Path planning using Reinforcement Learning

Alexandre Marques, Margarida Vila Chã, and Sebastião Santos Lessa

Faculdade de Ciências of Universidade do Porto,
Rua do Campo Alegre, Porto 4169-007, Portugal.

Abstract. Efficient path planning for autonomous robots in unknown or dynamic environments is a significant challenge with applications across various industries. Reinforcement learning (RL) offers a promising approach to enhance decision-making in these scenarios by enabling robots to learn from their interactions with the environment. This research develops RL-based path planning algorithms for mobile robots using LiDAR data, which provides detailed information about surroundings for informed navigation. We investigate several RL algorithms, including Deep Q Learning (DQN), Quantile Regression DQN (QRDQN), Proximal Policy Optimization (PPO), Trust Region Policy Optimization (TRPO), Advantage Actor-Critic (A2C), and Augmented Random Search (ARS). Each algorithm has unique strengths suited to different path planning challenges. We evaluate the performance of these RL algorithms in partially known environments. It relies on multiple real-time sensor data. By comparing the algorithms' performance, we aim to identify their strengths and weaknesses. Additionally, we focus on the computational efficiency and scalability of the algorithms, crucial for real-time, large-scale applications. Our findings will advance autonomous robotics by providing guidelines for selecting and implementing RL-based path planning algorithms in practical scenarios.

Keywords: Path Planning, Reinforcement Learning, Deep Reinforcement Learning, Autonomous Navigation, LiDAR Readings, Robot Navigation, Simulation

1 Introduction

Path planning involves determining the optimal route for a robot to traverse a set of points while avoiding obstacles. Traditional algorithms like Dijkstra, A*, and Rapidly-exploring Random Trees (RRT) have been used to tackle this problem. These methods, however, face limitations in dynamic or partially unknown environments where real-time adaptability is crucial. Reinforcement learning, which allows agents to learn optimal policies through interaction with the environment, offers a potential solution to these challenges.

1.1 Problem and Motivation

The primary problem addressed in this work is the development of RL-based path planning algorithms that can effectively navigate environments with varying degrees of unknown or dynamic elements. Traditional search and sampling algorithms struggle under these conditions, leading to inefficiencies and increased risk of collisions. By leveraging RL, we aim to create more robust and adaptable path planning strategies.

Solving the path planning problem using RL is highly relevant in real-world applications where robots must operate autonomously in unpredictable settings. Improved path planning can significantly enhance the efficiency and safety of autonomous systems, leading to advancements in fields such as autonomous delivery, emergency response, and industrial automation.

1.2 Hypothesis and Important Questions

This research addresses the following questions: Can reinforcement learning algorithms improve the efficiency and adaptability of path planning in dynamic and partially unknown environments?

This project has this question as it's starting point: do the algorithms work if part/all of the map is unknown? Our hypothesis is that is does.

1.3 Aims and Goals

The goal of this research is to develop and evaluate RL-based path planning algorithms using LiDAR data. We will implement advanced DRL models for that. The goal of the work is to train various models with different algorithms, to be able to traverse a map and reach a final position while avoiding bumping into walls. These models will be trained and tested in simulated environments to assess their performance under various conditions.

The paper is structured as follows: Section 2 provides a review of related work in path planning and reinforcement learning. Section 3 details the methodology, including the design and implementation of the RL algorithms. Section 4 presents the experimental setup and results, highlighting the performance differences between the algorithms. Section 5 discusses the implications of the findings, and Section 6 concludes the paper, suggesting directions for future research.

2 Related Work

In the domain of path planning using reinforcement learning (RL), various studies have explored different methodologies to enhance navigation efficiency and robustness. Reference [1] reveals one study conducted with a robot navigation simulation experiment in a maze scene, using the deep Q-network algorithm. The experiment utilized color images directly as input and generated outputs for three states: straight ahead, right turn, and left turn. Results demonstrated successful obstacle avoidance and self-learning capabilities.

Reference [2] presents another approach using a combination of double DQN and competitive architecture DQN, employing a multi-view strategy with data collected from four cameras. The method effectively verified the experiment's validity. These studies provide insights into the effectiveness of deep reinforcement learning methods in improving navigation capabilities. However, further exploration is needed to address challenges such as poor exploration ability and sparse rewards in environmental state space. Our proposed approach aims to overcome these issues by employing improved deep reinforcement learning with randomized start and goal positions to enhance exploration and adaptability.

3 Methodological approach

The Webots simulator was chosen since it has extensive and detailed documentation, experience due to the anterior work developed in Introduction to Intelligent Robots practical classes and the solution code of the exercises provided by the Professors. Since the goal was to use Reinforcement Learning, the first step was to define the environment.

3.1 Experimental Setup

For the training and the results, it was used 1 map with four sets of obstacles that the robot had to avoid. The map also contains two squares which place is randomized with the starting of each episode:

- Red square : Initial place of Robot;
- Blue square: Goal of each episode.

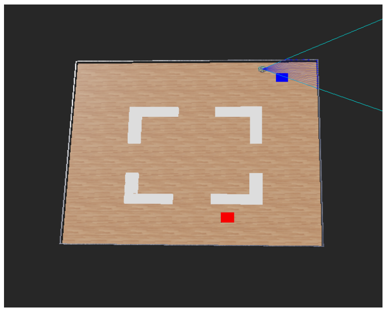


Fig. 1: Map used for training

The observation space is **continuous** and consists of a list with the Lidar Readings. Each position of the list refers to the information of a ray of the sensor, totaling 20 numbers from 0 to plus infinity based of the distance of the closest object they encountered.

The type of the action space is **discrete** and it has **3 different actions**. The actions are:

– Moving forward;
– Moving forward and rotating left;
– Moving forward and rotating right.

The reward system is dense, meaning that the agent receives feedback at every time step rather than only at the end of an episode. This continuous feedback is crucial for guiding the agent's learning process, helping it to understand the immediate consequences of its actions and adjust its behavior accordingly. The rewards are calculated differently based on the state of the episode and certain conditions.

Table 1: Rewards table for Reinforcement Learning training

Rewards		
State of the Episode	**Condition**	**Reward**
Over	Agent reached the goal	25
Over	Agent is too close to an obstacle	-5
Over	Exceeds the number of maximum allowed timesteps	-5
Not Over	Direction of the agent	[-1, 1]
Not Over	Agent's distance to the goal	$[\approx -1,5, \approx 1,5]$
Not Over	Agent's distance to obstacles is smaller than the safe one	-0.01

The number of permitted timesteps for an episode is 5000 and after that number is reached it is considered that the agent loses and is atributted a reward of **-5** similar to the case when it almost hits a wall. If the agent reaches the goal position with a thresold of 0.1, it is considered as a win and it is awarded **25** points. If the episode is not over, the reward is calculated based on a few factors:

– The distance to the goal;
– The agent's direction;
– The number of rays of Lidar that detect objects closer than the safe distance.

The final reward R is calculated using the normalized distance of the agent to the goal:

$$R = f\left(\frac{d_{\text{agent, goal}}}{L_{\text{map}}} \times 100\right) \tag{1}$$

where:

- $d_{\text{agent, goal}}$ is the distance from the agent to the goal,
- L_{map} is the length of the map,
- $f(x)$ is the function used to assign the final reward based on the normalized distance multiplied by 100.

The reward is much higher the closer the agent is to the desired location (blue square).

The direction reward takes into account the angle between the line that passes through the position of the robot and the goal, and the angle that the robot is facing with respect to the bottom edge of the map:

- Goal vector calculation:
$$\mathbf{v}_{\text{goal}} = (x_{\text{goal}} - x_{\text{current}}, y_{\text{goal}} - y_{\text{current}})$$
- Goal vector angle:
$$\theta_{\text{goal}} = \tan^{-1}\left(\frac{y_{\text{goal}} - y_{\text{current}}}{x_{\text{goal}} - x_{\text{current}}}\right)$$
- Compass vector angle:
$$\theta_{\text{compass}} = \frac{\pi}{2} - \tan^{-1}\left(\frac{y_{\text{compass}}}{x_{\text{compass}}}\right)$$
- Angle difference:
$$\Delta\theta = \theta_{\text{goal}} - \theta_{\text{compass}}$$
- Reward calculation:
$$R_{\text{direction}} = \cos(\Delta\theta)$$

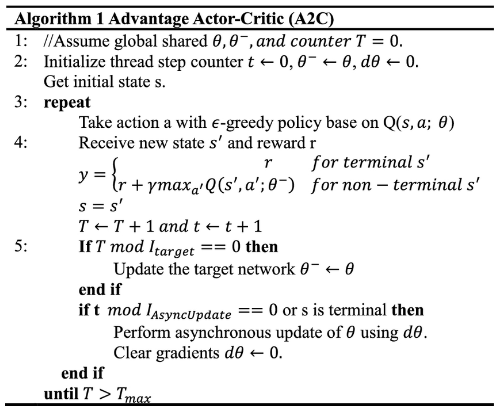
To punish the reinforcement learning agent for being close to obstacles, we subtract -0.01 to the reward when the distance is smaller than the established threshold.

The reward system is designed to provide the agent with **continuous and multifaceted feedback**, facilitating a learning process that balances efficiency, safety, and goal orientation.

When an episode ends, the agent is re-positioned at a random starting position (randomization of red square) and assigned a different goal position (randomization of blue square). This process guarantees enhanced diversity, robustness, and improved exploration capabilities in the reinforcement learning models.

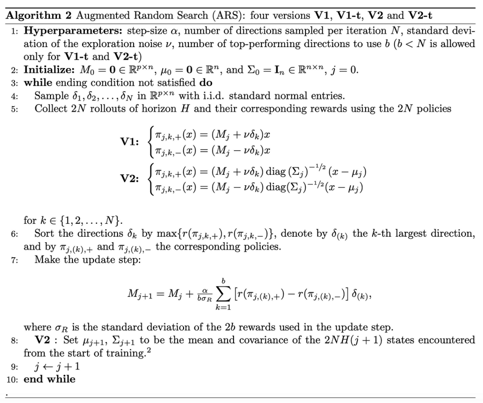
3.2 Algorithms

The models were trained using different algorithms from the Stable Baselines 3 library. Since not all of them are compatible with discrete actions, we chose A2C, ARS, DQN, PPO, QRDQN, and TRPO, which supported the type of actions of the environment.



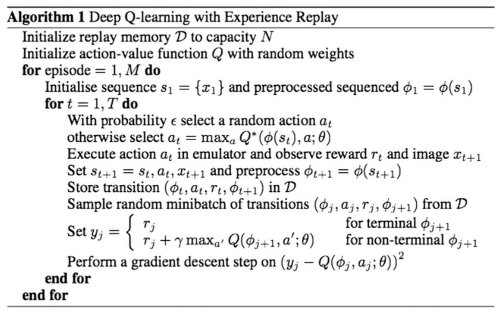
```
Algorithm 1 Advantage Actor-Critic (A2C)
1:  //Assume global shared $\theta, \theta^{-}, and\ counter\ T = 0$.
2:  Initialize thread step counter $t \leftarrow 0, \theta^{-} \leftarrow \theta, d\theta \leftarrow 0$.
    Get initial state s.
3:  repeat
        Take action a with $\epsilon$-greedy policy base on Q($s, a;\ \theta$)
4:      Receive new state $s'$ and reward r
        $y = \begin{cases} r & for\ terminal\ s' \\ r + \gamma max_{a'} Q(s', a'; \theta^{-}) & for\ non-terminal\ s' \end{cases}$
        $s = s'$
        $T \leftarrow T + 1\ and\ t \leftarrow t + 1$
5:      If $T\ mod\ I_{target} == 0$ then
            Update the target network $\theta^{-} \leftarrow \theta$
        end if
        if t $mod\ I_{AsyncUpdate} == 0$ or s is terminal then
            Perform asynchronous update of $\theta$ using $d\theta$.
            Clear gradients $d\theta \leftarrow 0$.
      end if
    until $T > T_{max}$
```

(a) Pseudo-code of A2C



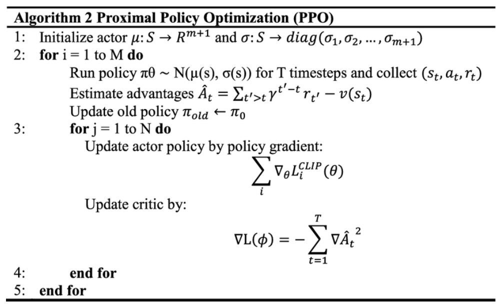
```
Algorithm 2 Augmented Random Search (ARS): four versions V1, V1-t, V2 and V2-t
1:  Hyperparameters: step-size $\alpha$, number of directions sampled per iteration $N$, standard deviation of the exploration noise $\nu$, number of top-performing directions to use $b$ ($b < N$ is allowed only for V1-t and V2-t)
2:  Initialize: $M_0 = \mathbf{0} \in \mathbb{R}^{p \times n}$, $\mu_0 = \mathbf{0} \in \mathbb{R}^n$, and $\Sigma_0 = \mathbf{I}_n \in \mathbb{R}^{n \times n}$, $j = 0$.
3:  while ending condition not satisfied do
4:      Sample $\delta_1, \delta_2, \ldots, \delta_N$ in $\mathbb{R}^{p \times n}$ with i.i.d. standard normal entries.
5:      Collect $2N$ rollouts of horizon $H$ and their corresponding rewards using the $2N$ policies
            V1: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)x \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)x \end{cases}$
            V2: $\begin{cases} \pi_{j,k,+}(x) = (M_j + \nu\delta_k)\,\mathrm{diag}\,(\Sigma_j)^{-1/2}\,(x - \mu_j) \\ \pi_{j,k,-}(x) = (M_j - \nu\delta_k)\,\mathrm{diag}(\Sigma_j)^{-1/2}(x - \mu_j) \end{cases}$
        for $k \in \{1, 2, \ldots, N\}$.
6:      Sort the directions $\delta_k$ by $\max\{r(\pi_{j,k,+}), r(\pi_{j,k,-})\}$, denote by $\delta_{(k)}$ the $k$-th largest direction, and by $\pi_{j,(k),+}$ and $\pi_{j,(k),-}$ the corresponding policies.
7:      Make the update step:
            $M_{j+1} = M_j + \frac{\alpha}{b\sigma_R} \sum_{k=1}^{b} \left[ r(\pi_{j,(k),+}) - r(\pi_{j,(k),-}) \right] \delta_{(k)},$
        where $\sigma_R$ is the standard deviation of the $2b$ rewards used in the update step.
8:      V2 : Set $\mu_{j+1}$, $\Sigma_{j+1}$ to be the mean and covariance of the $2NH(j+1)$ states encountered from the start of training.[2]
9:      $j \leftarrow j + 1$
10: end while
.
```

(b) Pseudo-code of ARS



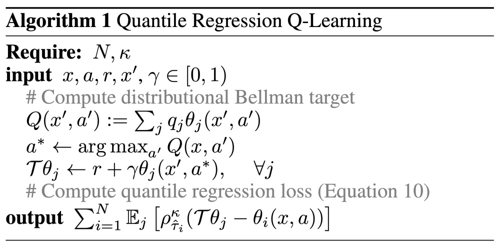
```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$
      end for
  end for
```

(c) Pseudo-code of DQN



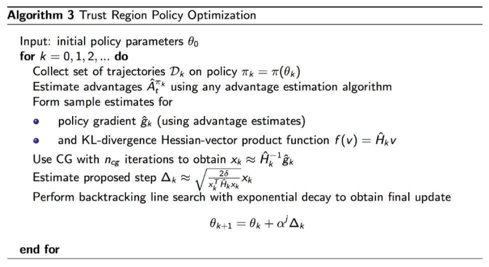
```
Algorithm 2 Proximal Policy Optimization (PPO)
1:  Initialize actor $\mu: S \rightarrow R^{m+1}$ and $\sigma: S \rightarrow diag(\sigma_1, \sigma_2, \ldots, \sigma_{m+1})$
2:  for i = 1 to M do
        Run policy $\pi\theta \sim$ N($\mu$(s), $\sigma$(s)) for T timesteps and collect $(s_t, a_t, r_t)$
        Estimate advantages $\hat{A}_t = \sum_{t'>t} \gamma^{t'-t} r_{t'} - v(s_t)$
        Update old policy $\pi_{old} \leftarrow \pi_0$
3:      for j = 1 to N do
          Update actor policy by policy gradient:
              $\sum_i \nabla_\theta L_i^{CLIP}(\theta)$
          Update critic by:
              $\nabla \mathrm{L}(\phi) = -\sum_{t=1}^{T} \nabla \hat{A}_t^{\,2}$
4:      end for
5:  end for
```

(d) Pseudo-code of PPO



```
Algorithm 1 Quantile Regression Q-Learning
Require: $N, \kappa$
input $x, a, r, x', \gamma \in [0, 1)$
    # Compute distributional Bellman target
    $Q(x', a') := \sum_j q_j \theta_j(x', a')$
    $a^* \leftarrow \arg\max_{a'} Q(x, a')$
    $\mathcal{T}\theta_j \leftarrow r + \gamma\theta_j(x', a^*), \quad \forall j$
    # Compute quantile regression loss (Equation 10)
output $\sum_{i=1}^{N} \mathbb{E}_j \left[ \rho_{\hat{\tau}_i}^{\kappa}(\mathcal{T}\theta_j - \theta_i(x, a)) \right]$
```

(e) Pseudo-code of QRDQN



```
Algorithm 3 Trust Region Policy Optimization
  Input: initial policy parameters $\theta_0$
  for $k = 0, 1, 2, \ldots$ do
      Collect set of trajectories $\mathcal{D}_k$ on policy $\pi_k = \pi(\theta_k)$
      Estimate advantages $\hat{A}_t^{\pi_k}$ using any advantage estimation algorithm
      Form sample estimates for
        •  policy gradient $\hat{g}_k$ (using advantage estimates)
        •  and KL-divergence Hessian-vector product function $f(v) = \hat{H}_k v$
      Use CG with $n_{cg}$ iterations to obtain $x_k \approx \hat{H}_k^{-1} \hat{g}_k$
      Estimate proposed step $\Delta_k \approx \sqrt{\frac{2\delta}{x_k^T \hat{H}_k x_k}} x_k$
      Perform backtracking line search with exponential decay to obtain final update
          $\theta_{k+1} = \theta_k + \alpha^j \Delta_k$
  end for
```

(f) Pseudo-code of TRPO

Fig. 2: Pseudo-code of our algorithms

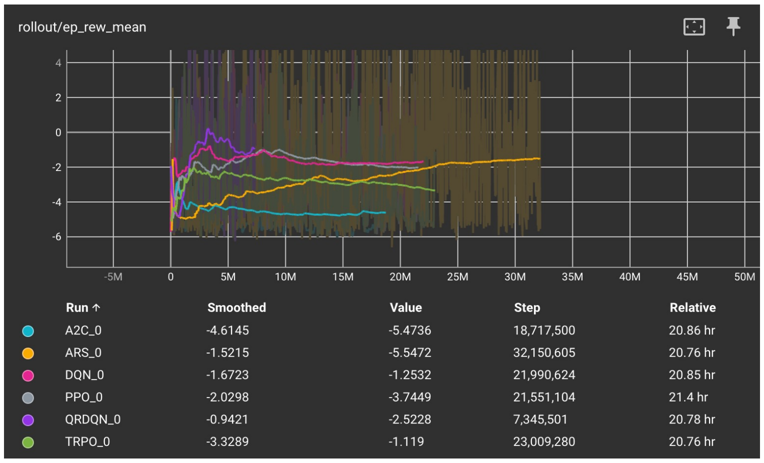
4 Experimental evaluation

In this section, we assess the performance of various reinforcement learning algorithms using the Webots simulator.

Several metrics were employed to evaluate the performance of the reinforcement learning algorithms during both the training and testing phases.

While training the models, we used *TensorBoard* to get access to the information of how the models are performing under several metrics and graphs, as the figures 3 and 4 illustrate. We used the following to get conclusions:

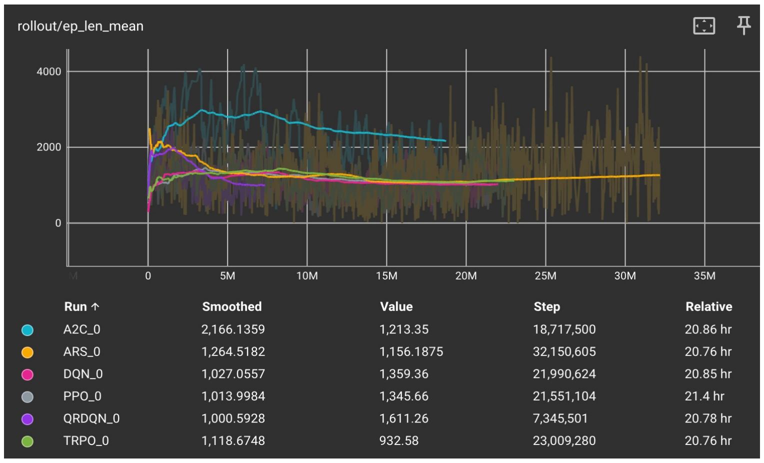
- **Episode Reward Mean**: Displayed as different lines for each algorithm, showing how the average reward changes over time and we can see if it is improving or not, and the biggest value it reached.
- **Smoothed values**: Represents the average reward after applying the smoothing process. This helps in understanding the general trend of the agent's performance without getting distracted by the high variability of individual episode rewards.
- **Number of steps simulated**: The number of training steps each algorithm has taken helps compare the learning progress at different stages. Comparing this to the total training time allows us to see which algorithm is the most efficient in simulating episodes.
- **Time to reach the maximum reward**: Metric that measures the efficiency of reinforcement learning algorithms. It quantifies how quickly an algorithm reaches its highest average reward during the training process. Helps identify the most efficient algorithms that balance high performance with fast learning.



	Run ↑	Smoothed	Value	Step	Relative
●	A2C_0	-4.6145	-5.4736	18,717,500	20.86 hr
●	ARS_0	-1.5215	-5.5472	32,150,605	20.76 hr
●	DQN_0	-1.6723	-1.2532	21,990,624	20.85 hr
●	PPO_0	-2.0298	-3.7449	21,551,104	21.4 hr
●	QRDQN_0	-0.9421	-2.5228	7,345,501	20.78 hr
●	TRPO_0	-3.3289	-1.119	23,009,280	20.76 hr

Fig. 3: Visual Representation of Episode Reward

– **Episode length**: The average number of steps per episode. Shorter episodes might indicate more frequent failures, while longer episodes might indicate better performance or in extreme cases might be a sign that the agent is moving aimlessly.



	Run ↑	Smoothed	Value	Step	Relative
●	A2C_0	2,166.1359	1,213.35	18,717,500	20.86 hr
●	ARS_0	1,264.5182	1,156.1875	32,150,605	20.76 hr
●	DQN_0	1,027.0557	1,359.36	21,990,624	20.85 hr
●	PPO_0	1,013.9984	1,345.66	21,551,104	21.4 hr
●	QRDQN_0	1,000.5928	1,611.26	7,345,501	20.78 hr
●	TRPO_0	1,118.6748	932.58	23,009,280	20.76 hr

Fig. 4: Visual Representation of Episode Length

To test the performance of the models we employed the following metrics:

– **Time to reach the goal**: This metric evaluates how effectively the agent reaches the desired position, by giving an average of the time it took to complete all episodes. A longer time suggests that it did not choose an optimal path and instead took a longer and less optimal way to get to the final position.
– **Distance to the goal**: The average distance that the robot is away from the exact location of the goal position. Bigger values indicate that the robot loses far away from the goal and with smaller ones we conclude that it was closer to complete the task.

4.1 Results

To obtain our results we ran 1000 episodes for each model and saved the results in a *csv file*. Then we used *Pandas* and *Matplotlib* to make the following histograms:

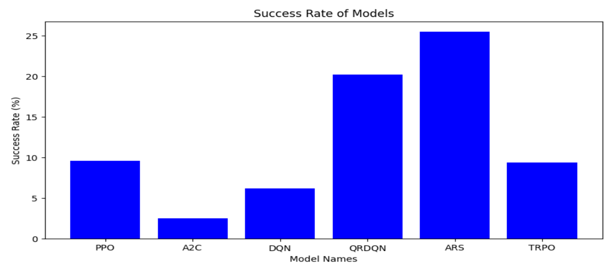


Fig. 5: Success Rate for each model

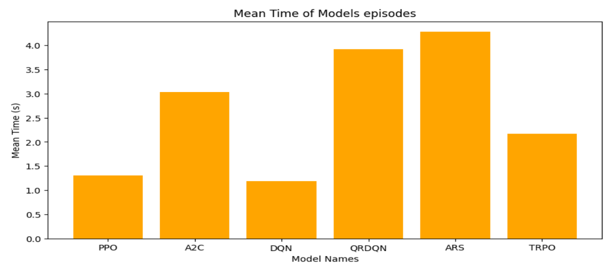


Fig. 6: Mean Time for each episode for each model

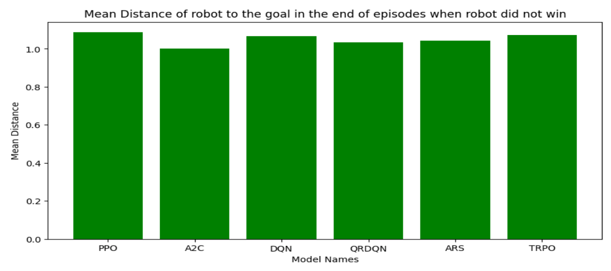


Fig. 7: Mean distance from the goal when agent loses for each model

The results, as visualized in Figures 5, 6, and 7, provide comprehensive insights into the efficacy and efficiency of these models.

The ARS model demonstrated the highest success rate, approaching 25%. This high success rate suggests that ARS is particularly adept at navigating the environment and finding successful paths. The QRDQN model also showed a commendable performance with a success rate near 20%, indicating its capability in handling the path planning task effectively.

In contrast, models like A2C and DQN exhibited significantly lower success rates, around 3% and 5%, respectively. This could be attributed to their potential limitations in handling complex path planning scenarios or the specific characteristics of the environment used in this study. PPO and TRPO models had moderate success rates of around 7% and 10%, respectively. These results suggest that while PPO and TRPO have some capacity for successful navigation, they may require further tuning or additional strategies to improve their effectiveness.

The mean time per episode provides insights into the efficiency of each model. Interestingly, the ARS model, despite its high success rate, took the longest mean time per episode. This indicates that while ARS is effective in finding successful paths, it does so by taking more time to explore and make decisions. This extended time per episode could be a result of ARS's exploration strategy, which might involve more thorough search processes to ensure success.

QRDQN also had a relatively high mean time, which aligns with its high success rate. The trade-off between success rate and time efficiency is evident here, as models that achieve higher success rates tend to take longer to complete each episode.

In contrast, the PPO model had the shortest mean time per episode, suggesting that it makes quicker decisions. However, its moderate success rate indicates that while PPO is efficient in terms of time, it may not be as effective in finding successful paths as ARS or QRDQN. This could be due to PPO's policy update mechanism, which, while fast, might not always converge to the optimal policy in complex environments.

A2C and TRPO models showed moderate mean times, reflecting a balance between decision speed and exploration. DQN had a lower mean time compared to ARS and QRDQN but was higher than PPO, indicating a moderate trade-off between efficiency and effectiveness.

The mean distance from the goal when not reached is an important measure of how close the models get to achieving the objective even when they fail.

This consistency across models indicates that the RL algorithms are learning to navigate the environment effectively, albeit with varying degrees of success. The similarity in mean distances implies that the models have a good understanding of the environment but might need additional refinements to convert near successes into actual successes.

The analysis reveals a clear trade-off between success rate and time efficiency among the models. ARS and QRDQN, while achieving higher success rates, take longer per episode, suggesting a more cautious and thorough exploration strategy.

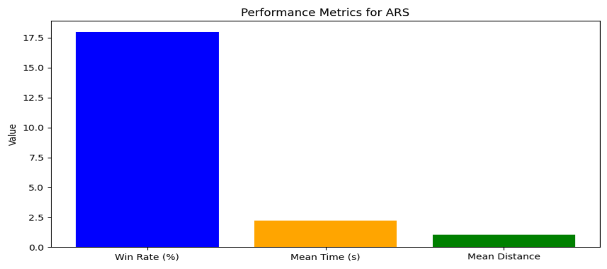


Fig. 8: ARS metrics tested on a random map

Since the ARS performed the best of all models, we decided to test it on a new map. We used a script to change the existing map by moving the walls randomly. After running for 100 episodes we can see the results on the figure 8. When comparing with the earlier metrics we can access that the success rate decreased. This can be due to the new distribution that potentially has less free space and more obstacles on the agent's path. The average time decreased indicating shorter episodes which can be a reflection of the smaller success rate. The mean distance stayed roughly the same indicating that the robot bumps into an object at the same distance of the goal as before.

5 Conclusions

The primary contribution of this manuscript is the comprehensive evaluation of various Reinforcement Learning (RL) models for efficient path planning in robotic applications. The study compared six models—PPO, A2C, DQN, QRDQN, ARS, and TRPO—highlighting their strengths and limitations in terms of success rate, mean time per episode, and proximity to the goal when unsuccessful.

The results demonstrated that ARS and QRDQN models achieved the highest success rates, suggesting their robustness in handling complex path planning scenarios. However, these models required more time per episode, indicating a trade-off between effectiveness and time efficiency. On the other hand, the PPO model exhibited the shortest mean time per episode, making it the most time-efficient, though with a moderate success rate. The consistency in mean distances from the goal across all models indicates a general proficiency in navigating the environment, with potential for further refinements to convert near-successes into actual successes.

In general the success rate was not very good since the starting and ending positions are randomized and the agent has no clear way of knowing where to go. Given this constraints we are content with particularly the result of ARS since it has the most potential of being successful in the proposed task.

When it comes to the performance of the model in an unknown map, we can conclude that the performance is slightly worse and its unclear if it is due to the nature of the map or not.

This work explored the use of RL in a controlled environment, typical of many industrial applications. The experiments conducted in simulation brought to light several key insights and future research directions to enhance the performance of RL-based path planning algorithms for real-world robotic applications.

5.1 Future Work

While this research provides valuable insights into reinforcement learning-based path planning, several areas warrant further investigation to enhance the applicability and robustness of these algorithms. Future work could focus on the following aspects:

- **Multi-Agent Path Planning:** Extending the current algorithms to support multi-agent scenarios where multiple robots need to coordinate and navigate simultaneously. This would involve developing strategies for collision avoidance and efficient task allocation among agents.
- **Algorithm Hybridization:** Combining the strengths of different reinforcement learning algorithms to create hybrid models that can adapt to a wider range of path planning challenges. For example, integrating model-based approaches with model-free reinforcement learning could enhance both learning efficiency and performance.
- **Fine Tune the Models:** Testing the best hyperparameters for the reinforcement learning algorithms could help improve the learning process by optimizing the agent's policy, enhancing its ability to maximize cumulative rewards, and ensuring more efficient and stable convergence during training.
- **Enhanced Reward Systems:** Developing more sophisticated reward systems that can better capture the nuances of path planning tasks. This could include incorporating additional rewards that could help the robot reach its goals or alter the weights and distribution of the existing ones.

References

1. Tai, L., Liu, M.: A robot exploration strategy based on q-learning network. In: Proceedings of the 2016 IEEE International Conference on Real-Time Computing and Robotics (RCAR), pp. 57–62. IEEE, Angkor Wat, Cambodia (2016). doi:10.1109/RCAR.2016.7784012
2. Chen, J., Bai, T., Huang, X., Yang, J., Guo, X., Yao, Y.: Double-task deep q-learning with multiple views. In: Proceedings of the IEEE International Conference on Computer Vision Workshops, pp. 1050–1058. Venice, Italy (2017). doi:10.1109/ICCVW.2017.131