Wall-following Robot Using Deep Reinforcement Learning

The Alphabet-following Robot

Beatriz Silva[1,2], Lara Sousa[1,2]

[1] FEUP - Faculty of Engineering, University of Porto
[2] FCUP - Faculty of Sciences, University of Porto
larasousapf@gmail.com, up202109782@up.pt, up202107955@fc.up.pt

Abstract. This project explores the development of a wall-following robot controller using deep reinforcement learning (DRL) to determine which alphabet letters create the most challenging mazes for robotic navigation. The motivation behind using alphabet letters as mazes stems from their diverse shapes and complexities, which provide a wide range of scenarios for testing the robot's capabilities. By navigating through these varied forms, we aim to assess the robustness and performance of our robot in real-world environments.

In this study, each letter of the alphabet is represented as a unique maze. The complexity of these mazes arises from the different geometrical shapes and curves inherent in each letter. For example, letters like 'A' and 'M' may present sharp turns and intersecting lines, while letters like 'O' and 'S' offer smooth, continuous curves. By systematically testing the robot's ability to follow the walls of these letters, we can identify which shapes are more challenging and thus evaluate the effectiveness of our DRL-based controller.

The robot's LiDAR readings serve as inputs to the DRL system, which then outputs actions to control the robot's movement. We implement a DRL-based controller, experimenting with various DRL algorithms such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG) and Twin-Delayed Deep Deterministic Policy Gradient(TD3). Python libraries such as Gymnasium and Stable-Baselines3 are utilized for simulation and model access, providing a robust environment for training and evaluation.

The primary goal of this project is to demonstrate the effectiveness of deep reinforcement learning in autonomous robot control tasks, particularly in scenarios like wall-following. By leveraging advanced machine learning techniques and testing the robot in diverse and complex mazes, we aim to achieve improved performance and potentially surpass the capabilities of conventional control approaches.

Keywords: wall-following robot, deep reinforcement learning, LiDAR sensors, alphabet maze navigation, robot control systems, proximal policy optimization (PPO), deep deterministic policy gradient (DDPG), twin-delayed deep deterministic policy gradient(TD3), autonomous robotics,

gymnasium, Stable-Baselines3, machine learning in robotics, robot performance evaluation, complex navigation scenarios, reinforcement learning algorithms.

1 Introduction

In the field of autonomous robotics, the ability for a robot to navigate complex environments reliably and efficiently is paramount. Wall-following robots, which use sensors to maintain a specified distance from walls while navigating, are commonly used in various applications such as inspection, cleaning, and surveillance. These robots must effectively handle a wide range of environmental conditions and obstacles, making their control systems critical to their success.

Traditional control methods, such as Proportional (P) controllers, often fall short in dynamic and unpredictable environments due to their limited adaptability. Recent advancements in machine learning, particularly in deep reinforcement learning (DRL), offer promising alternatives by enabling robots to learn and adapt to complex scenarios through experience. This project aims to explore the potential of DRL in enhancing the performance and robustness of wall-following robots.

The primary goal of this project is to develop and implement a DRL-based controller for a wall-following robot. By leveraging DRL techniques, we aim to improve the robot's ability to navigate through various configurations and obstacles. The project involves:

- Implementing a DRL-based controller.
- Experimenting with various DRL algorithms such as Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Twin Delayed Deep Deterministic Policy Gradient (TD3).
- Utilizing the Webots simulator and the e-puck robot model for training and testing.
- Tracking training on Tensorboard.

This study investigates how the shapes of different alphabet letters, used as mazes, affect the performance of a wall-following robot. The hypothesis is that the complexity and form of certain letters will present more significant challenges to the robot, thereby providing insights into the robustness and adaptability of the DRL-based controller.

Hypothesis:

- **Complex Shapes:** Letters with more complex geometrical shapes, such as 'M', 'W', or 'K', which involve sharp turns and intersections, will be more challenging for the robot to navigate compared to simpler shapes like 'O' or 'I'.
- **Curved Forms:** Letters with continuous curves, such as 'B' or 'S', will test the robot's ability to maintain a smooth path and could highlight the controller's proficiency in handling gradual changes in direction.
- **Performance Metrics:** By evaluating the robot's performance across these different letter-shaped mazes, we can identify which shapes lead

to higher success rates and smoother navigation, thus validating the robustness of the DRL-based controller.

The outcomes of these tests will demonstrate the effectiveness of DRL in enabling the robot to adapt to diverse and complex navigation tasks, providing valuable insights for real-world applications.

Section 2 reviews existing methods and approaches in wall-following robot control. Section 3 details the implementation of the DRL-based controller, including the algorithms used and the simulation setup. Additionally, not only presents the findings from the simulations, comparing the performance of different DRL algorithms, but also analyzes the results and discusses the implications and potential applications of the findings. Lastly, section 4 summarizes the project and suggests directions for future research.

2 Literature Review

This section reviews the relevant literature on traditional control methods, the application of deep reinforcement learning (DRL) in robotics, and the use of simulation environments for robotic control system development.

Traditional Control Methods: Wall-following robots have traditionally relied on classical control methods, such as Proportional (P), Proportional-Integral (PI), and Proportional-Integral-Derivative (PID) controllers. These controllers are designed to maintain a robot's position relative to a wall using sensor inputs. For example, a P controller adjusts the robot's steering angle based on the difference between the desired and actual distances from the wall. While these methods are simple and effective in controlled environments, they often lack the adaptability required for more complex and dynamic scenarios.

Proportional Controllers: Proportional controllers are widely used in robotic control due to their simplicity and ease of implementation. They adjust the robot's movement based on the proportional difference between the setpoint and the current measurement. Studies such as by Bolles and Fischler [1] have demonstrated the effectiveness of P controllers in basic wall-following tasks. However, these controllers can be sensitive to sensor noise and may struggle in environments with sudden changes in wall configurations. Above there is a representation of each proportional controller (Figure 1 [2], Figure 2 [3]and Figure 3 [4]).

Proportional-Integral-Derivative Controllers: PID controllers, which incorporate integral and derivative components in addition to the proportional term, offer improved performance over P controllers by considering the cumulative error and the rate of change of the error. Research by Franklin Franklin et al. [5] has shown that PID controllers can enhance stability and response time in wall-following robots. Nonetheless, tuning the parameters of PID controllers can be challenging, particularly in complex and unpredictable environments.

Deep Reinforcement Learning in Robotics: Recent advancements in machine learning, particularly in DRL [6], have opened new avenues for developing more robust and adaptive robotic control systems. DRL combines rein-

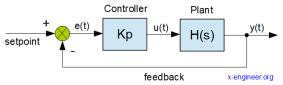

Fig. 1. P controller - Block diagram illustrating the working principle of a Proportional Controller

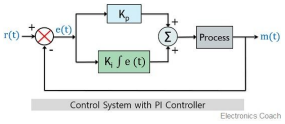

Fig. 2. PI controller - Diagram showing the structure and functioning of a Proportional Integral Controller

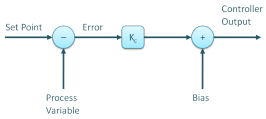

Fig. 3. PID controller - Schematic representation of a Proportional Controller

forcement learning with deep neural networks, enabling robots to learn optimal control policies through interaction with their environment.

Proximal Policy Optimization (PPO): PPO, proposed by Schulman et al. [7], is a policy gradient method that ensures stable training by limiting the policy update step size. PPO has been successfully applied to continuous control tasks in robotics, including wall-following scenarios. PPO's robustness and ease of implementation make it a popular choice for DRL applications.

Deep Deterministic Policy Gradient (DDPG): DDPG, introduced by Lillicrap et al. [8], extends DQN to continuous action spaces by using actor-critic architecture. DDPG has been employed in robotic tasks requiring precise control, such as robotic arm manipulation and autonomous driving. Its ability to handle continuous actions makes it suitable for complex navigation tasks.

Twin Delayed Deep Deterministic Policy Gradient (TD3): TD3, developed by Fujimoto et al. [9], addresses the overestimation bias in DDPG by using twin critics and delayed policy updates. TD3 has shown superior performance in continuous control tasks, including robot navigation and manipulation. The enhancements introduced by TD3 improve training stability and policy learning.

Simulation Environments for Robotic Control: Simulation environments play a crucial role in developing and testing robotic control systems. Webots (Figure 4 [10]) an open-source robot simulator, provides a versatile platform for simulating various robotic scenarios, including wall-following tasks.

Webots Simulator: Webots allows for detailed modeling of robots and their environments, supporting a range of sensors and actuators. Studies such as [11] have utilized Webots for simulating and validating DRL-based control algorithms in robotics. The simulator's ability to replicate real-world conditions makes it an invaluable tool for developing robust control systems.

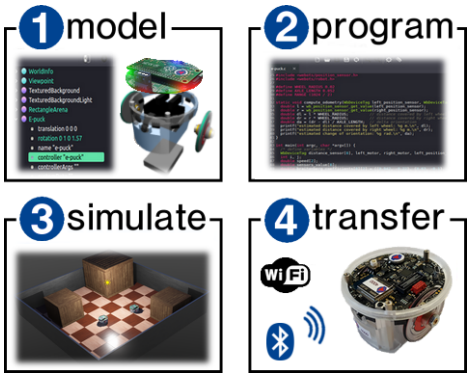

Fig. 4. Webots Simulator Functionalities - Guide on running external robot controllers, as detailed by Cyberbotics

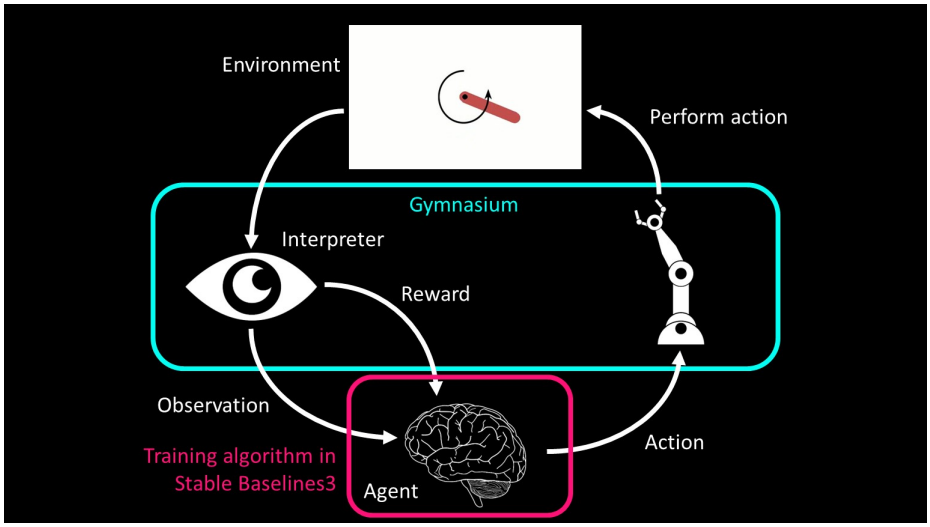

Fig. 5. Gymnasium and RL - Introduction to reinforcement learning using Gymnasium and Stable Baselines3, presented by Digi-Key

Gymnasium and Stable-Baselines3: Gymnasium and Stable-Baselines3 are Python libraries that facilitate the implementation and training of DRL algorithms. Gymnasium provides a standardized interface for reinforcement learning environments (Figure 5[12]), while Stable-Baselines3 offers a collection of state-of-the-art DRL algorithms [13]. These libraries have been widely adopted in the research community for developing and benchmarking DRL-based robotic control systems.

The literature highlights the evolution of control methods for wall-following robots from traditional PID controllers to advanced DRL algorithms. While traditional controllers are effective in simple environments, DRL approaches offer greater adaptability and robustness in complex scenarios. Simulation environments like Webots, coupled with libraries such as Gymnasium and Stable-Baselines3, provide powerful tools for developing and testing these advanced control systems. This project builds on this foundation by applying DRL to wall-following robots, using alphabet letters as mazes to evaluate the robustness and performance of the developed controllers.

3 Algorithm, Train and Test Strategy

3.1 Proposed Approach to Solve the Hypothesis

Description of the Setup

Simulator Used: Webots will serve as the primary simulator for the training and testing of the wall-following robot controller. This open-source simulation platform is chosen for its advanced capabilities in modeling and simulating robotic systems. The e-puck robot model (Figure 6 [14]) within Webots will be employed for our simulations due to its well-documented performance and suitability for educational and research purposes. Webots allows for detailed simulation of sensors and actuators, providing a realistic environment for testing DRL algorithms.

Robot's Sensors and Actuators

- LiDAR Sensors: The e-puck robot is equipped with LiDAR sensors, which provide precise distance measurements to surrounding obstacles. These sensors enable the robot to create a real-time map of its environment and to follow walls accurately by maintaining a consistent distance from them. The LiDAR data serves as the primary input for the deep reinforcement learning (DRL) system, enabling the robot to learn effective navigation strategies.
- Wheels and Actuators: The robot's movement is controlled by its wheels, which are actuated to adjust both linear and angular velocities. This dual control mechanism allows the robot to navigate through tight spaces and around sharp corners effectively.
- Touch Sensors: In addition to LiDAR, the e-puck robot utilizes touch sensors to detect collisions. These sensors are crucial for preventing damage to the robot and ensuring safety during navigation. The touch sensor

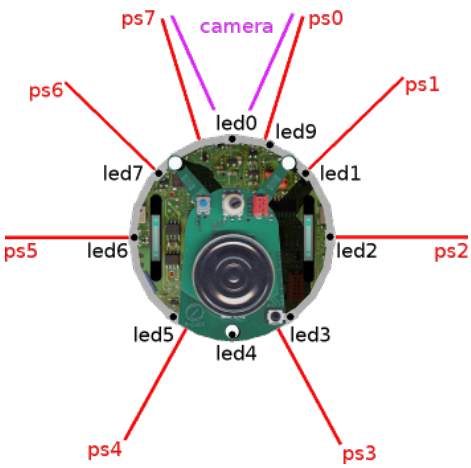

Fig. 6. e-puck robot - Detailed guide on the e-puck robot, provided by Cyberbotics

feedback will be integrated into the DRL system to refine the robot's collision avoidance strategies.

Environment/World Parameters

- Wall Configuration: The Webots environment will feature walls arranged in various configurations to simulate real-world navigation challenges. These configurations include straight corridors, sharp corners, and open spaces that the robot must navigate through.
- Training Environments: Three distinct training environments will be used to teach the robot different navigation strategies:
  - Acute Angle Environment: This environment will feature walls that meet at acute angles. It is designed to test the robot's ability to handle sharp turns and narrow passages.
  - Straight Angle Environment: In this setting, walls will form right angles, providing a more straightforward but still challenging environment for the robot to navigate.
  - Curved Environment: This environment will include a large, nearly circular curve. This setup is intended to train the robot in handling gradual, continuous turns, which require different navigation strategies compared to angular turns.
- Testing Environments: For testing, the robot will be evaluated in environments shaped like various letters of the alphabet. Each letter's unique shape presents a different navigation challenge, allowing us to assess the robot's adaptability and performance across diverse and complex scenarios. Letters with straight lines, curves, and intersections will be included to provide a comprehensive evaluation of the robot's capabilities.

Algorithms of Deep Reinforcement Learning (DRL)

Fig. 7. Acute Angle Environment

Fig. 8. Straight Angle Environment

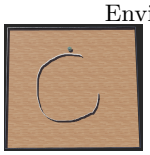

Fig. 9. Curved Environment

- Twin Delayed Deep Deterministic Policy Gradient (TD3): an advanced DRL algorithm that addresses some of the limitations found in its predecessor, Deep Deterministic Policy Gradient (DDPG). TD3 enhances the stability and performance of DDPG by introducing three key innovations:
  - Twin Critic Networks: TD3 employs two critic networks to estimate the value function, rather than one. During the training process, the algorithm selects the smaller value from the two critics to update the policy. This method helps to mitigate the overestimation bias that can occur in single critic networks, leading to more accurate and stable value estimates.
  - Delayed Policy Updates: The policy (actor network) in TD3 is updated less frequently than the value function (critic networks). Specifically, the actor is updated every two iterations of the critic updates. This delay in policy updates reduces the variance and instability of the learning process, allowing the critic networks to provide more reliable feedback before the policy is adjusted.
  - Target Policy Smoothing: To further stabilize learning, TD3 adds noise to the target action during the critic update, a technique known as target policy smoothing. This approach prevents the policy from overfitting to narrow peaks in the value function, promoting smoother and more robust policy updates.

These innovations [9] make TD3 particularly well-suited for continuous action spaces, where precise and stable control is critical, such as in robotic manipulation and navigation tasks.

- Proximal Policy Optimization (PPO):stands out for its adept balance between sample efficiency and simplicity of implementation. It innovatively retains the advantages of Trust Region Policy Optimization (TRPO)—a main type of policy gradient method—while being significantly simpler and more computationally economical. PPO achieves this

through a clipped objective function, which constrains the magnitude of policy updates. Such clipping ensures that policy modifications are not overly drastic, thereby promoting stability throughout the training phase.

  * Clipped Objective Function: PPO uses a surrogate objective function that includes a clipping mechanism. This function limits the ratio of the new policy to the old policy within a specified range, preventing large, destabilizing updates. The clipping ensures that policy updates do not deviate excessively from the previous policy, maintaining stability during training.
  * Multiple Epochs of Mini-batch Updates: PPO allows for multiple epochs of updates on mini-batches of data, improving sample efficiency. This means that the algorithm can make better use of collected data by optimizing the policy over several iterations, enhancing learning without requiring more interactions with the environment.
  * Simple Implementation: PPO is significantly simpler to implement. It achieves similar performance levels without the need for second-order optimization techniques, making it accessible for a wider range of applications and researchers.

These features [7] make PPO a robust and versatile choice for various reinforcement learning tasks, providing a good balance between performance, stability, and ease of implementation.

- Deep Deterministic Policy Gradient (DDPG): a crucial algorithm in the field of robotics for its ability to handle continuous action spaces with high precision. It integrates several key techniques to ensure stable and efficient learning:
  * Actor-Critic Methods: DDPG utilizes an actor-critic architecture where the actor network determines the optimal action for a given state, and the critic network evaluates the action by estimating the Q-value. This combination allows the algorithm to learn both the policy and the value function simultaneously, leading to more effective policy updates.
  * Experience Replay: To improve learning stability and efficiency, DDPG employs an experience replay buffer. This buffer stores past experiences and samples them randomly during training, breaking the temporal correlations between consecutive samples. This technique ensures more stable learning and better convergence properties.
  * Target Networks: DDPG uses target networks for both the actor and critic. These target networks are slowly updated versions of the original networks, providing consistent targets for the Q-value updates. This reduces the risk of divergence and instability during training.

DDPG's ability to calculate the optimal action directly for any given state ensures consistent and accurate control, making it particularly effective for tasks requiring precise and continuous adjustments, such as robotic arm manipula-

tion and autonomous vehicle navigation. This method significantly enhances robotic autonomy and task performance, highlighting its importance in advanced robotics applications [8].

Proposed experiments to evaluate the quality of the approach

In our proposed experiments, we use several key metrics to evaluate the performance and robustness of the wall-following robot trained with different deep reinforcement learning (DRL) models. These metrics are essential for understanding the effectiveness of our approach in various test environments ([15] and [16]). Below, we explain each of the metrics used:

- Total Collisions: The number of times the robot collides with obstacles during navigation.
  - Purpose: This metric measures the robot's ability to avoid obstacles and maintain a safe distance from walls. A lower number of collisions indicates better obstacle avoidance and safer navigation.
- Total Out of Limits: The number of instances where the robot exceeds the predefined boundaries of the environment.
  - Purpose: This metric evaluates the robot's ability to stay within the designated operational area. Exceeding these limits can indicate a failure in maintaining the desired path or handling turns and edges effectively.
- Mean Time Without Collisions: The average steps the robot navigates without colliding with obstacles.
  - Purpose: This metric reflects the robot's efficiency in continuous, collision-free navigation. Higher values suggest that the robot can maintain smooth and uninterrupted movement, which is crucial for practical applications.
- Learning Curve on TensorBoard: A visual representation of the robot's learning progress over time, typically showing the reward or loss against training iterations.
  - Purpose: The learning curve helps monitor the training process, providing insights into how quickly and effectively the model is learning. It indicates improvements in performance and highlights any potential issues such as overfitting or underfitting.

Metric Importance

- Total Collisions: Indicates the robot's proficiency in obstacle avoidance.
- Total Out of Limits: Assesses the robot's path adherence and control.
- Mean Time Without Collisions: Measures overall navigation smoothness and safety.
- Learning Curve on TensorBoard: Tracks the model's learning progress and performance over time.

3.2 Code, Robot and Environment Setup

Code Structure: The implementation of the wall-following robot controller is organized into five main Python files, each serving a specific purpose:

- "epuck wall following env": Defines the custom Gym environment for the e-puck robot, encapsulating the simulation details and interaction mechanisms.
- "train ddpg": Contains the script for training the robot using the Deep Deterministic Policy Gradient (DDPG) algorithm.
- "train td3": Contains the script for training the robot using the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm.
- "train ppo": Contains the script for training the robot using the Proximal Policy Optimization (PPO) algorithm.

Custom Gym Environment The custom Gym environment is tailored to simulate the wall-following behavior of the e-puck robot. Key components are described below.

- Imports and Logging Setup: The necessary libraries are imported, including Gym, NumPy, logging, and matplotlib for plotting. Logging is configured to track the execution process and output relevant information.
- Training Environments:The training environments are designed with various configurations (right angle environment, acute angle environment, curved environment) to challenge the robot's navigation capabilities.

Reward Calculation Function The reward calculation function is a critical component of the reinforcement learning process, providing the agent with feedback based on its performance in the environment [17; 18] . In the context of the wall-following robot, the reward function is designed to encourage behaviors that keep the robot at an optimal distance from the wall, promote safe navigation by avoiding collisions, and incentivize smooth, continuous movement.

- Objective of the Reward Function: The primary goal of the reward function is to shape the robot's behavior in a way that it learns to follow walls efficiently and safely. The reward function achieves this by:
  - Maintaining an Optimal Distance: Encouraging the robot to stay within a specific distance range from the wall.
  - Penalizing Deviations: Discouraging the robot from getting too close or too far from the wall.
  - Promoting Collision-Free Navigation: Providing additional rewards for time spent navigating without collisions.
- Components of the Reward Function
  - Optimal Distance Maintenance (Figure 10): The robot is rewarded for maintaining a distance between $\boldsymbol{a}$ meters and $\boldsymbol{b}$ meters from the wall. If the distance ($\boldsymbol{distMin}$) falls within this range, the robot receives a positive reward.This component ensures the robot learns to stay close enough to follow the wall but far enough to avoid collisions.
  - Penalty for Deviation: If the robot's distance from the wall is less than $\boldsymbol{a}$ meters or greater than $\boldsymbol{b}$ meters, a penalty is applied. A significant negative reward is given for deviations, guiding the robot to correct its path and return to the optimal range. This helps the

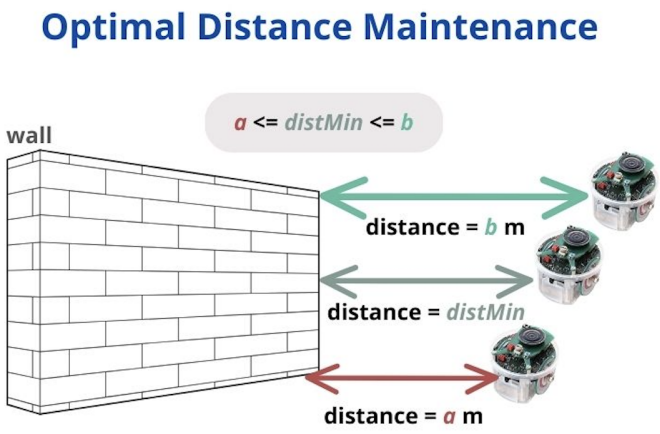

Fig. 10. Maintaining an Optimal Distance

robot learn to avoid getting too close (risking collision) or too far (losing track of the wall).

* Collision-Free Time: The robot receives a small, incremental reward for each time step it avoids collisions.This is calculated by multiplying the time spent without collision by a small factor (0.01).This component encourages the robot to maintain smooth and safe navigation over extended periods.

The design of the reward function is crucial for the learning process. By carefully balancing rewards and penalties, the function guides the robot to develop behaviors that ensure effective and safe wall-following. The incremental reward for collision-free time encourages the robot to prioritize smooth and continuous navigation, which is essential for real-world applications where safety and efficiency are paramount.

In summary, the reward calculation function in this wall-following robot setup is a finely-tuned mechanism that incentivizes the robot to maintain an optimal distance from the wall, avoid collisions, and navigate smoothly over time. This structured approach ensures that the robot learns to perform its task effectively through positive and negative reinforcement.

WallFollowingEnv Class The WallFollowingEnv class is a custom environment extending the Gym environment class, specifically designed to simulate the wall-following behavior of the e-puck robot. This class is essential for initializing the robot's sensors, actuators, and simulation parameters, providing a comprehensive framework for training the robot using deep reinforcement learning (DRL) algorithms.

Sensors - The robot is equipped with several sensors to perceive its environment accurately:

- LiDAR (Light Detection and Ranging): Measures distances to surrounding obstacles, providing a detailed 360-degree view of the environment. This is crucial for the robot to maintain an optimal distance from walls.
- GPS (Global Positioning System): Provides precise positioning information, enabling the robot to track its location within the simulation environment.
- Compass: Offers orientation data, allowing the robot to maintain and adjust its heading accurately.
- Touch Sensor: Detects collisions with obstacles, which is vital for safety and for the robot to learn collision avoidance.

Actuators - The robot's movement is controlled through its wheel actuators:

- Left and Right Wheel Motors: These actuators control the linear and angular velocities of the robot. By adjusting the speeds of the left and right wheels independently, the robot can move forward, backward, and turn smoothly.

Observation and Action Spaces - The observation and action spaces define the state and action representations for the robot within the environment:

- **Observation Space -** The state space comprises normalized LiDAR readings, which provide a comprehensive view of the robot's surroundings. The observation space is defined as a continuous space with values ranging from 0 to 1, representing the normalized distances.
- **Action Space -** The action space consists of continuous values representing the linear and angular velocities. This space is defined to allow the robot to take actions that involve moving forward or backward and turning left or right.

Command and Step Functions

- **Command Function (cmd)** The cmd function is responsible for setting the velocities of the robot's wheels based on the provided action. It ensures the robot's movements are accurately executed by adjusting the wheel velocities to achieve the desired linear and angular velocities. Process:

- Calculate the linear and angular velocities based on the input action.
- Determine the individual wheel velocities.
- Clamp the wheel velocities to ensure they remain within the robot's operational limits.
- Set the velocities of the left and right wheel motors to control the robot's movement.

Output: The function updates the robot's wheel velocities, enabling it to move accordingly.

- **Step Function (step)** The step function integrates the core loop of the environment, executing the provided action, updating the state, calculating the reward, and checking for termination conditions. Process:

- Smooth the input action to ensure gradual transitions in the robot's movement.

- Execute the cmd function to apply the action.
- Update the environment state by stepping the simulation.
- Retrieve observations from sensors, such as LiDAR, GPS, and touch sensors.
- Calculate the reward based on the robot's current state and performance metrics.
- Check termination conditions to determine if the episode should end.

Output: The function returns the new state (observation), the reward, a done flag indicating if the episode has terminated, and additional info if necessary.

Conditions to terminate The environment needs clear termination conditions to determine when an episode ends. These conditions ensure the robot learns to avoid failures and undesirable states.

- Collision Detection: The episode terminates if the touch sensor detects a collision.
- Out of Bounds: The episode ends if the robot's position exceeds predefined boundaries or if the minimum distance exceeds a defined value.
- Maximum Steps: The episode concludes after a maximum number of steps to prevent excessively long runs.

Reset Function The reset function reinitializes the environment to a starting state, preparing it for a new episode. This involves resetting sensors, actuators, and the robot's position. Process:

- Log the end of the current episode, including performance metrics.
- Reinitialize sensor and actuator states.
- Move the robot to its initial starting position and orientation.
- Clear episode-specific variables, such as cumulative rewards and step counters.

Output: The function returns the initial observation to begin the new episode.

"get obs" Function The "get obs" function is critical for collecting and normalizing sensor data from the robot's environment. This function primarily processes LiDAR data, converting it into a format suitable for the agent to use as observations.

Purpose: gather LiDAR sensor data and normalize the sensor data for use as input to the reinforcement learning agent.

Process:

- Collect Raw LiDAR Data: The function retrieves the raw range image from the LiDAR sensor.
- Check for Data Availability: If the raw observation is None, indicating no data is available, the function returns an array of zeros matching the observation space dimensions.
- Convert and Normalize Data: If data is available, the function converts the raw LiDAR data into a NumPy array.
- Infinite values, which can occur if the LiDAR detects no obstacles within range, are replaced with the maximum LiDAR range.
- Any NaN values in the data are replaced with zeros to avoid processing errors.

- The data is then normalized by dividing each distance measurement by the maximum LiDAR range, scaling all values to be between 0 and 1.
- The normalized observation data is returned for use by the reinforcement learning agent.

The train files are going to be discuss in the next subsection (3.3. Train and Test Strategy).

3.3 Train and Test Strategy

For the training process, we established several environments with distinct configurations to challenge the robot (Table 1):

Environment	**Initial Position**	**x_max x_min**	**y_max y_min**
Right Angle	(0.50, 0.76)	1.55, 0.15	1.55, 0.15
Acute Angle	(0.56, 0.42)	1.80, 0.25	1.35, 0.10
Curve 2	(1.04, 1.62)	1.80, 0.25	1.75, 0.15

Table 1. Training Environments with Initial Position and Boundaries

In the training process, the reward function uses specific values to guide the robot in maintaining an optimal distance from the wall:

Optimal Distance (0.026 to 0.044 meters): This range is chosen based on the robot's sensor capabilities and the desired safe distance for wall-following.

Maintaining this distance ensures that the robot can effectively follow the wall without risking collisions or losing track of its path.

- Reward: **+5.0** if the robot stays within this range.
- Penalty for Being Too Close (**less than 0.026** meters):
- Penalty: **-7.0** to discourage proximity that risks collisions.
- Penalty for Being Too Far (**greater than 0.044** meters):
- Penalty: **-7.0** to ensure the robot doesn't stray too far from the wall.

These values help the robot learn to stay at an optimal distance from the wall, balancing safety and effective wall-following behavior.

We employed three DRL models: TD3, PPO, and DDPG. The initial training phase consisted of two iterations of 10,000 timesteps each. After each iteration, the models were saved, allowing for incremental training. This method resulted in three training folders for each DRL model, named sequentially (e.g., TDO3, $\text{TDO3}_i, i_(2, 3)$.

This approach allowed us to refine and adjust models based on early test results, enhancing robustness for subsequent training phases with increased timesteps.

– Iterations: Two per model, each with **10,000 timesteps**.
– Checkpointing: Models saved after each iteration.
– Folders: Created for each model and iteration (e.g., PPO, where PPO has two zip files inside him: 10000 and 20000).

Extended Training:

- Iterations: Two per model, each with **20,000 timesteps**.
- Accumulated Training: Continued training from previous checkpoints, doubling the training timesteps for robust learning.

To evaluate the models, we tested the robot on letter-shaped mazes: **A, B, I, J, M,** and **O** (Figures 11 to 16). These shapes were chosen to represent diverse navigation challenges, such as straight lines, sharp turns, and curves, the environments are in

Fig. 11. 'A' Test Environment

Fig. 12. 'B' Test Environment

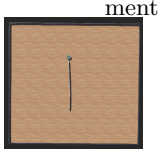

Fig. 13. 'I' Test Environment

Fig. 14. 'J' Test Environment

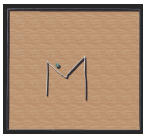

Fig. 15. 'M' Test Environment

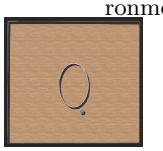

Fig. 16. 'O' Test Environment

Evaluation Metrics:

- Total Collisions: Number of collisions encountered during navigation.
- Out of Bounds Events: Instances where the robot exceeded the predefined boundaries.

– Mean Time Without Collision: Average number of steps taken without a collision.

Tracking and Analysis:

– TensorBoard: Used for real-time tracking of training progress, including time, reward curves, and loss metrics.
– Performance Analysis: Post-training analysis using TensorBoard to visualize the learning curves and performance metrics across different models and iterations.

This evaluation metrics and the tracking and analysis were discussed in more detail in a previous subsection (3.1 Proposed Approach to Solve the Hypothesis).

In the results, we will create a table to compare the performance of the models across different letter-shaped test maps. Following the final testing phase, it will be possible to identify specific areas for improvement and proposed adjustments for future iterations. This iterative process of training, testing, and tuning ensures continuous enhancement of the robot's wall-following capabilities.

3.4 Results

Environment	Round 1	Saved with name	Train Order		
(Webots Walls)	(60K timesteps)	(Folders with zips: 10 000 and 20 000)	PPO	DDPG	TD3
Straight Angle	• PPO_41 e PPO_42; • DDPG_9 e DDPG_10; • TD3_5 e TD3_6;	• PPO_2; • DDPG_2; • TD3_2.	2º	2º	2º
Curved	• PPO_38 e PPO_39; • DDPG_9 e DDPG_10; • TD3_3 e TD3_4;	• PPO; • DDPG_3; • TD3;	1º	3º	1º
Acute Angle	• PPO_43 e PPO_44; • DDPG_7 e DDPG_8; • TD3_7 e TD3_8;	• PPO_3; • DDPG; • TD3_3;	3º	1º	3º

Fig. 17. First Round of Training - 60,000 timesteps for each DRL model. For each environment, the corresponding DRL model names with associated numbers are provided in the second column. This corresponds to the name of the iteration visualized in TensorBoard. The third column lists the names of the folders containing the saved training data. The last three columns represent the order of training.

The **first round of training** began with 2 iterations of 10,000 timesteps each using **PPO**, starting in the **curved environment**. After that, we loaded the trained model from the last iteration and continued training in a different environment, **the acute environment**, repeating the process until we trained in the final environment. In the same round, we performed **2 iterations of 10,000**

timesteps each for **DDPG**, starting in the acute environment. After saving it, we loaded the model to continue training in **the straight environment**, and then repeated the process for the curved environment. For **TD3**, we started in the curved environment. After saving the training progress, we loaded it for another environment, the straight angle, and finally **retrained** from the last iteration of the straight angle to the last environment, the acute angle. In total, we trained with **60,000 timesteps** for each DRL model. To further clarify the first round of training, the entire process is represented in Figure 17 .

All the training sessions and iterations were tracked in TensorBoard. Figures 18 to 20 represent a comparison between the last iteration of all the training sessions for each DRL model.

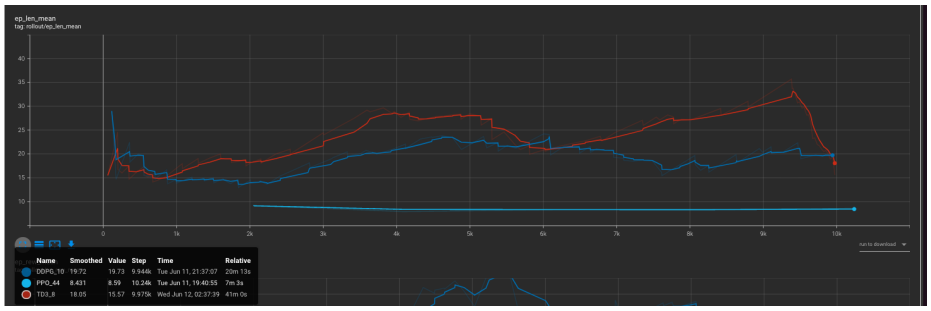

Fig. 18. Graph of "ep len mean" - This graph represents the mean episode length over time for the last train iteration of the three different DRL models

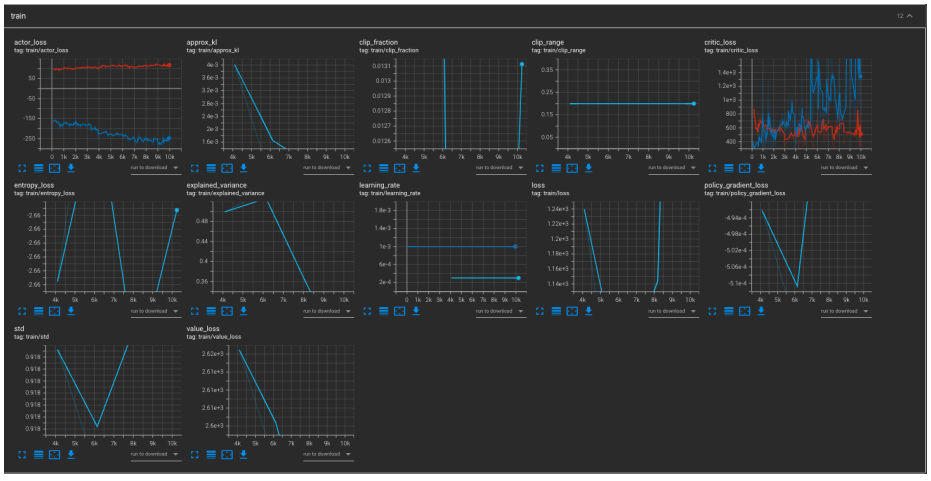

Fig. 19. Multiple training metrics tracked in TensorBoard for the different models.

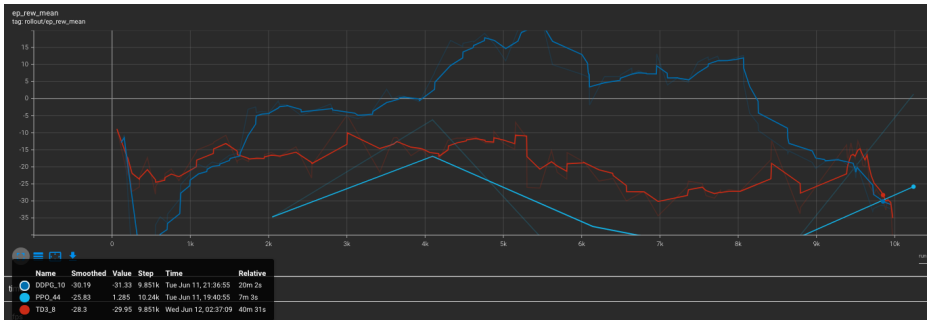

Fig. 20. The mean episode reward ("ep rew mean") over time for the last train iteration of the three different DRL models.

The first graph (Figure 18) represents the **mean episode length over time** for three different DRL models: DDPG, PPO, and TD3. The x-axis indicates the timesteps, while the y-axis indicates the mean episode length. The legend at the bottom identifies each model with a different color:

- **DDPG (Blue):** This model shows a fluctuating episode length initially but stabilizes around a lower episode length as training progresses. The final episode length is approximately 19.72.
- **PPO (Light Blue):** PPO starts and ends at an episode length of approximately 8.431 (none fluctuation).
- **TD3 (Red):** This model shows an upward trend initially but stabilizes at a higher episode length compared to the other models, ending at approximately 18.05.

The second graph (Figure 19) shows **multiple training metrics tracked** in TensorBoard for the different models:

- **actor loss:** Measures the loss of the actor network, indicating how well the policy is performing.
- **approx kl:** Measures the Kullback-Leibler divergence, indicating how the current policy diverges from the previous policy.
- **clip fraction:** The fraction of the updates that were clipped during training.
- **clip range:** The range used for clipping in the PPO algorithm.
- **critic loss:** Measures the loss of the critic network, indicating how well the value function is being approximated.
- **entropy loss:** Measures the loss in entropy, indicating the exploration versus exploitation trade-off.
- **explained variance:** Indicates how much of the variance in the returns is explained by the value function.
- **learning rate:** The learning rate used during training.
- **loss:** General loss metric for the model.
- **policy gradient loss:** Measures the loss due to policy gradient updates.
- **std:** The standard deviation of the returns.

- **value loss:** Measures the loss in value estimation.

The Third graph (Figure 20) represents **the mean episode reward over time** for the three different DRL models. The x-axis indicates the timesteps, while the y-axis indicates the mean episode reward. The legend at the bottom identifies each model with a different color:

- **DDPG (Blue):** Starts with lower rewards, fluctuates, and then shows a gradual increase. It ends with a mean reward of approximately -28.3 .
- **PPO (Light Blue):** PPO shows a similar pattern to DDPG, with initial fluctuations and eventual stabilization. The final mean reward is approximately -25.3.
- **TD3 (Red):** This model shows a significant drop in rewards initially but stabilizes around a lower reward value. The final mean reward is approximately -31.33.

Key interpretations:

The TD3 model shows the lowest final rewards and relatively stable but longer episode lengths. This indicates that **while the agent is able to complete episodes, it is not maximizing rewards effectively**. **The PPO model shows a stable pattern in episode length** but **ends with higher rewards compared to TD3**. This suggests that **PPO may be better at exploring and finding higher reward policies**. **DDGP** also shows a fluctuating pattern similar to PPO but **ends with rewards slightly better than TD3 but worse than PPO**. The episode lengths are also relatively high, indicating that **the agent takes longer to complete episodes**.

In summary, the PPO model seems to perform better in terms of reward optimization, while DDPG and TD3 show more stability in episode length but achieve lower rewards. PPO is also the fastest to stabilize, making it efficient in terms of training time.

After the first training round, it's time to test the models on the letter environments. The last iteration of training for each DRL model was tested over 100 episodes. The entire testing process was recorded and is shown in the figure 21.

Analysis and Discussion of the Table (Figure 21) The table presents the results of testing the last iteration of training for each DRL model (PPO, DDPG, and TD3) across various letter-shaped environments, using the **same initial position** as the one defined on Webots. The metrics include total collisions, total out-of-limits occurrences, and mean time without collisions (in steps).

Letter A:

- PPO: Total Collisions = 14; Total Out of Limits = 86; Mean Time Without Collisions = 119.98 steps.
- DDPG:Total Collisions = 14; Total Out of Limits = 86; Mean Time Without Collisions = 112.9 steps.
- TD3: Total Collisions = 21; Total Out of Limits = 79; Mean Time Without Collisions = 101.11 steps.
- Discussion: For Letter A, both PPO and DDPG performed similarly in terms of collisions and out-of-limits occurrences. However, PPO slightly outper-

Ronda 1 de Treino – Env Initial Position == Webots Initial Position

Test Map	Models	Metrics		
		Total Collisitions	Total Out Of Limits	Mean Time Without Collisions (steps)
Letter A	PPO	14	86	119.98
	DDPG	14	86	112.9
	TD3	21	79	101.11
Letter B	PPO	39	61	52.02
	DDPG	49	51	36.57
	TD3	40	60	49.48
Letter I	PPO	20	80	42.6
	DDPG	20	80	42.6
	TD3	20	80	42.6
Letter J	PPO	42	58	56.37
	DDPG	50	40	53.53
	TD3	40	60	53.53
Letter M	PPO	56	44	22.66
	DDPG	56	44	27.66
	TD3	56	44	22.66
Letter O	PPO	11	89	76.13
	DDPG	11	89	76.13
	TD3	11	89	76.13

Fig. 21. table highlights the strengths and weaknesses of each DRL model across different letter-shaped environments, for the round 1 of train. It used the same initial position as the one defined on Webots

formed DDPG in terms of mean time without collisions. TD3 had more collisions but fewer out-of-limits occurrences compared to the other two models.

Letter B:

- PPO:Total Collisions = 39; Total Out of Limits = 61; Mean Time Without Collisions: 52.02 steps.
- DDPG:Total Collisions = 49; Total Out of Limits = 51; Mean Time Without Collisions = 36.57 steps.
- TD3: Total Collisions = 40; Total Out of Limits = 60; Mean Time Without Collisions = 49.48 steps.
- Discussion: In the Letter B environment, PPO had the fewest collisions and the highest mean time without collisions, indicating better performance. DDPG had the highest number of collisions and the lowest mean time without collisions, suggesting poorer performance in this environment. TD3 had a balanced performance, with fewer collisions than DDPG but more than PPO.

Letter I:

- PPO:Total Collisions = 20 ; Total Out of Limits = 80; Mean Time Without Collisions: 42.6 steps.
- DDPG:Total Collisions = 20; Total Out of Limits: 80; Mean Time Without Collisions: 42.6 steps.
- TD3:Total Collisions: 20;Total Out of Limits: 80;Mean Time Without Collisions: 42.6 steps.
- Discussion: All three models performed identically in the Letter I environment, suggesting that this environment may not provide sufficient differentiation between the models' capabilities.

Letter J:

- PPO: Total Collisions = 42; Total Out of Limits = 58; Mean Time Without Collisions = 56.37 steps.
- DDPG: Total Collisions = 50; Total Out of Limits = 40; Mean Time Without Collisions = 53.53 steps.
- TD3: Total Collisions = 40; Total Out of Limits = 60; Mean Time Without Collisions = 53.53 steps.
- Discussion: For Letter J, PPO had the fewest out-of-limits occurrences and the highest mean time without collisions. TD3 had the fewest collisions but more out-of-limits occurrences. DDPG had the highest number of collisions but fewer out-of-limits occurrences, indicating a trade-off between these two metrics.

Letter M:

- PPO: Total Collisions = 56; Total Out of Limits = 44; Mean Time Without Collisions = 22.66 steps.
- DDPG: Total Collisions = 56; Total Out of Limits = 44; Mean Time Without Collisions = 27.66 steps.
- TD3: Total Collisions = 56; Total Out of Limits = 44; Mean Time Without Collisions = 22.66 steps.

– Discussion: All three models had identical results in terms of total collisions and total out-of-limits occurrences. However, DDPG had a slightly higher mean time without collisions.

Letter O:

– PPO: Total Collisions = 11; Total Out of Limits = 89; Mean Time Without Collisions = 76.13 steps.
– DDPG: Total Collisions = 11; Total Out of Limits = 89; Mean Time Without Collisions = 76.13 steps.
– TD3: Total Collisions = 11; Total Out of Limits = 89; Mean Time Without Collisions = 76.13 steps.
– Discussion: All three models performed identically in the Letter O environment, similar to their performance in Letter I. This again suggests that the environment may not provide sufficient differentiation between the models' capabilities.

General Observations:

– PPO tends to have a balanced performance with fewer collisions in most environments and higher mean time without collisions.
– DDPG, generally, has more collisions but performs better in terms of out-of-limits occurrences.
– TD3 shows balanced performance but does not consistently outperform the other models.

Overall, the table highlights the strengths and weaknesses of each DRL model across different letter-shaped environments. It indicates that while PPO may be more stable, DDPG and TD3 have specific environments where they perform better. The choice of model may depend on the specific requirements and constraints of the environment in question.

Alphabet-Following Observations:

– The PPO model performed best in the Letter A environment, with only 14 collisions, 86 out-of-limits occurrences, and a mean time without collisions of 119.98 steps. This indicates that PPO was able to navigate the Letter A environment more effectively than the others. The worst performance for PPO was observed in the Letter M environment, where it had 56 collisions, 44 out-of-limits occurrences, and a mean time without collisions of only 22.66 steps. This suggests that the Letter M environment was particularly challenging for PPO.
– DDPG's best performance was also in the Letter A environment, with the same metrics as PPO (14 collisions, 86 out-of-limits occurrences, and a mean time without collisions of 112.9 steps). This suggests that DDPG was similarly effective in this environment. DDPG performed worst in the Letter B environment, with 49 collisions, 51 out-of-limits occurrences, and a mean time without collisions of only 36.57 steps. This indicates significant difficulty navigating the Letter B environment.
– TD3 performed best in the Letter I environment, with 20 collisions, 80 out-of-limits occurrences, and a mean time without collisions of 42.6 steps. However, since this was the same performance as the other models in this environment,

it might not be the best indicator of TD3's strengths. The worst performance for TD3 was in the Letter M environment, similar to the other models, with 56 collisions, 44 out-of-limits occurrences, and a mean time without collisions of 22.66 steps.

The Best and Worst Letters Overall

- The best overall performance across all models was observed in the Letter A environment, where both PPO and DDPG had the fewest collisions and higher mean times without collisions. This suggests that the Letter A environment was easier to navigate for these models.
- The Letter M environment proved to be the most challenging across all models, with the highest number of collisions and the lowest mean times without collisions.

Expected Performance The performance in the Letter A environment being the best was somewhat expected as the simpler structure of Letter A might provide fewer navigational challenges. Conversely, the poor performance in the Letter M environment aligns with expectations, as the complex structure of Letter M could introduce more obstacles and decision points, increasing the difficulty for the models.

The main purpose is to follow the wall. Driven by curiosity, we also wanted to test the robustness of the models when the initial position is not provided, in other words, when the initial position of environment is different from the one implemented in Webots and training code. The results are shown in the table below (Figure 22).

This table (Figura 22) is similar to the previous table (Figura 21), but this particular difference of testing with different initial position.

Analysis of the table Letter A:

- PPO: Total Collisions = 42; Total Out of Limits = 58; Mean Time Without Collisions = 58.15 steps.
- DDPG: Total Collisions = 42; Total Out of Limits = 58; Mean Time Without Collisions = 58.12 steps.
- TD3: Total Collisions = 42; Total Out of Limits = 58; Mean Time Without Collisions = 54.07 steps.
- Discussion: For Letter A, all three models (PPO, DDPG, and TD3) performed identically in terms of total collisions and out-of-limits occurrences, with slight variations in the mean time without collisions. This suggests that the Letter A environment does not differentiate the models significantly under the given conditions.

Letter B:

- PPO: Total Collisions = 54; Total Out of Limits = 46; Mean Time Without Collisions = 32.39 steps.
- DDPG: Total Collisions = 60; Total Out of Limits = 40; Mean Time Without Collisions = 24.37 steps.
- TD3: Total Collisions = 60; Total Out of Limits = 40; Mean Time Without Collisions = 24.37 steps.

Ronda 1 de Treino – Env Initial Position != Webots Initial Position

		Metrics		
Test Map	Models	Total Collisitions	Total Out Of Limits	Mean Time Without Collisions (steps)
Letter A	PPO	42	58	58.15
	DDPG	42	58	58.12
	TD3	42	58	54.07
Letter B	PPO	54	46	32.39
	DDPG	60	40	24.37
	TD3	60	40	24.37
Letter I	PPO	28	72	36.84
	DDPG	28	72	36.84
	TD3	31	69	35.92
Letter J	PPO	50	50	60.14
	DDPG	50	50	58.88
	TD3	50	50	58.88
Letter M	PPO	53	57	23.8
	DDPG	52	48	26.43
	TD3	52	48	26.43
Letter O	PPO	38	62	64.24
	DDPG	38	62	64.24
	TD3	38	82	64.24

Fig. 22. table highlights the strengths and weaknesses of each DRL model across different letter-shaped environments, for the round 1 of train. It used a different initial position as the one defined on Webots

- Discussion: In the Letter B environment, PPO had fewer collisions and a higher mean time without collisions compared to DDPG and TD3. This indicates that PPO performed better in this environment. DDPG and TD3 had identical performance metrics, suggesting they faced similar challenges.

Letter I:

- PPO: Total Collisions = 28; Total Out of Limits = 72; Mean Time Without Collisions = 36.84 steps.
- DDPG: Total Collisions = 28; Total Out of Limits = 72; Mean Time Without Collisions = 36.84 steps.
- TD3: Total Collisions = 31; Total Out of Limits = 69; Mean Time Without Collisions = 35.92 steps.
- Discussion: For Letter I, PPO and DDPG performed identically, while TD3 had slightly more collisions and fewer out-of-limits occurrences. The mean time without collisions was similar across all models, indicating comparable performance in this environment.
-

Letter J:

- PPO: Total Collisions = 50; Total Out of Limits = 50; Mean Time Without Collisions = 60.14 steps.
- DDPG: Total Collisions = 50; Total Out of Limits = 50; Mean Time Without Collisions = 58.88 steps.
- TD3: Total Collisions = 50; Total Out of Limits = 50; Mean Time Without Collisions = 58.88 steps.
- Discussion: In the Letter J environment, all three models performed identically in terms of total collisions and out-of-limits occurrences. The mean time without collisions was slightly higher for PPO, suggesting marginally better performance.

Letter M:

- PPO: Total Collisions = 53; Total Out of Limits = 57; Mean Time Without Collisions = 23.8 steps.
- DDPG: Total Collisions = 52; Total Out of Limits = 48; Mean Time Without Collisions = 26.43 steps.
- TD3: Total Collisions = 52; Total Out of Limits = 48; Mean Time Without Collisions = 26.43 steps.
- Discussion: For Letter M, DDPG and TD3 performed identically, with slightly fewer collisions and out-of-limits occurrences compared to PPO. The mean time without collisions was also higher for DDPG and TD3, indicating better performance in this environment.

Letter O:

- PPO: Total Collisions = 38; Total Out of Limits = 62; Mean Time Without Collisions = 64.24 steps.
- DDPG: Total Collisions = 38; Total Out of Limits = 62; Mean Time Without Collisions = 64.24 steps.
- TD3: Total Collisions = 38; Total Out of Limits = 82; Mean Time Without Collisions = 64.24 steps.

– Discussion: In the Letter O environment, PPO and DDPG performed identically, while TD3 had more out-of-limits occurrences but the same mean time without collisions. This suggests that TD3 might have faced more challenges staying within the limits.

General Observations:

– The best overall performance was observed in the Letter O environment for all models, where they had fewer collisions and higher mean times without collisions. The worst overall performance was in the Letter M environment, where all models had the highest number of collisions and the lowest mean times without collisions.
– Consistency: PPO generally showed more consistent performance across different environments, whereas DDPG and TD3 had similar performance metrics in most cases but faced challenges in more complex environments.

Alphabet-following observations:

– The Best Performances:
  • PPO, DDPG: Letter O, A.
  • TD3: Letter O.
–
– The Worst Performances:
  • PPO: Letter M;
  • DDPG: Letter B;
  • TD3: Letter B,A.

Comparison between round 1 tables

– In both tables, Letter O showed the best overall performance for all models, with fewer collisions and higher mean times without collisions.Letter M consistently presented significant challenges across both tables, with the highest number of collisions and lowest mean times without collisions.
– Letter A: In the previous table, PPO and DDPG had fewer collisions and higher mean times without collisions, which is consistent with the current results. TD3, however, showed more variability in performance between the two tables.
– Letter B: DDPG and TD3 faced more challenges in the current table compared to the previous one, indicating that the initial position can significantly impact the model's performance.
– Letter I and Letter J: The results for these environments were relatively consistent across both tables, showing similar levels of collisions and mean times without collisions.

We created a video (**https://files.fm/u/a2ddaxth4e**) that compares each letter and model, showing the difference between using the same initial position and not.

The results indicate that while the models can handle simpler environments like Letter O, they struggle with more complex ones like Letter M. This suggests a need for further tuning and training to improve robustness in varied and complex environments.

Therefore, it was thought that the problem could be arising from the values of distance ***a*** and distance ***b***, which constrain ***distMin***. Consequently, these values were changed to the following:

Optimal Distance (0.021 to 0.056 meters): This range is chosen based on the robot's sensor capabilities and the desired safe distance for wall-following.

- Reward: **+5.0** if the robot stays within this range.
- Penalty for Being Too Close (**less than 0.021** meters):
- Penalty: **-7.0** to discourage proximity that risks collisions.
- Penalty for Being Too Far (**greater than 0.056** meters):
- Penalty: **-7.0** to ensure the robot doesn't stray too far from the wall.

In addition, it was considered relevant to increase the training timesteps. Thus, we will perform 2 iterations with 20,000 timesteps each (40k timesteps per environment and 120k per DRL model), continuing from the saved training in round 1.

In **round 2**, we repeated the same procedure as in round 1 but with different distance values and by doubling the timesteps per iteration. Similarly, a table was created to record the environments, the model names in TensorBoard, the names under which the folders were saved, and the order of training (Figure 23).

Environment	Round 2	Saved with name	Train Order		
(Webots Walls)	(120K timesteps)	(Folders with zips: 10 000 and 20 000)	PPO	DDPG	TD3
Straight Angle	• DDPG_17 e DDPG_18; • TD3_9 e TD3_10 • PPO_48 e PPO_49.	• PPO_4_5_2; • DDPG_4_5_3; • TD3_4_5.	2°	3°	1°
Curved	• DDPG_15 e DDPG_16; • TD3_13 e TD3_14; • PPO_46 e PPO_47.	• PPO_4_5; • DDPG_4_5_2; • TD3_4_5_3;	1°	2°	3°
Acute Angle	• TD3_11 e TD3_12; • DDPG_13 e DDPG_14; • PPO_50 e PPO_51.	• PPO_4_5_3; • DDPG_4_5; • TD3_4_5_2;	3°	1°	2°

Fig. 23. First Round of Training - 120,000 timesteps for each DRL model. For each environment, the corresponding DRL model names with associated numbers are provided in the second column. This corresponds to the name of the iteration visualized in TensorBoard. The third column lists the names of the folders containing the saved training data. The last three columns represent the order of training.

The graphs from **TensorBoard round 2**(Figures 24 to 26) show various metrics for three DRL models (PPO, DDPG, and TD3) during the training process, corresponding to their last iteration (40000 timesteps).

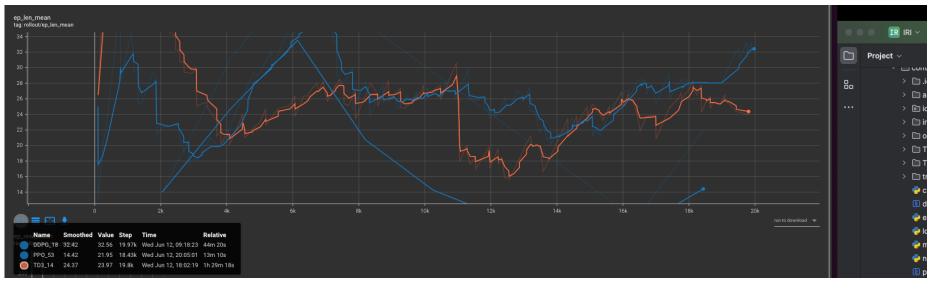

Fig. 24. "ep lean mean - Episode Length Mean : This graph represents the mean episode length over time for the last train iteration of the three different DRL models"

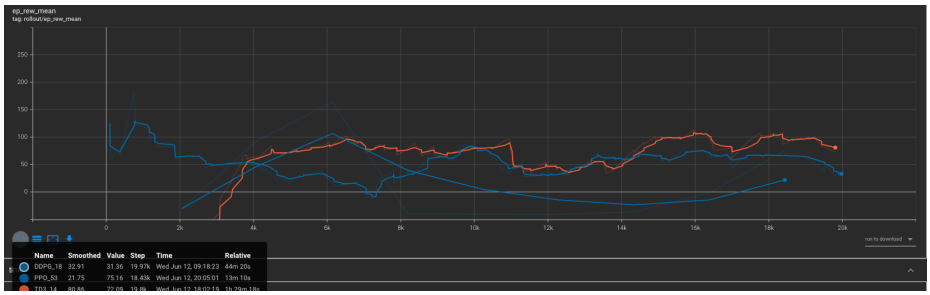

Fig. 25. "ep rew mean - Episode Reward Mean : The mean episode reward over time for the last train iteration of the three different DRL models"

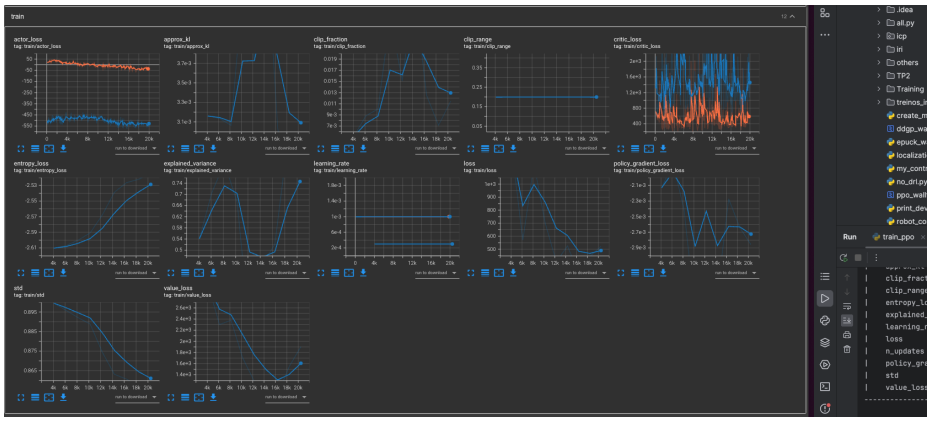

Fig. 26. Train curves : Multiple training metrics tracked in TensorBoard for the different models

The first graph (Figure 24)

- **PPO (Dark Blue):** Initially shows fluctuations but stabilizes around a value of approximately 14 towards the end of training.
- **DDPG (Light Blue):** Starts with high variability and shows a gradual increase in episode length, ending around 32.56.
- **TD3 (Orange):** Shows an overall increasing trend with high fluctuations, ending around 23.97.
- **Discussion:** The mean episode length indicates how long the robot can operate without hitting a failure condition. DDPG shows a significant increase in episode length over time, suggesting it learns to navigate the environment more effectively. PPO and TD3 also improve but with more fluctuations, indicating variability in their learning processes.

The second graph (Figure 25)

- **PPO (Dark Blue):**Shows steady improvement in rewards ate the beginning, but ends withe negative reward.
- **DDPG (Light Blue):**Starts with high rewards, but shows fluctuations and eventually stabilizes around 31.36.
- **TD3 (Orange):** Shows an increasing trend with fluctuations, ending around 80.86.
- **Discussion:** The episode reward mean indicates the effectiveness of the policy in maximizing rewards. TD3 shows the highest final rewards, indicating effective learning. DDPG also shows significant improvement, whereas PPO, despite improvements, shows more variability and lower final rewards compared to the other models.

The third Graph (Figure 26) This graph shows various training metrics tracked during the training process. **"actor loss":** TD3 shows a relatively stable actor loss compared to DDPG, indicating steady policy updates. **"approx kl":** Shows the Kullback-Leibler divergence, indicating how much the policy is changing. PPO show some fluctuations, with higher variability. **"clip fraction" and "clip range":** Indicate the fraction and range of updates that were clipped during training, essential for stabilizing PPO updates. Values are relatively stable. **"critic loss":** Indicates the loss in value function approximation. TD3 shows more stable values, while DDPG shows high variability. **"entropy loss":** Measures the exploration versus exploitation trade-off. PPO shows an increase, indicating exploration. **"explained variance":** Indicates how much of the return variance is explained by the value function. PPO shows good variance. **"learning rate:** Shows the learning rate schedule. No model changes over time, indicating learning rate adjustments during training. **"loss:"** General loss metric. PPO shows a decreasing trend, indicating improving model performance. **"policy gradient loss":** Indicates the loss due to policy gradient updates. Over the time this loss decreased in PPO. **"std":** Shows the standard deviation of returns. PPO shows decreasing std, indicating more consistent performance. **value loss:** Indicates the loss in value function approximation. PPO shows a decreasing trend, indicating improving value function approximation.

Overall Discussion The training metrics provide a comprehensive overview of the learning process for each DRL model.

- PPO: Shows consistent improvement across multiple metrics, including episode length, reward, and loss metrics. Despite ending with a negative reward, PPO demonstrates stable training updates and consistent policy improvements. The decrease in standard deviation indicates more consistent performance over time.
- DDPG: Exhibits high variability in training metrics, with significant fluctuations in episode length, reward, and loss. While it shows improvement, DDPG's performance is less stable compared to PPO and TD3. The variability in critic loss and other metrics suggests that DDPG may require further tuning or more stable training environments.
- TD3: Achieves the highest episode rewards and shows effective learning. However, TD3 also demonstrates high fluctuations in episode length and training metrics, indicating periods of instability. The relatively stable actor loss suggests steady policy updates, but the overall variability suggests room for improvement in training consistency.

Summarizing, **the PPO model demonstrates the most consistent and stable performance across various training metrics**, despite some fluctuations in reward outcomes. **DDPG, while showing potential for effective learning**, suffers from high variability and instability during training. **TD3, though achieving the highest rewards**, also exhibits significant fluctuations, indicating a need for more stable training procedures.

Now we are going to test our latest trained DRL model iterations of the training round 2 in the letter-shaped environments and discuss the results. We will present the results in two tables (Figures 27 and 28). Both highlight the strengths and weaknesses of each DRL model across different letter-shaped environments, for the round 2 of train. However, in one hand the first one (Figure 27) uses the same initial position as the one defined on Webots, and in other hand (Figure 28) don't use it. The metrics tracked include total collisions, total out-of-limits occurrences, and mean time without collisions (in steps).

Analysis and Discussion of the Table "Ronda 2 de Treino- Env Initial Position equals to Webots Initial Position" (Figure 27) **Alphabet-Following Observations**

Letter A:

- All Models (PPO, DDPG, TD3): Total Collisions = 21;Total Out of Limits = 79; Mean Time Without Collisions = 108.72 steps.
- Discussion: All models performed identically in this environment, suggesting it was equally navigable for each DRL approach.

Letter B:

- All Models (PPO, DDPG, TD3): Total Collisions = 43; Total Out of Limits = 57; Mean Time Without Collisions = 46.39 steps.
- Discussion: Again, the performance was identical across all models, indicating that this environment presented a consistent challenge for each model.

Ronda 2 de Treino – Env Initial Position == Webots Initial Position

Test Map	Models	Metrics		
		Total Collisitions	Total Out Of Limits	Mean Time Without Collisions (steps)
Letter A	PPO	21	79	108.72
	DDPG	21	79	108.72
	TD3	21	79	108.72
Letter B	PPO	43	57	46.39
	DDPG	43	57	46.39
	TD3	43	57	46.39
Letter I	PPO	21	79	45.99
	DDPG	27	73	36.86
	TD3	21	79	45.99
Letter J	PPO	40	60	60.6
	DDPG	45	55	51.62
	TD3	40	60	60.6
Letter M	PPO	57	43	25.77
	DDPG	57	43	25.77
	TD3	57	43	25.77
Letter O	PPO	15	85	83.93
	DDPG	15	85	83.93
	TD3	15	85	83.93

Fig. 27. table highlights the strengths and weaknesses of each DRL model across different letter-shaped environments, for the round 2 of train. It used the same initial position as the one defined on Webots

Ronda 2 de Treino – Env Initial Position != Webots Initial Position

Test Map	Models	Metrics		
		Total Collisitions	Total Out Of Limits	Mean Time Without Collisions (steps)
Letter A	PPO	37	63	57.58
	DDPG	37	63	57.58
	TD3	39	61	58.19
Letter B	PPO	43	57	48.53
	DDPG	49	51	35.57
	TD3	49	51	35.57
Letter I	PPO	28	72	42.17
	DDPG	23	77	41.14
	TD3	22	78	41.32
Letter J	PPO	35	65	70.42
	DDPG	35	65	70.42
	TD3	30	70	79.74
Letter M	PPO	53	47	25.02
	DDPG	53	47	25.02
	TD3	53	47	25.02
Letter O	PPO	11	89	83.19
	DDPG	11	89	83.19
	TD3	11	89	83.19

Fig. 28. table highlights the strengths and weaknesses of each DRL model across different letter-shaped environments, for the round 2 of train. It used a different initial position from the one defined on Webots

Letter I:

- PPO: Total Collisions = 21; Total Out of Limits = 79; Mean Time Without Collisions = 45.99 steps.
- DDPG: Total Collisions = 27; Total Out of Limits = 73; Mean Time Without Collisions = 36.86 steps.
- TD3: Total Collisions = 21; Total Out of Limits = 79; Mean Time Without Collisions = 45.99 steps.
- Discussion: PPO and TD3 performed identically and better than DDPG, which had more collisions and a shorter mean time without collisions.

Letter J:

- PPO: Total Collisions = 40; Total Out of Limits = 60; Mean Time Without Collisions = 60.6 steps.
- DDPG: Total Collisions = 45; Total Out of Limits = 55; Mean Time Without Collisions = 51.62 steps.
- TD3: Total Collisions = 40; Total Out of Limits = 60; Mean Time Without Collisions = 60.6 steps.
- Discussion: PPO and TD3 performed identically and better than DDPG, which had more collisions and a shorter mean time without collisions.

Letter M:

- All Models (PPO, DDPG, TD3): Total Collisions = 57; Total Out of Limits = 43; Mean Time Without Collisions = 25.77 steps.
- Discussion: Performance was identical across all models, indicating that this was a difficult environment for each model.

Letter O:

- All Models (PPO, DDPG, TD3):Total Collisions = 15;Total Out of Limits = 85; Mean Time Without Collisions = 83.93 steps.
- Discussion: All models performed identically in this environment, suggesting it was equally navigable for each DRL approach.

The Best and Worst Letters Overall

- Best Performance Overall: All models had the lowest number of collisions and the highest mean time without collisions for the letter A and O.
- Worst Performance Overall: All models had the highest number of collisions and the lowest mean time without collisions, indicating a challenging environment, for letter M.

Expected Performance Letters like A and O, which are less complex in structure, were expected to be easier to navigate, resulting in better performance metrics. Letter M, with its multiple turns and complex shape, posed a greater challenge, as reflected in the higher collision rates and lower mean times without collisions.

General Observations

- **Consistency Across Models:** In many environments (Letters A, B, and O), all models performed identically, indicating that the structure of these letters did not favor one model over another.

- **Variability in Performance:** In more complex environments (Letter I and J), there was some variability in performance, with PPO and TD3 generally performing better than DDPG.
- **Challenging Environments:** Letter M proved to be the most challenging environment for all models, with consistently high collision rates and low mean times without collisions.

Analysis and Discussion of the Table "Ronda 2 de Treino- Env Initial Position different from Webots Initial Position" (Figure 28)

Alphabet-Following Observations

- **Letter A:**
  - PPO: Total Collisions = 37; Total Out of Limits = 63; Mean Time Without Collisions = 57.58 steps.
  - DDPG: Total Collisions = 37; Total Out of Limits = 63; Mean Time Without Collisions = 57.58 steps.
  - TD3: Total Collisions = 39; Total Out of Limits = 61; Mean Time Without Collisions = 58.19 steps.
  - Discussion: All models performed similarly, with slight variations in mean time without collisions.
- **Letter B:**
  - PPO: Total Collisions = 43; Total Out of Limits = 57; Mean Time Without Collisions = 48.63 steps;
  - DDPG: Total Collisions = 49; Total Out of Limits = 51; Mean Time Without Collisions = 35.57 steps.
  - TD3: Total Collisions = 49; Total Out of Limits = 51; Mean Time Without Collisions = 35.57 steps;
  - Discussion: PPO performed better with fewer collisions and a higher mean time without collisions compared to DDPG and TD3.
- **Letter I:**
  - PPO: Total Collisions = 28; Total Out of Limits = 72; Mean Time Without Collisions = 42.17 steps.
  - DDPG: Total Collisions = 23; Total Out of Limits = 77; Mean Time Without Collisions = 41.14 steps.
  - TD3: Total Collisions = 22; Total Out of Limits = 78; Mean Time Without Collisions = 41.32 steps.
  - Discussion: TD3 and DDPG had fewer collisions than PPO, with TD3 performing slightly better in mean time without collisions.
- **Letter J:**
  - PPO: Total Collisions = 35; Total Out of Limits = 65; Mean Time Without Collisions = 70.42 steps.
  - DDPG:
  - Total Collisions = 35; Total Out of Limits = 65; Mean Time Without Collisions = 70.42 steps.
  - TD3: Total Collisions = 30; Total Out of Limits = 70; Mean Time Without Collisions = 79.74 steps.

  - Discussion: TD3 performed the best with the highest mean time without collisions and fewer collisions.
- **Letter M:**
  - All Models (PPO, DDPG, TD3):Total Collisions = 53; Total Out of Limits = 47; Mean Time Without Collisions = 25.02 steps.
  - Discussion: Performance was identical across all models, indicating a challenging environment.
- **Letter O:**
  - All Models (PPO, DDPG, TD3); Total Collisions = 11; Total Out of Limits = 89; Mean Time Without Collisions = 83.19 steps.
  - Discussion: All models performed identically in this environment, suggesting it was equally navigable for each DRL approach.

The Best and Worst Letters Overall

- Best Performance Overall: All models had the lowest number of collisions and the highest mean time without collisions for the letter O. Also, TD3 performed the best with the highest mean time without collisions for letter J.
- Worst Performance Overall: All models had the highest number of collisions and the lowest mean time without collisions for letter M, indicating a challenging environment.

Comparison Between the Two Tables for Round 2

- The consistent performance in Letters A, M, and O across both tables indicates that these environments were equally navigable regardless of the initial position.
- Differences in performance in Letters B, I, and J suggest that the initial position had a significant impact on how the models navigated these environments.
- PPO generally showed better performance in complex environments when the initial position was changed, while TD3 showed improvement in certain cases, highlighting its adaptability.

The comparison reveals that while some environments are navigable with consistent performance across different initial positions, others show variability depending on the starting point. This suggests the need for further optimization and robustness in the models to handle different initial positions effectively.

4 Conclusions and Future Work

In this work, we aimed to evaluate the performance and robustness of three different Deep Reinforcement Learning (DRL) models—PPO, DDPG, and TD3—in navigating various letter-shaped environments. Our main goal was to assess their ability to follow walls and handle complex navigational challenges.

Summary of Results:

- Initial Training Round:
  - Each DRL model was trained with 60,000 timesteps across different environments.

  - Performance metrics such as total collisions, total out-of-limits occurrences, and mean time without collisions were tracked using TensorBoard.
  - PPO showed the most consistent performance, with fewer collisions and higher mean times without collisions across different environments compared to DDPG and TD3.
  - Complex environments, like Letter M, posed significant challenges for all models, resulting in higher collisions and shorter mean times without collisions.
- Testing with Different Initial Positions:
  - We tested the robustness of the models by varying the initial positions.
  - The results highlighted the impact of initial positioning on model performance, with PPO maintaining consistent results, while DDPG and TD3 showed more variability.
  - Best performances were observed in simpler environments like Letter O and A, while more complex environments like Letter M continued to challenge the models.

Our study demonstrates that while PPO generally outperforms DDPG and TD3 in terms of consistency and fewer collisions, all models face significant challenges in more complex environments. The initial position of the robot plays a crucial role in its performance, highlighting the need for robust and adaptive training strategies. The adjustments in distance constraints and increased training timesteps aimed to address these issues, but the inherent challenges of complex navigation tasks remain.

Final Observations It was also concluded that PPO performs better on curved paths and shapes, while the other two models are better on straight paths. In terms of training and execution time, PPO is faster. All models have difficulties dealing with corners, as demonstrated with the letter M.

Whenever a reset is necessary, the robot continues training until it reaches the initial position, which affects both training and testing. DDPG handles curves and edges better than PPO.

The fact that the robot continues training while repositioning is a disadvantage. We tried using the supervisor to handle transportation instead of the move to robot function, but we encountered an error indicating that the robot type was missing a translation field. When we checked in Webots, this field did exist, so we found no other solution but to use the move to robot function.

Also, in total, combining rounds 1 and 2, each model was trained with 180K total timesteps, which we consider insufficient. However, we lacked the computational capacity to train for more due to the time it would take. Consequently, the results are heavily influenced by the limited training. We believe that training each model with millions of total timesteps would yield better results.

Future Work Further Investigations and Adjustments:

- We suspected that the problem might be related to the distance constraints and adjusted these values.

– Training timesteps were doubled, with 2 iterations of 20,000 timesteps each (40,000 timesteps per environment and 120,000 per DRL model) to enhance training quality.
– We observed that DDPG handled curves and edges better than PPO, but the constant straight-line movement led to resets when the robot reached the limit distances.

The identical performance of all models in simpler environments like Letters A and O suggests that these environments do not provide sufficient differentiation in model capabilities. Future work should focus on refining the models to handle more complex environments more effectively and exploring additional strategies to improve overall robustness and performance. Despite the challenges, this work provides valuable insights into the strengths and weaknesses of different DRL models in navigational tasks, paving the way for further advancements in robotic navigation.

Challenges and Technical Adjustments:

– The necessity of resetting the robot and its impact on training and testing was identified as a disadvantage.
– Attempts to use the supervisor for transportation encountered errors due to missing translation fields, forcing us to use the move to robot function instead.

Acknowledgments

The research conducted in this project was part of a university course: Introduction to Intelligent Robotics, in the Bachelor's degree program in Artificial Intelligence and Data Science. We chose the topic "Wall Following Robot with DRL." The responsible professors are Luís Paulo Reis and Gonçalo Leão.

References

[1] Bolles, R.C., Fischler, M.A.: A ransac-based approach to model fitting. Communications of the ACM 24(6), 381–395 (Jun 1986)
[2] X Engineer: Proportional controller. https://x-engineer.org/proportional-controller/, [Online; accessed 12-June-2024]
[3] Electronics Coach: Proportional integral controller. https://electronicscoach.com/proportional-integral-controller.html, [Online; accessed 12-June-2024]
[4] Opti Controls: Proportional controller. https://blog.opticontrols.com/archives/344, [Online; accessed 12-June-2024]
[5] Franklin, G.F., Powell, J.D., Emami-Naeini, A.: Feedback Control of Dynamic Systems. Prentice Hall, 5th edn. (2005)
[6] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature 518(7540), 529–533 (Mar 2015)

[7] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. In: Proceedings of the 34th International Conference on Machine Learning. pp. 1–11. arXiv preprint arXiv:1707.06347 (2017)
[8] Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning. In: Proceedings of the 4th International Conference on Learning Representations (ICLR). pp. 1–14. ICLR (2016)
[9] Fujimoto, S., van Hoof, H., Meger, D.: Addressing function approximation error in actor-critic methods. In: Proceedings of the 35th International Conference on Machine Learning (ICML). pp. 1–10. ICML (2018)
[10] Cyberbotics: Running extern robot controllers. https://cyberbotics.com/doc/guide/running-extern-robot-controllers, [Online; accessed 12-June-2024]
[11] Michel, O., et al.: Webots: Professional mobile robot simulation. International Journal of Advanced Robotic Systems 18(1), 1–10 (2021)
[12] Digi-Key: Intro to reinforcement learning using gymnasium and stable baselines3. https://www.digikey.com/en/maker/projects/intro-to-reinforcement-learning-using-gymnasium-and-stable-baselines3/28c6602f5d1e4ce1b5a90642a1ac7efc, [Online; accessed 12-June-2024]
[13] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Gymnasium: A toolkit for developing and comparing reinforcement learning algorithms. https://gymnasium.farama.org/ (2023)
[14] Cyberbotics: e-puck robot guide. https://www.cyberbotics.com/doc/guide/epuck?version=R2021a, [Online; accessed 12-June-2024]
[15] Chan, S.C., Korattikara, A., Canny, J., Guadarrama, S., Fishman, S.: Rliable: Better evaluation for reinforcement learning. In: Advances in Neural Information Processing Systems (NeurIPS) (2020), https://github.com/google-research/rliable
[16] Chan, S.C., Fishman, S., Canny, J., Korattikara, A., Guadarrama, S.: Rl reliability metrics. https://github.com/google-research/rl-reliability-metrics (2020), international Conference on Learning Representations (ICLR)
[17] Zhu, H., Jiang, G.: Deep reinforcement learning with optimized reward functions for robotic trajectory planning. IEEE Transactions on Automation Science and Engineering (2022)
[18] Anonymous: Rebel: A regularization-based solution for reward overoptimization in reinforcement learning from human feedback (2023), arXiv preprint arXiv:2312.14436