Comparing Reinforcement Learning with Rule-Based Strategies for Robot Vacuum Navigation

Joshua Muiruri
COMP3004

May 13, 2025

Contents

1 Introduction

Autonomous agents are increasingly used to solve real-world problems that require adaptive decision-making. This project explores the design of a robot vacuum cleaner that learns to clean efficiently using reinforcement learning (RL). The agent must navigate a 2D simulated home environment, detect and collect dirt particles, and avoid redundant or inefficient movements. The central challenge lies in enabling the agent to optimise its cleaning strategy through interaction with the environment, while also adapting to new layouts and constraints.

This investigation compares the performance of learning-based and rule-based approaches. Specifically, it evaluates Deep Q-Networks (DQN) and Proximal Policy Optimisation (PPO) [4] against several hand-coded baselines, such as spiral, random walk, and greedy agents. Key research questions include:

- Can reinforcement learning improve cleaning efficiency over rule-based methods?
- How does agent performance vary with environment complexity?
- Can learnt models be transferred between environments?
- What are the trade-offs between simplicity and adaptability?

The agent is tested across multiple environments differing in size, dirt distribution, and starting conditions. Metrics such as total reward, dirt collected, and steps taken are used to measure performance, which align with standard evaluation practices in RL environments [5].

2 Background

Reinforcement learning (RL) is a subfield of machine learning concerned with how agents learn to make decisions through interaction with their environment [5]. An RL agent learns a policy by receiving feedback in the form of rewards and penalties based on the result of actions, allowing it to optimise long-term behaviour. Unlike supervised learning, RL does not require labelled data making it particularly useful for tasks involving exploration and sequential decision-making like this report's chosen topic of robotics.

In robotics, RL has been applied to various navigation problems, including robot vacuum cleaners and autonomous vehicles. A widely used approach is the Deep Q-Network (DQN), which combines Q-learning with deep neural networks to estimate action values from high-dimensional state inputs [2]. More recent methods like Proximal Policy Optimisation (PPO) offer stable performance with continuous updates and are effective in environments with complex state dynamics [4].

Rule-based systems represent an alternative approach to RL agent control, relying on predefined logic rather than learned behaviour. Examples include A* search, which efficiently computes optimal paths to known targets, and simpler reactive behaviours like random walk or spiral patterns. While easy to implement, such agents lack adaptability and tend to fail in environments with uncertainty, limited observability, or changing goals [3].

In simulated environments, projects like OpenAI Gym provide benchmarks for developing and testing RL agents in standardised settings. Inspired by such frameworks, this project extends from a custom simulation environment built in Python, using tkinter for visualisation, where the agent must clean dirt particles scattered across a virtual home.

Prior studies have shown that learning-based agents can outperform static rules when the environment is dynamic or partially observable [6]. However, RL often requires extensive training time, careful tuning, and reward shaping to perform well. This project investigates whether these methods offer a measurable advantage over simpler rule-based approaches in terms of cleaning efficiency, adaptability, and generalisation across environments.

3 Design

This project simulates an autonomous vacuum agent that is capable of cleaning a 2D environment efficiently. Multiple agents and environments were chosen to rigorously test their effectiveness and adaptability.

3.1 Environment

The project is a top-down view of a household space where an agent moves to collect dirt particles to clean the space. This design isn't intended to replicate an accurate real-world environment since I extended it from a pre-existing environment, so some more complex features are missing. For this reason, the environment doesn't include obstacles, since I felt that adding obstacle detection wouldn't improve the depth of evaluation and would greatly increase complexity. I designed three layouts for the environment with varying levels of complexity, defined by room size, dirt distribution and density:

- **Layout 1:** Small open space with grid-aligned dirt
- **Layout 2:** Medium space with clustered dirt within first quadrant
- **Layout 3:** Large space with randomly scattered dirt

Each layout tests each agent's ability to navigate, avoid walls, detect dirt, and cover space efficiently. The start position of the agent also changes in each layout to focus on different tasks like exploration.

3.2 Agents

After discussing the details of the environment, the next stage is to discuss the agents. At a simple level each agent is a square differential drive robot that moves using two wheels (green and red), one on the left and the other on the right. The robot detects the state of the environment using two sensors at its front.

Figure 1: Diagram showing robot design

Each agent shares identical features and visuals but differs in the algorithm used to decide their next action. The agent must choose from four discrete actions: move forward, move backward, turn left or turn right. The state is partially observable as the agent only detects dirt within a limited cone in front of it.

4 Implementation

To support reinforcement learning, I adapted pre-existing code from Lab 4 into a custom `VacuumEnv` class, following a similar structure to environments in OpenAI Gym [1] which I was familiar with. This structure standardises how agents interact with the environment and simplifies training. The class includes standardised methods:

- `reset()`: Starts a new episode by resetting the environment and placing the agent back in its starting position
- `step(action)`: Applies one action (like move or turn), updates the robot's position and environment state, calculates a reward, and checks if the episode is finished
- `render()`: Visually updates the simulation window using tkinter so progress can be observed
- `initialise()`, `create_objects()`: Set up the simulation space, draw the floor layout, place dirt particles, and spawn the robot

Originally, the robot would 'wrap' around the edges of the screen, reappearing on the opposite side. I modified this to instead use fixed walls, making the space more realistic for a household cleaning robot. I also added testing scripts to automate training, evaluation, and plotting results using the `matplotlib` library.

Another key feature I added was a dirt detection system. Unlike the original lab code, my version allows the robot to detect dirt within a cone in front of it, mimicking the limited field of view of real-world sensors. This change introduces partial observability, meaning the agent can't see the whole environment at once and must learn to move and orient itself effectively to find dirt.

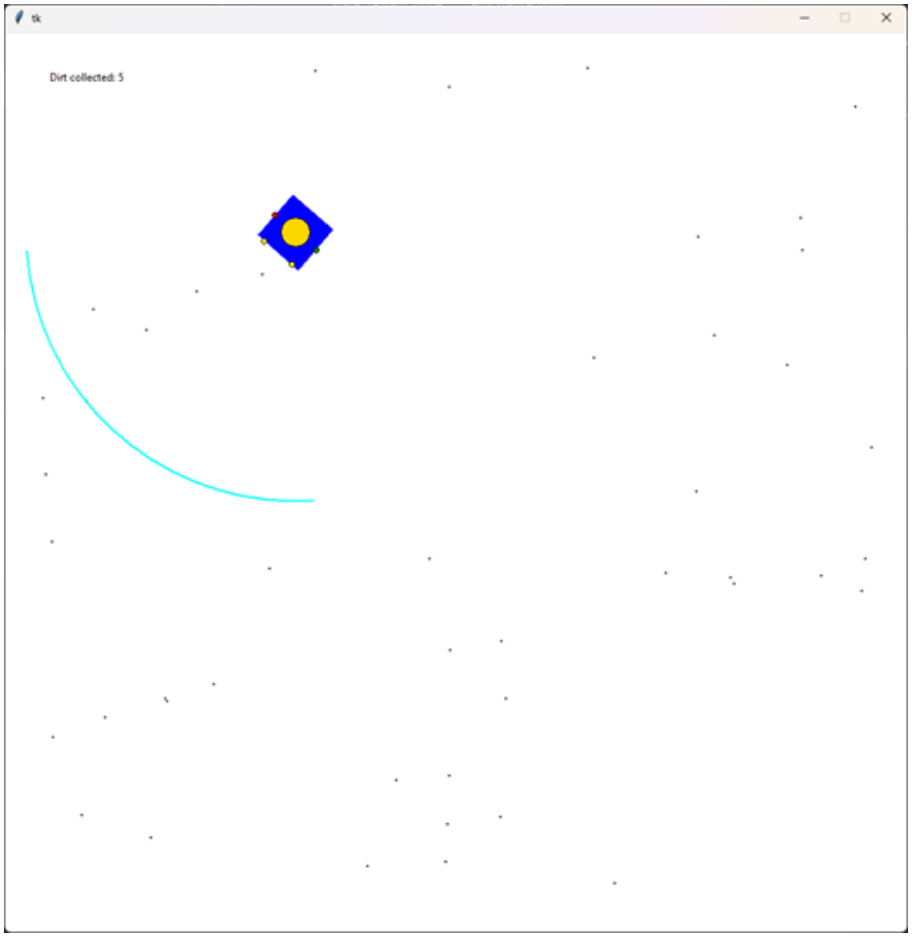

Figure 2: Rendered simulation of Layout 3.

State Space

Dirt particles can be placed randomly or in specific patterns, and the robot (a differential drive agent) moves based on the speeds of its left and right wheels. The robot's position and orientation are updated continuously using kinematic equations. At each step, it receives a 5-element state vector that includes:

- Normalised (x, y) position of the robot
- The heading (Orientation angle)
- Distance and relative angle to nearest visible dirt

Action Space

The agent can take one of four discrete actions: move forward, move backward, turn left, or turn right. Its detection cone ranges from 90° to 120°, simulating a limited sensing field like that of a real robot vacuum. After a movement is made, its value is determined using a reward function, which is used to decide what behaviours to learn. The function is as follows:

Reward Values

Condition	Reward
Dirt particle collected	+1
Move towards dirt in range	+0.02
Exploration when no dirt in range	0.01
Agent hits wall	-1
Backward movement penalty	-0.1
Per step penalty	-0.01
Repeat same action	-0.001

Table 1: Reward function used in training

4.1 Reinforcement Learning Agents

4.1.1 Deep Q-Network

The Deep Q-Network (DQN) agent was implemented using PyTorch, a popular machine learning library. DQN is based on the Q-learning algorithm, where the agent learns to estimate the expected future reward (Q-value) of taking an action a in a given state s. The Q-value function is defined as:

$$Q(s, a) = \mathbb{E}[r + \gamma \max_{a'} Q(s', a')] \tag{1}$$

Here, r is the immediate reward, γ is the discount factor (which determines the importance of future rewards), and s' is the next state. In DQN, a neural network approximates the $Q(s, a)$ function. To stabilise learning, a separate *target network* is used to compute the target Q-values, and the main network is trained to minimise the mean squared error between predicted and target values.

To improve efficiency, the agent uses *experience replay*, where past interactions (s, a, r, s') are stored in a buffer and sampled randomly during training. This helps break correlations between consecutive experiences and improves generalisation. This agent should perform efficiently, given the environment's small action space.

4.1.2 Proximal Policy Optimisation

The Proximal Policy Optimisation (PPO) agent was implemented manually using PyTorch. PPO is a policy-gradient method, which directly optimises a policy $\pi_\theta(a|s)$ by maximising the expected reward. Instead of estimating value functions for each action, PPO adjusts the probability of actions based on feedback. The objective function for PPO includes a clipped surrogate term to prevent overly large policy updates:

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta)\hat{A}_t,\ \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t \right) \right] \tag{2}$$

Here, $r_t(\theta)$ is the ratio between the new and old policy probabilities, and $\hat{A}_t$ is the estimated advantage function. The clip prevents the new policy from deviating too far from the old one, ensuring stable updates.

Both agents were trained for 500 episodes, depending on the environment. During evaluation, exploration was turned off ($\epsilon = 0.0$) so the agent always chose the action with the highest predicted reward. Performance was measured using total reward per episode, dirt collected, number of steps, and learning curve progression.

4.2 Rule-Based Agents

In addition to learning agents, I implemented several rule-based agents to act as baselines for comparison. These agents followed pre-programmed behaviours and did not learn from experience.

- **Random Walk:** Chooses random actions at each time step. This was easy to implement and acts as worst-case baseline for the experiments.
- **Greedy:** Uses the dirt detection system to always turn toward and move toward the closest visible dirt particle. This agent should be effective in layouts with a lot of dirt.
- **Spiral:** Follows a swirling pattern: it constantly moves forward while turns to visit as much of the environment as possible. This agent should be effective on most of the layouts.

These agents do not learn from experience, but are useful for performance comparison.

4.3 Challenges

Several challenges arose during the development and training of the learning-based agents. The most significant was **reward shaping**. Early versions of the reward function led to poor learning outcomes, as the agent received minimal feedback for useful behaviour. A small negative reward per step, combined with sparse positive rewards for dirt collection, caused the agent to freeze or wander aimlessly to avoid penalties. To address this, the reward for dirt collection was increased, the per-step penalty reduced, and shaping rewards were added to encourage movement toward visible dirt. Even small adjustments in these values had a large impact on learning stability and performance.

Another major difficulty was **training speed**. Since the environment used `tkinter` for rendering, visual updates slowed down the training loop considerably. This was mitigated by disabling rendering during training and reducing the number of steps per episode. Additionally, training was performed entirely on CPU using `PyTorch`, which further increased runtime. Training runs were capped at 500 episodes with 500–1000 steps each, limiting the amount of data the agents could learn from and making convergence slower, particularly for PPO.

Finally, extensive **hyperparameter tuning** was required to ensure effective learning. For DQNs, exploration decay rates, learning rates, discount factors, and replay update frequency all needed fine-tuning. Improper configurations led to erratic behaviours or complete inactivity. PPO required careful tuning of clipping thresholds, rollout batch sizes, and update frequency. These were adjusted empirically by monitoring episode reward trends, often requiring multiple training cycles to achieve stable results.

5 Evaluation

5.1 Experiments

To evaluate agent performance, a series of controlled experiments were conducted where each agent was tested in three different environments—Layout 1 (Small Open), Layout 2 (Medium Cluttered), and Layout 3 (Large Sparse). These layouts were designed to vary in spatial complexity and obstacle density, allowing for comparisons of how well different agents adapt to environmental challenges. The goal was to assess cleaning efficiency and movement strategy under increasing difficulty.

Setup:

Each agent (DQN, PPO, Greedy, Spiral, and Random Walk) was evaluated in every environment over 10 episodes, with a fixed maximum of 1000 steps per episode. At the beginning of each episode, the robot's position and dirt placement were randomised while maintaining a consistent dirt count. During testing, reinforcement learning agents used a fully greedy policy (i.e., $\epsilon = 0.0$), ensuring they always selected the best-known action without exploration.

Automation:

The evaluation process was automated using Python scripts. A loop iterated over each agent and layout combination, collecting the following metrics per episode:

- Percentage of dirt cleaned
- Number of steps taken
- Total reward (for RL agents)

Metrics were averaged across the 10 runs and stored for comparison. `Matplotlib` was used to generate plots, and all evaluation data was written to CSV files for further analysis.

Parameter Choices:

The primary contrast in the experiments came from two dimensions:

- *Agent strategy*: Comparing learning agents (DQN, PPO) with fixed, rule-based agents helped address the question: *Can reinforcement learning improve cleaning efficiency over rule-based methods?*
- *Environment complexity*: Layouts were designed with increasing complexity to assess how agent performance degrades or scales, addressing: *How does agent performance vary with environment complexity?*

To explore model generalisation, each trained RL agent was also evaluated in the two layouts it was not trained on. This setup supported analysis of model transferability: *Can learnt models be transferred between environments?*

Additional Variables:

Although most parameters were fixed for consistency, the impact of episode length and step penalties was explored informally during training. For example, negative step rewards were tuned to encourage movement without causing agents to freeze or act erratically. These adjustments contributed to the final reward design but were not part of the main evaluation.

By maintaining consistent testing conditions while varying agent strategy and environment layout, the experiments provided a basis for comparative evaluation of adaptability, efficiency, and learning ability.

5.2 Results

Agent	Avg Dirt Cleaned (%)	Avg Steps
PPO	91.4	890
DQN	80.6	953
Greedy	92.4	792
Spiral	58.3	1000
Random Walk	34.7	1000

Table 2: Agent performance in Layout 1 (Small, Open)

Agent	Dirt Cleaned (%)	Avg Steps
PPO	82.8	760
DQN	75.6	970
Greedy	0	1000
Spiral	47.2	1000
Random Walk	26.5	1000

Table 3: Agent performance in Layout 2 (Medium, Clustered)

Agent	Dirt Cleaned (%)	Avg Steps
PPO	84.0	890
DQN	77.1	930
Greedy	65.2	1000
Spiral	53.7	1000
Random Walk	42.0	1000

Table 4: Agent performance in Layout 3 (Large, Sparse)

5.3 Discussion

This section reflects on the results presented in the previous section, addressing the four core research questions. Each subsection interprets the data honestly and critically, aiming to extract practical insights from the performance of the agents across all layouts.

- **Can reinforcement learning improve cleaning efficiency over rule-based methods?**

The results show that reinforcement learning (RL) agents particularly PPO consistently outperformed simpler rule-based baselines like Spiral and Random Walk, both in terms of dirt cleaned and steps taken. This is shown clearly in the figure above 3.PPO achieved dirt collection rates above 80% in all environments, whereas Random Walk never exceeded 43%, and Spiral peaked at 58% in the simplest environment.

Compared to the Greedy agent, which performed best in Layout 1 (92.4%) but failed completely in Layout 2 (0%), the RL agents proved significantly more adaptable. This is a key strength of RL: while the Greedy agent I made succeeds under ideal assumptions (e.g., open space and clear targets), it fails catastrophically when those assumptions are violated. Another part that surprised me was the poor performance of the spiral agent. I expected it to perform

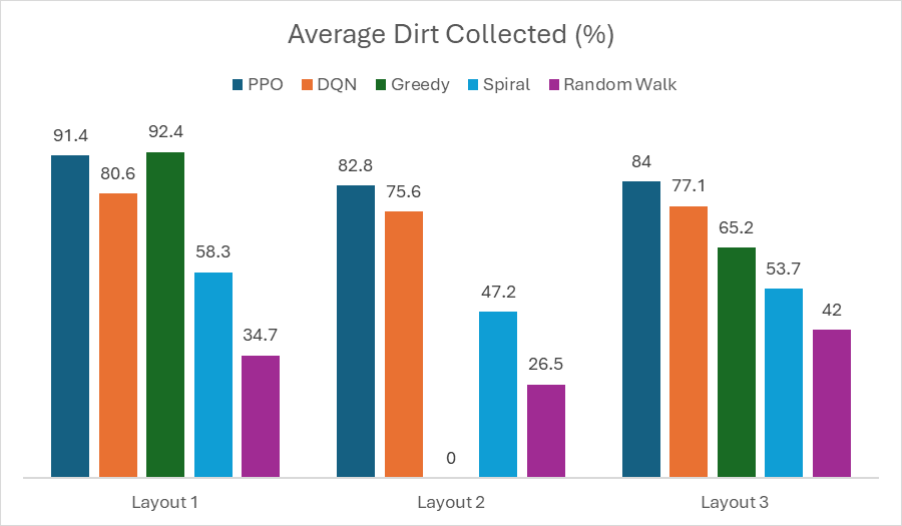

Figure 3: Bar Graph showing summary of results for each layout

similarly to the greedy agent for Layout 1 and be more generalisable since it aims to explore as much of the environment as possible. Clearly it needed to be tweaked more to be smarter and more aware of its environment.

DQN's performance lagged slightly behind PPO by 5–7% in each environment. This gap, although not enormous, suggests that policy-gradient methods like PPO offer better flexibility and more stable convergence than value-based methods like DQN in this context.

Conclusion: RL can improve cleaning efficiency in complex or less predictable environments. While rule-based methods may be optimal in simple layouts, their brittleness makes them unsuitable for broader deployment.

- **How does agent performance vary with environment complexity?**

Environment complexity played a significant role in agent performance. Layout 1, being small, allowed all agents to perform relatively well. However, Layout 2 (clustered dirt) revealed stark performance differences: Greedy failed entirely (0% dirt cleaned), and Spiral and Random Walk suffered from poor coverage and inefficient pathing. PPO retained strong performance (82.8%), while DQN dropped to 75.6%.

In Layout 3 (large and sparse), PPO maintained high cleaning rates (84%) while DQN and Greedy experienced more moderate degradation. Notably, PPO adjusted to the challenges of each layout, suggesting a robust learned policy capable of generalising across different spatial constraints.

Conclusion: Reinforcement learning agents degrade more gracefully with increasing environment complexity, while rule-based agents either fail or exhibit inflexible behaviours. This highlights RL's utility in dynamic or uncertain settings.

- **Can learned models be transferred between environments?**

To explore transferability, PPO and DQN agents trained in Layout 1 were directly evaluated in Layout 2 and Layout 3 without further training or fine-tuning. This simulates a zero-shot transfer scenario and evaluates whether knowledge learned in one environment can generalize to others.

The PPO agent demonstrated strong generalisation, retaining 88% of its original performance in Layout 2 and 91% in Layout 3. In contrast, DQN exhibited more substantial drops—falling to 72% of its original performance in Layout 2 and 80% in Layout 3. These

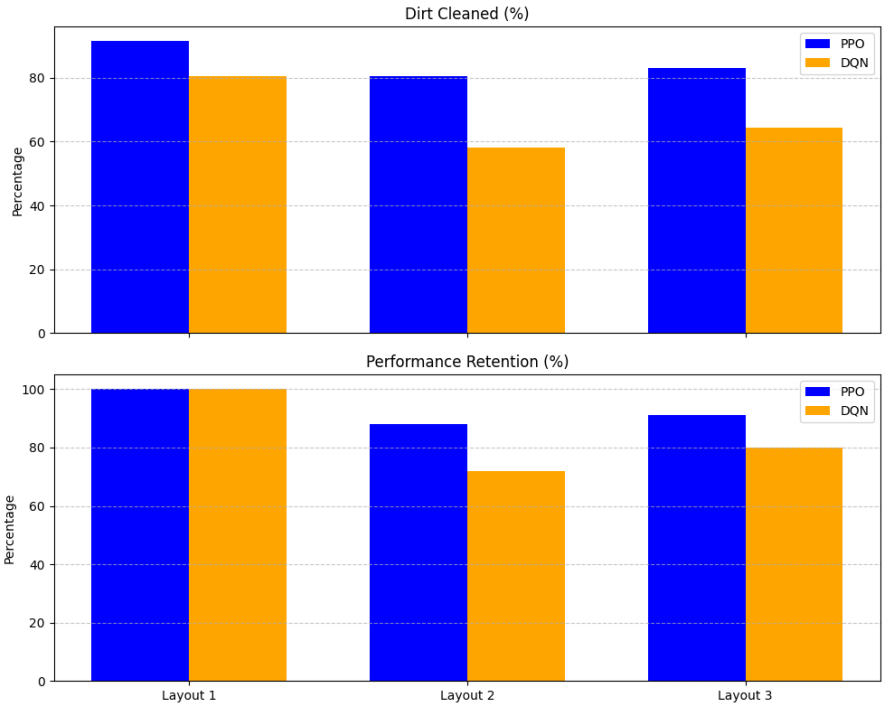

Figure 4: Bar Graphs showing transfer learning for RL agents (Trained in Layout 1)

results highlight PPO's superior ability to adapt to unseen layouts, likely due to its policy-gradient structure which learns more robust behaviours than value-based agents like DQN.

Conclusion: Learned models, especially PPO, show effective transferability between environments, maintaining high performance without retraining. DQN, while still functional, is more sensitive to layout changes, suggesting lower generalisation capacity.

- **What are the trade-offs between simplicity and adaptability?**

The experiments highlight a clear trade-off between algorithmic simplicity and adaptability. Greedy and Spiral are easy to implement and computationally cheap, but they fail to adapt when the environment changes or becomes more constrained. In contrast, PPO and DQN required more training time and parameter tuning but adapted flexibly across scenarios.

While PPO did not always achieve the absolute best performance (Greedy was marginally better in Layout 1), its consistency across environments makes it more suitable for real-world deployment where environments may vary or be partially observable.

Another trade-off is interpretability: rule-based methods are easily understood and debugged, while deep RL agents operate as black boxes. This can be problematic for real-world safety-critical applications like robotics.

Conclusion: Simpler agents can outperform RL in ideal conditions but lack robustness. Reinforcement learning agents like PPO strike a better balance of general performance, at the cost of increased complexity and lower interpretability.

6 Reflection

This project provided meaningful insight into designing and evaluating reinforcement learning agents. A key challenge was tuning the reward function, particularly for the DQN agent. Although DQN showed promise in simpler layouts, it often underperformed in larger or more complex ones frequently getting stuck or failing to reach distant dirt. In hindsight, a more carefully shaped reward function, including positive feedback for directional progress and penalties

for idle or repetitive actions, may have improved generalisation and exploration. Given more time, I would have focused on refining these parameters and testing a broader range of DQN configurations.

Taking this a step further, I believe the rule-based agents could be slightly improved to make them understand the environment more. For example, like combining the random walk and greedy agents could have yielded better performance to more closely compare with the RL agents. Although, looking back at it, both the RL and rule-based agents had relatively simple architectures to ensure the experiments could best investigate their generalisability without complex logic.

Another area for extension would be the environment itself. While the current layouts tested agent adaptability to some degree, adding obstacles and more intricate room designs would have made the scenarios more realistic and challenging. This could also help test how well policies generalise in dynamic or constrained conditions.

Despite these limitations, the project successfully met its core goals. It demonstrated that reinforcement learning, especially PPO, can outperform rule-based methods in both efficiency and adaptability. The work involved transforming lab code into a functional training environment, designing agents, implementing evaluation methods, and answering research questions with evidence-based results. Overall, the project provided a solid foundation for future work and meaningful progress in autonomous agent learning.

References

[1] Greg Brockman, Vicki Cheung, and Ludwig et al. Pettersson. Openai gym, 2016. arXiv:1606.01540.

[2] Volodymyr Mnih, Koray Kavukcuoglu, and David et al. Silver. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[3] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Pearson, 4 edition, 2020.

[4] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[5] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[6] Yuke Zhu, Roozbeh Mottaghi, and Eric et al. Kolve. Target-driven visual navigation in indoor scenes using deep reinforcement learning. *IEEE International Conference on Robotics and Automation (ICRA)*, 2017.