UNIVERSITY OF LIVERPOOL

BACHELOR'S THESIS

Mapless Navigation with Deep Reinforcement Learning

Author:
Minghong Xu (201601082)

Supervisor:
Dr Murat Üney

Assessor:
Professor Jason Ralph

Abstract

This project explores state-of-the-art model-free deep reinforcement learning (DRL) algorithms and their application to two-dimensional navigation tasks without the use of mapping techniques. While this approach does not provide theoretical guarantees for success rate or generalisation ability, it showcases the versatility of DRL and its potential for robotics. To facilitate research, the project includes the development of a Python library for implementing and testing DRL algorithms. The library has been tested on various simplistic training environments for validating the algorithm implementation. The project includes recorded videos on testing and training and a demonstration program for mapless navigation. While mapless navigation remains a challenging problem, this project demonstrates the potential of DRL and offers a foundation for future research of applying DRL to robotic tasks.

Declaration of academic integrity

I confirm that I have read and understood the University's Academic Integrity Policy. I confirm that I have acted honestly, ethically and professionally in conduct leading to assessment for the programme of study. I confirm that I have not copied material from another source nor committed plagiarism nor fabricated, falsified or embellished data when completing the attached piece of work. I confirm that I have not copied material from another source, nor colluded with any other student in the preparation and production of this work.

14th July 2023

Contents

1 Introduction **4**

1.1 Project Specifications 4

1.2 Deep reinforcement learning 6

1.3 Mapless navigation 8

2 Literature Survey **9**

3 Industrial Relevance, Real-world Applicability, and Scientific Impact **11**

3.1 Industrial relevance 11

3.2 Real-world applicability 11

3.3 Scientific impact 12

3.4 Conclusion 12

4 Theory **13**

4.1 Preliminaries 13

4.2 Policy Gradient 14

4.3 Actor-Critic 14

4.4 Deep Deterministic Policy Gradient (DDPG) 14

4.5 Twin-delayed Deep Deterministic Policy Gradient (TD3) 16

4.6 Soft Actor-Critic (SAC) 16

5 Design **18**

5.1 Algorithm library 18

5.2 Training environment 19

6 Experimental Method **21**

6.1 Integration test of the library 21

6.2 Training policies 21

7 Results **23**

7.1 Integration test 23

7.2 Trained policies 23

8 Discussion **24**

8.1 Escaping and sparse reward 24

8.2 Partial observability 25

8.3 Transfer learning 25

8.4 Catastrophic forgetting 26

9 Reflection on Learning **27**

9.1 Improvement in logging practices 27

9.2 Comprehensive specification report 28

9.3 Advance report drafting 28

9.4 Foundational knowledge acquisition 28

10 Conclusions **29**

References **30**

Appendices **33**

A Original Project Plan **33**

List of Figures

1 Typical model for reinforcement learning paradigm [1] 6

2 A non-exhaustive taxonomy of reinforcement learning algorithms [2] . . 7

3 SLAM-based autonomous navigation in Gazebo Classic visualised with RViz. 8

4 UML class diagram of the Soft Actor-Critic algorithm 18

5 Aerial view of the training environment in Gazebo Classic. 19

6 Environments with obstacles 20

7 Episodic return of TD3 in MuJoCo training environments. The abscissa is the episode count, and the ordinate is the return. The algorithm converges in all six environments. This indicates that the implemented algorithm is effective. 23

8 Episodic return curve is fluctuated between the highest and lowest return. 25

9 Catastrophic forgetting of a TD3 agent trained on the MuJoCo InvertedDoublePendulum environment. 26

10 A typical daily log [27]. 27

List of Tables

1 Specifications verification metrics [1] 6

1 Introduction

Mapless navigation with deep reinforcement learning is an innovative approach to autonomous navigation that aims to empower robotic systems with the ability to traverse and explore their environments without relying on pre-built maps or explicit mapping processes. The motivation behind this project is to reduce the dependency on prior knowledge of the environment and enable more adaptive navigation in dynamic and unpredictable settings. The project has successfully demonstrated that, by employing deep reinforcement learning, robots can acquire intelligent behaviours for navigating simple cluttered environment. The code and demonstration videos are available on `https://github.com/MinghonZi/final-year-proj`.

The report is organised as follows: In Section 1.1, the project specification is given. Section 1.2 and 1.3 is the background on deep reinforcement learning and mapless navigation. Section 2 is the literature survey on DRL and mapless navigation. Section 3 details the industrial relevance, real-world applicability, and scientific impact. Section 4 explains the theories used in this project. The project's software library design is given in Section 5. Section 6 details the experimental method used in this project. Section 7 reports the results. Section 8 provides a discussion of the results and DRL, in general. Section 9 reflects on this project-based learning. Finally, Section 10 concludes the report.

1.1 Project Specifications

The goal of this project is to gain insights into deep reinforcement learning by i) implementing the algorithm and a simulation environment, ii) training policies, and iii) performance evaluation [1]. The engineering tasks of this project mainly consists of implementing the algorithm and the environment, as well as training the policies. Therefore, this is a software engineering project that aims to enable research work in the field of deep reinforcement learning.

For this project, mapless navigation was chosen as a case study for deep reinforcement learning. From the perspective of deep reinforcement learning, any task that it can solve is nothing more than a dataset described by a Markov decision process. Therefore, any task that can be formulated as a Markov decision process is a potential candidate for this project. Initially, quadrupedal locomotion learning was considered, but due to its high engineering complexity, which is close to the level of doctoral research, the idea was eventually abandoned in favour of mapless navigation on account of time constraints.

Mapless navigation is a robotic task that requires the robot's motion planner to navigate the robot to a target location without relying on a map, while avoiding obstacles along the way. In the framework of deep reinforcement learning, the motion planner here can simply be modelled by a policy neural network. The performance of the motion planner can be gradually improved by updating its model weights according to a predefined reward signal through training using reinforcement learning algorithms.

The main line of this project was planned step by step. The first step was reading the literature to get more familiar with the algorithms. The next step was to implement

these algorithms and group them as a software package. The following step developed a simulation training environment for mapless navigation. In the final step, the algorithms were combined with the environment to train policies and gradually increase the complexity of the environment depending on the performance at each stage. The focus is on gaining a deeper understanding of reinforcement learning through the above process.

The project objectives are hence three-fold:

1) Implement deep reinforcement learning algorithms and a simulated mapless navigation training environment.
2) Train policies that move an agent towards a target on a level ground with no obstacles.
3) Policy training for the case with obstacles.

Tasks and deliverables encompass both research and engineering efforts. They are detailed below which will satisfy the objectives set out. The verification metrics are given in the Table 1.

Task 1, **literature survey**: This task will review the recent work on mapless navigation that use deep reinforcement learning. The outcome will be a brief report on the topic (D.1).

Task 2, **library implementation**: State-of-the-art deep reinforcement learning algorithms, including DDPG, TD3, and SAC, will be implemented and grouped together as a Python package (D.2).

Task 3, **training environment implementation**: A flat floor indoor environment will be implemented in the Gazebo Classic simulator. The environment can be completely obstacle-free, may contain some fixed obstacles, or may include randomly moving obstacles, depending on the performance of the training at each stage to gradually increase the complexity of the environment (D.3).

Task 4, **training near-optimal policies**: The policies will be trained by using the D.2 and D.3 until they reach their near-optimal point (the episodic return stops increasing). The performance of the near-optimal policies will be illustrated by episodic return curves and visualised in demonstration videos and a rendering program (D.4).

Task 5, **report insights gained**: The research outcomes will be reported in the project final report (D.5).

D. 1: Brief report on the recent work on mapless navigation using deep reinforcement learning.

D. 2: A library implementation.

D. 3: A training environment implementation.

D. 4: Trained policies with illustration and visualisation of their performance.

D. 5: Report the insights on deep reinforcement learning gained through experience.

Table 1: Specifications verification metrics [1]

Specification	Verification
Implement state-of-the-art deep reinforcement learning algorithms.	Test on baseline environments.
Train policies in an empty world.	Robot can reach the target position.
Train policies in an indoor environment filled with obstacles.	Robot can reach the target position and not collide with obstacles.

The project was expected to take seven months to complete. To ensure the success of this project, a timeline was established to keep track of the progress of each task. The timeline included specific milestones and deadlines for each task, as well as regular meetings to discuss the progress and make any necessary adjustments. The timeline was partially represented as a Gannt chart. See Appendix A.

1.2 Deep reinforcement learning

Reinforcement learning is a machine learning paradigm that optimises a function implementing a policy for a specific agent in a particular environment through trial-and-error experiences [2]. Deep reinforcement learning (DRL) is a type of reinforcement learning that uses deep neural networks, artificial neural networks (ANN) composed of multiple layers, for function approximation [3].

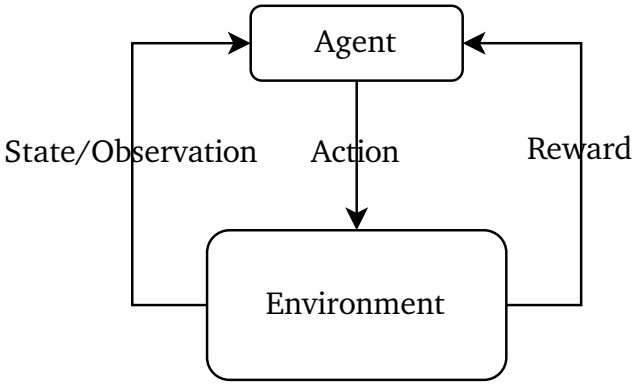


Figure 1: Typical model for reinforcement learning paradigm [1]

As illustrated in Figure 1, the agent learns to act by iteratively observing its own state and the surrounding environment, taking actions, and receiving rewards for those actions. At each time step t, with a given state $s_t \in S$, the agent takes an action $a_t \in A$ according to its policy $\pi\colon S \to A$, receives a reward $r(s_t, a_t)$, and transitions to a new state based on $p(s_{t+1} \mid s_t, a_t)$. This sequential decision-making process is formally modelled as a Markov decision process (MDP), which is a quadruple $\langle S, A, p, r \rangle$, where:

- S, the state space, is a set of all valid states,
- A, the action space, is a set of all valid actions,

- $p\colon S \times A \to \Delta S$, a transition probability function representing the probability of transitioning to the next state if a specific action is taken in the current state,
- $r\colon S \times A \to \mathbb{R}$, a reward function generating a reward signal.

The objective is to find an optimal policy that maximises the cumulative reward accumulated through each state.

$$\max_{\pi} \sum_{t} \mathbb{E}_{s_t \sim p_\pi, a_t \sim \pi} \left[r(s_t, a_t) \right]$$

Numerous algorithms have been proposed to address this problem, which can be broadly divided into:

- model-based, which utilises an environment model predicting state transitions,
- model-free, which is based on the common case where the model is not available to the agent.

Figure 2 presents a rough algorithm classification and representative algorithms under each classification. Note that some algorithms may possess multiple attributes.

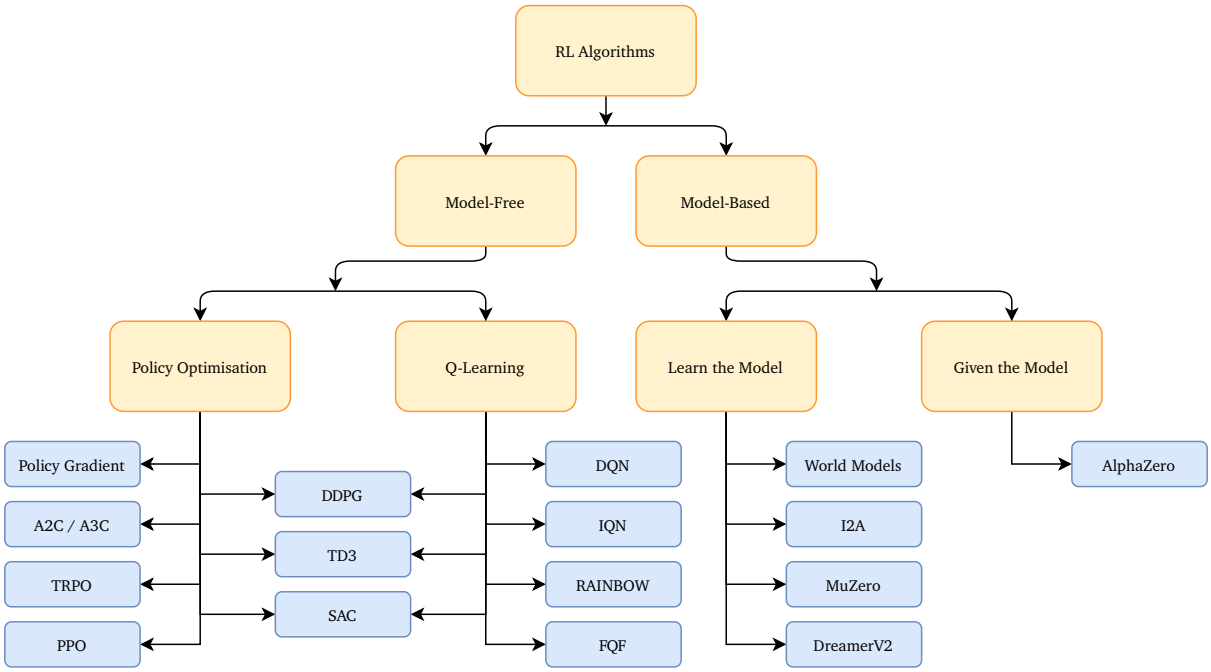


Figure 2: A non-exhaustive taxonomy of reinforcement learning algorithms [2]

The two main approaches for model-free algorithms are Policy Optimisation and Q-Learning. Policy Optimisation is stable and reliable but sample inefficient, while Q-Learning is sample efficient but more likely to fail due to its unprincipled nature. Of the three state-of-the-art model-free DRL algorithms, PPO, TD3, and SAC, PPO uses only the first approach, while the latter two are a mixture of both approaches.

In model-based DRL, the agent learns or is given a model of the environment's dynamics and uses this model for planning and decision-making. With a learned environment model, the agent can plan its actions by simulating different trajectories or sequences of actions in the model. Various planning algorithms, such as Monte Carlo Tree Search (MCTS) or Model Predictive Control (MPC), can be used to find the best action sequence. By explicitly exploiting a model of the environment, model-based DRL has the potential to achieve better sample efficiency and generalisation compared to model-free methods [4]. However, it also faces challenges, including the difficulty of learning accurate environment models and the computational complexity of planning algorithms.

1.3 Mapless navigation

DRL can be applied to accomplish robotic tasks, such as mapless navigation. Mapless navigation is a type of motion planning task that enable a robot to navigate its environment without relying on pre-built maps or prior knowledge of the area [5]. Instead, the system utilises various sensors, such as cameras, LiDAR, or other depth-sensing devices, to perceive its surroundings in real-time and plan its motion accordingly.

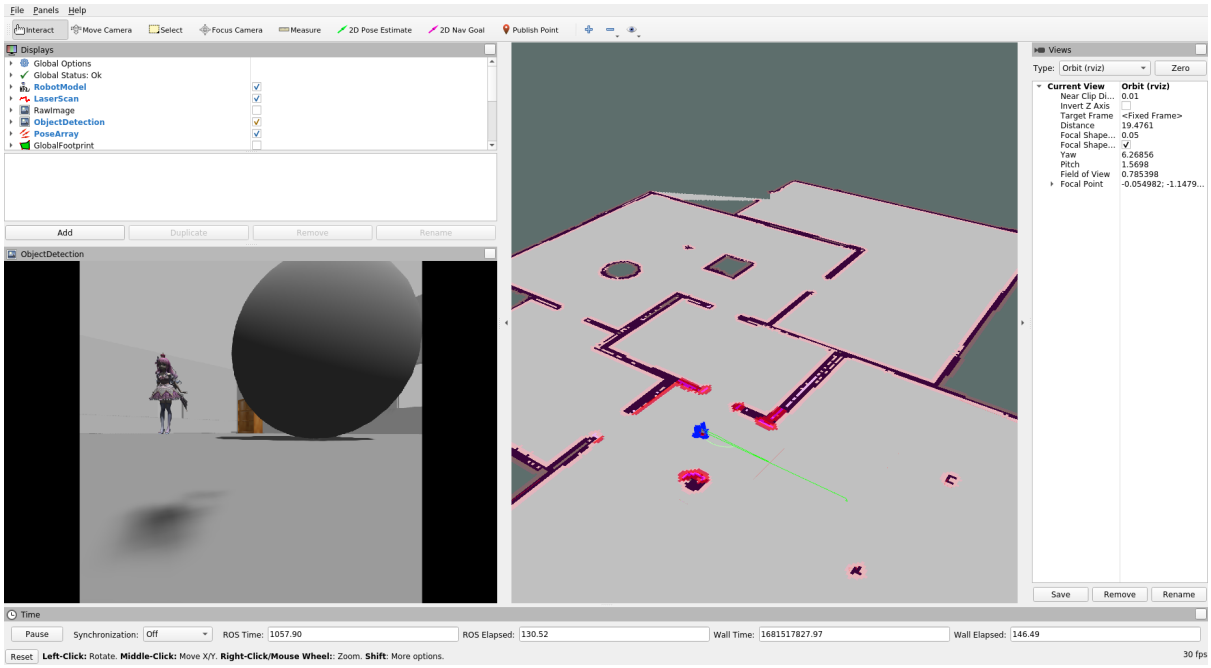


Figure 3: SLAM-based autonomous navigation in Gazebo Classic visualised with RViz.

Building and updating global obstacle maps is time-consuming and relies on precise depth sensors. Figure 3 shows autonomous navigation after building a map using SLAM in a simulation environment. DRL can offer a relatively low-cost solution to this task with the help of emerging localisation methods such as Wi-Fi localisation. Moreover, the mapless navigate ability is particularly useful in situations where maps are unavailable, outdated, or impractical, such as in unpredictable terrains or rapidly changing environments.

2 Literature Survey

This section aims to provide an overview of some key publications and developments in the field of DRL, as well as review works similar to this project.

[6] is an introductory-level textbook first published in 1998 that provides a comprehensive introduction to reinforcement learning, including foundational concepts, algorithms, and techniques. [7] treats the topic more formally, including rigorous proofs. [2] is an informal online educational resource covering basic concepts, key literature, and various baseline algorithms.

The emergence of Q-learning [8] represented a significant advancement in the field of Reinforcement Learning during its early stages. Later, [9] introduced Vanilla Policy Gradient (VPG), which forms the foundation for many later DRL algorithms. The paper demonstrates how policy gradients can be used with function approximation to learn policies in high-dimensional spaces.

[3] introduced the concept of using deep neural networks to approximate functions for reinforcement learning. This paper proposed the first DRL algorithm called Deep Q-Network (DQN), which approximates a state-value function from Q-Learning with a neural network, and demonstrated its effectiveness by training an agent to play Atari games. Two years later, [10] extended the approach in the previous paper to achieve human-level performance on a suite of Atari games. However, the major shortcoming of this approach is that it relies on the Q-learning framework, which can only be applied to reinforcement learning problems in discrete spaces.

[11] introduced Deterministic Policy Gradient (DPG), a policy gradient algorithm with a deterministic actor-critic architecture in which the critic estimates an action-value function while the actor ascends the gradient of the action-value function. [12] proposed the Deep Deterministic Policy Gradient (DDPG) algorithm, which combines the actor-critic architecture with deep neural networks to approximate both the policy and the value function. This extends DQN to continuous action spaces. DDPG is shown to be capable of learning control policies for a range of high-dimensional continuous control tasks. Later, [13] introduced twin critics technique and policy update delay technique to mitigate value function approximation error, ensuring the policy update at each step is not too large and does not cause unstable performance. The algorithm formed by applying these two techniques to DDPG is called Twin-Delayed DDPG (TD3).

[14] introduced a policy gradient algorithm named Trust Region Policy Optimisation (TRPO) that uses trust regions to ensure that policy changes are not excessively significant. [15] simplifies TRPO by using a clipped surrogate objective while retaining similar performance. The simplified algorithm is known as Proximal Policy Optimisation (PPO).

[16] proposed Stochastic Value Gradients (SVG), a generalisation of DPG, which introduces the idea of stochastic policies. The contribution of this work represents a stepping stone between DPG and maximum entropy methods. Based on SVG, [17] proposed Soft Actor-Critic (SAC) that balances a trade-off between maximising the expected return and maximising the entropy of the policy. In the same year, the original SAC adopted the twin critics technique from TD3 and improved the trade-off strategy by

automatically adjusting the ratio of expected return and policy entropy during training. This work is presented in [18].

AlphaGo [19] combined two deep neural networks with Monte Carlo tree search (MCTS), using both supervised learning and reinforcement learning to train the networks, to master the game of Go. Its successor, AlphaGo Zero [20], improved upon AlphaGo by relying solely on self-play reinforcement learning, excluding the need for human knowledge except for the rules of the game. AlphaZero [21] generalised the AlphaGo Zero approach to play other games like chess and shogi. MuZero [22] extended the AlphaZero approach by eliminating the need for knowing the game rules.

The first application of DRL to the mapless navigation task is described in [5]. Since then, numerous studies have used this application as an experimental case. For example, [23] tested the energy efficiency of deep and spiking neural networks with neuromorphic hardware on the mapless navigation task. [24] replaced the original differential wheeled robot with a Hybrid Unmanned Aerial Underwater Vehicle (HUAUV), a type of robot that can operate in both air and water media.

Some articles reveal the current challenges and difficulties of DRL. [25] discussed reproducibility issues and provided guidelines for more robust experimentation in DRL. [26] criticised DRL in many aspects, including the lack of problem-specific knowledge exploitation, difficult reward shaping, local optima, poor generalisation, and hard-to-reproduce results. [27] specifically discussed the difficulties of reproducing the results of DRL papers. [28] surveyed the sim-to-real transfer gap of DRL for robotics.

3 Industrial Relevance, Real-world Applicability, and Scientific Impact

DRL has garnered immense attention from both the academic and industrial communities due to its remarkable performance in complex decision-making tasks. This breakthrough has led to significant advancements in several domains, including robotics, finance, healthcare, and natural language processing. In the field of robotics, DRL has emerged as an approach for autonomously acquiring complex behaviours from sensor data. One such application is mapless navigation, an essential capability for most mobile robots. This section discusses the industrial relevance, real-world applicability, and scientific impact of mapless navigation with DRL.

3.1 Industrial relevance

Traditionally, autonomous navigation relies on the simultaneous localisation and mapping (SLAM) algorithm combined with motion planning algorithms. This method requires a pre-built map of the navigation environment based on dense depth sensors. However, it has three main drawbacks: 1) the need for global knowledge, i.e., the map, to enable robust navigation; 2) the resource-intensive nature of maintaining the map; and 3) the precision of the map highly depends on the quality of sensor data [5]. Rapid generation of navigation behaviours without global knowledge presents a challenge, and building a map based on sparse depth sensor data is nearly impossible [5].

A DRL-based mapless motion planner can overcome these difficulties with the help of lightweight localisation solutions, such as Wi-Fi localisation [5]. Mapless navigation is particularly useful in scenarios where pre-built maps are not available, such as in unexplored environments, or in situations where the environment changes frequently. Unlike traditional methods of building and updating maps, the mapless motion planner requires only limited global knowledge, specifically the real-time target location with respect to the robot coordinate frame. This method greatly reduces hardware requirements but is mainly suitable for indoor environments.

3.2 Real-world applicability

DRL is centred around the idea of trial-and-error learning. However, this trial-and-error training process may lead to unexpected damage to the real robot for certain tasks, such as obstacle avoidance. Moreover, due to the sample inefficiency of current DRL algorithms, training in the real world takes time that exceeds acceptable levels. Training in virtual environments not only prevents damage to real objects but also accelerates the training process through simulation time acceleration and parallel training techniques.

Transferring learned policies from simulation to reality is not straightforward, as it often encounters the so-called sim-to-real gap. Various methodologies have been proposed to address this challenge, including: i) zero-shot learning, which involves building

a realistic simulator or obtaining enough simulated experience for direct real-world application; ii) domain randomisation, which involves randomising the simulation to cover the real distribution of the real-world data rather than carefully modelling real-world parameters; and iii) domain adaptation, which uses data from the source domain to improve a model on a different target domain [28]. In robotic navigation, three promising methods have been proposed: curriculum learning [23], continual learning [28], and policy distillation for multiple tasks [28].

Although it is currently challenging to directly apply DRL-based mapless motion planners to real-world scenarios, it is not impossible.

3.3 Scientific impact

Mapless navigation has been widely used in research as a testbed for evaluating novel ideas. For instance, in [23], deep and spiking neural networks were evaluated for their energy efficiency in mapless navigation tasks using neuromorphic hardware. In another study, [24] utilised a Hybrid Unmanned Aerial Underwater Vehicle (HUAUV), capable of operating in both air and water environments, to replace the original differential wheeled robot.

Of particular interest is the deep reinforcement learning paradigm itself. Although DRL is not necessarily superior to traditional methods in all robotic tasks, many researchers are drawn to this paradigm due to its high generality and ability to solve high-dimensional problems. Research has already applied DRL to two types of high-dimensional robotic tasks, dexterous manipulation and legged robot locomotion, and achieved significant results.

3.4 Conclusion

In conclusion, mapless navigation with DRL has shown potential for overcoming the limitations of traditional SLAM with motion planning algorithms. Its industrial relevance lies in reducing hardware requirements and enabling rapid navigation behaviour generation without a precise map. However, the real-world applicability of DRL-based mapless motion planners is hindered by the sim-to-real gap, necessitating further research to bridge this gap and reduce the level of difficulty. Despite these challenges, DRL has been widely adopted in various research domains, and its generality and ability to solve high-dimensional problems make it an attractive approach for robotic tasks. As DRL continues to mature, it is expected to have a substantial impact on both scientific research and real-world applications, driving progress in autonomous navigation and other complex decision-making tasks.

4 Theory

4.1 Preliminaries

The reinforcement learning problem can be defined as a policy search in a sequential decision-making process. At each time step t, given a state $s_t \in S$, the agent takes an action $a_t \in A$ with respect to its policy $\pi : S \to A$, receives a reward $r(s_t, a_t)$, and transitions to a new state according to $p(s_{t+1} \mid s_t, a_t)$. Formally, this sequential decision-making process is modelled as a Markov decision process (MDP), which is a quadruple $\langle S, A, p, r \rangle$, where:

- S, called the state space, is a set of all valid states.
- A, called the action space, is a set of all valid actions.
- $p : S \times A \to \Delta S$ is a transition probability function that represents the probability of transitioning to the next state if a certain action is taken in the current state.
- $r : S \times A \to \mathbb{R}$ is a reward function that generates a reward signal.

The aim is to find a policy that yields the maximal sum of rewards:

$$\max_{\pi} \sum_{t} \mathbb{E}_{s_t \sim p_\pi, a_t \sim \pi} \left[r(s_t, a_t) \right]$$

A trajectory, also known as an episode or rollout, is a sequence of states and actions:

$$(s_0, a_0, s_1, a_1, \ldots)$$

The first state s_0 is sampled from an initial state distribution, sometimes denoted by ρ_0:

$$s_0 \sim \rho_0(\cdot)$$

For convenience, let ρ_π denote the state-action marginals of the trajectory distribution induced by the policy, and then simplify the subscript of the expectation operator of the aim as:

$$\max_{\pi} \sum_{t} \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} \left[r(s_t, a_t) \right]$$

Most of the time, the expectation operator is taken outside:

$$\max_{\pi} \mathbb{E}_{s_0, a_0, s_1, a_1, \ldots} \left[\sum_{t} r(s_t, a_t) \right],$$

where $s_0 \sim \rho_0(\cdot)$, $a_t \sim \pi(\cdot \mid s_t)$, and $s_t \sim p(\cdot \mid s_{t-1}, a_{t-1})$ as defined before. The sum of rewards $\sum r(s, a)$ is named as return (long-term cumulative reward), which is denoted by R. Thus, the goal can also be rephrased as finding a policy that maximises the expected return $\mathbb{E}[R]$.

4.2 Policy Gradient

The phrase "policy gradient" refers to a class of methods that search for an optimal policy by optimising a parameterised policy with respect to the expected return by gradient descent. Let ϕ denote the parameter of the policy, i.e., π_ϕ. Methods that optimise π_ϕ by:

$$\phi \leftarrow \phi - \lambda \nabla_\phi J(\phi),$$

where λ controls the amount that ϕ is updated, and $J(\phi) = \mathbb{E}\left[R\right]$ is the objective to maximise for the policy, are called policy gradient methods. "Policy gradient" can also refer to $\nabla_\phi J(\phi)$ in the above expression.

Policy optimisation can be done using gradient-based approaches other than policy gradient, e.g., TRPO [14] and PPO [15]. Instead of policy optimisation, policy search can be done using gradient-free approaches such as evolutionary computation.

4.3 Actor-Critic

Actor-Critic refers to an algorithm pattern that estimates a value function in addition to the policy function. The estimated value function is used to assess the policy's action selections. The policy is then optimised in the direction suggested by the value function. Thus, it's natural to think of the value function as a "critic" and the policy as an "actor." The value function could be the action-value (a.k.a. action-quality) $Q(s, a; \theta)$ or state-value $V(s; \theta)$ depending on the algorithm design. Taking action-value actor-critic as an example, the pattern is:

$$\theta \leftarrow \theta - \lambda_Q \nabla_\theta J_Q(\theta)$$
$$\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi),$$

where λ_Q and λ_π respectively control the amount that θ and ϕ are updated, and $J_Q(\theta)$ and $J_\pi(\phi)$ are objectives to maximise for both the critic and the actor. Specifically, the actor objective is to maximise the expected return under the action-value function Q, while the critic objective could be minimising Bellman error with optimal action-value Q^* described by Bellman equation.

4.4 Deep Deterministic Policy Gradient (DDPG)

DDPG can be classified as an action-value actor-critic policy gradient algorithm. It estimates both the action-value and policy and optimises the function approximators by gradient descent. The term "Deep" means that this algorithm uses backward propagation of errors (short for backpropagation), a technique from deep learning, to calculate gradients.

A deterministic policy is a special case of a standard stochastic policy that gives the probability of 1 to one available action and 0 to the remaining actions. DDPG treats the stochasticity, roughly equivalent to the ability to explore, of the policy separately. It

constructs an exploration policy by adding noise sampled from a stochastic process to the deterministic policy:

$$a_t \sim \pi_\phi(\cdot \mid s_t) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma)$$

DDPG adopts the Q-learning [8] strategy to optimise the action-value function approximator $Q_\theta(s, a)$ by minimising the loss:

$$L(\theta) = \mathbb{E}_{(s_t, a_t) \sim \rho_\pi} \left[(Q_\theta(s_t, a_t) - y_t)^2 \right]$$

where

$$y_t = r(s_t, a_t) + \gamma Q_\theta(s_{t+1}, \pi(\cdot \mid s_{t+1}))$$

is called *target*. Since the target also depends on θ, this leads to unstable learning. To solve this problem, [8] introduced the use of a *replay buffer* and a separate *target network* for calculating the target y_t. DDPG employ these, as shown in Algorithm 1.

A replay buffer serves as a memory unit that stores and manages the agent's past experiences. Sequential experiences collected by the agent are often highly correlated, which can lead to biased learning and slow convergence. The replay buffer helps break these correlations by randomly sampling experiences from its memory. In addition, the replay buffer also improves the sample efficiency since it enable the use of the past experiences, though this increase the computational cost due to the need to sample and process experiences from the buffer.

The target network is a delayed version of the regular network, achieved through polyak averaging (exponential moving average), which has been shown to stabilise training [10]. This involves computing a weighted average $\overline{\theta}$ of the regular $Q_\theta(s, a)$ network's weights instead of simply copying them when updating the target network at each iteration.

Algorithm 1 Deep Deterministic Policy Gradient [12]

Initialise a critic networks Q_θ and an actor network π_ϕ with random parameters θ and ϕ
Initialise target network weights $\overline{\theta} \leftarrow \theta$ and $\overline{\phi} \leftarrow \phi$
Initialise an empty replay buffer $\mathcal{B}$
for each iteration **do**
    for each environment step t **do**
        Sample action with exploration noise $a_t \sim \pi_\phi(\cdot \mid s_t) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$
        $s_{t+1} \sim p(\cdot \mid s_t, a_t)$
        $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
    for each gradient step **do**
        $\theta \leftarrow \theta - \lambda_Q \nabla_\theta J_Q(\theta_i)$
        $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
        # Update target networks' weights:
        $\overline{\theta} \leftarrow \tau\theta + (1-\tau)\overline{\theta}$
        $\overline{\phi} \leftarrow \tau\phi + (1-\tau)\overline{\phi}$

4.5 Twin-delayed Deep Deterministic Policy Gradient (TD3)

TD3 is a descendant of DDPG. It introduces twin critic technique and delayed policy and target networks update to address the action-value function approximation error which causes unstable performance DDPG suffers from [13].

Algorithm 2 Twin-delayed Deep Deterministic Policy Gradient [13]

Initialise two critic networks Q_{θ_1}, Q_{θ_2} and an actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialise target network weights $\overline{\theta}_i \leftarrow \theta_i$ for $i \in \{1,2\}$ and $\overline{\phi} \leftarrow \phi$
Initialise an empty replay buffer $\mathcal{B}$
for each iteration **do**
  for each environment step t **do**
    Sample action with exploration noise $a_t \sim \pi_\phi(\cdot \mid s_t) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma)$
    $s_{t+1} \sim p(\cdot \mid s_t, a_t)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  for each gradient step **do**
    $\theta_i \leftarrow \theta_i - \lambda_Q \nabla_{\theta_i} J_Q(\theta_i)$ for $i \in \{1,2\}$
    if t mod $d = 0$ **then**
      $\phi \leftarrow \phi - \lambda_\pi \nabla_\phi J_\pi(\phi)$
      # Update target networks' weights:
      $\overline{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\overline{\theta}_i$ for $i \in \{1,2\}$
      $\overline{\phi} \leftarrow \tau\phi + (1-\tau)\overline{\phi}$

4.6 Soft Actor-Critic (SAC)

To better treat the exploration and exploitation trade-off in mathematics, SAC add a policy entropy term into the classic goal:

$$\pi^* = \arg\max_\pi \sum_t \mathop{\mathbb{E}}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot \mid s_t)) \right],$$

where α is the temperature parameter that determines the relative importance of the entropy term versus the reward. Formally, the objective is to solve the constrained optimisation problem:

$$\max_\pi \sum_t \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[r(s_t, a_t) \right]$$

subject to

$$\mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[-log(\pi_t(a_t \mid s_t)) \right] \geq \mathcal{H} \ \forall t$$

which means maximise the expected return with the constraint that the policy entropy is greater or equal to the desired minimum expected entropy $\mathcal{H}$ for all t.

To reduce the difficulty of hyperparameter search, α is dynamically adjusted during training. The gradients for α are computed with the following objective:

$$J(\alpha) = \mathbb{E}_{a_t\sim\pi_t} \left[-\alpha log(\pi_t(a_t \mid s_t)) - \alpha\mathcal{H} \right].$$

SAC adopts the twin critic technique from TD3 to reduce the action-value function approximation error.

Unlike TD3, SAC doesn't have a target policy for stabilising the learning since it learns a stochastic policy which doesn't have the unstable learning issue.

Algorithm 3 Soft Actor-Critic with twin critics [13] and auto-tuned temperature [18]

Initialise two critic networks Q_{θ_1}, Q_{θ_2} and an actor network π_ϕ with random parameters θ_1, θ_2, ϕ
Initialise target network weights $\overline{\theta}_i \leftarrow \theta_i$ for $i \in \{1,2\}$
Initialise an empty replay buffer $\mathcal{B}$
for each iteration **do**
  for each environment step t **do**
    $a_t \sim \pi_\phi(\cdot|s_t)$
    $s_{t+1} \sim p(\cdot|s_t, a_t)$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(s_t, a_t, r(s_t, a_t), s_{t+1})\}$
  for each gradient step **do**
    # $\hat{\nabla} J$ denotes the estimated gradient of an objective function
    $\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i)$ for $i \in \{1,2\}$ ▷ *Backpropagation*
    $\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi)$ ▷ *Backpropagation*
    $\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha)$ ▷ *Auto-tune temperature by backpropagation*
    # Update target networks' weights:
    $\overline{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\overline{\theta}_i$ for $i \in \{1,2\}$ ▷ *Exponential moving average*

5 Design

5.1 Algorithm library

The library is designed to be modular. The code is aimed at supporting research in DRL. Most of it is written in python in a highly modular way, such that researchers can easily swap components, transform them or write new ones with little effort. Figure 4 is a UML class diagram for SAC. The SAC class contains the main logic of the algorithm. The actor network, critic network, and the replay buffer are spun off from the main logic to form separate classes. These three classes are then passed as arguments into the SAC class. In this way, SAC can use neural networks and replay buffers with different mechanisms, and can also share certain components with other algorithm classes.

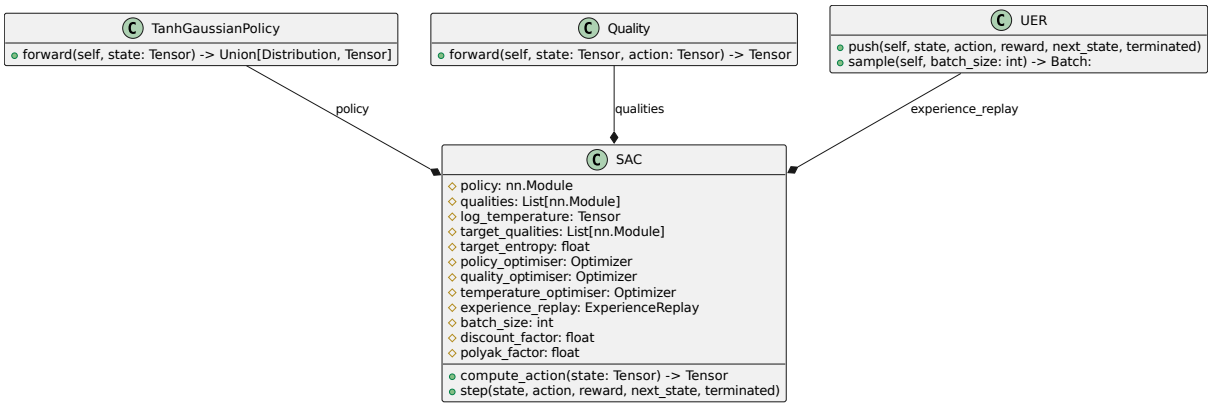


Figure 4: UML class diagram of the Soft Actor-Critic algorithm

To prevent algorithm classes from working with the wrong components, this library uses Python's duck typing feature for static type checking when passing arguments. This means that when the wrong component is passed in, static analysis tools like *mypy* will report an error to alert the user.

As shown in Figure 4, the API of the algorithm class is designed such that the argument list only exposes hyperparameters and necessary components, while member methods only expose gradient stepping and action computation. By designing the API in this way, it forces the API to be highly consistent with the algorithm in the paper in terms of cognitive load, thereby avoiding potential usage barriers caused by the API containing unfamiliar information to the user [29].

In software design, unit testing is commonly used to maintain the stability of the API and critical logic. This library also utilises unit testing, but not for maintaining API stability. This is because DRL is inherently an unstable field, with algorithms evolving rapidly. Therefore, maintaining API stability may become a hindrance in the future for continuously updating this library.

As this is an algorithm library, a relatively lower-level components in an application software, its compatibility is closely related to its usability. To achieve high compatibility, this library sacrifices the opportunity to use advanced Python features to simplify the code, keeping the minimum Python version at 3.8 to ensure compatibility with ROS

Noetic. On the other hand, the library tries to minimise dependencies on other software packages to avoid conflicting with dependencies of other packages. It only use *Pytorch* for tensor computation, *attr* for enhancing object-oriented paradigm, and *toolz* for enhancing functional programming paradigm. Other development-specific dependencies, such as debugger, profiler, and linter, are put in to a separated dependency group which only to be installed by developer manually.

The library is packaged with the latest Python packing standard, PEP 518 [30]. This ensures that the library can be easily distributed on the web and easily installed on any computer with a Python interpreter.

The development of this library is version-controlled using Git in conjunction with GitHub. The source code, development logs, and statistics can be found here: `https://github.com/MinghonZi/deeprl`.

5.2 Training environment

Since the purpose is to study DRL using mapless navigation as an example, the training environment is kept as simple as possible. The training environment is modelled in the Gazebo Classic simulator, with a square enclosure measuring $3m \times 3m$ and a flat ground, as shown in Figure 5. TurtleBot3, a simplest differential drive robot with a 2d LiDAR, is chosen as the agent. ROS Noetic is used for communication between the training program and the simulator as a message queue.

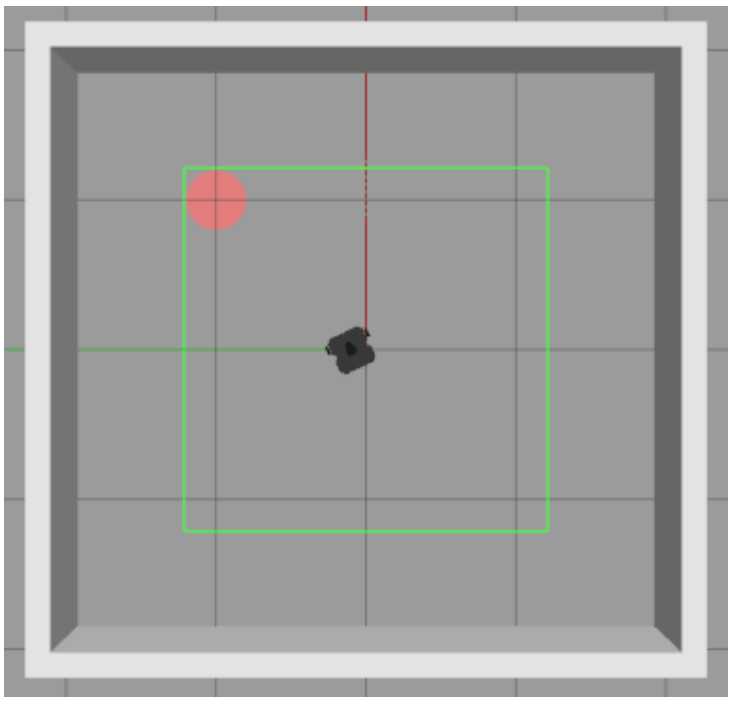


Figure 5: Aerial view of the training environment in Gazebo Classic.

The environment spaces and reward signal are defined below.

Observed state space:

$$S = (\text{lidar}, \text{imu}, \text{distance}),$$

where *lidar* is the data from LiDAR, *imu* is the data from the inertial measurement unit (IMU), and *distance* is the relative distance of the navigation goal w.r.t the robot.

Action space:

$$A = (\text{linear}, \text{angular}),$$

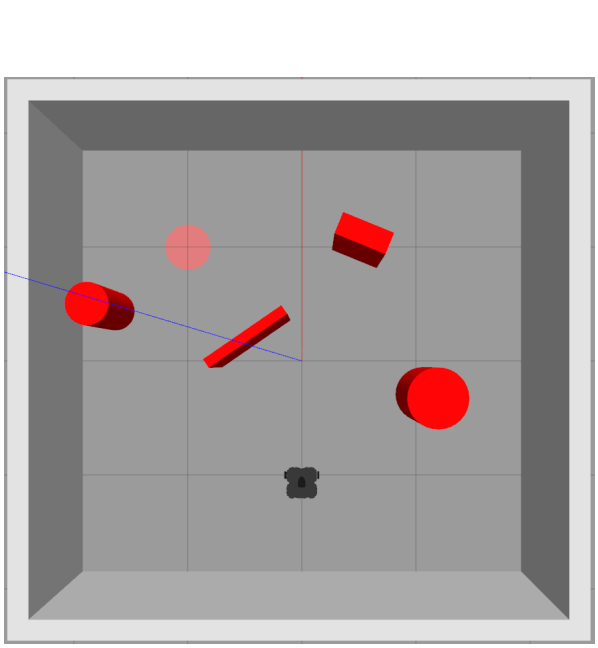
where *linear* and *angular* are respectively the linear and angular velocity of the robot.

Reward shaping

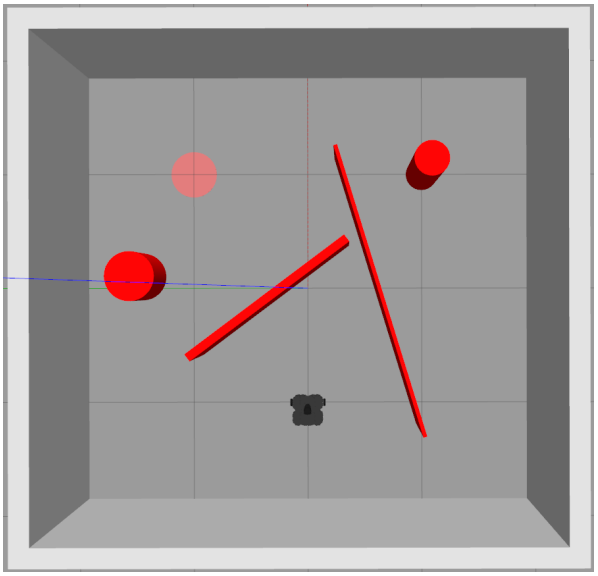
$$r(s_t, a_t) = \begin{cases} r_{\text{goal}}, & \text{if } d_t < d_{\text{th}} \\ r_{\text{obstacle}}, & \text{if } x_t < x_{\text{th}} \\ c(d_t - d_{t-1}), & \text{otherwise.} \end{cases}$$

Notice that c is an amplification factor and a parameter of the environment, $s_t \in S$, and $a_t \in A$.

To train the obstacle avoidance ability, two training environments with obstacles were designed. In the first environment, as shown in Figure 6a, the obstacles are relatively small, leaving a lot of space for the robot to move. In the other environment, as shown in Figure 6b, the obstacles are larger, and the paths available for the robot to move are relatively more limited.



(a) An obstacle slightly blocks the path of moving to the goal.



(b) Obstacles almost completely block the path of moving to the goal. The robot needs to first escape to one end of the obstacle before it can approach the goal.

Figure 6: Environments with obstacles

The code for the training environments is available on `https://github.com/TACPSLab/SNN4RoboticControl`.

6 Experimental Method

6.1 Integration test of the library

Before deploying the library to the mapless navigation training environment, integration testing was conducted on all algorithms in the library to ensure the implementation was valid. As mapless navigation is a physical environment, it was best to select physically simulated problems for the test cases as well. To quickly test the validness after each modification on the implementation, the test cases were expected to be as simple as possible for fast converging. Based on the above requirements, six locomotion control training environments from *MuJoCo* were selected as the testing bed. In addition, the hyperparameters selected were the same as those used in the original papers of each algorithm. This facilitated comparison between the testing results and the experimental results in the original papers, which was helpful for diagnosing the implementation.

6.2 Training policies

As DRL is not yet a mature field, there is no strict training process and policy evaluation metrics in place [25]. Tuning parameters and evaluating policies obtained from training were mostly based on past experience. Although hyperparameter search frameworks such as *Optuna* can be used to search for optimal hyperparameters, in practice, the parameter combinations proposed in the original papers were often sufficient. Therefore, the experiments in this project used the parameters from the original papers. As there was no widely recognised evaluation metric, the only criterion for determining whether a policy had completed training was to observe whether the episodic return curve converged. Once the curve remained stable for a period of time, it was considered to have reached the end of training. In addition, the lack of a standardised evaluation metric meant that comparing which of two policies was better could only be done by repeatedly running them in the environment and observing which policy's behaviour was closer to the expectation.

Although the training process for DRL was not systematic, there was still a fixed pattern that was followed. Initially, the agent started with a random or partially pre-trained policy. The agent then interacted with the environment by taking actions and observing the consequences. These interactions generated a sequence of states, actions, rewards, and new states, also known as a trajectory. As the agent interacted with the environment, it collected data on the trajectories, which were used to estimate the value function. The value function indicated the expected return that the agent could obtain from a given state while following a particular policy. With the value function estimated, the agent then optimised its policy. This step aimed to improve the agent's decision-making by adjusting the policy parameters to better align with the estimated value function. The updated policy should ideally result in the agent choosing actions that lead to higher return. The agent continued to interact with the environment using the updated policy, generating new trajectories and further refining the value function and policy. This iterative process continued until a specified stopping criterion was met, which could be a fixed number of iterations or convergence of the policy. This pattern

was also reflected in the algorithms listed in Section 4. In code, it was represented as follows:

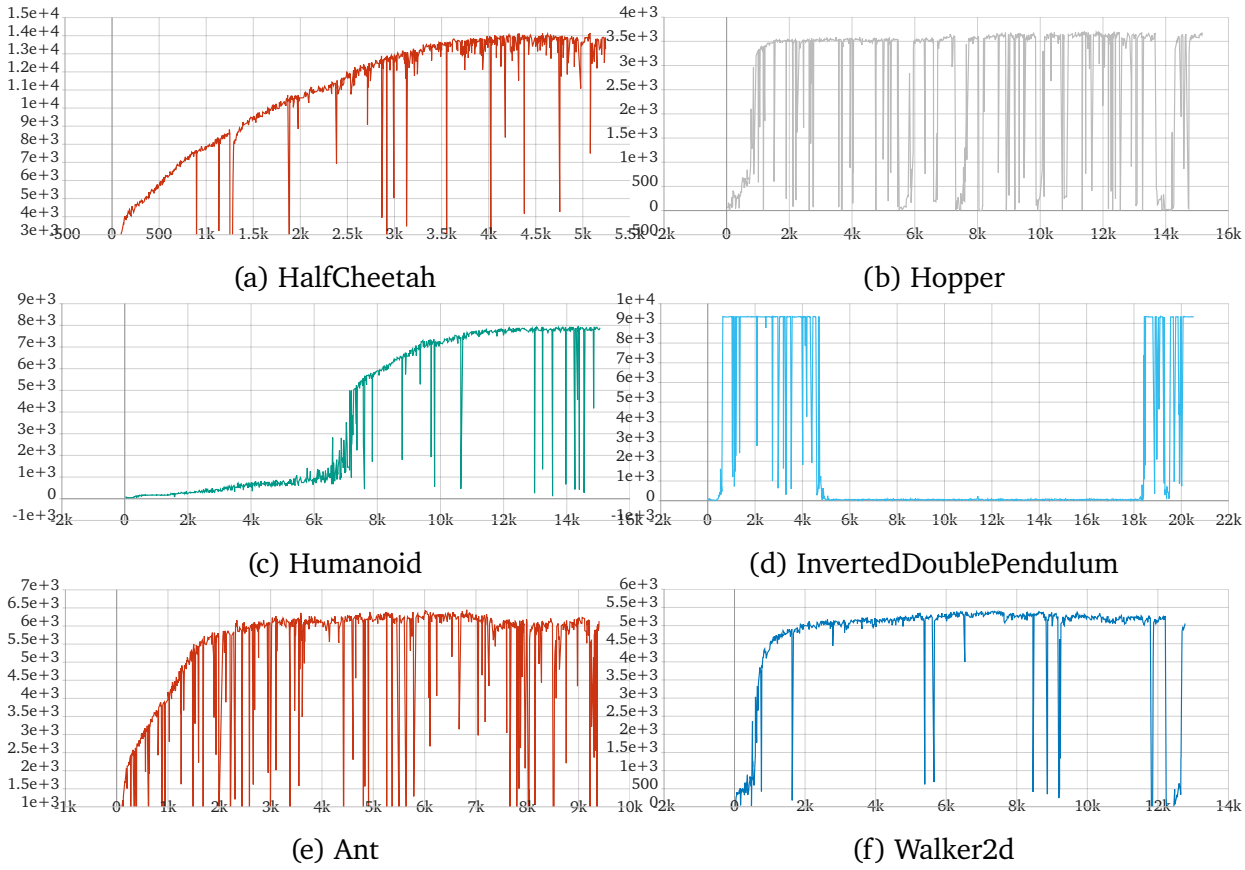
```
env = Env(arg1, arg2, ...)
observation = env.reset(seed)
for step in range(num_steps):
    action = agent.compute_action(observation)
    next_observation, reward, terminated, truncated = env.step(action)
    agent.step(observation, action, reward, next_observation, terminated)

    if not terminated or truncated:
        observation = next_observation
    else:
        observation = env.reset()
env.close()
```

The policy during the training process was checkpointed at a certain frequency and persisted to the same directory as the training program. The best policy candidates were then selected with respect to the episodic return curve. Finally, the best policy was determined by running each candidates repeatedly and observe the success rate. The policy had the highest success rate was considered to be the best policy. Note that the best policy is not necessarily the optimal policy, since there is no theory guarantee of the policy optimality.

7 Results

7.1 Integration test



(a) HalfCheetah

(b) Hopper

(c) Humanoid

(d) InvertedDoublePendulum

(e) Ant

(f) Walker2d

Figure 7: Episodic return of TD3 in MuJoCo training environments. The abscissa is the episode count, and the ordinate is the return. The algorithm converges in all six environments. This indicates that the implemented algorithm is effective.

Figure 7 is the integration test result of TD3. The algorithm successfully converges in all six MuJoCo training environments. This indicates that the algorithm's implementation is valid. Similar testing has been carried out on other algorithms in the library to validate the effectiveness of the implementation. To keep concise, the report ignores reporting the testing results of other algorithms.

7.2 Trained policies

The performance of trained policies were recorded as videos. Please refer to `https://github.com/MinghonZi/final-year-proj`.

8 Discussion

8.1 Escaping and sparse reward

Recall that the reward function are defined as:

$$r(s_t, a_t) = \begin{cases} r_{\text{goal}}, & \text{if } d_t < d_{\text{th}} \\ r_{\text{obstacle}}, & \text{if } x_t < x_{\text{th}} \\ c(d_t - d_{t-1}), & \text{otherwise.} \end{cases}$$

The last case encourages the agent to approach the goal. However, it assumes that there is no long barrier between the agent and the goal. A long obstacle would mean that the agent has to move a distance away from the goal in order to escape it before having a chance to approach the goal. Using this reward function does not give the policy the ability to escape, but it is very difficult to define a reward that can encourage escaping.

On the other hand, the last case can be interpreted as a vague expectation of the terrain on which the agent will be trained, which is a prior knowledge. Essentially, this is no different from the map-based method of building a map before navigation.

Removing the last case results in a reward function that can generate a more ideal reward signal. However, the reward generated by this type of reward function belongs to sparse reward.

$$r(s_t, a_t) = \begin{cases} r_{\text{goal}}, & \text{if } d_t < d_{\text{th}} \\ r_{\text{obstacle}}, & \text{if } x_t < x_{\text{th}} \end{cases}$$

Sparse reward means the agent may only receive positive feedback after completing a long sequence of actions. This can slow down the learning process, as the agent has to explore a large space of possible actions before finding a rewarding state. When using algorithms with poor exploration capability, such as TD3, the agent often learns to stay still after training for a period of time. This is because in the initial stage, it did not explore that reaching the goal would receive a big positive feedback, while moving around aimlessly would receive small negative feedback. Therefore, for the agent, staying still is the way to obtain the highest reward.

Even if the last case is not removed, the reward curve is still difficult to converge and stabilise at the highest point, as shown in Figure 8. Although the frequency of reaching the high point increases with the number of training iterations, the episodic return curve brought by a well-defined reward function should typically slowly converge to the highest point and hold steady, rather than fluctuating frequently.

The ideal DRL algorithm should not heavily rely on a well-defined reward function to learn expected behaviours. We have the above discussion is because the current DRL algorithms are far from idealism and are often depend heavily on reward shaping.

The ideal DRL algorithm should be able to learn expected behaviours effectively without an over-reliance on a well-defined reward function [25]. This is a crucial aspect to consider, as it allows the algorithm to be more adaptable and robust in real-world situations where the environment and desired outcomes are not always clearly defined.

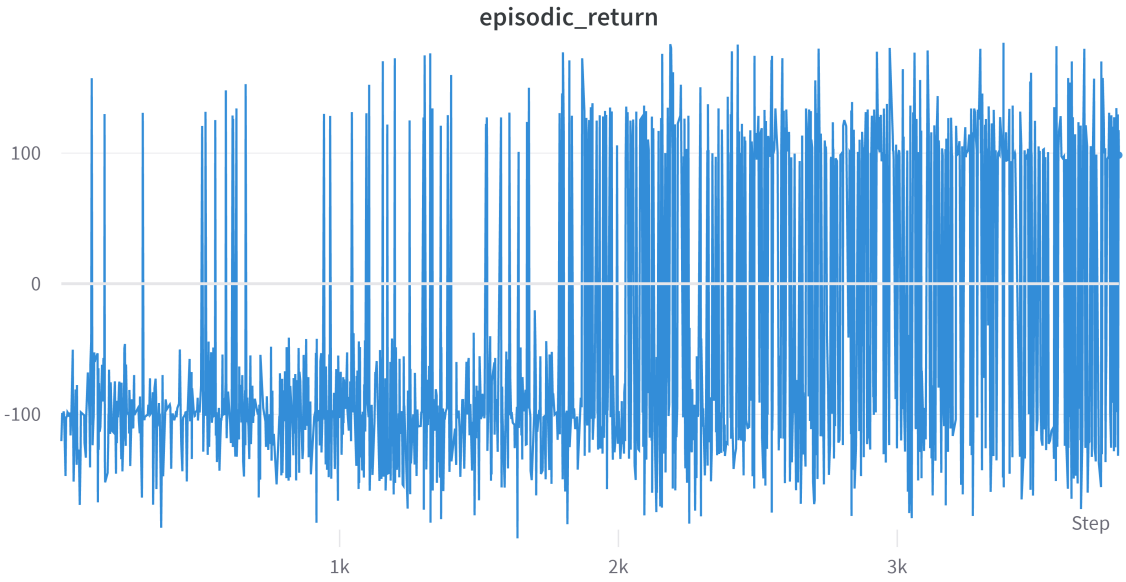


Figure 8: Episodic return curve is fluctuated between the highest and lowest return.

The reason of emphasising the importance of moving towards such an ideal algorithm is that, as it was discussed above, current DRL algorithms often relying heavily on reward shaping to guide their learning process.

8.2 Partial observability

Normally, in robotic tasks, the agent does not have complete access to the underlying state of the environment. In the case of this project, the agent was only aware of the relative position of the goal, the depth data of it surrounding from LiDAR, and the IMU data. This partial observability can make it difficult for DRL algorithms to learn optimal policies since the agent is not fully aware of its surroundings. From the experimental results, it can be observed that the agent often moves forward and backward multiple times without generate any meaningful movement that helps to achieve the goal. One possible explanation could be that the agent is unaware of its current motion state because the information about the velocity command executed in the previous state is not included in its state space. As the agent does not know whether it was moving forward or backward or turning in the previous state, it can only assess the current situation and generate a new velocity command that is independent of the previous motion state.

8.3 Transfer learning

Transfer learning is the ability to leverage knowledge acquired in one task or domain to improve performance in another related task or domain. Currently, DRL algorithms struggle to transfer knowledge across tasks or even across different instances of the same task, requiring extensive retraining. Due to this limitation, in this project, the policy is trained from scratch for every experiment, which wastes a great amount of time.

8.4 Catastrophic forgetting

Catastrophic forgetting describes the phenomenon where an artificial neural network optimised by backpropagation has a tendency to suddenly and severely lose memory of previously acquired knowledge when it learns new information. Figure 9 shows a typical example. At the beginning of the training, the TD3 agent mastered the control of the double inverted pendulum within 1k episodes. However, after continued training, at around 5k episodes, the episodic return curve suddenly dropped to nearly zero, indicating that the previously learned knowledge was completely forgotten. After this phenomenon occurred, it took the agent nearly 10k episodes to re-learn how to control the pendulum, and the control policy re-learned was worse than the one learned the first time, as can be seen from the frequency of the curve fluctuations.

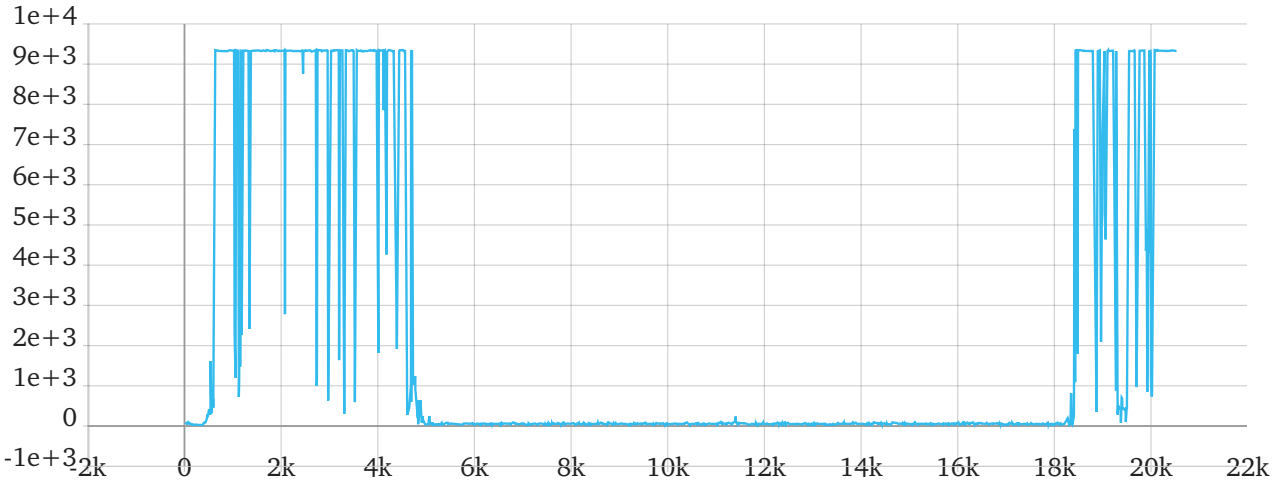


Figure 9: Catastrophic forgetting of a TD3 agent trained on the MuJoCo InvertedDoublePendulum environment.

9 Reflection on Learning

Throughout the course of the final year project, valuable experience and insight were gained into the challenges of research and development. The process provided opportunities to refine technical skills and enhance understanding of DRL. However, there were also areas where improvements could have been made, which will be addressed in this reflection on learning.

9.1 Improvement in logging practices

One challenge encountered during the project was reproducing issues that had previously occurred. This was mainly due to insufficient logging and documentation of work as it progressed. Although a version control system was used to maintain the development of the software, the importance of maintaining a detailed record of the steps taken, the hyperparameters used, and the outcomes observed became apparent. Such documentation would have allowed for efficient identification and addressing of problems, especially when repeating experiments.

In future projects, logging progress on a daily basis will be implemented to ensure that any issues encountered can be easily reproduced and resolved. Figure 10 provides a typical example.

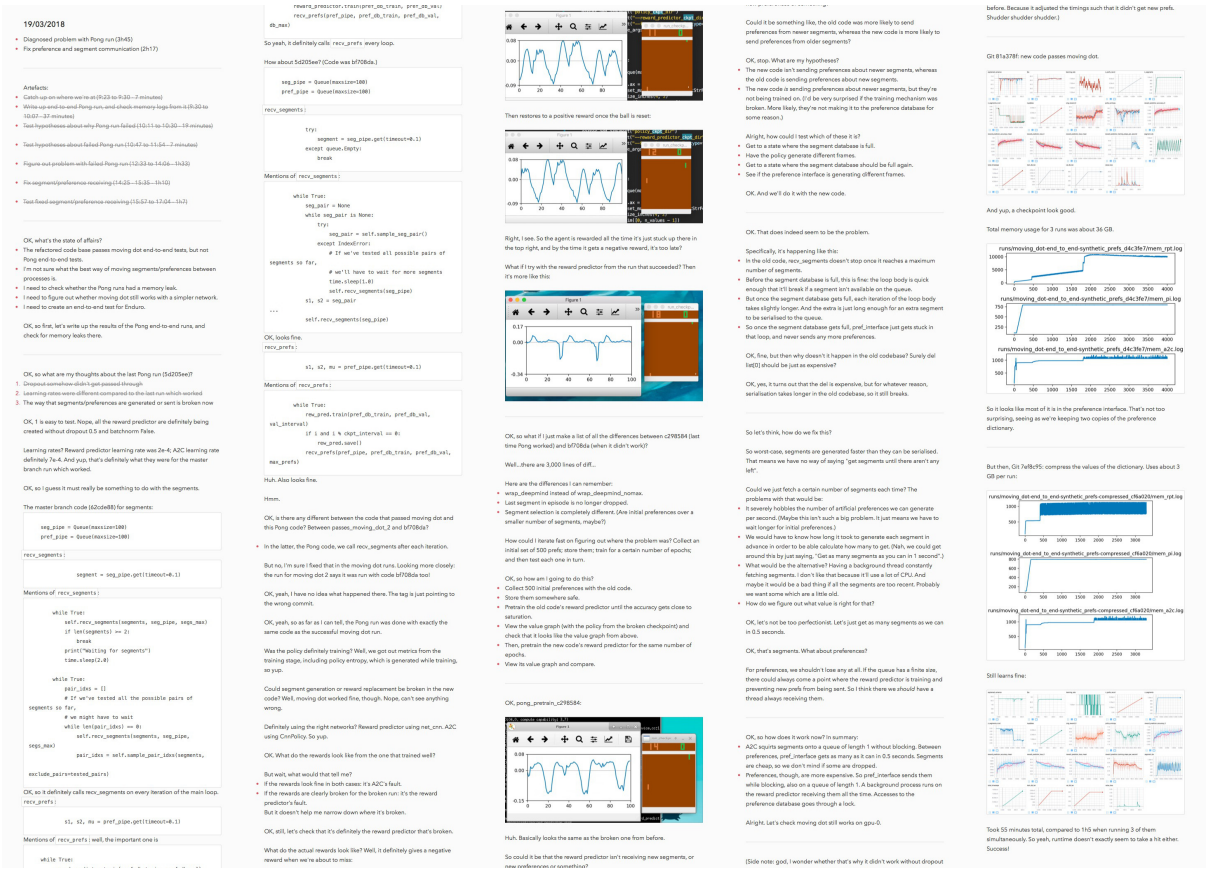


Figure 10: A typical daily log [27].

9.2 Comprehensive specification report

In hindsight, more time and effort should have been invested into producing a thorough and detailed specification report. This would have not only provided a more solid foundation for the project but also streamlined the process of preparing the thesis. Much of the information and content from the specification report could have been directly incorporated into the thesis, saving time and effort in the long run.

For future endeavours, the focus at the early stage of the project will be on creating a comprehensive specification report, as it will contribute significantly to the final project report.

9.3 Advance report drafting

The importance of writing the report earlier was learned. Due to time constraints and other obligations, there was a rush to complete the thesis towards the end of the project. This made the process stressful and challenging and resulted in a greater likelihood of errors or omissions in the thesis. Starting early and allocating more time to report writing could have prevented this situation.

Therefore, in future projects, more time will be set aside to work on reports to ensure that they are not rushed towards the end.

9.4 Foundational knowledge acquisition

During the course of the project, significant time was devoted to reading research papers in order to gain a deeper understanding of the subject matter. However, this approach did not provide a solid foundation in the fundamental concepts and principles of the field. In retrospect, more time should have been allocated to studying textbooks and other resources that offered a comprehensive overview of the basics before diving into the specialised research papers.

For future projects, a more balanced approach will be adopted by first focusing on mastering the foundational knowledge through textbooks and online lecture series. This will ensure a strong grasp of the essential concepts and better contextualise the findings and insights presented in research papers.

10 Conclusions

This project aimed to develop a DRL-based motion planner for a wheeled robot to navigate indoor without the use of a map. A DRL library was implemented, consisting of three SOTA algorithms: DDPG, TD3, and SAC. Integration tests for all algorithms were conducted on the MuJoCo locomotion control training environments to ensure their valid implementation. The algorithms were then applied to train policies for mapless navigation in simulated environments. The performance of trained policies was recorded as video and can be accessed through the project's GitHub repository.

Throughout the course of the project, several challenges were identified, including sparse reward, partial observability, transfer learning, and catastrophic forgetting. These challenges highlight the current limitations of DRL in robotic applications and provide opportunities for further research and improvement.

The project also provided valuable learning experiences. The reflection offers insights into areas for improvement in future research projects.

In conclusion, this project has successfully demonstrated the application of DRL for mapless navigation in a wheeled robot. The implementation of the DRL library and the training environment, coupled with the insights gained from experiments, establish a solid foundation for further research and exploration in the field of robotics and DRL.

References

[1] M. Xu, 'Final year project specification report,' University of Liverpool, Oct. 2022.

[2] J. Achiam, 'Spinning up in deep reinforcement learning,' 2018.

[3] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, 'Playing atari with deep reinforcement learning,' 2013, NIPS Deep Learning Workshop 2013. DOI: 10.48550/arXiv.1312.5602.

[4] T. M. Moerland, J. Broekens, A. Plaat and C. M. Jonker, 'Model-based reinforcement learning: A survey,' *Foundations and Trends in Machine Learning*, vol. 16, no. 1, pp. 1–118, 2023, ISSN: 1935-8237. DOI: 10.1561/2200000086.

[5] L. Tai, G. Paolo and M. Liu, 'Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,' in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, 2017, pp. 31–36. DOI: 10.1109/IROS.2017.8202134.

[6] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, Second. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html.

[7] D. P. Bertsekas and J. N. Tsitsiklis, *Neuro-Dynamic Programming*, 1st. Athena Scientific, 1996, ISBN: 1886529108.

[8] C. J. C. H. Watkins and P. Dayan, 'Q-learning,' *Machine Learning*, vol. 8, no. 3, pp. 279–292, May 1992, ISSN: 1573-0565. DOI: 10.1007/BF00992698.

[9] R. S. Sutton, D. Mcallester, S. Singh and Y. Mansour, 'Policy gradient methods for reinforcement learning with function approximation,' in *Advances in Neural Information Processing Systems (NIPS)*, vol. 12, MIT Press, 2000, pp. 1057–1063.

[10] V. Mnih, K. Kavukcuoglu, D. Silver *et al.*, 'Human-level control through deep reinforcement learning,' *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, ISSN: 00280836. DOI: 10.1038/nature14236.

[11] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller, 'Deterministic policy gradient algorithms,' in *Proceedings of the 31st International Conference on Machine Learning (ICML)*, ser. ICML'14, vol. 32, Beijing, China: JMLR.org, 2014, pp. 387–395.

[12] T. P. Lillicrap, J. J. Hunt, A. Pritzel *et al.*, 'Continuous control with deep reinforcement learning,' in *International Conference on Learning Representations (ICLR)*, Y. Bengio and Y. LeCun, Eds., 2016. DOI: 10.48550/arXiv.1509.02971.

[13] S. Fujimoto, H. van Hoof and D. Meger, 'Addressing function approximation error in actor-critic methods,' in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, Jul. 2018, pp. 1587–1596. DOI: 10.48550/arXiv.1802.09477.

[14] J. Schulman, S. Levine, P. Abbeel, M. Jordan and P. Moritz, 'Trust region policy optimization,' in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, F. Bach and D. Blei, Eds., ser. Proceedings of Machine Learning Research, vol. 37, Lille, France: PMLR, Jul. 2015, pp. 1889–1897. DOI: 10.48550/arXiv.1502.05477.

[15] J. Schulman, F. Wolski, P. Dhariwal, A. Radford and O. Klimov, 'Proximal policy optimization algorithms.,' *Computing Research Repository (CoRR)*, 2017. DOI: 10.48550/arXiv.1707.06347.

[16] N. Heess, G. Wayne, D. Silver, T. Lillicrap, T. Erez and Y. Tassa, 'Learning continuous control policies by stochastic value gradients,' in *Advances in Neural Information Processing Systems (NIPS)*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. DOI: 10.48550/arXiv.1510.09142.

[17] T. Haarnoja, A. Zhou, P. Abbeel and S. Levine, 'Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,' in *Proceedings of the 35th International Conference on Machine Learning (ICML)*, J. Dy and A. Krause, Eds., vol. 80, Jul. 2018, pp. 1861–1870. DOI: 10.48550/arXiv.1801.01290.

[18] T. Haarnoja, A. Zhou, K. Hartikainen *et al.*, 'Soft actor-critic algorithms and applications,' *Computing Research Repository (CoRR)*, 2018. DOI: 10.48550/arXiv.1812.05905.

[19] D. Silver, A. Huang, C. J. Maddison *et al.*, 'Mastering the game of go with deep neural networks and tree search,' *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016, ISSN: 1476-4687. DOI: 10.1038/nature16961.

[20] D. Silver, J. Schrittwieser, K. Simonyan *et al.*, 'Mastering the game of go without human knowledge,' *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017, ISSN: 1476-4687. DOI: 10.1038/nature24270.

[21] D. Silver, T. Hubert, J. Schrittwieser *et al.*, 'A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,' *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. DOI: 10.1126/science.aar6404.

[22] J. Schrittwieser, I. Antonoglou, T. Hubert *et al.*, 'Mastering atari, go, chess and shogi by planning with a learned model,' *Nature*, vol. 588, no. 7839, pp. 604–609, Dec. 2020, ISSN: 1476-4687. DOI: 10.1038/s41586-020-03051-4.

[23] G. Tang, N. Kumar and K. P. Michmizos, 'Reinforcement co-learning of deep and spiking neural networks for energy-efficient mapless navigation with neuromorphic hardware,' in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 6090–6097. DOI: 10.1109/IROS45743.2020.9340948.

[24] R. B. Grando, J. C. de Jesus, V. A. Kich *et al.*, 'Deep reinforcement learning for mapless navigation of a hybrid aerial underwater vehicle with medium transition,' in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 2021, pp. 1088–1094. DOI: 10.1109/ICRA48506.2021.9561188.

[25] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup and D. Meger, 'Deep reinforcement learning that matters,' *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, Apr. 2018. DOI: 10.1609/aaai.v32i1.11694.

[26] A. Irpan, *Deep reinforcement learning doesn't work yet*, Feb. 2018. [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html (visited on 04/04/2023).

[27] A. Fish, *Lessons learned reproducing a deep reinforcement learning paper*, Apr. 2018. [Online]. Available: http://amid.fish/reproducing-deep-rl (visited on 04/04/2023).

[28] W. Zhao, J. P. Queralta and T. Westerlund, 'Sim-to-real transfer in deep reinforcement learning for robotics: A survey,' in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 2020, pp. 737–744. DOI: 10.1109/SSCI47803.2020.9308468.

[29] Y. Wu, *How to maintain clean core apis for research*, Sep. 2022. [Online]. Available: https://ppwwyyxx.com/blog/2022/Maintain-Clean-Core-APIs-for-Research/ (visited on 17/04/2023).

[30] B. Cannon, N. Smith and D. Stufft, 'Specifying minimum build system requirements for python projects,' PEP 518, 2016. [Online]. Available: https://peps.python.org/pep-0518/ (visited on 10/11/2022).

Appendices

A Original Project Plan