Engineering Science and Technology, an International Journal 64 (2025) 101993

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Full length article

Human and environmental feature-driven neural network for path-constrained robot navigation using deep reinforcement learning☆

Nabih Pico [a,b], Estrella Montero [c], Alisher Amirbek [a], Eugene Auh [a], Jeongmin Jeon [a], Manuel S. Alvarez-Alvarado [b], Babar Jamil [d], Redhwan Algabri [e], Hyungpil Moon [a,*]

[a] Department of Mechanical Engineering, Sungkyunkwan University, Suwon, Republic of Korea
[b] Facultad de Ingeniería en Electricidad y Computación, Escuela Superior Politécnica del Litoral (ESPOL), Guayaquil, Ecuador
[c] Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon, Republic of Korea
[d] School of Physics, Engineering, and Technology, University of York, York, United Kingdom
[e] Research Institute of Engineering and Technology, Hanyang University, Ansan 15588, Republic of Korea

ARTICLE INFO

Keywords:
Autonomous robot navigation
Reinforcement learning
Path constraint
Motion and path planning

ABSTRACT

This paper introduces a neural network model designed for autonomous navigation in complex environments. It combines DRL methodologies to capture critical environmental features in the neural network. These features encompass data about the robot, humans, static obstacles, and path constraints. The representation, combined with weighted features from humans and environmental limitations, is processed through three multi-layer perceptrons (MLP) to calculate the value function and optimal policy, thereby enhancing navigation tasks. A novel reward function is proposed to accommodate path constraints and steer the robot's navigation policies during neural network training. Additionally, common metrics like success rate, collision avoidance, time to reach the goal, and new comprehensive log information are included to provide an overview of the robot's performance. The model's efficacy is demonstrated through navigation in simulation scenarios involving curved and cross pathways, with the agents' random position and velocity occasionally exceeding the maximum robot speed, as well as real experiments in limited spaces. The paper provides a GitHub repository that includes comparative performance videos with state-of-the-art models in path-constrained scenarios, along with strategies for reward functions. Link: https://github.com/nabihandres/Wallproximity_DRL.

1. Introduction

Autonomous robot navigation technology has advanced to meet a variety of practical applications, demanding intelligent systems capable of decision-making in dynamic and uncertain surroundings without collisions. These environments include crowded spaces like shopping malls, airports, and branding events. Warehouses and factories have seen the widespread adoption of mobile robots for tasks such as patrolling and material handling, to improve workflow efficiency. Service robots equipped with artificial intelligence capabilities operate in restaurants, assisting in serving or picking up food. In hotels, tasks are complicated by limited spaces in hallways, requiring them to manage distances between environmental objects and humans to complete their tasks. Deep Reinforcement Learning (DRL) has emerged as a promising solution, leveraging neural networks to learn intricate navigation policies directly from experiences of trial and error [1–3]. Previous work has primarily focused on solving crowd navigation environments with human behaviors characterized by similar velocities in flexible spaces. However, in real-world scenarios, static and dynamic agents move at varying velocities, posing challenges for DRL models to adapt due to limitations in training. Common challenges encountered with DRL methods include the necessity for a substantial number of interactions with the environment to acquire proficient policies. Moreover, certain algorithms trained within the same scenarios may cause neural networks to memorize specific challenges, thereby complicating decision-making in diverse environments. Poorly designed reward functions can lead to undesired behaviors, hindering task achievement. For example, when navigating through narrow spaces, as illustrated

☆ This work was supported by the Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government(MSIT) (No. 2022-0 01025, Development of core technology for mobile manipulator for 5G edge-based transportation and manipulation).

* Correspondence to: Seobu-Ro, Jangan-Gu, Suwon-Si, Gyeonggi-Do, (16419) 2066, Republic of Korea
E-mail addresses: npico@g.skku.edu (N. Pico), emontero@g.skku.edu (E. Montero), 011129a@g.skku.edu (A. Amirbek), egauh@g.skku.edu (E. Auh), nicky707@g.skku.edu (J. Jeon), mansalva@espol.edu.ec (M.S. Alvarez-Alvarado), babar.jamil@york.ac.uk (B. Jamil), redhwan@hanyang.ac.kr (R. Algabri), hyungpil@skku.edu (H. Moon).

https://doi.org/10.1016/j.jestch.2025.101993
Received 22 October 2024; Received in revised form 9 January 2025; Accepted 31 January 2025
Available online 25 February 2025

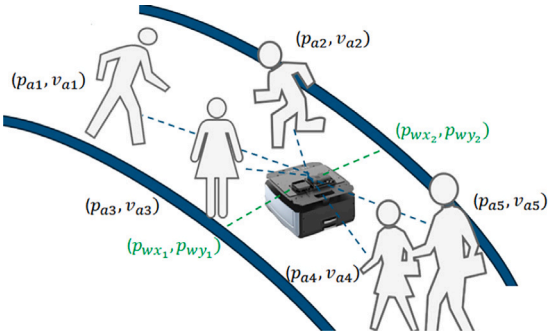


Fig. 1. Sensing the environment, including the proximity of the wall in narrow spaces.

in Fig. 1, previous approaches may fail to make intelligent decisions. This can result in freezing, getting stuck, or colliding during the navigation task, primarily due to the absence of path constraints in the training environment. This paper aims to integrate DRL techniques with insights from human–robot interaction, obstacle avoidance, and path constraint information, which have not been extensively addressed in prior models. The neural network is trained for proper decision-making in navigation tasks with path limitations. LiDAR scan data is processed to extract the positions and velocities of humans, static obstacles, and their proximity to walls. These data are combined with odometry information for robot location, providing high-level insight into the state of the environment. Therefore, the main contributions of this paper are:

- A neural network that captures essential environmental features, including information about robots, humans, static obstacles, and walls for navigation tasks through a series of multi-layer perceptrons (MLP), creating a higher-level representation before calculating the value function, and facilitating intelligent decision-making.
- A new reward function is designed to accommodate path limitations, guide the robot through limited spaces, and provide comprehensive reward log information.
- A comparative analysis of robot performance using state-of-the-art neural networks and our proposed model with custom reward functions, validating it in both simulation and real-world scenarios.

The rest of the paper is organized as follows: Section 2 presents the related work on autonomous navigation tasks for mobile robots. Section 3 discusses the problem formulation, agents, state information, reward functions, and optimal policy calculation. Section 4 introduces the proposed neural network, trained parameters, the navigation strategy for path-constrained spaces, and log information collected during navigation.

Section 5 discusses simulation and experimental results. Finally, in Section 6, presents the conclusions.

2. Related work

Navigating through crowds presents a significant challenge for autonomous systems. Traditional navigation methods have relied on various techniques, including obstacle–robot interaction, path planning, SLAM algorithms, and terrain recognition to adapt to diverse environments. Model predictive control has emerged as an alternative approach to improving robot performance [4,5]. Algorithms like Social Force [6], Reciprocal Velocity Obstacles [7], and ORCA [2] replicate human behaviors during training phases in reinforcement learning approaches before real-world experiments. In [8], a warning zone around humans was proposed based on their orientation and speed to indicate collision risks with the robot, along with a strategy to assess the remaining distance to reach the goal. Authors in [9] investigate the trade-off between safety and efficiency in robot navigation. In [10], the authors provide a comprehensive guide on the advantages and disadvantages of multiple reward functions, addressing critical issues such as robot freezing [11] and collision avoidance. Multi-objective reinforcement learning (MODRL) develops control strategies and joint cost functions for urban environments [12,13]. Fixed training scenarios, such as uniform agent velocities, often hinder the performance of DRL approaches in real-world complex environments where robots encounter both static and dynamic obstacles. To address this, [14] introduces modifications to neural networks to enhance generalization across diverse environments. Similarly, the SOADRL neural network [15] separates pedestrians and static obstacles to explicitly provide information related to the type of obstacles, for dynamic obstacles used information from [16] and two different representations for static obstacles: angular maps and occupancy grids [17]. NavRep provides a simulation framework for testing and training to compare end-to-end methods with unsupervised methods, where sensor data is directly used as input without detection or tracking algorithms [18]. The HGAT-DRL algorithm introduces the human–robot environment in a heterogeneous graph with four types of nodes as follows: human, robot, static obstacles, and large static obstacles. The studies include the robots' acceleration constraints [19]. Collision avoidance with deep reinforcement learning (CADRL) a pioneering algorithm in learning-based crowd navigation [15,16], utilizes deep neural networks (DNNs) to extract the implicit reciprocal motion features of human states, including positions and velocities. These extracted features are then fed into a DRL-based policy to optimize motion planning. A subsequent algorithm, long short-term memory reinforcement learning (LSTM-RL) [20,21], leverages an LSTM network to model interactions between the robot and all nearby humans in dynamic crowd environments. It organizes input data by ranking agents based on their proximity to the robot, ensuring that the closest agent exerts the most significant influence on the robot's decision-making process. The SARL approach improves the modeling of social interactions employing a self-attention mechanism [22]. This technique effectively captures pairwise interactions between the robot and surrounding humans, allowing the robot to prioritize and respond to critical interactions in complex scenarios. The ST^2 model improves robot navigation by encoding both spatial and temporal states, crucial for understanding interactions with pedestrians. The authors highlight that traditional methods often neglect temporal interactions, resulting in suboptimal decisions. The model uses a Transformer-based architecture with two key components: a global spatial state encoder and a temporal state encoder [23]. Recently, [24] introduced a radar-based navigation system for wheeled robots that combines dynamic obstacle detection with bi-directional gated recurrent unit (BiGRU)-enabled DRL framework for robust navigation in dynamic environments. The system utilizes filtering and tracking algorithms to enhance obstacle detection. The BiGRU-enabled DRL model processes sequential environmental data to generate robust navigation policy. Additionally, frameworks such as MultiROS package enable training across multiple environments simultaneously [25]. Despite advancements, challenges persist, especially regarding limited experiences or missing information in the neural network, including path constraints. Drawing insights from prior works and existing DRL models [14,16,20,22], we introduce a neural network model in the subsequent sections that considers human behaviors and incorporates wall constraints to enhance robot navigation capabilities.

3. Problem formulation

The robot navigates among multiple agents, it makes n decisions until reaching the goal. The surrounding agents can be static or dynamic in scenarios with walls and curves. For simplicity, the neural network is trained in a 2D environment; human agents and the robot are represented by circles with radii r_{ai} and r_o, respectively.

3.1. Agents and states

The human agents have random starting positions p_{ai} and goals g_{ai}. Their behavior in the scenarios is based on three motion patterns: static, dynamic with random velocity until reaching their goals, and dynamic without stopping. In the dynamic without stopping pattern, humans move continuously between their starting point and goal without pausing. Once they reach their goal, they reverse the direction and return to their initial starting point, creating a cyclic motion. This persistent motion ensures the robot encounters ongoing dynamic challenges during navigation. The speed settings include speeds faster than the maximum speed of the robot $v_o^{\max}$ to teach the robot to stop and let objects pass [10]. Agents make decisions a_i at time t based on their observation O_{ai} of the environment, which includes the position p_{ai}, velocity v_{ai}, and radii r_{ai} of other agents i using the Python-RVO2 library. The following parameters are included in the state of the agents $\tau_a(t)$: O_{ai} observation of the environment, d_{ai} are the distances between the agents. (NeighborDist) n_d represents the maximum distance at which an agent considers other agents. (MaxNeighbors) n_m is the maximum number of neighbors that the agent considers in the state. (TimeHorizon) t_H is the time for the agent to compute its velocity concerning other agents. (TimeHorizonObst) t_{Ho} is the time the agent computes its velocity concerning obstacles, in our case, the walls that are included in the simulation environment. The robot state $\tau_o(t)$ includes its current position p_o, velocity v_o, goal g_o, robot radius r_o, velocity preferences v_o^{pref}, and θ_o the angle of a vector represented by the velocity components v_{ox} (velocity in the x-direction) and v_{oy} (velocity in the y-direction). Eq. (1) shows the equation of the observation of the agents O_{ai}, human agent's states $\tau_a(t)$, robot's states $\tau_o(t)$, and $\tau(t)$, the information of all the agents.

$$\begin{aligned} O_{ai}(t) &= [p_{ai}, v_{ai}, r_{ai}], \\ \tau_a(t) &= [O_{ai}, d_{ai}, n_d, n_m, t_H, t_{Ho}], \\ \tau_o(t) &= [p_o, v_o, g_o, r_o, v_o^{\text{pref}}, \theta_o], \\ \tau(t) &= [\tau_o(t), \tau_{a1}(t), \tau_{a2}(t), \ldots, \tau_{an}(t)]. \end{aligned} \tag{1}$$

3.2. Rewards

In reinforcement learning, reward functions guide the agent's behavior for various purposes. During navigation tasks, the reward function may provide incentives for reaching the objective, avoiding obstacles, staying on a safe path, or exhibiting any other desired behavior. Conversely, it may impose penalties for colliding with obstacles, deviating from the optimal path, colliding with walls, or engaging in any other undesirable behavior. Three formulations of reward functions are utilized to compare and validate the DRL approach: Eq. (2) depicts the "if" and "else" clauses used in several previous works [16,26], where $R_g = 10$ represents a positive reward for reaching the goal. $R_c = -0.5$ indicates a negative reward given to the robot when it collides. $R_d = 0.5(d_i - d_{ud})$ denotes a reward that the robot receives based on its proximity to others, where d_i is the distance between the robot and the agent, and d_{ud} is the maximum comfortable distance allowed. Additionally, the robot obtains $R_{hg} = 0.01(d_g(t) - d_g(t-1))$, where it receives positive or negative incentives based on its position relative to the goal d_g. The reward function R_1 is a combination of the individual rewards described previously, and is defined as follows:

$$R_1 = \begin{cases} R_g = 10, & \text{if Reach the goal} \\ R_c = -0.5, & \text{elif Collision} \\ R_d = 0.5(d_i - d_{ud}), & \text{elif Danger} \\ R_{hg} = 0.01(d_g(t) - d_g(t-1)), & \text{otherwise} \end{cases} \tag{2}$$

Eq. (3) summarizes the reward function at each time step, incorporating a novel reward $R_s = 0.02$ when the robot stops, allowing other agents to cross without collision [10].

$$R_2 = R_g + R_c + R_d + R_{hg} + R_s \tag{3}$$

Eq. (4) defines the dynamic warning zone reward. These zones are circular regions created around the obstacles, formulated according to human size and velocity [8]. This enables the robot to anticipate and predict human motion while ensuring safe navigation. The robot incurs an exponential penalty when entering or exceeding these zones, calculated as $0.2\left(e^{(d_{ai}-r_{wz}-r_{ai})} - 0.3\right)$, where d_{ai} denotes the distance between the agents, r_{wz} is the radius of the circular region, and r_{ai} is the agents radii.

$$R_3 = \begin{cases} R_g = 10, & \text{if Reach the goal} \\ R_c = -0.1, & \text{elif Collision} \\ R_d = 0.25(d_i - d_{ud}), & \text{elif Danger} \\ R_{wz} = 0.2\left(e^{(d_{ai}-r_{wz}-r_{ai})} - 0.3\right), & \text{elif Warning zone} \\ R_{hg} = 0.01\left(d_g(t) - d_g(t-1)\right), & \text{otherwise.} \end{cases} \tag{4}$$

Eq. (5) incorporates the R_{pf} reward, which encourages the robot to prevent freezing, a common issue in autonomous navigation. The robot gains a reward $R_{pf} = 0.01$ if the difference between its current position and previous position exceeds a threshold of 0.03; otherwise, it is penalized with $R_{pf} = -0.01$, indicating that the robot has stopped. R_s represents the stop reward used in Eq. (3). To address the challenge of navigating near walls in constrained environments, we introduce a novel reward R_{wall}, which enhances the robot's ability to maintain safe distances from walls and avoid collisions. This aspect has not been previously addressed in existing approaches and represents a key contribution to our work.

$$R_4 = R_g + R_c + R_d + R_{hg} + R_s + R_{pf} + R_{wall} \tag{5}$$

R_{wall} consists of a summation of three reward functions: R_{wc}, a penalty for collision with walls, and $R_{w_{min}}$ and $R_{w_{max}}$ are rewards used to maintain a safe distance from the walls and ensure the robot's motion stays within the map boundaries, as detailed in the next section.

3.3. Optimal policy

The strategy and actions that the robot takes during navigation tasks result from the optimal policy $\pi^*(\tau)$, which predicts its cumulative reward over time in the environment through the states $\tau(t)$. This policy is learned by training a neural network to approximate the optimal value function $V^*(\tau)$, Eq. (6), representing the expected cumulative reward for each state $\tau(t)$. In this equation, a represents the robot's actions, γ is the discount factor determining the present value of future rewards (with values close to zero emphasizing immediate rewards and close to one emphasizing future rewards), Δ_t is the interval between two actions, and $R(\tau, a)$ is the reward received when the optimal action a^* is taken from state $\tau(t)$.

$$V^*(\tau_t) = \sum_{i=o}^{K} \gamma^{i\Delta_t v_{pref}} R(\tau_t, a_t{}^*). \tag{6}$$

The optimal policy, as expressed in Eq. (7), maximizes the cumulative reward and guides the robot to take the optimal action to reach the goal based on the current state. In other words, it represents the sum of the immediate reward and the discounted future reward.

$$\pi^*(\tau) = \arg\max_{a_t \in A} R(\tau_t, a_t) + \gamma^{\Delta_t v_{pref}} V^*(\tau_{t+\Delta_t}). \tag{7}$$

4. Neural network

During the navigation task, environmental observation plays a crucial role in path planning and determining the robot's actions at each step. In previous research, the primary challenge lay in the interaction between humans and robots, and the training simulations typically occurred in open spaces across most scenarios [16,20,22]. Consequently, when the robot operates in environments with limited space, it may struggle to adopt appropriate behavior to reach its goal, potentially leading to freezing or rotating in place. This drove us to develop a neural model that incorporates comprehensive information about neighboring humans, as suggested by [20,27,28], along with the positions of walls in our work.

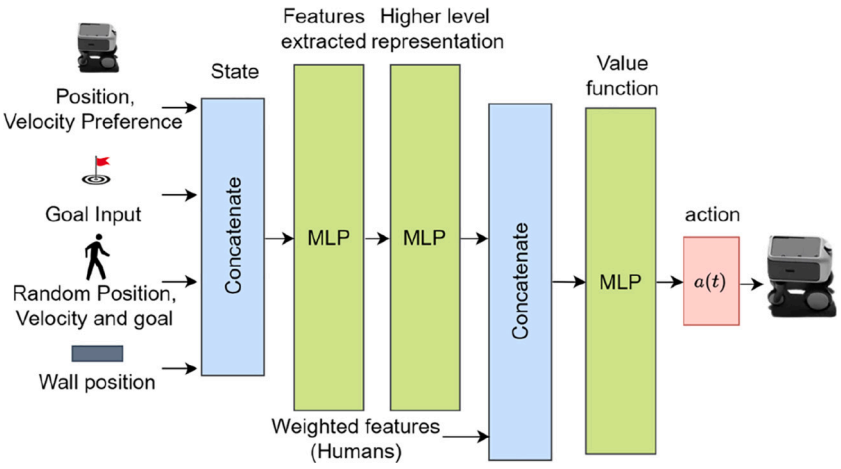


Fig. 2. Network architecture for autonomous navigation: Integration of humans and robot states with wall constraints.

4.1. Overview of neural model

Fig. 2 illustrates a flow diagram of the neural network model used in this work, where inputs from robots, humans, and walls are integrated to extract essential features, enabling intelligent responses in autonomous navigation tasks. Drawing on previous research [16, 20], we employed a multi-layer perceptron (MLP), widely used for various machine learning tasks. The first MLP captures crucial information about the environment and wall constraints. Subsequently, the extracted features undergo processing through the second MLP to create a higher-level representation, capturing complex relationships and context, that serves as an input for the third MLP to calculate the value function, prioritizing actions based on expected outcomes, and incorporating weighted features from humans. The final output corresponds to the action to be taken by the robot in terms of linear and angular velocity. In the training environment, we define a list of coordinates that represent the path constraints marked in red. This list also guides the random positioning of humans and goals inside the map. Additionally, it guides the agents' trajectory in the ORCA algorithm, preventing them from crossing blocked areas. In the real world, this information is obtained from the map through a mapping process by LiDAR in PNG and YAML files. The PNG file is a grayscale image format that contains the dimensions and pixel data from the map. The YAML file includes the path of the PNG file, the resolution of the occupancy grid map in meters per pixel, the origin of the occupancy grid map in the coordinate system, and the threshold value for considering a grid occupied or free for navigation. Both files serve as inputs for an algorithm that takes the environmental constraint coordinates in pixel units and transforms them into meters. The neural model receives them and calculates the value function to obtain the optimal robot actions.

4.2. Neural network training process

The aim is to train the robot to maximize cumulative rewards by learning from experience and iterative optimization. As detailed in Algorithm 1, the model initializes the parameters of training and simulation for the DRL model (steps 1–2). In step 3, the experience memory E is initialized through imitation learning, providing the agent with a set of demonstrations to approximate the optimal policy from the beginning. Following this, the value function $V^*(\tau_t)$, which estimates the expected reward for a given state (τ_t), is initialized based on the experiences stored in E (step 4). From steps 6 to 17, the model undergoes a series of iterations, where the agent interacts with the environment until it reaches the goal, collides, or exceeds the maximum allowed time for the scenario. At each step, the agent selects an action a_t following an ϵ-greedy policy, balancing exploration and exploitation

Algorithm 1 Neural Networking Training

1: Set DRL parameters
2: Set simulation parameters
3: Initiate experience memory $E \leftarrow$ Imitation
4: Initiate $V^*(\tau_t) \leftarrow E$
5: **procedure** COMPUTE($\hat{V}^* \leftarrow V^*$)
6: **for** episode=1,2,..,N **do**
7: Initiate joint state $\tau(t=0)$
8: **repeat**
9: Random a with ϵ probability
10: Calculate Reward:
11: $R_4 = R_g + R_c + R_d + R_{hg} + ... + R_{wall}$;
12: $a_t \leftarrow \arg\max_{a_t \in A} R(\tau_t, a_t) + \gamma^{\Delta_t v_{pref}} V^*(\tau_{t+\Delta_t})$
13: Value $\leftarrow R(\tau_t, a_t) + \gamma^{\Delta_t v_{pref}} V^*(\tau_{t+\Delta_t})$
14: Store $E \leftarrow (\tau_t, a_t, \tau_{t+\Delta_t})$
15: Mini-batches are sampled from E
16: Optimize $V^*(\tau_t)$ by gradient descent
17: **until** goal, collision or $t \geq t_{max}$
18: Update $\hat{V}^*(\tau_t) \leftarrow V^*(\tau_t)$
19: **end for**
20: **end procedure**

(step 9–12). ϵ represents the probability of taking a random action (explore), while $1-\epsilon$ indicates the probability of selecting the best-known action based on the current policy (exploit). Initially, ϵ is set to 0.5, meaning that 50% of the actions are chosen randomly to explore new rewards. Over the first 2000 episodes, ϵ gradually decays to 0.1, reducing randomness and increasing reliance on the learned policy, i.e., in the subsequent episodes, 90% of decisions are derived from stored experiences, while 10% are exploratory actions. The agent receives R_4 and determines the optimal action by $1-\epsilon$ probability (steps 10–12). From steps 13 to 14, the agents store their experience (state, action, next state) in the memory buffer E. Subsequently, in steps 15 to 16, mini-batches of experiences are sampled from E to train $V^*(\tau_t)$. The function is optimized using gradient descent, which minimizes the difference between the predicted and actual values. The gradient descent method is selected over other methods, such as Gauss–Newton, due to its simplicity and computational lightweight nature, which aligns with the real-time processing requirements. At the end of each episode, $\hat{V}^*(\tau_t)$ is updated for the current state (steps 18–20). The training process continues across N episodes (steps 5–20), and the neural network finally converges to an optimal policy.

Table 1
Neuronal network and simulation parameters.

DRL parameters		Simulation parameters	
Parameter	Value	Parameter	Value
Discount factor γ	0.9	Agents (a_i)	8-15
Time step	0.25 s	Static (a_i)	0–8
Test Size	3000	Dynamic (a_i)	0–8
Min Dist Human–robot	0.3 m	Robot Radius	0.2 m
collision penalty	−0.5 m	Agent Radius	0.1-0.5 m
Evaluation Interval	1000	Object Velocity	0.3-1.5 m/s
Il episodes	3000	Pref robot speed	0.8 m/s
RL Learning Rate	0.001	Kinematics	Unicycle
Train Batches	100	Robot Position	(4,−4)
Train Episodes (N)	2000	Goal	(−4,4),(0,4)
Scenarios	4	Time Limit	45 s

4.3. Neural network parameters

The neural network parameters and simulation environment outlined in Table 1 are crucial for enabling effective learning of navigation tasks. The training and evaluation of our proposed neural network model were conducted on a system running Ubuntu 18.04.6. The hardware setup included an Intel i5-8400 CPU and an NVIDIA GeForce GTX 1050 Ti GPU. The entire training process spanned 48 hours for the curve path scenario and 62 hours for the cross path scenario. The model was trained for 8000 episodes, each with a maximum duration of 45 s per scenario and a time step of 0.25 s. We evaluated the neural network's performance every 1,000 episodes and collected log data for the entire duration of the episodes. Each episode simulated four scenarios: no obstacles, static obstacles, dynamic obstacles, or a mix of both, involving static and dynamic agents. The training set comprised scenarios with varying numbers of humans, where the robot successfully avoided between 8 and 15 humans simultaneously, demonstrating the potential scalability of our approach.

The agents exhibited random positions, velocities, and radii, with the robot's maximum velocity set at 0.8 m/s and the human agents' velocities ranging from 0.3 to 1.5 m/s. The inclusion of human agents with random velocities, some exceeding the robot's maximum velocity, was motivated by the need to simulate realistic and crowded environments, rather than assuming that all objects move at the same speed. This design introduces a challenge, as the robot's constrained speed makes it more difficult to navigate among faster-moving agents. Similar challenges were observed in previous research, where a TurtleBot3 with a maximum velocity of 0.22 m/s was used [8,14]. In crowded scenarios, robots typically operate at moderate speeds for safety, while other agents maintain higher velocities. This setup can be visualized in our GitHub repository, where the robot demonstrates adaptive behaviors, such as reducing its velocity to avoid collisions and allowing other agents to pass, ensuring safe and efficient navigation. This variability ensures that our model generalizes effectively across diverse and challenging navigation tasks.

4.4. Wall proximity functions

In previous research, training was conducted in open spaces, causing the robot to learn collision avoidance without considering walls. Although the robot acquires information about the environment during mapping, it often falls short in making optimal decisions to avoid dynamic and static objects by maintaining a safe distance from walls. Therefore, this section elaborates on the reward function and subfunctions designed to enhance the robot's navigation, specifically addressing proximity to walls, denoted as R_{wall}. The robot incurs a penalty when it is close to the wall and receives a reward when it maintains a safe distance. The robot's and the wall's position is continually assessed during the navigation task. The Euclidean distance between two points is calculated using Eq. (8), determining the closest and furthest points

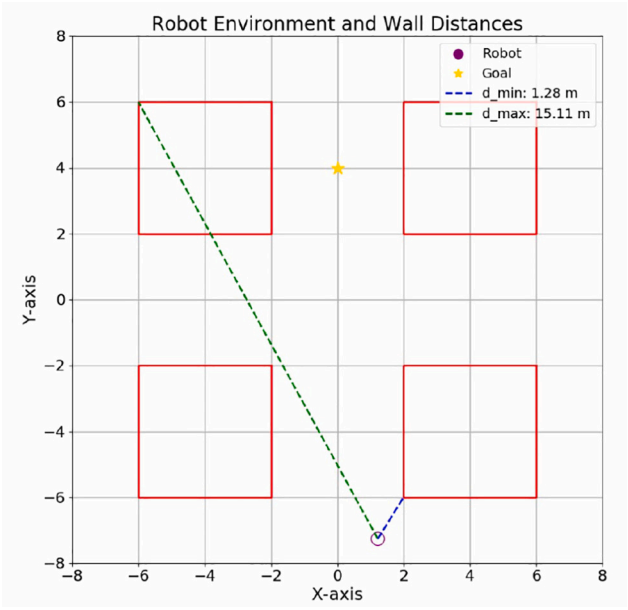


Fig. 3. Minimum and maximum distances between the robot and the walls for calculating R_{wall}.

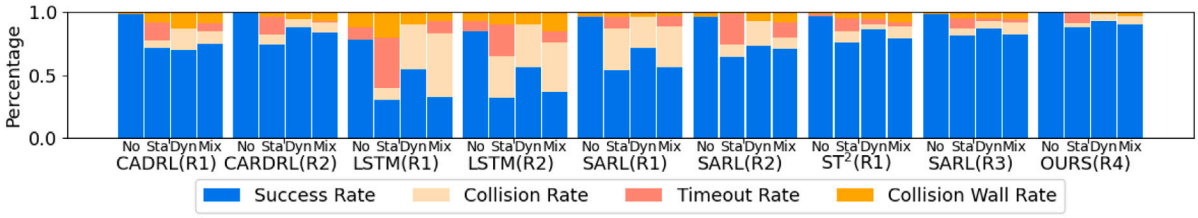
of the robot concerning each wall.

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{8}$$

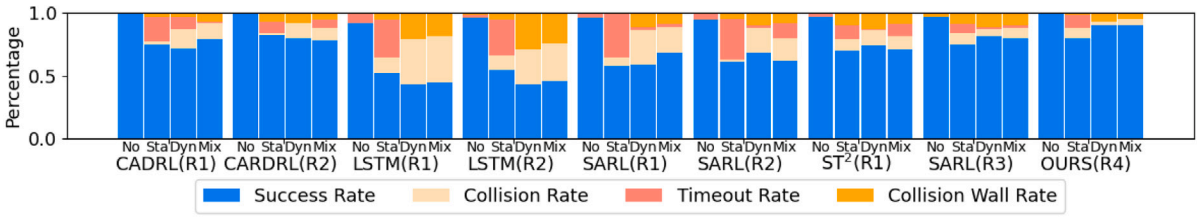
Algorithm 2 outlines the procedure for calculating R_{wall}. The function ComputeMinDistance takes the coordinates of walls as arguments (steps 1-10). $d_{\min}$ represents the minimum distance from the robot to each wall, and $p_{\min}$ denotes the coordinates of the closest point on the wall to the robot (step 2). The algorithm iterates over each wall, updating $d_{\min}$ whenever a distance smaller than the current minimum is found (steps 3-9). A similar procedure, ComputeMaxDistance, is used to find the maximum distance between the robot and the walls, logging $d_{\max}$ and $p_{\max}$ (step 11). The distances $d_{\min}$ and $d_{\max}$ are illustrated in Figure Fig. 3.

Algorithm 2 Wall Proximity Functions

1: **procedure** ComputeMinDistance($pos, walls$)
2: $d_{\min}, p_{\min} \leftarrow \infty$, None
3: **for** w **in** $walls$ **do**
4: $d \leftarrow$ DistToWall(pos, w)
5: **if** $d < d_{\min}$ **then**
6: $d_{\min} \leftarrow d$
7: **end if**
8: **end for**
9: **return** $d_{\min}$
10: **end procedure**
11: ComputeMaxDistance adjusts the comparison accordingly by replacing $d_{\min}$ with $d_{\max}$.
12: **procedure** Reward($pos, walls$)
13: $d_{\min} \leftarrow$ ComputeMinDistance($pos, walls$) $- r_o$
14: $d_{\max} \leftarrow$ ComputeMaxDistance($pos, walls$)
15: $Rw_{min} \leftarrow$ **if** ($d_{\min} < 0.1$) **then** -0.5 **else** 0.01
16: $Rw_{max} \leftarrow$ **if** ($d_{\max} > 13$) **then** -0.05 **else** 0.1
17: $Rw_c \leftarrow$ **if** ($collision$) **then** -0.5
18: $R_{wall} \leftarrow Rw_c + Rw_{min} + Rw_{max}$
19: **return** R_{wall}
20: **end procedure**



(a) Curve path scenario



(b) Cross path scenario

Fig. 4. Comparison of metrics for curve path and cross path scenarios across 8000 episodes in state-of-the-art neural models CADRL, LSTM, SARL, ST^2 and OURS with the reward functions (R_1), (R_2), (R_3) and (R_4).

Next, the procedure for computing the reward functions is derived from steps (12-20). In steps 13 and 14, the algorithm logs $d_{\min}$ and $d_{\max}$. A threshold of $d_{\min} < 0.1$ m indicates an imminent risk of collision or a breach of the robot's safe operating perimeter, penalizing the robot with $R_{w_{\min}} = -0.5$. Otherwise, the robot receives a small reward, $R_{w_{\min}} = 0.01$, each time step for safe navigation (step 15). In step 16, If $d_{\max}$, represented by the green color in Figure Fig. 3, exceeds a threshold of 13, the robot receives a penalty of $R_{w_{\max}} = -0.05$ at each time step when the robot moves away from the map boundaries. Otherwise, if the robot stays within the map, it receives a reward $R_{w_{\max}} = 0.1$ at the end of the scenario. This reward is particularly valuable during the early stages of training, as it helps guide the robot to stay within the map during exploration. In step 17, if the robot collides with a wall, it receives a penalty of $R_{w_c} = -0.5$. Finally, in steps (18-20) R_{wall} is computed as the sum of $R_{w_{\min}}$, $R_{w_{\max}}$, and R_{w_c} ensuring the robot navigates effectively while avoiding wall collisions.

4.5. Log information

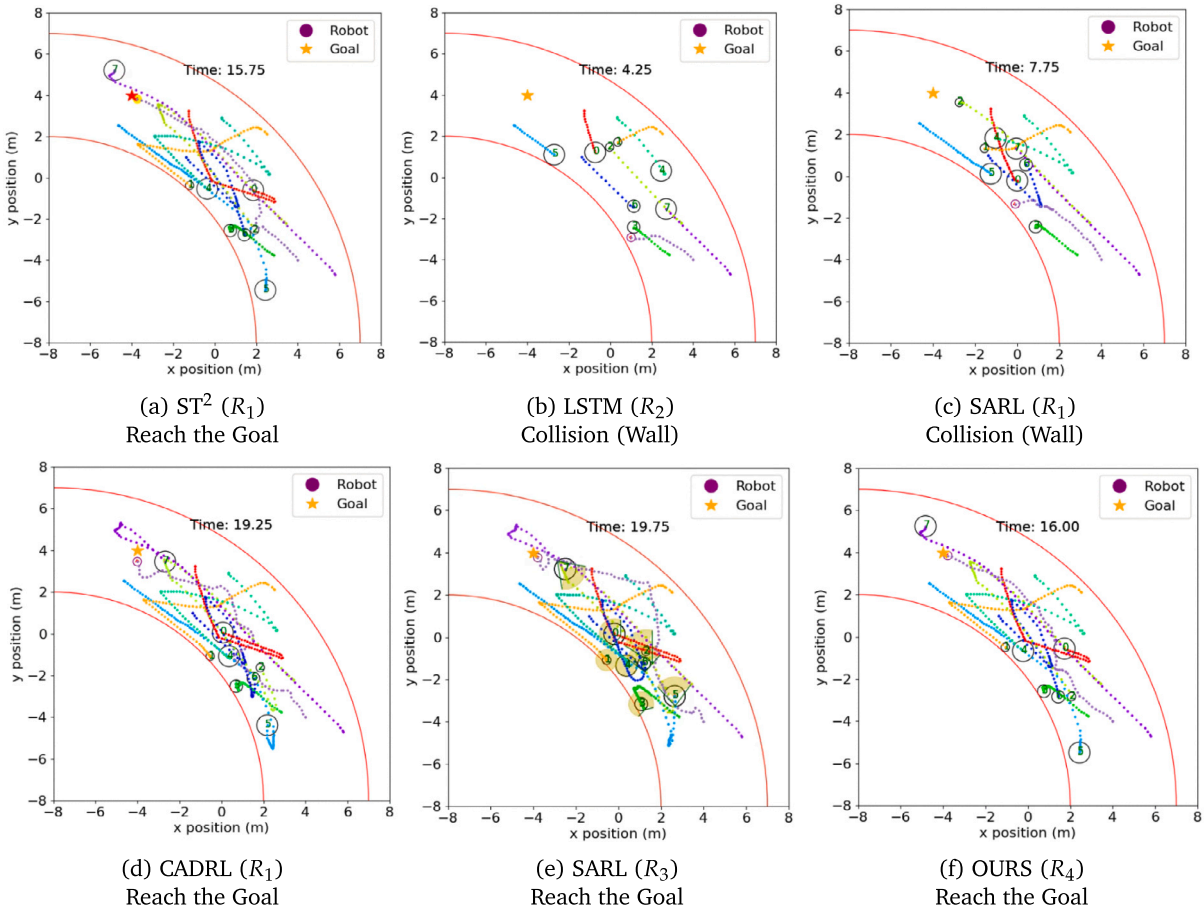
During the training and evaluation process, detailed logs are recorded for each episode, covering four scenarios, to monitor the agent's performance and assess the effectiveness of the reward structure. Traditional key metrics, including success rate (SR), collision rate (CR), time traveled (T), distance to the goal (DG), and cumulative reward (TR) obtained by the agent, are logged. Additionally, the values of each reward component are analyzed to assess their impact on the navigation task. These include the risk of robot collisions with other agents (R_d), stopping ability (R_s), heading to the goal (R_{hg}), reaching the goal (R_g), collisions (R_c), pushing forward (R_{pf}) to prevent freezing, and the rewards associated with collisions with walls (Rw_c), (Rw_{min}) and (Rw_{max}), which represent the rewards or penalties for safe distances with the walls. This comprehensive logging strategy ensures a clear understanding of the agent's learning progress and facilitates fine-tuning of the training process.

5. Simulation and experiments

In this section, we detail the simulation and the experiment results, as well as the metrics and log information, to provide a comprehensive understanding of the neural network's performance.

5.1. Simulation setup and results

The comparative analysis provided in this section employs four state-of-the-art neural models: LSTM [20,21], SARL [14,22], CADRL [15,16], and ST^2 [23] with reward functions (R_1), (R_2), and (R_3) compared with our model and reward function (R_4). The parameters used in the training are shown in Table 1. It is worth noting that the velocities of some objects are approximately twice the maximum velocity of the robot (1.5 m/s versus 0.8 m/s) to increase the complexity of the environments and improve the robot's decision-making. In our previous study [14], we evaluated DRL approaches without path constraints where the main goal was to avoid obstacles in crowd environments with static and dynamic obstacles. SARL demonstrates superior performance over LSTM and CADRL. In this case, including the wall limitations, CADRL improves its adaptability and performs better than the others. However, only our model, which includes the wall information in the neural network, steadily achieved higher performance in task navigation with path constraints, as shown in the metrics results in Fig. 4. LSTM obtained the lowest performance in both cross path and curve path scenarios using both rewards functions R_1 and R_2, with the robot reaching the goal in only a few cases. SARL neural model had slightly better results, especially when using R_2 functions, despite not receiving information on path constraints in the state or having a corresponding reward function to avoid collisions with walls. ST^2 used the R_1 reward function as proposed by the authors, showing higher results than LSTM and SARL, and similar results to CADRL. The implementation of a novel reward function, R_{stop}, enables the robot to improve decision-making in challenging scenarios with objects moving at higher velocities than the robot. R_1, the commonly used reward function in most previous research, exhibits lower performance during navigation tasks and collides more frequently with dynamic obstacles. CADRL using R_1 and R_2 had similar results, but in testing, CADRL with R_2 shows higher performances. The SARL method with R_3 has superior performance than CADRL and LSTM, it avoids the obstacles with a lower CR and achieves higher SR. The robot with R_3 prioritizes safety by adapting its navigation behavior for environments with obstacles and various speeds. However, the fact of not considering environmental limitations as walls and constrained scenarios, the robot tended to collide with the walls. Our proposed neural network model exhibits the highest performance. We include path constraint information through the three MLP layers, and R_4 improves robot decision-making by enabling the robot to gain



(a) ST^2 (R_1) Reach the Goal
(b) LSTM (R_2) Collision (Wall)
(c) SARL (R_1) Collision (Wall)
(d) CADRL (R_1) Reach the Goal
(e) SARL (R_3) Reach the Goal
(f) OURS (R_4) Reach the Goal

Fig. 5. Comparison of reward functions (R_1), (R_2),(R_3), and (R_4) is conducted across state-of-the-art neural network models (CADRL, LSTM, SARL, ST^2, and OURS). The scenario includes a curve path scenario with 8 dynamic obstacles.

small rewards for maintaining a safe distance from walls and incurring penalties for moving too close to them. Figs. 5 and 6 depict the 2D simulation test environments for the curve path scenario with 8 dynamic obstacles and the cross path scenario with 5 static and 10 dynamic obstacles, utilizing state-of-the-art neural network models. Several video tests with the four reward functions are available in the GitHub repository. A circle with a red arrow represents the robot, and its purple trajectories are shown in the graph along with the simulation test time. Dynamic objects' trajectories are also displayed, while static obstacles can be distinguished without any trajectory. In the curve path test, ST^2 (R_1) and OURS (R_4) reached the goal in 15.75 and 16 s, respectively. LSTM (R_2) and SARL (R_1) collided with the walls. While, CADRL (R_1) and SARL (R_3) navigated without collision until reaching the goal in 19.25 and 19.75 s, respectively. Conversely, in the cross path scenarios, CADRL (R_2) collided with a dynamic obstacle, LSTM (R_1) collided with the wall. SARL (R_3) attempted to avoid the walls, but this strategy resulted in a collision with the obstacle in front. SARL (R_2), ST^2 (R_1), and OURS (R_4) reached the goal in 20.75, 17.25 and 19.5 s, respectively. This demonstrates through simulation the capacity of OURS (R_4) to complete the task even in less time. Figs. 7(a) and 7(b) illustrate the average rewards obtained by state-of-the-art models and our model during navigation tasks across 8000 training episodes in the curve-path and cross-path scenarios. These plots reinforce the findings presented in Fig. 4, which displays metrics such as success rate, collision rate, timeout rate, and wall collision rate. As observed, our reward function consistently achieves higher rewards in both the cross-path and curve-path scenarios.

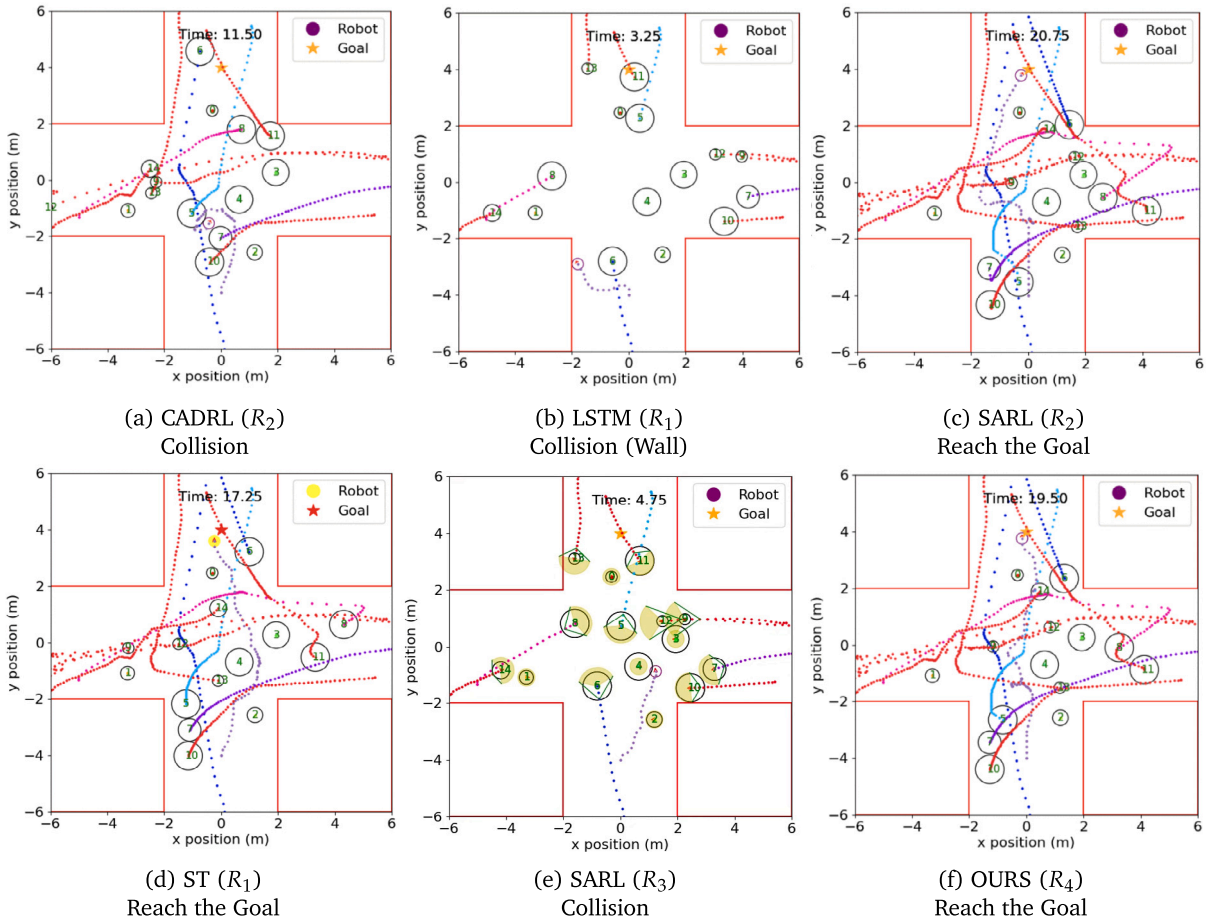
5.2. Log information results

Table 2 presents the log information of our neural network and reward functions (R_4) in both curve path and cross path scenarios during the evaluation of 3000 episodes, after training is completed with the 8000 episodes. The goal is to assess the impact of each reward function, highlighting its contribution to decision-making, human interaction, obstacle avoidance, and maintaining safe distances from walls. Each reward function is detailed in Section 3, and based on the results, we can adjust the gain values according to task requirements. The average total reward is shown in the first row, followed by the average of each reward contributing to navigation tasks' decision-making.

Particularly noteworthy are the new functions that enhance the robot's performance compared to previous works. The positive value of R_s enables the robot to navigate around objects moving faster than itself by stopping and allowing them to pass, thereby avoiding collisions. R_{pf} consistently provides a moderately high positive reward to prevent freezing, while our novel reward function R_{wall}, a combination of R_{w_c}, $R_{w_{\min}}$, and $R_{w_{\max}}$, ensures the robot maintains safe distances from walls. In these scenarios, R_{w_c} and $R_{w_{\min}}$ play a significant role in avoiding collisions with the walls and maintaining safe distances, while $R_{w_{\max}}$ rewards the robot for staying within the map boundaries.

5.3. Statistical analysis

To assess the statistical significance of differences between our proposed method and baseline methods, a paired Wilcoxon Signed-Rank Test was employed. This non-parametric test evaluates if the median differences between paired samples are significantly different.



(a) CADRL (R_2) Collision
(b) LSTM (R_1) Collision (Wall)
(c) SARL (R_2) Reach the Goal
(d) ST (R_1) Reach the Goal
(e) SARL (R_3) Collision
(f) OURS (R_4) Reach the Goal

Fig. 6. Comparison of reward functions (R_1), (R_2),(R_3), and (R_4) is conducted across state-of-the-art neural network models (CADRL, LSTM, SARL, ST^2, and OURS). The scenario includes a cross-path scenario with 5 static and 10 dynamic obstacles.

Table 2
The log information for the model(R_4) across 3,000 evaluation episodes, after training is complete, in both cross-path and curved-path scenarios.

	Curve path				Cross path			
Metric	No	Static	Dyn	Mix	No	Static	Dyn	Mix
T_R	12.02	11.21	12.05	11.79	11.81	9.19	9.61	10.34
R_g	9.98	9.22	9.04	9.32	9.99	7.94	7.51	7.83
R_c	0.00	−0.01	−0.04	−0.02	−0.01	−0.01	−0.23	−0.31
R_d	0.0	0.06	0.12	0.17	0.0	0.05	0.10	0.14
R_{hg}	0.98	0.73	0.82	0.70	0.91	0.89	0.85	0.92
R_s	0.0	0.0	0.56	0.42	0.0	0.0	0.28	0.21
R_{pf}	0.48	0.53	0.80	0.62	0.42	−0.13	0.51	0.78
Rw_c	−0.03	−0.01	−0.05	−0.09	0.0	−0.01	−0.06	−0.04
Rw_{min}	0.51	0.60	0.72	0.58	0.40	0.38	0.56	0.72
Rw_{max}	0.10	0.09	0.08	0.09	0.10	0.08	0.09	0.09

A one-tailed test with a *less than* alternative hypothesis was conducted at a significance level of 0.05. As presented in Table 3, the Wilcoxon statistic, W, was calculated for each metric. A p-value was less than 0.05, it indicated a statistically significant difference favoring our proposed method. The results revealed that our method significantly outperformed the baseline methods in terms of success rate ($p < 0.05$). In contrast, the alternative hypothesis was accepted ($p > 0.05$) for collision rate, timeout rate, and collision wall rate, indicating that these metrics had higher values, demonstrating the effectiveness of our method in improving them.

5.4. Experimental setup and results

The robot's width is 0.48 meters. It utilizes a 2D LiDAR for mapping and obstacle detection, along with odometry data and the AMCL package for robot localization. Communication between the sensors and the neural network was implemented using ROS. During the experimental test, we used the neural network models CADRL (R_2), SARL (R_3), and OURS (R_4) based on their higher success rates in training results. The validation test for the neural network was conducted in three scenarios:

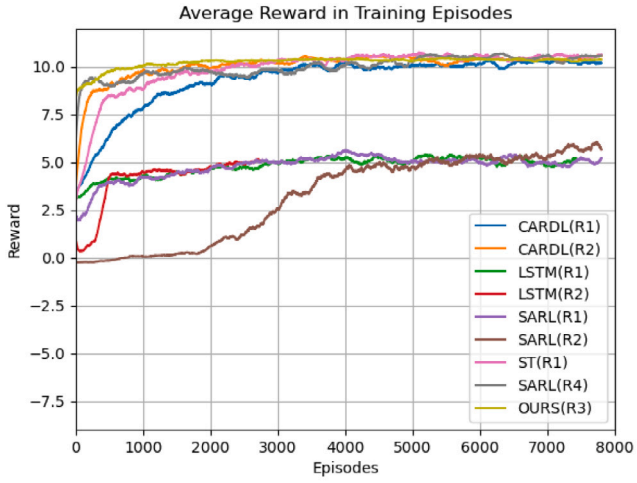
Scenario 1: Involves navigating a narrow space with six static obstacles, including empty and occupied chairs. The goal is located 7 meters from the initial position. The robot starts on a path with a width of 1.20 m, which narrows to 0.8 meters due to the placement of chairs in the lab scenario, making it challenging to navigate and reach the goal without collisions (see Figs. 8(a) and 8(b)).

Scenario 2: Includes static and dynamic obstacles, with a total distance of 12 meters from the start point to the goal. The robot begins its navigation along a narrow, curved path with an average curvature of 0.98 m^{-1} and a width of 0.7 meters. It then moves into a larger space containing static obstacles, including six chairs (either empty or occupied) and a static humanoid robot positioned at the center of the pathway. Additionally, five humans move around, attempting to avoid the robot's path, testing its navigation performance (see Fig. 8(c)).

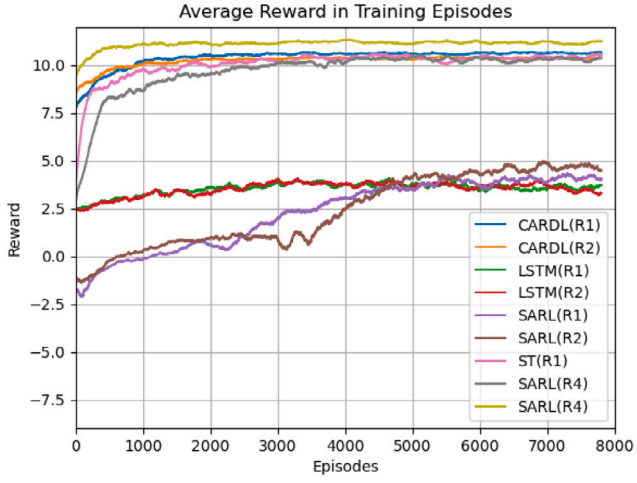
Scenario 3: The robot moves through a map with static obstacles (six chairs) and 7 moving people. The distance between the initial position and the goal is 21 meters. The robot must cross the same narrow spaces

Table 3
Statistical analysis on training results using Wilcoxon Signed-Rank Test.

Metrics	Method vs. Ours													
	CADRL(R1) [15]		CADRL(R2) [15]		LSTM(R1) [20]		LSTM(R2) [21]		SARL(R1) [22]		SARL(R2) [10]		ST^2 [23]	
	W	p-value	W	p-value	W	p-value	W	p-value	W	p-value	W	p-value	W	p-value
Success Rate	0.0	0.008	1.0	0.02	0.0	0.004	0.0	0.04	0.0	0.04	0.0	0.004	0.0	0.004
Collision Rate	18.0	0.942	16.0	0.876	21.0	0.986	21.0	0.986	21.0	0.986	19.0	0.962	19.0	0.962
Timeout Rate	15.0	0.978	8.0	0.863	15.0	0.978	15.0	0.978	36.0	1.0	21.0	0.986	36.0	1.0
Collision Wall Rate	24.0	0.954	21.0	0.986	28.0	0.991	36.0	1.0	19.5	0.971	28.0	0.991	28.0	0.991



(a) Curve path scenarios



(b) Cross path scenarios

Fig. 7. Comparison of the average reward in the 8000 scenarios across state-of-the-art neural network models (CADRL, LSTM, SARL, ST^2, and OURS).

and follow a curved path as in the previous experiments (see Fig. 8(e)). This experiment is related to dynamic scenarios in the training set.

The RVIZ displays the map and the robot's position in the instant of the picture. The quantitative results are shown in Table 4. In scenario 1, the CADRL (R_2) model collided with a person sitting on a chair, as illustrated in Fig. 8(a). Then, the robot proceeded with its navigation task until reaching the goal in 21 seconds. OURS (R_4) reaches the goal without collisions in 14 seconds (Fig. 8(b)). In scenario 2, CADRL (R_2) could not continue its path and became frozen in the narrow space. The robot moved only 3 meters in 48 seconds. This demonstrates that previous approaches and state-of-the-art methods have difficulty navigating through limited spaces, resulting in the robot failing to reach its destination. In scenario 3, SARL (R_3), as illustrated in Fig. 8(d), encounters challenges when navigating narrow spaces, resulting in a longer arrival time. However, it reaches its destination while avoiding dynamic and static obstacles. In contrast, OURS (R_4) required less time to traverse the constrained space before continuing its path, successfully avoiding obstacles and reaching the goal, as shown in Table 4. This experiment demonstrates that the robot can navigate narrow environments and efficiently handle typical crowd scenarios, a key challenge frequently addressed in previous research. Videos of the experiments are available in the GitHub repository for further analysis. Overall, our proposed model exhibits superior performance by integrating path constraints into its decision-making process and incorporating novel reward functions that ensure safe navigation in real-world applications.

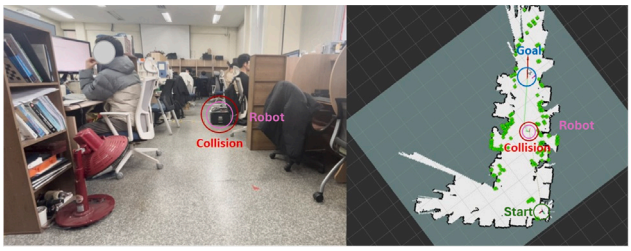
Table 4
Quantitative results from experimental tests with CADRL(R_2), SARL(R_3), and OURS(R_4): number of dynamic objects (Dy), number of static objects (St), reach the goal (RG), number of collisions with dynamic obstacles (C(dy)) and with static obstacles (C(st)), navigation time (t), and travel distance (d).

Scenarios	Model	Dy	St	RG	C(dy)	C(st)	t(s)	d(m)
Test 1	CADRL (R_2)	–	6	Yes	–	1	21	7
	OURS (R_4)	–	6	Yes	–	–	14	7
Test 2	CADRL (R_2)	5	7	No	–	–	48	3
	OURS (R_4)	5	7	Yes	–	–	58	12
Test 3	SARL (R_3)	7	6	Yes	–	–	93	21
	OURS (R_4)	7	6	Yes	–	–	50	21

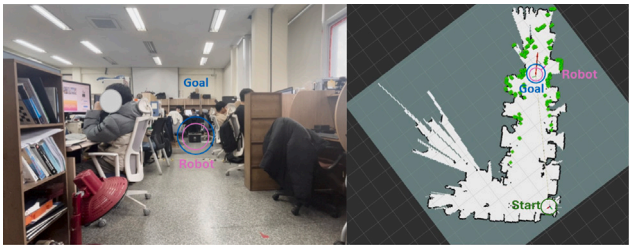
6. Conclusions

The paper's results demonstrate a comprehensive approach to autonomous navigation using deep reinforcement learning (DRL) techniques. We compare our proposed neural network model with CADRL, LSTM, SARL, and ST^2 neural networks. Our OURS (R_4) model effectively captures crucial environmental features, facilitating intelligent decision-making in navigation tasks, achieving higher success rates, and improving collision avoidance. Our experiments highlight the significance of incorporating wall information into the neural network and introducing novel reward functions to enhance navigation performance. A key factor contributing to the model's success is using Euclidean distance to evaluate wall proximity. This method was chosen for its lightweight computational nature, enabling real-time processing during navigation tasks.

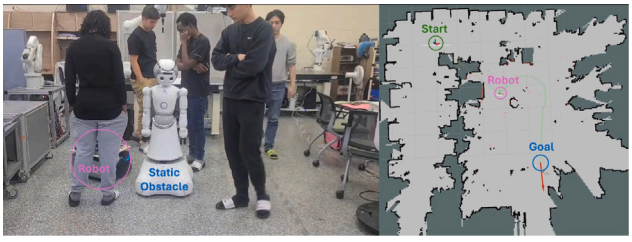
Future work will further optimize reward functions, explore additional environmental factors, and test alternative methods, such as the potential field approach. This will allow us to assess both its performance and computational cost, particularly in more complex environments where dynamic obstacle avoidance and smooth navigation are crucial.



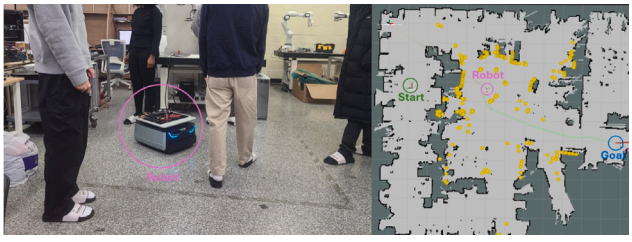
(a) CADRL(R_2) collides with a seated person in scenario 1



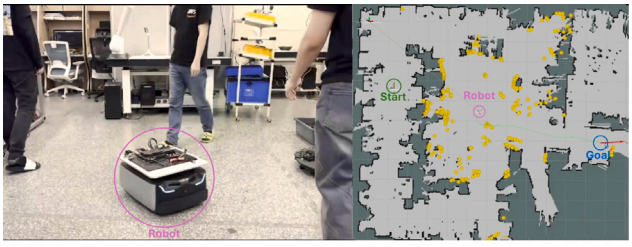
(b) OURS(R_4) reaches the goal without collision in scenario 1



(c) OURS(R_4) reaches the goal without collision in scenario 2



(d) SARL(R_3) takes longer to reach the goal in scenario 3



(e) OURS(R_4) reaches the goal without collision in scenario 3

Fig. 8. Experimental tests were conducted in scenarios (a) and (b) (narrow space), (c) (static and dynamic obstacles), and (d) and (e) (dynamic obstacles).

CRediT authorship contribution statement

Nabih Pico: Writing – original draft, Software, Methodology, Investigation, Formal analysis, Conceptualization. **Estrella Montero:** Writing – original draft, Validation, Software, Formal analysis. **Alisher Amirbek:** Writing – original draft, Software, Investigation. **Eugene Auh:** Writing – review & editing, Validation, Resources. **Jeongmin Jeon:** Writing – review & editing, Validation, Resources. **Manuel S. Alvarez-Alvarado:** Writing – review & editing, Validation, Methodology. **Babar Jamil:** Writing – review & editing, Validation, Methodology. **Redhwan Algabri:** Writing – review & editing, Validation, Methodology. **Hyungpil Moon:** Writing – review & editing, Supervision, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors would like to thank the editors and anonymous reviewers for providing insightful suggestions and comments to improve the quality of research paper.

References

[1] K. Zhu, T. Zhang, Deep reinforcement learning based mobile robot navigation: A review, Tsinghua Sci. Technol. 26 (5) (2021) 674–691, http://dx.doi.org/10.26599/TST.2021.9010012.

[2] P. Long, W. Liu, J. Pan, Deep-learned collision avoidance policy for distributed multiagent navigation, IEEE Robot. Autom. Lett. 2 (2) (2017) 656–663.

[3] L. Tai, J. Zhang, M. Liu, W. Burgard, Socially compliant navigation through raw depth inputs with generative adversarial imitation learning, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, IEEE, 2018, pp. 1111–1117.

[4] J. Wei, B. Zhu, Model predictive control for trajectory-tracking and formation of wheeled mobile robots, Neural Comput. Appl. 34 (19) (2022) 16351–16365.

[5] A.F. Foka, P.E. Trahanias, Predictive autonomous robot navigation, in: IEEE/RSJ International Conference on Intelligent Robots and Systems, vol. 1, IEEE, 2002, pp. 490–495.

[6] D. Helbing, P. Molnar, Social force model for pedestrian dynamics, Phys. Rev. E 51 (5) (1995) 4282.

[7] J. Van den Berg, M. Lin, D. Manocha, Reciprocal velocity obstacles for real-time multi-agent navigation, in: 2008 IEEE International Conference on Robotics and Automation, Ieee, 2008, pp. 1928–1935.

[8] E.E. Montero, H. Mutahira, N. Pico, M.S. Muhammad, Dynamic warning zone and a short-distance goal for autonomous robot navigation using deep reinforcement learning, Complex Intell. Syst. (2023) 1–18, http://dx.doi.org/10.1007/s40747-023-01216-y.

[9] M. Nishimura, R. Yonetani, L2b: Learning to balance the safety-efficiency trade-off in interactive crowd-aware robot navigation, in: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS, IEEE, 2020, pp. 11004–11010.

[10] N. Pico, J. Lee, E. Montero, E. Auh, M. Tadese, J. Jeon, M.S. Alvarez-Alvarado, H. Moon, Enhancing autonomous robot navigation based on deep reinforcement learning: Comparative analysis of reward functions in diverse environments, in: International Conference on Control, Automation and Systems, Iccas, ICROS, 2023, pp. 1415–1420, http://dx.doi.org/10.23919/ICCAS59377.2023.10316876.

[11] A.J. Sathyamoorthy, U. Patel, T. Guan, D. Manocha, Frozone: Freezing-free, pedestrian-friendly navigation in human crowds, IEEE Robot. Autom. Lett. 5 (3) (2020) 4352–4359.

[12] G. Ferrer, A. Sanfeliu, Anticipative kinodynamic planning: multi-objective robot navigation in urban and dynamic environments, Auton. Robots 43 (6) (2019) 1473–1488.

[13] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, W. Matusik, Prediction-guided multiobjective reinforcement learning for continuous robot control, in: International Conference on Machine Learning, PMLR, 2020, pp. 10607–10616.

[14] N. Pico, B. Lee, E. Montero, M. Tadese, E. Auh, M. Doh, H. Moon, Static and dynamic collision avoidance for autonomous robot navigation in diverse scenarios based on deep reinforcement learning, in: 20th International Conference on Ubiquitous Robots, UR, 2023, pp. 281–286, http://dx.doi.org/10.1109/UR57808.2023.10202431.

[15] L. Liu, D. Dugas, G. Cesari, R. Siegwart, R. Dube, Robot navigation in crowded environments using deep reinforcement learning, in: IEEE International Conference on Intelligent Robots and Systems, 2020, pp. 5671–5677, http://dx.doi.org/10.1109/IROS45743.2020.9341540.

[16] Y.F. Chen, M. Liu, M. Everett, J.P. How, Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning, in: Proceedings - IEEE International Conference on Robotics and Automation, IEEE, 2017, pp. 285–292, http://dx.doi.org/10.1109/ICRA.2017.7989037.

[17] M. Pfeiffer, G. Paolo, H. Sommer, J. Nieto, R. Siegwart, C. Cadena, A data-driven model for interaction-aware pedestrian motion prediction in object cluttered environments, in: 2018 IEEE International Conference on Robotics and Automation, ICRA, 2018, pp. 5921–5928, http://dx.doi.org/10.1109/ICRA.2018.8461157.

[18] D. Dugas, J. Nieto, R. Siegwart, J.J. Chung, NavRep: Unsupervised Representations for Reinforcement Learning of Robot Navigation in Dynamic Human Environments, in: Proceedings - IEEE International Conference on Robotics and Automation 2021-May, Vol. 2021-May, Icra, (Icra) IEEE, 2021, pp. 7829–7835, http://dx.doi.org/10.1109/ICRA48506.2021.9560951, arXiv:2012.04406.

[19] Z. Zhou, Z. Zeng, L. Lang, W. Yao, H. Lu, Z. Zheng, Z. Zhou, Navigating robots in dynamic environment with deep reinforcement learning, IEEE Trans. Intell. Transp. Syst. 23 (12) (2022) 25201–25211, http://dx.doi.org/10.1109/TITS.2022.3213604.

[20] A. Alahi, K. Goel, V. Ramanathan, A. Robicquet, L. Fei-Fei, S. Savarese, Social lstm: Human trajectory prediction in crowded spaces, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 961–971.

[21] N. Guo, C. Li, T. Gao, G. Liu, Y. Li, D. Wang, A fusion method of local path planning for mobile robots based on LSTM neural network and reinforcement learning, Math. Probl. Eng. 2021 (1) (2021) 5524232.

[22] K. Li, Y. Xu, J. Wang, M.Q. Meng, SARL*:Deep reinforcement learning based human-aware navigation for mobile robot in indoor environments, in: EEE International Conference on Robotics and Biomimetics, ROBIO 2019, (no. December) 2019, pp. 688–694, http://dx.doi.org/10.1109/ROBIO49542.2019.8961764.

[23] Y. Yang, J. Jiang, J. Zhang, J. Huang, M. Gao, ST ^{2} : Spatial-temporal state transformer for crowd-aware autonomous navigation, IEEE Robot. Autom. Lett. 8 (2) (2023) 912–919.

[24] N. Pico, E. Montero, M. Vanegas, J.M. Erazo Ayon, E. Auh, J. Shin, M. Doh, S.-H. Park, H. Moon, Integrating radar-based obstacle detection with deep reinforcement learning for robust autonomous navigation, Appl. Sci. 15 (1) (2024) 295, http://dx.doi.org/10.3390/app15010295.

[25] J. Kapukotuwa, B. Lee, D. Devine, Y. Qiao, Multiros: ROS-based robot simulation environment for concurrent deep reinforcement learning, in: 2022 IEEE 18th International Conference on Automation Science and Engineering, CASE, 2022, pp. 1098–1103, http://dx.doi.org/10.1109/CASE49997.2022.9926475.

[26] C. Chen, Y. Liu, S. Kreiss, A. Alahi, Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning, in: Proceedings - IEEE International Conference on Robotics and Automation, vol. 2019-May, 2019, pp. 6015–6022, http://dx.doi.org/10.1109/ICRA.2019.8794134.

[27] P. Trautman, A. Krause, Unfreezing the robot: Navigation in dense, interacting crowds, in: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2010, pp. 797–803.

[28] Y. Hoshen, Vain: Attentional multi-agent predictive modeling, Adv. Neural Inf. Process. Syst. 30 (2017).