Wall Following with Reinforcement Learning

Ana Batista, Gonçalo Monteiro, Mafalda Aires

FEUP - Faculty of Engineering, University of Porto
FCUP - Faculty of Ciences, University of Porto

Abstract. Wall-following is a fundamental task in robotics that underpins more complex challenges such as mapping and localization. Effective wall-following enables robots to navigate environments accurately, which is crucial for autonomous operations This paper evaluates the performance of various reinforcement learning (RL) algorithms and reward strategies to solve the wall-following problem. The objective is to identify the best combinations of algorithms, reward structures, and action spaces for achieving high accuracy and reliability in guiding a robot to follow walls. We conducted experiments using a Lidar sensor for object detection and a collision sensor for obstacle avoidance. The study examines both discrete and continuous action spaces, assessing the performance of multiple RL algorithms, including Q-learning, Twin Delayed DDPG (TD3), and Proximal Policy Optimization (PPO), with different reward strategies. Our experimental findings reveal that PPO, when utilized with both continuous and discrete action spaces and an optimized reward structure, delivers superior performance, resulting in smoother and more efficient wall-following behavior.. The results indicate that the combination of RL algorithm, reward structure, and action space significantly affects performance in wall-following tasks. PPO with continuous action spaces and well-designed rewards is particularly effective. These insights can inform future research and the development of autonomous navigation systems in robotics.

Keywords: Wall Following, Reinforcement Learning, Robotics, Cyberbotics, Webots

1 Introduction

Wall following is a Robotics task where a robot maintains a consistent proximity to a wall. The robot uses sensors to detect and measure the distance to the wall, and continuously adjusts its path to keep itself aligned with the wall while moving forward. This task is crucial for various applications. Developing an effective wall-following algorithm not only enhances the robot's ability to navigate autonomously but also serves as a stepping stone for more complex tasks such as mapping and localization.

This project aims to develop and optimize a model for the wall following task. we will tailor reward functions for both continuous and discrete action spaces. We will compare various algorithms and evaluate some rewards to identify the most suitable ones for our specific tasks and conditions. The effectiveness of these algorithms will be assessed by testing them on different maps. The best-performing model will then be fine-tuned using transfer learning on a more complex map to further enhance its performance and adaptability.

The remaining structure of this work includes the following sections: the methodological approach, detailing the software used, the setup of the environment, and the reward design; the experimental section, presenting the experiments and showing the results in detail, comparing algorithms and action spaces to identify their strengths and weaknesses; and the conclusions, where we discuss the results.

2 Methodological approach

2.1 Simulator and Robot

For this experiment, we used `Webots`, an open-source 3D robot simulator. We chose `Webots` for its extensive support for robotics modeling and simulation, as well as its simplicity and ease of use.

Our chosen robot is the E-Puck, a small, differential two wheeled mobile robot widely used in education and research. The e-puck is equipped with a range of sensors, such as proximity sensors, a 3-axis accelerometer, a microphone array, and a camera, making it suitable for plenty of tasks. For this project, we utilized only the Lidar and Bumper touch sensors.

Lidar: The `Lidar` sensor determines the distance to surrounding objects. For this experiment, it is configured with one layer of 40 rays covering a 180º field of view a maximum range of 0.3 meters, and a minimum range of 0.05 meters, which is the lowest possible value. Any readings outside this range are returned as infinity.

Bumper: The bumper is a touch sensor for detecting collisions.

2.2 Reinforcement Learning

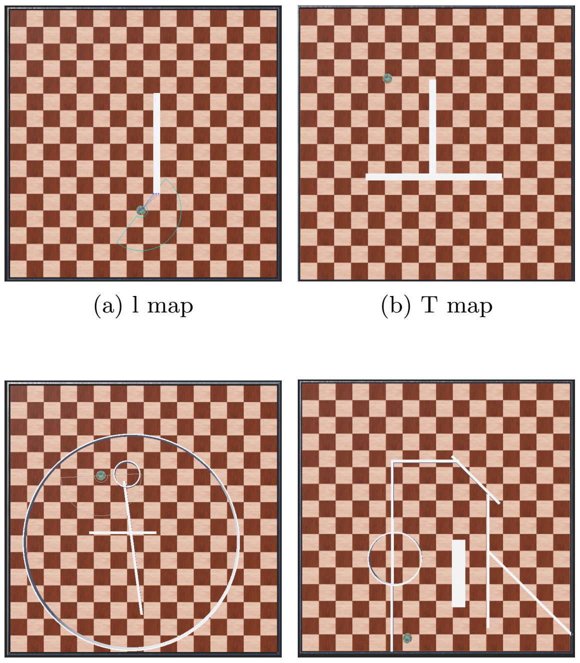
We customized a Gymnasium OpenAI environment for the virtual setup, where the Lidar readings constituted the observation space, and the possible movement values of the robot formed the action space.For the reinforcement learning algorithms, we utilized Stable Baselines3 and integrated TensorBoard for monitoring the training process. Together, these tools offer a robust framework for conducting Renforcement Learning experiments efficiently.

Algorithms and Policy For our experiments, we selected a combination of algorithms from the Stable Baselines3 library, along with the MlpPolicy:

- **PPO (Proximal Policy Optimization)**: Chosen for its robustness and efficiency in handling both discrete and continuous action spaces, PPO updates the policy in a localized manner, which is beneficial for managing the high dimensions of sensor observations and robot actions.
- **DQN (Deep Q-Network)**: Particularly effective for problems with discrete actions, DQN employs a neural network to estimate the Q-function, enabling the agent to learn optimal actions, including discrete movements like robot motion or rotation.
- **TD3 (Twin Delayed DDPG)**: A variation of the DDPG algorithm, TD3 is optimized for continuous action spaces. It utilizes neural networks to approximate both the policy and Q-function, facilitating learning of continuous actions, such as velocity and direction of robot movement.

2.3 Maps

To train and test the agent in different scenarios, we created various distinct Webots worlds with placed walls: two simple and two complex. The robot starts in random positions within a map whenever the environment resets, ensuring diverse training experiences.


(a) l map (b) T map

(c) x map (d) maze map

Fig. 1. Maps used

2.4 Reward Function

The reward function was designed to control four parameters:

- Collision: Obstacles closer than 0.05 meters are considered collisions.
- The difference in distance between the leftmost value of the Lidar and the desired distance (since the robot must follow the wall on the left).
- The linear velocity, with the desired outcome being to maintain a maximum and constant linear velocity.
- The smoothness of the movement, which is evaluated by comparing the angular velocity value from the previous step to the current one, penalizing higher variance.

Discrete Action Space

In the case of the discrete action space, there are only 3 actions with fixed values, so the reward function mainly depends on the distance to the wall and the collision value.

The idea was to build a function that returns normalized values, meaning it returns 1 when the left value has the desired distance, and -1 when it has the maximum difference, which is 0.3 (max range) − 0.15 = 0.15. Given that, we could multiply it by some factor to represent the importance we want to give to this reward parameter relative to the others.

$$-\frac{800}{9}(s[0]-d)^2+1$$

between 0 and 0.3 is shown below. We assume $d = 0.15$.

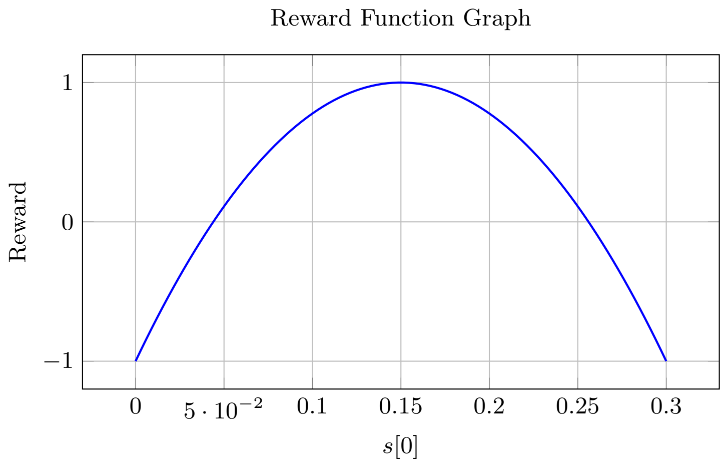


Fig. 2. Graph of the reward function between 0 and 0.3.

Given:

- s: the current state (with $s[0]$ representing the left value),
- a: the action,
- d: the desired distance (default $d = 0.15$),
- r: the reward,
- b: the bumper value,
- min_range: the minimum range value (0.005 threshold),
- max_range: the max range value is 0.3,
- f: a factor (default $f = 2$).

The reward function $r(s, a)$ is defined as:

$$r(s,a) = \begin{cases} -10 & \text{if } b = 1.0 \text{ or } \min(s) \leq (\text{min\_range} + 0.005) \\ f\left(-\frac{800}{9}(s[0]-d)^2+1\right) & \text{otherwise} \end{cases}$$

This function ensures that the robot receives a higher reward for maintaining a distance close to the desired one d, with strong penalties for collisions or getting too close to obstacles.

Modeling this reward for the discrete action space is straightforward. It works effectively for this task (as evidenced in the examples provided) and is direct to the point. The limitation is that we don't have control over the other parameters, which can result in a less robust and sub-optimal reward.

Continuous Action Space

For the continuous action space, we have all four parameters to modulate and establish a relationship between them.

Collision parameter. The collision reward remains the same as in the discrete action space, with a strong penalty for a collision.

Distance parameter. The distance reward parameter remains the same as in the discrete action space. The `following_reward` function is defined as follows:

$$\text{following\_reward}(x, \text{factor} = 2) = \left(-\frac{800}{9}(x - \text{desired\_distance})^2 + 1\right) \times \text{factor}$$

where:
- x is the input left value.
- desired_distance is the desired distance (0.15).
- factor is a scaling factor for representing the importance of the parameter on the overall reward.

Linear Velocity parameter

The linear velocity is provided to the reward function as an argument already normalized between -1 and 1, so we just need to multiply by the scaling factor for this parameter.

The `linear_velocity_reward` function is defined as follows:

$$\text{linear\_velocity\_reward}(\text{factor} = 2) = \text{linear\_vel\_norm} \times \text{factor}$$

where:
- linear_vel_norm is the normalized linear velocity.
- factor is a scaling factor with a default value of 2.

Smoothness Parameter The smoothness reward depends on the relationship between the angular velocity of the current and previous steps. This function is necessary because in the continuous action space there is too much freedom of movement, sometimes causing the robots to agitate excessively. This reward prevents such behavior, allowing for faster and more stable training.

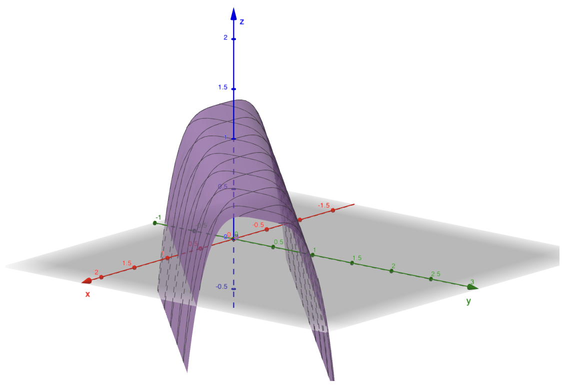
To model this function, we want to reward (+1) when the angular velocities are relatively close to each other and penalize (-1) when they are too far apart. The following function achieves this goal effectively:

The `angular_smoothness_reward` function is defined as follows:

$$\text{angular\_smoothness\_reward}(x, y, \text{factor} = 1) = \left(1 - 2 \cdot (x - y)^4\right) \times \text{factor}$$

- x is the current angular velocity.
- y is the previous angular velocity.
- factor is a scaling factor with a default value of 1.

Below is the 3D visualization of the function:



(a) Smoothness reward

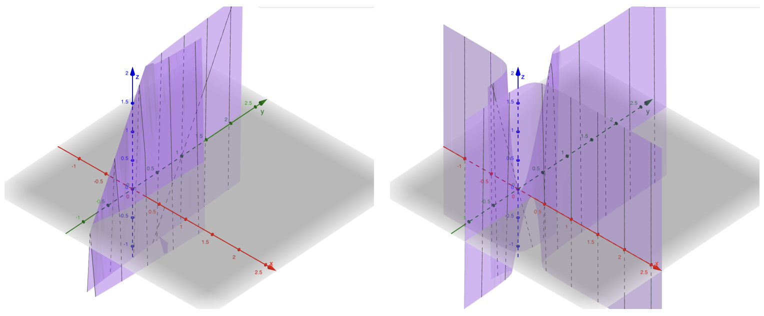
Fig. 3. The x and y axes represent the previous and current normalized angular velocities, respectively.

Observe that the maximum value for this reward parameter is 1 and the minimum is -1, as we intended. We use a quartic function instead of a quadratic function because we want it to be steeper at the limits.

With this, we just need to multiply by the factor. For this reward, factor = 1 is the best choice as we don't want smoothness to have a significant impact on the reward.

Combination of parameters Given the above, we had to combine the three parameters (distance, linear velocity, and smoothness) and observe which combination performs best. These were the combinations we tried:

- (1) Total reward = dist_reward(f=2) + linear_vel(f=1) + smooth(f=1)
- (2) Total reward = dist_reward(f=2) + linear_vel(f=1.5) + smooth(f=1)
- (3) Total reward = dist_reward(f=2) + linear_vel(f=2) + smooth(f=1)
- (4) Total reward = dist_reward(f=2) * linear_vel(f=2) + smooth(f=1)



(a) reward (3) (b) reward (4)

Fig. 4. Comparison between sum reward and multiplication reward strategies. The y-axis represents the linear parameter range from [-1, 1], while the x-axis represents the distance parameter range from [0, 0.3].

2.5 The Approach

The Aproach is divided into four main phases: Initial Training, Initial Testing and Selection, Evaluating Transfer Learning, and Final Testing and Analysis.

The first step is to evaluate the performance of different reinforcement learning algorithms, namely Proximal Policy Optimization (PPO), Twin Delayed DDPG (TD3), and Deep Q-Network (DQN), for both discrete and continuous action spaces. Each algorithm will be trained for 100,000 steps using the same reward structure. This training will be conducted on two different maps: the I map and the T map. After the training, the models will be tested on a maze map to determine which algorithms perform better.

In addition to comparing algorithms, the impact of different reward structures on the PPO algorithm in a continuous action space will be evaluated. Various reward structures will be experimented with, and the training will be conducted exclusively on the I map using the PPO algorithm.

Following the initial training phase, the next step is to identify the best-performing algorithm and reward structure. The models trained in the previous steps will be tested on the maze map. By comparing their performances, the best algorithm and reward configuration for both continuous and discrete action spaces will be determined.

To assess the effectiveness of transfer learning compared to training from scratch, two approaches will be undertaken. The first approach, "From Scratch

Training," involves training the selected best algorithms and reward configurations for 300,000 steps on the X map. The second approach, "Transfer Training," involves taking the best models from the initial 100,000-step training and further training them for an additional 200,000 steps on the X map.

The final phase involves evaluating and comparing the performance of models trained from scratch and those trained using transfer learning. Both sets of models will be tested on the maze map to assess their optimization and performance. A detailed analysis of the results will be conducted to determine the effectiveness of the training and fine-tuning processes, with a focus on whether transfer learning offers a significant advantage.

3 Experimental evaluation

3.1 Reward Experiments

Training graphics for the different corresponding rewards

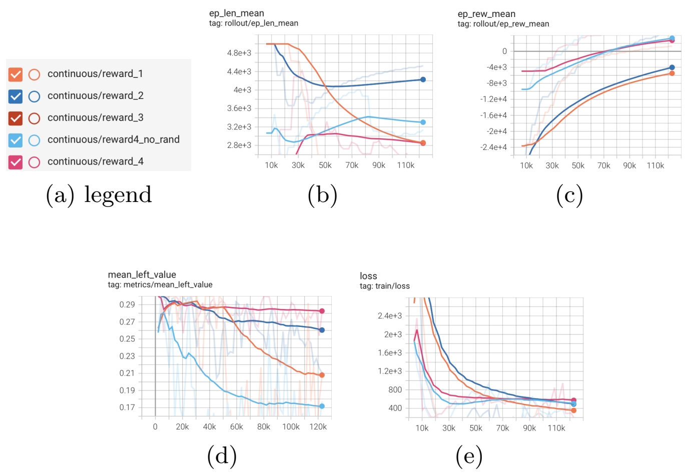


Fig. 5. Reward Training

All combinations performed well during training, but we observed that the second combination avoided block moments more frequently. This makes sense since we assigned greater importance to linear velocity through the factor value. However, this also made the distance to the wall parameter less rigorous, and the robot did not consistently maintain the desired distance.

We prioritized the prevention of block moments because they can significantly disrupt the training process. While reaching the desired distance can be managed effectively through an increased number of training steps, block moments pose a more substantial challenge, potentially leading to inefficient learning and prolonged training times. Therefore, our focus was on minimizing these occurrences to ensure a smoother and more efficient training experience.

through graph analysis and the training graphic, we could hypothesized that the multiplication method might yield superior results. However, throw practical results from training sessions showed that the sum method was more reliable and stable, making it the preferred choice for the remaining experiments.

Note 1. A block moment occurs when the robot becomes stuck or unable to move effectively within the environment.

Note 2. The rewards were trained and tested on the "l map," all using the PPO algorithm.

3.2 Evaluation Method

The evaluation is based on the assessment of training metrics and on the performance in the test worlds. The key training metrics utilized in our evaluation include:

- **Mean Reward Per Episede**: it s important.
- **Training Time**: The duration required to train the model.
- **Mean Minimum Distance**: The average minimum distance to obstacles, measured by the leftmost lidar sensor.
- **Loss**: The training loss, indicating the convergence quality of the model.

we also considered the graphs from tensor board and by visualizing the tests on maze map that will be shared as videos

"'latex

3.3 Results

After training the models for 100,000 steps, we proceeded to test them all in a separate environment ("Maze World"). The metrics of these tests can be found in Table 1.

Table 1. Metrics of the first round of experiments.

Action Space	**Algorithm**	**World**	**Mean Minimum Distance**	**Mean Reward**	**Mean Episode Length**
Continuous	PPO	l	0.25	9239	3164
Continuous	PPO	T	0.25	6256	2248
Continuous	TD3	l	0.29	5970	4010
Continuous	TD3	T	0.26	-320	79
Discrete	PPO	l	0.24	1428	1033
Discrete	PPO	T	0.29	45	67
Discrete	DQN	l	0.18	9647	5000
Discrete	DQN	T	0.17	79	141

While the obtained metrics are important, we found that a more useful way to assess whether the agent is truly learning, is to visualize its performance in test environments. By watching the agent in action along with the values in the table above, we can draw the following conclusions:

- Compared to the models trained in map T, those trained in map I (a simpler map) were able to generalize much better, achieving better results in the test world.
- The agent trained with the PPO model (in various situations) demonstrates superior learning compared to the TD3 and DQN models, for continuous and discrete action spaces, respectively.
- The agents trained with the PPO model in map I show the best learning outcomes.

After evaluating the performance of these models, we further tested PPO (which gave the best results) with and without fine-tuning on a more complex map. The results are presented in Table 2.

Table 2. Metrics of the second round of experiments with 300000 steps.

Action Space	**Fine tuning**	**Mean Minimum Distance**	**Mean Reward**	**Mean Episode Length**
Continuous	No	0.23	2609	1423
Discrete	No	0.15	9274	5000
Continuous	Yes	0.26	12437	4999
Discrete	Yes	0.11	6499	5000

Based on the results shown in Table 2 and the visual analysis of the agent in the Maze Map, we conclude that:

- In both discrete fine-tuned and non-fine-tuned agents, the model was not successful in generalizing its knowledge, as it did not perform well in the test map. The Mean Episode Length equals 5000 (the maximum steps for an episode), indicating that the agent may have been stuck and not moving.
- The use of PPO with our continuous action space yielded the best results. This model was able to follow the walls almost perfectly in our test map. The only issue was that it got stuck in a particularly challenging section of the map. However, before reaching that point, it navigated the maze flawlessly.
- To optimize the results of the discrete action space in our experiments, adding a penalty for repeatedly choosing turn right/left actions could be beneficial.

3.4 Implications for Autonomous Navigation

Our findings underscore the importance of carefully selecting and tuning both RL algorithms and reward structures for specific robotic tasks. The insights gained from this study can inform the development of more advanced autonomous navigation systems, particularly in environments where precise maneuvering and obstacle avoidance are critical.

3.5 Future Work

Several avenues for future research and improvement have emerged from this study:

- Transfer Learning: Apply the trained models to different and more complex environments to assess their generalizability and adaptability.
- Hybrid Approaches: Explore the integration of discrete and continuous action spaces within a single framework to leverage the strengths of both.
- Advanced Reward Functions: Develop more sophisticated reward functions that incorporate additional parameters such as energy efficiency, task completion time, and safety margins.
- Multi-Agent Systems: Investigate the application of these RL techniques in multi-robot systems where coordination and cooperation are required.
- Real-World Deployment: Transition from simulated environments to real-world testing to validate the practical applicability of the models and algorithms.
- Enhanced Sensing: Incorporate additional sensors (e.g., cameras, IMUs) to provide richer environmental feedback, potentially improving navigation accuracy and robustness.

In conclusion, our work provides a solid foundation for the development of advanced wall-following algorithms in robotics. By refining RL techniques and reward structures, we can achieve significant improvements in autonomous navigation, paving the way for more versatile and capable robotic systems.