Low Compute End-to-End Robot Navigation from Depth Images Using Deep Reinforcement Learning

Shahrukh Hussain[1], Abdul Wahid[1] and Anayatullah[2]
[1]Control, Automation and Robotics Lab, National Centre of Robotics and Automation, Rawalpindi, Pakistan.
[2]Balochistan University of Information Technology, Engineering and Management Sciences, Quetta, Pakistan.
*__Correspondence__: abdul.wahid@buitms.edu.pk

Citation | Hussain. S, Wahid. A, Anayatullah, "Low Compute End-to-End Robot Navigation from Depth Images Using Deep Reinforcement Learning", IJIST, Special Issue pp 447-462, May 2026
DOI | https://doi.org/10.33411/IJIST/1816
Received | March 22, 2026 **Revised |** May 02, 2026 **Accepted |** May 08, 2026 **Published |** May 12, 2026.

Effective and compute-efficient navigation is essential for mobile robots to operate autonomously in complex environments such as crowded places like airports and shopping malls. Sensors mounted on mobile robots play a vital role in decision-making by providing information about the surroundings. Among these sensors, depth cameras are a type of visual sensor that provides rich depth information of the surroundings, enabling robots to comprehend the 3-dimensional structure of the environment, assisting the robot in robust obstacle avoidance and navigation. In this paper, we aim to achieve autonomous navigation of a differential drive robot using only the depth information of the environment by employing a simple Convolutional Neural Network (CNN) architecture. CNN interprets the depth images captured by the depth camera and generates corresponding actions for the robot, while maintaining computational efficiency due to the limited computational resources of mobile robots. We employ Deep Reinforcement Learning (DRL) with curriculum learning paradigm to train our CNN in two Gazebo robotic simulation environments with increasing complexity. This approach increases the generalization of the model and enhances its adaptability in unobserved environments. The CNN learns to navigate autonomously using only depth information from the environment. The trained model is then evaluated by deploying it in an unseen simulation environment. Results show that the agent converged in 1,100 episodes in the primary environment. Furthermore, to demonstrate the model's adaptability, it is deployed in a real-life laboratory environment where it achieved a 70% success rate after training for 1,000 episodes.

Keywords: Autonomous Navigation, Deep Reinforcement Learning, Depth Images, Convolutional Neural Networks, Mobile Robots.

Introduction:

The field of robotics has undergone significant breakthroughs with the incorporation of visual sensors, enabling them to perceive the world as humans, enhancing their pattern recognition ability, and subsequently improving their problem-solving skills like autonomous navigation [1] and manipulation [2]. Among visual sensors, RGB cameras are widely used as they provide rich 2D information about the environment, including color and texture features. However, they do not provide explicit depth information, which is essential for estimating the distance of objects from the robot. This limitation negatively impacts the performance of robots relying solely on RGB input, particularly in autonomous navigation tasks where accurate distance estimation is critical for obstacle avoidance.

Alternatively, sensors such as LiDAR and ultrasonic sensors can be used to estimate distance. While these sensors provide depth information, they are often limited to 1D or 2D representations, which are insufficient for complex environments such as crowded indoor spaces or disaster scenarios. Although 3D LiDAR systems offer more detailed spatial perception, they are typically expensive and require high power consumption, making them impractical for low-cost mobile robots. Sensor fusion techniques combining RGB and ranging sensors have also been explored [3]; however, these approaches increase system complexity, computational cost, and hardware requirements.

In contrast, depth cameras provide dense depth information in a 2D image format, where each pixel corresponds to the distance from the scene. This enables a richer 3D understanding of the environment while maintaining a compact representation, making depth cameras well-suited for autonomous navigation in resource-constrained robotic systems.

In terms of processing, Convolutional Neural Networks (CNNs) have demonstrated strong performance in interpreting visual data, enabling robust feature extraction and spatial understanding [4][5]. When combined with depth images, CNNs can effectively learn perception-to-action mappings, enhancing navigation and obstacle avoidance capabilities in mobile robots [6][7].

Similarly, CNNs trained using Deep Reinforcement Learning (DRL) have proven highly effective in solving various decision-making problems, including visual navigation with high throughput performance and generalization, surpassing human benchmark in video games [8][9] and board games [10][11]. This success has inspired the use of DRL for the training of different types of robots enabling them to learn complex behaviors to accomplish tasks by interacting with the environment in a trial-and-error fashion.

Recent advancements in deep reinforcement learning (DRL) have significantly improved autonomous navigation in dynamic and unstructured environments [12][13]. Contemporary studies demonstrate enhanced robustness and adaptability of DRL-based agents in crowded and unknown environments, particularly when trained using end-to-end perception-to-action frameworks. Furthermore, recent survey works highlight a growing shift toward lightweight and compute-efficient navigation models for deployment on embedded robotic platforms [14]. However, despite these advancements, challenges such as generalization, computational efficiency, and reliable real-world deployment remain open research problems [15].

To achieve autonomous navigation, the mobile robot is deployed in an environment where it interacts with the environment by observing, taking actions, and receiving feedback in the form of carefully orchestrated rewards or penalties, allowing the robot to develop an efficient and reliable navigation and obstacle-avoiding strategy.

Therefore, combining depth cameras with CNNs and training them using deep reinforcement learning (DRL), enables map-less navigation in unknown environments by learning perception-to-action strategies that are computationally efficient for resource-constrained mobile robots.

Research Gap:

Despite significant advancements in deep reinforcement learning (DRL) for autonomous navigation, several limitations remain in existing approaches. Many methods rely on high-compute architectures, such as deep convolutional networks or multi-sensor fusion systems, which are unsuitable for deployment on resource-constrained mobile robots. Additionally, approaches utilizing RGB-based perception often require auxiliary networks for depth estimation, increasing computational overhead and reducing real-time applicability.

Furthermore, while depth-based navigation has shown promising results, most existing methods either depend on complex network architectures or lack robust generalization when transferred from simulation to real-world environments. The sim-to-real gap remains a critical challenge, particularly for lightweight models operating under limited hardware constraints.

Therefore, there is a need for a compute-efficient, end-to-end navigation framework that utilizes depth information directly, minimizes architectural complexity, and demonstrates reliable performance in both simulated and real-world environments.

Objectives:

The primary objectives of this study are as follows:

To develop a lightweight 2-layer CNN-based DRL model for mapless navigation using only raw depth data.

To validate a curriculum learning strategy that facilitates sim-to-real transfer on low-compute hardware.

Novelty:

We have devised a simple and small-sized CNN architecture combined with curriculum DRL. The model was specifically designed for low power mobile robots utilizing resource constrained hardware such as Raspberry Pi 4., differentiating this work from current state-of-the-art DRL models that depend on high-resolution LiDAR or costly GPUs."

Related Works:

DRL has enabled robots to achieve robust perception-to-action strategies in the navigation of their surroundings and avoidance of collisions. The selection of sensors and the nature of action space of the robot determines the performance and efficiency of the autonomous vehicle. State-of-the-art methods involve usage of visual sensors, ranging sensors, and integration of multiple sensors to provide the robot with comprehensive information of the environment to achieve navigation.

Recent literature [16][17] further explores DRL-based navigation using both visual and depth-based inputs, showing strong performance in complex and dynamic environments. Several works emphasize improving computational efficiency by designing lightweight models suitable for real-time inference on embedded systems. Additionally, modern approaches investigate advanced architectures and training strategies to enhance generalization and sim-to-real transfer [18]. Despite these developments, many existing methods still rely on high-compute architectures or multi-sensor fusion, limiting their applicability in low-resource robotic platforms.

Monocular camera-based perception:

Although RGB images provide rich visual features and spatial cues, they lack explicit depth information required for reliable navigation. As a result, additional processing is often required to estimate depth or relative distance from the scene. As in [19], where the CNN model is trained to navigate in the GTA V game perceiving the environment through RGB images. The observed states are pre-processed by an additional pretrained network performing object detection from which the results are fed into CNN that is also pre-trained on a large dataset to interpret the visual data and encodings from the preprocessing network. This

approach requires significant computational resources, making it unsuitable for deployment on resource-constrained mobile robots.

Similarly, in [20] the agent is trained to navigate in an office environment using only RGB images. However, they also rely on depth information which they predict from the images to enhance the navigation, the agent is trained on a dataset that imitates real-life office environment and is then fine-tuned on real-life images using RGB-D camera. This approach increases computational overhead and reduces adaptability when deployed in more complex and dynamic environments. In [21] the agent is trained to navigate on depth information where the depth is estimated from the observed RGB images through an additional network. This results in increased processing time and reduced accuracy, limiting its effectiveness in real-time applications. In similar context, [22] estimates depth from a monocular camera and samples 10 distance points from the depth map and trains the RL model with 3 discrete actions i.e., moving forward, right and left. This approach uses additional computation to estimate distances from scene while in comparison 2D LiDAR provides more accurate distance measurements and results in less computation.

LiDAR-based perception:

Among the ranging sensors, LiDARs offer precise distance information of the surroundings. In most cases [23-31], 2D LiDARs are used as a perception unit for the robot. While LiDARs with the advantage of accurate distance measurements, lack in observing the dimensionality of the environment, making them less effective in crowded environments. As [28] states the inability of LiDAR to provide a detailed representation of the environment leading to reduced performance of the robot in navigation. To improve the perception of the agent, often the data from LiDAR is fused with RGB images. In [32], the RGB images are fed to pre-trained ResNet to extract features and are concatenated with LASER scan data before feeding into DRL model. Although a continuous action space enables more complex navigation behavior, it reduces overall computational efficiency and increases system complexity. In [33], 3D LiDAR is used for the depth perception while still using 2D LiDAR for determining the distance from the obstacles, increasing the cost and compute complexity of the system.

Depth Perception:

The depth images provide rich spatial information that enables robust navigation of the robot in complex and dynamic situations. In [23], DRL is used to train a robot with depth-wise separable CNN architecture in order to learn to navigate to a goal position while a predefined map is provided, the continuous action space allows for complex maneuvers in complex environment, while compromising the computational effectiveness of the system. Similarly, in [24] the depth images are interpreted by introducing layer normalization in the CNN model, resulting in overall larger networking resulting in slower processing time and high computation power making it unsuitable for applications in small computational units, as is the mostly case with mobile robots.

Deep learning and other methods:

Authors in [34] train a DRL model in simulation environment that avoids obstacles and navigates where the distance to obstacles are known. Similarly, in [35] the RL agent is trained in a 2D simulation environment consisting of static and dynamic obstacles, the distance from obstacles are known to the agent by using ray casting. These approaches often rely on LiDAR and other ranging sensors during real-world deployment, resulting in a significant sim-to-real gap.

While other methods offer abstraction in navigation by considering parameters such as distance from the obstacles and pre-known maps of the environment.

Some methods use 2D maps to perform obstacle avoidance and navigation in the environment. In [36] the agent is trained to navigate in the environment when a 2D map is

provided where the goal and starting positions are known. In [37], navigation is achieved by considering the optimal points that lead to clear paths in 2D map which are generated in real-time using 2D LiDAR and similarly, in [32], the mobile robot learns to navigate while the 2D map is created by converting the observed point cloud from depth camera into a cost map. The methods like above often depend on supplementary algorithms for path planning to find an optimal path in the 2D map to follow.

The objective of this work is to achieve accurate and reliable navigation in a mobile robot that is equipped with a depth camera. We introduce a simple CNN architecture that can understand the depth images acquired from depth cameras and guide the mobile robot to roam and navigate in the deployed environment.

Background:

Reinforcement Learning:

Popularized by Richard S. Sutton and Andrew G. Barto [38], Reinforcement Learning (RL) is a branch of machine learning where an agent learns from the consequences of its actions.

In RL, the agent is rewarded or penalized according to its actions when it is deployed in an environment where it can observe the environment and take actions accordingly.

The goal of every reinforcement learning agent is to take actions in such a way, to maximize the cumulative rewards it receives over the period it interacts with the environment. Our agent's objective is to navigate without colliding in complex scenarios based on the observed depth information from the environment. Figure-1 shows the control loop of a RL agent.

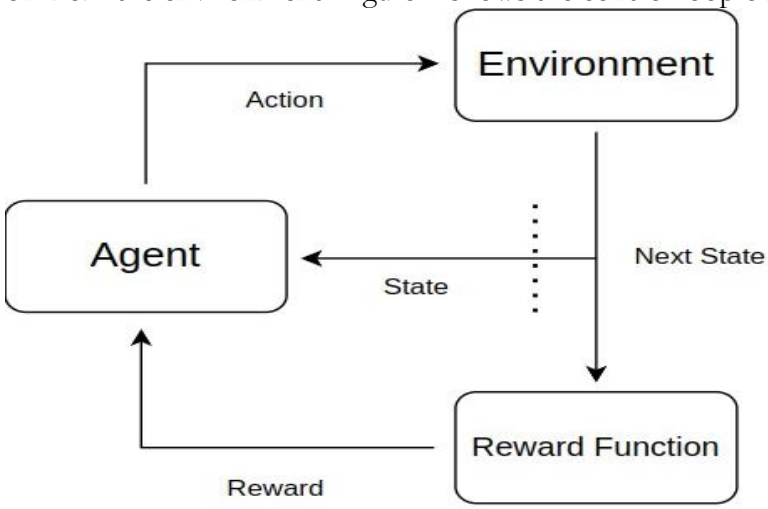


Figure 1. Reinforcement Learning Control Loop.

State Space (Observation Space):

The state space is the set of all information an agent can observe in an environment. Here, the agent's state space consists of depth information from the environment that is observed as 640 × 360-pixel grayscale images using an Intel RealSense D455 depth camera [39]. Each pixel in the 2D frame represents the distance to objects in the scene, where bright pixels indicate far objects and dark pixels represent near ones. An illustration of a depth image captured from the camera is shown below.

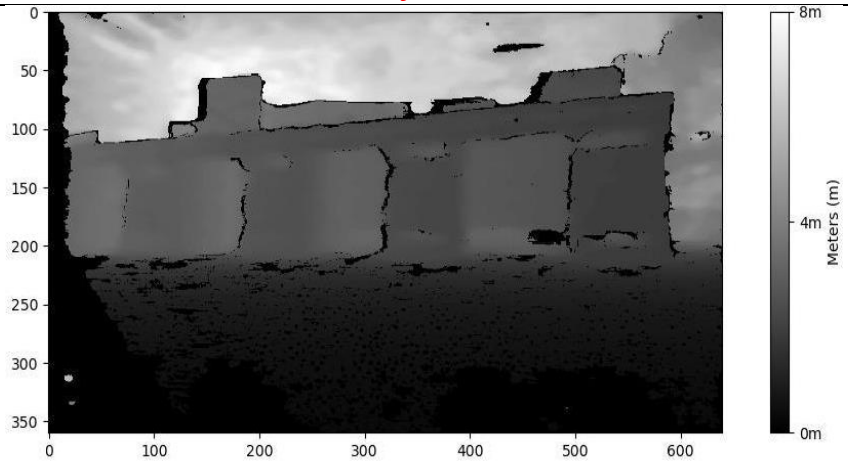


Figure 2. Raw depth image captured from IntelRealSense D455

Action Space:

The action space defines the set of values from the available parameters through which an agent can interact with the environment. Our agent's action space is continuous values of the linear and angular velocities for the differential drive robot. The agent's action can be represented as a vector of linear and angular velocities, as given below:

$$a = [v_t - \omega_t]$$

Where v_t is the linear velocity and ω_t is the angular velocity of the robot at time step t . The valid ranges of the velocities for the agent in our case are given as:

$0 \leq v \leq 0.22$ and $-0.5 \leq \omega \leq 0.5$. Where the linear velocity is measured in m/s (meters per second) and angular velocity in rad/s (radians per second).

During training, the policy operates in a stochastic manner, where actions are sampled from the Gaussian distributions defined by the predicted mean and standard deviation for both linear and angular velocities. This stochastic sampling encourages exploration and allows the agent to learn robust navigation strategies.

During deployment, the policy is executed in a deterministic manner by selecting the mean values of the learned distributions for both linear and angular velocities. This ensures stable and consistent behavior of robots in real-world environments. No discretization of the action space is performed, and the control signals are directly applied as continuous velocity commands.

Reward Function:

In RL, the rewards tell the agent how good the action was for the particular state. In this case the objective to navigate autonomously without colliding while observing the depth information through depth images. The collision is detected if the number of dark pixels (representing near objects) of certain values exceed the set threshold. We are considering the collision occurs when number of pixels with value ≤ 7 exceeds 5% of the image frame. So, for a depth image of size 640×360, the collision occurs if the number of pixels with values ≤ 7 exceeds the value of 11520. Overall, the agent is rewarded for its movement and is penalized when it collides. The reward function can be summarized by the following equation:

$$r(s, v, \omega) = \{-100\,, if\ collides\ \frac{v}{|\omega| + 0.1}\,, otherwise$$

Where, v is the linear velocity, ω is the angular velocity and s is the state of the agent.

The reward function is designed to balance safety with efficiency to achieve the goal. The collision penalty ($rcollision = -100$) was determined through sensitivity analysis. Setting the values to be less than -50 resulted in reckless behavior where the agent prioritized speed over safety, while values exceeding -150 led to overly cautious, stagnant policies. Moreover, the movement reward incentivizes high linear velocity (v) and penalizes sharp,

energy-inefficient angular turns (ω). The constant +0.1 acts as a survival bonus to prevent the robot from remaining stationary. Furthermore, the collision threshold—defined as 5% of the frame having a depth value ≤7—was calibrated to represent a physical proximity of 0.2m using the D455 sensor. This threshold area of 5% is critical to filter out sensor noise and small artifacts as they do not represent a traversable threat, ensuring the agent only reacts to significant obstacles.

Policy function:

The real objective of an RL agent is to achieve an optimal policy that can maximize the cumulative return from the environment. The RL agent takes actions based on the policy it has learned. The policy function represented by π(s) defines the behavior of the agent and is responsible for producing actions against the observed states. It can either be deterministic, meaning that it outputs the exact action for a given state, or it can be stochastic, which outputs a probability distribution for the action space, where the action with highest probability is selected from the distribution. In this case, we will be using stochastic policy to introduce randomness in the selection of the action. The stochastic policy function outputs the mean and standard deviation values for both linear and angular velocities, creating two independent gaussian distributions. The actions are then sampled from the distribution.

State-Value function:

The state-value function outputs the expected return (cumulative reward) of the observed state. It indicates how beneficial it is for the agent to be in that state. To maximize the cumulative rewards, the policy function will try to produce such actions to visit states with higher values. The state-value function is given by:

$$V(s) = E[\sum_{t=0}^{T} \gamma^t r_t]$$

Where is the discount factor and r_t is the reward at time step t.

Proximal Policy Optimization:

Proximal Policy Optimization (PPO) [40] belongs to actor-critic family of DRL algorithms which consist of the above defined two functions i.e., the policy and value, often referred to as actor and critic. The PPO extends Advantage Actor Critic (A2C) algorithm demonstrating faster convergence and stable training by leveraging clipped surrogate objective which makes PPO less prone to hyperparameters tuning. Its adaptability has led to its usage in various applications, while outperforming humans in video games [9] and performing robust navigation tasks [35]. The PPO updates the policy parameters based on policy gradient algorithm, given by the equation:

$$\nabla_\theta J(\pi\theta) = E[\sum_{t=0}^{\infty} \nabla log\pi\theta(a_t|s_t) . A(s_t, a_t)]$$

Where, J(πθ) is the objective function which includes the advantage function and the log probability of policy, the goal here is to maximize the probability of the actions generated by the policy $\pi\theta$ with θ parameters that result in high advantage values over the time period T. The PPO achieves its stability by using clipped surrogate objective which makes sure that the gradient updates are not very large. The modified objective is given by:

$$J(\theta) = min\,[\frac{\pi_\theta}{\pi_{\theta old}} . A(s,a), clip\left(\frac{\pi_\theta}{\pi_{\theta old}}, 1 - \epsilon, 1 + \epsilon\right) . A(s,a)]$$

Where, ϵ is the value of clipping parameter which is usually set between 0.2 to 0.8 and A(s, a) the advantage function. This function evaluates the taken action a_t compared to average actions in state s_t . The advantage function is given by:

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t)$$

As discussed earlier, V (st) is the state-value function, while Q(st, at) is the Q-function

or state-action value function that tells the value of action a_t in state s_t The PPO algorithm estimates the advantage function using Generalized Advantage Estimation (GAE) [41], rather than incorporating another neural network for the approximation of Q-function. The GAE reduces the variance of the policy and stabilizes the training of the algorithm.

Methodology:

Figure 3 illustrates the complete system architecture, detailing the end-to-end pipeline from data acquisition and multi-stage pre-processing (downscaling, cropping, and Telea inpainting) to action execution and the continuous feedback loop during deployment (sim-to-real transfer). To achieve navigation and obstacle avoidance we are using a prototype differential drive robot TurtleBot3 burger [42] that is equipped with the Intel RealSense D455 depth camera. The robot is trained to navigate based on the observed depth information from the environment through depth images, and the CNN is used to encode the depth images into actions for the mobile robot. Considering the applications on mobile robots with limited processing power, this approach maintains computational efficiency by downscaling the depth images and using a simple, small-sized CNN architecture. The CNN is trained using the PPO algorithm, the robot is deployed in simulation environment so that it can collect training data by observing, acting and receiving feedback in the environment. To simulate and establish communication between various components of the robot, we have utilized Robot Operating System (ROS) as a base framework. The publish and subscribe messaging model of ROS simplifies acquisition of data from sensors, such as the depth camera, and sending of linear and angular velocities values to the micro controller which converts them into motor rotation. Furthermore, it bridges the gap between simulation and real-life environments by enabling the effective transfer of the trained algorithm to real-world applications with minimal modifications.

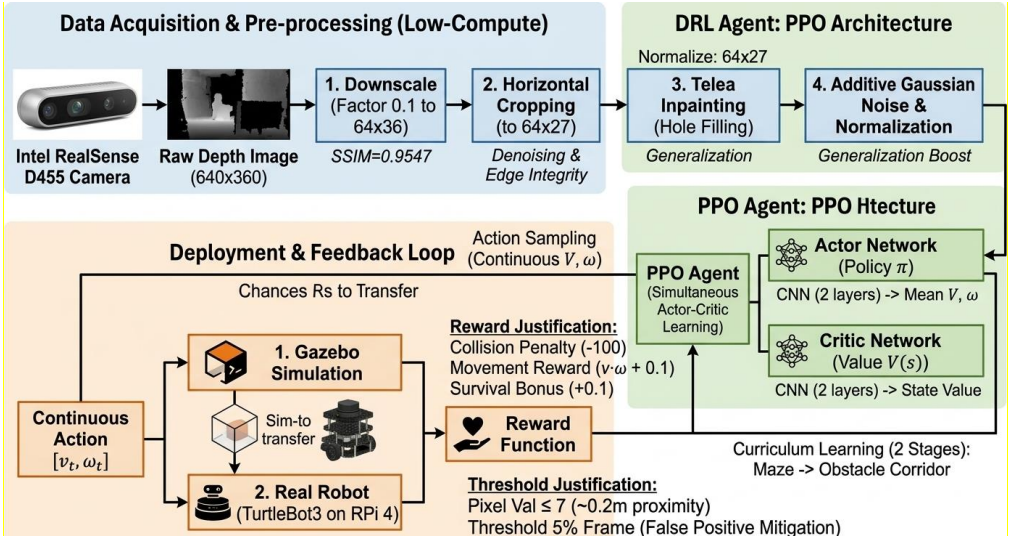


Figure 3. Full system architecture with end-to-end pipeline from data acquisition and multi-stage pre-processing to action execution and the continuous feedback loop during deployment.

The step-by-step explanation of Figure 3 is as under:

Data Acquisition: Raw 640 × 360 depth frames are captured via the Intel RealSense D455 sensor.

Pre-processing: Images are downsampled to 64x27 to reduce computational load. Telea inpainting is applied to fill 'holes' (no-data pixels), and horizontal cropping focuses the agent's field of view on traversable space.

Feature Extraction: A 2-layer CNN extracts spatial features from the normalized depth map.

Policy Execution: The PPO Actor network outputs continuous linear and angular velocities (v,ω).

Feedback Loop: The agent receives a reward based on its progress toward the goal or a −100 penalty if the 5% pixel threshold (proximity ≤0.2m) is breached. This cycle runs at 10 Hz.

Experimental Setup:

The training sessions are conducted in Gazebo robotic simulation software as it is compatible with ROS and provides high-fidelity physics simulation, enabling accurate modeling of real-world environments. It also provides plugins for acquiring realistic depth images from the simulation environment which is the crucial part for training the CNN model.

Observation Pre-Processing: The observed depth images from the environment are of size 640×360 pixels. To reduce the computation and processing time the depth images are downsized by factor of 0.1 resulting in a resolution of 64 × 36. We observe that resizing preserves structural features while providing an advantage in the form of computational cost of the system. To validate this statement, we employ Structural Similarity Index Measure (SSIM) to measure the similarity between the original and resized images considering their structures and features. The SSIM score of the comparison is 0.9547, indicating high similarity between the observed and downsized images. Additionally, the lower portion of the depth image contains constant values as the perception of the robot remains the same for the floor without falling edges. This area is cropped out further decreasing the size of image to 64 × 27. To further enhance the training performance of the agent, we increase the amount of data by adding random noise with a mean of 0 and standard deviation of 15. This increases the state space variability, which improves generalization of the agent during navigation. The images are finally normalized to a range of -1 to 1 using mean and standard deviation normalization. This standardization of scale has shown faster convergence.

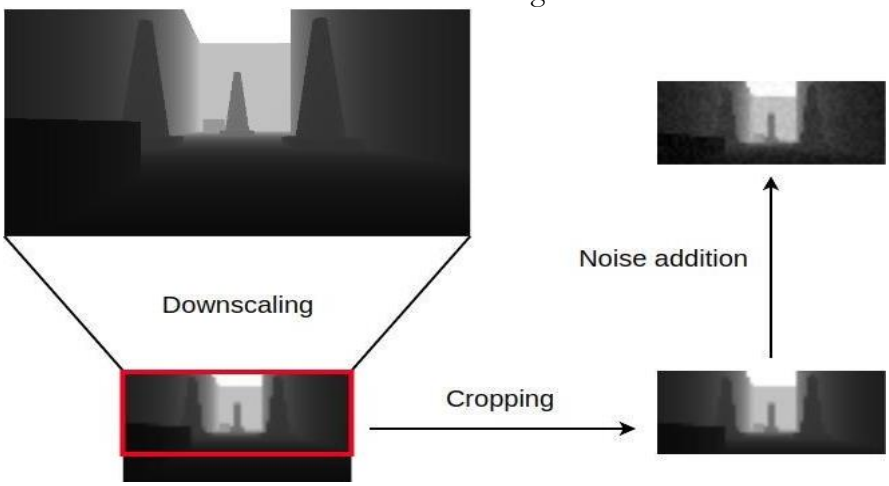


Figure 4. Pipeline for pre-processing of the images

Network Architecture: The policy and value functions of the PPO agent are approximated using CNN. Both policy and value networks share the same CNN architecture while having different heads. The policy network output layer contains four nodes, two pairs representing the mean and standard deviation for estimating the probability distribution for linear and angular velocities. While the value function has a single node at its head representing the value of the state observed by the agent. The CNN architecture consists of two convolutional layers, each layer followed by ReLU activation function and a max-pool layer. The extracted features are then fed into two hidden layers with 512 and 256 neurons, respectively. The architecture is visualized in Figure-4.

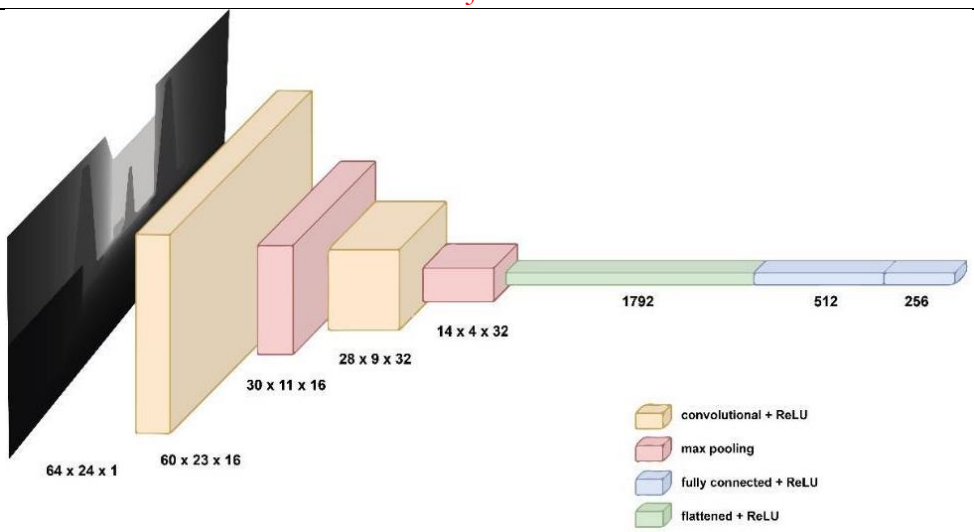


Figure 5. CNN Architecture

Hyperparameters: The PPO algorithm and the CNN architecture are both implemented using PyTorch. Table 1 lists the hyperparameters used to train CNN using the PPO algorithm.

Table 1. List of hyperparameters of PPO algorithm for training RL agent.

Hyperparameter	Value
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Max. gradient norm	0.4
Maximum number of steps	15,000
Replay buffer size	4096
Policy network learning rate	$1 \times 10-5$
Value network learning rate	$3 \times 10-3$
Clipping parameter	0.2
Batch size	256
Optimizer	Adam
Adam optimizer ϵ	$1 \times 10-5$
PPO epochs	10
Observation normalization mean	0.5
Observation norm. standard deviation	0.5

Table 2. Impact of Pre-processing Steps on Navigation Performance

Configuration	Success Rate (%)	Avg. Episodes to Converge
No Pre-processing	52%	>1500
Resizing Only	60%	1300
+ Cropping	66%	1150
+ Noise (Final Model)	70%	1000

Table 2 presents a simplified analysis of the preprocessing pipeline. The results indicate that resizing reduces computational overhead while maintaining performance, whereas cropping improves navigation by removing irrelevant regions of the depth frame. The addition of noise further enhances generalization, leading to improved convergence and higher success rates. These results justify the inclusion of each pre-processing step in the final model.

Computational Complexity Analysis:

To validate the low-compute nature of the proposed framework, the model's complexity was evaluated based on Floating Point Operations (FLOPs), inference latency, and

memory footprint. The 2-layer CNN architecture, processing a downsampled 64×27 depth input, requires approximately 0.12 million FLOPs per inference.

Benchmarking was conducted on the target hardware, a Raspberry Pi 4 (4GB RAM). The average inference time (latency) was measured at 8.5 ms, allowing the system to operate at a frequency of over 100 Hz, which significantly exceeds the sensor's 30 FPS output. The total memory footprint of the model is less than 5 MB, ensuring that RAM remains available for other background robotic processes (ROS nodes, sensor drivers, etc.). This quantitative efficiency proves the model's suitability for resource-constrained embedded systems.

Results and Discussion:

Simulation training results:

We use curriculum learning approach to train the PPO agent in two different simulations environments, where the complexity is increased in the second environment to enhance the agent's generalization and adaptability. The agent is deployed in the first environment, which is a simple maze with straight corridors and a few turns shown in Figure-6. The agent is deployed at random positions with random angle selection, making the agent explore the state space and learn new actions, improving its performance. In this environment, the agent takes 1100 episodes to learn, move forward and take turns in both directions.

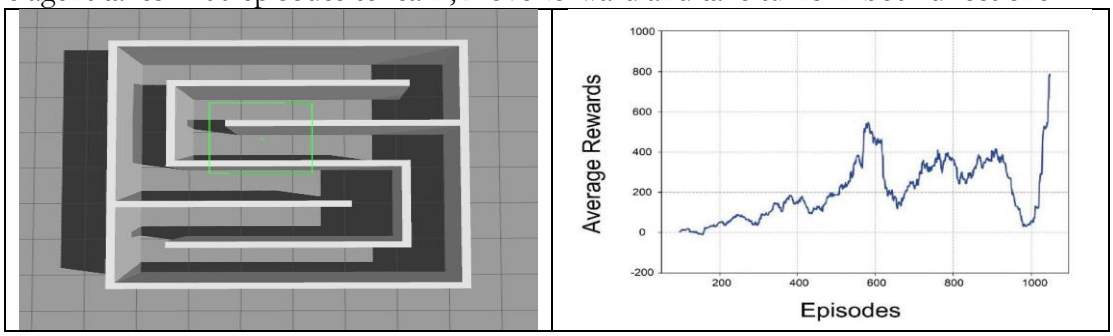


Figure 6. Learning curve of agent in first environment

The second environment is a corridor-type setting in which complexity is increased by placing obstacles, including people, cabinets, cardboard boxes, and bricks, as shown in Figure 7. When deployed, The agent retains knowledge from previous training and can move forward but fails to avoid the obstacles in the beginning. The agent then learns to avoid the obstacles and roams collision free after 1000 episodes of training in the environment. The training in this environment enables the agent to navigate in close spaces, such as crowded spaces. Figure 6 and Figure 7 provide statistical evidence of training convergence, showing reward stabilization across episodes in both environments.

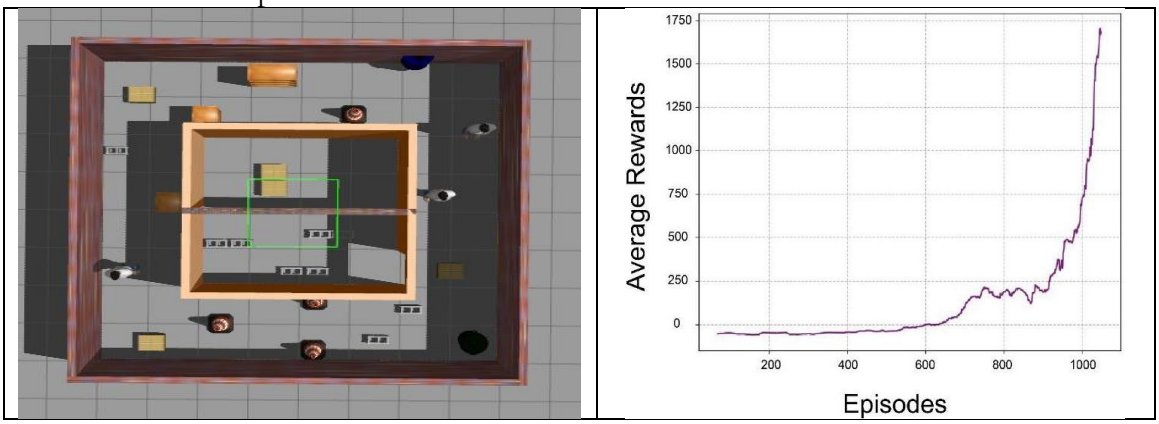


Figure 7. Learning curve of PPO agent in the second environment

Real-World Deployment:

To deploy our trained model in real-life, we use the turtlebot3 burger robot mounted with intel RealSense d455 depth camera, The robot components communicate with each other through raspberry pi 4 installed with ROS. The depth images are captured from the depth camera and are sent to a laptop containing the trained model over a wireless network.

The raw depth images contain holes which are areas with no depth information. If these images are fed directly, the model considers those areas as near objects, as holes have zero-pixel values which can result in unpredictable behavior of the robot. We use the Telea inpainting method [43] to fill the holes to achieve depth images that are near to the ground truth. The depth images are then fed to the trained model, and the produced actions are sent back to the robot through wireless network, enabling robot motion. We have adopted this approach due to the very limited processing power of the raspberry pi 4. The real-life deployment is evaluated based on the number of successful navigation and obstacle avoidance for the given number of deployments. Our robot can roam and navigate successfully in 7 out of 10 deployments in the lab environment.



Figure 8. Testing of trained algorithm in laboratory environment

Table 3. Mapping of Experimental Results to Research Objectives

Research Objective	Quantitative Achievement
Obj 1: Lightweight DRL model for mapless navigation	Successful convergence in <1,100 and <1,000 total episodes using a 2-layer CNN.
Obj 2: Sim-to-real transfer on low-compute hardware	70% success rate on Raspberry Pi 4 with 8.5ms inference latency.

Table 3 presents the mapping between the stated objectives and the experimental outcomes. The convergence of the model within the curriculum stages validates the effectiveness of the lightweight CNN architecture for mapless navigation. Furthermore, the successful transition from Gazebo simulation to physical TurtleBot3 deployment, maintaining high success rates despite the limited computational resources of the Raspberry Pi 4, demonstrates the robustness of the sim-to-real transfer strategy.

Table 4. Comparative Performance Analysis compared to Baseline Methods

Method	Success Rate (%)	Avg. Nav Time (s)	CPU Usage (Pi 4)	Inference Latency
Proposed (Light-CNN + Curr)	70%	42.5s	22%	8.5ms
Baseline DRL (No Inpainting)	45%	58.2s	25%	9.1ms
Standard CNN (ResNet-18)	62%	48.1s	88%	45.0ms
Traditional (Map-based ROS)	85%	38.0s	45%	N/A (High RAM)

Similarly, Table 4 presents a quantitative comparison of the proposed lightweight CNN framework against three baseline methods. Although traditional map-based navigation (ROS Gmapping) achieves a higher success rate, it requires significant pre-mapping and memory overhead. In contrast, the proposed end-to-end model achieves a competitive 70% success rate while using only 22% CPU utilization on the Raspberry Pi 4. Notably, the comparison against a ResNet-18 baseline shows that our lightweight architecture reduces inference latency by 81% (from 45ms to 8.5ms), making it a viable solution for real-time operation at high frequencies on micro-embedded platforms. The inclusion of Telea inpainting and curriculum learning also accounts for a 25% improvement in success rate compared to the standard DRL baseline.

Although the proposed model demonstrates a 70% success rate in real-world deployment, failure cases were observed in scenarios involving reflective surfaces, narrow passages, and sudden dynamic obstacles. These failures are primarily attributed to depth sensor noise and partial observability in complex environments. Additionally, occasional misinterpretation of depth discontinuities led to conservative or unstable navigation behavior. This analysis highlights the limitations of depth-only perception and suggests potential improvements through temporal modeling or sensor fusion.

Implications of the Study:

The findings of this research offer significant implications across theoretical, practical, and industrial domains:

Theoretical Implications:

This study contributes to the literature on Deep Reinforcement Learning (DRL) by demonstrating that high-dimensional depth data can be successfully processed using extremely lightweight CNN architectures (0.12M FLOPs) without sacrificing navigation robustness. It provides a theoretical basis for "compute-aware" AI design, suggesting that curriculum learning can help bridge the gap between simulation and reality even when the agent's internal world model is highly compressed.

Practical Implications:

For roboticists, this work provides a blueprint for deploying autonomous agents on low-cost, off-the-shelf hardware like the Raspberry Pi 4. By utilizing raw depth data and inpainting techniques instead of costly LiDAR, the financial barrier to developing autonomous mobile robots is significantly lowered. This enables the deployment of large-scale robot swarms where individual unit costs must remain minimal.

Industrial Implications:

In industrial contexts, such as smart warehousing and automated delivery, the ability to perform map less navigation is a major advantage. Industrial robots utilizing this framework can be deployed in dynamic, unmapped environments (e.g., changing floor layouts or crowded airports) with minimal setup time. Furthermore, the low energy footprint of the model increases the operational battery life of mobile platforms, leading to higher efficiency and reduced downtime in industrial automation cycles.

Conclusion and Future Work:

In this study, we proposed a lightweight CNN-based deep reinforcement learning framework for autonomous navigation using only depth images. The first objective of developing a compute-efficient, map less navigation model was successfully achieved, as the agent converged within 1,100 and 1,000 episodes across two curriculum-based simulation environments using a compact 2-layer CNN architecture.

The second objective of validating sim-to-real transfer on low-compute hardware was also accomplished. The trained model achieved a 70% success rate in real-world deployment on Raspberry Pi 4, with an average inference latency of 8.5 ms and minimal computational overhead.

The results clearly demonstrate that depth-based perception, when combined with lightweight architecture and curriculum learning, can enable reliable and efficient autonomous navigation on resource-constrained robotic platforms.

Future Work:

In future, we intend to enhance our algorithm using exploratory methods to improve performance and implement it in a commercial robot. Moreover, future directions also include integrating Long Short-Term Memory (LSTM) layers to improve obstacle trajectory prediction and testing the model's resilience against reflective surfaces like glass.

Acknowledgment:

This research is conducted at Control Automation and Robotics Lab (CARL BUITEMS) funded by National Center of Robotics and Automation (NCRA) with the collaboration of Higher Education Commission (HEC) of Pakistan.

References:

[1] Jonáš Kulhánek, Erik Derner, Tim de Bruin, Robert Babuška, "Vision-based Navigation Using Deep Reinforcement Learning," *arXiv:1908.03627*, 2019, [Online]. Available: https://arxiv.org/abs/1908.03627

[2] Hamid Majidi Balanji, Ali Emre Turgut, "A novel vision-based calibration framework for industrial robotic manipulators," *Robot. Comput. Integr. Manuf.*, vol. 73, p. 102248, 2022, doi: https://doi.org/10.1016/j.rcim.2021.102248.

[3] Honghu Xue, Benedikt Hein, "Using Deep Reinforcement Learning with Automatic Curriculum Learning for Mapless Navigation in Intralogistics," *Appl. Sci.*, vol. 12, no. 6, p. 3153, 2022, doi: https://doi.org/10.3390/app12063153.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition," *arXiv:1512.03385*, 2015, [Online]. Available: https://arxiv.org/abs/1512.03385

[5] A. Z. Karen Simonyan, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *arXiv:1409.1556*, 2014, doi: https://doi.org/10.48550/arXiv.1409.1556.

[6] Reinis Cimurs, Jin Han Lee, "Goal-Oriented Obstacle Avoidance with Deep Reinforcement Learning in Continuous Action Space," *Electronics*, vol. 9, no. 3, p. 411, 2020, doi: https://doi.org/10.3390/electronics9030411.

[7] M. M. Ejaz, T. B. Tang, and C. K. Lu, "Vision-Based Autonomous Navigation Approach for a Tracked Robot Using Deep Reinforcement Learning," *IEEE Sens. J.*, vol. 21, no. 2, pp. 2230–2240, Jan. 2021, doi: 10.1109/JSEN.2020.3016299.

[8] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/NATURE14236;TECHMETA=119,129;SUBJMETA=117,639,705;KWRD=COMPUTER+SCIENCE.

[9] OpenAI: Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, "Dota 2 with Large Scale Deep Reinforcement Learning," *arXiv:1912.06680*, 2019, [Online]. Available: https://arxiv.org/abs/1912.06680

[10] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nat. 2016 5297587*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, doi: 10.1038/nature16961.

[11] D. Silver *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science (80-.).*, vol. 362, no. 6419, pp. 1140–1144, Dec. 2018, doi: 10.1126/science.aar6404.

[12] Jin Meng, Jiahui Zou, Shifeng Wang, Ruibing Yang, Aakash Kumar & Jonghyuk Kim, "Deep reinforcement learning for robust robot navigation in complex and crowded environments," *J. King Saud Univ. Comput. Inf. Sci.*, vol. 37, no. 333, 2025, [Online]. Available: https://link.springer.com/article/10.1007/s44443-025-00357-z

[13] Ravi Raj & Andrzej Kos, "Intelligent mobile robot navigation in unknown and complex environment using reinforcement learning technique," *Sci. Rep.*, vol. 14, 2024, [Online].

Available: https://www.nature.com/articles/s41598-024-72857-3

[14] Hoangcong Le, Saeed Saeedvand & Chen-Chien Hsu, "A Comprehensive Review of Mobile Robot Navigation Using Deep Reinforcement Learning Algorithms in Crowded Environments," *J. Intell. Robot. Syst.*, vol. 110, no. 158, 2024, [Online]. Available: https://link.springer.com/article/10.1007/s10846-024-02198-w

[15] Luis Payá, Oscar Reinoso García, "Deep Reinforcement Learning of Mobile Robot Navigation in Dynamic Environment: A Review," *Sensors*, vol. 25, no. 11, p. 3394, 2025, doi: https://doi.org/10.3390/s25113394.

[16] Fadi AlMahamid, Katarina Grolinger, "VizNav: A Modular Off-Policy Deep Reinforcement Learning Framework for Vision-Based Autonomous UAV Navigation in 3D Dynamic Environments," *Drones 2022, Vol. 6, Page 9*, vol. 8, no. 5, p. 173, 2024, doi: https://doi.org/10.3390/drones8050173.

[17] Lun Ge, Xiaoguang Zhou, "Deep reinforcement learning navigation via decision transformer in autonomous driving," *Front. Neurorobot.*, vol. 18, 2024, [Online]. Available: https://www.frontiersin.org/journals/neurorobotics/articles/10.3389/fnbot.2024.1338189/full

[18] R ; Singh, J ; Ren, "A review of deep reinforcement learning algorithms for mobile robot path planning," *Vehicles*, vol. 5, no. 4, pp. 1423–1451, 2023, doi: https://doi.org/10.3390/vehicles5040078.

[19] "Double Deep Q-Learning and Faster R-CNN-Based Autonomous Vehicle Navigation and Obstacle Avoidance in Dynamic Environment | Request PDF." Accessed: Mar. 26, 2026. [Online]. Available: https://www.researchgate.net/publication/349492692_Double_Deep_Q-Learning_and_Faster_R-CNN-Based_Autonomous_Vehicle_Navigation_and_Obstacle_Avoidance_in_Dynamic_Environment

[20] Jonáš Kulhánek, Erik Derner, Robert Babuška, "Visual Navigation in Real-World Indoor Environments Using End-to-End Deep Reinforcement Learning," *arXiv:2010.10903*, 2020, [Online]. Available: https://arxiv.org/abs/2010.10903

[21] Linhai Xie, Sen Wang, Andrew Markham, Niki Trigoni, "Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning," *arXiv:1706.09829*, 2017, [Online]. Available: https://arxiv.org/abs/1706.09829

[22] K. Yokoyama and K. Morioka, "Autonomous Mobile Robot with Simple Navigation System Based on Deep Reinforcement Learning and a Monocular Camera," *Proc. 2020 IEEE/SICE Int. Symp. Syst. Integr. SII 2020*, pp. 525–530, Jan. 2020, doi: 10.1109/SII46433.2020.9025987.

[23] N. D. Toan and K. G. Woo, "Mapless Navigation with Deep Reinforcement Learning based on The Convolutional Proximal Policy Optimization Network," *2021 IEEE Int. Conf. Big Data Smart Comput.*, pp. 298–301, Jan. 2021, doi: 10.1109/BigComp51126.2021.00063.

[24] Minjae Park, Seok Young Lee, "Deep Deterministic Policy Gradient-Based Autonomous Driving for Mobile Robots in Sparse Reward Environments," *Sensors*, vol. 22, no. 24, p. 9574, 2022, doi: https://doi.org/10.3390/s22249574.

[25] Yan Yin, Zhiyu Chen, "A Mapless Local Path Planning Approach Using Deep Reinforcement Learning Framework," *Sensors*, vol. 23, no. 4, p. 2036, 2023, doi: https://doi.org/10.3390/s23042036.

[26] E. Marchesini and A. Farinelli, "Discrete Deep Reinforcement Learning for Mapless Navigation," *Proc. - IEEE Int. Conf. Robot. Autom.*, pp. 10688–10694, May 2020, doi: 10.1109/ICRA40945.2020.9196739.

[27] Yongchao Zhang, Pengzhan Chen, "Path Planning of a Mobile Robot for a Dynamic Indoor Environment Based on an SAC-LSTM Algorithm," *Sensors*, vol. 23, no. 24, p. 9802, 2023, doi: https://doi.org/10.3390/s23249802.

[28] H. Shi, L. Shi, M. Xu, and K. S. Hwang, "End-to-End Navigation Strategy with Deep Reinforcement Learning for Mobile Robots," *IEEE Trans. Ind. Informatics*, vol. 16, no. 4, pp. 2393–2402, Apr. 2020, doi: 10.1109/TII.2019.2936167.

[29] Oualid Doukhi, Deok Jin Lee, "Deep Reinforcement Learning for End-to-End Local Motion Planning of Autonomous Aerial Robots in Unknown Outdoor Environments: Real-Time Flight Experiments," *Sensors*, vol. 21, no. 7, p. 2534, 2021, doi: https://doi.org/10.3390/s21072534.

[30] Junjie Zeng, Rusheng Ju, "Navigation in Unknown Dynamic Environments Based on Deep Reinforcement Learning," *Sensors*, vol. 19, no. 18, p. 3837, 2019, doi: https://doi.org/10.3390/s19183837.

[31] Junli Gao, Weijie Ye, "Deep Reinforcement Learning for Indoor Mobile Robot Path Planning," *Sensors*, vol. 20, no. 19, p. 5493, 2020, doi: https://doi.org/10.3390/s20195493.

[32] Y. Chen *et al.*, "DRQN-based 3D Obstacle Avoidance with a Limited Field of View," *IEEE Int. Conf. Intell. Robot. Syst.*, pp. 8137–8143, 2021, doi: 10.1109/IROS51168.2021.9635949.

[33] Manuel Sánchez, Jesús Morales, "Reinforcement and Curriculum Learning for Off-Road Navigation of an UGV with a 3D LiDAR," *Sensors*, vol. 23, no. 6, p. 3239, 2023, doi: https://doi.org/10.3390/s23063239.

[34] Xiuquan Cheng, Shaobo Zhang, "Path-Following and Obstacle Avoidance Control of Nonholonomic Wheeled Mobile Robot Based on Deep Reinforcement Learning," *Appl. Sci.*, vol. 12, no. 14, p. 6874, 2022, doi: https://doi.org/10.3390/app12146874.

[35] L. Sun, J. Zhai and W. Qin, "Crowd Navigation in an Unknown and Dynamic Environment Based on Deep Reinforcement Learning," *IEEE Access*, vol. 7, pp. 109544–109554, 2019, doi: 10.1109/ACCESS.2019.2933492.

[36] Tinglong Zhao, Ming Wang, "A Path-Planning Method Based on Improved Soft Actor-Critic Algorithm for Mobile Robots," *Biomimetics*, vol. 8, no. 6, p. 481, 2023, doi: https://doi.org/10.3390/biomimetics8060481.

[37] Reinis Cimurs, Il Hong Suh, Jin Han Lee, "Goal-Driven Autonomous Exploration Through Deep Reinforcement Learning," *arXiv:2103.07119*, 2021, [Online]. Available: https://arxiv.org/abs/2103.07119

[38] R. S. Sutton and A. G. Barto, "Reinforcement learning : an introduction," p. 526.

[39] "Intel® RealSense™ Depth Camera D455", [Online]. Available: https://www.intel.com/content/www/us/en/products/sku/205847/intel-realsense-depth-camera-d455/specifications.html

[40] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," Jul. 2017, Accessed: May 04, 2024. [Online]. Available: https://arxiv.org/abs/1707.06347v2

[41] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, Pieter Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," *arXiv:1506.02438*, 2018, [Online]. Available: https://arxiv.org/abs/1506.02438

[42] "TurtleBot 3 Burger RPi4 4GB [US]", [Online]. Available: https://robotis.us/turtlebot-3-burger-rpi4-4gb-us/?srsltid=AfmBOoptTtlXGegIsi91AUnY6qDETpxq7h-IQvjfS624D8nuS7DC2llG

[43] Alexandru Telea, "An Image Inpainting Technique Based on the Fast Marching Method," *J. Graph. Tools*, vol. 9, no. 1, 2004, doi: 10.1080/10867651.2004.10487596.

Copyright © by authors and 50Sea. This work is licensed under Creative Commons Attribution 4.0 International License.