Results in Engineering 24 (2024) 102874

Contents lists available at ScienceDirect

Results in Engineering

journal homepage: www.sciencedirect.com/journal/results-in-engineering

Optimized TD3 algorithm for robust autonomous navigation in crowded and dynamic human-interaction environments

Husam A. Neamah [a,b,c,*], Oscar Agustin Mayorga Mayorga [a]

[a] Department of Electrical Engineering and Mechatronics, Faculty of Engineering, University of Debrecen, Ótemetö Utca 2-4, 4028, Debrecen, Hungary
[b] Technical Engineering College, Al-Ayen University, Thi-Qar 64001, Iraq
[c] Department of Business Management, Al-imam University College, Balad, Iraq

ARTICLE INFO

Keywords:
Mobile robot
Human interaction
Deep reinforcement learning
Autonomous navigation

ABSTRACT

Mobile robots have been incorporated into human society to help perform tasks that can affect or endanger health and life. One of the challenges is in the mobile robot-human interaction, as unexpected movements made by people can cause collisions with the robots when accomplishing a task. In this paper an optimized algorithm based on TD3 (Twin Deep Deterministic Policy gradient) used in Deep Reinforcement Learning is proposed, which allows the robot to take its own actions based on the observations it makes, without defining any trajectory beforehand. Using this algorithm that uses the actor-critical policy that helps to determine the linear and angular velocity of the robot, which allows the robot to move in unknown dynamic environments avoiding collisions. It is proposed to use a buffer that stores the values sent by the neural network and analyses them together with the odometry parameters of the robot to send the best decision to the robot to achieve a collision-free path and meet the objectives. The purpose of this algorithm is to meet as many consecutive targets as possible, i.e. the robot never returns to its initial position, it is assigned new targets regardless of the position it has reached. Finally, the results obtained by training the algorithm correspond nearly 0 for the actor and critic training, with a training of 12,000 episodes, and with an evaluation results 92 % of effectivity of our algorithm, based on 772 steps performed by the rob ten in a time of 11 s.

1. Introduction

Since the beginning of social robotics, the interaction between humans and robots has predominantly been limited to assisting with small, well-defined tasks. Early applications were straightforward, leveraging basic algorithms to manage limited data sets [1]. However, with technological advancements, particularly in processor capabilities and data handling algorithms, the scope and complexity of robotic assistance have significantly expanded. Modern mobile robots are now integrated into various facets of human life, tasked with navigating and performing in dynamic, unpredictable environments alongside humans [2]. One of the foremost challenges in this domain is ensuring safe and efficient mobile robot-human interaction. This involves real-time path planning and collision avoidance in environments where human movements can be sudden and unpredictable [3]. Traditional statistical methods, such as trajectory tracking algorithms based on Monte Carlo methods and Expectation-Maximization (EM), were foundational but often inadequate in highly dynamic settings. For instance, the use of obstacle speed to calculate robot velocity and direction, though useful, frequently led to collisions due to the variability in human movement patterns [4].

To address these limitations, researchers have explored different approaches [5–9]. The main variables that have been most studied have been the calculation of the robot's speed and direction, as in Ref. [10] they use the speed of the obstacles as a reference to calculate these parameters. Other authors often implement social rules based on the psychological study of human behavior [11], using Varoni diagrams to arrive at the final calculation of their velocities. On several occasions these velocities have caused damage or injury to humans, so one technique is to define a safety zone for the robot and the human interacting with the robot [12], often the undetectable shape of the objects causes a miscalculation of the safety zones and therefore the velocities and angles calculated by statistical methods are erroneous [13,14].

As new sensors have been developed to understand the behavior of the environment in which the robot develops [15,16], algorithms have been developed based on two points of view: i) engineering point of view

* Corresponding author.
E-mail address: husam@eng.unideb.hu (H.A. Neamah).

https://doi.org/10.1016/j.rineng.2024.102874
Received 16 June 2024; Received in revised form 20 August 2024; Accepted 8 September 2024
Available online 10 September 2024

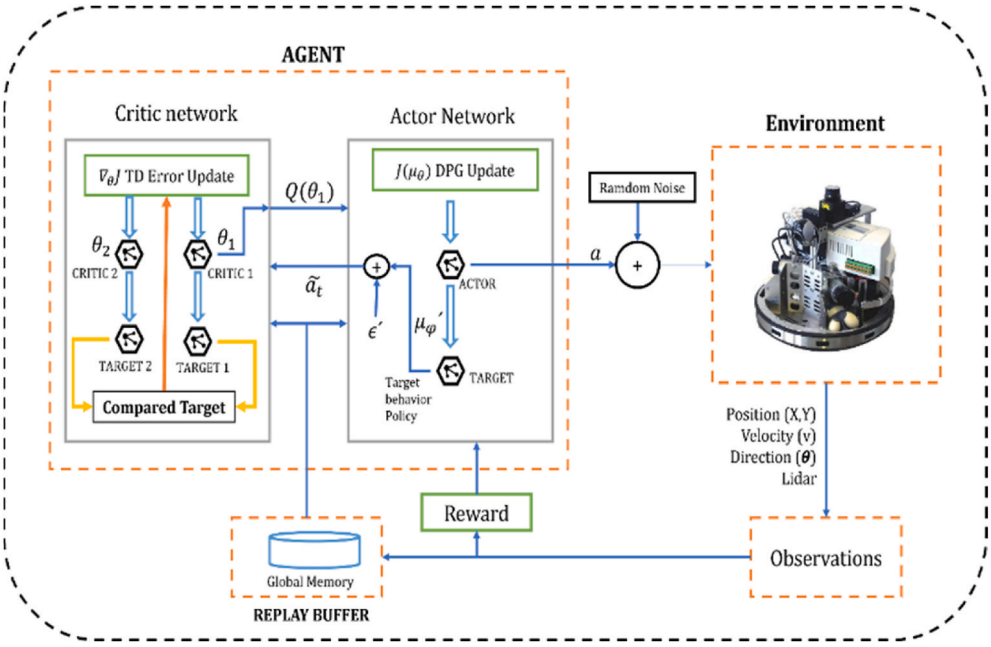

Fig. 1. Control algorithm based on Deep Reinforcement Learning.

for a correct trajectory control and ii) the social point of view, where the human being is the one who accepts the safe movements of the robot [15]. After several studies they realized that the robot must always have reference points in both its X and Y axes [17], all the algorithms described above were put to the test and chosen which ones worked well to continue with the development of these [18].

The development of artificial intelligence algorithms, especially machine learning algorithms, started to be based on Reinforcement Learning specifically with the use of the Q-table [19]. The high cost of developing robots using artificial intelligence algorithms allowed the development of MPC control algorithms and algorithms for low-cost robots [20,21]. Following this line, several studies were created imitating the behavior of human cells to carry out a task [22]. With the advancement of graphical memories, concepts were also created that use their power to create decision trees with specific rules [23]. The use of Deep Reinforcement Learning algorithms has been evolving, starting with the policy gradient method [24], which was tested and implemented on several robots causing them to collide due to low decision data [25]. In recent years, Deep Learning algorithms implemented in Reinforcement Learning have been used, but their implementation has been complicated by the data processing rate required [26–28].

The focus in SLAM is to create a safe path where the robot can avoid the dynamic obstacles and accomplish its goal, an A* algorithm is used to find the static objects and the shortest path between the start point and the end point, to all this, it is integrated with the object velocity method proposed in Ref. [29], and thus a real-time method is realized. Several novel methods have been implemented, one of them is the hybrid FA-TPM method, this FA method is a metaheuristic algorithm developed by Xin Yang [30], which is inspired by the behavior of fireflies.

In order to know the position of the robots in Ref. [31] a sensor network has been developed in an industrial environment to know the floor, walls, equipment and if there are other robots moving in the same area. Simultaneously to this network the robot has its own odometry and with this it has achieved the avoidance of dynamic objects. A novel algorithm has been developed in Refs. [32,33] that simulates the search process of ants to find food. Because it first performs a total mapping of the target called the visual domain and based on this it creates small targets that must be fulfilled until the main target is met [34,35].

The technology of autonomous mobile robots has been introduced to real life, interacting with the random movement of humans. During this interaction several problems have arisen, one of them is the localization and mapping of where the mobile robot is executing tasks [36]. Therefore, several techniques have been developed to help improve the localization and collision with humans known as dynamic objects by researchers. The algorithm developed in Ref. [37], proposes a Long-Term Static Mapping and Cloning Localization, which helps to improve localization in dynamic environments by statically reflecting objects to improve the mapping process [38].

One of the biggest problems that has been faced in the development of algorithms for human-robot mobile interaction is the information processing and thus the computational burden. The non-parametric algorithm developed for low-cost mobile robots in Ref. [39], this algorithm predicts the position of objects using the auto-regressive Gaussian process, to overcome the limitations partially observed by the robot. When trying to avoid the path of dynamic targets, algorithms that predict the future of dynamic targets have been developed such as WMR, TBM and PSO developed by Refs. [40–42]respectively. These algorithms have included the use of neural networks and machine learning algorithms to find the most suitable path for the robot to follow.

The use of the reinforcement learning algorithm used in Ref. [43], incorporates the Q-Learning mechanism by choosing the policy focused on the penalty every time the robot has a collision. Then, several modifications have been introduced, such as the seeker optimization algorithm (SOA) used in Ref. [44], which improves the interaction of the robot with humans. As this artificial intelligence algorithm performs well, advanced control algorithms have been introduced together with reinforcement learning, as in Refs. [5,45], combining DRL with fuzzy logic and Curriculum Prioritization. And using channel-based Convolutional Laser Network architecture where no prior knowledge of the dynamic obstacles is needed.

The primary contribution of this work is the development and optimization of a Deep Reinforcement Learning algorithm tailored for human-robot interaction in dynamic environments. Most robots have a limitation which is the trajectory they must follow. To overcome this barrier the new algorithm allows to have a dynamic physical interaction in a safe way. This means that the robot does not have to follow a desired trajectory, based on its training it decides its movements and generates a new trajectory that allows it to avoid collision with humans. Ensuring a safe exploration and thus the fulfillment of the objective. This involves

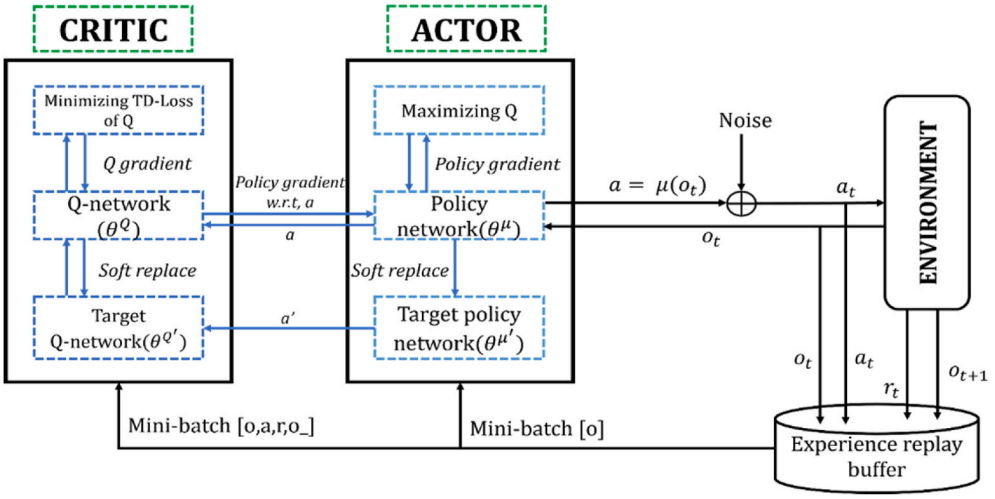

Fig. 2. Deep deterministic Policy Gradient Method.

enhancing the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm by fine-tuning key parameters such as batch size, discount factor, learning rate, noise, and update frequency. These optimizations allow the algorithm to make more accurate decisions and train more efficiently, enabling the robot to navigate and reach multiple targets consecutively without returning to its starting point. The use of a replay buffer and deep neural networks further strengthens the algorithm's robustness by enabling learning from diverse scenarios. Validated in static, semi-dynamic, and fully dynamic environments, the optimized TD3 (O-TD3) algorithm demonstrates superior performance in terms of safety, efficiency, and decision-making when compared to DQN and standard TD3. Metrics such as actor and critic losses, average rewards, and success rates substantiate these improvements. Additionally, the algorithm's efficient path planning and reduced computational burden optimize energy consumption, making it suitable for robots with limited resources. These advancements address fundamental challenges in autonomous navigation and human-robot interaction in crowded and dynamic environments, providing a robust and efficient solution for real-world applications.

2. Controller algorithm

The proposed control algorithm consists of Deep Reinforcement Learning techniques that allow the programming and execution of the method in an efficient way. The method used is the TD3 (Twin Delayed DDPG) algorithm, which allows us to have a closed-loop system with feedback, also allows us to reduce the over estimation on the approximation function (Variable Controller) and to take quick corrective actions when there is a sudden change of the human beings in their direction of movement.

Fig. 1 proposes the control structure used, according to the parameters received by the robot's sensors. This Deep reinforcement Learning algorithm consists of four stages: *i) Agent,* where the TD3 method is used to perform the calculation of the corresponding control action, *ii) Environment*, where the robot receives the action and transforms it into robot movements, *iii) Observations*, these are given by the robot sensors after executing the control commands and *iv) Award*, based on the robot states sent by the sensors, a relation is applied where the robot is told whether the control action was correct or incorrect.

The Agent uses the TD3 method which corresponds to the use of the twin delayed Deep Deterministic Policy Gradient (DDPG). This algorithm has been developing since its inception with the gradient policy to use one actor and one critic. The development of the TD3 policy is detailed below, through the policy gradient methods to finals statements. Considering the case of stochastic parameterized policy, π_θ, *Policy gradient method* estimates weights of an optimal policy through gradient ascent by maximizing expected cumulative reward ($J(\pi_\theta) = E[R(\tau)]$), which an agent gets offer taking optional action in each state.

$$J(\theta) = J(\theta) + \alpha \nabla_\theta J(\pi_\theta) \tag{1}$$

where:

J = cumulative reward, α = learning rate, θ = parameter of target-policy.

Subsequently, in the case of the *deterministic policies*, policy μ takes in state space **S** and outputs an action space **A.** Rather than calculating the integral over the actions, it simply need to add over the state space, since the action is deterministic:

$$J(\mu_\theta) = \int_S \rho^\mu r(S, \mu_\theta(S)) ds \tag{2}$$

An advantage of the deterministic policy gradient is that compared to the stochastic policy gradient the deterministic version is simpler and can be computed more efficiently.

$$\nabla_\theta J(\mu_\theta) = \sum_{(S \rho^{\mu^k})} \left[\nabla_\theta \mu_\theta(S) \nabla_\alpha Q^{\mu^k}(S, \alpha) \alpha = \mu\theta(s) \right] \tag{3}$$

Deep Determinist Policy Gradient (DDPG) is an algorithm that learns a Q function and a policy concurrently using off-policy data and the Bellman equation to learn the Q-Function. The algorithm then uses the Q-function to learn the policy. It has two sides, the first is the learning side, where the objective of DDP training is to minimize the Mean-Squared Bellman Error (MSBE) Function, showed in equation (4). Two methods are used to successfully train the network: *I) Replay buffer,* by having a sufficient capacity to hold a valuable collection of past experiences, one can greatly enhance their ability to navigate through life's challenges and achive their goals. *II) Target network,* a method to avoid MSBE minimization instability.

$$L(\phi, D) = E_{(S,\alpha,r,S',\delta)D} \left[\left(Q_\phi(S, \alpha) - \left(r + \gamma(1 - \delta) max\, Q_\phi(S', \alpha') \right) \right)^2 \right] \tag{4}$$

On the other hand, the policy learning Side of DDPG whose aim is to learn a deterministic policy $\mu_\theta(S)$ which gives the action that maximizes $Q_\varnothing(S, \alpha)$. The action space is continuous, and it assume the Q-function is differentiable with respect to action. So, gradient ascent is used to solve the above equation.

$$\max_\theta E_{S-D}[Q_\varnothing(S, \mu_\varnothing(s))] \tag{5}$$

DDPG can archive great performance sometimes, but it is very

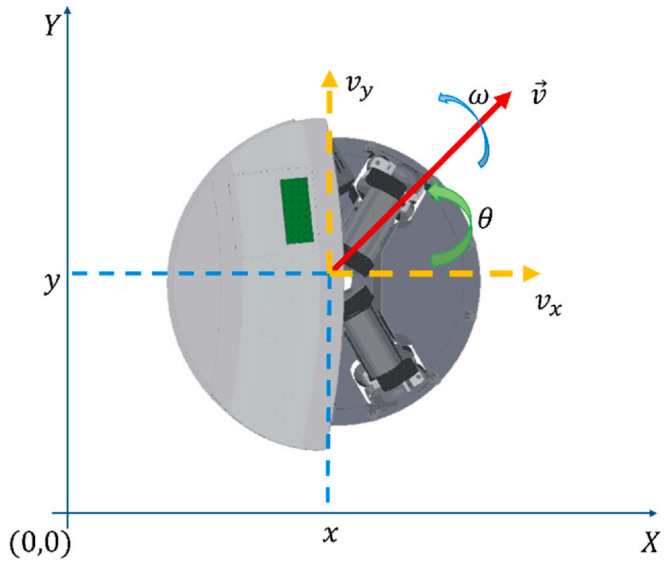

Fig. 3. Robot model in the environment.

sensitive to hyperparameters and the other kinds of tuning. Often, learned Q-function begins to dramatically overestimate Q-values, which then leads to the policy breaking, because it exploits the error in the Q-function. *Twin Delayed DDPG (TD3)* overcomes this issue by introducing three critical tricks: i) Clipped Double-Q learning, it learns two Q-functions instead of one, and employs the lower of the two Q-values to construct the goals in Bellman's error loss functions. ii) "Delayed" Policy Updates, TD3 updates the policy and its target networks at lower frequency than the Q-function. It is suggested to perform one update of the policy for every two updates of the Q-function. iii) Target policy Smoothing. The algorithm introduces noise into the objective action to prevent the policy from taking advantage of errors in the Q-function, thus smoothing Q throughout the action changes.

As can be seen in Fig. 2, the agent is constituted by an actor-critic who integrates a Deep $Q-$ *network*, and where their functions are linked to determine the agent's output action. This type of actor-critical architecture takes advantage of value-based and policy-based learning methods. Here's how the Deep deterministic policy gradient algorithm works. The **critic** estimates the output function Q which is the value-action. The values collected by the observations go into the agent and Q determines a reward according to the parameters they enter o_t in accordance with the policy π. This is denoted in equation (6).

$$Q^{\pi}(o_t, a_t) = \mathbb{E}_{\pi}[R_t|o_t, a_t] \tag{6}$$

Where, $\mathbb{E}_{\pi}$ it is the expected output with the state space (equivalent to the observations taken) and the subsequent actions following the policy π. In Deep Reinforcement Learning, the Q it is represented concurrently by Bellman's equation (7):

$$Q^{\pi}(o_t, a_t) = \mathbb{E}_{\pi}[r(o_t, a_t) + \gamma \mathbb{E}_{\pi}[Q^{\pi}(o_{t+1}, a_{t+1})]] \tag{7}$$

In equation (8), it can see how the $Q^{\pi}(o_t, a_t)$ can be approximated by means of a Q-network with parameters θ^{Q}, and then, by means of a deep neural network, the $Q-$ *network*.

$$Q(o_t, a_t|\theta^{Q}] = Q^{\pi}(o_t, a_t) \tag{8}$$

The critic of his gradient policy suggests the direction in which the actor should update the distribution policy. Gradient policy is used in reinforcement learning-free models because the change of states is unknown to the agent. On the other hand, when policy is deterministic, the actor directly assigns observation o_t to action a_t. This can be approximated by a parameterized policy network with θ_{μ} as indicated in equation (9) and so the deep neural network implements the policy network.

$$a_t = \mu(o_t|\theta^{\mu}) \tag{9}$$

After this, the DDPG algorithm has a target network (Q') and a target policy network (μ'). These networks are copies with a network lag Q and the policy network. The networks are then parameterized with: $\theta^{Q'}$ for the network target Q y $\theta^{\mu'}$ for the target policy network. Fig. 2 shows the architecture of the actor-critic of the algorithm that includes 4 neural networks i.e., the Q-network, the deterministic political network, the objective Q-network and the target political network.

When the neural network is deployed, the updated Q-value $(\widetilde{Q})$ according to Bellman's equation is described in equation (10).

$$\widetilde{Q}_t = r(o_t, a_t) + \gamma Q' o_{t+1}, \mu'(o_{t+1}|\theta^{\mu'})|\theta^{Q'}) \tag{10}$$

The Q(Q) network is updated by minimizing the time difference loss (TD) as follows:

$$\min L = \sum \left[Q(o_t, a_t|\theta^{Q}) - \widetilde{Q}_t\right]^2 \tag{11}$$

Where, the sum is the average sum of a mini batch. The aim is to maximize the expected profitability through the political function of the actor μ, that is, the discounted reward that accrues is given by equation

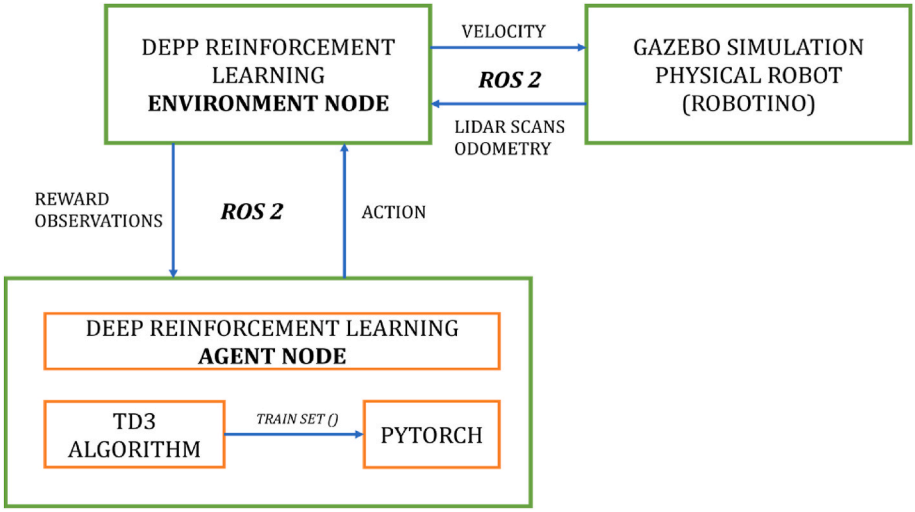

Fig. 4. Navigation System for TD3 DRL algorithm.

Table 1
OTD3 algorithm.

Algoritm 1. Training algorithm for O-TD3 Deep reinforcement Learning with buffer delay	
1	Create the deep neural network for both Critic $(Q_{\theta_1}, Q_{\theta 2})$ and actor π_φ with $\theta_1, \theta_2, \varphi$
2	Initialization of the networks θ_1', θ_2'
3	Initialization of the buffer replay $\mathbb{R}$
4	**For** e = 1 to M **do**
5	Initialization of Local buffer ב
6	**For** t = 0 to T −1 **do**
7	Choose a ramdom goal point q
8	Select a action with noise
9	$a_t = \mu_\varphi\left(q_t \middle\| q_{goal}\right) + \epsilon, \epsilon \sim N(0, \sigma)$
10	$q_{t+1} = q_t + \alpha^* a_t$
11	**If** $q_t + \alpha\left(\mu_\varphi\left(q_t \middle\| q_{goal}\right) + \epsilon\right) \notin$ **Q** then
12	$q_{t+1} = q_t$
13	**Else if** $q_{t+1} \in Q_{collide}$ **then**
14	$q_{t+1} = q_t$
15	**Else if** $\left\|q_{t+1} - q_{goal}\right\| \leq 0.3^*\alpha$ **then**
16	Terminate by goal arrival
17	**End if**
18	
19	$r_t := r\left(q_t, a_t, q_{goal}\right)$
20	Store the transition $\left(q_t \middle\| q_{goal}, a_t, r_t, q_{t+1} \middle\| q_{goal}\right) in \mathbb{R},$ ב
21	
22	Sample mini batch of n transitions from $\mathbb{R}$
23	Update critics with the temporal difference error
24	
25	**If** t mod d **then**
26	Update actor by the deterministic policy gradient
27	Update targets networks
28	θ_i'
29	φ'
30	**End if**
31	**End for**
32	
33	**If** $q_t \neq q_{goal}$ **then**
34	Set additional goal $q'_{goal} = q_t$
35	**For** t = 0 to T-1 do
36	Sample a transition $\left(q_t \middle\| q_{goal}, a_t, r_t, q_{t+1} \middle\| q_{goal}\right)$ *from* ב
37	$r'_t := r\left(q_t, a_t, q'_{goal}\right)$
38	Store the transition $\left(q_t \middle\| q'_{goal}, a_t, r_t, q_{t+1} \middle\| q'_{goal}\right)$ *in* $\mathbb{R}$
39	**End for**
40	**End if**
41	**End for**

(12).

$$\max J = \mathbb{E}\left[Q(o_t, a_t)|_{a=\mu(o_t)}\right] \tag{12}$$

Because the range of actions is continuous, the $Q(o_t, a_t)$ function in relation to the action parameter a is smooth and can be differentiated. Given the differentiability of the policy function, it can use the chain rule to find the derivative of the objective function concerning the policy parameter. Consequently, the policy network (μ) gets updated utilizing the sampled policy gradient as is showed in equation (13).

$$\nabla_{\theta\mu} J = \sum \nabla Q\left(o_t, a|\theta^Q\right)\big|_{a=\mu(o_t)} \nabla_{\theta\mu}(o_t|\theta^\mu) \tag{13}$$

Equations (14) and (15) show that after the process performed by the neural network, the values learned by the Q-network and the network-policies are slowly searched by the target networks. Where, $\rho(0 < \rho \ll 1)$ it controls that the update rate will be slow, thus improving the stability of learning.

$$\theta^{\acute{Q}} \leftarrow \rho\theta^Q + (1-\rho)\theta^{\acute{Q}} \tag{14}$$

$$\theta^{\acute{\mu}} \leftarrow \rho\theta^\mu + (1-\rho)\theta^{\acute{\mu}} \tag{15}$$

Once the DDPG algorithm has been realized, it is very fragile when the Q-function starts to overestimate the Q-value. That is why the TD3 algorithm implements another critical value where after performing the algorithm process a comparison is made, the smallest value of the target in the Bellman loss error function is used to follow the actor. This small value provides greater approximation stability and improves the stability of the algorithm.

The environment refers to the robot and the values of the variables perceived by the sensors of the place where it is located. The robot is an omnidirectional system of three coaxial wheels shown in Fig. 3. After that the action is sent by the robot as a 2x1 vector, which corresponds to the lateral force and the rotational force. By using Newton's second law it can be converted into linear velocity and angular velocity to be used by the kinematic description of the robot to provoke movement.

The kinematic model of the omnidirectional robot can be calculated with the derivation of equation (16). What it can observe in this equation: v is the velocity vector in the inertial frame, $\dot{x}$ *and* $\dot{y}$ are the translational velocities along the corresponding axis and the inertial coordinate system, ω is the rotational velocity, $R(\theta)$ is the rotation matrix from body coordinates to inertial coordinates, J is a matrix that contains constraints provided by wheels, $\dot{\theta}$ is a vector that contains rotational velocities of the wheels.

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \omega \end{bmatrix} = R(\theta)^* J^{-1} * \dot{\theta} \tag{16}$$

Substituting the Rotation Matrix $R(\theta)$ and the matrix of contrainst J, the direct kinematics is showed in equation (17).

$$v = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \omega \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} -\frac{r}{\sqrt{3}} & 0 & \frac{r}{\sqrt{3}} \\ \frac{r}{3} & -\frac{2r}{3} & \frac{r}{3} \\ \frac{r}{3L} & \frac{r}{3L} & \frac{r}{3L} \end{bmatrix} * f \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{17}$$

Where: r is the Wheel radius, L is the distance between the center of the robot and the center of the wheel, f is the gear ratio of the robot. After the movement of the robot, the velocities of the wheels of the robot must be known. Base on the inverse kinematic of the robot as showed in

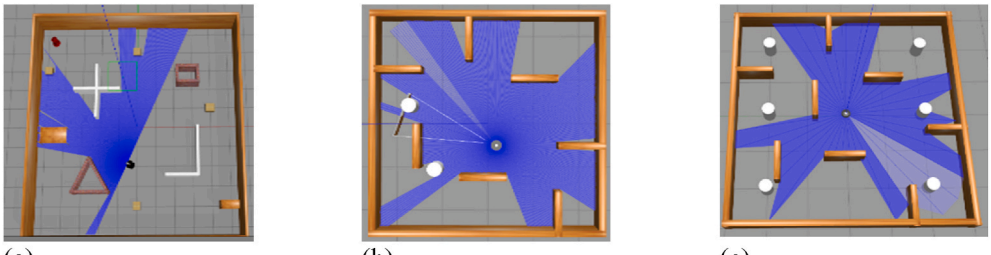

Fig. 5. Environments of DRL training a) Static objects, b) 2 Dynamic obstacles and c) Crowded environment with 6 dynamic obstacles.

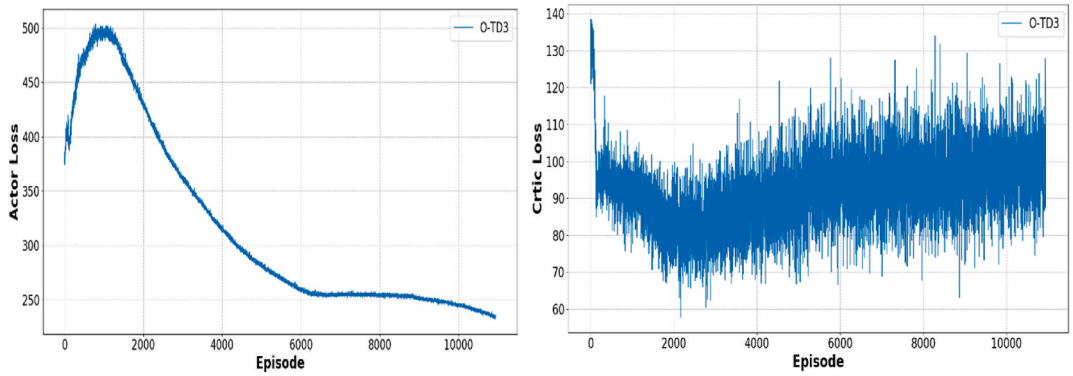

Fig. 6. Average actor and critic loss per episode during the training in crowded environment.

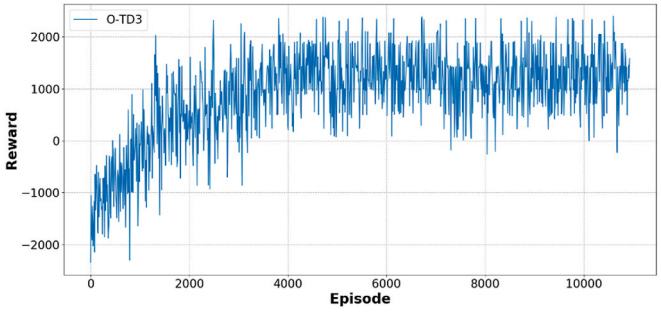

Fig. 7. Average reward per episode during the training in a crowded environment.

equation (18), it can be calcualted.

$$\begin{bmatrix}\dot{\theta}_1\\ \dot{\theta}_2\\ \dot{\theta}_3\end{bmatrix}=\frac{1}{r}\begin{bmatrix}V_1\\ V_2\\ V_3\end{bmatrix}=\frac{1}{r}\begin{bmatrix}-\sin(\varphi+\omega_1) & \cos(\varphi+\omega) & R\\ -\sin(\varphi+\omega_2) & \cos(\varphi+\omega) & R\\ -\sin(\varphi+\omega_3) & \cos(\varphi+\omega) & R\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \omega\end{bmatrix} \tag{18}$$

where: $\dot{\theta}$ is the angular speed of the wheel, V_1 is the linear velocity of the wheel, r is the radius of the wheel, φ is the direction of the robot.

The observations contain parameters that the agent needs to perform the necessary calculations and propose a new action for the future episode. These observations consist of a vector that includes parameters of the robot and environmental variables via the sensors. In this case the parameters needed from the robot are the corresponding odometry: position $P(x,y)$, angular velocity of the wheels $\dot{\theta}(\dot{\theta}_1,\dot{\theta}_2,\dot{\theta}_3)$, velocity $v(\dot{x},\dot{y})$, direction of the robot φ. The last two variables are of vital importance, namely the collision variable and the distance points of the lidar sensor of 360° of the environment *LD*. All these data are sent as a vector as shown in equation 219.

$$O_t=[P,v,\varphi,\dot{\theta},collision,LD]' \tag{19}$$

Once the robot performs a movement, ***the reward*** is necessary because the control algorithm needs to understand whether the sent actions were correct or wrong to achieve the goal. This is done by means of an equation called reward that corresponds to the use of observations. The variables needed are linear velocity, angular velocity, distance between the robot and the target, direction of the target, distance between the nearest obstacle taken from the lidar sensor. When the robot reaches

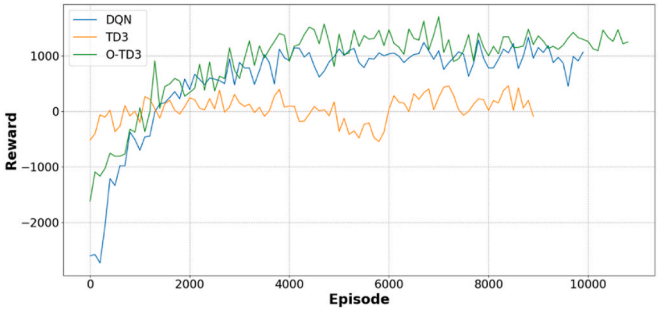

Fig. 9. Reward comparation Between DQN, TD3 and O-TD3 DRL Algorithm.

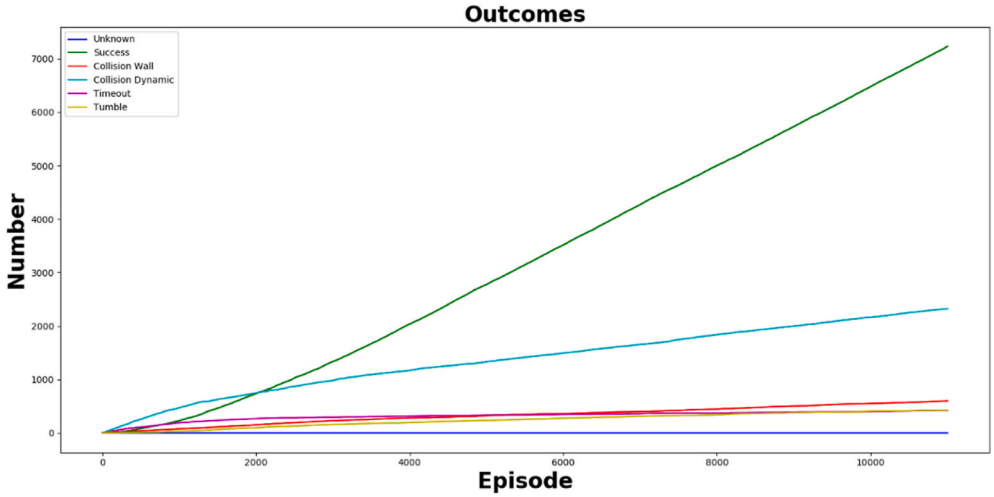

Fig. 8. Learning results of the Deep ReinforcementTD3 Learning algorithm.

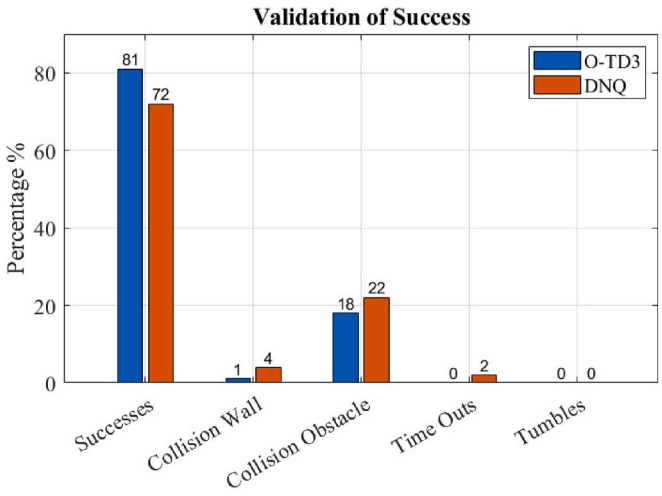

Fig. 10. Comparation of success between DNQ and O-TD3 DRL Algorithm.

the target the agent receives a reward of 2500 points, while if the robot is in collision, it receives a penalty of −2500. If the robot does not meet the two parameters mentioned above, the robot will use the following equation (22) to send a reward value to the agent.

$$reward = v + \omega + minOD + rD + \varphi - 1 \tag{22}$$

Where: v is the module of linear velocity, ω is tthe module of angular velocity, *minOD* correspond to the minimum distance from robot to obstacle, *rD* are calculated from the position of the robot and the position of the goal, φ is the direction of the robot.

3. Methodology

The experiments were carried out using the following software: Robot Operating System (ROS v2), Gazebo Simulator and PyTorch. The whole communication system used is shown in Fig. 4. For the training of the neural network and the processing of the signals that are sent by the agent to the environment, the observations that are made in the environment and directed to the agent, a computer with an Intel Core i7 9th generation processor, 16 Gb RAM, with an NVIDIA GeForce RTX 2060 graphics card with 6 GB VRAM graphics memory was used.

Using tools and software libraries Robot Operating System (ROS2) allows us to create communication nodes through Gazebo and Pytorch, this communication facilitates the training of the proposed algorithm. First the TD3 algorithm is generated in phyton, which sends the training values to Pytorch, which is a library used for deep learning, with these values the algorithm is trained thanks to the use of the CUDA tool, which ensures the use of the graphics card to accelerate the training.

The received action is sent to the environment node created and is transformed into velocity, which is sent to the robot, after the movement the environment node collects the scanned lidar sensor values and odometry values to generate the reward and together with the

Table 2
Results of the evaluation performance in 3 Environments.

Place	Algorithm	Success rate	Time	Steps
Environment 1	*O-TD3*	*69 %*	*8.6*	*477*
	TD3	66 %	16.46	805
	DQN	64 %	10.15	604
Environment2	*O-TD3*	*92 %*	*11.303*	*703*
	TD3	83 %	17.4	876
	DQN	81 %	14.12	912
Environment 3	*O-TD3*	*81 %*	*24.82*	*1163*
	TD3	64 %	43.8	2541
	DQN	72 %	22.975	1458

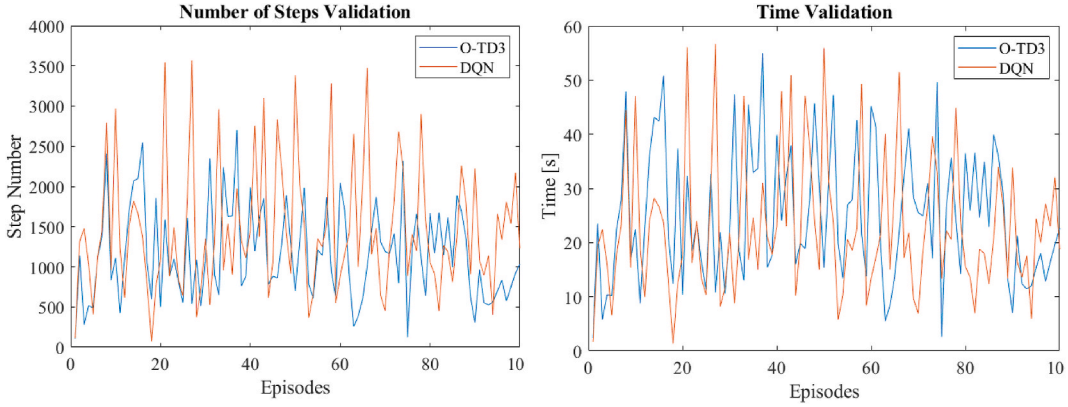

Fig. 11. Comparation of number of steps and Time between DNQ and O-TD3 DRL Algorithm.

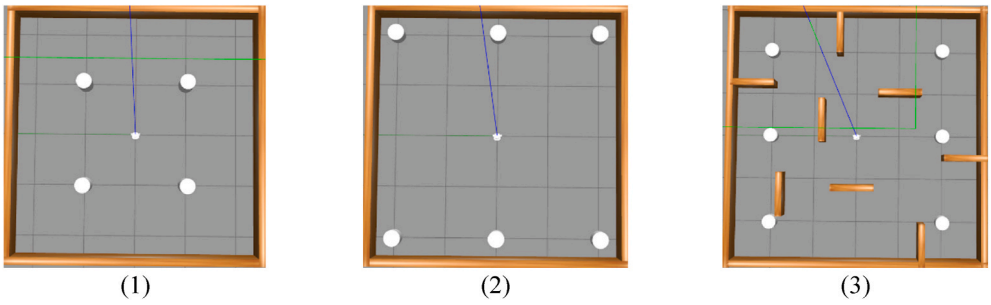

Fig. 12. Evaluation Environments for the algorithm O-TD3.

observations are sent to the training node that corresponds to the agent. The training corresponds to closed-loop control, that is why this whole process is repeated a high number of episodes, where the training values are stored in Pytorch for decision making, and thus achieve an adequate training. The training algorithm is shown in Table 1.

4. Results and discussion

The first outcome is the training results of the proposed algorithm for autonomous navigation of mobile robots with human interaction in dynamic environments. The control algorithm has been implemented in three different scenarios, the first one has been developed in a structured environment where the obstacles are fixed during the DRL training and then in its testing. The second scenario is an environment with fixed walls and two moving objects. These moving objects do not have a defined trajectory, their trajectory is random within the established environment. And finally, to simulate dynamic environments, the DRL algorithm has been trained in an environment with static objects and with 6 dynamic objects that move randomly. All this can be seen in Fig. 5 respectively.

The performance of the proposed algorithm is shown in Fig. 6, the parameters used after a carefully tuning in the TD3 DRL algorithm are: batch size = 1024, discount factor = 0.99, learning rate = 0.003, tau = 0.003, policy noise = 2, policy update frequency = 2. The training in the environments has been done in a progressive way first in the static environment with 5000 training episodes, then in the dynamic environment with 7000 training episodes and finally to fulfil the objective of the autonomous navigation of the mobile robot in dynamic environments.

Fig. 6 shows the average number of actor losses per episode, each episode consisting of a maximum of 100 actions and an execution time of 60 s. The learning curve starts at an average value of 480, takes its maximum value of 500 in episode 1500 and then drops to approximately 280 in episode 6000. This value converges until episode 10000 where the learning curve starts to decrease to zero in episode 15000. When the average tends to zero it means that the algorithm has been trained correctly.

According to Fig. 6, the average loss rate of the agent's critic indicates that the losses start at high values in the first 100 episodes, then decrease with the training and fluctuate between 100 and sixty in the 400th episode. In the last training of the algorithm, the values remain varying between 120 and 80, which is positive for the learning process.

The training results denote how the robot starts to train by having bad actions in the first learning, in which the robot receives a penalty and starts to make new decisions. In episode 1000 the values of wall collision and turning converge, i.e. they take the same value in approximately every situation that occurred in 300 episodes. By episode 1500 the robot's dynamic collision value starts to stop and the victory value starts to rise rapidly, i.e. the actions taken by the agent actor are successful and the robot avoids the static and dynamic obstacles and thus achieves the goal. Finally, Fig. 7, shows the reward received per training episode starting from −2000 and finishing in 2000 in the las episodes. From episode 2100 the algorithm takes correct actions, and the success rate grows dramatically, reaching a difference of 8000 episodes where the robot manages to reach the target without any interruption and with a dynamic collision that stays close to 2000. All this can be seen in Fig. 8.

The validation of the training has been compared with two other algorithms used in Deep Reinforcement Learning that are DQN and the Normal TD3, obtaining a good difference in the reward obtained with the optimization of the algorithm (O-TD3). Because in the episode 100 its value begins to be positive and then this begins to grow until having an average of 1500 per episode in comparison to the normal TD3 that has a value in the middle of zero and DQN that reaches a value of 100 in

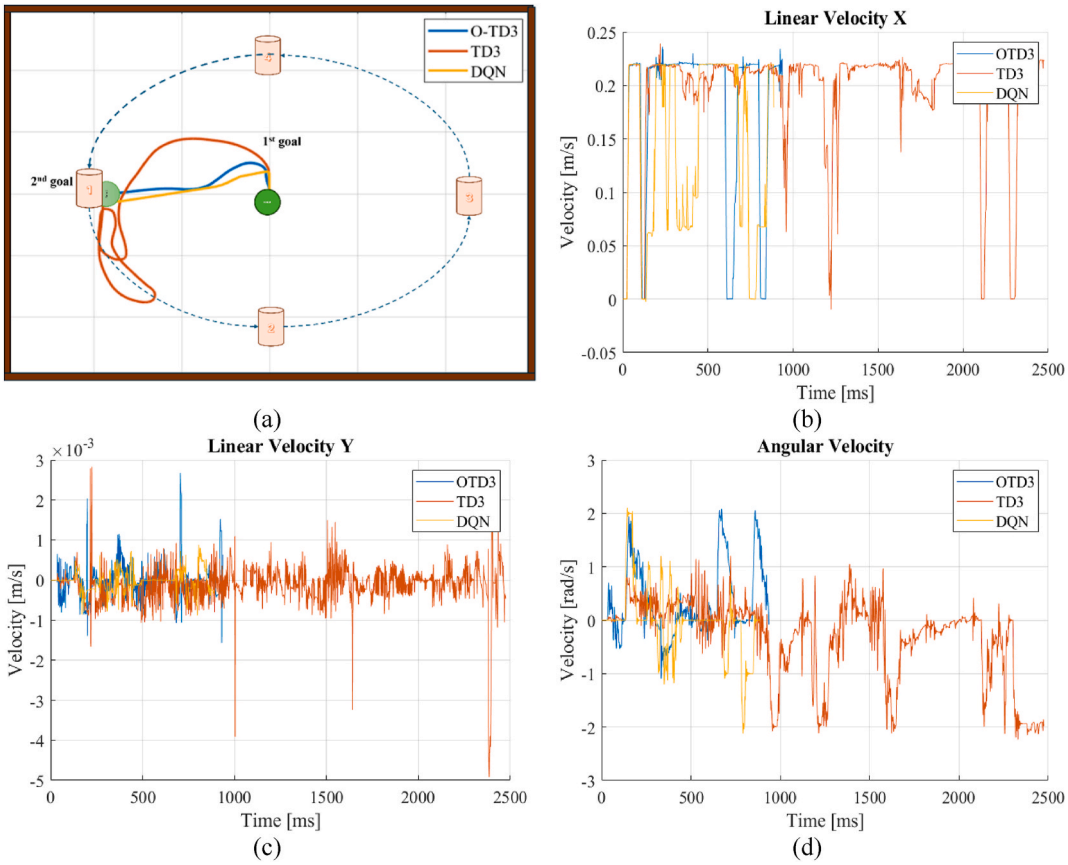

Fig. 13. Path and velocities made by the OTD3, TD3 and DQN algorithm in the performance evaluation in the environment 1.

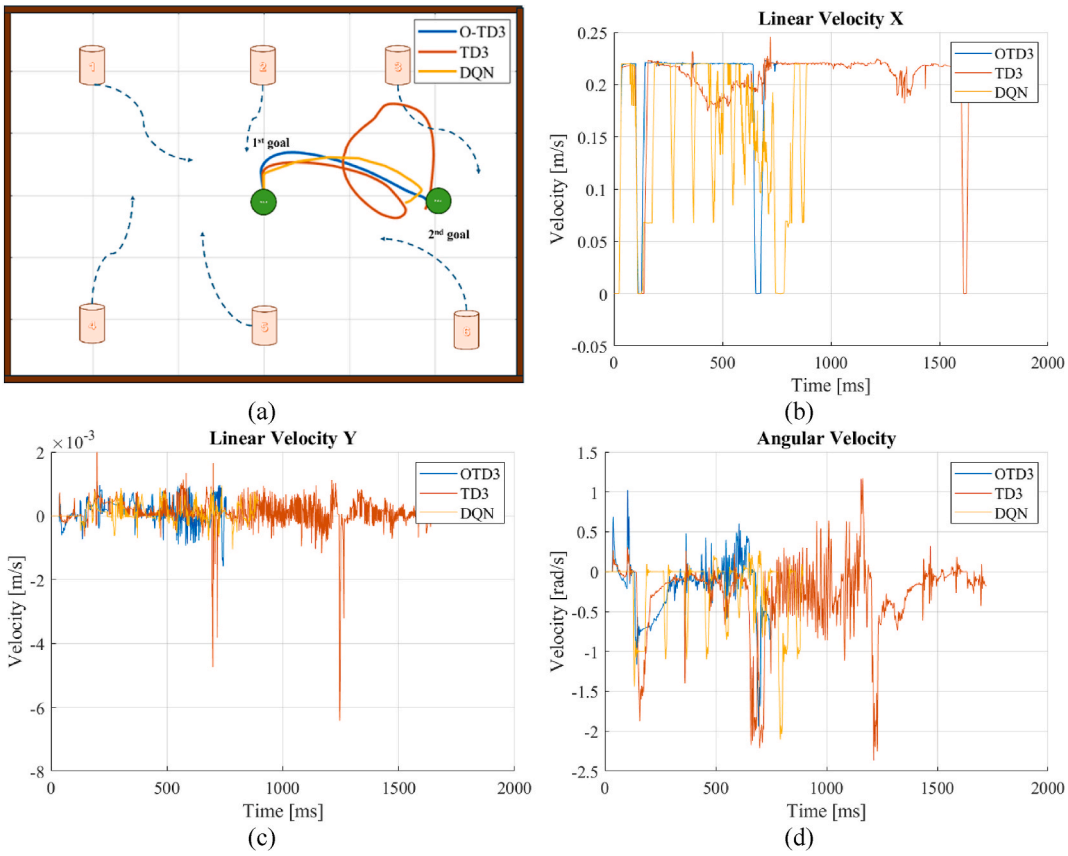

Fig. 14. Path and velocities made by the OTD3, TD3 and DQN algorithm in the performance evaluation in environment 2.

its gain. All this is shown in Fig. 9.

After the comparation, the next step to validate the optimized TD3 algorithm, is to test the robot on 100 consecutive targets. These were compared with the DQN algorithm that showed the best training results and the following results were obtained. The number of successes achieved is 81 %, compared to the 1 % and 18 % collisions with walls and objects respectively, with no time loss and no anomalous robot falls. The number of steps performed by the algorithm averaged 1000 steps compared to 1500 steps performed by O-TD3 and DQN respectively. Finally, the execution time has been relatively low in each episode with an average of 23 s per accomplished goal. All this can be seen in Figs. 10 and 11,

Finally, ***the second result*** is the evaluation, where the performance of our optimized TD3 algorithm was implemented in three different environments shown in Fig. 12. All the movements of the dynamic objects representing the movement of human beings for the mobile human-robot interaction are random. Our algorithm has presented a better response in achieving goals based on better timing and better planned route optimising resources.

The performance results of our algorithm show that it was always superior in all 3 evaluation environments. The first environment has been the most difficult as the success rate for 100 episodes is lower compared to the remaining 2 environments. The highest performance can be observed in environment 2, where nearly 100 % of the episodes were completed, taking in comparison the average time of each episode and the number of steps performed, the performance of the algorithm is highly accepted. When fixed obstacles such as walls are added, the robot takes more steps in planning the path to follow. This influences the time in which the robot reaches its target and therefore its success rate decreased by over 10 points compared to the performance achieved in the previous environment. This is reflected in the time reflected in the achievement of targets in the environment being three times longer than the times reflected in the environment one. This is reflected in Table 2.

Fig. 13 shows the comparison of our OTD3 algorithm with the other two algorithms in environment 1. Part (a) indicates how the performance was in terms of the path taken to reach the target, with our algorithm indicated in blue being the shortest and most direct in the two targets that the robot has to reach. Part (b), (c) and (d) indicate the linear and angular velocities that the robot performed to achieve the objective, where the OTD3 algorithm presented in blue showed better performance and shorter time.

A comparison in environment 2 shows that the OTD3 algorithm has a better performance represented by the blue line. The path taken by the robot is completely clean, while the other algorithms have a very oscillatory and long path. Thus, the linear velocity of our algorithm is slightly constant, while the others are oscillatory in time. In addition, it can be seen that the angular velocity with the best presentation is OTD3, making small turns and changes in its value while maintaining stability in the robot, so it can be seen that its movement is direct and without deviations in the path taken. All this can be seen in Fig. 14, where (a) path, (b) linear velocity X axis, (c) linear velocity Y axis, and (d) angular velocity are shown.

Moreover, the evaluation in environment 3 is shown in Fig. 15. The path taken by the robot to avoid the dynamic obstacles and the static obstacles is similar, because it was the environment used for the training of the algorithms. But it can be observed that the presentation of the OTD3 algorithm presents a notable difference in terms of the path taken and the linear velocities X and Y, where the blue line presents a constant of 0.23 m/s for the X velocity, and an average of 0.001 for the Y velocity. The rotation of the robot represented by the blue angular velocity of the OTD3 algorithm is noticeably smoother compared to the other algorithms that reflect a constant oscillation, which leads to higher energy consumption and longer time to meet the objectives. The figure is divided into 4 parts: (a) realized path, (b) linear velocity X, (c) line

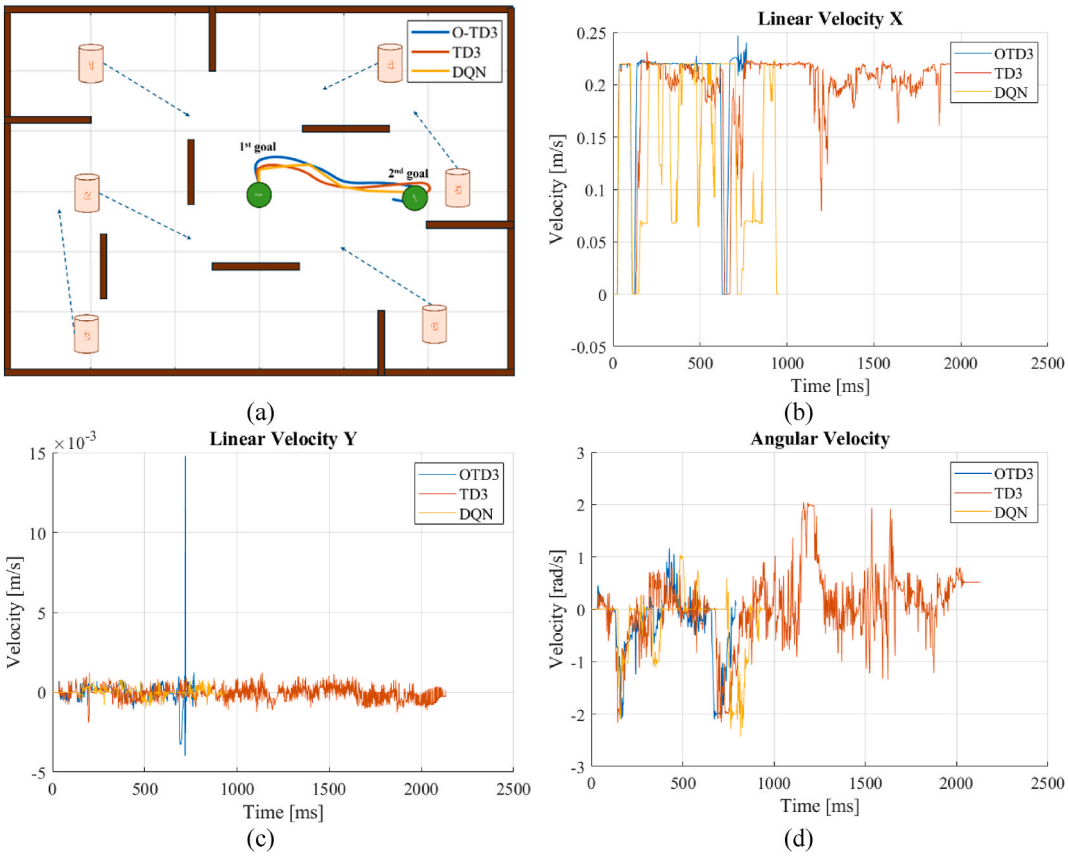

Fig. 15. Path and velocities made by the OTD3, TD3 and DQN algorithm in the performance evaluation in environment 3.

velocity Y and (d) angular velocity.

Eventually, the proposed O-TD3 algorithm presents several technical advantages over existing methods like DQN and standard TD3, particularly in dynamic and complex environments. By employ continuous control, O-TD3 enables more precise and fluid robot movements, essential for navigating around unpredictable obstacles and interacting with humans in real time. Its use of a replay buffer optimized for diverse experience sampling allows for more robust learning, while the delayed policy updates improve stability by reducing variance and overestimation bias common in reinforcement learning. Additionally, the architecture's ability to parallelize computations accelerates training and increases efficiency, making O-TD3 better suited for real-time applications. In comparison to DQN and TD3, O-TD3 demonstrates superior adaptability, higher success rates, and smoother trajectories, particularly in scenarios with dynamic obstacles. This combination of enhanced decision-making, computational efficiency, and real-time adaptability positions O-TD3 as a more effective and reliable solution for autonomous navigation in complex environments.

4.1. Hypothesis and limitations

The hypothesis of the developed method is that the Optimized Twin Delayed Deep Deterministic Policy Gradient (O-TD3) algorithm will notably improve autonomous navigation in dynamic environments, particularly in settings with human interaction, by offering better adaptation to real-time changes, higher success rates in complex scenarios, and increased computational efficiency compared to existing algorithms like DQN and standard TD3.

However, several limitations must be acknowledged. The algorithm's performance, while promising in simulated environments, may not fully generalize to real-world scenarios, which present additional unpredictability in human behavior, environmental variability, and unexpected obstacles. Furthermore, the robot's effectiveness is reliant on high-quality sensor data, which can be compromised by noise, occlusions, or limited field of view, potentially impacting obstacle detection and decision-making. The heavy dependence on the replay buffer and stored experiences means the algorithm may struggle to adapt in environments that deviate significantly from the training data, leading to suboptimal navigation. Although optimized, the computational demands of training the O-TD3 algorithm remain substantial, particularly in more complex settings, which could pose challenges for real-time application or deployment on resource-constrained hardware. Finally, while designed for human-robot interaction, the algorithm may still face difficulties when responding to highly erratic or aggressive human behavior, necessitating further refinements for enhanced reliability in dynamic environments.

5. Conclusion

The evolution of Deep Reinforcement learning algorithms has led to the development of algorithms that allow robots to interact with humans in the real dynamic environment. Therefore, in this article an autonomous control for autonomous navigation for mobile robot-human interaction using Deep Reinforcement Learning is proposed to avoid pedestrians and obstacles in dynamic environments.

The use of software that allows simulate a real world such as Gazebo and Robot Operating System (ROS2) allows.

- Send the robot's observations to the deep neural network, where the policy of Optimized Twin delay algorithm (O-TD3) developed in PyTorch can train these values.
- Simulation results show how the agent based on the observations that includes the scans of the lidar sensor and odometry and the reward, it makes an action, this action contains the value to be

transformed in a linear movement and angular movement of the robot.

Thanks to the comparison of the critic in the actor the actor decides a better output for the robot and makes a desired path avoiding obstacles, collisions and pedestrians that are represented by the dynamic obstacles in the simulation.

- The validation Between three other methods compared with the used algorithm allows to demonstrate that the optimization was successful and present fantastic results.
- The evaluation of the algorithm in untrained environments indicated that its design was correct and that it can perform well in any environment with dynamic and static objects.
- The robot took correct actions sent by the OTD3 algorithm to perform a clean path without oscillations or deviations to meet the objectives.
- The speeds were almost constant, which helps the energy efficiency of the robot. And finally, the turns were smooth, according to the obstacles presented now and looking for the best route to meet the objective in the shortest time possible, always avoiding collisions with dynamic or static objects.

The scalability of this algorithm is very versatile, i.e. based on the training it can be implemented in real environments or in different robot configurations, following the recommendations below.

- The robot needs the same configuration as omnidirectional robots because it can generate the same movements.
- The implementation in a differential robot you can train the same algorithm with the same parameters of training.

The next step is to implement the algorithm in a physical robot and evaluate the results in a real environment with human beings in crowded environments like malls and city centers.

CRediT authorship contribution statement

Husam A. Neamah: Writing – review & editing, Supervision, Methodology, Investigation, Funding acquisition, Formal analysis, Conceptualization. **Oscar Agustin Mayorga Mayorga:** Writing – original draft, Visualization, Methodology, Data curation, Conceptualization.

Declaration of competing interest

The authors have no relevant financial or non-financial interests to disclose.

Data availability

No data was used for the research described in the article.

Acknowledgement

This work was supported by the Hungarian Research Fund (OTKA K143595).

References

[1] K. Katona, H.A. Neamah, P. Korondi, Obstacle avoidance and path planning methods for autonomous navigation of mobile robot, Sensors 24 (11) (Jan. 2024) 11, https://doi.org/10.3390/s24113573.

[2] S. Jameel Al-Kamil, R. Szabolcsi, Optimizing path planning in mobile robot systems using motion capture technology, Results Eng 22 (Jun. 2024) 102043, https://doi.org/10.1016/j.rineng.2024.102043.

[3] N. Promkaew, S. Thammawiset, P. Srisan, P. Sanitchon, T. Tummawai, S. Sukpancharoen, Development of metaheuristic algorithms for efficient path planning of autonomous mobile robots in indoor environments, Results Eng 22 (Jun. 2024) 102280, https://doi.org/10.1016/j.rineng.2024.102280.

[4] W. Burgard, C. Stachniss, D. Hähnel, Mobile robot map learning from range data in dynamic environments, in: C. Laugier, R. Chatila (Eds.), Autonomous Navigation in Dynamic Environments, Springer Tracts in Advanced Robotics, vol. 35, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 3–28, https://doi.org/10.1007/978-3-540-73422-2_1, 35.

[5] Y. Yin, Z. Chen, G. Liu, J. Yin, J. Guo, Autonomous navigation of mobile robots in unknown environments using off-policy reinforcement learning with curriculum learning, Expert Syst. Appl. 247 (Aug. 2024) 123202, https://doi.org/10.1016/j.eswa.2024.123202.

[6] C.-L. Cheng, C.-C. Hsu, S. Saeedvand, J.-H. Jo, Multi-objective crowd-aware robot navigation system using deep reinforcement learning, Appl. Soft Comput. 151 (Jan. 2024) 111154, https://doi.org/10.1016/j.asoc.2023.111154.

[7] C. Yan, G. Chen, Y. Li, F. Sun, Y. Wu, Immune deep reinforcement learning-based path planning for mobile robot in unknown environment, Appl. Soft Comput. 145 (Sep. 2023) 110601, https://doi.org/10.1016/j.asoc.2023.110601.

[8] E.S. Low, P. Ong, C.Y. Low, An empirical evaluation of Q-learning in autonomous mobile robots in static and dynamic environments using simulation, Decis. Anal. J. 8 (Sep. 2023) 100314, https://doi.org/10.1016/j.dajour.2023.100314.

[9] P. Li, D. Chen, Y. Wang, L. Zhang, S. Zhao, Path planning of mobile robot based on improved TD3 algorithm in dynamic environment, Heliyon 10 (11) (Jun. 2024) e32167, https://doi.org/10.1016/j.heliyon.2024.e32167.

[10] Z. Shiller, F. Large, S. Sekhavat, C. Laugier, Motion planning in dynamic environments, in: C. Laugier, R. Chatila (Eds.), Autonomous Navigation in Dynamic Environments, Springer Tracts in Advanced Robotics, vol. 35, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 107–119, https://doi.org/10.1007/978-3-540-73422-2_5, 35.

[11] A.K. Pandey, R. Alami, A framework towards a socially aware Mobile Robot motion in Human-Centered dynamic environment, in: *2010 IEEE/RSJ International Conference On Intelligent Robots And Systems*, Taipei: IEEE, Oct. 2010, pp. 5855–5860, https://doi.org/10.1109/IROS.2010.5649688.

[12] R. Alami, K.M. Krishna, T. Siméon, Provably safe motions strategies for mobile robots in dynamic domains, in: C. Laugier, R. Chatila (Eds.), Autonomous Navigation in Dynamic Environments, Springer Tracts in Advanced Robotics, vol. 35, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 85–106, https://doi.org/10.1007/978-3-540-73422-2_4, l. 35.

[13] B. Kluge, Recursive agent modeling with probabilistic velocity obstacles for mobile robot navigation among humans, in: E. Prassler, G. Lawitzky, A. Stopp, G. Grunwald, M. Hägele, R. Dillmann, I. Iossifidis (Eds.), Advances in Human-Robot Interaction, Springer Tracts in Advanced Robotics, vol. 14, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 89–103, https://doi.org/10.1007/978-3-540-31509-4_8. . 14,.

[14] R. Philippsen, B. Jensen, R. Siegwart, Towards real-time sensor-based path planning in highly dynamic environments, in: C. Laugier, R. Chatila (Eds.), Autonomous Navigation in Dynamic Environments, Springer Tracts in Advanced Robotics, vol. 35, Springer Berlin Heidelberg, Berlin, Heidelberg, 2007, pp. 135–148, https://doi.org/10.1007/978-3-540-73422-2_7, 35.

[15] J. Guzzi, A. Giusti, L.M. Gambardella, G. Theraulaz, G.A. Di Caro, Human-friendly robot navigation in dynamic environments, in: 2013 IEEE International Conference on Robotics and Automation, IEEE, Karlsruhe, Germany, May 2013, pp. 423–430, https://doi.org/10.1109/ICRA.2013.6630610.

[16] X. Teng, Z. Shen, L. Huang, H. Li, W. Li, Multi-sensor fusion based wheeled robot research on indoor positioning method, Results Eng 22 (Jun. 2024) 102268, https://doi.org/10.1016/j.rineng.2024.102268.

[17] I. Nishitani, T. Matsumura, M. Ozawa, A. Yorozu, M. Takahashi, Human-centered X – Y – T space path planning for mobile robot in dynamic environments, Robot. Auton. Syst. 66 (Apr. 2015) 18–26, https://doi.org/10.1016/j.robot.2014.12.018.

[18] R.S. Pol, M. Murugan, A review on indoor human aware autonomous mobile robot navigation through a dynamic environment survey of different path planning algorithm and methods, in: 2015 International Conference on Industrial Instrumentation and Control (ICIC), IEEE, Pune, India, May 2015, pp. 1339–1344, https://doi.org/10.1109/IIC.2015.7150956.

[19] M.A. Kareem Jaradat, M. Al-Rousan, L. Quadan, Reinforcement based mobile robot navigation in dynamic environment, Robot. Comput.-Integr. Manuf. 27 (1) (Feb. 2011) 135–149, https://doi.org/10.1016/j.rcim.2010.06.019.

[20] M.R. Azizi, A. Rastegarpanah, R. Stolkin, Motion planning and control of an omnidirectional mobile robot in dynamic environments, Robotics 10 (1) (Mar. 2021) 48, https://doi.org/10.3390/robotics10010048.

[21] S. Choi, E. Kim, K. Lee, S. Oh, Real-time nonparametric reactive navigation of mobile robots in dynamic environments, Robot. Auton. Syst. 91 (May 2017) 11–24, https://doi.org/10.1016/j.robot.2016.12.003.

[22] G.-C. Luh, W.-W. Liu, An immunological approach to mobile robot reactive navigation, Appl. Soft Comput. 8 (1) (Jan. 2008) 30–45, https://doi.org/10.1016/j.asoc.2006.10.009.

[23] T. Xuan Tung, T. Dung Ngo, Socially aware robot navigation using deep reinforcement learning, in: *2018 IEEE Canadian Conference On Electrical & Computer Engineering (CCECE)*, Quebec City, QC, IEEE, May 2018, pp. 1–5, https://doi.org/10.1109/CCECE.2018.8447854.

[24] S.S. Samsani, M.S. Muhammad, Socially compliant robot navigation in crowded environment by human behavior resemblance using deep reinforcement learning, IEEE Robot. Autom. Lett. 6 (3) (Jul. 2021) 5223–5230, https://doi.org/10.1109/LRA.2021.3071954.

[25] J. Lin, X. Yang, P. Zheng, H. Cheng, End-to-end decentralized multi-robot navigation in unknown complex environments via deep reinforcement learning, in: 2019 IEEE International Conference on Mechatronics and Automation (ICMA), IEEE, Tianjin, China, Aug. 2019, pp. 2493–2500, https://doi.org/10.1109/ICMA.2019.8816208.

[26] T. Fan, P. Long, W. Liu, J. Pan, Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios, Int. J. Robot Res. 39 (7) (Jun. 2020) 856–892, https://doi.org/10.1177/0278364920916531.

[27] S. Pouyanfar, et al., A survey on deep learning: algorithms, techniques, and applications, ACM Comput. Surv. 51 (5) (Sep. 2019) 1–36, https://doi.org/10.1145/3234150.

[28] A. Shrestha, A. Mahmood, Review of deep learning algorithms and architectures, IEEE Access 7 (2019) 53040–53065, https://doi.org/10.1109/ACCESS.2019.2912200.

[29] N.H. Singh, A. Laishram, K. Thongam, Optimal path planning for mobile robot navigation using FA-TPM in cluttered dynamic environments, Procedia Comput. Sci. 218 (2023) 612–620, https://doi.org/10.1016/j.procs.2023.01.043.

[30] H. Li, A.V. Savkin, An algorithm for safe navigation of mobile robots by a sensor network in dynamic cluttered industrial environments, Robot. Comput.-Integr. Manuf. 54 (Dec. 2018) 65–82, https://doi.org/10.1016/j.rcim.2018.05.008.

[31] Y.-C. Lee, LSMCL: long-term Static Mapping and Cloning Localization for autonomous robot navigation using 3D LiDAR in dynamic environments, Expert Syst. Appl. 241 (May 2024) 122688, https://doi.org/10.1016/j.eswa.2023.122688.

[32] Q. Zhu, J. Hu, W. Cai, L. Henschen, A new robot navigation algorithm for dynamic unknown environments based on dynamic path re-computation and an improved scout ant algorithm, Appl. Soft Comput. 11 (8) (Dec. 2011) 4667–4676, https://doi.org/10.1016/j.asoc.2011.07.016.

[33] D. Dudarenko, J. Rubtsova, A. Kovalev, O. Sivchenko, Reinforcement learning approach for navigation of ground robotic platform in statically and dynamically generated environments, IFAC-Pap. 52 (25) (2019) 445–450, https://doi.org/10.1016/j.ifacol.2019.12.579.

[34] H.A. Neamah, R. Butdee, Optimization modeling parameters for industrial AMR slippage using ANFIS system in dynamic environment, in: P. Janmanee, S. Chujuarjeen, S. Butdee, P. Srikhumsuk, A.D.L. Batako, A. Burduk, M.A. Xavior (Eds.), *Advanced In Creative Technology- Added Value Innovations In Engineering, Materials And Manufacturing*, Vol. 979, Lecture Notes in Networks and Systems, vol. 979, Springer Nature Switzerland, Cham, 2024, pp. 214–223, https://doi.org/10.1007/978-3-031-59164-8_18.

[35] H. Almusawi, M. Al-Jabali, A. Khaled, P. Korondi, G. Husi, Self-Driving robotic car utilizing image processing and machine learning, IOP Conf. Ser. Mater. Sci. Eng. 1256 (Oct. 2022) 012024, https://doi.org/10.1088/1757-899X/1256/1/012024.

[36] H.A. Neamah, E. Donát, P. Korondi, Optimizing autonomous navigation in unknown environments: A novel trap avoiding vector field histogram algorithm VFH+T, Results in Engineering 23 (2024) 102625. https://doi.org/10.1016/j.rineng.2024.102625.

[37] M.U. Shafiq, et al., Real-time navigation of mecanum wheel-based mobile robot in a dynamic environment, Heliyon 10 (5) (Mar. 2024) e26829, https://doi.org/10.1016/j.heliyon.2024.e26829.

[38] H.A. Neamah, D.A. Ardi, P. Korondi, Enhancing autonomous robot perception for precision positioning and localization, in: 2024 IEEE/SICE International Symposium on System Integration (SII), Jan. 2024, pp. 1058–1063, https://doi.org/10.1109/SII58957.2024.10417351.

[39] F. Kamil, T.S. Hong, W. Khaksar, M.Y. Moghrabiah, N. Zulkifli, S.A. Ahmad, New robot navigation algorithm for arbitrary unknown dynamic environments based on future prediction and priority behavior, Expert Syst. Appl. 86 (Nov. 2017) 274–291, https://doi.org/10.1016/j.eswa.2017.05.059.

[40] W. Yaonan, et al., Autonomous mobile robot navigation system designed in dynamic environment based on transferable belief model, Measurement 44 (8) (Oct. 2011) 1389–1405, https://doi.org/10.1016/j.measurement.2011.05.010.

[41] B. Wang, S. Li, J. Guo, Q. Chen, Car-like mobile robot path planning in rough terrain using multi-objective particle swarm optimization algorithm, Neurocomputing 282 (Mar. 2018) 42–51, https://doi.org/10.1016/j.neucom.2017.12.015.

[42] M.R. Panda, P. Das, S. Pradhan, Hybridization of IWO and IPSO for mobile robots navigation in a dynamic environment, J. King Saud Univ. - Comput. Inf. Sci. 32 (9) (Nov. 2020) 1020–1033, https://doi.org/10.1016/j.jksuci.2017.12.009.

[43] G.G. Yen, T.W. Hickey, Reinforcement learning algorithms for robotic navigation in dynamic environments, ISA Trans. 43 (2) (Apr. 2004) 217–230, https://doi.org/10.1016/S0019-0578(07)60032-9.

[44] X. Xing, H. Ding, Z. Liang, B. Li, Z. Yang, Robot path planner based on deep reinforcement learning and the seeker optimization algorithm, Mechatronics 88 (Dec. 2022) 102918, https://doi.org/10.1016/j.mechatronics.2022.102918.

[45] H. Jiang, et al., iTD3-CLN: learn to navigate in dynamic scene through Deep Reinforcement Learning, Neurocomputing 503 (Sep. 2022) 118–128, https://doi.org/10.1016/j.neucom.2022.06.102.