1 Introduction

Autonomous ground robots are increasingly being deployed in cluttered, dynamic environments such as construction sites, disaster-response zones, and large, ever-changing campuses. In these settings, static maps become outdated quickly, sensor noise is inevitable, and hand-tuned path planners that assume perfect knowledge of the environment often fail in unexpected ways. The state of the art in commercial navigation still relies heavily on map-based planners (e.g., variants of A* and D*) tightly coupled with carefully engineered perception pipelines. While these approaches work well in structured settings, they are brittle under layout changes, occlusions, and partial observability, leading to safety concerns, increased engineering effort for constant retuning, and high deployment and maintenance costs.

Our motivation for this project is both practical and educational. Practically, we are interested in understanding whether an RL model, trained purely from local perception without a global map, can achieve reliable goal-directed navigation in "undesirable" terrains that consist of obstacles that the agent will be trained to navigate around. A more robust navigation stack of this kind would have clear downstream impact: safer operation in unstructured environments, and a more scalable path to deployment on ground robots in real-world scenarios. Theoretically and educationally, we chose this problem because it sits at the intersection of reinforcement learning, path planning, and autonomous systems, giving us hands-on experience with Markov Decision Processes(MDPs), reward design and exploration strategies. It also allows us to critically compare modern RL algorithms within a unified experimental framework.

The overall objective of our project is to design, implement, and evaluate a learning-based navigation policy that can guide an autonomous agent through 2D obstacle environments using only perception that consists of the agents position, and the position of the goal. We will benchmark its performance against an A* path-planning baseline. More concretely, our goals are:

1. to build gridworld environments to train and compare several RL agents (Q-learning, DQN, and PPO) on these environments using only local observations

2. to implement and tune an A* planner as a reference method, measuring success rate, path length, and robustness to map perturbations and

3. to implement this into a continuous 2D car-simulation environment that capture key difficulties of undesirable terrains, such as narrow passages and sharp turns. We will evaluate them using, A* deviation ratio and learning curves based on reward functions and analyze how reward shaping, and hyperparameter choices influence stability, sample efficiency, and learned behavior.

Through these objectives, we aim not only to achieve competitive navigation performance, but also to draw clear conclusions about where learning-based methods currently stand relative to classical approaches in terms of technology readiness.

2 Scenario design

The Gridworld environment is a discrete environment of grids, used as a test 'playground' to train RL agents and optimize hyperparameters. The environment, forked from gym_gridworlds on Github, is built on pygame. The position of the agent is given as a grid of numbers

```
0 1 2
3 4 5
6 7 8
```

And is encoded using a matrix wrapper such that it returns discrete values for the position of the agent and the position of the goal on the grid.

```
[np.int64(24), np.int64(154)]
```

This format is chosen as DQN's training is more robust on discretized values. As PPO also supports discrete observation spaces, we use this format for both grids, and train the agents on an MLP neural network.

The agents were trained using Gymnasium and stable-baselines3.

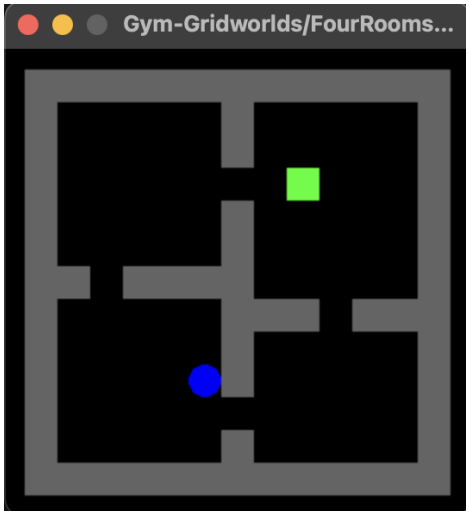
The action space is defined as

```
action_arrows = {0: '←', 1: '↓', 2: '→', 3: '↑'}
```

2.1 Reward Design

Two scenarios were chosen to test the models responsiveness on: Four Rooms and Travel Field.

Four Rooms is designed such that there are sparse rewards in the environment itself, and the model has to rely on distance based rewards to determine a policy for the environment, as seen in Figure 1.



	AGENT	
	GOOD	+1
	BAD	-10
	WALL	n/a
	EMPTY	0

Figure 1: Four Rooms environment layout (left) and rewards (right)

The Travel Field environment, on the other hand, is designed to have comparatively denser rewards, where each tile has either a penalty or a reward, as shown in Figure 2.

2.2 Distance rewards

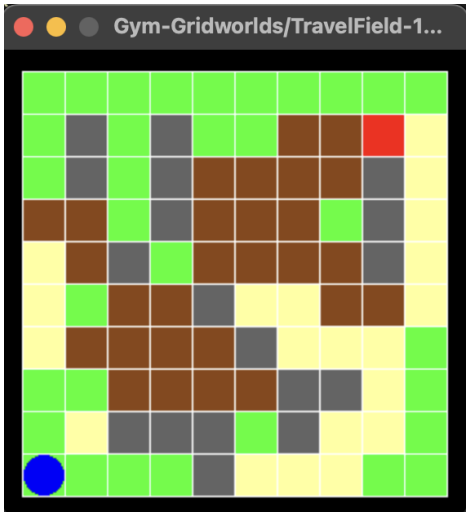
All environments implemented distance based rewards based on the Manhattan Distance:

$$\frac{\text{Manhattan Distance}}{\#\text{ of grid tiles}}$$

3 Machine learning methods

We progress through 3 increasingly sophisticated models, Q-learning, DQN and PPO, and test the performance of these in each environment. As we are looking for optimal results that can be applied to larger, continuous environments, we aim to balance the trade-off between sample efficiency and accuracy when looking at the results.

In small gridworlds, Q-learning returns the most optimal learning curve, and can be used as a benchmark to compare DQN and PPO models with. We introduce neural networks and test their performance to compare how DQNs and PPOs can scale up to larger environments, as Q-learning can be computationally demanding for larger environments. While DQNs cannot scale up to continuous environments, we test its effectiveness in discretized environments. The goal when optimizing the



	AGENT	
	GRASS	-1
	SWAMP	-2
	ROAD	-0.25
	GOAL	+1
	ROCK/WALL	n/a

Figure 2: Travel Field environment layout (left) and rewards (right)

performance of the neural networks is to tune the hyperparameters such that we can get as close to the benchmark of Q-learning as possible.

The three models are tested on two gridworld environments: Four Rooms (as shown in Figure 1) and Travel Field, shown in Figure 2, alongside their respective rewards. The basis for choosing these environments is to test how the models perform with sparse and dense rewards. Four Rooms relies purely on distance based rewards to get the agent to the goal, whereas Travel Field is based on dense rewards, where each tile has a penalty, except for the goal. The

**Note: there are other parameters not mentioned that are defaults in sb3.

3.1 Q-learning

Q-learning is a value-based, model-free and off-policy method, that iteratively updates Q-tables to determine the optimal Q-value for a given environment.

We use the update rule

$$Q^*(s_{t+1}, a_{t+1}) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max Q(s_{t+1}, a') - Q(s_t, a_t)]$$

Q-learning focuses on learning the value function that estimates the reward of different state-action pairs, and chooses the best possible action each time.

3.2 Deep Q-Learning (DQN)

The value-based Deep Q-learning model is good for modeling small, discrete spaces like gridworlds, and builds upon [policies] learned in Q-learning. DQN's are built on Q-learning and are off-policy algorithms that determine the best action based on the max Q-value, and approximates the function using neural networks. Similar to Q-learning, we balance the exploration and exploitation by the exploration fraction.

3.3 Proximal Policy Optimization (PPO)

PPO is a flexible policy gradient, and is fast and stable for a wide range of applications, making it optimal for both discrete and continuous environments, as well as higher dimensional data. Compared to Q-learning and DQN, it is not as sample efficient, although it tends to be more stable as the environment size and inputs increase. As it is an on-policy algorithm, it uses data collected from the most recent data received.

PPO balances exploration and exploitation by adding entropy to introduce randomness to the function.

3.4 Hyperparameters

Using Optuna, a hyperparameter tuning framework, we optimized the hyperparameter values such that the rewards are maximized. The values used are shown in Table 1 and Table 2 for the Four Rooms and Travel Field environments respectively. The following is a description of what parameters were tuned and why they were chosen.

- **Learning rate:** Controls the rate at which the agent updates its weights. The learning rate controls the stability at which the network, is a trade-off between learning quickly and learning efficiently.
- **Discount factor:** the discount factor determines how much the agent values future rewards to immediate ones. This is useful for both sparse and dense rewards, as the agent will value reaching a goal that is further away.
- **Number of steps per episode:** the number of steps is a limit on how many steps an agent can take per episode. We limit this so that the agent can learn the shortest route to reach the goal. This is also useful for termination if the agent gets stuck in a local maxima.
- **Exploration fraction:** Used in Q-learning and DQNs, it determines how often the agent takes a random action. i.e. 0.1 would encourage the agent to take a random action 10% of the time to encourage exploration.
- **Entropy:** Similar to the exploration fraction, entropy introduces randomness to PPO to encourage exploration.

	Learning Rate	**Discount Factor**	**Number of Steps per ep.**	**Exploration Fraction**	**Entropy**
Q-learning	0.0265	0.93	200	0.08	n/a
DQN	0.0007	0.91	200	0.17	n/a
PPO	0.0004	0.8	200	n/a	0.1

Table 1: Parameters for Four Rooms Environment

	Learning Rate	**Discount Factor**	**Number of Steps per ep.**	**Exploration Fraction**	**Entropy**
Q-learning	0.0798	0.94	500	0.06	n/a
DQN	0.0009	0.92	500	0.07	n/a
PPO	0.0007	0.97	500	n/a	0.1

Table 2: Parameters for Travel Field Environment

3.5 Results and Discussion

The results show that overall PPO is a robust and highly versatile option, as it returns relatively stable learning curves. We plot the results for the learning curves for all models, and compare them to determine which model performed well and returned stable results.

In the Four Rooms environment, we can see that DQN learned and stabilized faster than PPO, however PPO outperformed Q-learning while returning the mean values.

In Travel Field, both Q-learning and PPO stabilized to relatively high values. A negative value is expected as the only positive reward in the environment is a goal of +1. PPO stabilized around -24.7

	Mean Value	Num of eval episodes
Q-learning	0.62 ± 0.25	10
DQN	-2.39 ± 3.64	10
PPO	0.53 ± 0.31	10

Table 3: Mean reward value for Four Rooms

	Mean Value	Num of eval episodes
Q-learning	-7.65 ± 6.59	10
DQN	-97.54 ± 92.30	10
PPO	-24.71 ± 51.91	10

Table 4: Mean reward value for Travel Field

while Q-learning performed significantly better, stabilizing around -7.7. DQN returned high negative values, as the Q-value function that it derived was likely not optimal. We can likely attribute this to the reward shaping. Because the rewards were shaped such that there was a negative reward based on each tile, in addition to a distance reward, the max Q value chosen could lead the agent oscillating in a direction away from the goal. A potential solution to this could be to normalize the rewards, such the weight of the penalty of each tile does not outweigh the weight of the distance penalty.

In general, the high standard deviation across all models can be attributed to when the agent cannot find its way to the goal. This is because the state-value function and policies determined by the agents can result in local maxima. We can see this in the value grids in the Appendix, where there are areas where the arrows point towards each other. This will result in the agent oscillating in one location, thereby decreasing rewards for that episode.

3.6 Implementation issues

- There are local minima, found in the plots where if the agent gets stuck there, it will oscillate around there.
- It is worth noting that Optuna is a tool that iterates through all values to return the best (maximized) value for the mean reward. This was not always true for all parameters, as manually testing the parameters (as in the case for PPO travel field), gave a more stable learning curve than optuna's hyperparameter training. Thus, we should note that while the parameters may not be the most optimal, however, they will still be close.

4 Appendix

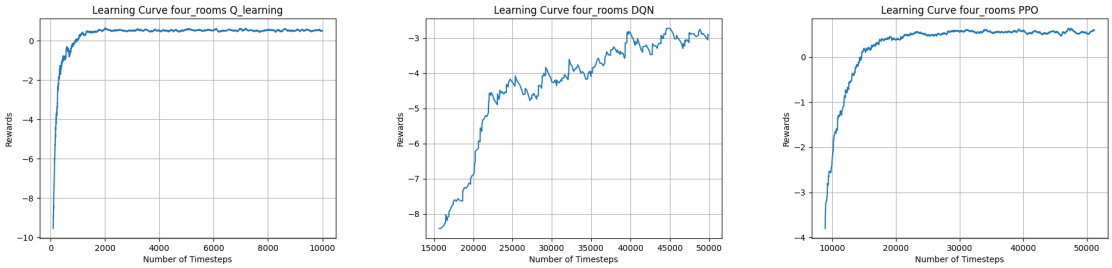


Figure 3: Learning curve for Four Rooms environment for Q-learning, DQN and PPO respectively.

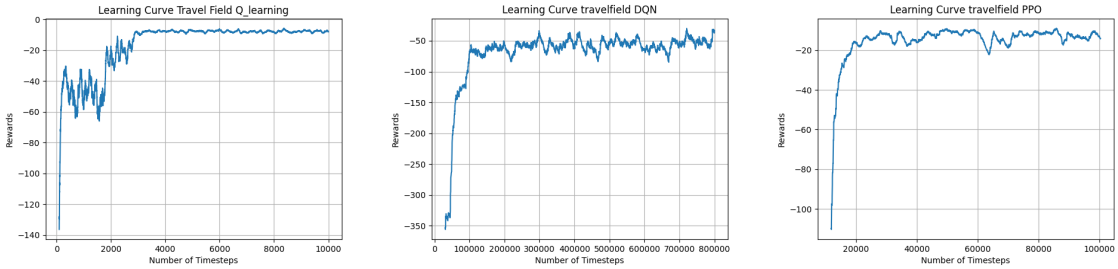


Figure 4: Learning curve for Travel Field environment for Q-learning, DQN and PPO respectively.

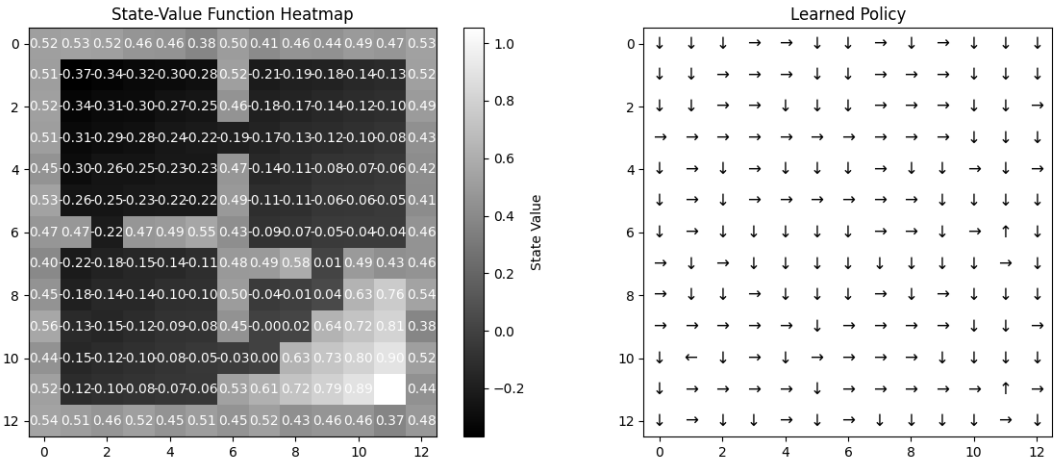


Figure 5: Values at each location for Four Rooms environment determined by DQN by maximizing the state-action values.

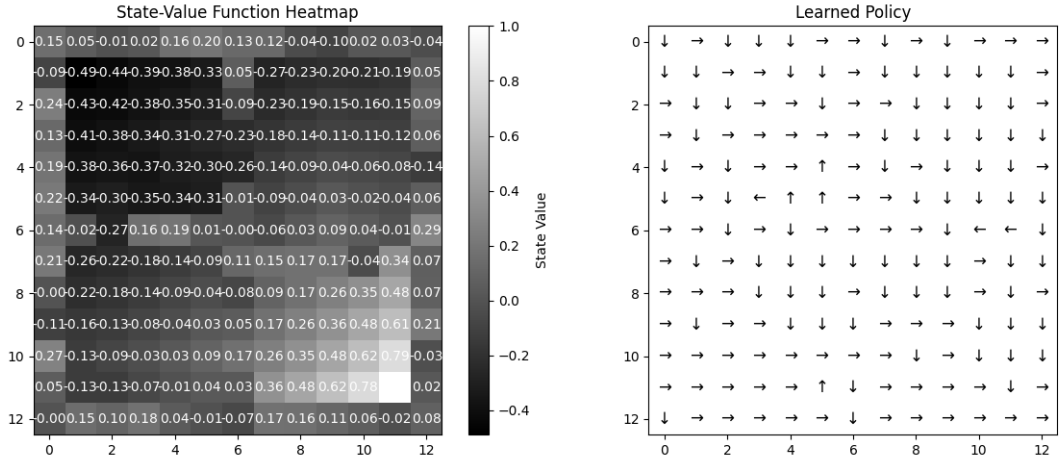


Figure 6: Values at each location for Four Rooms environment determined by the PPO policy function.

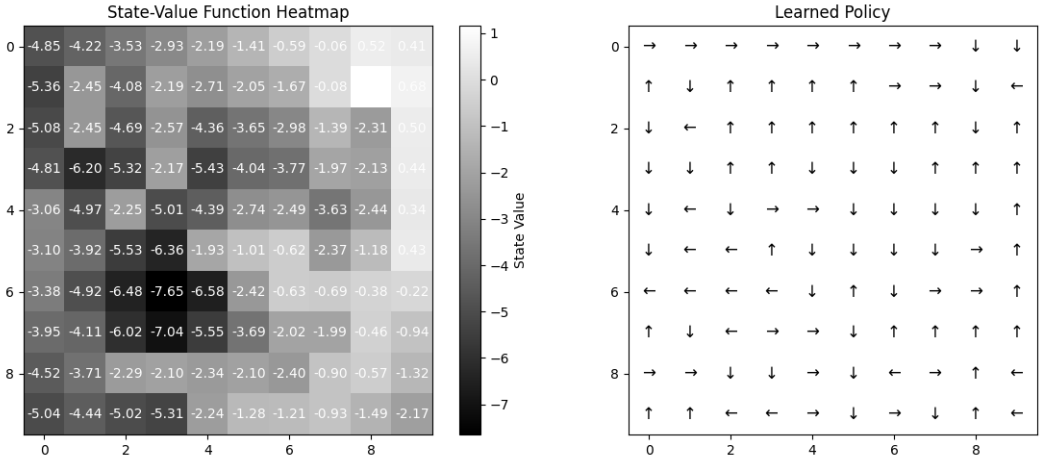


Figure 7: Values at each location for Travel Field environment determined by DQN by maximizing the state-action values.

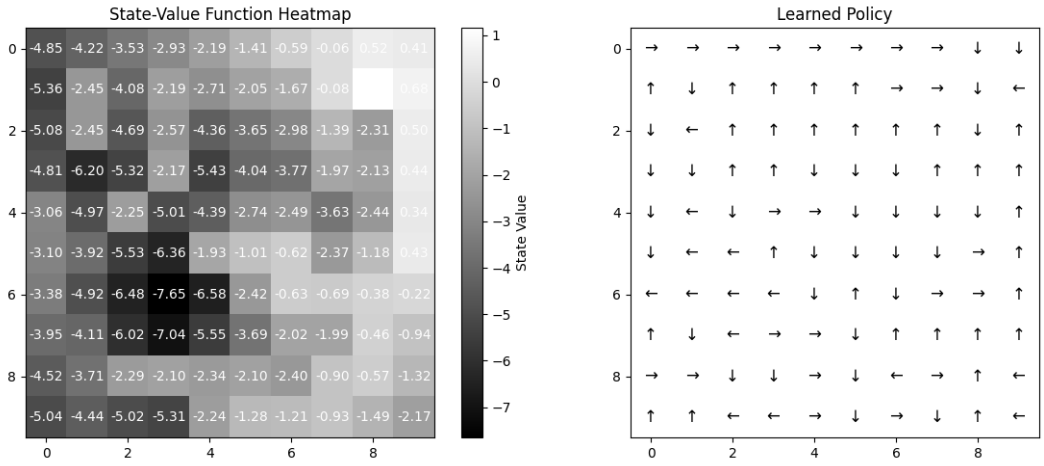


Figure 8: Values at each location for Travel Field environment determined by the PPO policy function.