Enhancing Cooperative Multi-Agent Systems with Self-Advice and Near-Neighbor Priority Collision Control

Elizabeth Palacios-Morocho, *Student Member, IEEE,* Saúl Inca and Jose F. Monserrat, *Senior Member, IEEE,*

***Abstract*—The coordination of actions to be executed by multiple independent agents in a dynamic environment is one of the main challenges of multi-agent systems. To address this type of scenario, a key technology called Reinforcement Learning (RL) has emerged, which enables the training of optimal cooperative policies among agents. However, traditional value decomposition methods suffer from unstable convergence when the number of agents increases. To address this problem, this paper proposes a novel algorithm based on centralized learning that employs a self-advice module to replace the joint action, thereby reducing the algorithmic complexity. The proposed algorithm uses the Joint Action Learning (JAL) concept to find an optimal approach and a collision controller module that was designed to further mitigate the risk of collisions. A comparison of the algorithm proposed is carried out with two benchmark algorithms. The first one focuses on decomposing the reward signal and the second one trains a different actor-critic network for each agent. Furthermore, multiple target points are defined to enhance cooperative scenarios during the learning process. According to the results, the proposed approach outperforms the two benchmarks by 8% and 49%, thus highlighting the effectiveness of the centralized learning approach in multi-agent systems.**

***Index Terms*—Cooperative Multi-Agent System, Reinforcement Learning, Independent Learning, Joint Action Learning, k-Nearest Neighbors, Deep Deterministic Policy Gradient.**

I. Introduction

THE ability of human social groups to cooperate and interact with each other is a characteristic observed in microorganisms and animal groups. Achieving this collective intelligence that enables cooperation between different agents represents one of the great challenges of Industry 4.0 [1]. Many research studies have focused on using Artificial Intelligence (AI) techniques and the ability of agents to share their knowledge to achieve this cooperation. However, the complexity of these systems is increasing as more factors are involved [2]–[5].

RL is a field within the Machine Learning (ML) and AI that studies the learning of agents through their interaction with their environment. Therefore, the agent will learn how to behave in order to achieve its goals through trial and error. For instance, a possible task could be to plan a trajectory to reach a specific position. To accomplish this, the agent first uses its sensors to determine its state and then acts based on the information collected. This action will lead it into a new state, and, as a result, it will finally obtain a reward that will measure the quality of the action performed. This process is repeated continuously until the agent learns an optimal behavior policy [6], [7].

While RL focuses on solving problems involving a single agent, Multi-Agent Reinforcement Learning (MARL) is a standard framework for multi-agent environments. One of the main characteristics of these systems is the possibility of sharing data among all of them. However, due to the large amount of data collected, finding a behavior policy based on state and action tables as RL does is not easy. Therefore, deep neural networks are used to handle the vast amount of data collected [8], [9].

A MARL system can interact in a competitively, cooperatively, or mixed manner. For instance, in the cooperative mode, agents learn to coordinate with each other while sharing information. One of the approaches to address this type of system is a Independent Reinforcement Learning (IRL). In this approach, each agent treats the others as a static part of its environment and, as there is no communication between them, it will only know its own state and the action performed [10]. On the other hand, other methods assume that the agents can collect information from the others and use a reward obtained as a team [11], [12]. In such systems, one of the challenges to be solved is the problem of lazy agents. A lazy agent is one that learns a suboptimal behavior policy instead of the desired optimal one. This means that it prefers to choose those actions that do not maximize its reward in order to let others learn an obstacle-free optimal policy. In this way, it avoids possible collisions that would result in a negative reward for the team [13].

A. Related work

This section reviews from the point of view of the authors the most outstanding research focuses on MARL. However, most of them have been proposed and tested in small environments.

In multi-agent systems, one of the main concerns is the exchange of data between them, not only in terms of security but also in terms of latency or bandwidth that the telecommunication network must meet. That is why Yu et al. suggested an algorithm in which the raw data is not shared among the agents, as each one trains its model locally. However, there is still a need for communication between the agents and a central element in charge of calculating a global model. Hereby the global model is defined as the average of all local ones, and it is broadcasted to all agents in order to replace their local model [8]. Conversely, Kraemer et al. introduced an approach for learning using Decentralized

Partially Observable Markov Decision Process (Dec-POMDP). This method assumes that agents can train their models with information collected by sensors from other agents. However, they do not use this data to choose the action to be executed [14]. Likewise, Zhang et al. proposed an algorithm in which the agent shares its state and reward at each step. Thus, the immediate local reward is replaced by the frequency of obtaining the maximum immediate global reward for that action [15].

In a multi-agent environment, behavioral policies are very sensitive to variations in the conditions for which they were trained, such as an increase in the number of agents or the number of actions they can perform. For this reason, Chai et al. proposed a self-weighted mixture network, which is based on the factorization of the joint action value and the mapping of the Q-values in a nonlinear way [16]. On the other hand, to improve performance, Lee et al. suggested an imitation learning technique combining a benchmark policy with the Dec-POMDP. This improves the model of each agent's policy by using only a partial observation of the environment [17].

Another of MARL's major challenges is the coordination of multiple agents, to address this, Wang et al. proposed a method based on multi-agent interaction through sparse coordination, whose aim is to find an equilibrium solution, i.e., the solution that represents the maximum reward for all agents [18]. With the same purpose, Zhang et al. introduced the idea of transforming the environment into a P-model, which means that the reward return has only two possibilities: "one" when the agent obtains the maximum reward (success) or "$zero$" in the opposite case (failure) [19]. Whereas, Park et al. proposed recurrent neural networks based on actor-critical networks and deterministic policy gradients [20].

Other authors have proposed the use of different types of neural network architectures to solve these challenges, as in the case of Zhang et al., who presented a method using a Recurrent Neural Network (RNN) [21]. Sunehang et al. introduced a method that allows the agent first to learn using an individual architecture and then combine the weights of the different agents in a linear dual layer [13]. Wang et al. presented a method for factoring the joint action-value function of a duplex dueling network architecture into individual action-value functions [22], and Song et al. proposed an actor-critic policy delay to manage feature encoding and make navigation decisions [23].

Qie et al. suggested an approach in which each agent trains its own actor and critic networks. The actor-network receives only its observation, while the critic network uses its information and the action performed by the other [24]. Alternatively, Sun et al. proposed an algorithm based on the decomposing of the holistic reward signal into multiple sub-rewards and updates the policy by summing the distributed value functions [25]. However, these approaches represent a significant challenge since, on the one hand, it requires constant communication between agents, and on the other hand, it requires equipment with a higher computational capacity to handle and process the data. Moreover, as they focus on feedback from joint actions or shared rewards, these represent an effect on the rewards expected by each agent, which hinders a rapid convergence of the behavior policy.

Beyond the aforementioned works, some other authors have proposed novel approaches to improve the efficiency and performance of RL learning in specific autonomous navigation tasks, such as line change, tactical decision-making, prediction of future movements of surrounding vehicles, adjusting the speed of vehicles to avoid waiting in queues that may exist at intersections, or the combination of speed control and lane change to reduce energy consumption in the case of electric cars [26]–[31]. Liu et al, proposed a novel approach to solve the dispatching problem in areas of high dispatching concurrency using a single agent, for which it uses a list of recommendations for reassignments [32]. However, our goal is to achieve successful iterations among many agents using a common policy while reducing the number of existing communication links between them.

B. Contribution of the paper

This work focuses on the cooperation of neighboring agents to achieve their goals while avoiding collisions using a semi-centralized architecture. During the training phase, a centralized approach is adopted where the observations made by the agents are stored in shared memory and trained as if they were a single entity. In contrast, during the execution phase, a decentralized approach is taken, where each agent makes decisions based on its own observations. Despite the fact that execution is decentralized, all agents use the same policy over time. The contributions of this work detailed below focus on the design of an algorithm that allows the successful cooperation of neighboring agents.

1) We opted for a semi-centralized architecture using Deep Deterministic Policy Gradients (DDPG) to achieve global knowledge but with independence when the agent chooses the action to perform. On the one hand, this configuration allows all agents to act as if they were Independent Learning (IL) since they use only their own data to determine the action to be executed. But on the other hand, they have the advantage of having the same policy that was built from the knowledge of all of them. Moreover, this policy will have periodic updates coming from the cloud that is in charge of performing the training in a centralized way.
   Since we are seeking to learn a global policy, we decided to use the JAL concept; however, we replace the joint action that corresponds to the neighboring agents for the action suggested by the self-advice module. This is in order to avoid the dimensionality problem that appears in the joint action approach when there are many agents that must interact with each other.
   In order to reduce the lazy agent problem in MARL, we propose a future collision controller based on the k-Nearest Neighbors (kNN) algorithm and the priorities assigned to the agents in each task.

The rest of the paper is organized as follows: Section II covers the technical background. Section III presents the formulation of the proposal. Section IV shows the proposed algorithm. Section V discusses and compares the results

obtained from the proposed algorithm with the algorithms proposed by [24], [25]. Section VI draws the main conclusions of the paper and highlights some future research lines.

II. Technical Background

This section presents a brief introduction to RL, IL, JAL, and DDPG which are the underlying fundamentals of the proposed algorithm.

A. Reinforcement Learning

The field of ML that studies how an agent learns through interactions with its environment is RL. These iterations are used to train a behavior policy (π) based on the state (s_t) and reward (r_t) of the actions (a_t) performed over time t. Once the optimal policy has been found, the agent executes the action with the maximum Q-value, where this maximum Q-value $(max\ Q_a)$ indicates the best action to be performed.

As mentioned above, a multi-agent system where RL is applied is called MARL. This redefines the Markov framework as $(\mathrm{n}, \mathrm{S}, \mathrm{A}_{1...\mathrm{n}}, \mathrm{R}_{1...\mathrm{n}}, \mathrm{T})$, where n represents the number of interacting agents, S is the set of states, A is the set of actions, R is the set of rewards assigned to each agent considering the joint actions, and T is the transformation function $\mathrm{T} : \mathrm{S\ x\ A_1\ x\ A_2\ x\ ...\ x\ A_n} \rightarrow \mathrm{S}$. The number of joint actions will depend on the number of agents since it represents the action executed by each one of them [2], [33].

In Markov processes, it is known that any stationary distribution will converge over time. However, in a multi-agent system, agents are continuously in motion. This causes their environment to change from stationary to non-stationary, breaking the Markov assumptions of the need for a stationary distribution to achieve convergence and causing instability in training [34]. Within these systems, agents can learn approaches such as IL or JAL.

B. Independent Learning

An agent can operate as IL when it learns its policy using only its own knowledge. Moreover, the actions are chosen, assuming that the other agents are a static part of their environment even when they are constantly in motion. One advantage of IL is that it reduces inter-agent communications since this approach does not require the broadcasting of states or actions to be performed among all agents. On the other hand, it is easy to implement due to its simple structure [35].

In order to represent the Q-value of the neural network (Q_a) defined in [36] as a function of the executed action $(Q_a(a_t))$, the Q-value update is redefined as:

$$\begin{aligned} Q_a(a_t) \leftarrow & (1-\alpha) Q_a(a_t) \\ & + \alpha(R_{t+1} + \gamma\, Q_a(a_t)), \end{aligned} \tag{1}$$

where a_t belongs to the set of actions A_t, α represents the learning factor of the algorithm, and γ is the discount factor that determines the importance of future rewards.

C. Joint Action Learning

In contrast, an agent acting as a JAL learns the Q-value using not only their actions but also the actions of the other agents. This means that all agents will be aware of the actions actions performed by others. In this approach, the Q-value of each agent will be affected by the joint action space corresponding to all the agents in the system $(Q_a(a_t, A_t^{-m}))$. Therefore the update of the Q-value will be defined as:

$$\begin{aligned} Q_a(a_t, A_t^{-m}) \leftarrow & (1-\alpha) Q_a(a_t, A_t^{-m}) \\ & + \alpha(R_{t+1} + \gamma\, Q_a(a_t, A_t^{-m})), \end{aligned} \tag{2}$$

where $\mathrm{A_t^{-m}}$ is the set of actions corresponding to each of the remaining agents at time t.

D. Deep Deterministic Policy Gradient

This algorithm improves the actor-critic algorithm since DDPG focuses on learning policies in continuous action spaces. It consists of a critic-network and an actor-network, where each network contains two sub-networks corresponding to the online network (Q_a) and the target network (Q_b), both with the same architecture. The critic network evaluates state-action pairs, while the actor-network is trained to generate a deterministic policy (π), which is responsible for choosing action a_t based on π [37], [38].

Mathematically, the update of Q-value and the loss function (L) of DDPG can be expressed as a function of its state and action [34], [39].

$$\begin{aligned} Q_a(S_t, A_t) \leftarrow & (1-\alpha) Q_a(S_t, A_t) \\ & + \alpha(R_{t+1} + \gamma\, Q_b(S_{t+1}, a)), \end{aligned} \tag{3}$$

$$a = \max_{a \in A_t} Q_a(S_{t+1}, A_t), \tag{4}$$

$$L = \mathbb{E}\left[(Q_a(s_t, a_t) - (r_{t+1}^i + \gamma \cdot Q_a(s_{t+1}, A_t)))^2\right], \tag{5}$$

where $\mathbb{E}[\cdot]$ represents the expected loss value.

III. Formulation of the Proposal

A. Architecture

The deployment of multi-agent algorithms can be accomplished using three types of architecture: centralized, decentralized, and semi-decentralized. In a centralized architecture, the cloud is in charge of training the behavior policy and making the decisions of each agent in the system. However, this approach could present problems with the dimensionality of the data since the action space will become more extensive as the number of agents increases.

In a decentralized approach, each agent trains its policy and determines the action to be performed. In this type of approach, the policy trained by the agent can be affected by the overgeneralization problem. This is because the agent's reward will be affected by the actions performed by others and the agent will not understand why the reward is not equal to the expected one [40].

In this paper, a semi-decentralized system is applied, as illustrated in Fig. 1. This architecture separates tasks with high computational processing from tasks with lower processing

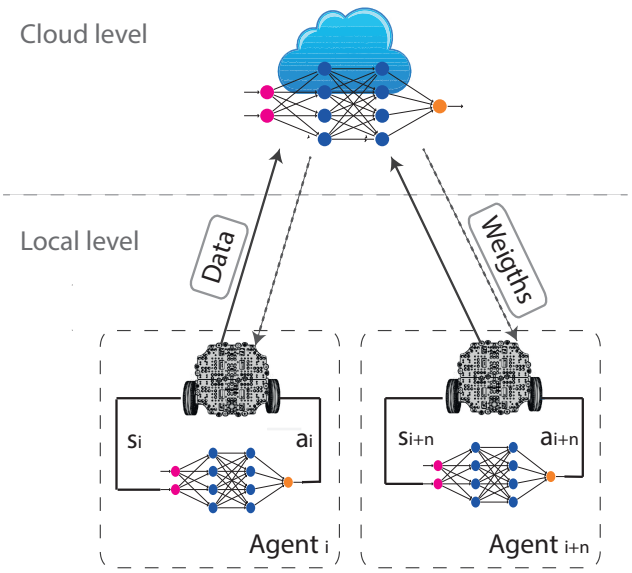

Fig. 1. In a semi-decentralized architecture, the cloud trains the policy and sends the weights regularly to each agent to update its policy. Meanwhile, the agent is in charge of reading the data, sending it to the cloud, and choosing the action to execute using the policy received from the cloud.

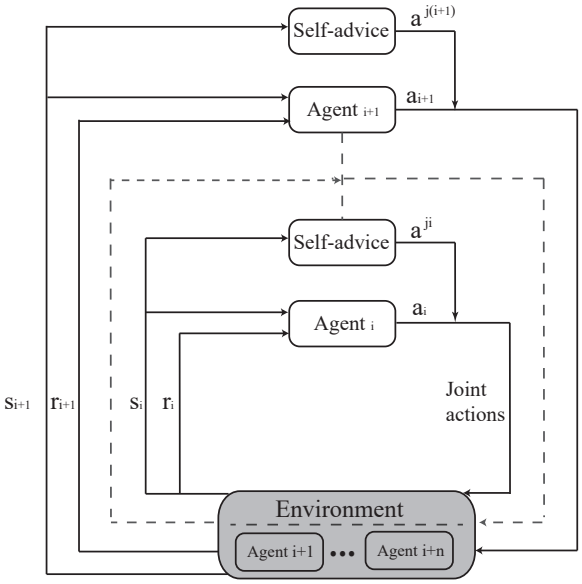

Fig. 2. Our approach is based on the fact that the agent chooses the action to execute considering only its information. However, the policy training will also use a joint action which is determined by a self-advice module that will consider the other agents as part of its environment but in a temporary way.

requirements. On the one hand, the cloud trains a global behavior policy and creates a 2D map of the environment using data from all agents. On the other hand, the agent determines the action to be performed using only the information from its sensors.

This approach reduces policy overgeneralization and data overfitting. Since the size of the memory storing the data to be sent for training will always be the same. Consequently, the number of agents involved in the cooperative system will not increase the algorithm's complexity.

B. Variation of JAL

This paper proposes an algorithm based on the idea of [24], where the authors proposed a cooperative algorithm using the actions performed by the other agents (JAL). However, if the number of agents increases, the complexity of this approach will also increase.

In order to solve that problem, this study focuses on using the concept of JAL to find the optimal behavior policy. However, instead of using the actions of other agents, the actions will be provided by a self-advice module (see Fig. 2).

Considering that all the samples from the different agents in the system are sent to a common memory in the cloud, the common policy is achieved by training the neural network with all these samples, treating them as if they came from the same agent. In this way, a simple learning model is obtained, which does not increase in complexity when the number of agents does. Since the number of actions will always be two, one corresponding to the action chosen by the neural network and the other corresponding to the one suggested by the self-advice module, which is explained in the following subsections.

Furthermore, this approach reduces the manipulation of data by third parties because even if agents send their sensor data to the cloud, only the cloud will have access to the data. In view of the above, the Q-value will be defined as:

$$\begin{aligned} Q_a(S_t, A_t, A_t^j) = & \, Q^i(s_t, a_t^i, a_t^{ji}) \\ & + Q^{i+1}(s_t, a_t^{i+1}, a_t^{j(i+1)}) \\ & + ... \\ & + Q^{i+n}(s_t, a_t^{i+n}, a_t^{j(i+n)}) \\ = & \sum_{l=1}^{P} Q_a^l(s_t, a_t^l, a_t^{jl}), \end{aligned} \tag{6}$$

where $Q^i(s_t, a_t^i, a_t^{ji})$ represents the Q-value of agent i as a function of its state, its action, and the action suggested by the self-advice module at time t, a^l being the action determined by each agent's policy, a^j the action given by the self-advice module, and P the number of agents participating in the common learning process.

Therefore, the update will be given by

$$\begin{aligned} Q_a(S_t, A_t, A_t^j) \leftarrow & (1-\alpha) Q_a(S_t, A_t, A_t^j) \\ & + \alpha(R_{t+1} + \gamma \, Q_b(S_{t+1}, a)), \end{aligned} \tag{7}$$

$$a = \max_{a, a^j \in A} Q_a(S_{t+1}, A_t, A^j), \tag{8}$$

and losses will be minimized by

$$L = \mathbb{E}\left[(Q_a(s_t, a_t, a_t^j) - y_t^i)^2 \right], \tag{9}$$

where y_t^i is equivalent to:

$$y_t^i = r_{t+1}^i + \gamma \cdot Q_a(s_{t+1}, a_t, a_t^j). \tag{10}$$

C. Self-advice Module

The self-advice module has two parts. First, a 2D map is constructed in the cloud (Phase 1) using the data received by all the agents. Then, the routing and obstacle avoidance algorithm proposed in [41] is adapted (Phase 2). The adaptation is necessary because the agent in this approach cannot perform backward actions, and the original algorithm has to move "backward" several steps when it collides.

Phase 1 2D global mapping created in the cloud with joint data

```
Get agent state
Initialize matrix for free space (m_f)
Initialize matrix for the occupied space (m_o)
Initialize matrix for the temporarily occupied space (m_t)
Initialize constants corresponding to occupied space (c_0), free space (c_f), and temporarily occupied space (c_t)
Split map into segments of the same size
Sets each segment of the map as an empty space
for each Laser do
    Read distance traveled
    Determine the angle of the laser and the set of segments it goes through
    for each segment in the set of segments do
        Read the segment
        if segment == length(set of segments) then
            Add c_o to the current value
        else
            Add c_f to the current value
        end if
    end for
    Assign the values of the segments to the corresponding matrix
    Divide the occupied and free space matrices into each other
    Define whether it corresponds to free, occupied, or temporarily occupied space according to the result of the division
    After every 10 samples, overlay the free and occupied space matrices and send the updated map to each agent
end for
```

D. k-Nearest Neighbors Algorithm

kNN is a simple and low computational time ML algorithm used in classification prediction and regression. This algorithm assigns each input data a certain class, corresponding to its nearest neighbor. To calculate the nearest neighbors, the Euclidean distance (d) is used, i.e., it is calculated between the test sample and the specified training samples [42], [43].

For this paper, a collision controller was designed using kNN (see Fig. 3), and it is defined in Phase 3. Within this module, each agent calculates its future position given the action chosen by its policy. Therefore, it has to be considered that each action has a given linear and angular velocity. In other words, the future positions are calculated considering the current position, the action to be executed, and the linear and angular velocities.

Phase 2 Self-advice (Routing and obstacle avoidance algorithm adaptation)

```
for each step do
    Get agent state
    Get the latest version of the map sent by the cloud
    Update the map with its status and add temporary objects
    Check the agent's current process
    if process is "follow_the_path" then
        Desired angle "determined by A* algorithm"
        action = arg min [heading - desired angle]
    else if process is "orientation_heading" then
        Desired angle "zero "
        action = arg min[heading - desired angle]
    else if process is "driving_straight" then
        action = 2
        if free_distance > target_distance then
            action = 2
        else
            Change process to "follow_the_path"
        end if
    end if
    return action
end for
```

For those cases in which the previous action has not been applied an angular velocity equal to zero, the principle of conservation of angular momentum has to be taken into account. Consequently, to obtain a prediction of a future position very close to the real one, both the linear (v_l) and the angular (v_ω) velocity will be represented as the average of the velocity of the past action $(t-1)$ and the velocity of the chosen action (t), in the following equations:

$$v_l = \frac{v_t + v_{t-1}}{2}, \tag{11}$$

$$v_\omega = \frac{\omega_t + \omega_{t-1}}{2}. \tag{12}$$

Therefore, to calculate future positions with linear velocities different from zero, and angular velocities equal to zero, first the rotation matrix (R_v) of the Yaw angle (ψ) of the agent is calculated. Where ψ represents the rotation of the agent around the vertical axis.

$$\mathbf{R_m} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \tag{13}$$

$$\vec{R_v} = R_m(\theta) \cdot \begin{bmatrix} 1 \\ 0 \end{bmatrix}, \tag{14}$$

$$\theta = -\psi. \tag{15}$$

The next step is calculating the distance (l) traveled during the action time (t_a) using:

$$l = v_l \cdot t_a. \tag{16}$$

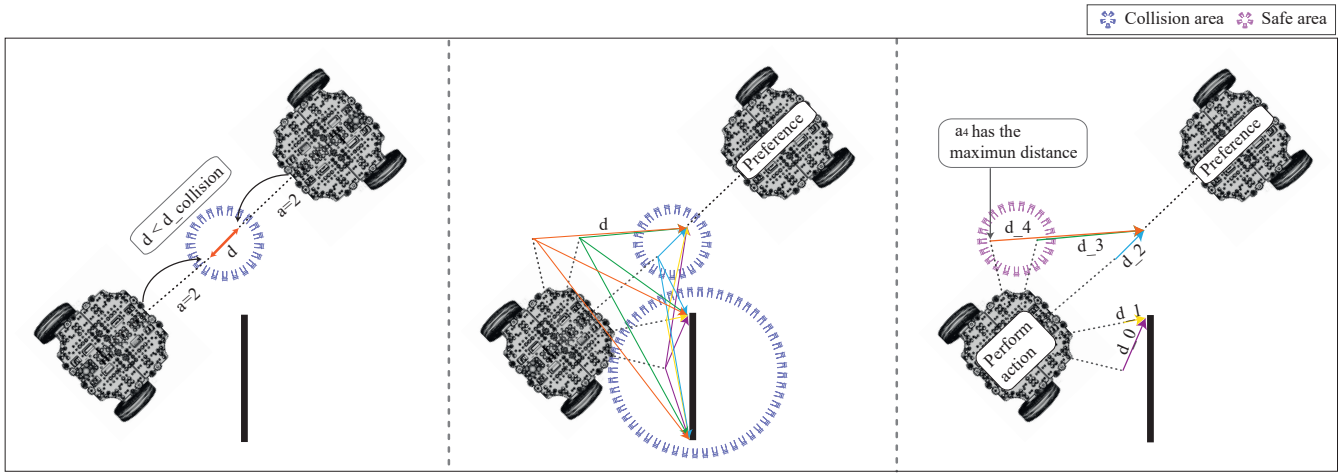

Fig. 3. The collision controller first checks whether the distance (d) between the future positions of the agents will produce a collision. If so, it checks which agent has the preference label and then uses the kNN algorithm to determine which action moves the non-preference agent the largest distance with respect to all collision points.

Phase 3 Collision Controller

```
Initialized variable possible_action = []
for Every step do
    Calculate its own future position
    if action == 2 then
        Calculate new position using Eq. (18)
    else
        Calculate new position using Eq. (19)
    end if
    Send future position to its neighbors
    Each agent read its preference tag
    if tag == True then
        return [False, None]
    else
        Calculate distance to its neighbors
        if Distance is ≤ safe_distance then
            Create an array of possible obstacles using sensor data and the position of the other agent.
            Train kNN using the matrix with possible obstacles
            for Every action do
                Calculate the near neighbor using kNN
                Save in possible_action
            end for
            action = max(possible_action)
            return [True, action]
        else
            return [False, None]
        end if
    end if
    Clear variable possible_action
end for
```

Finally, the position in the next step (p_{t+1}) is defined as:

$$p_t = \begin{bmatrix} p_x \\ p_y \end{bmatrix}, \tag{17}$$

$$p_{t+1} = p_t + lR_v, \tag{18}$$

where p_t represents the current position in x (p_x) and y (p_y).

On the other hand, in the case of non-vanishing angular velocities, the future positions are calculated according to:

$$p_{t+1} = c_r + (R_m(\zeta) \cdot (-R_m(\delta) \cdot R_v)) \cdot r, \tag{19}$$

where ζ represents the angle displaced by the agent during the execution time of the action, δ is the rotation angle with respect to the x-axis, r is the length of the vector (distance traveled), and c_r is the new origin coordinate,

$$\zeta = -v_\omega \cdot t_a, \tag{20}$$

$$\delta = \frac{\pi}{2}, \tag{21}$$

$$r = \frac{l}{\zeta}, \tag{22}$$

$$c_r = p_t + rR_\sigma. \tag{23}$$

IV. Coordinated Algorithm

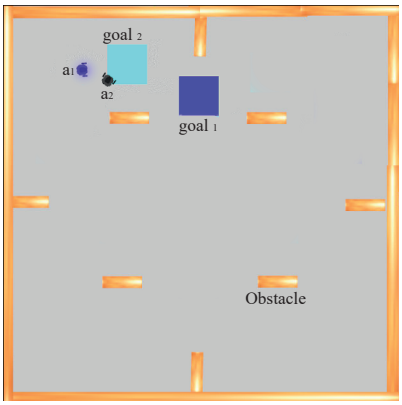

Fig. 4. Environment in which the algorithms were tested consists of two agents on random goals and several obstacles.

A. Observation and Action Space

In order to deploy the proposed algorithm, we have used Robot Operating System (ROS) Melodic and Gazebo 9 simulator. The chosen software is widely used for the deployment of robotic applications as they are open source and provide all the necessary libraries and tools to simulate real scenarios with high fidelity. Additionally, ROS is also compatible with Python and TensorFlow libraries used to program the algorithm.

Regarding the deployment characteristics, all the agents participating in the system have the same characteristics, such as:

- Agent's radius of 105 mm.
- Knowledge of the near agent's future position at each step.
- Each episode ends when the agent completes 500 steps or when it collides.
- Same training environment (see Fig. 4).
- Same objectives.
- Every agent has five possible actions, corresponding to turning left or right or moving forward (Table I).

TABLE I
LINEAR VELOCITY (ν) AND ANGULAR VELOCITY(ω) CORRESPONDING TO THE 5 POSSIBLE ACTIONS

type of action	ν (m/s)	ω (rad/s)
turn	+1.2	$\pm$ 1.5
turn	+1.2	$\pm$ 0.75
forward	+1.5	0

B. Proposed algorithm (CMA-SA&CC)

In summary, to solve the multi-agent cooperation, an off-policy CMA-SA&CC algorithm is proposed (Algorithm 1 and 2). According to this algorithm, a single global policy is trained with the information of all the agents, considering them as a single agent.

This policy will consider not only the action that was executed but also the one recommended by the self-advice module. However, the action to be executed by each agent is chosen using only the information collected by its sensors.

Before the agent executes the action, the collision control module makes sure that it will not cause a collision with a nearby agent in the near future. If this is the case, it will choose another action that moves it away from its neighbor (see Fig. 6).

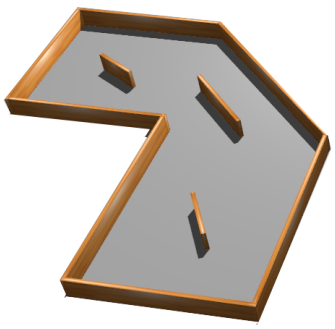

Fig. 5. The second test environment has an irregular shape with diagonal walls and is twice the size of the first one.

Algorithm 1 CMA-SA&CC (Agent)

Initializing the actor-network with random weights (π)
for each episode **do**
  Initialize the environment and set the agent's starting point and goal point
  for each step **do**
    Get action $a = \pi(s_t)$
    Get action a^j from the self-advice module
    new_action = Collision Controller function
    if new_action[0] = True **then**
      action = new_action[1]
    else
      Execute the action chosen by $\pi(s_t)$
    end if
    Temporarily store s_t,a_t,r_t,a_t^j
    Send s_t,a_t,r_t,a_t^j to the cloud
    if Collision status == True **then**
      Request update to the cloud
      Update (π)
    else
      Check for new policies sent from the cloud
      if An update is available **then**
        Update (π)
      end if
    end if
  end for
end for

Algorithm 2 CMA-SA&CC (Cloud)

Initialize learning rate (α), discount factor (γ), update rate (κ), increase factor (δ), tau(τ), time (t)
Initialize the actor networks and critic networks with the same weights
for each step **do**
  Get s_t,a_t,r_t,a_t^j from all the agents
  Store sample in the memory
  Call Phase 1
  Sample Mini-batch
  Actor and critic network training
  if t % κ == zero **then**
    Update critic network (target)
    $\theta\_q' \leftarrow \tau\theta\_q$ + (1-τ)θ_q' with $\tau < 1$
    Update actor-network (target)
    $\theta\_\pi' \leftarrow \tau\theta\_\pi$ + (1-τ)θ_π' with $\tau < 1$
  end if
  if (step % 200) == 0 **then**
    Send 2D map to agents
  end if
  if (step % 20) == 0 **then**
    Send weights (π) to agents
  end if
end for

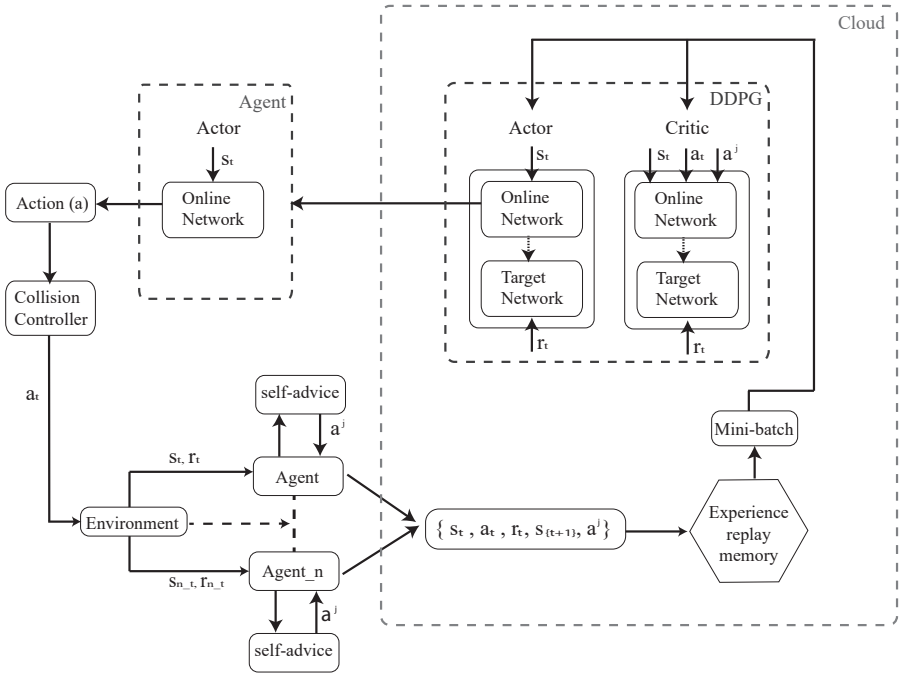

Fig. 6. As it is a semi-decentralized architecture, obtaining an optimal behavior policy is done in the cloud by training the data sent by the agents. This policy is sent to each agent responsible for choosing the action using only its own data. Before executing the action, the collision controller checks that the action does not produce collisions between agents. It will set a different value to the action if the received action produces a collision shortly.

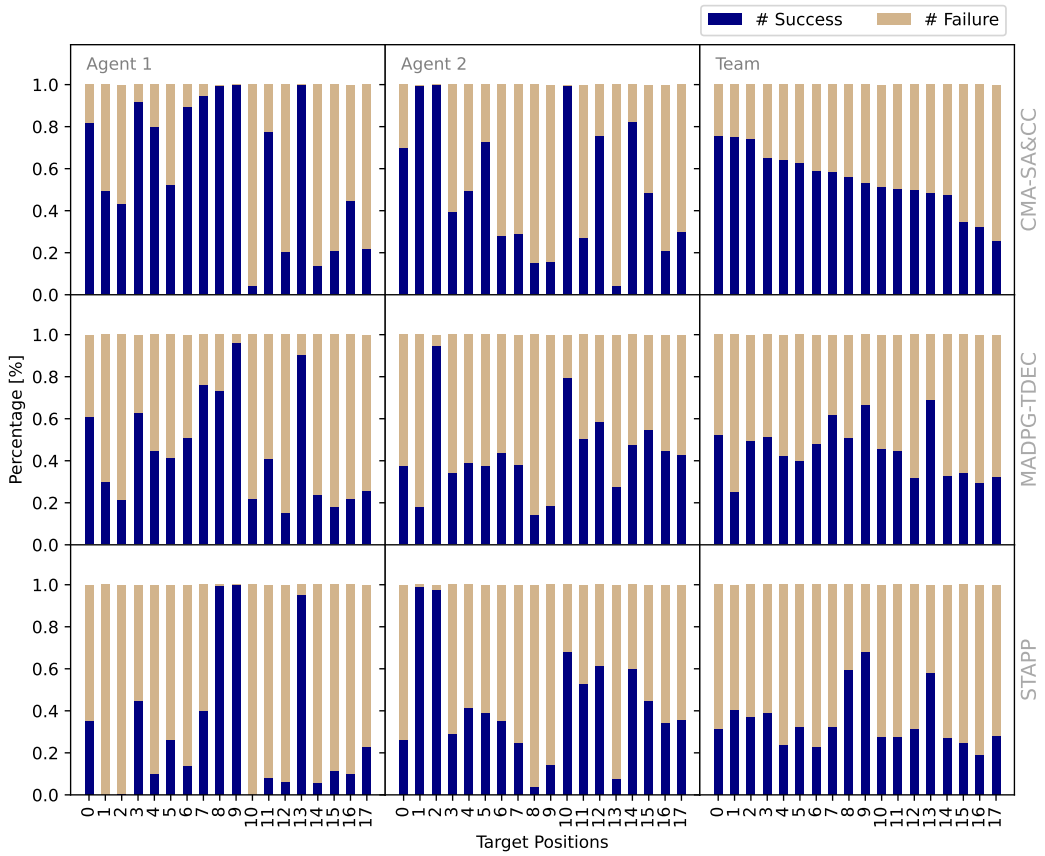

Fig. 7. Each subfigure shows the ratio between successes and failures for each agent and their average as a team during the entire learning process. Where each row represents CMA-SA&CC, MADPG-TDEC, and STAPP, respectively.

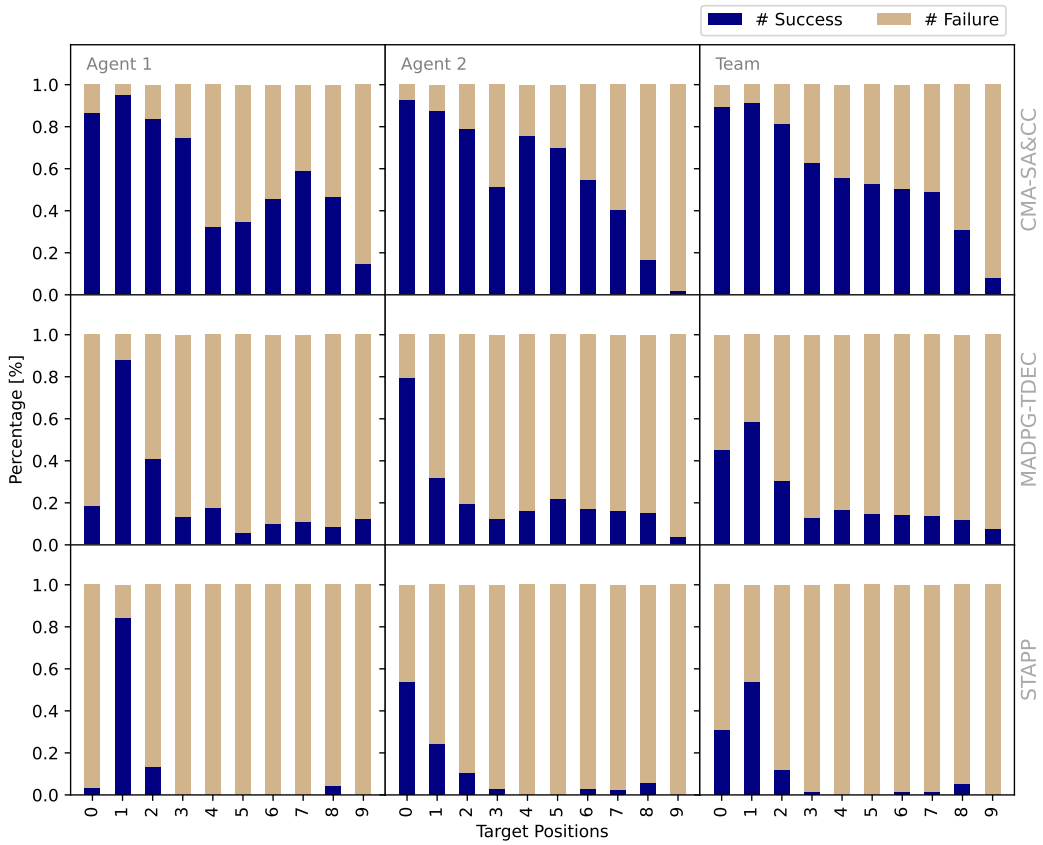

Fig. 8. Each subfigure shows the ratio between successes and failures of the second environment for each agent and their average as a team during the entire learning process. Each row represents CMA-SA&CC, MADPG-TDEC, and STAPP, respectively.

V. Results Discussion

In this section, the performance of the proposed algorithm [1] is compared with two proposals mentioned in Section I-A.

The training and evaluation scenarios (see Fig. 4, Fig. 5) contemplate a team of two agents that have random targets to force more interaction between them.

The first algorithm to be compared is presented in [25] and the second in [24]. In this paper, we will refer to them as MADPG-TDEC and STAPP, respectively.

A. Performance during the training phase

1) First Environment: Note that all the algorithms share the same learning parameters since they use two hidden layers with 512 neurons per layer and a learning rate of 0.001. Furthermore, they were trained in the same environment for 19 hours using the same targets but randomly selected.

As shown in Fig. 7, the performance during the learning process of the proposed algorithm CMA-SA&CC as a team is higher than MADPG-TDEC and STAPP, achieving 54.53%, 44.98%, and 35% of their targets during the whole training, respectively. Moreover, it can be observed that the STAPP algorithm presents the problem called lazy agent since the agent 2 presents a much higher number of successful hits than agent 1. The reason for this behavior is that the policy learned by agent 1 is a collision avoidance policy that always gives priority to agent 2. In this way, it reduces collisions between them and prevents negative rewards for their behavior as a team.

Concerning the other two algorithms, it is observed that the number of objectives reached by each agent is more balanced than in the case of STAPP, so the lazy agent problem is less pronounced in those algorithms.

2) Second Environment: In this setting, the size of the environment is doubled with respect to the first one. In addition, the training time was reduced to 15 hours to determine the performance of the proposed algorithm using less training time and a larger environment. As shown in Fig. 8, the performance during the learning process of the proposed algorithm is the highest as in the first environment. The number of goals achieved by CMA-SA&CC as a team is higher than MADPG-TDEC and STAPP, as they achieved 57.76%, 22.61%, and 11.01% of their targets during the training, respectively. Therefore, reducing the number of training hours and using a larger environment does not affect the performance of our proposed algorithm, but it does affect the performance of the two algorithms used for comparison.

B. Interaction Between Agents

1) First Environment: To analyze the interaction between the agents, the map corresponding to the spaces traversed by

[1] https://github.com/ELIZABETH1611/Cooperative_multi_agent.git

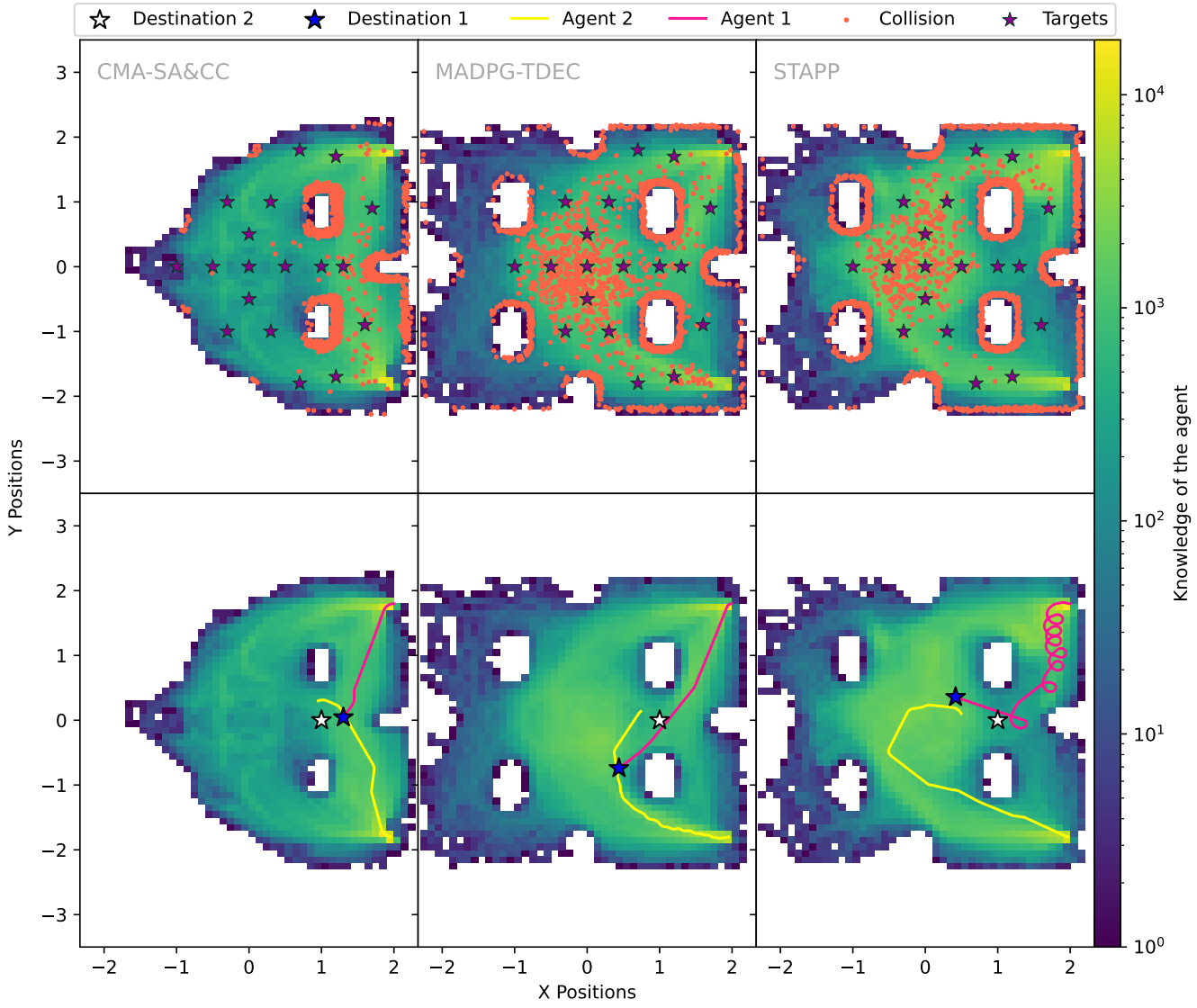

Fig. 9. In the top row of subfigures, the heat map showing the positions the agent traversed during its 19 hours of training and the positions where a collision occurred is shown. The bottom row instead shows the route of each agent to their respective targets using the same time window.

the agents within the environment can be seen in the top row of Fig. 9. The orange dots show the places where a collision occurred during the training, either between agents or with a fixed object in the environment. It can also be observed that most of the targets represented by stars are located in the center of the environment. Therefore, the probability of interaction between agents in this area was higher.

As observed, the proposed algorithm has very few collisions in the central zone of the environment where most targets are located. In contrast, MADPG-TDEC shows a greater number of collisions in that zone. However, STAPP shows the highest number of collisions in the center of the environment compared to the other algorithms. Consequently, it can be deduced that CMA-SA&CC can better handle situations where agents must interact with each other since, in 19 hours, it did not present many collisions in that area. In summary, the proposed algorithm has a total of 427390 collisions, which is 48.94% and 47.44% less than MADPG-TDEC (837078) and STAPP (813172). Since all algorithms had different success rates per target and in order to compare the iteration of the agent 2 over time, a specific target reached by all agents in the last hours of training was defined.

The chosen target corresponds to the point (1,0) indicated as a white star on the map (Fig. 9).

To determine the target corresponding to agent 1, first, the time window in which agent 2 goes from its point of origin to its target is calculated. Then, in that time window, the target to be reached or reached by agent 1 is searched. It is noted that although the target of agent 2 is close to its starting point, the route defined by the MADPG-TDEC and STAPP algorithms presents much longer paths than the one obtained with CMA-SA&CC.

Furthermore, agent 1 in STAPP clearly shows that its policy prioritizes the other agent by performing many incessant turns toward its target. With this type of policy, the interaction between agents will be almost null since one of the agents

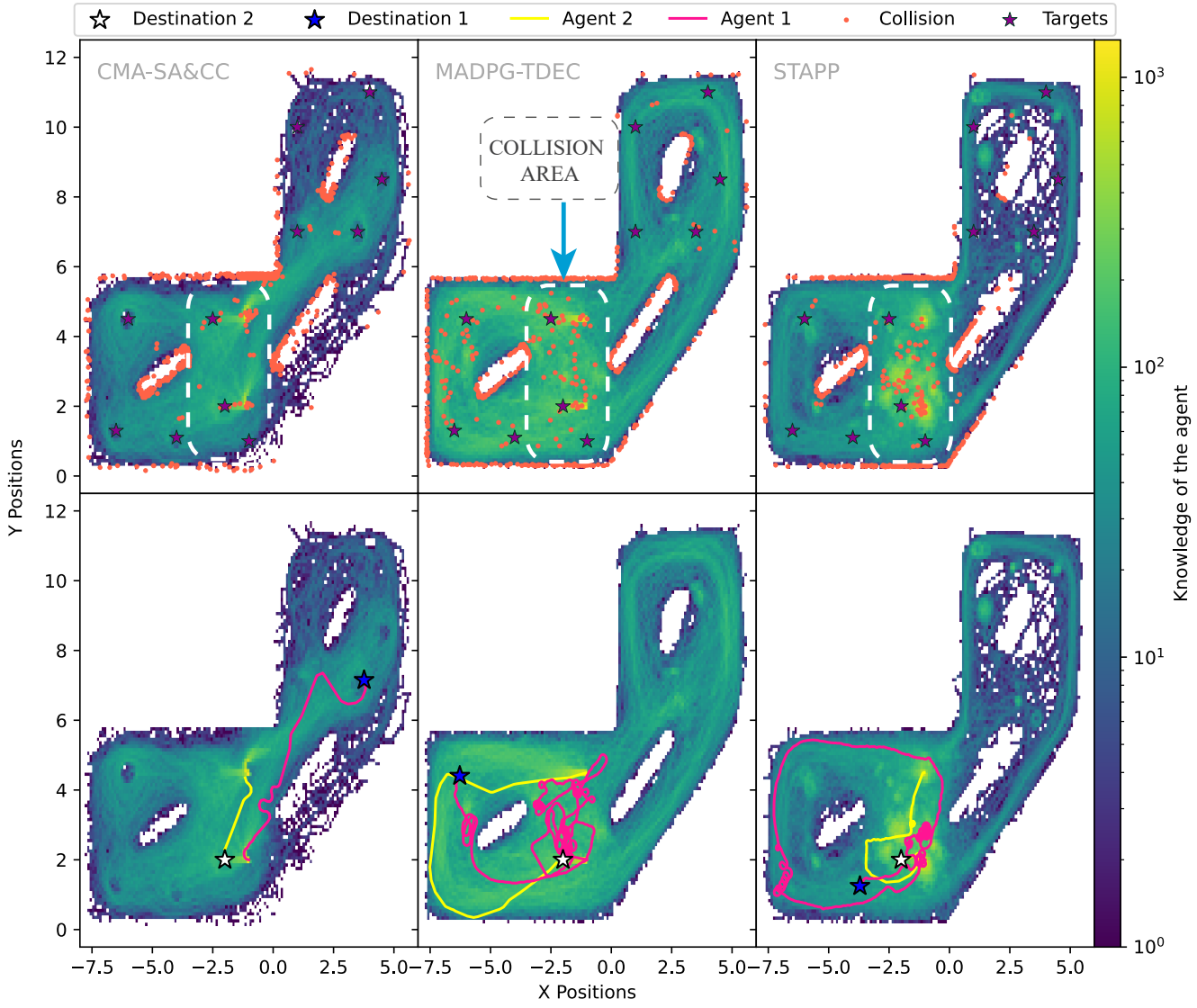

Fig. 10. Second environment: In the top row of subfigures, the heat map showing the positions the agent traversed during its 15 hours of training and the positions where a collision occurred is shown. It also shows the area with the highest probability of collision (white box); this is because the starting position of both agents is in that area, so there is a higher probability of iteration there. In contrast, the bottom row instead shows the route of each agent to their respective targets using the same time window.

prefers to leave the way accessible to the other agent to avoid negative rewards between them.

2) Second Environment: In this environment, the proposed algorithm also presents a lower number of collisions during the area with a higher probability of agent interaction, as can be observed in Fig. 10. This area is located between the starting positions of the agents, which are (-1,2) and (-1,4.5), and it is observed that the highest number of orange dots (collisions) belong to the MADPG-TDEC and STAPP algorithms.

As in the first environment, since all algorithms had different success rates per target, the point (-2,2) is chosen as the target for agent 2, and its interaction with agent 1 is compared over time. Furthermore, it can be concluded from the trajectories traced to their goals that the behavioral policy trained by CMA-SA&CC is optimal, as it allows agents to successfully interact and plan short and smooth trajectories. On the other hand, MADPG-TDEC and STAPP present long and convoluted paths even when the starting point is close to the goal.

C. Performance of the trained policy

1) First Environment performance: In addition, the performance is tested by executing the policy five times for a given time (see Fig. 11, Fig. 12).

In the first one, the position of the agent is (2.0,1.8) and (2.0,-1.8), the same as in the training, while in the second one, the position changes to (0.0,1.5) and (0.0,-1.5), which corresponds to the center of the environment.

As a result, it can be seen that the median obtained in the first set of tests with the proposed algorithm is higher since it presents a value of 67%, while the others present values of 42% and 36%, respectively. In the second set of

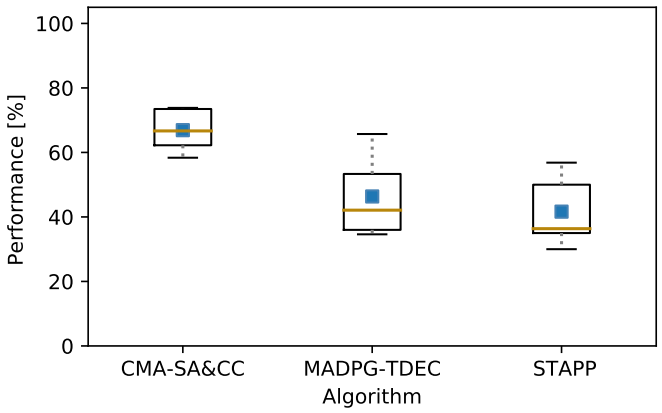

Fig. 11. Performance achieved as a team when agents use the same point used to train the loaded training policy as a starting point.

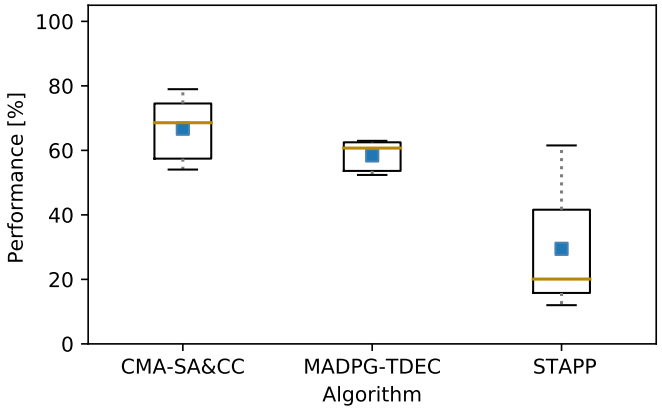

Fig. 12. Performance achieved as a team when agents use a different starting point than the one used to train the loading policy. This point is located in the central zone of the environment where the highest number of targets is found. Thus, the interaction between them is higher.

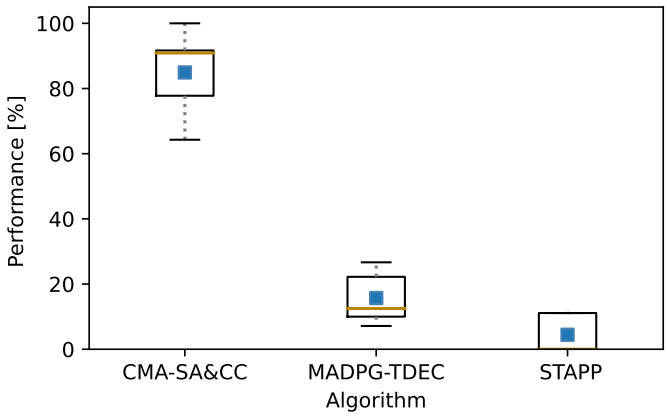

Fig. 13. Performance achieved in the second environment as a team when agents use the same point used to train the loaded training policy as a starting point.

tests, the CMA-SA&CC algorithm continues to show higher performance compared to the other algorithms, reaching a median of 69%, 61%, and 20%, respectively.

2) Second Environment performance: In this environment, the performance was tested using the same agent starting position as during the training process. As a result, it can be observed that the median obtained with the proposed algorithm is superior since it presents a value of 91% performance, while the others present values of 13% and 0.01%, respectively. The performance obtained was as expected since, as shown in Fig. 8, the number of targets achieved by MADPG-TDEC is low, and almost none with STAPP. From this performance during the training phase, we can expect a low or null performance in the testing phase since these algorithms would need to increase the training hours to improve their policy and eventually perhaps reach their optimal policy. Therefore, taking into account the performance obtained during the whole training process and in the test (Fig. 13), it is shown that our cooperative algorithm outperforms the others even when the size of the environment is increased and the training hours are reduced.

VI. Conclusions

Achieving successful multi-agent cooperation represents a significant challenge since each agent aims to achieve different goals in a shared environment without a preloaded map. Therefore, a cooperative multi-agent algorithm with self-advice and priority collision control between near neighbors has been proposed.

The algorithm has been designed with the aim of mitigating the lazy agent problem and achieving the highest possible cooperation between agents. This goal has been completed, as shown in Fig. 7 and 9. Additionally, apart from having the fewest collisions in the area of major interaction between agents, the path traced to its objectives is the best in terms of distance.

Finally, the success rate of the final model of each algorithm has been evaluated five times using random targets for five minutes. Results show a median success rate of 67%, 42%, 36% in the first set of tests and y 69%, 61%, 20% in the second set of tests for CMA-SA&CC, MADPG-TDEC, and STAPP, respectively. Based on the results obtained, we showed that the proposed algorithm performs better in a multi-agent environment. Moreover, its policy is robust since it performs better than the other algorithms, no matter if its starting point differs from the one it was trained in. Therefore, the proposed algorithm outperforms MADPG-TDEC by 25% and STAPP by 31% when evaluated under the same training conditions; even when the agents are in a collaborative environment with a high probability of interaction it outperforms MADPG-TDEC by 8% and STAPP by 49%. It is important to mention that the robustness of our collaborative agent algorithm was validated by the high performance achieved (Fig. 13) in a large environment and with fewer training hours.

As a future line, we propose to use agents with different sensor technologies to create a global policy of collaborative behavior compatible with heterogeneous agents.

Acknowledgments

Elizabeth Palacios's research is funded by the Research and Development Grants Program (PAID-01-19) of the Universitat

Politècnica de València.

REFERENCES

[1] C. Yu, M. Zhang, F. Ren, and G. Tan, "Emotional Multiagent Reinforcement Learning in Spatial Social Dilemmas," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 12, pp. 3083–3096, 2015.

[2] S. Zheng and H. Liu, "Improved Multi-Agent Deep Deterministic Policy Gradient for Path Planning-Based Crowd Simulation," *IEEE Access*, vol. 7, pp. 147 755–147 770, 2019.

[3] J. Lu, L. Han, Q. Wei, X. Wang, X. Dai, and F.-Y. Wang, "Event-Triggered Deep Reinforcement Learning Using Parallel Control: A Case Study in Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 4, pp. 2821–2831, 2023.

[4] Y. Yang, Y. Wen, J. Wang, L. Chen, K. Shao, D. Mguni, and W. Zhang, "Multi-Agent Determinantal Q-Learning," in *Proceedings of the 37th International Conference on Machine Learning*, vol. 119, 2020, pp. 10 757–10 766.

[5] G. Wen, J. Fu, P. Dai, and J. Zhou, "DTDE: A New Cooperative Multi-Agent Reinforcement Learning Framework," *The Innovation*, vol. 2, no. 4, 2021.

[6] S. Aradi, "Survey of Deep Reinforcement Learning for Motion Planning of Autonomous Vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 740–759, 2022.

[7] M. Liu, F. Zhao, J. Niu, and Y. Liu, "Reinforcementdriving: Exploring trajectories and navigation for autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 22, no. 2, pp. 808–820, 2021.

[8] J. Yu, J. A. Vincent, and M. Schwager, "DiNNO: Distributed Neural Network Optimization for Multi-Robot Collaborative Learning," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 1896–1903, 2022.

[9] S. Omidshafiei, D.-K. Kim, M. Liu, G. Tesauro, M. Riemer, C. Amato, M. Campbell, and J. P. How, "Learning to Teach in Cooperative Multiagent Reinforcement Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6128–6136.

[10] X. Lin, S. C. Adams, and P. A. Beling, "Multi-Agent Inverse Reinforcement Learning for Certain general-Sum Stochastic Games," *Journal of Artificial Intelligence Research*, vol. 66, pp. 473–502, 2019.

[11] L. Peshkin, K.-E. Kim, N. Meuleau, and L. P. Kaelbling, "Learning to Cooperate Via Policy Search," *arXiv preprint cs/0105032*, 2001.

[12] D. Lee, N. He, P. Kamalaruban, and V. Cevher, "Optimization for Reinforcement Learning: From a single agent to cooperative agents," *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 123–135, 2020.

[13] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls *et al.*, "Value-Decomposition Networks for Cooperative Multi-Agent Learning," *arXiv preprint arXiv:1706.05296*, vol. 8, 2017.

[14] L. Kraemer and B. Banerjee, "Multi-agent Reinforcement Learning as a Rehearsal for Decentralized Planning," *Neurocomputing*, vol. 190, pp. 82–94, 2016.

[15] Z. Zhang, D. Zhao, J. Gao, D. Wang, and Y. Dai, "FMRQ—A Multiagent Reinforcement Learning Algorithm for Fully Cooperative Tasks," *IEEE Transactions on Cybernetics*, vol. 47, no. 6, pp. 1367–1379, 2017.

[16] J. Chai, W. Li, Y. Zhu, D. Zhao, Z. Ma, K. Sun, and J. Ding, "UNMAS: Multiagent Reinforcement Learning for Unshaped Cooperative Scenarios," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–12, 2021.

[17] H.-R. Lee and T. Lee, "Improved Cooperative Multi-agent Reinforcement Learning Algorithm Augmented by Mixing Demonstrations from Centralized Policy," in *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, 2019, pp. 1089–1098.

[18] Z. Wang, Z. Wang, and C. Chen, "A Coordinated Multiagent Reinforcement Learning Method Using Chicken Game," in *2020 IEEE International Conference on Networking, Sensing and Control (ICNSC)*, 2020, pp. 1–6.

[19] Z. Zhang, D. Wang, and J. Gao, "Learning Automata-Based Multiagent Reinforcement Learning for Optimization of Cooperative Tasks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4639–4652, 2021.

[20] Y. J. Park, Y. J. Lee, and S. B. Kim, "Cooperative Multi-Agent Reinforcement Learning With Approximate Model Learning," *IEEE Access*, vol. 8, pp. 125 389–125 400, 2020.

[21] Z. Zhang, L. Zheng, Z. Chen, L. Kong, and H. R. Karimi, "Mutual-Collision-Avoidance Scheme Synthesized by Neural Networks for Dual Redundant Robot Manipulators Executing Cooperative Tasks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 3, pp. 1052–1066, 2021.

[22] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang, "QPLEX: Duplex Dueling Multi-Agent Q-Learning," *arXiv preprint arXiv:2008.01062*, 2020.

[23] C. Song, Z. He, and L. Dong, "A Local-and-Global Attention Reinforcement Learning Algorithm for Multiagent Cooperative Navigation," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022.

[24] H. Qie, D. Shi, T. Shen, X. Xu, Y. Li, and L. Wang, "Joint Optimization of Multi-UAV Target Assignment and Path Planning Based on Multi-Agent Reinforcement Learning," *IEEE Access*, vol. 7, pp. 146 264–146 272, 2019.

[25] C. Sun, W. Liu, and L. Dong, "Reinforcement Learning With Task Decomposition for Cooperative Multiagent Systems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 5, pp. 2054–2065, 2020.

[26] Y. Huang, J. Du, Z. Yang, Z. Zhou, L. Zhang, and H. Chen, "A Survey on Trajectory-Prediction Methods for Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 7, no. 3, pp. 652–674, 2022.

[27] K. Min, H. Kim, and K. Huh, "Deep Distributional Reinforcement Learning Based High-Level Driving Policy Determination," *IEEE Transactions on Intelligent Vehicles*, vol. 4, no. 3, pp. 416–424, 2019.

[28] X. He, H. Yang, Z. Hu, and C. Lv, "Robust Lane Change Decision Making for Autonomous Vehicles: An Observation Adversarial Reinforcement Learning Approach," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 184–193, 2023.

[29] J. Wu, Z. Huang, and C. Lv, "Uncertainty-Aware Model-Based Reinforcement Learning: Methodology and Application in Autonomous Driving," *IEEE Transactions on Intelligent Vehicles*, vol. 8, no. 1, pp. 194–203, 2023.

[30] B. Peng, M. F. Keskin, B. Kulcsár, and H. Wymeersch, "Connected autonomous vehicles for improving mixed traffic efficiency in unsignalized intersections with deep reinforcement learning," *Communications in Transportation Research*, vol. 1, p. 100017, 2021.

[31] H. Ding, W. Li, N. Xu, and J. Zhang, "An enhanced eco-driving strategy based on reinforcement learning for connected electric vehicles: cooperative velocity and lane-changing control," *Journal of Intelligent and Connected Vehicles*, vol. 5, no. 3, pp. 316–332, 2022.

[32] Y. Liu, F. Wu, C. Lyu, S. Li, J. Ye, and X. Qu, "Deep dispatching: A deep reinforcement learning approach for vehicle dispatching on online ride-hailing platform," *Transportation Research Part E: Logistics and Transportation Review*, vol. 161, p. 102694, 2022.

[33] X. Chu and H. Ye, "Parameter Sharing Deep Deterministic Policy Gradient for Cooperative Multi-agent Reinforcement Learning," *arXiv preprint arXiv:1710.00336*, 2017.

[34] S. Li, Y. Wu, X. Cui, H. Dong, F. Fang, and S. Russell, "Robust Multi-Agent Reinforcement Learning via Minimax Deep Deterministic Policy Gradient," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 4213–4220.

[35] Z. Zhang, D. Wang, and J. Gao, "Learning Automata-Based Multiagent Reinforcement Learning for Optimization of Cooperative Tasks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4639–4652, 2020.

[36] W. Jiang, G. Feng, S. Qin, T. S. P. Yum, and G. Cao, "Multi-Agent Reinforcement Learning for Efficient Content Caching in Mobile D2D Networks," *IEEE Transactions on Wireless Communications*, vol. 18, no. 3, pp. 1610–1622, 2019.

[37] C. Qiu, Y. Hu, Y. Chen, and B. Zeng, "Deep Deterministic Policy Gradient (DDPG)-Based Energy Harvesting Wireless Communications," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 8577–8588, 2019.

[38] J. K. Gupta, M. Egorov, and M. J. Kochenderfer, "Cooperative Multi-agent Control Using Deep Reinforcement Learning," in *AAMAS Workshops*, 2017.

[39] C. E. Mariano and E. F. Morales, "DQL: A New Updating Strategy for Reinforcement Learning Based on Q-Learning," in *Machine Learning: ECML 2001*, L. De Raedt and P. Flach, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 324–335.

[40] R. Pina, V. D. Silva, J. Hook, and A. Kondoz, "Residual Q-Networks for Value Function Factorizing in Multiagent Reinforcement Learning," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–11, 2022.

[41] E. Palacios-Morocho, S. Inca, and J. F. Monserrat, "Multipath Planning Acceleration Method With Double Deep R-Learning Based on a Genetic Algorithm," *IEEE Transactions on Vehicular Technology*, pp. 1–16, 2023.

[42] H. Samet, "K-Nearest Neighbor Finding Using MaxNearestDist," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 30, no. 2, pp. 243–252, 2008.

[43] K. Taunk, S. De, S. Verma, and A. Swetapadma, "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification," in *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*, 2019, pp. 1255–1260.

VII. Biography Section

Elizabeth Palacios-Morocho Ing. Elizabeth Palacios-Morocho received her MSc. in Systems Technologies and Communication Networks from the Universitat Politècnica de València (UPV) in 2019, and her degree in Electronics and Telecommunications Engineering from the University National of Loja (UNL), in 2018. She is currently participating as a predoctoral researcher at the Universitat Politècnica de València at the Institute of Telecommunications and Multimedia Applications (iTEAM-UPV).

Saúl Inca Dr.-Ing. Saúl Inca received his MSc. in Systems Technologies and Communication Networks and his Ph.D. in Telecommunications Engineering from the Universitat Politècnica de València (UPV) in 2014 and 2019. He is currently participating as a postdoctoral researcher in European Projects at the Institute of Telecommunications and Multimedia Applications (iTEAM-UPV). His main research areas are radio channel modeling and developing visualization platforms for 5G and B5G networks.

Jose F. Monserrat Dr.-Ing. Jose F. Monserrat received his MSc. degree with High Honors and Ph.D. degree in Telecommunications Engineering from the Universitat Politècnica de València (UPV) in 2003 and 2007, respectively. He was the recipient of the First Regional Prize of Engineering Studies in 2003 for his outstanding student record, also receiving the Best Thesis Prize from the UPV in 2008. In 2009 he was awarded the best young researcher prize in Valencia. In 2016 he received the merit medal from the Spanish Royal Academy of Engineering in the young researcher category. Jose Monserrat is a senior member of the IEEE, holds six patents, and has published more than 50 journal papers.