Path Planning Using Reinforcement Learning for E-Puck Robot

Daniel Dias[1,2], Lucas Santiago[1,2], Rafael Conceição[1,2]

[1] FCUP - Faculty of Sciences, University of Porto
[2] FEUP - Faculty of Engineering, University of Porto

Abstract. Path planning is a challenging problem essential to autonomous robotics, with significant implications for efficiency and adaptability in various applications. This paper presents a method for path planning using reinforcement learning (RL) models for an E-Puck robot equipped with LiDAR sensors. The approach involves training the robot in a simulated environment using the Proximal Policy Optimization (PPO) algorithm implemented in the Webots simulator. The observation space includes LiDAR readings, the angle to the target, and the distance to the target. The robot's action space comprises rotating left, rotating right, moving forwards and moving backwards. Experimental results demonstrate that this method effectively navigates the robot to its target, even in dynamic and partially unknown environments. Therefore, this solution has the potential to enhance the autonomy and efficiency of robotic systems in various settings.

Keywords: Reinforcement Learning, Path Planning, PPO, Simulation, Webots, E-Puck

1 Introduction

Path planning for robotic systems involves determining an optimal path for a robot to navigate from a starting point to a target location while avoiding obstacles. This task is critical in various domains such as autonomous driving, warehouse automation, and service robotics [1]. Traditionally, path planning has relied on deterministic algorithms, such as Dijkstra's and A*, but with the advent of more complex environments and the need for adaptive behaviors, reinforcement learning (RL) has emerged as a promising approach [2].

Automating path planning through reinforcement learning can lead to significant advancements in the efficiency and adaptability of robotic systems. RL allows agents to learn optimal policies through interactions with their environment, enabling them to handle dynamic and unforeseen obstacles more effectively. However, the application of RL to path planning poses several challenges, including the need for efficient exploration strategies and the computational complexity of training models in high-dimensional spaces.

Some of the state-of-the-art RL models for a task of this profile are the Proximal Policy Optimization (PPO) and the Deep Deterministic Policy Gradient

(DDPG). The PPO is an algorithm that balances exploration and exploitation by using a clipped objective function to ensure stable and efficient policy updates, making it well-suited for continuous control tasks like this one.

This paper focuses on developing a robot that uses PPO for path planning. The robot is trained within the Webots simulation platform, which provides a versatile environment for testing and validating robotic behaviors. The goal is to create a computationally efficient and robust path planning algorithm that can generalize well across different environments. By leveraging PPO, the proposed solution aims to achieve a balance between exploration and exploitation, ensuring that the robot can navigate complex terrains while optimizing its path efficiency.

Section 2 provides a brief review of the literature on path planning and reinforcement learning. Section 3 details our approach for implementing the Proximal Policy Optimization (PPO) algorithm, including the experimental setup and the specifics of the Webots simulation platform. In Section 4, we present the results, highlighting the computational efficiency and effectiveness of our algorithm in various scenarios. Finally, Section 5 concludes with a discussion of our contributions and potential directions for future work.

2 Literature Review

The application of reinforcement learning (RL) to robotics has gained significant traction over the past few decades. RL allows robots to learn optimal policies through interactions with their environment, making it particularly suitable for dynamic and complex tasks such as path planning. The foundation of RL in robotics can be traced back to the early works of Thrun and Schwartz [3] and Kaelbling, Littman, and Moore [4], who explored the integration of learning algorithms with robotic control systems.

Among the various RL algorithms, Proximal Policy Optimization (PPO) has gained prominence for its effectiveness in continuous control tasks. Introduced by Schulman et al. [5], PPO strikes a balance between exploration and exploitation by using a clipped objective function to ensure stable and efficient policy updates. Its application in robotic path planning has shown promising results, enabling robots to navigate complex environments with improved efficiency.

Several studies have demonstrated the potential of RL in robotic navigation. Thrun et al.'s work on probabilistic robotics provides a overview of the principles underlying autonomous robot navigation, including the use of sensors and algorithms for environment mapping and path planning [**?**].

In addition to academic research, practical implementations have highlighted the versatility of RL in real-world scenarios. The DoraPicker project, for example, showcased an RL-based system that participated in the 2015 Amazon Picking Challenge, demonstrating the feasibility of using RL for complex manipulation tasks in unstructured environments [6].

The use of simulation platforms, such as Webots, has further accelerated the development and testing of RL algorithms for robotics. Webots provides a realistic environment for training and validating robotic behaviors, enabling researchers to experiment with various scenarios and refine their approaches before deploying them in real-world applications [7].

In summary, the integration of reinforcement learning with robotic path planning represents a significant advancement in the field, offering adaptive and efficient solutions for navigating complex environments. While traditional algorithms provide a solid foundation, the adaptability of RL, as demonstrated by algorithms like PPO, showcases the potential for future developments. The use of simulators like Webots and sensors like LiDAR further enhances the capabilities of these systems, paving the way for more intelligent and autonomous robotic solutions.

3 Methodology and Experimental Setup

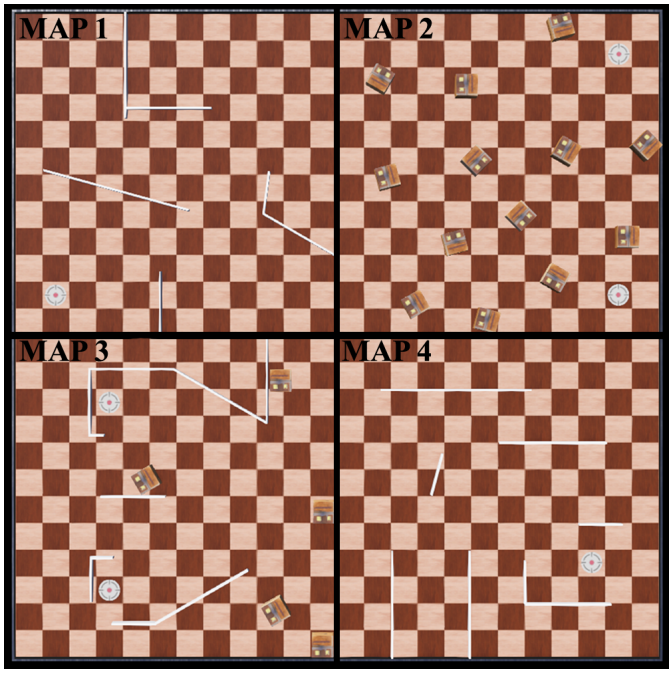

Fig. 1. Maps Created to train and test the robot

3.1 Setup

This work was conducted using the Webots Robot Simulator. The robot utilized for the experiments is the E-Puck, equipped with various sensors, including LiDAR, which was extensively used in this study. LiDAR (Light Detection and Ranging) is a remote sensing method that uses laser light to measure distances to objects. A LiDAR sensor emits laser pulses that bounce off objects and return to the sensor. By measuring the time it takes for the pulses to return, the sensor calculates the distance to each object. This sensor returns point clouds representing the physical surfaces of the walls and objects in the environment. Multiple approaches were employed to process these point clouds.

3.2 Training Methodology

The 4 maps in **Fig.1** were used to train the robot. It was decided to have more than one map in order to allow the robot to experience diverse scenarios,

adapting to whatever situation it encounters. Map 2 and map 3 have more than one target which allow for double target path planning scenarios.

The training was divided into 4 categories: EASY, MEDIUM, HARD, ALL. This approach is known as "Curriculum Learning." Curriculum Learning is a training strategy in machine learning where a model is progressively trained on increasingly difficult tasks. The idea is to first train the model on simpler categories (EASY and MEDIUM) and gradually increase the complexity (HARD and ALL). This method can lead to better generalization and faster training times compared to training the model on difficult tasks from the start. We also used a learning rate of 0.0003

Fig.2 represents: as green – the easy/medium start positions for the robot; as red - the hard/advanced starting positions for the robot.

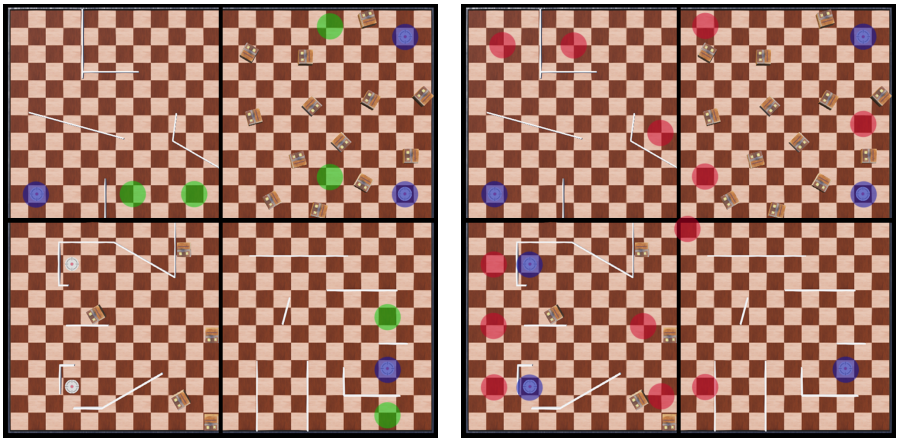

Fig. 2. Start position/target combinations

3.3 Training Levels

EASY In the beginning of every episode the robot will start in a randomly selected green position and will aim for a single blue target. Every single combination that can occur, will always grant that between the robot and the target there is only one obstacle. The robot is also manually rotated into the target's direction.

This simple setup was designed with the idea of briefly help the robot understand that going for the target is as good action, while going towards a wall is not a good idea.

Observed Results: After 50k episodes, the robot was fully capable of going to the target without hitting any obstacle.

MEDIUM Position and target are selected in the same way as the EASY level. The only difference is that in the MEDIUM setup, the robot will start with a random orientation at the beginning of each episode.

Observed Results: Even without being "hard coded", the robot soon learned to start every single episode by autonomously rotating to the target's direction, since that was the best action to take (only in these 1 obstacle scenarios). The learning of that behaviour was the reason of the MEDIUM level creation.

HARD Initial and target position are chosen by randomly selecting a red and a blue position, respectively. In the cases where the map has more than one target (map 2 and map 3), the robot will face a double target path planning scenario. In this setup, the robot is also manually rotated into the closest target direction.

The intention of the HARD level is to teach the robot that not in every scenario its good to start by rotating towards the target, since this level contains much more scenarios where there are several obstacles between the robot and the target.

Observed Results: The robot had a lot of difficulty to complete any of the scenarios since he was "forced" to start every episode by rotating towards the target, which most of the times had a very close obstacle. Either way, the purpose of the HARD level was achieved and prepared the robot for the next step: the ALL level.

ALL The ALL level gathers all the start position/target combinations from the EASY and HARD levels. This time the robot is not hard coded into rotating to the target.

The ALL level was the last step of the Curriculum Learning approach.

3.4 Partial Results

The results from the first "levels" of the Curriculum Learning approach were very positive. EASY, MEDIUM and HARD levels ran for 50000 episodes each and allowed the robot to learn very important behaviours:

- Going towards the target will result in a positive reward
- Hitting a wall results in a negative reward
- Not all scenarios should be started by rotating into the target

These behaviours were learnt way faster thanks to the Curriculum Learning approach. Now the robot can train on the complex ALL setup, while being able to perform good and achieve consistent results way faster that just starting in the ALL level without any previous information.

The ALL level ran for 1M episodes.

4 Experiments, Algorithms and Results

First Experiment In this experiment, the robot action space was defined by just 3 actions: go forward, rotate left and rotate right. The observation space was composed by the Euclidean distance to the nearest target, the angle difference between the robot's current orientation the target and 8 rays of the LiDAR. The reward function of this experiment is represented in **Algorithm 1**.

Algorithm 1 Reward Calculation Algorithm (First Experiment)

Input: $None$
Output:$reward, done$

$Target\_count \leftarrow Num(targets)$
$DIST\_THRESHOLD\_TO\_TARGET \leftarrow 0.1$
$Step()$
if *touch_sensor* activated **then**
    $done \leftarrow True$
    $reward \leftarrow -50$
    return $reward, done$
$(distance, closest\_target) \leftarrow calculate\_distance\_to\_target()$
if $distance < DIST\_THRESHOLD\_TO\_TARGET$ **and** $Target\_count == 1$ **then**
    $done \leftarrow True$
    $reward \leftarrow 10$
else if $distance < DIST\_THRESHOLD$ **and** $self.targs == 2$ **then**
    $self.targs \leftarrow 1$
    $done \leftarrow False$
    $reward \leftarrow 10$
    $remove(closest\_target)$
else
    $done \leftarrow False$
    $reward \leftarrow -distance$
return $reward, done$

In this first experiment, the reward function was very simple which caused several problems such as the robot suddenly stopping and being stuck on the same spot. We also found errors in the The `process_lidar()` responsible for calculating the distance to obstacles of the 8 LiDAR rays considered in the observation space.

Second Experiment In the second experiment, the `process_lidar()` function was fixed and updated to `process_lidar_v2()`. Instead of wrongfully considering a 360 LiDAR FOV, it considered 9 evenly distributed points from the 100 points of the front 90 degree FOV. This had the consequent change of the observation space, that now included 9 LiDAR points instead of 8, the Euclidean distance

to the target and the angle to the target, forming an 11-element array. The rewarding function was also modified and can be seen in **Algorithm 2**.

Algorithm 2 Reward Calculation Algorithm (Second Experiment)

Input: $None$
Output: $reward, done$

$Target\_count \leftarrow Num(targets)$
$DIST\_THRESHOLD\_TO\_TARGET \leftarrow 0.1$
$Previous\_dist\_to\_target \leftarrow Calculate\_distance\_to\_target\_in\_previous\_step()$
Step()
if *touch_sensor* activated **then**
    $done \leftarrow True$
    $reward \leftarrow -50$
    return $reward, done$
$(distance, closest\_target) \leftarrow calculate\_distance\_to\_target()$
if $distance < DIST\_THRESHOLD\_TO\_TARGET$ **and** $Target\_count == 1$ **then**
    $done \leftarrow True$
    $reward \leftarrow 10$
else if $distance < DIST\_THRESHOLD$ **and** $Target\_count == 2$ **then**
    $Target\_count \leftarrow 1$
    $done \leftarrow False$
    $reward \leftarrow 10$
    $remove(closest\_target)$
else
    $done \leftarrow False$
    $reward \leftarrow -distance$
$reward \leftarrow reward + 10 * (Previous\_dist\_to\_target - distance)$
$reward \leftarrow reward - 0.01$
$Same\_spot\_steps \leftarrow 0$
if $distance >= Same\_spot\_steps$ **then**
    $Same\_spot\_steps \leftarrow Same\_spot\_steps + 1$
else
    $Same\_spot\_steps \leftarrow 0$
if $Same\_spot\_steps > 10$ **then**
    $reward \leftarrow reward - 5$
    $Same\_spot\_steps \leftarrow 0$
$Previous\_dist\_to\_target \leftarrow distance$
return $reward, done$

The new rewarding function rewarded the robot for getting closer to the target, added a small time penalty to encourage faster movement and penalized the robot for staying in the same spot for too long.

Third Experiment In the third and last experiment, several changes were made. The action space was changed to have a new action: moving backwards, with the objective of escaping tight spaces. Other changes in **Algorithm 3**:

- Reward/penalty values revised.
- Increased speed for forward and turning actions.
- New penalties and rewards added to the reward function.

The most important changes made in the **Algorithm 3** included:

- Penalty for being too close to the wall.
- Reward for visiting new locations by tracking previous locations.
- Reduced maximum steps to 800 for faster episode termination.
- Reward for traveled distance.

All these changes led to our final version of the environment and the rewarding function. Very robust implementations that allow for good results when paired with the previous developed Curriculum Learning approach.

Algorithm 3 Reward Calculation Algorithm (Version 6)

Input: $None$
Output: $reward, done$

$Target\_count \leftarrow Num(targets)$
$Step()$
$(distance, closest\_target) \leftarrow calculate\_distance\_to\_target()$
$Previous\_dist\_to\_target \leftarrow Calculate\_distance\_to\_target\_in\_previous\_step()$
if *touch_sensor* activated **then**
    $done \leftarrow True$
    $reward \leftarrow -100$
    return $reward, done$
if $distance < DIST\_THRESHOLD$ **and** $Target\_count == 1$ **then**
    $done \leftarrow True$
    $reward \leftarrow 100$
else if $distance < DIST\_THRESHOLD$ **and** $Target\_count == 2$ **then**
    $Target\_count \leftarrow 1$
    $done \leftarrow False$
    $reward \leftarrow 80$
    $remove(closest\_target)$
else
    $done \leftarrow False$
    $reward \leftarrow -distance * 0.5$
$reward \leftarrow reward + 5 * (Previous\_dist\_to\_target - distance)$
$reward \leftarrow reward - 0.8$
$gps\_readings \leftarrow gps.getValues()$
$robot\_position \leftarrow (gps\_readings[0], gps\_readings[1])$
$current\_position \leftarrow (round(robot\_position[0], 1), round(robot\_position[1], 1))$
if $current\_position$ **not in** $self.visited\_locations$ **then**
    $self.visited\_locations.add(current\_position)$
    $reward \leftarrow reward + 2$
$distance\_moved \leftarrow np.linalg.norm(np.array(robot\_position) - np.array(self.previous\_position))$
$reward \leftarrow reward + distance\_moved * 2$
$self.previous\_position \leftarrow robot\_position$
$point\_cloud \leftarrow np.array(self.lidar.getPointCloud())$
$lidar\_features \leftarrow self.process\_lidar\_v2(point\_cloud)$
for all $feature$ **in** $lidar\_features$ **do**
    if $feature \leq 0.02$ **then**
        $reward \leftarrow reward - 0.75$
if $distance \geq Previous\_dist\_to\_target$ **then**
    $self.same\_spot\_steps \leftarrow self.same\_spot\_steps + 1$
else
    $self.same\_spot\_steps \leftarrow 0$
if $self.same\_spot\_steps > 10$ **then**
    $reward \leftarrow reward - 40$
    $self.same\_spot\_steps \leftarrow 0$
$Previous\_dist\_to\_target \leftarrow distance$
return $reward, done$

4.1 Results

Using the Curriculum Learning technique explained in **3.3 Training Levels** the robot was able to achieve very interesting results.

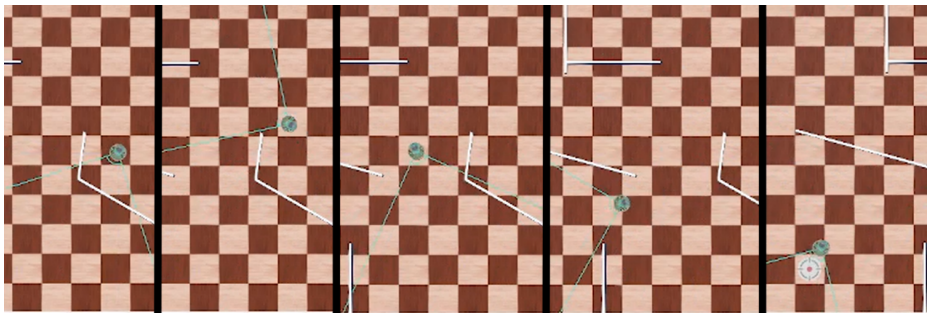

Fig. 3. Sequence of the robot starting and reaching the target.

After training the robot for several hundred thousand episodes, it shows very interesting behaviours and statistics.

```
removed closest target: self.targetpos len = 1
-----------------------------------------
| rollout/                |             |
|    ep_len_mean          | 52.2        |
|    ep_rew_mean          | 59.1        |
| time/                   |             |
|    fps                  | 31          |
|    iterations           | 29          |
|    time_elapsed         | 1867        |
|    total_timesteps      | 59392       |
| train/                  |             |
|    approx_kl            | 0.0024949433 |
|    clip_fraction        | 0.0297      |
|    clip_range           | 0.2         |
|    entropy_loss         | -0.272      |
|    explained_variance   | 0.617       |
|    learning_rate        | 0.0003      |
|    loss                 | 178         |
|    n_updates            | 2720        |
|    policy_gradient_loss | -0.00296    |
|    value_loss           | 447         |
-----------------------------------------
removed closest target: self.targetpos len = 1
```

Fig. 4. Important features and training statistics

Despite these improvements, challenges remain. There is no particular scenario where the robot struggles too much but there is a lot of room for improvement. The small struggles that the robot still faces can easily be mitigated by simply letting the robot train for more episodes.

5 Conclusion and Future Work

In this work, we explored the application of reinforcement learning (RL) for path planning in robotic systems, specifically using the Proximal Policy Optimization (PPO) algorithm within the Webots simulation platform. The experimental results demonstrated that our approach effectively navigates an E-Puck robot equipped with LiDAR sensors through complex and dynamic environments. The robot successfully learned to reach its target while avoiding obstacles, showcasing the potential of RL in enhancing robotic autonomy and efficiency.

Despite these promising results, several challenges and opportunities for improvement remain. During training, the robot occasionally got stuck in certain positions, indicating that the reward function and exploration strategy might need further refinement. Additionally, the algorithm's performance could be further evaluated in more diverse and realistic environments to ensure better generalization.

5.1 Future Work

To build upon the findings of this study, the following future research directions are proposed:

- **Refinement of Reward Function**: Investigate and implement more sophisticated reward functions that can better incentivize the robot to avoid getting stuck and encourage more efficient pathfinding. This could involve incorporating penalties for repeated actions in the same location or introducing intermediate goals.
- **Experiment different algorithms**: Test other reinforcement learning algorithms such as Deep Deterministic Policy Gradient (DDPG) and compare their performance with PPO. DDPG, being an actor-critic algorithm designed for continuous action spaces, could provide insights into the advantages and limitations of different approaches in the context of robotic path planning.
- **Exploration Strategies**: Explore advanced exploration strategies such as curiosity-driven exploration or intrinsic motivation to enhance the robot's ability to discover optimal paths in complex environments.
- **Real-World Implementation**: Transfer the learned policies from the simulation environment to real-world robotic systems. This will involve addressing the sim-to-real gap and ensuring that the policies remain effective in real-world conditions.
- **Multi-Robot Coordination**: Extend the approach to scenarios involving multiple robots. Investigate the use of multi-agent reinforcement learning techniques to enable coordinated path planning and task execution among multiple robotic agents.
- **Dynamic and Uncertain Environments**: Test and enhance the algorithm's robustness in highly dynamic and uncertain environments. This includes environments with moving obstacles or changing targets.

– **Hardware Enhancements**: Integrate additional sensors and improve the hardware setup to provide richer environmental feedback and improve the robot's decision-making capabilities.

By addressing these future work directions, we aim to develop more robust and adaptable robotic systems capable of performing complex path planning tasks in a wide range of applications. The advancements in reinforcement learning and robotics presented in this paper lay the groundwork for further innovations in the field of autonomous robotics.

References

[1] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. MIT Press, Cambridge, MA, USA, 2005.

[2] Luís Paulo Reis, Nuno Lau, David Simões, and Armando Sousa. Motion planning, 2021/2022. Course Lecture Slides.

[3] Sebastian B Thrun and Anton Schwartz. Issues in using function approximation for reinforcement learning. *Proceedings of the 1993 Connectionist Models Summer School*, 6:255–263, 1993.

[4] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 222–235. JMLR.org, 2017.

[6] Tianyu Zhang, Xin Sun, Yingxin Xu, Shanzheng Zhang, Congcong Liu, Hongbin Yang, Jia Pan, and Haojian He. Dorapicker: An autonomous picking system for general objects. *IEEE Transactions on Automation Science and Engineering*, 13(4):1440–1452, 2016.

[7] Olivier Michel. Cyberbotics ltd. webots™: Professional mobile robot simulation. *International Journal of Advanced Robotic Systems*, 1(1):40–43, 2004.