent communication sc_t^j into received communication rc_{t+1}, and build communication map m_t
13: Take action a_t^j for all agents j
14: Sample reward r_t^j and new observation o_{t+1}^j for all agents j
15: $t \leftarrow t+1$
16: terminal o_t^j for all agents j or $t - t_{start} = t_{max}$
17: **for** agent $j = 1, J$ **do**
18: $R^j = \begin{cases} 0 \text{ for terminal state } o_t^j \\ V(o_t^j, \vartheta_v^j) \text{ otherwise} \end{cases}$
19: $L_c \leftarrow 0$
20: **for** step $i = t-1, t_{start}$ **do**
21: $R \leftarrow r_i^j + \gamma R$
22: Value loss $L_{v_i}^j \leftarrow (R - V(o_i^j, \vartheta_v^j))^2$
23: Actor loss $L_{a_i}^j \leftarrow \log \pi(a_i^j | o_i^j, rc_i^j, \vartheta_a^j)(R - V(o_i^j, \vartheta_v^j)) - \beta H(\pi(a_i^j | o_i^j, rc_i^j, \vartheta_a^j))$
24: **for** step $i = t, t_{start}+1$ **do**
25: Received communication loss $L_{rc_i}^j \leftarrow \frac{\partial L_{a_i}^j}{\partial rc_i^j}$
26: Map received communication loss $L_{rc_{i+1}}$ into sent communication loss L_{sc_i} using communication map m_i for all agents
27: **for** agent $j = 1, J$ **do**
28: **for** step $i = t-1, t_{start}$ **do**
29: Accumulate gradients $d\theta_c^j \leftarrow d\theta_c^j + \frac{\partial L_{sc_i}^j}{\partial \vartheta_c^j}$
30: Accumulate gradients $d\theta_a^j \leftarrow d\theta_a^j + \frac{\partial L_{a_i}^j}{\partial \vartheta_a^j}$
31: Accumulate gradients $d\theta_v^j \leftarrow d\theta_v^j + \frac{\partial L_{v_i}^j}{\partial \vartheta_v^j}$
32: Update network weights $\theta_a^j \leftarrow \theta_a^j + \eta d\theta_a^j$, $\theta_v^j \leftarrow \theta_v^j + \eta d\theta_v^j$, and $\theta_c^j \leftarrow \theta_c^j + \eta d\theta_c^j$ for all agents j

Actions are executed on the environment (line 14), an action-state reward is obtained for each agent (line 15), and a new set of observations is sampled (line 15). After enough steps, or when a terminal state is reached, the loss of local networks is computed (lines 21-30), the gradients derived from the loss are applied to the global networks (lines 31-38), and these are then copied back into local memory (line 5). This process is repeated until convergence has been found.

A3C2 supports both intra- and inter-agent parameter sharing. In the intra-agent case, networks with the same input can be merged, and instead of optimizing separate networks, each agent optimizes a single network with multiple output layers. In inter-agent parameter sharing, homogeneous agents may share the same actor and communication networks, and agents with the same reward function may share the same critic network. This allows multiple agents to update the same network, resulting in a speed-up of the learning phase.

3.1 Actor Network

The policy network, shown in Figure 4, takes as input the current time-step's observation o_t^j, as well as any received messages from other agents rc_t^j, and outputs a probability distribution over the agent's action-space.

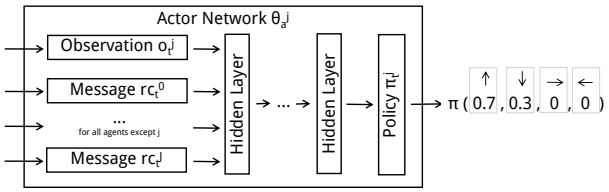


Figure 4. An exemplary architecture of agent j's actor network. In this case, the actor aggregates the observation and the broadcast message of all other agents as its input. The network's output layer then outputs a probability distribution for agent j's movement in four possible directions. The output layer is directly based on the environment's action space.

At each time-step t, agent j samples an action a_t^j based on his policy $\pi(a_t^j|o_t^j, rc_t^j, \theta_a^j)$. At each optimization cycle, a mini-batch of samples is used to optimize the actor network. The loss $L_{a_t}^j \leftarrow \log\pi(a_i^j|o_t^j, rc_i^j, \vartheta_a^j)(R - V(o_t^j, \vartheta_v^j)) - \beta H(\pi(a_i^j|o_t^j, rc_i^j, \vartheta_a^j))$ is given by the difference of the actual returns R and the critic's expectation $V(o_t^j, \vartheta_v^j)$ applied to the taken action a_i^j, and an additional entropy factor H with weight β to discourage premature convergence.

3.2 Critic Network

The value network, shown in Figure 5, takes as input the current time-step's observation o_t^j, and outputs a critic expectation, representing the expected returns for the current observation.

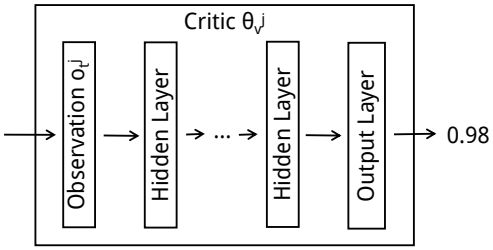


Figure 5. An exemplary architecture of agent j's critic. In this case, the network's output layer estimates the value of the current observation o_t^j, a value used for the advantage estimation during optimization. After the learning phase, the critic network is no longer necessary.

At each time-step t, agent j obtains a reward r_t^j. The actual returns are calculated, backwards, as $R_t = r_t + \gamma R_{t+1}$. The final expected return is given by 0 if the episode is terminal, or by the critic expectation $V(o_t^j, \vartheta_v^j)$ for the last taken observation o_t^j otherwise. At each optimization cycle, the loss $L_{v_t}^j \leftarrow (R - V(o_t^j, \vartheta_v^j))^2$ is given by the squared difference of the actual returns R and the critic's expectation $V(o_t^j, \vartheta_v^j)$.

3.3 Communication Network

The communication network, shown in Figure 6, takes as input the current time-step's observation o_t^j, and outputs an n-channel message (a vector of n values) the agent will send at the current time-step, and which may be received by others in the following time-step. Each channel has a continuous value based on the activation function (if any) of the communication network's output layer. For example, an 8-node output layer using binary activations computes single-byte messages.

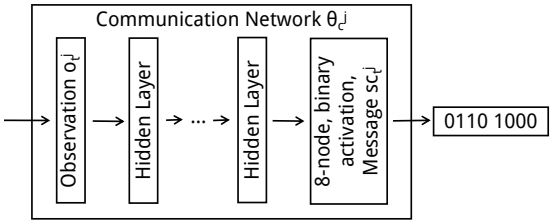


Figure 6. An exemplary architecture of agent j's communication network. In this case, the network's output layer uses a binary activation function and has 8 nodes, generating single-byte messages. Other output architectures are supported, including continuous valued messages. For example, a 10-node layer with *tanh* activations outputs a vector with elements $x_i, i = 1, \ldots, 10$, where each element $x_i \rightarrow [-1, 1]$.

Each environment defines communication constraints and properties, ranging from size to range properties. Messages can be classified as unreliable if they are lost with a probability $P_{\text{loss}} \neq 0$ or randomly delayed. A standard value rc_{initial} for a non-received message is used as the actor network input for both the initial turn (no agent has sent messages yet) and when messages are lost. The environment's connectivity is based on whether agents send messages to all other agents (broadcast), spatially close agents (local broadcast), or specific agents (unicast).

Agents may output a single message to all receivers, or distinct messages to each one, and may

receive each message distinctly, or aggregate all messages as one. This property may affect whether A3C2 supports dynamic amounts of agents, since the communication and actor networks' architecture depends on the sent and received messages. For example, if the communication network is built such that agent j sends unique messages to each member of its team, agent j cannot send messages to more agents than its expected team size. If, on the other hand, the team's size decreases, A3C2 supports noisy communications and simply handles a message that was not sent as a lost message.

A3C2 also supports noisy messages, where Gaussian noise $\mathcal{N}(0, V_{\text{noise}})$ with $V_{\text{noise}} \neq 0$ affects each message's content, which commonly happens in many wireless analog communication mediums. Finally, A3C2 handles internal interference, commonly found in systems where multiple entities transmit information through a global transmission medium (like humans talking aloud, or WiFi), where messages are sent simultaneously with a probability $P_{\text{jumble}} \neq 0$. In such cases, receivers do not distinguish between messages, and instead interpret a sum or average of the sent information, which may prove intelligible, depending on the chosen communication protocol and the amount of interference.

The output of agent j's communication network at time-step t is referred to as sc_t^j, and received by other agents in the next time-step as rc_{t+1}^j. At each optimization cycle, the error of received messages, is given by $L_{rc_t}^j \leftarrow \frac{\partial L_{a_t}^j}{\partial rc_t^j}$, representing how received messages impacted the agent's policy. The loss L_{sc_t} is then computed as the sum or average of this error, based on the compiled communication map m_t. In other words, L_{sc_t} represents the negative impact of agent j's sent messages on the policies of agents who received its messages. An example can be seen in Figure 7, where $J = 3$ agents broadcast their messages to each other.

The intuition behind this is that an agent j optimizes its sent messages in order to improve the policies of other agents. By doing this for all agents, a team learns to output information that is useful for others, thus learning coordination. The error of a sent message can be summed, averaged, or otherwise combined from the gradients of that message, which were computed by all the agents that have received it. Summing the error may lead to very large network updates, while averaging them leads to safer but slower updates.

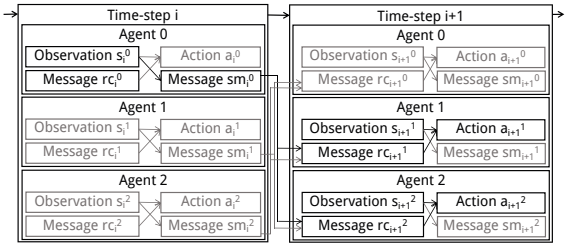


Figure 7. Diagram of how broadcast communication with three agents is performed across two time-steps (arrow direction), and of how gradients are propagated backwards (emphasized lines). Agent 0 sends messages to Agents 1 and 2, which calculate the gradients of their policy error with respect to that message. Those gradients are pushed as the error of the originally sent message into Agent 0, which uses them to optimize its Communication network.

4 Testing Environments

A3C2 is tested in two distinct groups of multi-agent environments, the POC and the MPE suites. Both are now described.

4.1 POC Suite

The Partially-Observable with Communication (POC) suite is a group of partially-observable multi-agent environments we have developed, where communication is crucial to overcome the partial observability of the environments.

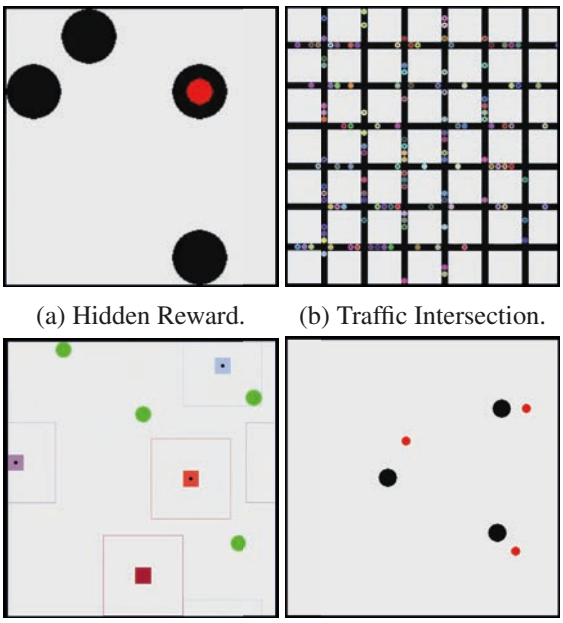
The scenarios, shown on Figure 8, are:

- *Hidden Reward* - Four agents. Focus on classic exploration, and agents need to learn how to efficiently explore the environment and alert team members when the target is found.

- *Traffic Intersection* - Dynamic amount of agents, total of forty. Close-horizon game, where agents need to learn to adhere to rules and overcome multiple indistinguishable intersections.

- *Pursuit* - Four agents and four hard-coded prey. Classical benchmark where agents need to explore and coordinate in order to capture the prey.

– *Navigation* - Two agents and two landmarks. Focus on goal assignment, and agents learn how best to distribute themselves in order to complete the task.



(a) Hidden Reward. (b) Traffic Intersection.

(c) Pursuit. (d) Navigation.

Figure 8. The POC suite. (a) Agents (black) only know their own position and explore the map until the reward zone (red) is found. They must then broadcast that location to remaining agents and converge on it. (b) Agents (colored) must cross intersections without colliding. They are given a desired direction (black arrow), know whether other vehicles are around them, and are penalized if they collide (red marker). (c) Predators (red) see only a small local area, and must chase, surround and capture the prey (green), which are hard-coded to run from the closest predator (shown by the blue line). (d) Agents (black) know the beacons' coordinates, but not each others' positions. They must cover all the beacons (red), and are rewarded by how close any agent is to each beacon (shown by the green line)

These environments are performance-oriented and provide a controlled environment with which to test multiple aspects of a multi-agent algorithm. While agents receive observations about the environment, it is possible to access the environment's underlying state directly, without any additional computations (aside from those already performed by the actual environment). This allows algorithms with the *centralized learning, distributed execution* method to augment their learning phase with additional information in an efficient manner. The environments are also targeted at cooperative teams, and built in such a way that a team of agents must use an information-sharing method to achieve successful coordinated strategies. For example, agents in the Traffic environment must share their intent to turn in order to avoid collisions.

4.1.1 Hidden Reward

The Hidden Reward challenge consists of several agents having to move across a toroidal map until they find a reward zone. Each agent has a local partial observation of the environment, with their own coordinates and whether they're in that zone or not. The complete state-space consists of a concatenation of all the agents' partial observations. There's both a global time limit since the challenge starts, and a smaller one since any agent finds the reward zone. In other words, agents have some time to explore the map and find the hidden reward, and a short time to gather there once it has been found. Because the time is not enough for a single agent to fully explore it, this not only forces spread coordinated exploration, but also an alert protocol when the reward is found.

Agents receive individual rewards each cycle, 0 points if not on the reward zone, and n points if on the reward zone, where n is the total amount of agents there, so cooperation is encouraged. At each time-cycle, agents can move in four directions or remain in the same position. Agents can broadcast messages to all other agents. The optimal strategy is to explore the map until the reward zone is found, and then broadcast its position to other agents.

4.1.2 Traffic Intersection

The Traffic Intersection simulator consists of several road intersections, which must be crossed by multiple vehicles. Each agent has a local partial observation of the environment, knowing their desired direction and sensing close vehicles. The complete state-space consists of the positions and intended directions of all vehicles.

Agents get small penalties for stalling traffic, big penalties if they crash, and even larger penalties if they crashed without having priority. At each

time-cycle, agents can move forward or remain in the same position. The communication range of agents is geographically limited to close vehicles (agents do not broadcast messages to all others, just to other agents in the same intersection). The optimal strategy is for a vehicle to inform others at intersections whether it needs to turn or not, and allow the vehicle with priority to cross the intersection.

4.1.3 Pursuit

The Pursuit game consists of two teams of agents, where one team must capture the other. The prey team is hard-coded, has a global vision, and each prey runs from the closest predator. Predators have local partial observations of the environment, sensing a small local area equivalent to less than 10% of the total map and their own global coordinates. The complete state-space consists of the positions of all predators and prey.

Agents get small penalties as time passes, and get penalized and randomly placed if they collide. At each time-cycle, agents can move in four directions or remain in the same position. Agents can broadcast messages to all other agents. A high-level strategy is for predators to explore the map until a prey is found, and then broadcast the prey's position so that all predators can converge and capture it.

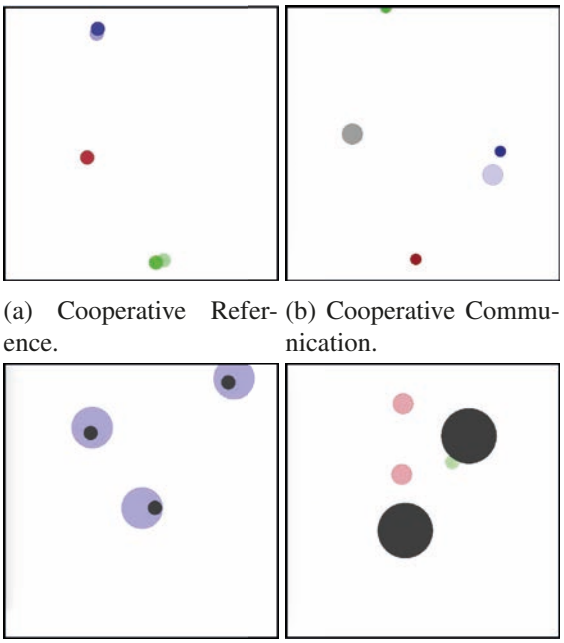
4.1.4 Navigation

The Navigation task consists of several agents having to cover all the beacons spread throughout the map. Each agent knows only its own position and the beacon positions. The complete state-space consists of the positions of all agents and beacons. There is a time-limit, but the episode ends early if all beacons are covered.

The team gets points at the end of each time-limited episode, based on how close an agent was to each beacon. At each time-cycle, agents can move in four directions or remain in the same position. Agents can broadcast messages to all other agents. The optimal strategy is for each agent to broadcast its own position and for the team to decide which agent should cover which beacon.

4.2 Multi-Agent Particle Environment

The Multi-Agent Particle Environment (MPE) [15] suite is a set of local continuous observation- and discrete action-space scenarios with simulated physics, which may incorporate communication.



(a) Cooperative Reference. (b) Cooperative Communication. (c) Cooperative Navigation. (d) Tag Challenge.

Figure 9. The MPE suite. (a) Agents (light colored) know their positions and their ally's target landmark (heavy colored). They must shared that information and converge on their corresponding targets. (b) The gray agent cannot move and knows the light blue agent's target (colored) must cross intersections without colliding. They are given the desired direction (black arrow), know whether other vehicles are around them, and are penalized if they collide (red marker). (c) Predators (red) see only a small local area, and must chase, surround and capture the prey (green), which are hard-coded to run from the closest predator (shown by the blue line). (d) Agents (black) know the beacons' coordinates, but not each others' positions. They must cover all the beacons (red), and are rewarded by how close any agent is to each beacon (shown by the green line).

The scenarios, shown in Figure 9, include:

- *Cooperative Reference* - Two allies, three landmarks. Allies know the target landmark of the other agent, and not their own, which they are rewarded for being close to.
- *Cooperative Communication* - The same as *Co-*

operative Reference, but only one ally can move, while the other knows its target.

– *Cooperative Navigation* - N allies and landmarks. Agents are rewarded by being close to each landmark.

– *Tag Challenge* - One adversary, N allies and landmarks. Allies try to touch as many times as possible the faster adversary.

In order to successfully complete these tasks, mechanisms to handle partial-observability are required. These may range from memory of previous states to communication protocols. In addition, some tasks also require strong coordination skills, and might not be possible to complete within the context of a single-agent system. We refer the reader to Lowe et al. [15] for further information.

5 Results

A3C2 is compared against IL and JAL implementations of A3C. The JAL method gathers the observations of all agents and outputs a joint-action for the entire team. However, it could not learn any successful policy in any POC environment within reasonable time, and its results are not shown in the following Sections. A3C2 is further compared with MADDPG, COMA, VDN, and QMIX, state-of-the-art multi-agent deep reinforcement learning algorithms, all of which are outperformed by A3C2. We publish a video of learned policies on our repository, linked below.

Unless otherwise stated, we used a learning rate $\eta = 10^{-4}$, a future reward discount factor $\gamma = 0.9$, an entropy weight $\beta = 0.01$, $J = 4$ agents, and $N = 3$ concurrent worker threads. All networks' initial random weights were computed with the Glorot initializer [31] with default parameters, and were optimized with the Adam optimizer [32] with default parameters.

A grid parameter search was used to find the most adequate network architectures for each environment, with tests on 1 to 3 layers, and 10 to 120 nodes. We describe the fastest architectures we found that yielded successful policies, along with other hyper-parameters, in Table 1.

The chosen learning rate η was also the highest and fastest we found that consistently allowed the networks to converge. The future reward discount factor γ depends on the importance of future rewards, with 0 leading to policies that focus on short-term rewards, and 1 focusing on long-term expected rewards.

5.1 Effects of Communication

The architecture of the communication network affects the complexity of the learned information sharing protocol. Different amounts of communication channels (CC) are tested, demonstrating how explicitly sharing information with learned protocols can also, improve agent policies. The CC represent the width of the communication network's output layer and the length of sent messages. The communication network's output layer is activated with a hyperbolic tangent function, such that each CC outputs a continuous value $[-1, 1]$.

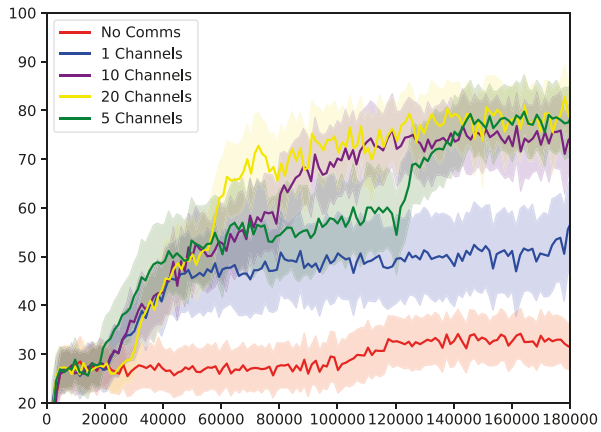
Figure 10 demonstrates that sharing information is vital to complete the tasks in the POC suite, and teams without communication severely underperform those with. Within A3C2 teams, a larger amount of CC often increases the performance of a team and speeds up the learning process. Abrupt performance increases can be observed when agent policies and communication protocols converge into a coordinated behavior. This can be observed in the POC: Hidden Reward environment, with 5 CC, around $120k$ training episodes.

In all scenarios, a single CC provides a noticeable benefit for the team's average obtained reward. For example, in the Traffic Simulator, it is enough for vehicles to signal their intention to turn and adhere to road rules. Unsurprisingly, different populations learn different protocols, and Figure 11 shows how it is possible to learn two diametrically opposite protocols in this environment, signaling either intent to turn, or intent to move forward.

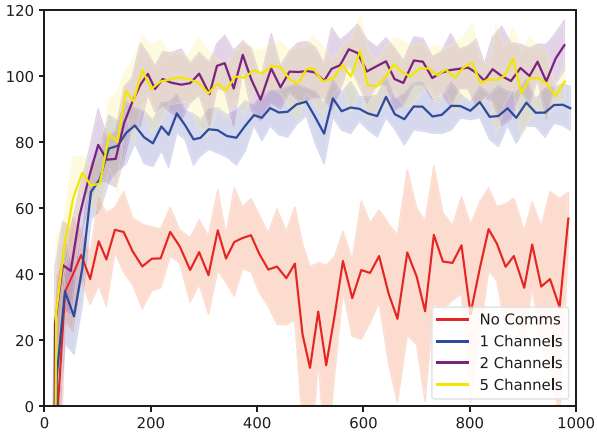
Teams in Hidden Reward learned to explore the map with a formation that maximized the covered ground. Agents also broadcast the reward zone's position when it was found, and others quickly converged in that position. In the Pursuit game, map exploration also improves and agents coordinate to surround and capture prey as soon as any predator finds it. Finally, teams in Navigation decide and coordinate which agent will cover which beacon.

Environment	J	N	γ	η	CC	x
POC: Hidden Reward	4	3	0.95	10^{-4}	20	2
POC: Traffic Simulator	40	3	0.1	10^{-4}	1	2
POC: Pursuit	3	12	0.95	5×10^{-5}	10	6
POC: Navigation	2	3	0.95	10^{-4}	20	4
MPE: Coop Navigation	3	3	0.001	10^{-4}	10	8
MPE: Coop Communication	2	3	0.001	10^{-4}	10	8
MPE: Coop Reference	2	3	0.001	10^{-4}	10	8
MPE: Tag	3	12	0.95	10^{-4}	10	8

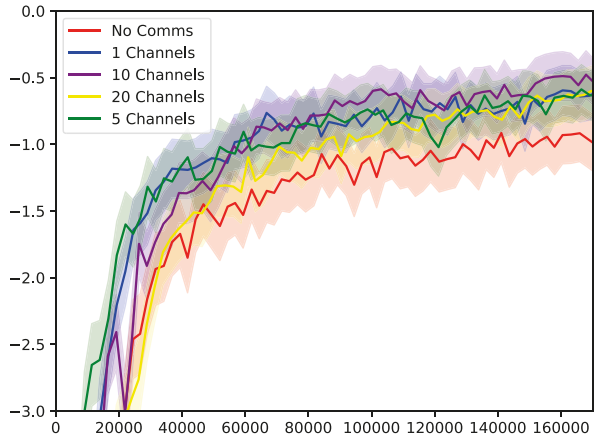
Table 1. The hyper-parameters used for the tests conducted in this Section. This table lists the amount of agents J, the amount of workers N, the future reward discount γ, the learning rate η, the amount of communication channels (CC), and the layer size modifier x. Critic and actor networks used two fully connected hidden layers of $20x$ and $10x$ nodes activated with a ReLU function. The communication network used a single hidden layer with $10x$ nodes activated with a ReLU function, and an output layer of CC nodes, activated with a hyperbolic tangent function. The non-received message rc_{initial} default value is all zeros.



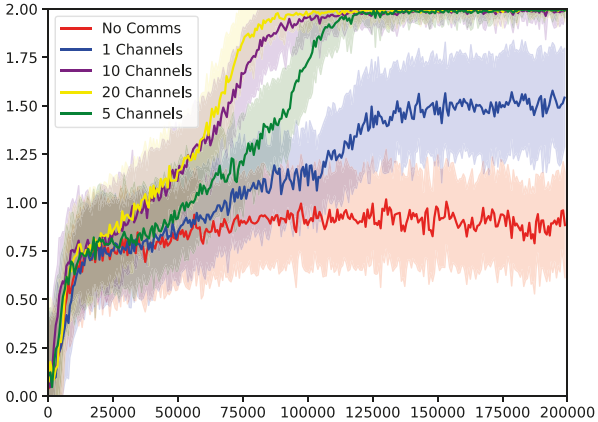
(a) POC: Hidden Reward.



(b) POC: Traffic Intersection.



(c) POC: Pursuit.



(d) POC: Navigation.

Figure 10. Results of A3C2 with different amounts of CC for the POC suite. The plots represent the average reward and standard deviation obtained by agents, over training episodes.

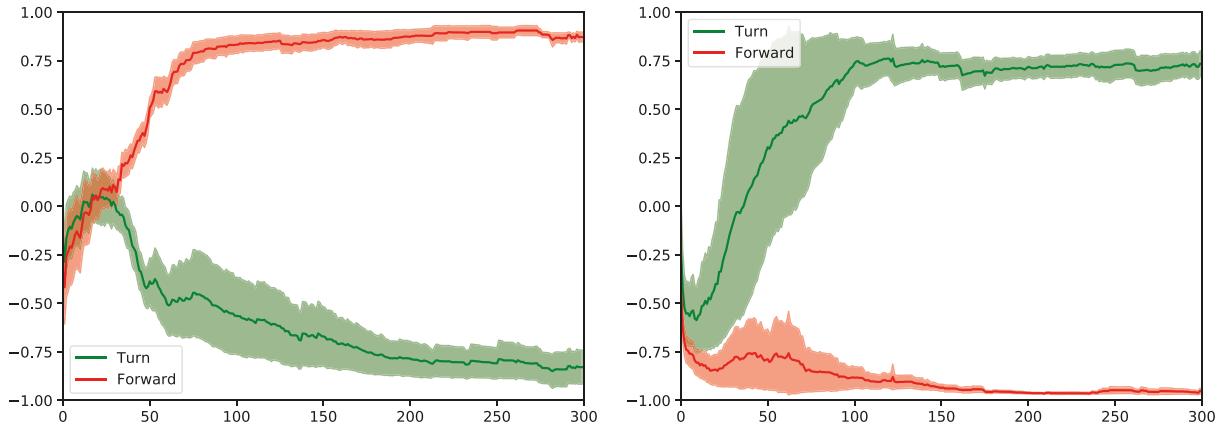


Figure 11. The evolution of two separate 1-channel communication protocols learned in the Traffic Simulator environment, using two separate agent populations. The plots represent the output message's single channel value across training episodes, when two agents stand at an intersection. The values are averaged over possible intersection situations (vehicles behind or in front of the agent), and split based on whether the agent intends to turn or move forward.

5.2 Evaluation in the POC Suite

A3C2 is also compared with COMA, VDN and QMIX in the POC suite, as shown in Figure 12. These algorithms rely on centralized critics for coordination, and feature no communication. To compensate for that lack of information sharing, the networks use a recurrent layer such that agents can remember information from their multiple partial observations of the environment, and their centralized critic has access to the underlying environment state. However, COMA, VDN and QMIX cannot achieve successful results in any of the environments.

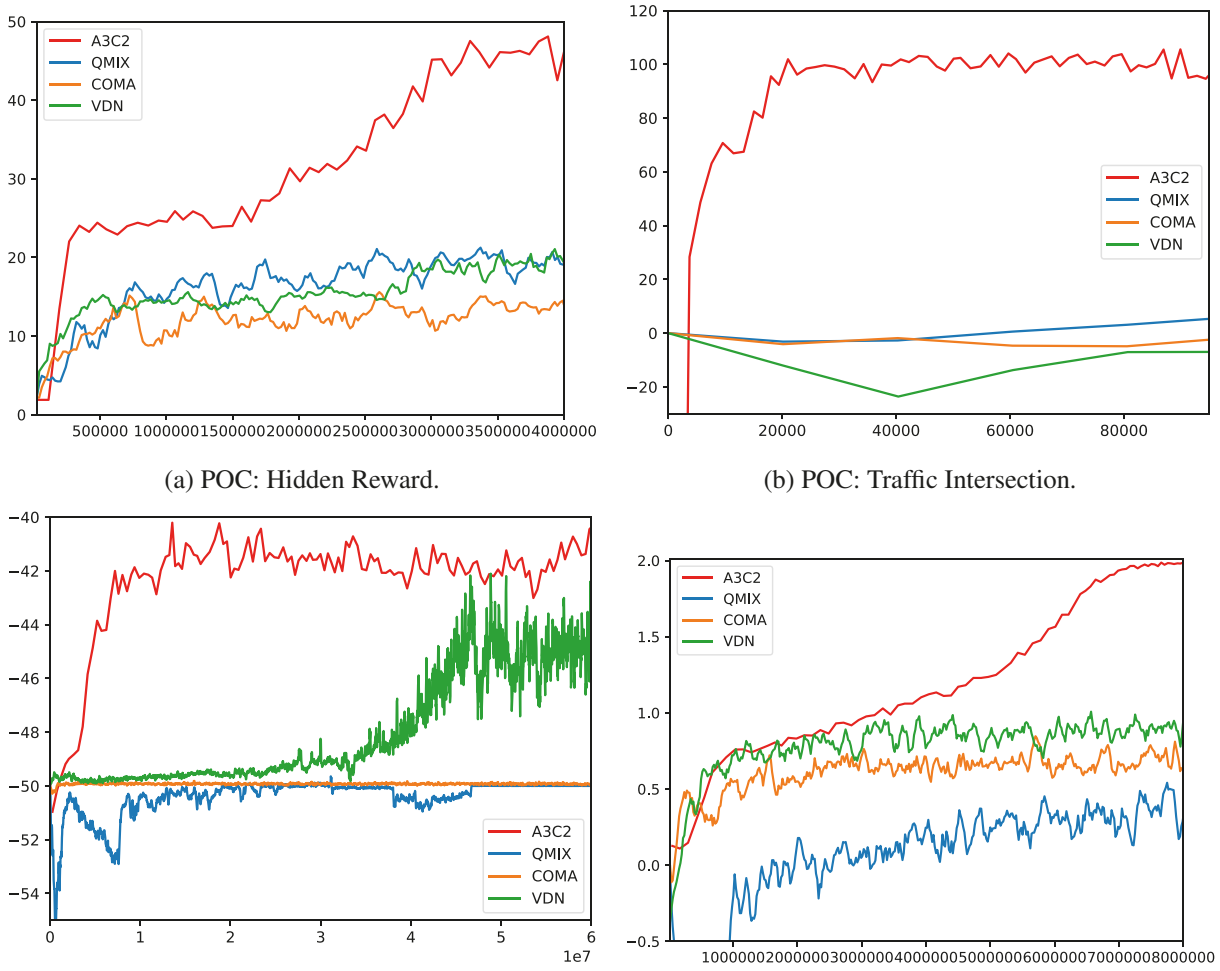
They match the performance of independent A3C in the cooperative tasks that can be partially completed without communication. In the Traffic Intersection simulator, the amount of agents impacts the performance of algorithms and they are unable to learn an adequate policy. In the Pursuit game, QMIX and COMA agents cannot overcome the partial-observability of the environment and the prey constantly elude them. Predators are only able to learn not to collide with each other. A3C2 clearly outperforms other state-of-the-art non-communication algorithms in all these environments.

5.3 Evaluation on the MPE Suite

Tests are now conducted on A3C, A3C2, DDPG, and MADDPG, on the MPE suite. While this Section summarizes each environment, the reader is referred to Lowe et al. [15] for further information. Apart from tag Challenge, all MPE environments provide instant progressive rewards, so a future reward discount factor $\gamma = 0.001$ is used in these for simplicity.

Tests conducted on MPE's Cooperative Communication and Cooperative Navigation evaluate communication and coordination. MADDPG agents integrate communication as an additional action-space with a predetermined vocabulary in these environments. In Cooperative Communication, one agent behaves as a speaker, and informs a listener agent of which of $L = 3$ targets is his. In Cooperative Navigation, $J = 3$ agents need to cover $L = 3$ available targets.

The performance of teams on these environments is given by $r = \sum_l^L -d_l - C$, where $L = 3$ is the total amount of landmarks, d_a is the minimum distance of each landmark l to any agent, and C is the amount of collisions on the environment. The results are shown in Table 2. In Cooperative Communication, A3C2 can learn policies that are more frequently successful and also achieve shorter distances to the target positions than DDPG and MADDPG. In the Cooperative Naviga-



(a) POC: Hidden Reward.

(b) POC: Traffic Intersection.

(c) POC: Pursuit.

(d) POC: Navigation.

Figure 12. Results of COMA, VDN, QMIX, and A3C2 for the POC suite. The plots represent the average reward obtained by the algorithms, over training steps.

	MPE: Cooperative Navigation		MPE: Cooperative Communication	
	Average Distance	# Collisions	Average Distance	Target Reached
A3C2	**0.219**	**1.223**	**0.007**	**99.6%**
MADDPG	1.767	0.209	0.133	84.0%
DDPG	1.858	0.375	0.456	32.0%

Table 2. Results of A3C2, MADDPG, and DDPG, for Cooperative Navigation and Cooperative Communication environments.

tion environment, A3C2 achieves more collisions but smaller distances, maximizing the rewards obtained by agents.

Tests conducted on MPE's Cooperative Reference also evaluate communication and coordination, but MPE's Tag Challenge does not require sharing information to achieve successful policies, and a team can rely solely on implicit coordination. In Cooperative Reference, agents know each other's target landmarks and must communicate this information to each other in order to find their own targets. In Tag Challenge, a team of predators chases a prey that moves at twice their speed with global map vision and a continuous observation space.

Because Tag Challenge is an adversarial environment, we used MADDPG in self-play to learn a policy for the prey, and then trained A3C, A3C2 and MADDPG against that static policy. Figure 13 demonstrates the evolution of policies learned by A3C and A3C2 against MADDPG baselines. A3C2 agents completed both the proposed tasks with higher performance than MADDPG, while A3C, lacking agent communication, was unable to do so in Cooperative Reference.

5.4 Effects of Communication Noise

We also test the effects of noise in the communication medium against a baseline where communication is unhindered. We focus on three major types of noise, all described above. A probability of losing a message $P_{\text{loss}} = 0.5$ represents how often sent messages are not received by their targets, covering both lost and delayed messages. Gaussian noise $\mathcal{N}(0, 0.5)$ represents external interference in the continuous-valued messages, simulating messages in analog mediums. Finally, a probability $P_{\text{jumble}} = 0.5$ represents how often internal interference occurs and agents receive jumbled messages, instead of distinct ones. We test each individual effect, and all three simultaneously (shown as "All"), and compare the results with a noiseless baseline.

Figure 14 shows that, as expected, noisy communication hinders the team's performance. Losing messages is the most disruptive perturbation, while noise or interference can often be off-set by robust communication protocols. Even when all noise types are enabled, teams outperform teams without any communication. This demonstrates that some communication is better than no communication at all, and how A3C2 supports imperfect communication mediums. Similarly to Figure 10, Noisy and Lossy communication protocols and policies achieve coordination in the POC: Navigation environment around $150k$ episodes, which leads to an abrupt performance increase.

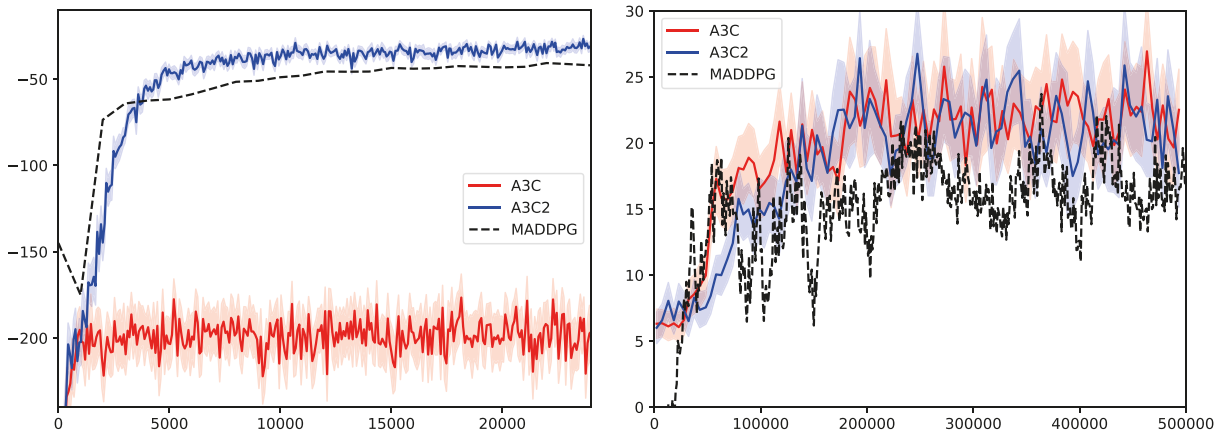
6 Conclusion

This paper describes A3C2, a multi-agent deep learning algorithm that follows the *centralized learning, distributed execution* paradigm. Agents learn independent policies and communication protocols simultaneously, achieving state-of-the-art results in multiple environments. A3C2 supports dynamic team sizes, noisy communications, and partially-observable domains. It is compared against MADDPG, COMA, VDN and QMIX, algorithms that also rely on *centralized learning*, and results demonstrate that the learned communication protocols surpass both centralized value-function techniques and techniques where a discrete communication alphabet is used as an additional action-space. A3C2's source-code and environments are published at https://github.com/bluemoon93/A3C2.

In the future, integrating centralized value-functions into A3C2 can help stabilizing and speeding-up the learning process. The value-function can also be augmented with additional information, such as agent actions or messages, to further exploit the *centralized learning* property. Additional network architectures can also be explored, such as the inclusion of recursive layers, allowing agents to maintain an internal memory, and possible learning more complex communication protocols.

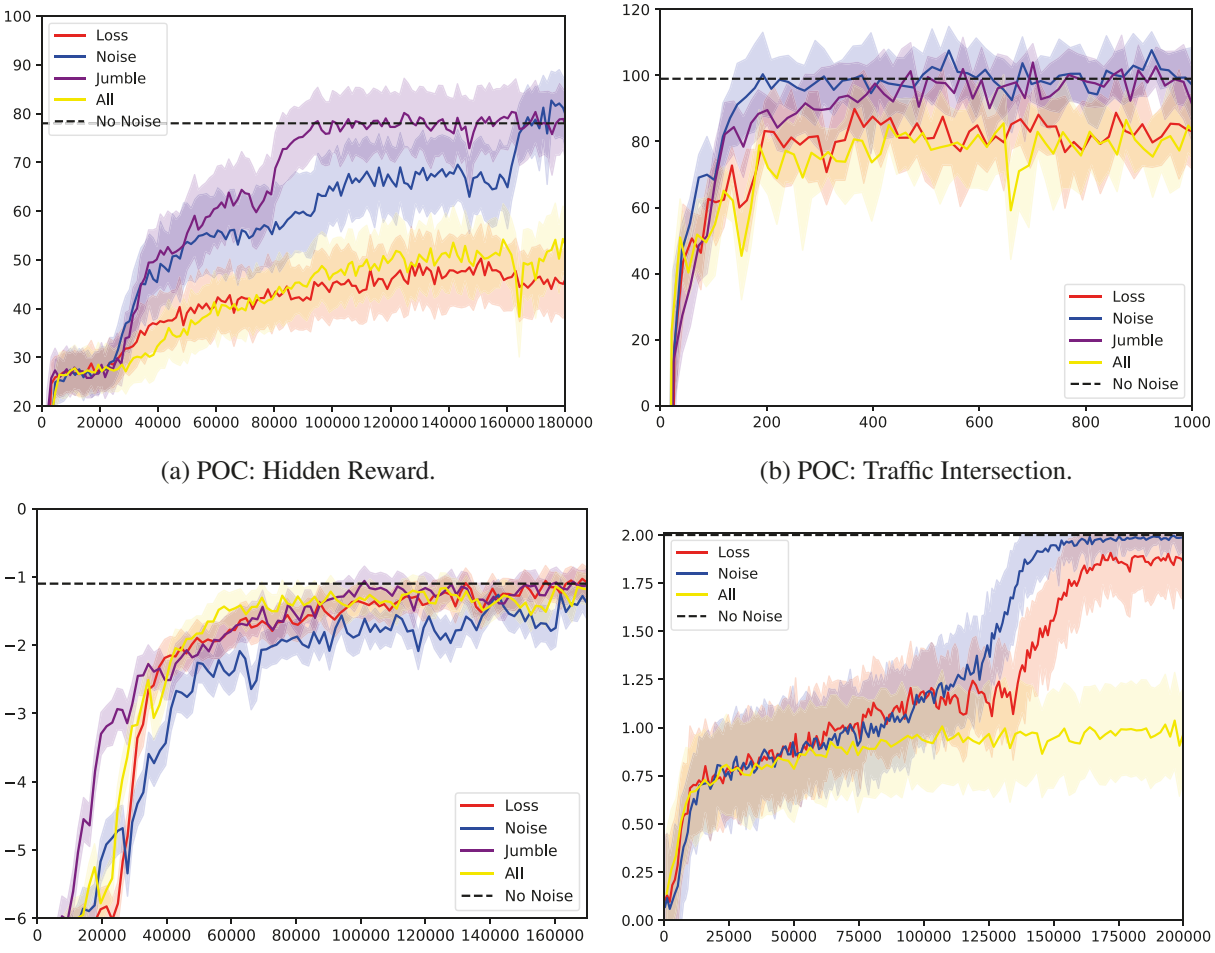
Acknowledgements

The first author is supported by FCT (Portuguese Foundation for Science and Technology) under grant PD/BD/113963/2015. This work was financially supported by: Base Funding - UIDB/00027/2020 of the Artificial Intelligence and Computer Science Laboratory – LIACC - funded by national funds through the FCT/MCTES (PIDDAC). This research was also supported by IEETA (UIDB/00127/2020).



(a) MPE: Cooperative Reference. (b) MPE: Tag Challenge.

Figure 13. Results of A3C, A3C2, and MADDPG for Cooperative Reference and Tag Challenge. The colored plots represent the average reward and standard deviation obtained by A3C and A3C2 agents, and the dashed plot represent MADDPG agents' average reward, over training episodes.



(a) POC: Hidden Reward. (b) POC: Traffic Intersection.

(c) POC: Pursuit. (d) POC: Navigation.

Figure 14. Results of the effects of noise for the POC suite. The solid plots represent the average reward and standard deviation obtained by agents over training episodes. The dashed plot represents the average reward obtained by agents without communication noise at the end of the learning phase.

References

[1] Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. arXiv preprint arXiv:1709.06011, 2017.

[2] Lili Ma and Naira Hovakimyan. Vision-based cyclic pursuit for cooperative target tracking. Journal of Guidance, Control, and Dynamics, 36(2):617–622, 2013.

[3] Patrick Mannion, Jim Duggan, and Enda Howley. An experimental review of reinforcement learning algorithms for adaptive traffic signal control. In Autonomic road transport support systems, pages 47–66. Springer, 2016.

[4] P Skobelev, E Simonova, and A Zhilyaev. Using multi-agent technology for the distributed management of a cluster of remote sensing satellites. Complex Syst: Fundament Appl, 90:287, 2016.

[5] Oriol Vinyals, Timo Ewalds, Sergey Bartunov, Petko Georgiev, Alexander Sasha Vezhnevets, Michelle Yeo, Alireza Makhzani, Heinrich Küttler, John Agapiou, Julian Schrittwieser, et al. Starcraft ii: A new challenge for reinforcement learning. arXiv preprint arXiv:1708.04782, 2017.

[6] Jonathan Raiman, Susan Zhang, and Filip Wolski. Long-term planning and situational awareness in openai five. arXiv preprint arXiv:1912.06721, 2019.

[7] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. CoRR, abs/1803.11485, 2018.

[8] L. Busoniu, R. Babuska, and B. De Schutter. A comprehensive survey of multiagent reinforcement learning. Trans. Sys. Man Cyber Part C, 38(2):156–172, March 2008.

[9] Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In Advances in Neural Information Processing Systems, pages 2137–2145, 2016.

[10] Jakob N Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[11] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. In Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative Applications of Artificial Intelligence, AAAI '98/IAAI '98, pages 746–752, Menlo Park, CA, USA, 1998. American Association for Artificial Intelligence.

[12] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. PLOS ONE, 12(4):1–15, 04 2017.

[13] Maxim Egorov. Multi-Agent Deep Reinforcement Learning. Technical report, University of Stanford, Department of Computer Science, 2016.

[14] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In International Conference on Autonomous Agents and Multiagent Systems, pages 66–83. Springer, 2017.

[15] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In Advances in Neural Information Processing Systems, pages 6379–6390, 2017.

[16] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning based on team reward. In Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '18, pages 2085–2087, Richland, SC, 2018. International Foundation for Autonomous Agents and Multiagent Systems.

[17] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In Advances in Neural Information Processing Systems, pages 2244–2252, 2016.

[18] Peng Peng, Quan Yuan, Ying Wen, Yaodong Yang, Zhenkun Tang, Haitao Long, and Jun Wang. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. CoRR, abs/1703.10069, 2017.

[19] Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In Proceedings of the eleventh international conference on machine learning, volume 157, pages 157–163, 1994.

[20] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. nature, 323(6088):533, 1986.

[21] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In International conference on machine learning, pages 1310–1318, 2013.

[22] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson. Stabilising experience replay for deep multi-agent reinforcement learning. In Proceedings of the 34th International Conference on Machine Learning-Volume 70, pages 1146–1155. JMLR. org, 2017.

[23] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. In Thirty-Second AAAI Conference on Artificial Intelligence, 2018.

[24] Abhishek Das, Satwik Kottur, José MF Moura, Stefan Lee, and Dhruv Batra. Learning cooperative visual dialog agents with deep reinforcement learning. In Proceedings of the IEEE International Conference on Computer Vision, pages 2951–2960, 2017.

[25] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine Learning, 8(3):229–256, May 1992.

[26] David B D'Ambrosio, Skyler Goodell, Joel Lehman, Sebastian Risi, and Kenneth O Stanley. Multirobot behavior synchronization through direct neural network communication. In International Conference on Intelligent Robotics and Applications, pages 603–614. Springer, 2012.

[27] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. Proceedings of the International Conference on Learning Representations, 2017.

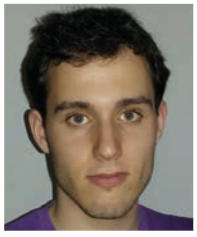
[28] Mike Lewis, Denis Yarats, Yann Dauphin, Devi Parikh, and Dhruv Batra. Deal or no deal? end-to-end learning of negotiation dialogues. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 2443–2453. Association for Computational Linguistics, 2017.

[29] Qiyang Li, Xintong Du, Yizhou Huang, Quinlan Sykora, and Angela P. Schoellig. Learning of coordination policies for robotic swarms. CoRR, abs/1709.06620, 2017.

[30] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In International Conference on Machine Learning, pages 1928–1937, 2016.

[31] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, volume 9 of Proceedings of Machine Learning Research, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[32] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. 3rd International Conference for Learning Representations, San Diego, 2015.



David Simoes obtained a M.Sc. (2015) in Computer and Telematics Engineering from the University of Aveiro, Portugal, and is currently a Ph.D. student in a joint Ph.D. program at the Universities of Minho, Aveiro and Porto (Portugal). His thesis topic is on learning coordination in multi-agent systems. He has worked on simulated humanoid robots and achieved different ranks in Robocup competitions including 4 world championships, and has worked in robotic and simulated maze-solving competitions, winning several national Micro-Rato competitions. His main research interests include multi-agent systems, deep learning, and game theory.

Nuno Lau is Assistant Professor at Aveiro University, Portugal and Researcher at the Institute of Electronics and Informatics Engineering of Aveiro (IEETA), where he leads the Intelligent Robotics and Systems group (IRIS). He got his Electrical Engineering Degree from Oporto University in 1993, a DEA degree in Biomedical Engineering from Claude Bernard University, France, in 1994 and the Ph.D. from Aveiro University in 2003. His research interests are focused on Intelligent Robotics, Artificial Intelligence, Multi-Agent Systems and Simulation. Nuno Lau participated in more than 15 international and national research projects, having the tasks of general or local coordinator in about half of them. Nuno Lau won more than 50 scientific awards in robotic competitions, conferences (best papers) and education. He has lectured courses at Ph.D. and M.Sc. levels on Intelligent Robotics, Distributed Artificial Intelligence, Computer Architecture, Programming, etc.

Luís Paulo Reis is an Associate Professor at the Faculty of Engineering of the University of Porto in Portugal and Director of LIACC - Artificial Intelligence and Computer Science Laboratory at the same University. He is an IEEE Senior Member and he was president of the Portuguese Society for Robotics and is vice-president of the Portuguese Association for Artificial Intelligence. During the last 25 years, he has lectured courses on Artificial Intelligence, Intelligent Robotics, Multi-Agent Systems, Simulation and Modelling, Games and Interaction, Educational/Serious Games and Computer Programming. He was the principal investigator of more than 10 research projects in those areas. He won more than 50 scientific awards including wining more than 15 RoboCup international competitions and best papers at conferences such as ICEIS, Robotica, IEEE ICARSC and ICAART. He supervised 20 Ph.D. and 102 M.Sc. theses to completion and is supervising 8 Ph.D. theses.