Path Planning Using Reinforcement Learning

Diogo Mendes[1,2], Gonçalo Brochado[1,2], Tiago Coelho[1,2]

[1] FCUP - Faculty of Science, University of Porto
[2] FEUP - Faculty of Engineering, University of Porto
up202108102@up.pt, up202106090@up.pt, up202105004@up.pt

Abstract. Path planning is a crucial subject for autonomous navigation in robotics, continually advancing to assist autonomous vehicles and robots in finding the shortest and most obstacle-free path from a start to a goal state. Despite significant progress, existing methodologies often struggle in dynamic and uncertain environments. This paper introduces a method for path planning using reinforcement learning (RL), where an agent learns optimal navigation strategies through interactions with a simulated environment on Webots. The approach leverages sensors such as Lidar to analyse obstacle distances and avoid collisions. There were also implemented various deep reinforcement learning algorithms, including Deep Q-Network (DQN) or Proximal Policy Optimization (PPO), alongside simpler RL algorithms, such as Sarsa and Q-Learning. The RL agent is trained to navigate grid-based maps in Webots, effectively avoiding obstacles and reaching target locations. Experimental results demonstrate the effectiveness of the approach, highlighting its ability to adapt to dynamic and previously unseen environments. These findings suggest that RL-based path planning can significantly enhance autonomous navigation, offering a scalable and flexible solution to complex path planning challenges.

Keywords: reinforcement learning, path planning, lidar, simulation, webots, gym, stable baselines3

1 Introduction

Path planning involves guiding a robot from its current position to a desired state through a sequence of actions, forming a path to the target destination [1]. This process breaks down into sub-tasks like identifying positions and analyzing obstacle distances using Lidar readings, ensuring safe navigation and collision avoidance.

As interest in autonomous navigation grows, effective path planning remains crucial, demanding advanced algorithms capable of object detection, collision avoidance, and efficient route finding using sensors like Lidar, radar, GPS, and cameras [2]. However, real-time sensor noise and the challenge of finding optimal paths pose significant hurdles, hindering complete autonomy in dynamic environments.

Solving adaptive and efficient path planning is pivotal for enhancing safety and performance across fields such as search and rescue, automated delivery, and transportation systems. Addressing these challenges will enable more effective autonomous operations in real-world settings.

This paper focuses on training robot models in the Webots simulator using various reinforcement learning (RL) algorithms. The robot starts from a fixed position but targets randomly vary among four locations. The goal is to develop models capable of navigating to the target without collisions using Lidar for proximity detection, GPS for location, and a bumper sensor for collision detection. RL algorithms like DQN, PPO, Sarsa, and Q-Learning will be trained and evaluated on maps of varying difficulty levels.

This research aims to answer the following question:

- How do various RL algorithms compare in terms of performance for path planning tasks?

Section 2 consists of a brief review of the related work in the field of path planning. Section 3 presents the methodology, detailing the environment setup, sensors and RL algorithms used. Section 4 presents the experimental results and analyses, highlighting the performance of different algorithms. Lastly, section 5 concludes the paper and presents future work.

2 Literature review

Path planning for autonomous robots has advanced significantly with the integration of Lidar technology, enabling efficient and safe navigation while avoiding obstacles. The foundations of path planning were laid decades ago, with early works focusing on grid-based methods like A* provided foundational algorithms for navigation but struggle with dynamic environments [3].

Lidar (Light Detection and Ranging), has dramatically improved the capabilities of autonomous navigation systems. By emitting laser pulses and measuring their return times, Lidar creates detailed 3D maps of the surroundings [4]. This precision has enabled more sophisticated path planning algorithms that can navigate complex and dynamic environments.

Despite its advantages, Lidar technology presents several challenges. The presence of noise and reflections in the data can lead to inaccuracies. Techniques such as statistical outlier removal [5] and data fusion with other sensors(e.g., cameras, radar) are often employed to enhance reliability [6]. Moreover, real-time processing of Lidar data requires significant computational resources.

Machine learning, particularly reinforcement learning (RL), has opened new avenues for path planning. RL algorithms can learn optimal navigation strategies from Lidar data through trial and error. These models adapt to new environments by continuously improving their performance based on feedback from their actions. Sarsa [7] and Q-learning [8] are two of the most widely used RL algorithms for path planning. These methods rely on tabular approaches to learn the value of actions in specific states, helping the robot to navigate efficiently while avoiding obstacles detected by Lidar and GPS data.

More advanced RL algorithms such as Proximal Policy Optimization (PPO) [9] and Deep Q-Network (DQN) [10] utilize neural networks to handle larger state and actions spaces. PPO optimizes the policy directly while ensuring stable updates, making it robust for complex navigation tasks. DQN, on the other hand, combines Q-learning with deep learning, enabling the robot to learn effective navigation policies from high-dimensional sensory inputs.

The future of path planning continues to evolve with ongoing research in AI algorithms and sensor technology, aiming to enhance accuracy and reliability. Lidar-based path planning is poised to play a crucial role in autonomous navigation across diverse applications.[2].

3 Methodological approach

3.1 Experimental Setup

For the implementation of our path planning system, we used the simulator Webots, a robust open-source simulator, to implement our path planning system. This environment allowed us to create and manipulate four distinct 2x2 maps tailored for different phases of development:

- **Easy Map:** Simplified setup, (Figure 1a), with a single obstacle to facilitate basic navigation and random target location training.
- **Medium Map:** Introduces multiple obstacles to challenge the robot's path finding capabilities and strategy development. (Figure 1b).
- **Test Map:** Designed to assess the robot's ability to apply learned strategies to new, moderately challenging scenarios. (Figure 1c)
- **Challenge Map:** Features a complex layout to test navigation skills under real-world-like conditions. (Figure 1d)

Collision detection is integrated into each map via a bumper sensor, crucial for training the robot to avoid obstacles and teaching the reinforcement learning model to prioritize safer routes.

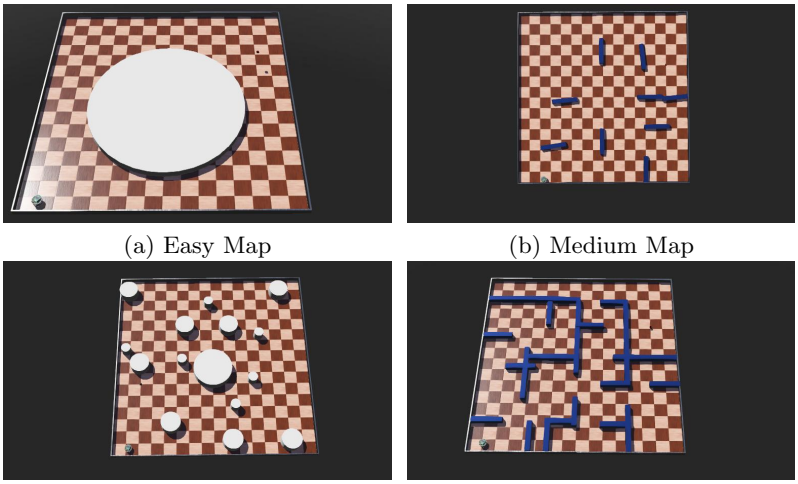
3.2 Sensors

The robot's navigation is supported by these sensors:

- **Lidar:** Essential for detecting and measuring distances to obstacles, allowing the robot to safely navigate by avoiding collisions.
- **GPS:** Used to continuously monitor the robot's current position. This allows the robot to calculate the distance to the destination and determine if it is moving closer to or further from the target.
- **Bumper:** Activates upon contact with an obstacle, triggering penalties that reinforce the importance of safe navigation.

3.3 Environment

We design a custom environment similar to those found in the OpenAI Gym library. This environment is implemented using the gymnasium library and integrates various sensors and devices from the webots simulation framework. Many



(a) Easy Map (b) Medium Map

(c) Test Map (d) Challenge Map

Fig. 1. Webot Maps

functions used in gymnasium environments are adapted to our specific requirements. Below are the key components and functions implemented in our environment:

1. **Initialization:**
   - Initializes necessary variables, including action space, target position, sensors, and essential parameters.
2. **Functions:**
   - **reset():** Reinitializes the environment at the end of each episode, selects a new target position, and resets the reward system.
   - **distance to goal():** Uses GPS to calculate the distance between the robot and the target position.
   - **detect collision():** Uses the bumper sensor to detect collisions.
   - **obstacle proximity():** Uses Lidar readings to prevent collisions by penalizing the robot when it comes too close to an object.
   - **get state(lidar readings):** Constructs the state representation by aggregating Lidar readings and the distance to the target, replacing infinite Lidar values with 100.
   - **get reward():** Adjusts rewards based on several criteria:
     - **Collision Penalty:** Applies a significant penalty if a collision is detected.
     - **Goal Reward:** Provides a substantial reward when the robot reaches the target.
     - **Proximity Penalty:** Penalizes the robot for getting too close to objects or walls.
   - **step(action):** Executes the selected action, updates the environment state, calculates the reward, checks for episode termination conditions,

and returns the new state, reward, episode completion status, and additional information about the distance to the target.

3.4 Models

In this paper, we implement and test four different models to evaluate their effectiveness in handling path planning tasks. The models include two deep reinforcement learning (DRL) algorithms, Deep Q-Network (DQN) and Proximal Policy Optimization (PPO), as well as two simpler reinforcement learning (RL) algorithms, Sarsa and Q-learning. All models use the custom environment previous built for the study.

For the simpler RL algorithms, the environment provides data in a continuous format characterized by coordinates (x,y) from points detected by the Lidar, as well as the distance to the target. To adapt this data to Q-learning and Sarsa, which require discrete values, we develop a methodology to simplify the robot's decision-making process.

Initially, the 'process Lidar points' function, algorithm 1, is applied to reduce the dimensionality of the data grouping the Lidar points and calculating the average x and y coordinates for these groups. This procedure simplifies the data, which is necessary due to memory constrains, while retaining its essential characteristics in a less complex format. Subsequently, the grouped data is converted into discrete states through the 'discretize' function, algorithm 2, which uses a binarization method to map the average coordinates into predefined fixed intervals.

This processing results in a simplified, discretized state space suitable for our Q-table, enhancing the agent's interaction with the environment.

Algorithm 1 Process LIDAR data into reduced number of points with average x and y coordinates

Input: *lidar_data* - Raw LIDAR Data
Output: *reduced_lidar_points* - Processed LIDAR Points

if len(*lidar_data*) $\neq 43$ **then**
    raise Exception("Expected 43 elements in lidar_data: 42 for points (x and y for each) plus one for the distance to the obstacle.")
$num\_groups \leftarrow 3$
$points\_per\_group \leftarrow 14$
$reduced\_lidar\_points \leftarrow \emptyset$
for $i \in \{0, 14, 28\}$ **do**
    $group\_x \leftarrow \text{mean}(lidar\_data[i : i + points\_per\_group : 2])$
    $group\_y \leftarrow \text{mean}(lidar\_data[i + 1 : i + points\_per\_group : 2])$
    $reduced\_lidar\_points \leftarrow reduced\_lidar\_points \cup \{group\_x, group\_y\}$
$distance\_to\_obstacle \leftarrow lidar\_data[-1]$
$reduced\_lidar\_points \leftarrow reduced\_lidar\_points \cup \{distance\_to\_obstacle\}$
return *reduced_lidar_points*

Algorithm 2 Discretize the continuous state into indices

Input: *state* - Current State
Output: *stateIdx* - Discretized State Indexes

if len(*state*) $\neq 7$ **then**
    raise Exception("State should have exactly 7 dimensions based on the lidar input specified.")
$stateIdx \leftarrow \emptyset$
for $i \in \{0, 1, 2, 3, 4, 5, 6\}$ **do**
    $binIndex \leftarrow \text{digitize}(state[i], bins) - 1$
    $stateIdx[i] \leftarrow \max(0, \min(binIndex, \text{len}(bins) - 2))$
return *stateIdx*

These are the models used:

1. **Deep Q-Network (DQN):**
   - **Implementation:** We use the gymnasium version of DQN, which employs a neural network to approximate the Q-value function. This function estimates the expected rewards for each action in a given state.
   - **Code Explanation:** The model initializes a neural network with input dimensions matching the state space and output dimensions matching the action space. It uses experience replay to store and sample past experiences (state, actions, rewards, next state) to break correlation and improve learning stability. The DQN algorithm updates the Q-values using the Bellman equation during training, ensuring that the neural network learns to predict the optimal Q-values.
2. **Proximal Policy Optimization (PPO):**
   - **Implementation:** We implement the gymansium version of PPO, a policy gradient method that optimizes the policy directly while maintaining stable updates.
   - **Code Explanation:** PPO initializes a neural network to represent the policy and value function. It collects a batch of experiences through interaction with the enviroment. The algorithm uses these experiences to update the policy by minimizing the clipped surrogate objective function, ensuring that updates do not deviate significantly from the previous policy, which helps maintain stability and prevents performance collapse.
3. **Sarsa:**
   - **Implementation:** We implement our version of the Sarsa algorithm, a model-free RL algorithm that updates the Q-value based on the action actually taken (State-Action-Reward-State-Action).
   - **Code Explanation:** The model uses an epsilon-greedy policy to choose actions, alternating between exploration and exploitation. In the SARSA algorithm, the Q-values are updated based on the reward received and the Q-value of the next state-action using the equation:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \tag{1}$$

where s and s' represent the current and next states, a and a' the current and next actions, r the received reward, alpha the learning rate and gamma the discount factor. The process repeats until the Q-values converge.

4. **Q-learning:**
   - **Implementation:** Implementation: We implement our version of the Q-Learning algorithm, another model-free RL algorithm that updates the Q-value based on the maximum future reward (off-policy learning).
   - **Code Explanation:** Q-learning begins with a Q-table filled with state-action pairs. Actions are chosen using an epsilon-greedy policy, which alternates between random exploration (with probability epsilon) and exploitation of actions with the highest Q-values. The update rule for Q-values is expressed by the equation:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \max_{a'} Q(s',a') - Q(s,a)] \tag{2}$$

where s and s' represent the current and next states, a and a' the current and next actions, r the received reward, alpha the learning rate and gamma the discount factor. This formula ensures Q-values incorporate the highest future rewards, guiding the agent towards an optimal policy over time. The Q-table is updated continuously through multiple episodes as the agent encounters various states and actions converging to maximizes accumulated rewards.

Each model is trained and evaluated using the custom environment designed to simulate various path planning challenges. By comparing the performance of these algorithms, we aim to identify the most effective approach for autonomous navigation in different scenarios.

The code implementation is present on the github repository

4 Results

The path planning task was evaluated using four distinct maps to train and test the robot's ability to navigate diverse environments. Various rewards were tried, but the final ones with the best results were chosen. All models trained in this study followed a specific reward scheme:

- **Proximity to Obstacles:** A penalty of -10 is imposed if the robot approaches an obstacle within a minimal threshold, promoting safe navigation. No penalty is applied otherwise.
- **Collision:** A significant negative reward of -1000 is assigned if a collision occurs, ensuring safe navigation.
- **Distance to Goal:** A high positive reward of 1000 is provided for reaching the goal, incentivizing accurate navigation.
- **Progress:** Additional rewards of 3 are given for progress towards the goal, while a penalty of -5 is applied if no progress is made.

The deep reinforcement learning models, implemented using Gym and Stable-Baselines3 libraries, showed faster training times compared to Q-learning and

Sarsa. Specifically, PPO and DQN were trained for 2.5 million timesteps per map, while Sarsa and Q-learning were trained for 20,000 timesteps per map. A Lidar sensor with 21 rays and a max range of 3 was used.

The performance of the models in path planning was evaluated using several metrics:

- **Success rate in reaching the goal:** Indicates the effectiveness of the path planning strategy.
- **Average time taken to reach the goal:** Assesses whether the robot follows an efficient and optimal path.
- **Reward received:** Provides insights into the effectiveness of the learned policy by indicating if the robot takes an optimal path.
- **Minimal Distance to Goal:** If the robot fails to reach the goal, this metric indicates the closest proximity to the target achieved, offering insights into the navigation strategy's effectiveness.
- **Distance to Goal:** Similar to the above but Indicates the distance to the goal achieved when the episode finished.

To compare the performance and results of the metrics on the maps across the four reinforcement learning algorithms, (PPO, DQN, Sarsa, and Q-Learning), three tables were structured, each representing a distinct map. The tables are organized as follows:

- **Table 1:** This table presents the analysis of the results on the training map, (Figure 1a). It details the performance metrics for each algorithm, highlighting how well they learned from the training environment.
- **Table 2:** This table focuses on the test map, (Figure 1c), providing insights into how each algorithm performs in a previously unseen environment. The comparison here is crucial for evaluating the generalization capabilities of the algorithms.
- **Table 3:** This table summarizes the performance on the challenge map, (Figure 1d). This map introduces unique and more complex scenarios designed to test the robustness and adaptability of the algorithms under challenging conditions.

Table 1. Metrics for the Training Map

		PPO	DQN	SARSA	Q_Learning
Reach Goal	True	15	4	1	1
	False	10	21	24	24
Reward	Mean	370,88	-172,6	-975,4	-932,36
Mean Min Distance to Goal		0,486271	0,901497	1,279625	1,320329
Mean Final Distance to Goal		0,132671	0,774161	1,142209	1,246111
Mean Distance to close object		0,242587	0,111548	0,155886	0,095928
Time	Mean	6,942102	6,075438	2,700368	0,453273
	Mean when Reach Goal	3,662214	4,456322	3,456432	3,426464

Table 2. Metrics for the Test Map

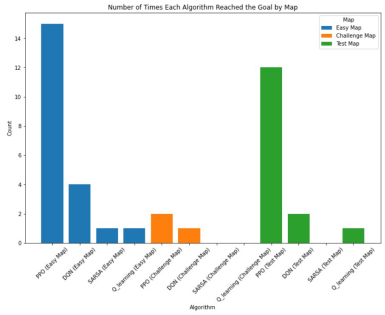
		PPO	DQN	SARSA	Q_Learning
Reach Goal	True	12	2	0	1
	False	13	23	25	25
Reward	Mean	25,28	-783,24	-738,16	-1001
Mean Min Distance to Goal		0,657301	1,070092	1,077644	1,096486
Mean Final Distance to Goal		0,553273	1,021413	0,855381	1,026417
Mean Distance to close object		0,268554	0,248787	0,224085	0,178535
Time	Mean	2,706301	3,057593	5,210403	2,523093
	Mean when Reach Goal	4,879446888	3,865432323	Null	4,565782121

Table 3. Metrics for the Challenge Map

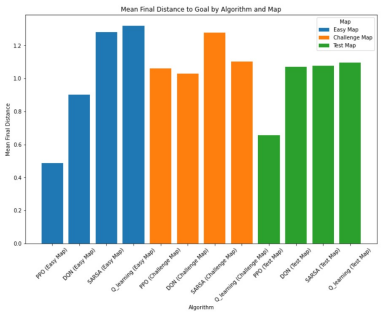
		PPO	DQN	SARSA	Q_Learning
Reach Goal	True	4	1	0	0
	False	21	24	25	25
Reward	Mean	-477,96	-880,96	-977,56	-907,28
Mean Min Distance to Goal		1,062498	1,028902	1,276725	1,103923
Mean Final Distance to Goal		0,956652	0,9858	1,205952	1,055156
Mean Distance to close object		0,150795	0,142065	0,136294	0,183063
Time	Mean	3,113501	0,838512	7,07621	5,627288
	Mean when Reach Goal	4,55432435	5,213214544	Null	Null

In addition to the tables, several graphs were created to visualize the performance of the four reinforcement learning algorithms across the same three maps: easy map, test, and challenge. These graphs provide a more intuitive understanding of the results and highlight key performance metrics. The graphs are as follows:

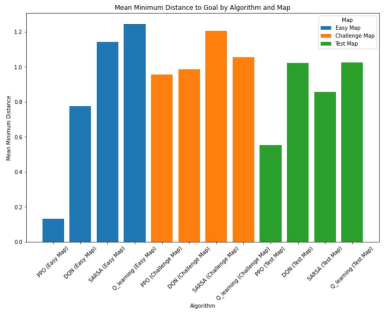
- **Graph 2a:** This graph illustrates the frequency with which each algorithm successfully reached the goal across the training, test, and challenge maps. It provides a clear comparison of the effectiveness of each algorithm in achieving the objective in different environments.
- **Graph 2b:** This graph shows the average final distance to the goal for each algorithm on each map. It helps in understanding how close the algorithms get to the goal on average, indicating their precision and efficiency.
- **Graph 2c:** This graph presents the average of the minimum distances to the goal achieved by each algorithm on each map. It highlights the best performance instances and the ability of the algorithms to get as close as possible to the goal.
- **Graph 2d:** This graph depicts the average time each algorithm takes to reach the goal across the three maps. It is crucial for assessing the speed and efficiency of the algorithms in completing the tasks.



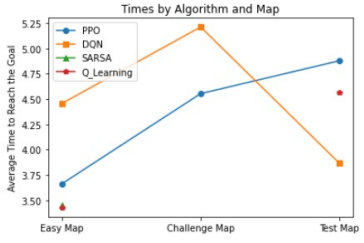
(a) Number of Times Each Algorithm Reached the Goal by Map



(b) Mean Final Distance to Goal by Algorithm and Map



(c) Mean Minimum Final Distance to Goal by Algorithm and Map



(d) Times by Algorithm and Map

Fig. 2. Graphical analysis

5 Conclusions and Future Work

This paper introduces a reinforcement learning (RL) path planning algorithm for navigating robots in complex, dynamic environments. PPO integrates data from multiple sensors to map optimal routes to specified targets. PPO incorporates heuristic constraints for obstacle avoidance and path smoothing, ensuring safe and efficient navigation. Experimental results demonstrate PPO's effectiveness in identifying and navigating paths, showcasing its versatility for various robotic applications.

Despite promising results, areas for improvement include refining the reward function to enhance path finding efficiency and investigating its performance in unknown environments. Future research aims to deploy the algorithm in more challenging settings, conducting extensive testing to identify limitations and enhance robustness for real-time applications. Enhancements in sensor technology and dataset enrichment will further strengthen the algorithm's capabilities for autonomous navigation in diverse and unpredictable environments.

References

[1] Sánchez-Ibáñez, J.R., Pérez-del Pulgar, C.J., García-Cerezo, A.: Path planning for autonomous mobile robots: A review. Sensors 21(23) (2021), https://www.mdpi.com/1424-8220/21/23/7898

[2] Reda, M., Onsy, A., Haikal, A.Y., Ghanbari, A.: Path planning algorithms in the autonomous driving system: A comprehensive review. Robotics and Autonomous Systems 174, 104630 (2024), https://www.sciencedirect.com/science/article/pii/S0921889024000137

[3] Stentz, A.: Optimal and efficient path planning for partially-known environments. In: Proceedings of the 1994 IEEE International Conference on Robotics and Automation. pp. 3310–3317 vol.4 (1994)

[4] Li, J., Zhang, X., Li, J., Liu, Y., Wang, J.: Building and optimization of 3d semantic map based on lidar and camera fusion. Neurocomputing 409, 394–407 (2020), https://www.sciencedirect.com/science/article/pii/S0925231220309693

[5] Saravanarajan, V.S., Chen, R.C., Chen, L.S.: Lidar point cloud data processing in autonomous vehicles. In: 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT). pp. 1–5 (2021)

[6] Ravindran, R., Santora, M.J., Jamali, M.M.: Camera, lidar, and radar sensor fusion based on bayesian neural network (clr-bnn). IEEE Sensors Journal 22(7), 6964–6974 (2022)

[7] Mohan, P., Sharma, L., Narayan, P.: Optimal path finding using iterative sarsa. In: 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS). pp. 811–817 (2021)

[8] Maoudj, A., Hentout, A.: Optimal path planning approach based on q-learning algorithm for mobile robots. Applied Soft Comput-

ing 97, 106796 (2020), https://www.sciencedirect.com/science/article/pii/S1568494620307341

[9] Moon, W., Park, B., Nengroo, S.H., Kim, T., Har, D.: Path planning of cleaning robot with reinforcement learning. In: 2022 IEEE International Symposium on Robotic and Sensors Environments (ROSE). pp. 1–7 (2022)

[10] Wang, W., Wu, Z., Luo, H., Zhang, B.: Path planning method of mobile robot using improved deep reinforcement learning. Journal of Electrical and Computer Engineering 2022(1), 5433988 (2022), https://onlinelibrary.wiley.com/doi/abs/10.1155/2022/5433988