IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received August 6, 2020, accepted August 14, 2020, date of publication August 19, 2020, date of current version August 31, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3018026

Reinforcement Learning for Position Control Problem of a Mobile Robot

GONZALO FARIAS[1], GONZALO GARCIA[2], GUELIS MONTENEGRO[1], ERNESTO FABREGAS[3], SEBASTIÁN DORMIDO-CANTO[3], AND SEBASTIÁN DORMIDO[3], (Member, IEEE)

[1]Escuela de Ingeniería Eléctrica, Pontificia Universidad Católica de Valparaíso, Valparaíso 2362804, Chile
[2]Department of Ocean and Mechanical Engineering, Florida Atlantic University (FAU), Boca Raton, FL 33431, USA
[3]Departamento de Informática y Automática, Universidad Nacional de Educación a Distancia, 28040 Madrid, Spain

Corresponding author: Gonzalo Farias (gonzalo.farias@pucv.cl)

This work was supported in part by the Chilean National Research and Development Agency (ANID) through the Project Fondo Nacional de Desarrollo Científico y Tecnológico (FONDECYT) under Grant 1191188, in part by the Dirección de Investigación (DI)-Pontificia Universidad Católica de Valparaíso under Grant 039.437, in part by the Spanish Ministry of Economy and Competitiveness through the Projects under Grant RTI2018-094665-B-I00 and Grant ENE2015-64914-C3-3-R, and in part by the Spanish Ministry of Science and Innovation through the Project under Grant PID2019-108377RB-C32.

ABSTRACT Due to the increase in complexity in autonomous vehicles, most of the existing control systems are proving to be inadequate. Reinforcement Learning is gaining traction as it is posed to overcome these difficulties in a natural way. This approach allows an agent that interacts with the environment to get rewards for appropriate actions, learning to improve its performance continuously. The article describes the design and development of an algorithm to control the position of a wheeled mobile robot using Reinforcement Learning. One main advantage of this approach concerning traditional control algorithms is that the learning process is carried out automatically with a recursive procedure forward in time. Moreover, given the fidelity of the model for the particular implementation described in this work, the whole learning process can be carried out in simulation. This fact avoids damages to the actual robot during the learning stage. It shows that the position control of the robot (or similar specific tasks) can be done without the need to know the dynamic model of the system explicitly. Its main drawback is that the learning stage can take a long time to finish and that it depends on the complexity of the task and the availability of adequate hardware resources. This work provides a comparison between the proposed approach and traditional existing control laws in simulation and real environments. The article also discusses the main effects of using different controlled variables in the performance of the developed control law.

INDEX TERMS Mobile robot, position control, reinforcement learning.

I. INTRODUCTION

Robotics has been very popular over the last few years. Today it is very common to find robots performing different tasks in many areas of our daily lives including social environments, agriculture [1], logistics [2], manufacturing [3], medicine [4], and education [5], just to mention a few of them.

In this context, mobile robots deserve a special mention because of their importance in performing useful tasks. In these kinds of applications, the robot has to navigate in different environments by controlling its own position [6]. This experiment is known as point stabilization or position control of a nonholonomic mobile robot. It deals with the problem of carrying a robot from its current position to a known destination point [7]. It has been addressed in many ways by using different traditional control techniques, such as nonlinear model predictive control [8], back-stepping control with asymptotic stability [9], continuous time-varying adaptive controllers [10], PID controllers [11], etc.

Currently, many efforts are dedicated to improving the existing algorithms by optimizing the trajectories described by the robot when it goes from a starting point to a destination point. With the traditional control approaches in general, a controller is designed by using the model of the system and obtaining its parameters. These parameters can be adjusted using different current techniques. The result is an algorithm, often called control law, that takes the inputs and calculates the action to make the control most efficiently.

The associate editor coordinating the review of this manuscript and approving it for publication was Sotirios Goudos.

On the other hand, Machine Learning (ML) methods in general, obtain a control law from data, or an agent learns how to solve a specific problem adjusting the internal parameters automatically. Most of these algorithms need a learning stage, where the agent learns to solve the problem. Recently, these new paradigms have been applied instead of traditional ones. Using ML technique, this problem can be solved in a different way, and the results can be interesting. See, for example, the application of genetic algorithms [12], neural networks [13], and fuzzy logic [14], among others.

Recently, the authors have published [15], which is an improvement of an existing control law [16] using traditional techniques. The motivation of the present article is to improve the results of the previous work by using artificial intelligence methods, such as Reinforcement Learning (RL) [17]. RL is an area of ML where an agent interacts with the environment to get some rewards for its actions. From this interaction, the agent must learn how to make a specific task finding a balance between the new information obtained from the environment and its current accumulated knowledge [18].

Some works can be found in the literature that apply RL to mobile robots with different applications. For example, in [19], the authors present the control of a two-wheeled mobile robot when its dynamical model is not available. The algorithm builds a performance evaluation function by using its own artificial neural networks to learn online. The simulation results demonstrate that it can successfully achieve self-learning balance control of two-wheeled robot system in a short time. Moreover, in [20], the authors propose an algorithm based on a reinforcement learning agent, for self-adapting multiple low-level PID controllers in mobile robots. For the formulation of the artificial expert agent, they developed an incremental model-free algorithm version of the double Q-Learning algorithm for fast on-line adaptation of multiple low-level PID controllers.

Furthermore, in [21], the authors propose an approach using deep reinforcement learning for the navigation of mobile robots in an unknown environment. With this algorithm, the robot can learn from the environment gradually, and it can navigate to the target destination autonomously using an RGB-D camera. The simulation results show that the mobile robot can reach the desired targets without colliding with any obstacles. Also, in [22] the authors present an approach based on a learning system to control the navigation of a nonholonomic mobile robot in an unknown environment. In this algorithm, the robot learns how to navigate and build a map of the environment. The results are shown with virtual and real experiments.

In [23] the authors provide several approaches to develop trajectory tracking based on ML for a wheeled mobile robot in simulation. In particular, they apply some techniques from RL together with the use of dynamic inversion for the cancellation of nonlinearities, and neural networks as universal approximators for the estimation of the vehicle's dynamics. Stability of the closed loop system was achieved by the use of an additional supervisory line in the control law. Finally, in [24], the authors present the application of deep reinforcement learning algorithms for mobile robots and formation path planning with a specific focus on reliable obstacle avoidance in constrained maritime environments. The designed RL path planning algorithm is able to solve other complex issues such as the compliance with vehicle motion constraints.

In our case, the robot must implement the point stabilization problem with RL by learning how to reach a target point from its current position. This method is divided into two phases: 1) the learning stage, and 2) operational stage. During the first stage, the robot is trained in simulation to obtain the learning matrix. During the second stage, the robot uses this matrix to carry out the control of its position. The main advantage of this approach with respect to the traditional control algorithms is that the learning process is carried out automatically in a recursive process in a computer. This process is carried out by trial and error, but learning has shown that the system converges to increasingly better behaviour, if the reward function is adequate. The case of using RL is taking a step further and aiming for optimal control, which does not depend on any model. It learns an optimal control law for the plant itself and the current environment. The downside is the time needed for the learning stage, but eventually, it delivers the best possible controller.

The result is a controller that can make a specific task, and no dynamic model of the system is needed. As the learning is carried out online in simulation, the implemented control law allows its continuous improvement even in the real environment. If the model of the robot in simulation is well-adjusted, the controller is directly implemented in the real robot for operation with excellent results. In this way, the damages to the robot can be avoided during the learning stage. Its main drawback is that the learning stage can take a long time to finish, depending on the complexity of the task to solve, and the hardware resources of the computer used to train and obtain the model.

The main goal of this paper is the design, development, and implementation of an algorithm to control the position of a wheeled mobile robot using an RL approach. A summarised list of contributions from this work is as follows: 1) the design and implementation of a control law for point stabilization of a mobile robot based on RL, 2) the implementation of the proposed algorithm in simulation and real environments, including obstacles, 3) the comparison of the results of the new approach with traditional existing control laws, 4) the evaluation of the performance of the proposed control law using performance indexes, and 5) the evaluation of the effect of different controlled variables in the performance of the developed control law.

The remainder of the paper is organized as follows: Section II presents some theoretical aspects about the Reinforcement Learning approach; Section III describes its application to mobile robot position control; Section IV shows and discusses some simulation and experimental results of this research, and Section V presents the main conclusions and future works.

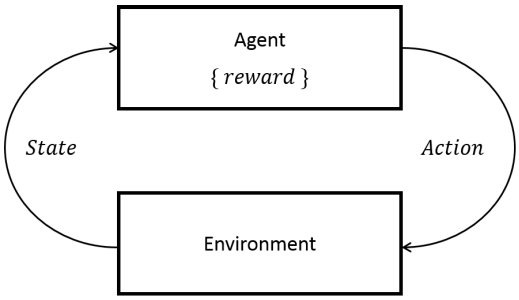

FIGURE 1. Interaction between agent and environment.

II. REINFORCEMENT LEARNING

Due to the increase in complexity in autonomous vehicles, most of the existing control systems are proving to be inadequate. Reinforcement Learning is gaining traction as it is posed to overcome these difficulties natively. As part of machine learning, RL allows the agent to learn an optimal policy by interacting directly with the environment. The policy has the ability to learn how this particular robot deals with the surroundings, by memorizing combinations of states and actions that deliver higher rewards, defined by a reward function, providing a controller that upon convergence it is an optimal one. In the present work, the robot has two inputs and two outputs, imposing a challenge for the multivariable interactions. This condition limits the performance that classical controllers can achieve.The interaction between the learning algorithm and the system and the flow of the signals involved is depicted in Figure 1.

In this interaction, the Agent that encompasses the learning algorithm, generates the action delivered to the Environment, generally based on a combination between purely random values, and values that reflect the learning degree. The system, which is part of the Environment together with the external world, changes its states upon reception of the action. This new state is sent back to the Agent where it is fed into the learning algorithm, and processed using the associated reward, who is an essential part of the algorithm ([17], [18], and [25]).

Based on the principles of Dynamic Programming (MDP), RL assumes that the system can be modeled as a Markov Decision Process, characterized by the relation $x_{k+1} = f(x_k, u_k)$, and the reward $r_{k+1} = \rho(x_k, u_k)$, where x_k is the state of the system, and $u_k = l(x_k)$ is the control law to be determined in a recursive way. In the present case, both relations, the MDP and the reward function are known or given by the designer. An infinite-horizon return is defined by:

$$\sum_{k=0}^{\infty} \gamma^k r_{k+1} = \sum_{k=0}^{\infty} \gamma^k \rho(x_k, l(x_k)), \tag{1}$$

with $0 < \gamma < 1$ a discount factor that seeks to penalize future rewards. This relation represents the accumulated discounted rewards into the future starting from the current state x_0 and after the application of the policy l. From this, a function called action-value is defined:

$$\begin{aligned} Q(x_0, u_0) &= \rho(x_0, u_0) + \sum_{k=1}^{\infty} \gamma^k \rho(x_k, l(x_k)) \\ &= \rho(x_0, u_0) + \gamma \sum_{k=0}^{\infty} \gamma^k \rho(x_{k+1}, l(x_{k+1})), \end{aligned} \tag{2}$$

This function captures the degree of goodness for the system to perform the action (u_0) from the state x_0, and then following the policy l. The optimal results is obtained by the maximization of the future reward with $l^*(x) = \gamma \max_v Q(x, v)$, giving an optimal action-value function $Q^*(x, u)$. Written recursively, gives:

$$\begin{aligned} Q^*(x_k, u_k) &= \rho(x_k, u_k) + \gamma \max_v Q^*(x_{k+1}, u_k) \\ &= \rho(x_k, u_k) + \gamma \max_v Q^*(f(x_k, u_k), v), \end{aligned} \tag{3}$$

These relations capture the Optimally Principle by showing that future optimal control values are not defined by past optimal control values, but only by the current state. As in Dynamic Programming, this equation solves the optimal problem by conducting recursive computations backward in time. The optimal action u while in state x corresponds to the summation of an immediate reward, and the optimal value obtained by the best action in the state reached by using u previously, under the discount factor. A major breakthrough in these calculations allowed the inversion of the backward-in-time marching to a forward-in-time iterative algorithm, called Q-learning, that asymptotically converges to Q^*, as time k goes to infinity, replacing any system's model by the model itself [27] and [28].

As this process is developed in time during the operation of the system, the action-value function is constantly being updated without the need for any backward computation, or the requirement of a mathematical model function f to predict the behavior of the system. This is done by the system itself while it is interacting with the surroundings. Using u_k, and the transition from x_k to x_{k+1}, the action-value function is updated as following, with $0 < \alpha < 1$ a learning rate:

$$\begin{aligned} Q^{i+1}(x_k, u_k) = Q^i(x_k, u_k) + \alpha(\rho(x_{k+1}, u_k) \\ + \gamma \max_v Q^i(x_{k+1}, v) - Q^i(x_k, u_k)) \end{aligned} \tag{4}$$

The Q-learning recursive equation has a familiar interpretation, representing a discrete filter, computing a weighted average between the accumulated knowledge $Q^i(x_k, u_k)$ and the new data, encapsulated into the temporal-difference $t_D^i(x_k, u_k)$ by:

$$Q^{i+1}(x_k, u_k) = \alpha t_D^i(x + k, u_k) + (1 - \alpha) Q^i(x_k, u_k) \tag{5}$$

with $t_D^i(x_k, u_k) = \rho(x_{k+1}, u_k) + \gamma \max_v Q^i(x_{k+1}, v) - Q^i(x_k, u_k)$. The initial condition of Q is either set to zero, i.e. $Q^0 = 0$ reflecting no prior knowledge, or to the reward function.

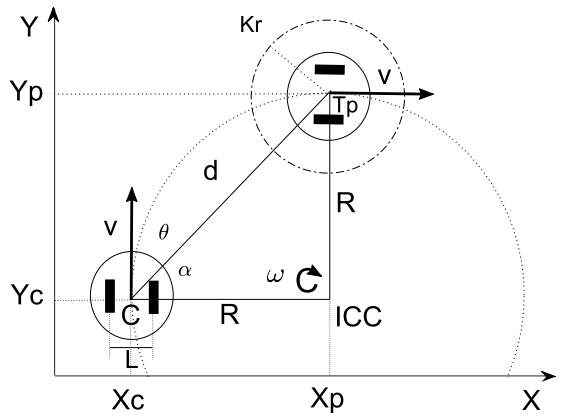

FIGURE 2. Position control of a differential robot.

III. MOBILE ROBOT POSITION CONTROL WITH RL

A. KINEMATIC MODEL OF THE ROBOT

A differential wheeled robot is a mobile robot who bases its movement on two separately driven wheels placed on each side of its body. The two drive velocities (v_L, v_R) are perpendicular vectors to the wheels axis. Furthermore, the wheels are assumed to roll without slipping. These conditions impose some restrictions known as nonholonomic constraints [29]. The robot can change its direction by varying the relative rotation between the wheels, so it does not need an additional steering motion to turn. The kinematic model of the robot in Cartesian coordinates is the following [6], [30]:

$$\begin{cases} \dot{x}_c = v\cos(\theta) \\ \dot{y}_c = v\sin(\theta) \\ \dot{\theta} = \omega, \end{cases} \tag{6}$$

where θ is the heading direction angle of the robot and it is perpendicular to the turning radius (R). The instant linear velocity $v = (v_L + v_R)/2$ is the average of the linear velocities of the left and right wheels, v_L and v_R, respectively. The angular velocity $\omega = (v_L - v_R)/L$ is defined with respect to the *ICC (Instantaneous Center of Curvature)*, where L is the distance between the wheels. Naturally, the mobile robot has a maximum linear velocity $v_{\max}$ and, usually, also a minimum turning radius $R_{\min}$, i.e, it cannot freely rotate [15].

B. POSITION CONTROL PROBLEM

This experiment consists of driving a mobile robot from point C (current position of the robot) to point Tp (target point), by manipulating its angular (ω) and linear (v) velocities. Note that these velocities are then transformed into speeds for the left and right motors as the robot is a double wheeled one. Figure 2 shows the variables involved in this experiment.

This problem has been widely studied in recent years. Although the kinematic behavior of these robots may seem simple, the nonholonomic constraints introduce a challenging problem when designing a control law. This has been explained in more detail in some of the author's previous works, [15], [31], and [32]. In regular motion, the differential robot describes a circular trajectory of radius R with

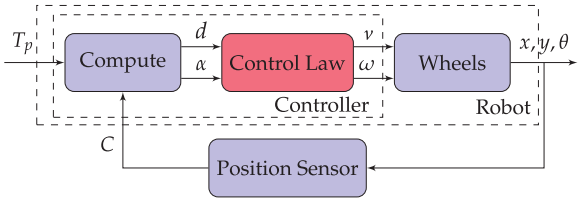

FIGURE 3. Block diagram of the position control problem.

center *ICC*. The position control algorithm seeks to minimize the orientation error, $\theta_e = \alpha - \theta$, where α is the current angle to the target point and θ is the current orientation of the robot. At the same time, the robot tries to reduce the distance to the target point ($d \to 0$).

Figure 3 shows the block diagram of the control algorithm for this experiment, where the inner dashed square represents the controller and the outer dashed square represents the robot. Note that the Position Sensor is an IPS (Indoor Positioning System), which provides the absolute position and orientation of the robot [5], [33].

Equation (7) calculates the distance (d) and equation (8) calculates the angle to the target point (α). In both cases, the values used for the calculation are the coordinates of $Tp(x_p, y_p)$ and $C(x_c, y_c)$. Note that both equations are implemented in the block *Compute*.

$$d = \sqrt{(y_p - y_c)^2 + (x_p - x_c)^2} \tag{7}$$

$$\alpha = \tan^{-1}\left(\frac{y_p - y_c}{x_p - x_c}\right) \tag{8}$$

Taking the time derivative of Eq. 7, 8 and after some mathematical manipulations, the following dynamical system is obtained [15]:

$$\begin{aligned} \dot{d} &= -v\cos(\theta_e) \\ \dot{e}_\theta &= \frac{v}{d}\sin(\theta_e) - \omega. \end{aligned} \tag{9}$$

With the distance and angle to the destination point, the algorithm must obtain the corresponding angular and linear velocities to reach it, using the implementation of the block *Control Law*. This control law can be designed in different ways: see, for example, a traditional approach called here *Villela* [16] for short, or the one developed previously by the authors named *IPC* [15]. This latter control law is described by Eq. 10.

$$\begin{aligned} v &= \min\left\{K_v p\left(\theta_e\right) d, v_{\max}\right\}, \\ \omega &= K_p \sin\left(\theta_e\right) + K_i \int_0^t \theta_e dt, \end{aligned} \tag{10}$$

where, $p(\theta_e) = 1 - |\theta_e|/\pi$, for $\theta_e \in [-\pi, \pi]$ and K_v, K_p, and K_i are tuning parameters of the control law.

C. MOBILE ROBOT POSITION CONTROL WITH RL

The main idea is to control the mobile robot using RL instead of a traditional control algorithm. To this end, the block *Control Law* has to be replaced with a trained RL model. During the training stage, this algorithm learns how to obtain

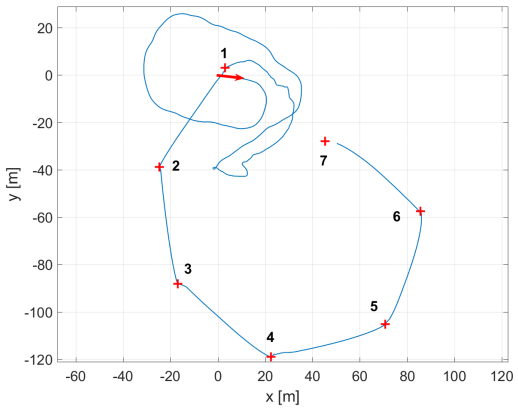

FIGURE 4. Learning stage example.

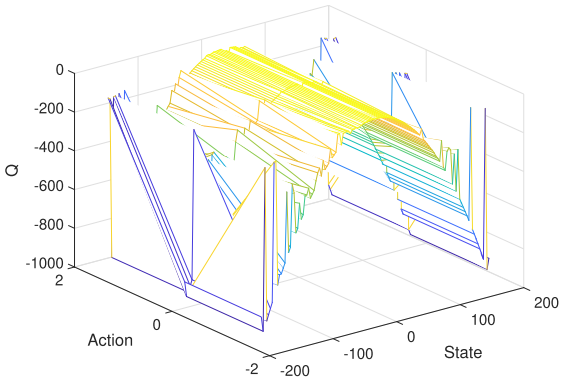

FIGURE 5. Matrix Q at the end of the learning stage, States in degrees (θ_e), and Action in radians per second (ω).

the velocities (v, ω) using distance (d) and angle (α) to the destination point.

For simplicity, the model is trained initially to manipulate only the angular velocity (ω) depending on the angle error (θ_e). This means that the linear velocity (v) can be held constant, at a maximum value, while the robot is outside of the docking area. The case that considers the manipulation of both velocities (v and ω) at the same time is also studied here.

As was mentioned before, the process to build the RL control law is divided into two stages. During the first stage, the learning matrix Q is built. The entries to this matrix are composed of pairs (state, action), which are the states of the robot and its corresponding actions. The state, in this case, is the angle error (θ_e), and the action (control signal) can be the angular velocity (ω) of the robot. The other state, distance d, is not presently used in the algorithm. Each pair (state, action) generates the rewards that comprise the matrix Q. These rewards depend on the criterion that wants to be applied for the specific task to solve. In this approach, the criterion is to negatively penalize the absolute deviations of the controlled signal, which are any changes in the error angle of the robot, i.e. $\rho(x_k, u_k) = -|\theta_e|$.

At the beginning of this stage, the algorithm finds in the matrix Q the current state of the robot using the current angle error. From this state the robot determines if it advances randomly or if it advances taking into account the principle of switching between exploration (uncharted territory) and exploitation (of current knowledge). The outcome is a value of the angular velocity selected from the actions set, which is then executed by the robot. This process is repeated until a value defined by the user indicates the end of the search. The number of iterations depends on the obtained results and it is adjusted by trial and error. Note that the implementation of the algorithm considers the inclusion of disturbances by the modification of the distance and angle error to the target point after each trial. The idea is to cover different scenarios to improve learning. These disturbances are supplied at the end of each learning run. At this time the accumulated rewards are rewritten into the matrix Q using equation 4. Figure 4 shows an example of this stage after 5 million iterations.

The blue line represents the trajectory followed by the robot and the red crosses represent the destination points. The red arrow represents the initial position (0;0) and orientation of the robot. As can be seen, when the process starts, the robot has to reach point number 1 from its initial position at the origin. In this part of the learning process, its behavior is mostly random because it is building the matrix Q using aleatory actions without prior knowledge of the environment. After a big number of trials, the first target is reached.

From point number 1 to point number 2, the built matrix Q is used to reach this destination point, which means that the robot uses the current knowledge to make that task. For the rest of the target points (3, 4, 5, 6 and 7), the process is repeated. Note that for these last targets the behavior of the robot is much more precise, which means that the robot has learned how to go from its current position to a predefined target. Figure 5 shows the resulting matrix Q after the learning stage of this example for 5 million iterations.

In the case of the use of artificial neural networks for the implementation of RL, i.e. the mapping of the policy function and of the value function, stability is to be achieved upon convergence to the optimal fixed points ([17] and [23]). The present case is based on the Q-learning approach, where an n-dimensional matrix is obtained during the learning method, so no direct stability analysis is made.

IV. SIMULATION AND EXPERIMENTAL RESULTS

This section shows the results of the application of the Reinforcement Learning algorithm in simulation and experimental environments. The results are compared with the application of the Villela and IPC algorithms for the position control problem. Note that all the experiments are performed with the same initial conditions.

A. SIMULATION RESULTS

This subsection shows the results of the simulation for different tests on various iterations to build the matrix Q. The matrix Q is obtained in MATLAB during the learning stage, and it is exported to Python (Spyder-Anaconda IDE). The V-REP software is connected to Spyder via Remote API, making it compatible for Python programming. Simulation tests are carried out in V-REP using the KH4VREP

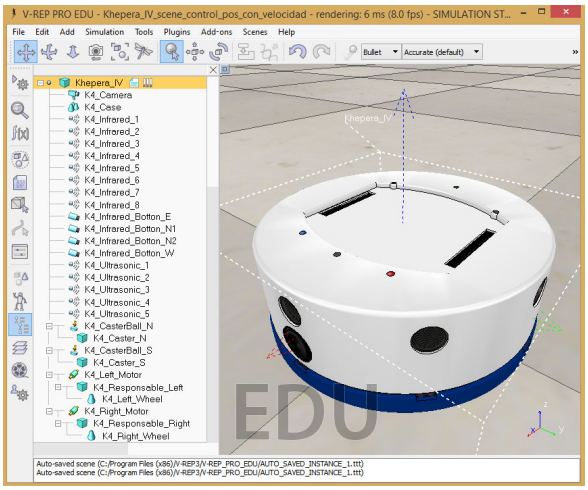

FIGURE 6. Software V-REP with KH4VREP library.

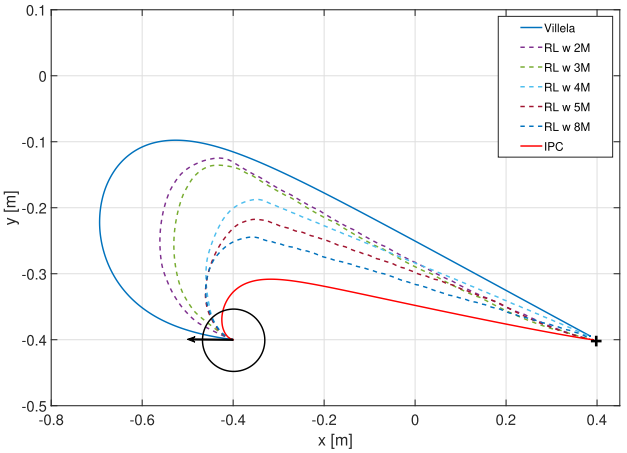

FIGURE 7. Obtained trajectories for each control algorithm (Villela, IPC and RLw).

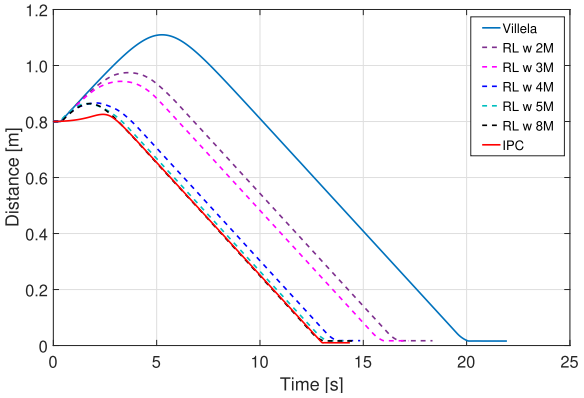

FIGURE 8. Distance to the target vs. time for each control algorithm (Villela, IPC and RLw).

library, which has been developed by the authors [33]–[35]. Figure 6 shows a view of the GUI of the library.

In the learning stage, the algorithm builds the matrix Q to learn how to reach the destination point. To this end, the angle error (θ_e) is used to obtain the angular velocity (ω) in order to control the position of the robot. Note that initially, the linear velocity (v) is kept constant at its maximum value until the robot reaches the docking area.

The matrix Q is composed of the sets (*state*, *action*), where the state is the angle error (θ_e), and the action is the angular velocity (ω). The criterion for obtaining the rewards of matrix Q is to penalize the significant changes in the angle error of the robot and the little changes in the angular velocity of the robot. The states have been divided into 126 regularly spaced values between $-180°$ and $180°$. The actions have been divided into 20 regularly spaced values between $-\pi/2$ and $\pi/2$ in radians per second. Thus, the matrix Q is 2-dimensional, and its size is 126×20. The procedure to populate the matrix Q (i.e. the learning stage) in MATLAB takes around 20 minutes. Note that, the discretization interval can be reduced arbitrarily. However, it should be considered that the more values the speed and error have, the more complex the Q-matrix becomes, and the training time can increase considerably, with no guarantee that this will result in better system performance.

Figure 7 shows the results of the position control experiment for different algorithms (Villela, IPC, and RL). The lines describe the trajectories followed by the robot for each control law. The initial position of the robot is represented by the black circle at $(-0.4; -0.4)$, and the target point is represented by the black cross located at $(0.4; -0.4)$.

The initial orientation of the robot is represented by the black arrow and its value is $180°$. In all cases, the initial conditions are the same, which means that the angle error between the initial position of the robot and the target point is $180°$. Note that, the parameters used in the IPC algorithm were finely tuned, then closely giving the best response possible for that kind of controller ($K_v = 0.15$, $K_p = 1.5$ and $K_i = 10^{-5}$).

The continuous blue line represents the control algorithm of Villela. This control law shows the longer trajectory of all the algorithms. On the other hand, the shortest trajectory is shown by the IPC control law, which is represented by the continuous red line. The rest of the dashed lines are the results of the implementation of the RL algorithm for different numbers of iterations of the matrix Q learning stage. For example, RLw2M represents the performance of the implemented control law with the matrix Q after 2 million iterations, and the line RLw8M describes the results of the RL algorithm for 8 million iterations on the learning stage. As can be seen, when the number of iterations is increased, the trajectory to the destination point is improved, which means that the robot describes a shorter path to reach the target point. Note that all RL trajectories show shorter paths than the Villela algorithm but, all of them are larger than the IPC control law.

Figure 8 shows the distance to the destination point for these experiments. The y-axis represents the distance in meters and the x-axis represents the time in seconds. As can be seen, the distance that takes more time (around 20s) to reach the destination point is the Villela algorithm (continuous blue line).

TABLE 1. Time to reach the target point for each algorithm.

Algorithm	Time (s)
Villela	20.05
RLw2M	16.70
RLw3M	15.95
RLw4M	13.75
RLw5M	13.30
RLw8M	13.10
IPC	13.05

TABLE 2. Performance indexes for each algorithm.

Index	Villela	RL2M	RL3M	RL5M	RL8M	IPC
IAE	13.89	10.15	9.37	6.8	6.63	6.57
ISE	11.86	7.7	6.89	4.47	4.32	4.21
ITSE	76.71	38.94	32.77	16.34	15.48	15.37
ITAE	104.56	61.85	54.04	31.51	30.24	30.14

On the other hand, the IPC distance (continuous red line) needs a shorter period of time (around 13s) to reach the destination. The remaining distances (dashed lines) depict the expected behavior for the RL algorithms. The more iteration has the matrix Q, the less time to reach the target point is needed. These results show that the improvements of the trajectories are not only in terms of distance, but also in time. Note that, the difference between the best and the worst trajectory is around 8s, while the difference between the IPC and RLw8M is only 0.05s. Table 1 shows the time to reach the target point for all experiments.

The quality of each control algorithm can be evaluated by using performance indexes. These indexes use the integral of the error, which is, in our case, the distance to the target point. The performance indexes considered in this work are the following: 1) Integral Square Error (ISE), 2) Integral Absolute Error (IAE), 3) Integral time Squared Error (ITSE), and 4) Integral time Absolute Error (ITAE). Note that the last two also include the time in the analysis [36]. Table 2 shows the performance indexes to compare the results of each algorithm. Note that a lower value of the index implies a shorter path.

As can be seen, the IPC algorithm shows the lower value for all indexes, which means that this algorithm has better performance because the trajectory is the shortest. While the RL algorithm shows good performance indexes, they never improve the IPC results mainly due to the RL algorithm only manipulates the angular velocity of the robot (ω). That is the reason it never surpasses the IPC algorithm performance no matter the number of iteration in the calculation of the matrix Q.

A way to improve the results of the RL algorithm is to include the linear velocity of the robot (v) in the learning stage. As in the previous case, the robot learns how to reach the destination point, but in this case, by manipulating the angular velocity (ω) and also, the linear velocity (v).

To this end, the matrix Q is composed of the sets (*state*, *action1*, *action2*), and it is also built in the learning stage. In this case, the state is the angle error (θ_e), and the actions are linear (v) and angular (ω) velocities. The criterion for obtaining the rewards of matrix Q is to penalize the

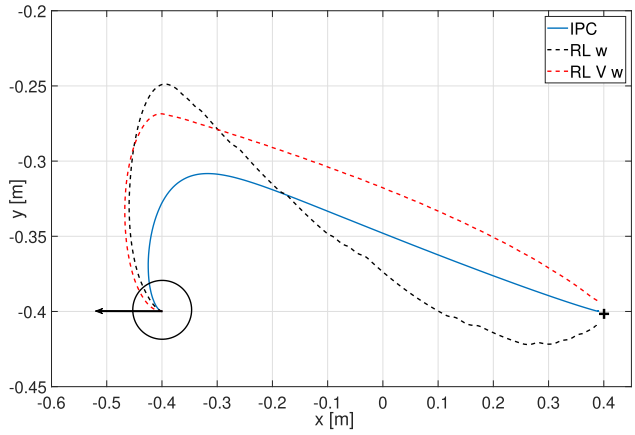

FIGURE 9. Trajectories obtained for control algorithm IPC and Reinforcement Learning (RLVw).

TABLE 3. Performance indexes for IPC and RL algorithms.

Index	IPC	RLw	RLVw
IAE	6.57	6.68	6.49
ISE	4.21	4.34	4.23
ITSE	15.38	15.62	14.94
ITAE	30.14	30.78	29.16

significant changes in the error angle of the robot and the little changes in the velocities of the robot.

The state is divided into 126 regularly spaced values between $-180°$ and $180°$. The actions have been divided into 20 regularly spaced values between $-\pi/2$ and $\pi/2$ for ω, and divided into 27 regularly spaced values between 0 and 0.08 for v. Thus, the matrix Q is 3-dimensional, and its size is 126×20x27. The procedure of completing the matrix Q in MATLAB is very complicated, and that increases the training time up to two hours.

Figure 9 shows the results of the simulation for the position control of the robot using IPC and RL. For this latter algorithm, the RL considers two cases: RLw (i.e. manipulating only ω), and RLwV (i.e. manipulating both velocities v and ω). As in the previous case, the initial position of the robot is represented by the black circle at $(-0.4; -0.4)$. The target point is represented by the black cross located at $(0.4; -0.4)$. The initial orientation of the robot is represented by the black arrow and its value is $180°$. The lines describe the trajectories followed by the robot for each algorithm. In the figure, RLw is represented by the black dashed line, RLwV is represented by the red dashed line and the IPC is represented by the solid blue line.

For the RL algorithms, the simulation was performed with the Q matrix of 100 million iterations in both cases. As can be seen, the IPC algorithm shows the shortest trajectory (but not necessarily the fastest) to the target point. As in the previous case, table 3 shows the performance indexes obtained for the IPC algorithm and the RL algorithms.

As can be seen, the performance indexes are very similar. However, the performance indexes IAE, ITSE, and ITAE show better results in the case of RLVw in comparison to the IPC algorithm.

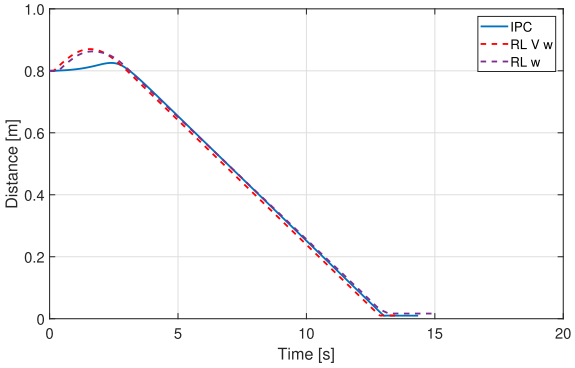

FIGURE 10. Distance to the target for control algorithm IPC and Reinforcement Learning (RLVw).

Figure 10 shows the distance to the destination point for these experiments. The y-axis represents the distance in meters and the x-axis represents the time in seconds. The solid blue line represents the control algorithm IPC and the dashed lines represent the RL algorithms.

As can be seen, the RLVw algorithm shows the fastest performance, reaching the target point at 12.85s. The IPC algorithm takes 13.10s to reach the target point. Finally, the longest duration corresponds to RLw with 13.25s. Note that, the maximum peaks of the distance belong to the RL algorithms, even RLVw arrives a few moments before the IPC algorithm. This is because the IPC algorithm has a small delay at the beginning, when it is turning to get the right orientation. Due to the integral action of the IPC controller, the linear speed increases slowly until the robot can modify its orientation.

Once the position of the robot is controlled, it is interesting to add obstacles in the trajectory of the robot and try to avoid them. This problem can be addressed in different ways. In this case, due to the type of proximity sensors of the Khepera IV robot, the well-known Braitenberg algorithm [39] was selected. This algorithm creates a weighted matrix that converts the sensor inputs into motor speeds. This matrix is a two-dimensional array with the number of columns corresponding to the number of obstacle sensors (8) and the number of rows corresponding to the number of motors (2). The weights of the matrix are determined empirically depending on the location of the sensors in the robot [33]. Note that the obstacles avoidance algorithm can be considered as a disturbance in the motors speeds for the control position problem (see figure 3).

Figure 11 shows the obstacle avoidance experiment using the control laws involved in this research. The configuration shows one obstacle in grey color. The robot (solid small circle) starts the motion from position (-0.4;-0.4) with $-180°$ of orientation (represented by the black arrow). The dotted circle represents the vision margin of the object detection sensors used in the Braitenberg algorithm. The robot must reach the target point, which is represented by the black cross situated at (0.4;-0.4).

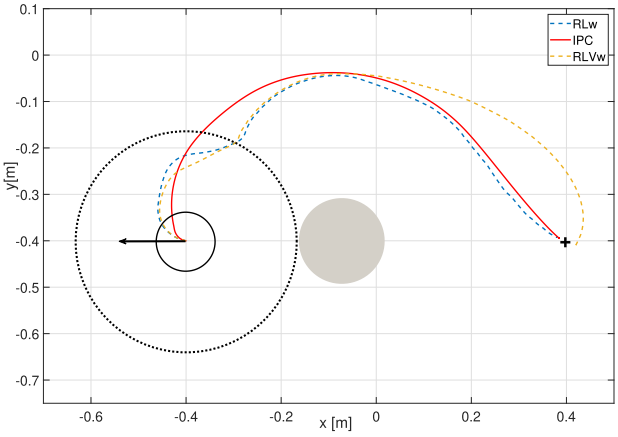

FIGURE 11. Braitenberg algorithm for obstacles avoidance with IPC, RLw, and RLVw.

TABLE 4. Performance indexes for Braitenberg Algorithm with IPC and RL control laws.

Index	IPC	RLw	RLVw
IAE	10.56	8.61	8.91
ISE	7.25	5.71	5.79
ITSE	42.48	26.88	28.49
ITAE	73.69	50.17	54.65

As can be seen, in all cases, the robot reaches the target point avoiding the obstacle and describing similar trajectories. Table 4 shows the performance indexes obtained for all experiments. As can be seen, the performance indexes are very similar; however, the RLw algorithm shows the best results.

B. EXPERIMENTAL RESULTS

This subsection shows the experimental results that have been carried out in the platform [5] previously developed by the authors. The components of the platform are the following: a Khepera IV robot, an arena with a dimension of 2m x 1.5m, a USB camera, a desktop computer that runs SwisTrack [37] and Easy Java Simulations (EJS) [38], and a WI-FI router, which is in charge of the communications between the robot and the computer. Note that the camera, the SwisTrack software, and the WI-FI router work as an Indoor Positioning Sensor (IPS) or indoor GPS, to provide the absolute position and orientation of the robot.

Figure 12 shows the overall block diagram of the control architecture. The images obtained by the camera are processed by the SwisTrack software (block 3), which obtains the current position and orientation of the robot. After that, the distance and angle error are calculated. The control action (velocities v and ω) are then obtained by applying the RL algorithm (block 5). Finally, these velocities are sent by the WI-FI network to the robot, which updates the left and right motor speeds (block 2).

Figure 13 shows the position control experiment. The initial conditions of the experiments results are the same as in simulation. The blue line represents the Villela algorithm and the brown line represents the IPC control law. The rest of the lines are the results for the implementation of the RLw

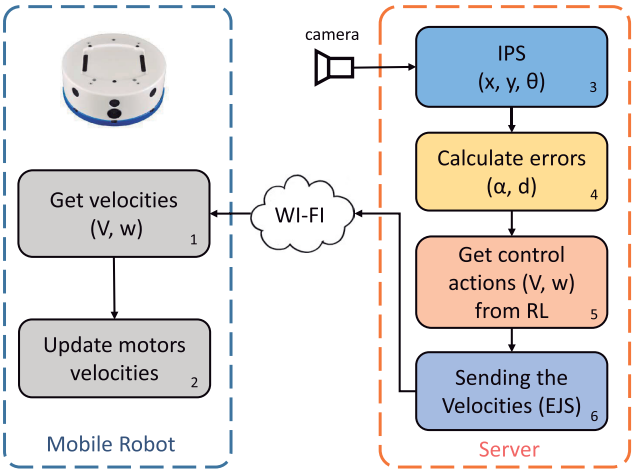

FIGURE 12. Block diagram of the proposed control architecture.

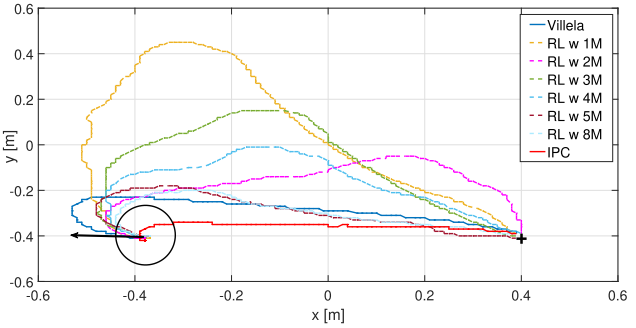

FIGURE 13. Position control experiment for each algorithm (Villela, IPC and RLw).

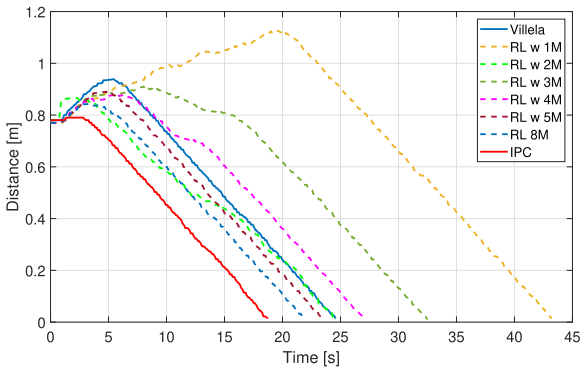

FIGURE 14. Distance to the target for each control law (Villela, IPC and RLw).

algorithm for different numbers of iterations in the calculation of the matrix Q.

As expected, when the number of iterations is increased, the trajectory to the destination point is improved, which means that the robot describes a shorter trajectory to reach the target point. The experimental results on the increase in the number of iterations of the RL algorithm confirm the results obtained in simulation. The Villela algorithm shows a better performance than the RLw algorithms with few iterations (less than 5 millions of iterations), but RLw cases with higher iterations (5 and 8 millions of iterations) show better results.

TABLE 5. Arrival time for each algorithm in experimental environment.

Algorithm	time (s)
Villela	24.58
RLw1M	43.21
RLw2M	24.49
RLw3M	32.50
RLw4M	27.16
RLw5M	23.51
RLw8M	21.84
IPC	18.74

TABLE 6. Performance indexes for each algorithm.

Index	Villela	RL2M	RL3M	RL5M	RL8M	IPC
IAE	13.99	12.36	20.43	12.74	11.28	8.82
ISE	9.99	7.77	15.3	8.7	7.34	5.3
ITSE	73.06	53.33	167.71	60.12	46.3	27.16
ITAE	124.12	108.53	256.31	107.48	87.97	57.61

Figure 14 shows the results for the distance to the target point. The y-axis represents the distance to the target in meters and the x-axis represents the time in seconds. As can be seen, the results confirm that the best performance corresponds to the IPC algorithm, where the robot reaches the destination point in 18s. In the case of the Villela algorithm, the robot reaches the destination in 25s.

Table 5 shows the time to reach the destination for the algorithms represented in figure 14. As expected, the IPC algorithm shows the best results reaching the destination point in 18.74s. While the RLw2M, RLw5M and RLw8M show better results than Villela algorithm.

Finally, table 6 shows the performance indexes for experimental results with the Villela, IPC and RLw algorithms. As can be seen in the last column, the IPC algorithm has the best performance, showing the lowest values for all errors indexes. The results also show that when increasing the number of iterations for the RLw algorithm, the trajectories are improved. As expected, the RLw algorithm never surpassed the IPC results, due to the reasons exposed in previous section.

V. CONCLUSION

In this paper, an algorithm to control the position of a wheeled mobile robot using Reinforcement Learning has been tested in simulation and experimentally. The proposed algorithm is compared with traditional control approaches.

The results show that the proposed controller is able to perform the point stabilization problem successfully. The trajectories of the robot to the destination point are improved with the increase in the iterations on the learning stage. However, manipulating only the angular velocity (ω), it is not enough to overpass the performance of the IPC algorithm. In order to improve the results, the linear velocity (v) was also included in the RL algorithm. This modification makes the learning process more complex and it also increases the training time, but in turn, it is able to get a better performance than the IPC control law in simulation. The implementation in the experimental environment in the lab shows similar results as in simulation.

Future works include the implementation of different control problems in mobile robot using this approach, such as, larger and complex obstacles avoidance, trajectory tracking, master-slaves formation control and consensus algorithm.

REFERENCES

[1] A. Bechar and C. Vigneault, "Agricultural robots for field operations. Part 2: Operations and systems," *Biosyst. Eng.*, vol. 153, pp. 110–128, Jan. 2017.
[2] L. Haiming, L. Weidong, Z. Mei, and C. An, "Algorithm of path planning based on time window for multiple mobile robots in warehousing system," in *Proc. Chin. Control Conf. (CCC)*, Guangzhou, China, Jul. 2019, pp. 2193–2199.
[3] A. Liaqat, W. Hutabarat, D. Tiwari, L. Tinkler, D. Harra, B. Morgan, A. Taylor, T. Lu, and A. Tiwari, "Autonomous mobile robots in manufacturing: Highway code development, simulation, and testing," *Int. J. Adv. Manuf. Technol.*, vol. 104, nos. 9–12, pp. 4617–4628, Oct. 2019.
[4] A. G. Ozkil, Z. Fan, S. Dawids, H. Aanes, J. K. Kristensen, and K. H. Christensen, "Service robots for hospitals: A case study of transportation tasks in a hospital," in *Proc. IEEE Int. Conf. Autom. Logistics*, Aug. 2009, pp. 289–294.
[5] G. Farias, E. Fabregas, E. Peralta, H. Vargas, S. Dormido-Canto, and S. Dormido, "Development of an Easy-to-Use multi-agent platform for teaching mobile robotics," *IEEE Access*, vol. 7, pp. 55885–55897, 2019.
[6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*. Cambridge, MA, USA: MIT Press, 2011.
[7] S. G. Tzafestas, "Mobile robot control and navigation: A global overview," *J. Intell. Robotic Syst.*, vol. 91, no. 1, pp. 35–58, Jul. 2018.
[8] A. Rezaee, "Model predictive controller for mobile robot," *Trans. Environ. Electr. Eng.*, vol. 2, no. 2, pp. 18–23, 2017.
[9] B. Dumitrascu, A. Filipescu, and V. Minzu, "Backstepping control of wheeled mobile robots," in *Proc. 15th Int. Conf. Syst. Theory, Control Comput.*, Oct. 2011, pp. 1–6.
[10] R. Gonzalez, M. Fiacchini, T. Alamo, J. L. Guzman, and F. Rodriguez, "Adaptive control for a mobile robot under slip conditions using an LMI-based approach," *Eur. J. Control*, vol. 16, no. 2, pp. 144–155, Jan. 2010.
[11] S. Zihao, W. Bin, and Z. Ting, "Trajectory tracking control of a spherical robot based on adaptive PID algorithm," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Nanchang, China, Jun. 2019, pp. 5171–5175.
[12] C. Caceres, J. M. Rosario, and D. Amaya, "Approach of kinematic control for a nonholonomic wheeled robot using artificial neural networks and genetic algorithms," in *Proc. Int. Conf. Workshop Bioinspired Intell. (IWOBI)*, Jul. 2017, pp. 1–6.
[13] O. Mohareri, R. Dhaouadi, and A. B. Rad, "Indirect adaptive tracking control of a nonholonomic mobile robot via neural networks," *Neurocomputing*, vol. 88, pp. 54–66, Jul. 2012.
[14] H. Omrane, M. S. Masmoudi, and M. Masmoudi, "Fuzzy logic based control for autonomous mobile robot navigation," *Comput. Intell. Neurosci.*, vol. 2016, pp. 1–10, 2016.
[15] E. Fabregas, G. Farias, E. Aranda-Escolastico, G. Garcia, D. Chaos, S. Dormido-Canto, and S. D. Bencomo, "Simulation and experimental results of a new control strategy for point stabilization of nonholonomic mobile robots," *IEEE Trans. Ind. Electron.*, vol. 67, no. 8, pp. 6679–6687, Aug. 2020.
[16] V. J. Gonzalez, R. Parkin, M. L. Para, and J. M. Dorador, "A wheeled mobile robot with obstacle avoidance capability," *Mechanica Technologia*, no. 1, pp. 150–159, 2004.
[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[18] L. Busoniu, R. Babuska, B. De Schutter, and D. Ernst, *Reinforcement Learning and Dynamic Programming Using Function Approximators*. Boca Raton, FL, USA: CRC Press, 2017.
[19] S. Liang and F. Gan, "Balance control of two-wheeled robot based on reinforcement learning," in *Proc. Int. Conf. Electron. Mech. Eng. Inf. Technol.*, Aug. 2011, pp. 3254–3257.
[20] I. Carlucho, M. De Paula, and G. G. Acosta, "Double Q-PID algorithm for mobile robot control," *Expert Syst. Appl.*, vol. 137, pp. 292–307, Dec. 2019.
[21] X. Ruan, D. Ren, X. Zhu, and J. Huang, "Mobile robot navigation based on deep reinforcement learning," in *Proc. Chin. Control Decis. Conf. (CCDC)*, Jun. 2019, pp. 6174–6178.
[22] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst. (IROS)*, Vancouver, BC, Canada, Sep. 2017, pp. 31–36.
[23] M. Szuster and Z. Hendzel, "Control of Mechatronic Systems," in *Proc. Intell. Optim. Adapt. Control Mechatronic Syst.* Berlin, Germany: Springer, 2017, ch. 8, pp. 255–297.
[24] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 165262–165278, 2019.
[25] T. Jaksch, R. Ortner, and P. Auer, "Near-optimal regret bounds for reinforcement learning," *J. Mach. Learn. Res.*, vol. 11, pp. 1563–1600, 2010.
[26] C. C. White, "Markov decision processes," in *Encyclopedia of Operations Research and Management Science*. New York, NY, USA: Springer, 2001.
[27] C. Watkins, *Learning From Delayed Rewards*. Cambridge, MA, USA: King's College, 1989.
[28] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, nos. 3–4, pp. 279–292, 1992.
[29] Y. Ma, V. Cocquempot, M. El Badaoui El Najjar, and B. Jiang, "Actuator failure compensation for two linked 2WD mobile robots based on multiple-model control," *Int. J. Appl. Math. Comput. Sci.*, vol. 27, no. 4, pp. 763–776, Dec. 2017.
[30] M. C. G. C. Morales, V. V. Alexandrov, and J. E. M. G. Arias, "Dynamic model of a mobile robot with two active wheels and the desing an optimal control for stabilization," in *Proc. IEEE 9th Electron., Robot. Automot. Mech. Conf.*, Nov. 2012, pp. 219–224.
[31] E. Fabregas, G. Farias, S. Dormido-Canto, M. Guinaldo, J. Sánchez, and S. Dormido Bencomo, "Platform for teaching mobile robotics," *J. Intell. Robotic Syst.*, vol. 81, no. 1, pp. 131–143, Jan. 2016.
[32] D. Galán, E. Fabregas, G. Garcia, J. Sáenz, G. Farias, S. Dormido-Canto, and S. Dormido, "Online virtual control laboratory of mobile robots," *IFAC-PapersOnLine*, vol. 51, no. 4, pp. 316–321, 2018.
[33] E. Fabregas, G. Farias, E. Peralta, H. Vargas, and S. Dormido, "Teaching control in mobile robotics with V-REP and a khepera IV library," in *Proc. IEEE Conf. Control Appl. (CCA)*, Sep. 2016, pp. 821–826.
[34] E. Peralta, E. Fabregas, G. Farias, H. Vargas, and S. Dormido, "Development of a Khepera IV library for the V-REP simulator," *IFAC-PapersOnLine*, vol. 49, no. 6, pp. 81–86, 2016.
[35] G. Farias, E. Fabregas, E. Peralta, E. Torres, and S. Dormido, "A Khepera IV library for robotic control education using V-REP," in *Proc. IFAC-PapersOnLine 20th IFAC World Congr.*, Toulouse, France, 2017, pp. 9150–9155.
[36] Y. K. Soni, R. Bhatt, "BF-PSO optimized PID controller design using ISE, IAE, IATE and MSE error criteria," *Int. J. Adv. Res. Comput. Eng. Technol.*, vol. 2, no. 7, pp. 2333–2336, 2013.
[37] T. Lochmatter, P. Roduit, C. Cianci, N. Correll, J. Jacot, and A. Martinoli, "SwisTrack–A flexible open source tracking software for multi-agent systems," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, Nice, France, Sep. 2008, pp. 4004–4010.
[38] F. Esquembre, "Easy java simulations: A software tool to create scientific simulations in java," *Comput. Phys. Commun.*, vol. 156, no. 2, pp. 199–204, Jan. 2004.
[39] M. Shayestegan and M. H. Marhaban, "A braitenberg approach to mobile robot navigation in unknown environments," *Proc. Int. Conf. Intell. Robot., Automat., Manuf.* Berlin, Germany: Springer, Nov. 2012, pp. 75–93.

GONZALO FARIAS received the degree in computer science from the Universidad de la Frontera, Temuco, Chile, in 2001, the Ph.D. degree in control engineering from the National University of Distance Education (UNED), in 2010, and the Ph.D. degree in computer science from the Complutense University of Madrid (UCM), Madrid, Spain, in 2013. Since 2012, he has been with the Electrical Engineering School, Pontificia Universidad Católica de Valparaíso (PUCV). His current research interests include machine learning, pattern recognition, simulation and control of dynamic systems, and engineering education.

GONZALO GARCIA received the B.S. degree in electronics engineering (radar design) from the Naval Polytechnic Academy, Chile, in 1994, the M.S. degree in electronics engineering (automatic control) from Universidad Técnica Federico Santa Maria, Valparaíso, Chile, in 2006, and the Ph.D. degree in aerospace engineering from The University of Kansas, in 2013. He is currently a Postdoctoral Fellow with the Department of Ocean and Mechanical Engineering, Florida Atlantic University (FAU). His research and teaching interests include design of advanced control systems for unmanned aerial, terrestrial, underwater vehicles, including robust control, nonlinear control, optimal control, predictive control, and reinforcement learning.

GUELIS MONTENEGRO received the B.S. degree in electronics engineering from Universidad Técnica Federico Santa Maria (UTFSM), Valparaíso, Chile, in 2016. He is currently pursuing the M.S. degree in sciences of engineering with the Pontificia Universidad Católica de Valparaíso (PUCV). His current research interests include simulation and control of dynamic systems and mobile robotics applied to engineering education.

ERNESTO FABREGAS received the B.S. degree in automation control and the M.S. degree in digital systems from Polytechnic Jose Antonio Echeveria (CUJAE), Havana, Cuba, in 2004 and 2008, respectively, and the Ph.D. degree in computer science from the Universidad Nacional de Educación a Distancia (UNED), Madrid, Spain, in 2013. From 2015 to 2019, he was a Postdoctoral Fellow with UNED, where he has been an Assistant Professor, since 2019. His current research interests include control of multiagent systems, machine learning, mobile robot control, remote laboratories, and engineering education.

SEBASTIÁN DORMIDO-CANTO received the M.S. degree in electronics engineering from Universidad Pontificia de Comillas (ICAI), Madrid, Spain, in 1994, and the Ph.D. degree in physics from the Universidad Nacional de Educación a Distancia (UNED), Madrid, in 2001. Since 1994, he has been with the Department of Computer Science and Automatic Control, UNED, where he is currently a Full Professor of control engineering. His research and teaching interests include automatic control, machine learning, and parallel processing.

SEBASTIÁN DORMIDO (Member, IEEE) received the B.S. degree in physics from the Complutense University of Madrid, Madrid, Spain, in 1968, the Ph.D. degree in science from the University of the Basque Country, Bilbao, Spain, in 1971, and the Doctor Honorary degree from the Universidad de Huelva, in 2007, and the Universidad de Almeria, in 2014. In 1981, he was a Professor of control engineering with UNED, Madrid. He has authored or coauthored more than 350 technical articles in international journals and conferences. He has supervised 38 Ph.D. theses. His scientific research interests include computer control of industrial processes, model-based predictive control, event-based control, and web-based laboratories for distance education. From 2001 to 2006, he was the President of the Spanish Association of Automatic Control, CEA-IFAC. He received the National Automatic Control Prize from the IFAC Spanish Automatic Control Committee, in 2008, and the IFAC Control Education Lifetime Achievement Award, in 2019.