Accepted Manuscript

Obtaining Fault Tolerance Avoidance Behavior using Deep Reinforcement Learning

Fidel Aznar, Mar Pujol, Ramn Rizo

PII: S0925-2312(19)30142-0
DOI: <https://doi.org/10.1016/j.neucom.2018.11.090>
Reference: NEUCOM 20413

To appear in: *Neurocomputing*

Received date: 29 June 2018
Revised date: 23 October 2018
Accepted date: 21 November 2018

Please cite this article as: Fidel Aznar, Mar Pujol, Ramn Rizo, Obtaining Fault Tolerance Avoidance Behavior using Deep Reinforcement Learning, *Neurocomputing* (2019), doi: <https://doi.org/10.1016/j.neucom.2018.11.090>

This is a PDF file of an unedited manuscript that has been accepted for publication. As a service to our customers we are providing this early version of the manuscript. The manuscript will undergo copyediting, typesetting, and review of the resulting proof before it is published in its final form. Please note that during the production process errors may be discovered which could affect the content, and all legal disclaimers that apply to the journal pertain.

Obtaining Fault Tolerance Avoidance Behavior using Deep Reinforcement Learning[☆]

Fidel Aznar*, Mar Pujol*, Ramn Rizo*

Abstract

In this article, a mapless movement policy for mobile agents, designed specifically to be fault-tolerant, is presented. The provided policy, which is learned using deep reinforcement learning, has advantages compared to the usual mapless policies: this policy is capable of handling a robot even when some of its sensors are broken. It is an end-to-end policy based on three neuronal models capable not only of moving the robot and maximizing the coverage of the environment but also of learning the best movement behavior to adapt it to its perception needs. A custom robot, for which none of the readings of the sensors overlap each other, has been used. This setup makes it possible to determine the operation of a robust failure policy, since the failure of a sensor unequivocally affects the perceptions. The proposed system exhibits several advantages in terms of robustness, extensibility and utility. The system has been trained and tested exhaustively in a simulator, obtaining very good results. It has also been transferred to real robots, verifying the generalization and the good functioning of our model in real environments.

Key words: Deep Reinforcement Learning, Obstacle Avoidance, Fault Tolerance

1. Introduction

One of the key challenges in building robust autonomous navigation systems is the development of a strong intelligence pipeline that is able to efficiently gather incoming sensor data and take suitable control actions with good repeatability and fault tolerance. In the past, this challenge was addressed in a modular fashion, where specialized algorithms were developed for each subsystem and integrated with fine tuning. More recent trends reveal a revival of end-to-end approaches [1, 2, 3] that learn complex mappings directly from the input to the output by leveraging large volumes of task-specific data and the remarkable abstraction abilities afforded by deep neural networks [3].

Autonomous robots have been employed for applications such as military operations, surveillance, security, mining operations, and planetary exploration. In these application domains, the

[☆]This work has been supported by the Ministerio de Economía y Competitividad (Spain), project TIN2013-40982-R (project co-financed with FEDER funds) and by the Erasmus+ EUROBOTIQUE Project with reference 2016-1-ES01-KA201-024990

*Department of Computer Science and Artificial Intelligence at University of Alicante, San Vicent del Raspeig, Alicante (E-03080), Spain

Email address: {fidel,mar,rizo}@dccia.ua.es (Fidel Aznar, Mar Pujol, Ramn Rizo)

Preprint submitted to Neurocomputing

February 2, 2019

robot must operate in unknown environments where human intervention is expensive, slow, unreliable, or impossible. It is therefore essential for the robots to be fault-tolerant and independently utilize the remaining functional capabilities to automatically overcome the limitations imposed by failures [4]. In this way, there are several investigations regarding self-adaptive behaviors oriented to use the sensorial capabilities that remain in the agent in the best possible way. A self-adaptive approach implies that even with reduced sensor functionality, it is possible to carry on with mission objectives, by making use of what sensor functionality is still available [5].

Furthermore, at present, low-cost methods, such as WiFi localization and visible-light communication, provide lightweight solutions for mobile robot localization. Thus, mobile robots are able to obtain the real-time target position with respect to the robot coordinate frame. In this way, the navigation tasks are simplified by not needing a global map, thus making mapless policies feasible [6].

One advantage of an intrinsic, end-to-end approach is that actions are not divorced from representation but rather learnt together, thus ensuring that task-relevant features are present in the representation. However, the massive difference between the structural simulation environment and the highly complicated real-world environment is the central challenge of transferring the trained model to a real robot directly, and this difference must be taken into account.

The ability to navigate efficiently within an environment is fundamental for intelligent behaviors. Whereas conventional robotics methods, such as simultaneous localization and mapping (SLAM), address navigation through an explicit focus on position inference and mapping, recent works in deep reinforcement learning propose that navigational abilities could emerge as the byproduct of an agent learning a policy that maximizes reward [7].

Learning-based methods have demonstrated clear advantages in controlling robot tasks, such as their information fusion abilities, strong robustness, and high accuracy [8]. Recently, researchers have made significant progress in combining advances in deep learning for learning feature representations with reinforcement learning (RL) [9, 10]. One of the most notable examples is the training of several agents capable of playing Atari games, taking the pixels of the games as input. These agents have been able to develop human control skills in several games [11].

Motivated by the success of the application of RL to various problems and given its good performance in its recent application to robotic navigation tasks [7], we have decided to analyze the use of deep reinforcement learning strategies to learn a mapless policy capable of avoiding obstacles while maximizing the coverage of an environment. This policy must be tolerant to failures such that an error in the sensory system does not limit its operation.

The main contributions of this work can be summarized as follows:

- Two policies for developing obstacle avoidance and covering behaviors will be presented. For both policies, the modeling of the system will be analyzed as a Markov decision process. In addition, a specific reward function that coherently guides learning through RL for the proposed task will be provided.
- The first policy will develop a classic end-to-end strategy using a simple neural network model. The policy will be compared to a Braitenberg vehicle to assess its performance. In addition, this policy will be used as a baseline to compare its operation taking into account various sensory failures.
- A second policy that separates in three modules the traditional sensor-motor mechanism used in the end-to-end policies will be provided. This policy will allow not only obtaining

the best movement policy but also adapting the agent's perception to the environment and helping to reuse parts of the learned model.

- This policy will be able to fuse the sensors' perceptions to integrate them into a finite number of robot recommendation vectors.
- Both policies will be compared using robots with specific sensory failures. We consider that must be no overlap between sensors for these failures to be decisive. In this way, a robot with only three sensors located on the sides and front of the robot will be used.
- Finally, policy transfer tests of our fault-tolerant policy (obtained through a simulator) will be performed in a real environment.

This paper will be divided in the following sections. First, the state of the art regarding fault tolerance in robotics and the use of reinforcement learning to obtain obstacle avoidance policies will be analyzed. Second, the problem and the robotic platform to be used will be introduced. Next, the first movement policy will be designed, and its operation will be discussed. This policy will be improved in the next section with a more complex neuronal model, aimed at increasing its robustness to failures. Finally, we will analyze the robustness of both policies, and the obtained results will be discussed.

2. Related Work

In this section, the state of the art of two main topics related to this paper will be reviewed. On the one hand, fault tolerance in the field of mobile robotics will be studied. On the other hand, the current deep reinforcement learning strategies will be reviewed in terms of the development of obstacle avoidance behaviors.

2.1. Fault Tolerance and Uncertainty

In the last decade, research regarding fault-tolerant systems has become increasingly popular, with several modeling approaches being developed, including fault detection and diagnosis (FDD), fault-tolerant control (FTC) and fault detection estimation (FDE)[12].

Several researchers have recently investigated mobile robots reliability and fault tolerance. In [5], an autonomic 'self-adaptive' approach is presented; with this approach, even with reduced sensor functionality, it is possible to carry on with the mission objectives by making use of what sensor functionality is still available. This approach is based mainly on the development of a model that allows the agent, based on the readings of nearby sensors, to determine the proper functioning of a particular sensor. This model requires overlap between sensors.

A fuzzy approximator is designed in [13] to approximate the uncertain system dynamics such that the controller does not require prior knowledge of the bounds on the uncertainties and faults. The proposed system is able to drive the attitude control of a spacecraft.

In [14], the authors provide useful insights regarding the use of fault detection and diagnosis (FDD) approaches that best suit the different characteristics of robotic system. Robotics poses unique requirements that are very challenging for traditional FDD approaches. Some papers establish FDD strategies as a modeling and learning process through deep learning [15], with many advantages relative traditional modeling [16].

2.2. Navigation with Deep Reinforcement Learning

Although there are many works concerning navigation [17, 18] and control [19, 20] in robotics, we will focus exclusively on navigation strategies using deep reinforcement learning (RL).

Deep RL has recently been used in the navigation domain [7]. In this way, several works have been developed to provide end-to-end systems capable of performing navigation tasks even using multidimensional data as input. For example, [21] presents a system capable of developing coherent navigation tasks in 3D environments using a pretrained ResNet network.

[22] investigated the performance of a variety of networks with external memory on simple navigation tasks in the Minecraft 3D block world environment. [23] trained a convnet DQN-based agent using depth channel inputs for obstacle avoidance in 3D environments.

In [16], a learned policy for obstacle avoidance in unmanned marine vessel is presented. In this case, the presented policy is able to navigate even with unknown environmental disturbances. The final policy is obtained using deep reinforcement learning with very good performance.

3. Problem formulation

Motivated by the proposal of [14], in which FDD strategies are established as a modeling and learning process using reinforcement learning, and also by the current advances in this field, we propose in this work to design and develop a robust and fault-tolerant policy.

As previously mentioned, we want to obtain behaviors able to move a robot avoiding obstacles in the environment. Moreover, those behaviors that tend to cover a greater area will be rewarded, which means that apart from the obstacle avoidance behavior, we will also reward mapless behaviors that are more suitable to cover the entire environment.

The collision avoidance problem is defined in the context of a nonholonomic differential drive robot moving on the Euclidean plane with obstacles. The robot is the main component that interacts with an environment, which is in our case unknown. In each step t , the robot observes its state $s_t \in S$. We will assume that the robot will characterize this state using only three ultrasonic sensors and a compass.

Once the environment is perceived, an action $a \in A$ is selected from the possible ones to make a transition between a state s_t and the following one s_{t+1} . For each transition, the agent will immediately receive a reward, such that $r_t = r(s_t, a_t, s_{t+1}) \in \mathbb{R}$. This is the process that the system will follow to handle the robot in each step t , providing actions that maximize the total reward of the behavior. We can assume that transitions between states satisfy the Markovian property:

$$P(s_{t+1}|s_t, a_t, s_{t-1}, a_{t-1}, \dots) = P(s_{t+1}|s_t, a_t)$$

In this way, we can consider this process of covering and avoiding obstacles as a discrete-time Markov decision process (MDP). The agent's goal is to learn a π policy, which is determined stochastically as $\pi(a|s) = P(a|s)$ or deterministically as $a = \mu(s)$. This π policy should maximize the reward of all steps taken during the h_π history of the agent:

$$R(h_\pi) = \sum_{t=1}^T \gamma^{t-1} r(s_t, a_t, s_{t+1})$$

Therefore, the optimal policy is that which maximizes

Figure 1: Representation of the differential drive robot located at the position $R_{x,y}$ and angle R_α used in this article. As can be observed, the robot has three ultrasonic range sensors (u_1, u_2, u_3), a compass sensor θ and two actuators (v_l, v_r). The ultrasonic sensors have 40° of angular coverage and 4 meters of detection distance. The characteristic values of a differential motor are also shown (R_{ICC}, R_ω, R_R).

$$\pi^* = \arg \max_{\pi} R(h_\pi)$$

The general framework of obstacle avoidance is demonstrated in Figure 1. A differential drive robot with two actuators (v_l, v_r), three ultrasonic sensors (u_1, u_2, u_3) and one compass sensor is presented. Each ultrasonic sensor covers a range of 40 degrees and 4 meters of distance. This is a very limited robotic platform, where no sensor overlaps the sensing area, so any sensory failure is highly penalized. The robot is positioned at $R_{x,y}$ with angle R_α .

A differential drive consists of two drive wheels mounted on a common axis, where each wheel can independently being driven either forward or backward. With this configuration a robot must rotate (at a rate ω) about a point that lies along their common left and right wheel axes, called the instantaneous center of curvature (R_{ICC}). By varying the velocities of the two wheels, we can vary the trajectories that the robot takes. There are three interesting cases with these kinds of drives: if $v_l = v_r$, then the robot develops a forward linear motion in a straight line. If $v_l = v_r$, then $R = 0$, and we have rotation about the midpoint of the wheel axis. If $v_l = 0$, then we have rotation about the left wheel (which is symmetric with v_r). At time t , we can predict the next position of the robot $t + \delta t$:

$$\begin{bmatrix} R'_x \\ R'_y \\ R'_\alpha \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_x - R_{ICCx} \\ R_y - R_{ICCy} \\ R_\alpha \end{bmatrix} + \begin{bmatrix} R_{ICCx} \\ R_{ICCy} \\ \omega\delta t \end{bmatrix}$$

We can describe the position of any robot capable of moving in a particular direction $\alpha(t)$ at a given velocity $V(t)$ as

$$R_x(t) = \int_0^t V(t) \cos[\alpha(t)] dt$$

$$R_y(t) = \int_0^t V(t) \sin[\alpha(t)] dt$$

$$R_\alpha(t) = \int_0^t \omega(t) dt$$

Unfortunately, a differential drive robot imposes nonholonomic constraints on establishing its position, so answering the question of how to control the robot to reach a given configuration is not trivial (we cannot simply specify a robot pose and find the velocities to move to this position). This consideration motivates the classical strategy of moving the robot in a straight line, rotating for a turn in place, and then moving straight again, repeating this process until the goal is reached. Next, we will see an end-to-end policy that will develop continuous movements, away from this classical approach.

4. Design of the end-to-end avoidance behavior

In this section, we will focus on learning an end-to-end policy for robot control. Once learned, it will be evaluated under ideal conditions. This approach will allow us to obtain a baseline policy to compare with our fault-tolerant approach.

Given the sensory readings and the status of the robot, we want to obtain what action is needed to avoid collisions and try to cover most of the environment. A mapless end-to-end behavior will receive the robot's sensors as an input and output the commands required by its actuators.

4.1. Reinforcement Learning Setup

Reinforcement learning has been widely applied in robotic tasks [24]. In [25], a detailed explanation of its operation and application to obtaining robotic control policies is shown.

A reinforcement learning problem contains various integral components, such as observations, actions, and rewards. Below, we describe the details of each of these parts.

4.1.1. Observation Space

As previously mentioned, the robot has three ultrasonic sensors. For this policy, a state will be determined exclusively by the three readings obtained by these sensors at a given moment, their gradient from the previous instant, the speed of each of the actuators of the robot obtained through their respective encoders, and an environment covering metric that is $s_t = (o_t^{1..3}, \nabla o_t^{1..3}, vl_t, vr_t, \lambda)$. The readings will not be taken continuously but rather will establish a movement development time t_{mov} while the robot will execute the action indicated by the policy. The compass readings in this case are considered negligible, since training and tests are performed in different random environments and therefore the correlation between directions and reading is nonexistent.

4.1.2. Action Space

The action space is a set of permissible velocities in continuous space. In our case, we will develop an end-to-end policy that is able to directly control the speed of each of the robot's wheels. In this way, $a_t = (vl_t, vr_t)$, where $vl_t \in \mathbb{Z}$ is the speed of the left wheel of the robot at the time t and $vr_t \in \mathbb{Z}$ is the respective speed for the right wheel. The range of these speeds is defined by the used motor. In our case, it will be an integer value in the range of $[-50, 50]$.

4.1.3. Network Architecture

Given an observation entry, that characterizes the state of the robot s_t and its output a_t , we can define the network that will develop the policy to generate a_t from s_t .

Thus, a four-layer network has been designed (see Figure 2). We have selected a prediction network that directly emits the speed of each of the robot wheels. Although in the case of using gradient-based strategies, the regression networks suffer from a greater training complexity compared to classification networks, we have not found any difficulty in training it using the CMAES algorithm (CMAES will be introduced in subsection 5.2).

Figure 2: Neural model used in this policy. It is a 4-layer regression network in which the inputs are the readings of the ultrasonic sensors and their gradient, along with the reading of the encoders of the robot wheels. The output of the network are two real values that determine the speed of each of the wheels.

The first layer receives the perceptions of the robot ($o_t^{1..3}, \nabla o_t^{1..3}, vl_t, vr_t$). The covering metric λ is not included in the network, since it is not directly accessible by the robot because it is a mapless policy.

It is a fully connected layer with 32 hyperbolic tangent function activation units. This layer connects to another 16 fully connected layer with the same activation function. The process is repeated for another layer, which is the one that connects with the output layer, formed by two linear neurons, which will be responsible for developing the speed of each of the wheels of the robot (vl_t, vr_t).

4.1.4. Reward

Reinforcement learning algorithms require an instruction-centered feedback signal, which makes it difficult to design an appropriate reward function. The reward function is designed such that $r(s_t, a_t, s_{t+1})$. This reward signal must guide two subtasks: on the one hand, we must develop an obstacle avoidance subtask to be able to navigate correctly through the environment. On the other hand, we want to cover as much of the environment as possible, rewarding policies with greater navigability.

We want to obtain a mapless policy; therefore, the robot cannot be located on the map, and it cannot use its λ observation in a simple way (the robot cannot know where it is, and therefore it cannot know what areas are still to be visited or what areas it has already covered). This means that for this subtask, we cannot establish the classic reward policy in map covering systems. Even so, we know that different obstacle avoidance strategies can unleash substantially different map coverings. That is why our reward function includes in its calculation the covering matrix λ : those avoidance policies that achieve a greater generic coverage will be positively evaluated.

In this way, we propose the following reward function:

$$r_t = rf_t \times \sum_x \sum_y \lambda(x, y)$$

where rf_t rewards the combinations of velocities that perform advances in the robot and $\lambda(x, y) \in \{0, 1\}$ is our covering metric, a discrete boolean matrix that is used to reward behaviors that provide greater coverage. This boolean matrix discretizes the environment into a determined number of cells $|\lambda|$, setting the component $(x, y) = 1$ as long as the robot has passed through this position. In this way, as is observed in this reward function, those agents that have traveled most of the environment will receive a higher reward per iteration.

For a complex task such as robotic obstacle avoidance, we require a function reward specifically designed to positively assess several states and actions. In this way, several objectives and limitations in the design of the goals required within this evaluation function have been taken into account.

We will value that the left and right wheels turn in opposite directions and that the absolute difference in the speed between the two wheels is reduced. These two factors help us to develop coherent behaviors that produce progress. Although the speeds that will be sent to the robot will always be in the appropriate range, there are displacements at very slow speeds that do not produce advancement. In this way, we prefer that the wheels turn at least at the minimum speed required by the engine to develop an acceptable movement. These criteria are reflected below:

$$\begin{aligned} rf_t^1 &= (vr_t \cdot vl_t) > 0 \wedge vr_t > 0 \\ rf_t^2 &= |vr_t - vl_t| < v_{err} \\ rf_t^3 &= vr_t > v_{min} \wedge vl_t > v_{min} \\ rf_t &= \begin{cases} 1 & rf_t^1 \wedge rf_t^2 \wedge rf_t^3 \\ 0 & \text{otherwise} \end{cases} \end{aligned}$$

where $v_{err} \in \mathbb{N}$ is the maximum variation of speed allowed between the wheels (that deviates from a desired rectilinear trajectory) and $v_{min} \in \mathbb{N}$ is the minimum speed allowed (which is characterized by the minimum supported speed of the engines used in the robot). Finally, when an agent collides, the training episode ends, as the agent has failed in its goal.

The provided evaluation function achieves coherent obstacle avoidance strategies, taking into account the specific characteristics of our vehicle. This reward lead to coherent behaviors and avoid oscillations that can increase free collision time but that do not produce progress (for example, a robot turning on itself is an unwanted behavior, as it increases the time without collision but does not produce an advance). In addition, we provide a reward function specifically designed to encourage policies that develop a greater coverage, even when we are using a mapless policy (and therefore the robot cannot be located in the map or even which areas are covered determined).

4.2. Training

Policy gradient methods are a popular family of algorithms for solving this class of problems. These methods perform gradient descent on the objective using empirical estimates of the policy gradient. However, it is not trivial to estimate the gradient of reward signals given to the agent in the future to an action performed by the agent now, especially if the reward is realized many timesteps in the future. Evolution strategies (ESs) have become a valid alternative to classical gradient RL methods. The ability to abandon gradient calculations allows such algorithms to be

evaluated more efficiently. It is also easy to distribute the computation for an ES algorithm to thousands of machines for parallel computation [26].

These strategies have recently become popular for training deep neural networks, in particular on reinforcement learning tasks, a special form of controller design. Compared to classic problems in continuous direct search, deep networks pose extremely high-dimensional optimization problems, with many thousands or even millions of variables [27].

For this paper, we use the covariance-matrix adaptation evolution strategy (CMA-ES). This algorithm is one of the most popular gradient-free optimization algorithms, being the choice for many researchers. The algorithm is ideally suited for reinforcement learning because it is based on ranking policies (and therefore robust against noise), efficiently detects correlations between parameters, and infers a search direction from scalar reinforcement signals [28].

To train policies with CMA-ES, we used a population size of 16. For each candidate network, we used a batch of 8 trajectories and ran each for up to 60 policy steps. We computed the mean reward for the batch and followed the CMA-ES algorithm to generate the next candidate networks. We ran CMA-ES from 200 to 1000 iterations until a good policy with sufficient reward was found.

All the experiments were performed with a value of $\sigma = 0.1$ and the origin of coordinates (of the weights search space) as the initial point. Once the optimal policy was obtained, a subsequent fine training could be performed, starting at the solution found and iterating with $\sigma = 0.01$. This step is optional for the classification networks but fundamental for the regression networks, in which the fine adjustments of the weights of the network are important such that the outputs of the network are within the required range.

4.2.1. Simulation

We developed our own library of reinforcement learning and our own physical simulator, based on the chipmunk library. This library is very common for realistic 2D physical simulations and is used in several robotic simulators, such as [29]. We oriented the development to reduce to the time of execution of an episode, since this type of training requires the execution of the policy thousands of times. A realistic simulation of both the actuators and the sensors has been developed, with the goal of modeling the physical platform used later in the policy transfer. More specifically, we used the beam model of ultrasonic sensor and the differential actuator presented in [30].

We created a multiprocessor implementation of the reinforcement learning algorithm, as discussed in [26], that allows to run in approximately an hour 700 iterations using the eight processing cores of an i7 processor at 3.2 Ghz.

For the training process, 8x4 meter structured environments that were generated randomly for each training episode were used. Moreover, this space was divided into seven cells per dimension, generating a perfect labyrinth (graph) that does not contain loops. This setup makes it possible to effectively reward covering behaviors and prevent the model from learning the special characteristics of a given environment, thus promoting generalization. The robot is included in the environment at a free random position with a random angle.

4.2.2. Policy Training

An episode ends if it reaches 60 policy steps or the agent collides with the environment. As previously mentioned, a population of 16 individuals is used per iteration, each individual evaluating the same policy a total of 8 times. Therefore, in each iteration, 128 episodes are simulated.

4.3. Policy results

Taking into account the policy defined in the previous section, we used the following parameters for its development: $(|\lambda|, v_{min}, v_{err}, t_{mov}) = (8 \times 4 \text{ cells}, 5, 20, 4 \text{ s})$. In this way, a policy step is equal to 4 seconds of simulation time.

Next, we will evaluate the policy when all the sensors work correctly. For this task, we will use the same labyrinth environment throughout the testing phase, obtained through our labyrinth generator.

The policy learned through reinforcement learning shows robustness in the development of the obstacle avoidance and covering task. In Figure 3, we observe several selected sequences of movement developed by the optimal policy, which corresponds to a reward $r = 546.25$ learned using approximately 1000 epochs. It is observed how the robot is able to avoid most obstacles without colliding.

Figure 3: Movements developed by the best policy found. Although it prefers wide zones, it is able to leave the narrow corridors without colliding. The movement of the agent was simulated for 60 policy steps or until it collided with the environment. The environment is shown in green, the robot trail in blue and the three rays that indicate each of the ultrasonic sensors in red. The units of the axes are cm.

To evaluate the real operation of the policy, we executed it 100 times using the same environment, with the robot placed at random starting points and angles. In Figure 4 a), a histogram of the time that each episode of the policy has been active (has not collided with the environment), grouping the time of all the episodes from 10 to 10 for a maximum of 60 steps, is shown.

We can observe that most of the episodes (more than 80%) develop the behavior for the entire time of the simulation.

The previous figure is not sufficient to determine the operation of the policy, since we could find policies that are focused on avoiding collisions even if they do not cause any change of position of the robot (for example, policies that rotate the robot on itself). That is why we present Figure 4 b), in which a box graph representing the total distance covered by the policy is shown. It is observed how most of the episodes develop a distance of 11 meters (which is practically the maximum achievable with the time available per episode).

Figure 4: Statistical analysis of the best policy. All the presented graphs have been obtained from the execution of 100 episodes in the same test environment. a) Histogram of time per episode without colliding. An episode is executed for a maximum of 60 policy steps. b) Box graph of the travel distance per episode (in meters). c) Covering map of discrete zones (80×40 cells) traveled by the agent. The color represents the number of times that a robot has passed through a specific zone.

Furthermore, in the graph Figure 4 c), the specific areas covered by the policy are analyzed for the 100 episodes executed using this policy. In the figure, we observe how the agent prefers to traverse the wide corridors of the labyrinth, since in this case, the exposure to collisions is very low. It is also able to move through the horizontal corridors to end up accessing the wider corridor areas.

To assess the performance of this behavior, we performed a comparison of the obstacle avoidance process with a classic reactive behavior. We chose to use a Braitenberg behavior, which can develop obstacle avoidance with a simple controller exhibiting seemingly intelligent behavior. This behavior has been used in recent works [31, 8] as a baseline to compare with end-to-end policies such as those provided in this article.

In a Braitenberg vehicle, direct or inverse linear connections of the sensory system are established with respect to the agent's motor system. In Figure 5a, the diagram of the used Braitenberg vehicle is shown. It is observed how the three sensors affect the speed of the wheels. The lateral sensors affect them with an inverse relationship. By contrast, the front sensor affects both equally but in a linear fashion. More specifically,

$$vl = \eta \left(\beta_1 \frac{u_{max} - u_3}{u_{max}} - \beta_2 \frac{u_2 - u_{max}}{u_{max}} \right) \quad (1)$$

$$vr = \eta \left(\beta_1 \frac{u_{max} - u_1}{u_{max}} - \beta_2 \frac{u_2 - u_{max}}{u_{max}} \right) \quad (2)$$

where u_{max} is the maximum range (without obstacles) returned by the ultrasonic sensor, β_1 is an intensity factor to increase rotation by the lateral obstacles, β_2 is an intensity factor to increase the influence of frontal obstacles, and η is a normalization term that allows to set the speed within the operating range of the motor. We will use the following values for experimentation: $(u_{max}, \beta_1, \beta_2, \eta) = (400, 1, 2, \frac{motor_{max}}{vl+vr})$. To facilitate the operation of the Braitenberg vehicle, its policy will be executed once per second.

As shown in Figure 5b, the vehicle is capable of developing avoidance behaviors using the reactive behavior.

However, when we compare the statistical results of analyzing this behavior, as we did with our end-to-end policy, we see important differences. In Figure 6, we present the statistical analysis

Figure 5: a) Braitenberg vehicle diagram. White arrows specify linear relationships, and black arrows indicate linear inverse relationships. b) Movements developed by the Braitenberg vehicle. The environment is shown in green, the robot trail in blue and the three rays that simulate each ultrasonic sensors in red. The units of the axes are cm.

of the Braitenberg vehicle. All the presented graphs have been obtained from the execution of 100 episodes in the same test environment as our end-to-end policy. For the comparison, it must be borne in mind that four steps of the reactive policy are equivalent to one step of the end-to-end policy presented in terms of execution time.

In Figure 6a and b, we clearly see how this reactive policy is inferior to our learned policy. It is capable of developing some correct executions, but in the medium term, most of the environment runs finish quickly in collisions. This result is clearly seen in Figure 6c, where we observe that there is no continuity in the covering map. As expected, a reactive system does not have sufficient capacity to satisfactorily develop an obstacle avoidance task while exhibiting coherent navigation in this environment. The learned policy greatly improves the results of the Braitenberg vehicle while being able to provide greater coverage of the environment.

We can see how the learned policy fulfills the desired expectations in terms of obstacle avoidance and covering. In this way, we can observe how the neuronal model learned through reinforcement learning is not only capable of performing the avoidance task but also clearly superior to the Braitenberg vehicle, developing a much more complex policy than a reactive behavior.

However, subsequent tests will determine (see section 6) that this policy is not useful in environments where the agent's sensors may fail. That is why we designed a new policy that considers such failures to substantially increase its robustness.

Figure 6: Statistical analysis of the Braitenberg vehicle. All the presented graphs have been obtained from the execution of 100 episodes in the same test environment of Figure 4. a) Histogram of time per episode without colliding. An episode is executed for a maximum of 240 policy steps (to simulate the same time as the policy presented in Figure 4). b) Box graph of the travel distance per episode (in meters). c) Covering map of discrete zones (80×40 cells) traveled by the agent. The color represents the number of times a robot has passed through a specific zone.

5. Sensor Failure-Aware Policy

The end-to-end model learned to control a given agent works consistently in terms of moving the agent through a certain environment. However, it is expected that it will not work correctly with faulty sensory systems since the policy has not been trained for any type of failure.

Learning a fault tolerance robust policy is not trivial. If we trained the system simply taking into account faulty sensors, we could obtain nonoptimal behaviors because we need a more complex perception phase that allows the policy to improve the perceptions of the environment that cannot be acquired due to sensory failure.

In this paper, we use a robot for which none of the sensor readings overlap each other. This setup makes it possible to determine the operation of a robust failure policy since the failure of a sensor unambiguously affects the perceptions of the robot. We will assume that the malfunction of a defective sensor is due to a total sensor failure and therefore always emits the maximum reading (indicating that there is no obstacle in front of the sensor or that it is malfunctioning).

In this section, a new policy that explicitly determines perception, movement and sensor fusion will be provided.

5.1. Reinforcement Learning Setup

We will start from the base of the previous policy, modifying exclusively those sections that must be adapted for the new policy.

5.1.1. Observation Space

With a fault-tolerant policy, we cannot assume that we will always have available the readings of all the sensors in the system. We assume that there is a boolean vector ξ of the size of the number of sensors available (three in this case) that will indicate the functional sensors during an entire episode. It is assumed that there must be at least one functional sensor.

Moreover, a state will be determined by the readings of all the sensors (those that are not available will return a maximum range value), its gradient and the speed of each of the actuators of the robot.

As will be seen later, the perception module can take several readings of the environment before deciding the movement to be made. The orientation of the robot θ_t is therefore required

to determine how to use these readings. To perceive this orientation, the robot compass will be used. In this way, the observation space of the new policy is as follows:

$$s_t = \xi^{1..3}, \underbrace{o_t^{1..3}, \nabla o_t^{1..3}}_{\times pa}, \theta_t, vl_t, vr_t, \lambda$$

where pa is the number of perceptions to be made by the policy prior to the execution of the robot movement.

5.1.2. Action Space and Reward

The action space of the robot will be exactly the same defined in the previous policy for the perception module. However, the movement module will use discrete actions to move the robot. These selected actions facilitate that the movement module develops a move-forward behavior. Specifically, this module can perform the following actions: $(vl, vr) \in \{(20, 20), (10, 10), (30, 30), (2, -2), (10, -10), (-2, 2), (10, -10), (-2, 2), (-10, 10), (5, 5)\}$

Furthermore, the same reward of previous policy will be used, which will be applied exclusively to the movement model. The perception model will not directly collaborate in the calculation of the reward to give it freedom to perform the perception actions that it will consider best for the policy.

5.1.3. Network Architecture

Figure 7: Neural model used in this policy. Three modules are used: sensory fusion, perception and movement. The fusion module generates several state vectors that will be used by the perception module for taking readings and by the movement module to move the robot.

Figure 7 introduces the new network architecture to support this policy. In the figure, we observe how the network is formed by three differentiated modules:

- The **fusion model** is responsible for combining the sensory information available in the robot. As previously noted, this module can use not only one sensorial reading (for each of the available sensors) but also pa readings before executing the command indicated by the movement module. All this sensorial information is combined into 16 state vectors that

determine the robot status. These state vectors will be used as input in the following network modules.

- The **perceptual model** is responsible for determining a sequence of pa robot movements emitting a set of pairs (vl, vr) . After executing each pair, and during t_{per}/pa seconds, a sensor reading will be taken. All this sensory information will be communicated to the fusion model for the next iteration.
- The **movement model** is responsible for making a systematic advance of the robot without collision and promoting covering. For this task, it will receive the robot state vectors and decide which of the 8 available actions it will develop during t_{mov} seconds.

We consider that this modular system allows extensibility (using more models connected to the sensory fusion system) and reuse (taking one of the three models learned independently) and ultimately provides facilities for the adaptability of the policy to other sensors or tasks. In addition, this model gives fundamental support to highly fault-tolerant navigation systems, in which the failure of a sensory element requires changing the movement policy in order to perceive the environment correctly.

5.2. Training

For the previous policy, an episode will end when it has reached 60 policy steps or there is a collision. Moreover, 16 individuals will be used per iteration, each individual will be evaluated 8 times. Therefore, in each iteration, 128 episodes will be simulated.

5.3. Policy results

In this section, we will see the policy performance for obstacle avoidance and covering assuming that there are no faults in the sensory system. We will follow the same analysis scheme developed in the previous section.

To test our policy, we used the following parameters: $(|\lambda|, v_{min}, v_{err}, t_{mov}, t_{per}, pa) = (8 \times 4 \text{ cells}, 5, 20, 4 \text{ s}, 3 \text{ s}, 3)$. In this way, a policy step is equal to 7 seconds of simulation time.

In Figure 8, we observe how the policy developed is similar to the simple policy in the case in which all of the sensors work correctly. The obtained policy correspond to a reward $r = 158.375$ learned using approximately 1000 epochs. It should be noted that this reward is not directly comparable to the simple policy. Although the movement developed per step is the same, the simple policy has a higher frequency of execution, which leads to a higher reward.

Observing the statistical analysis presented in Figure 9, we observe that this policy is approximately as robust as the one obtained previously, with 80% of episodes without collision. In Figure 9b), we can see how most of the episodes develop a distance of 11 meters but with 1 meter of variation more than with the previous policy. Regarding the covering map presented in Figure 9c), we see that this policy also prefers the wide corridors, moving inside them even though it is able to move through the other labyrinth areas.

Therefore, we conclude that this policy is suitable to address the avoidance and covering task even though it is much more complex than the previous one. Now, we will analyze if this complexity contributes to improve it in terms of robustness to faults in the sensory system.

Figure 8: Movements developed by the best policy found. Although it prefers wide zones, it is able to leave the narrow corridors without colliding. The movement of the agent is simulated for 60 simulation steps or until it the agent collides with the environment. The environment is shown in green, the robot in blue and the three rays that simulate each ultrasonic sensors in red. The units of the axes are cm.

Figure 9: Statistical analysis of the best policy. All the presented graphs have been obtained from the execution of 100 episodes in the same test environment. a) Histogram of time per episode without colliding. An episode is executed for a maximum of 60 policy steps. b) Box graph of the travel distance per episode (in meters). c) Covering map of discrete zones (80×40 cells) traveled by the agent. The color represents the number of times that a robot has passed through a specific zone.

6. Fault Tolerance Analysis

In the previous section, two policies obtained through reinforcement learning have been presented. They are capable of effectively developing a obstacle avoidance and covering behavior. In this section, we will analyze if both policies are functional in the case of faults in the sensory system that disable any of the sensors available to the robot.

Moreover, two experiments will be presented, analyzing the policy behavior when the side sensors of the robot or the front sensor fail. The result of the simple policy, the failure aware policy and the real execution using a physical robot will be shown.

Figure 10: Robot used in the development of this article. This is a custom robot equipped with 2 motors with optical encoders, 3 ultrasonic sensors and a compass module.

First, we want to briefly describe the robotic platform that we will use to test the fault-tolerant policy. We will use a physical robot identical to the one used within the simulator, which is shown in Figure 10. It is based on the Arduino Mega Auriga platform and developed with the available motors and sensors available in <https://www.makeblock.com/steam-kits/mbot-ultimate>. This platform is ideal for our needs because it allows to design the robot specifically for the tests required in this section. This platform is also used by other recent research works [32, 33]. More specifically, the robot is equipped with 2 motors with optical encoders, 3 ultrasonic sensors and a compass module. As noted before, a very limited platform for which any sensory failure is highly penalized has been chosen.

6.1. Front sensor fails

The most important sensor for the development of an obstacle avoidance behavior is the front sensor, since the robot is usually directed forward to provide advancement. If this sensor does not work and emits the maximum sensor reading (4 m), we find that the behavior of the policy deteriorates substantially. In order to evaluate the real operation of the policy, we executed 100 episodes with the same labyrinth and the robot placed at random starting points and angles.

Figure 11 shows the result of the statistical analysis performed. It can be seen how the simple policy is much less robust to failures, developing in the case of lateral failures worse results than

Figure 11: This figure presents a comparative analysis of the simple policy with respect to the fault-tolerant one when the frontal ultrasonic sensor does not work. One hundred episodes were executed using the same environment for the two policies, obtaining the following: a) Histogram of time per episode without colliding. An episode is executed for a maximum of 60 policy steps. b) Covering map of discrete zones (80×40 cells) traveled by the agent. The color represents the number of times a robot has passed through a specific zone. c) An example of execution of an episode for each policy.

the advanced policy. It is observed in Figure 11a) that the fault-tolerant policy is able to complete 50% of the episodes, whereas the simple policy fails for all 100 cases. In Figure 11b), we can observe the cover maps of both policies. Simple policy shows discrete spots because the policy fails in the first steps. The fault-tolerant policy covers more area; thus, the cover map is more homogenous.

We could see two executions of each of these policies in Figure 11c). The simple policy is not able to avoid a collision in the first seconds. It should be noted that in this case, we can already perceive the advantages of using a specific model responsible for perception: the robot oscillates to obtain the three readings required by perceptual model ($pa = 3$) before developing the movement, which allows it to move in a more informed way through the environment without colliding.

In Figure 12, the box charts obtained with respect to the distance traveled per episode are shown. These graphs summarize the movement made by each of the policies in the 100 episodes studied. In this case, the fault-tolerant policy exhibits a greater average of distance moved. Moreover, this policy is able to develop episodes with more distance than the simple policy.

Figure 12: Box graph of the travel distance per episode (in meters) for 1) simple policy and 2) fault-tolerant policy. These results are obtained from 100 random episodes executed for each policy.

Although the results of our fault-tolerant policy under simulation are very good, we want to know how this policy behaves when transferred to real robots. To validate our policy in a real environment we designed two experiments. We used an environment of $1.6\text{ m} \times 2\text{ m}$, surrounded by walls, for both experiments. Figure 13 shows an episode of 90 seconds, in which the robot is able to move around the environment using only its lateral sensors. It is observed how the behavior is similar to that obtained in the simulator: the robot oscillates to be able to perceive with its lateral sensors.

Figure 14 shows the second experiment, in which an obstacle has been added to the front of the robot. As expected, the robot is not able to detect it because it does not have a functional front sensor (and the obstacle is only 50 cm from the robot position), ultimately colliding with it.

Considering the performed tests, we can determine that the fault-tolerant policy has been able to move around the environment in an effective way, avoiding collisions and developing a perception strategy that allows it to perform its task effectively. In addition, the same policy is transferable to a physical robot, developing similar behaviors to those obtained under simulation.

Figure 13: Execution of the advanced policy in a real environment of $1.6\text{ m} \times 2\text{ m}$. The behavior will run for 90 seconds. a) Long exposure image that combines the entire robot's movement sequence. The circles that are observed correspond to the upper robot led ring. b) Captures of the episode at time 0 s, 30 s, 60 s and 90 s.

Figure 14: Selected sequences from the execution of the advanced policy in a real environment of $1.6\text{ m} \times 2\text{ m}$ with an obstacle arranged in the front part of the robot. The behavior will run for 15 seconds. a) Long exposure image that combines the entire robot's movement sequence. The circles that are observed correspond to the upper robot led ring. b) Captures of the episode at the instants 0 s, 5 s, 10 s and 15 s.

6.2. Side sensors fail

In the previous experiment, we evaluated the operation of our policies when a robot sensor stopped working. Next, we will evaluate what occurs when only one sensor (the front one) works in our robot. This limit even more the operation of the agent, since it only has a reading given a certain position and angle.

The results of this experiment are observed in Figure 15. In this case, we can see in a) how the simple policy practically does not have collisions, which contrasts with the fault-tolerant policy. In addition, in Figure 16, we can see that most of the executions of the simple policy advance the same distance, higher than the failure-tolerant policy and with less variance.

Figure 15: This figure presents a comparative analysis of the simple policy with respect to the fault-tolerant one when both side ultrasonic sensors do not work. One hundred episodes were executed using the same environment for the two policies, obtaining the following: a) Histogram of time per episode without colliding. An episode is executed for a maximum of 60 policy steps. b) Covering map of discrete zones (80×40 cells) traveled by the agent. The color represents the number of times a robot has passed through a specific zone. c) An example of execution of an episode for each policy.

An explanation of these values can be obtained by analyzing Figure 15 b): the robot is rotating on itself, producing unwanted behavior but improving its survival score. This does not occur in the fault-tolerant policy, which does perform exploration tasks, thus making the environment coverage more uniform. A typical execution of the simple policy (rotation on itself) with respect to the complex policy (exploration) can be observed in Figure 15 c).

More specifically, we could observe that the robot oscillates slightly, which allows it to obtain the three readings in its perceptual model $pa = 3$ and to operate as if it had three effective sectors. This is an advantage derived directly from the use of a separate perceptual model of the robot's movement model. Therefore, the presented architecture provides substantial advantages, since it provides flexibility to the system and makes it robust to important failures in its sensory system.

Figure 16: Box graph of the travel distance per episode (in meters) for 1) the simple policy and 2) the fault-tolerant policy. These results are obtained from 100 random episodes executed for each policy.

We carried out a test to evaluate the transfer of the fault-tolerant policy to a physical robot. The results are shown in Figure 17. We could observe that the robot is able to navigate through the environment without colliding even when only the front robot is functional.

Therefore, we consider that the fault-tolerant policy meets our expectations: it is able to develop an obstacle-avoidance behavior that invites exploration and yields better results in terms of sensory failures in a system that is already limited (with only three ultrasonic sensors).

7. Conclusion and future research

In this article, we have contributed with two behavioral policies capable of developing an obstacle avoidance and covering task efficiently, specifically, a first policy based on a simple neuronal model and a second policy that separates into three modules the traditional sensor-motor mechanism used in end-to-end policies. In both cases, we have observed how the policies obtained through deep RL have substantially improved a Braitenberg vehicle for the development of the obstacle avoidance task, which reinforces our assumption that obtaining valid policies through deep RL is feasible.

The design of this second policy is mainly aimed at providing greater robustness to the behavior and allowing robot handling even with sensory failures. A very limited robotic platform has been chosen for this task, where any sensory failure is highly penalized.

Figure 17: Execution of the advanced policy in a real environment of $1.6\text{ m} \times 2\text{ m}$. The behavior will run for 90 seconds. a) Long exposure image that combines the entire robot's movement sequence. The circles that are observed correspond to the upper robot led ring. b) Captures of the episode at times 0 s, 30 s, 60 s and 90 s.

To evaluate the operation of these policies, several experiments were carried out with a robot with only one or two functional ultrasonic sensors. The obtained results indicate that although both policies work in a similar way in environments without sensorial failures, the fault tolerant policy develops much better behaviors with robots with faulty sensors. In fact, it is capable of performing coherent behaviors in very limited perception environments, substantially improving the navigation of the basic policy and allowing the robot to finish its task even when it does not have its entire sensory system.

It should be noted that the fault-tolerant policy provided several advantages. This policy uses a fusion model that projects sensory perception to a vector of characteristics that defines the state of the robot. It also uses a model that guides the perception of the robot, necessary to support fault tolerance, and that allows the learning of specific movement policies required by the sensory system. Finally, it defines a movement model that transforms the state of the robot into policy movements to advance in the avoiding task. This model gives fundamental support to highly fault-tolerant navigation systems, where the failure of a sensory element requires changing the movement policy in order to perceive the environment correctly.

As noted previously, we consider that this modular system allows extensibility (using more models connected to the sensory fusion system) and reuse (taking one of the three models learned independently) and ultimately provides facilities for the adaptability of the policy to other sensors or tasks. It has also been observed that in cases of extreme sensory failures, the policy is able to develop obstacle avoidance tasks by developing movements guided by the sensory system (such as oscillatory movements) that increase the safe navigability of the robot.

Furthermore, in our real-world tests, we have observed how these policies can be transferred to a physical robot, working similar to the results obtained from the simulator.

We now note several promising lines of future work. First, we are interested in applying our model to different tasks, combining more complex sensors, such as cameras or depth perception devices. Second, another interesting line is to analyze the neuronal models to determine the best parameters for different tasks, because for more complex sensors, the number of neurons

per layer (and possibly the number of layers) should be increased. Another interesting line is to determine how effective the policy transfer is with more complex sensors, for which realistic simulations are prohibitive.

References

- [1] Rahmatizadeh, R., Abolghasemi, P., Bölöni, L., Levine, S.: Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration. arXiv preprint arXiv:1707.02920 (2017)
- [2] Pfeiffer, M., Schaeuble, M., Nieto, J., Siegwart, R., Cadena, C.: From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE (2017) 1527–1533
- [3] Liu, G.H., Siravuru, A., Prabhakar, S., Veloso, M., Kantor, G.: Learning end-to-end multimodal sensor policies for autonomous navigation. arXiv preprint arXiv:1705.10422 (2017)
- [4] Zhuo-Hua, D., Zi-Xing, C., Jin-Xia, Y.: Fault diagnosis and fault tolerant control for wheeled mobile robots under unknown environments: A survey. In: Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on, IEEE (2005) 3428–3433
- [5] Doran, M., Sterritt, R., Wilkie, G.: Autonomic sonar sensor fault manager for mobile robots. World Academy of Science, Engineering and Technology, International Journal of Computer and Information Engineering **4** (2017)
- [6] Tai, L., Paolo, G., Liu, M.: Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In: Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on, IEEE (2017) 31–36
- [7] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A.J., Banino, A., Denil, M., Goroshin, R., Sifre, L., Kavukcuoglu, K., et al.: Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673 (2016)
- [8] Bing, Z., Meschede, C., Huang, K., Chen, G., Rohrbein, F., Akl, M., Knoll, A.: End to end learning of spiking neural network based on r-stdp for a lane keeping vehicle. In: 2018 IEEE International Conference on Robotics and Automation (ICRA), IEEE (2018) 1–8
- [9] Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., et al.: Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. IEEE Signal processing magazine **29** (2012) 82–97
- [10] Krizhevsky, A., Sutskever, I., Hinton, G.: Imagenet classification with deep convolutional neural networks. advances in neural information processing systems **25** (nips 2012) (2017)
- [11] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518** (2015) 529
- [12] Liu, L., Liu, Y.J., Tong, S.: Neural networks-based adaptive finite-time fault-tolerant control for a class of strict-feedback switched nonlinear systems. IEEE Transactions on Cybernetics (2018)
- [13] Van, M.: An enhanced robust fault tolerant control based on an adaptive fuzzy pid-nonsingular fast terminal sliding mode control for uncertain nonlinear systems. IEEE/ASME Transactions on Mechatronics (2018)
- [14] Khalastchi, E., Kalech, M.: On fault detection and diagnosis in robotic systems. ACM Computing Surveys (CSUR) **51** (2018) 9
- [15] Lv, F., Wen, C., Bao, Z., Liu, M.: Fault diagnosis based on deep learning. In: American Control Conference (ACC), 2016, IEEE (2016) 6851–6856
- [16] Cheng, Y., Zhang, W.: Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. Neurocomputing **272** (2018) 63–73
- [17] Milde, M.B., Blum, H., Dietmüller, A., Sumslawska, D., Conradt, J., Indiveri, G., Sandamirskaya, Y.: Obstacle avoidance and target acquisition for robot navigation using a mixed signal analog/digital neuromorphic processing system. Frontiers in neurorobotics **11** (2017) 28
- [18] Pandey, A., Pandey, S., Parhi, D.: Mobile robot navigation and obstacle avoidance techniques: A review. Int Rob Auto J **2** (2017) 00022
- [19] Li, D.P., Liu, Y.J., Tong, S., Chen, C.P., Li, D.J.: Neural networks-based adaptive control for nonlinear state constrained systems with input delay. IEEE Transactions on Cybernetics (2018)
- [20] Gao, T., Liu, Y.J., Liu, L., Li, D.: Adaptive neural network-based control for a class of nonlinear pure-feedback systems with time-varying full state constraints. IEEE/CAA Journal of Automatica Sinica **5** (2018) 923–933
- [21] Zhu, Y., Mottaghi, R., Kolve, E., Lim, J.J., Gupta, A., Fei-Fei, L., Farhadi, A.: Target-driven visual navigation in indoor scenes using deep reinforcement learning. In: Robotics and Automation (ICRA), 2017 IEEE International Conference on, IEEE (2017) 3357–3364
- [22] Oh, J., Chockalingam, V., Singh, S., Lee, H.: Control of memory, active perception, and action in minecraft. arXiv preprint arXiv:1605.09128 (2016)

- [23] Tai, L., Liu, M.: Towards cognitive exploration through deep reinforcement learning for mobile robots. arXiv preprint arXiv:1610.01733 (2016)
- [24] Kober, J., Bagnell, J.A., Peters, J.: Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* **32** (2013) 1238–1274
- [25] Tai, L., Liu, M.: Deep-learning in mobile robotics-from perception to control systems: A survey on why and why not. arXiv preprint arXiv:1612.07139 (2016)
- [26] Salimans, T., Ho, J., Chen, X., Sidor, S., Sutskever, I.: Evolution strategies as a scalable alternative to reinforcement learning. arXiv preprint arXiv:1703.03864 (2017)
- [27] Müller, N., Glasmachers, T.: Challenges in high-dimensional reinforcement learning with evolution strategies. arXiv preprint arXiv:1806.01224 (2018)
- [28] Heidrich-Meisner, V., Igel, C.: Neuroevolution strategies for episodic reinforcement learning. *Journal of Algorithms* **64** (2009) 152–168
- [29] Pincioli, C., Trianni, V., O’Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., et al.: Argos: a modular, multi-engine simulator for heterogeneous swarm robotics. In: *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on, IEEE* (2011) 5027–5034
- [30] Thrun, S., Burgard, W., Fox, D.: *Probabilistic robotics*. 2005. Massachusetts Institute of Technology, USA (2005)
- [31] Bing, Z., Meschede, C., Röhrbein, F., Huang, K., Knoll, A.C.: A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in neurorobotics* **12** (2018) 35
- [32] McLin, J., Hoang, N., Deneke, W., McDowell, P.: A mirror world-based robot control system. In: *Information Systems Engineering (ICISE), 2017 Second International Conference on, IEEE* (2017) 74–77
- [33] Qian, Z., Zhang, T., Cai, M., Lin, W., Zhang, W.: On rapid prototyping of assembly systems—a modular approach. *International Journal of Materials and Product Technology* **56** (2018) 220–233

Fidel Aznar received his Ph.D. degree in Computer Engineering And Computing at the University of Alicante (Spain) in 2006. He is now professor at the University of Alicante Polytechnic School in the Department of Computer Science and Artificial Intelligence. His research interests include swarm robotics, mobile robotics, agent systems, reinforcement learning, deep learning and artificial intelligence.

Mar Pujol received her B.A. in Mathematics at the University of Valencia (Spain) in 1985, and the Ph.D. degree in Computer Science at the University of Alicante in 2000. She is full professor at the Department of Computer Science and Artificial Intelligence in the University of Alicante (Spain). Her research interests include swarm robotics, mobile robotics, agent systems, reinforcement learning, deep learning and artificial intelligence.

Ramón Rizo Ramon Rizo received his Ph.D. in Computer Science from the Polytechnic University of Valencia, Spain, in 1991. He is full professor and doctoral supervisor at the University of Alicante Polytechnic School in the Department of Computer Science and Artificial Intelligence. He has directed 13 doctoral theses, made more than 120 publications and scientific documents in international journals and has participated in more than 130 conferences on topics related to artificial intelligence. His fields of research include deep learning, computer vision, artificial intelligence, and software integration.